DirectCore Advanced Microcontroller Bus Architecture - Bus Functional Model

User's Guide

Microsemi

Table of Contents

1 – Instantiating and Using the BFMs

This document describes how to use the AMBA® BFM Models that may be included with Microsemi® DirectCores as part of the verification environment.

The AMBA BFMs support both master and slave bus functional models.

The following section outlines how the BFM models described in this document can be used for verification. There are three master BFM models and four slave BFM models as listed in Table 1-1 and Table 1-2.

***Table 1-1* • Master BFM Models**

Master BFM's	Buses	Purpose
BFM_AHBL	AHB-Lite	Testing AMBA High-Performance Bus (AHB)-Lite slaves.
BFM_APB	APB	Testing Advanced Peripheral Bus (APB) slaves. Contains the main AHB BFM with an AHB to APB bridge to expose an APB interface.
BFM_AHBLAPB	AHB-Lite APB	Testing systems requiring both AHB and APB buses (for example Ethernet). Contains the main AHB-Lite BFM with an AHB to APB bridge to expose an APB interface.

***Table 1-2* • Slave BFM Models**

Slave BFM's	Buses	Purpose
BFM_AHBSLAVE	AHB-Lite	AHB Slave model provides a simple read write memory. (Instantiates BFM_AHBSLAVEEXT).
BFM_APBSLAVE	APB	APB Slave model provides a simple read write memory. (Instantiates BFM_APBSLAVEEXT).
BFM_AHBSLAVEEXT	AHB-Lite	AHB Slave model provides a simple read write memory. Also has external memory interface.
BFM_APBSLAVEEXT	APB	APB Slave model providess a simple read write memory Also has external memory interface.

APB Master BFM

In this case the UUT is relatively simple APB based block such as the GPIO function. Figure 1-1 shows how the testbench would be created.

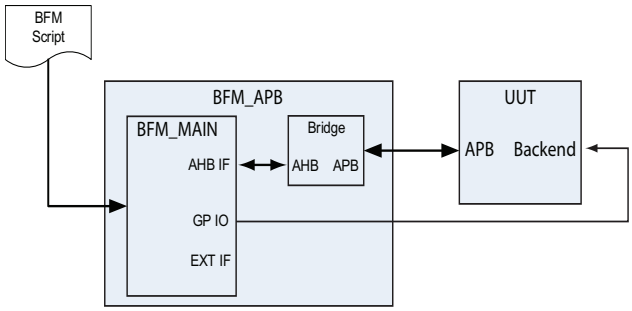

***Figure 1-1* • Testing an APB-based Block**

In Figure 1-1 we can see that the UUT is connected to the BFM_APB BFM. The BFM drives the APB input of the UUT and also has the ability to set and monitor signals on the back end of the UUT through the general purpose I/O (GPIO) interface on the BFM.

This setup allows the BFM to write to the APB register set and to verify that the backed behaves as expected, or vice versa to set a backed input and verify that the core responds correctly in it APB register set.

AHB-Lite Master BFM

In this case the UUT is AHB slave such as the memory function. Figure 1-2 shows how the testbench is created.

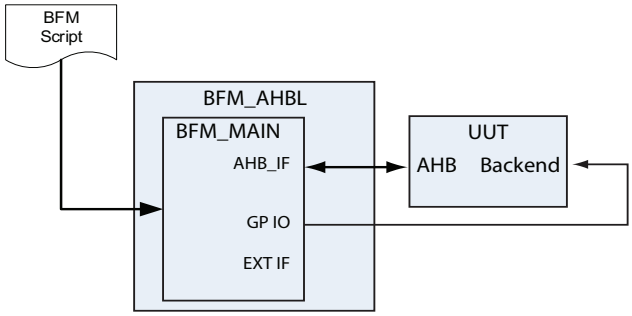

***Figure 1-2* • Testing an AHB-based Block**

In the Figure 1-2 we can see that the UUT is connected to the BFM_AHBL BFM. The BFM drives the AHB input of the UUT and also has the ability to set and monitor signals on the backend of the UUT through the GP I/O interface on the BFM.

The operation is identical to the previous APB example.

APB Slave BFM

In this case the UUT is a core with an APB master interface, this could be the AHB to APB bridge core. Figure 1-3 shows how the testbench would be created.

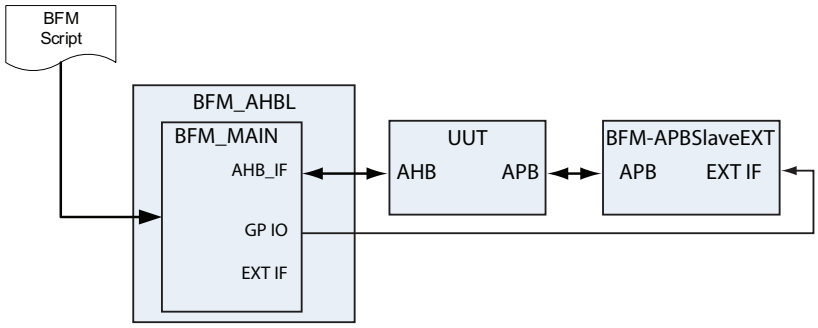

***Figure 1-3* • Testing an APB Master Block**

In Figure 1-3 we can see that the UUT has an AHB slave and an APB master interface. The AHB master interface is driven as shown previously by the BFM_AHBL BFM. The APB master interface of the UUT is connected to an APB slave BFM.

This setup allows the BFM-AHB to perform read/writes through the UUT to the APB slave BFM. The APB slave BFM looks like a memory but has advanced features that allow it to vary its response rates, etc.

In this case the BFM_APBSlaveEXT model is used allowing the AHB master BFM to verify or modify the contents of the slave memory array

AHB Slave BFM

In this case the UUT is a core with an AHB master interface; this could be the Ethernet function with a DMA feature. Figure 1-4 shows how the testbench would be created, note that in this case the Ethernet core also has a APB slave interface.

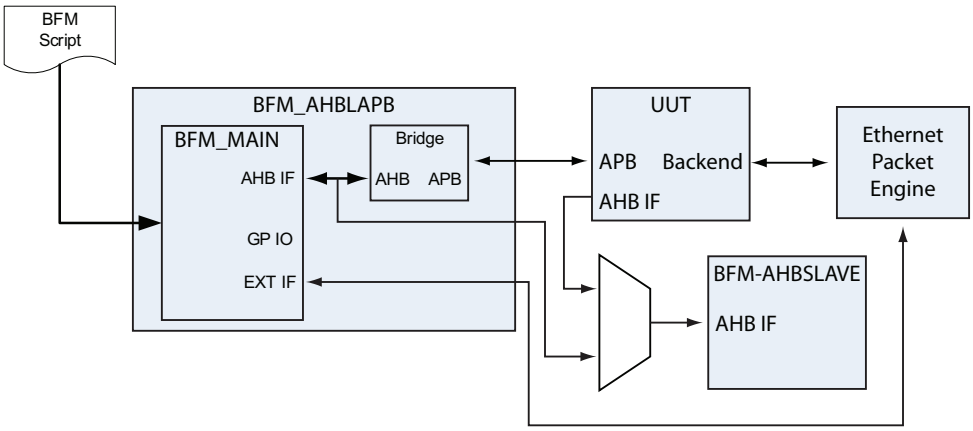

***Figure 1-4* • Testing an APB Master Block**

In Figure 1-4 we can see that the UUT has an AHB-Lite master interface that is connected via an AHB-lite arbitration and multiplexer function to the BFM-AHBSLAVE block. This allows both the BFM_AHBLAPB BFM and the UUT to read and write to the AHB slave BFM.

This setup also includes an Ethernet packet engine connected to the external interface on the AMBA BFM, allowing the BFM script to generate and verify Ethernet data packets. The external interface provides an address/data type interface rather than simple general purpose I/O (GPIO).

The BFM script would initially write known data frames to the BFM-AHBSLAVE, then program the UUT through its APB interface to transmit the data frame, and then wait for an interrupt event. Once started the UUT would read the BFM-AHBSlave and transmit the data frame, which would be captured by the Ethernet packet engine. When the UUT generates it completion interrupt the BFM script would continue and verify the expected data frame has been received by the packet engine.

The AMBA BFM by writing to special locations in the BFM-AHBSlave can cause it to vary its response rates etc. to allow extended testing the AHB Interface on the UUT.

2 – BFM_AHBL, BFM_APB and BFM_AHBLAPB

This lists the top level ports of the BFM-AHBAPL BFM, other BFMs have a subset of these signals. Table 2-1 lists the BFM Master Interface signals.

***Table 2-1* • BFM Master Interface Signals**

Signal	Type	Description
SYSCLK	In	Master clock input
SYSRSTN	In	Master reset input, active low
HCLK	Out	As per AHB specification
HRESETN	Out	As per AHB specification
HADDR[31:0]	Out	As per AHB specification
HBURST[2:0]	Out	As per AHB specification
HPROT[3:0]	Out	As per AHB specification
HSIZE[2:0]	Out	As per AHB specification
HTRANS[1:0]	Out	As per AHB specification
HWDATA[31:0]	Out	As per AHB specification
HWRITE	Out	As per AHB specification
HRDATA[31:0]	In	As per AHB specification
HREADYIN	In	As per AHB specification
HREADYOUT	Out	Indicates that the AHB bus is non READY. Internally in the BFM there is an AHB slave device that performs the AHB-APB bridge function.
HRESP	In	As per AHB specification
INTERRUPT[255:0]	In	Interrupt input. Supports 256 Interrupt inputs.
HSEL[15:0]	Out	The HSEL outputs. A[31:28] are used as a simple decode to generate the 16 select signals. When APB functions uses the APB slots overlap the HSEL(1) signal
PCLK	Out	As per APB specification
PRESETN	Out	As per APB specification
PADDR[31:0]	Out	As per APB specification
PENABLE	Out	As per APB specification
PWRITE	Out	As per APB specification
PWDATA[31:0]	Out	As per APB specification
PRDATA[31:0]	In	As per APB specification
PREADY	In	As per APB specification
PSLVERR	In	As per APB specification

***Table 2-1* • BFM Master Interface Signals (continued)**

Signal	Type	Description
PSEL[15:0]	Out	The PSEL outputs are generated based on the mode of the BFM. Default mode is that A[27:24] is used as a simple decode to generate the 16 select signals and actives when A[31:28] is “0001”, that means, the APB slots is at address 0x1n000000.
EXT_WR	Out	Extension Bus write signal. Synchronous to HCLK. Is asserted for a single cycle along with EXT_ADDR and EXT_DOUT.
EXT_RD	Out	Extension Bus read signal. Synchronous to HCLK. Is asserted for a single cycle along with EXT_ADDR, data samples on the following clock edge after the EXT_RD pulse, that is synchronous read assumed similar to AMBA buses.
EXT_WAIT	In	Extension Bus wait input used by EXT_WAIT instruction.
EXT_ADDR[31:0]	Out	Extension Bus address bus. Synchronous to HCLK.
EXT_DATA[31:0]	Inout	Extension Bus data bus. Synchronous to HCLK. Data is driven out when EXT_WR is true, otherwise is 'Z's. Data is sampled on the clock edge after EXT_RD is active (synchronous type read).
GP_OUT[31:0]	Out	Output signals that the BFM script can be set.
GP_IN[31:0]	In	Input signals that the BFM script can be tested.
FINSHED	Out	BFM has executed the quit instruction.
FAILED	Out	Indicates that the BFM detected an error.

Table 2-2 lists the parameters on the simulation model.

***Table 2-2* • BFM Master Generics**

Parameter	Default	Description
VECTFILE	test.vec	Specifies the vector file name.
MAX_INSTRUCTIONS	16364	Sets the maximum supported number of instruction words.
MAX_STACK	1024	Sets the maximum size of the internal stack used for the call/return instructions and local storage.
MAX_MEMTEST	65536	Sets the maximum memory size that the memtest command supports.
TPD	1	Sets the internal delay from SYSCLK to all outputs (except HCLK and PCLK) in ns. Can be used to offset clock insertion delays in the UUT.
ARGVALUE*N*	0	Sets the value that the $ARGVALUE*N* script label returns. *N* is from 0 to 99. Allows one hundred integer values to be passed into the script from the BFM instantiation.
DEBUGLEVEL	-1	Sets the default debug level 0 to 4 ("BFM Control" on page 27). If set to -1 (default) the DEBUG script command is enabled. When set (0-5) the DEBUG script command have no effect.

3 – BFM_AHBSLAVE and BFM_AHBSLAVEXT

This is a simple AHB based slave core; its function is similar to CoreAHBSRAM. There are two versions of this slave model (Figure 3-1):

- BFM_AHBSLAVE - provides an AHB interface
- BFM_AHBSLAVEEXT - provides a backdoor interface using the EXT* interface

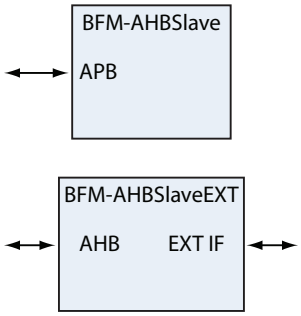

***Figure 3-1* • BFM_AHBSLAVE and BFM_AHBSLAVEEXT**

Parameters

Table 3-1 lists the BFM_AHBSLAVE parameters.

***Table 3-1* • Parameters**

Parameter	Type	Description
AWIDTH	Integer	Specifies the address bus width.
DEPTH	Integer	Specified the number of memory bytes actually implemented, may be less than 2**AWIDTH to reduce memory consumption by the simulator.
INITFILE	String	If specified the memory is initialized from the specified file.
ENFUNC	Integer	If set (>0) enables special control features. See "BFM Commands - Slave Cores" on page 35.
ENFIFO	Integer	If set (>0) enables the FIFO modelling function in the BFM, the parameter value sets the First In, First Out (FIFO) size (only on SLAVEEXT).
DEBUG	Integer	Sets the debug level 0: Disabled 1: Enabled
TPD	1	Sets the internal delay from HCLK to all outputs in ns. Can be used to offset clock insertion delays in the UUT.

***Table 3-1* • Parameters (continued)**

Parameter	Type	Description
ID	Integer	ID value is appended to debug messages to allow identification of the message source when multiple AHB slave models are being used.
EXT_SIZE	0,1,2	Configures the size of the external memory interface. 0: Byte Wide - data is read/written using EXT_DATA[7:0] 1: Half Word Wide- data is read/written using EXT_DATA[15:0] 2: Word Wide- data is read/written using EXT_DATA[31:0]

Interface Signals

Table 3-2 lists the BFM_AHBSLAVE interface signals.

***Table 3-2* • Interface Signals**

Signal	Type	Description
HCLK	In	As per the AHB specification
HRESETn	In	As per the AHB specification
HADDR[AWIDTH-1:0]	In	As per the AHB specification
HBURST[2:0]	In	As per the AHB specification
HMASTLOCK	In	As per the AHB specification
HPROT[3:0]	In	As per the AHB specification
HSIZE[2:0]	In	As per the AHB specification
HTRANS[1:0]	In	As per the AHB specification
HWDATA[31:0]	In	As per the AHB specification
HWRITE	In	As per the AHB specification
HRDATA[31:0]	Out	As per the AHB specification
HREADYIN	In	As per the AHB specification
HREADYOUT	Out	As per the AHB specification
HRESP	Out	As per the AHB specification
HSEL	In	As per the AHB specification
EXT_EN	In	Extension Bus enable signal. Must be active (high) for a read or write to occur
EXT_WR	In	Extension Bus write signal. Synchronous to HCLK. Is asserted for a single cycle along with EXT_ADDR and EXT_DATA.
EXT_RD	In	Extension Bus read signal. Synchronous to HCLK. Is asserted for a single cycle along with EXT_ADDR, data is generated on the clock edge after the EXT_RD pulse, that is, synchronous read assumed similar to AMBA buses.
EXT_ADDR[AWIDTH-1:0]	In	Extension Bus address bus. Synchronous to HCLK.

***Table 3-2* • Interface Signals (continued)**

Signal	Type	Description
EXT_DATA[31:0]	inout	Extension Bus data bus. Synchronous to HCLK. Data is sampled when EXT_WR is active. Data is sampled on the clock edge after EXT_RD is active (synchronous type read). Data is not driven at other times.
TXREADY	Out	Indicates that the internal TXFIFO is full, active high. (EXT Model only).
RXREADY	Out	Indicates that the internal RXFIFO is empty, active high (EXT Model only).

Note: The INITFILE is not reloaded when HRESETN is asserted. The EXT_EN, EXT_RD, EXT_WR, EXT_ADDR, EXT_DATA ports and EXT_SIZE generic are only on the BFM_AHBSLAVEXT model, the BFM_AHBSLAVE does not support the external interface. Only WORD (32-bit) aligned read and write cycles should be performed through the external interface. If an AHB write and External write to the same location occur at the same time the extension write wins.

FIFO Model

The AHB slave model has the ability to emulate FIFO as well as normal memory behavior. This is enabled by the ENFIFO generic. When enabled a TXFIFO and RXFIFO are created in the model in addition to the normal memory array. These FIFOs are at a fixed address controlled by the ENFUNC generic (see "BFM Commands - Slave Cores" on page 35).

The TXFIFO is set up to emulate a transmit FIFO in a device such as UART, data is intended to be written into the FIFO by the AHB side and read by the external interface, and the TXREADY flag indicates that the FIFO is not full.

The RXFIFO is set up to emulate a receive FIFO in a device such as UART, data is intended to be written into the FIFO by the external interface and read by the AHB interface, and the RXREADY flag indicates that the FIFO is not empty.

The FIFO model also supports special flag control logic to force empty/full conditions and to add latency to the READY signals to model latency caused posted writes within a system. Two counters are provided:

- LATCNT - sets the latency that the model de-asserts the READY signal.
- FEMCNT - sets the duration that the FIFO signals a full or empty after each data cycle.

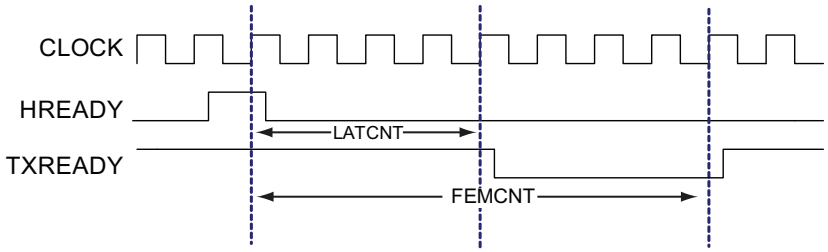

***Figure 3-2* • BFM_AHBSLAVE and BFM_AHBSLAVEEXT**

In Figure 3-2 the HREADY signal indicates the data transfer cycle. LATCNT delays the de-assertion of the TXREADY (or RXREADY) signal, TXREADY de-asserts if the FIFO becomes full or of FEMCNT is greater that LATCNT. TXREADY is then be reasserted when the FIFO is no longer full or when FEMCNT count expires assuming that the FIFO is not full. The same system applies to RXREADY. Should a second data transfer be attempted within the LATCNT or FEMCNT period the BFM detects an error and stop the simulation.

4 – BFM_APBSLAVE

his is a simple APB based slave core (Figure 4-1). There are two version of this slave model:

- BFM_APBSLAVE - Provides an APB interface
- BFM_APBSLAVEEXT - Provides a backdoor interface using the EXT* interface

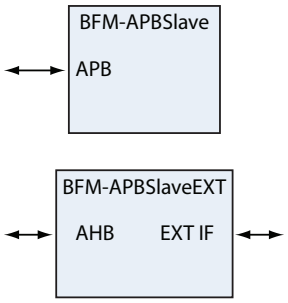

***Figure 4-1* • BFM_AHBSLAVE and BFM_AHBSLAVEEXT**

WARNING: The APB Slave is modelled as per the Microsemi APB byte handling guidelines with APB, the lowest two bits of the address bus are ignored and all transfers assume 32-bit data writes. This model correctly models 8-bit and 16-bit APB devices by setting the DWIDTH generic.

Parameters

Table 4-1 lists the BFM_AHBSLAVE parameters.

***Table 4-1* • Parameters**

Parameter	Type	Description
AWIDTH	Integer	Specifies the address bus width.
DEPTH	Integer	Specified the number of memory bytes actually implemented, may be less than 2**AWIDTH to reduce memory consumption by the simulator.
INITFILE	String	If specified the memory is initialized from the specified file.
ENFUNC	Integer	If set (>0) enables special control features. See "BFM Commands - Slave Cores" on page 35.
ENFIFO	Integer	If set (>0) enables the FIFO modelling function in the BFM, the parameter value sets the FIFO size (only on SLAVEEXT).
DEBUG	Integer	Sets the debug level 0: Disabled 1: Enabled
TPD	1	Sets the internal delay from PCLK to all outputs in ns. Can be used to offset clock insertion delays in the UUT.

***Table 4-1* • Parameters (continued)**

Parameter	Type	Description
ID	Integer	ID value is appended to debug messages to allow identification of the message source when multiple AHB slave models are being used.
EXT_SIZE	0,1,2	Configures the size of the external memory interface. 0: Byte Wide - data is read/written using EXT_DATA[7:0] 1: Half Word Wide- data is read/written using EXT_DATA[15:0] 2: Word Wide- data is read/written using EXT_DATA[31:0]

Interface Signals

Table 4-2 lists the BFM_APBSLAVE interface signals.

***Table 4-2* • Interface Signals**

Signal	Type	Description
PCLK	In	As per the APB specification
PRESETn	In	As per the APB specification
PADDR[AWIDTH-1:0]	In	As per the APB specification
PENABLE	In	As per the APB specification
PWRITE	In	As per the APB specification
PWDATA[DWIDTH-1:0]	In	As per the APB specification
PRDATA[DWIDTH-1:0]	Out	As per the APB specification
PREADY	Out	As per the APB specification
PSLVERR	Out	As per the APB specification
PSEL	In	As per the APB specification
EXT_EN	In	Extension Bus enable signal. Must be active (high) for a read or write to occur
EXT_WR	In	Extension Bus write signal. Synchronous to PCLK. Is asserted for a single cycle along with EXT_ADDR and EXT_DATA.
EXT_RD	In	Extension Bus read signal. Synchronous to PCLK. Is asserted for a single cycle along with EXT_ADDR, data is generated on the clock edge after the EXT_RD pulse, that is, synchronous read assumed similar to AMBA buses
EXT_ADDR[AWIDTH-1:0]	In	Extension Bus address bus. Synchronous to PCLK
EXT_DATA[DWIDTH-1:0]	inout	Extension Bus data bus. Synchronous to PCLK. Data is sampled when EXT_WR is active Data is sampled on the clock edge after EXT_RD is active (synchronous type read). Data is not driven at other times.

Note: The PWDATA and PRDATA bus widths can be modified by the DWIDTH generic. The INIT FILE is not reloaded when PRESETN is asserted. The EXT_EN, EXT_RD, EXT_WR, EXT_ADDR, EXT_DATA ports and EXT_SIZE generic are only on the BFM_APBSLAVEXT model, the BFM_APBSLAVE does not support the external interface. Only WORD aligned read and write cycles should be performed through the external interface. If an APB write and External write to the same location occur at the same time the extension write wins.

5 – Programming the BFMs

Master Models

The BFM-AMBA is scripted through a text file containing a list of bus cycles. The BFM supports BFM scripts similar to those used with the CoreMP7 and Cortex-M1 processors.

The BFM script is converted to a binary sequence by the BFM Compiler; it also verifies the syntax of the script. The binary file (*.vec) contains a sequence of 32-bit values, each represented by an 8 digit hexadecimal value. Libero® Integrated Design Environment (IDE) is configured to automatically compile the BFM script when ModelSim is invoked.

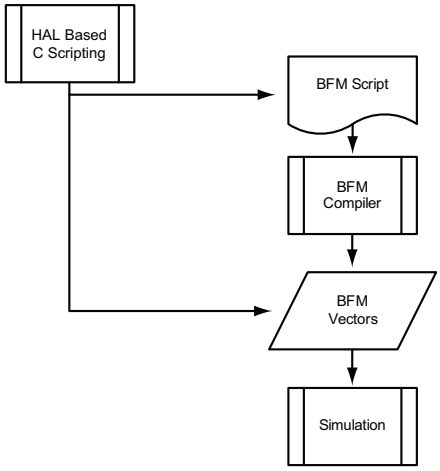

***Figure 5-1* • BFM Tool Flow**

Hello World BFM Script

The following example shows a BFM script that prints “Hello World” and then stops the simulation:

```
procedure main
  print "hello world"
return
```

Memory Read Write Test BFM Script

The following example is a BFM script that verifies the operation of memory space using simple read and write commands:

```
memmap MEMBASE 0x20000000

procedure main
  print "Memory Test"
  write     w MEMBASE 0x0000 0x12345678
  readcheck w MEMBASE 0x0000 0x12345678
  readcheck h MEMBASE 0x0000 0x5678
  readcheck h MEMBASE 0x0002 0x1234
  readcheck b MEMBASE 0x0000 0x78
  readcheck b MEMBASE 0x0001 0x56
  readcheck b MEMBASE 0x0002 0x34
  readcheck b MEMBASE 0x0003 0x12
return
```

In this example the base address of the memory device is set by the memmap command. Then a word write is used to write the test data, and it read and checked using word, half word and byte transfers.

The BFM also supports a memory test command that can be used to verify memory access rather than having to create a list of write and read operations as above

```
memmap MEMBASE 0x20000000
procedure main
  print "Automatic Memory Test"
  memtest MEMBASE 0x0000 1024 0 4000 566
return
```

In this example the BFM tests a block of memory at MEMBASE+0x0000 whose size is 1024 bytes, 4000 random memory write and read cycles is performed. The additional 0 parameter allows the memtest to be configured for some special conditions, see the full memtest command description.

"Simple BFM Script" on page 37 contains a complete example BFM master script.

Slave Models

The AHB and APB slave models are simple memory based core, write cycles write data and read cycles provide the same data back.

The model enables the memory locations to be initialized as well as some special control functions to vary response times etc.

WARNING: The APB Slave is modelled as per the Microsemi APB byte handling guidelines with APB, the lowest two bits of the address bus are ignored and all transfers assume 32-bit data writes. This model correctly models 8-bit and 16-bit APB devices by setting the DWIDTH generic.

6 – BFM Commands - Master Cores

The following commands are supported by the AMBA MASTER BFM.

The Clocks column indicates how many clock cycles an instruction takes; V indicates that it is variable based on the instruction parameters or AHB/APB response times.

See the example scripts in "Simple BFM Script" on page 37 for how the commands may be used.

Refer www.microsemi.com/soc/documents/SF_Bus_Functional_Model_UG.pdf for SmartFusion® BFM commands.

Basic Read and Write Commands

Table 6-1 lists basic read and write commands. These commands are compatible with the MP7 and M1 processors.

***Table 6-1* • Basic Read and Write Commands**

Basic Read and Write Commands	Description	Clocks
memmap *resource address*	Sets the base address of the associated with the resource.	0
write *width resource address data*	Perform a write cycle.	V
read *width resource address*	Perform a read cycle and echo the read data to the simulation log.	V
readcheck *width resource address data*	Perform a read cycle and check the read data.	V
poll *width resource address data*	Perform a read cycle until the read data matches the specified value. In this case a match is read_data & data = data.	V
waitfiq	Wait until an interrupt event occurs (FIQ pin). This is provided for Cortex-M1 compatibility reasons and assumes FIQ is connected INTERRUPT(0).	V
waitirq	Wait until an interrupt event occurs (IRQ pin). This is provided for Cortex-M1 compatibility reasons and assumes IRQ is connected INTERRUPT(1).	V
wait *cycles*	Wait for the specified number of clock cycles.	V

Enhanced Read and Write Commands

The commands listed in Table 6-2 provide enhanced read and write functions.

***Table 6-2* • Enhanced Read and Write Commands**

Enhanced Read and Write Commands	Description	Clocks
readstore *width resource address variable*	Perform a read cycle and stores the data in the specified variable.	V
readmask *width resource address data mask*	Perform a read cycle and check the read data. The data is masked as follows read_data & mask = data & mask.	V

***Table 6-2* • Enhanced Read and Write Commands (continued)**

Enhanced Read and Write Commands	Description	Clocks
pollmask *width resource address data mask*	Perform a read cycle until the read data matches the specified value. The data is masked as follows read_data & data = data & mask.	V
pollbit *width resource address bit val01*	Perform a read cycle until the specified bit matches the specified value.	V
waitint *intno*	Wait until an interrupt event occurs. Intno 0-255 specifies the interrupt input to monitor. If set to 256 any interrupt causes the instruction to complete.	V
memtest *resource addr size align cycles seed*	Perform a random based memory test. The BFM performs a sequence of mixed random byte, half and word, read or write transfers keeping track of the expected read values. It is ensured that a write occurs prior to a read of an address. Resource: base address of resource Addr: address offset in resource Size: size of block to be tested, must be power of 2. The maximum supported memory size is set by the MAX_MEMTEST generic. Align Bits [15:0] (values are integer values, not bit positions) 0: No special alignment occurs. 1: All transfers are forced to be APB byte aligned. 2: All transfers are forced to be APB half word aligned. 3: All transfers are forced to be APB word aligned as per Microsemi norms. 4: Byte writes are prevented Align Bits [18:16](values refer to bit positions). 16: fill - the memory array is pre-filled before random read/write cycles starts. 17: scan - the memory array is verified after the random read/write cycles complete. 18: restart - the memory test restarts, expecting the memory contents to remain unchanged from the previous memtest. Cycles: Specifies the number of accesses to be performed. May be set to zero allowing just fill or scan operation. Seed: Specified the seed value for the random sequence, any non zero integer.	V

***Table 6-2* • Enhanced Read and Write Commands (continued)**

Enhanced Read and Write Commands	Description	Clocks
memtest2 *baseaddr1 baseaddr2 size align cycles seed*	Similar to memtest command but two separate memory blocks are tested at the same time, set by the two baseaddr values. The same size and alignment is used for each block. The maximum size supported is MAX_MEMTEST/2.	V
ahbcycle *width resource address data control*	Perform an AHB cycle setting the address, data and control lines to the specified values. This command may be used to insert IDLE cycles etc. The control value is as follows Bit 0: HWRITE Bits [5:4]: HTRANS this sets the value placed on the HTRANS signals during the AHB cycle. Bits [10:8]: HBURST, this sets the value placed on the HBURST signals during the AHB cycle. Bit 12: HMASTLOCK, this sets the value placed on the HMASTLOCK signal during the AHB cycle. Bits [19:16]: HPROT, this sets the value placed on the HPROT signals during the AHB cycle. Multiple ahbcycle commands can be used to create non standard AHB test sequences.	V

When the write, read, readcheck, or readmask and all the following burst commands are used the AHB BFM pipelines the AHB bus operation, that means, it starts the next command in the following clock cycle, and checks the read data in a following clock cycle. A wait or flush command can be inserted to cause AHB idle cycles to be inserted between cycles.

The poll, pollmask, pollbit and readstore instructions are not pipelined, the AHB master inserts idle bus operations until the read operation completes and the read data has been checked.

Burst Support

Table 6-3 lists commands that enable you to create AMBA burst instructions. They also simplify memory filling and creating data tables.

***Table 6-3* • Burst Support**

Burst Support	Description	Clocks
writemult *width resource address data1 data2 data3 … data4*	Write multiple data values to consecutive addresses using a burst AMBA cycle.	V
fill *width resource address length start increment*	Fills memory starting with start value and increments each value as specified. To zero fill the last two values should be 0 0.	V
writetable *width resource address tableid length*	Writes the data specified in the specified tableid to consecutive addresses using a burst AMBA cycle.	V
readmult *width resource address length*	Reads multiple data values from consecutive addresses using a burst AMBA cycle. Data is discarded.	V
readmultchk *width resource address data1 data2 data3*	Reads multiple data values from consecutive locations and compares against the provided values.	V
fillcheck *width resource address length start increment*	Reads multiple data values from consecutive compares against the specified sequence specified as per the fill command.	V

***Table 6-3* • Burst Support (continued)**

Burst Support	Description	Clocks
readtable *width resource address tableid length*	Reads multiple data values from consecutive compares against the specified table values.	V
table *tableid data1 data2 data3 data4…datan*	Specifies a table of data containing multiple data values.	V
writearray *width resource address array length*	Writes the data contained in the array to consecutive addresses using a burst AMBA cycle.	V
readarray *width resource address array length*	Reads the AHB bus and stores the data in the array.	V

BURST OPERATION NOTES

1. Default operation of the BFM is to perform AHB BURST operations with HBURST="001", setting HTRANS to NONSEQ for the first transfer and to SEQ for all following transfers.
2. Using the setup noburst command the BFM can be made to initiate consecutive single cycles instead to achieve the required data transfers. In this case multiple AHB cycles HTRANS set to NONSEQ for all transfers.
3. During burst transfers the address increments based on the required transfer width. Thus if a byte transfer is requested the address increments by 1. If the X transfer width is used then the address increment can be controlled. This is very useful for bursting data to APB byte wide devices (see SETUP command).
4. A table may only contain 255 values.
5. Arrays are declared using int blah[100] instruction. In the read and write array instructions the command transfers data from the array element provided, the following starts the transfer at array item 0:

```
int array[100]
writearray w ahbslave 0x1000 array[0] 16
```

I/O Signal Support

Table 6-4 lists commands that support the 32 general purpose inputs and outputs on the BFM.

***Table 6-4* • I/O Signal Support**

External Interface Support	Description	Clocks
iowrite *data*	Write the data value to the IO_OUT output.	1
ioread *variable*	Reads the IO_IN input and stores the data in the specified variable.	1
iocheck *data*	Check the IO_IN input matches data.	1
iomask *data mask*	Check the IO_IN input matches data after applying the mask, io_in & mask=data & mask.	1
iosetbit *bit*	Set IO_OUT bit.	1
ioclrbit *bit*	Clear IO_OUT bit.	1
iotstbit *bit val01*	Test IO_IN bit is the specified value.	1
iowaitbit *bit val01*	Wait until IO_IN bit is the specified value.	V

External Interface

Table 6-5 lists the set of interfaces that enables the connection of external functions to the BFM. For instance, there may be an Ethernet packet generator used in the testbench that the BFM script can control.

***Table 6-5* • External Interface**

IO Signal Support	Description	Clocks
extwrite *addr data*	Write the data value to the extension interface at address.	1
extwrite *addr data1 data2 datan*	Write the data value to the extension interface starting at address. Address is incremented by 1 for each write.	1
extread *addr variable*	Read the extension interface and stores the data in the specified variable.	1
extcheck *addr data*	Read and check the extension interface.	1
extmask addr data mask	Read and check the extension interface ext_in & mask = data & mask.	1
Extwait	Wait for the EXT_WAIT input to be in active.	V

Flow Control

Table 6-6 lists the BFM flow control commands.

***Table 6-6* • Flow Control**

Flow Control	Description	Clocks
label *labelid*	Set a label in the BFM script, used to label instructions for jumps within a procedure. A label's scope is limited to the procedure it is used in.	0
procedure *labelid para1 para3 … para8*	Set a label in the BFM script for a call and name its parameters.	0
jump *labelid*	Jump to the specified label within the current procedure.	0
jumpz *labelid data*	Jump if the specified data value is zero.	0
jumpnz *labelid data*	Jump if the specified data value is non zero.	0
call *procedure para1 para2 para3 para4 etc*	Call the routine at the specified procedure in the BFM script. Up to eight parameters may be passed to the called routine. Calls can be recursive.	0
return	Return from the routine.	0
return data	Return from the routine returning the data value or variable. Return value is accessed using the $RETVALUE variable.	0
loop *para1 start end inc*	Repeat the instructions between loop and end loop. Para1 must have been declared using the int command. If not all the parameters are specified then the command is interpreted as below loop para 8: loop para 1 8 1 loop para 1 5: loop para 1 5 1 loop para 5 1: loop para 5 1 -1 loop para 1 5 1: loop para 1 5 1 The loop parameter can be used and modified within the loop. To exit a loop early set the loop variable to the termination value using the set command.	0

***Table 6-6* • Flow Control (continued)**

Flow Control	Description	Clocks
endloop	End of loop.	0
if *variable*	The instructions between if and the following else or endif is performed if variable is non zero. If/else/endif can be nested. Supported operators are listed in Table 6-8 on page 26.	0
if *variable op variable*	The instructions between if and the following else or endif is performed if the expression is true, for example a >=b. If/else/endif can be nested.	0
Ifnot *variable*	The instructions between if and the following else or endif is performed if variable is zero. Ifnot/else/endif can be nested.	0
Ifnot *variable op variable*	The instructions between if and the following else or endif is performed if the expression is false, for example a >= b. Ifnot/else/endif can be nested. Supported operators are listed in Table 6-8 on page 26.	0
else	May be inserted between the if and endif statements	
endif	End of if.	0
case *variable*	Specifies the variable to use in the case/when sequence.	0
when *data*	If the preceding case statement variable matches the data value then the following set of instructions is executed. (See notes below)	0
default	If non of the when clauses are true then the default is executed in a case statement.	0
endcase	End of the case statement.	
while *variable*	The instructions between while and endwhile is performed as long as variable is non zero. While/endwhile can be nested.	0
endwhile	End of while loop.	0
compare *variable data mask*	Compares variable to the specified data value. The mask value is optional. If the compare fails then an error is recorded.	0
nop	Do nothing for a clock cycle (same as wait 1).	1
stop *N*	Stop the simulation. N specifies the VHDL assertion level or the Verilog generated message. 0:Note 1:Warning, 2:Error, 3:Failure	0
wait *N*	Pause the BFM script operation for N clock cycles.	V
waitns *N*	Pause the BFM script operation for N nano seconds *Note: The BFM waits for the specified time to expire and then restart at the next clock edge. Any single wait longer than 5 microseconds causes a simulation error. Instead, use multiple waits with less wait time.*	V
waitus *N*	Pause the BFM script operation for N micro seconds *Note: The BFM waits for the specified time to expire and then restart at the next clock edge. Any single wait longer than 5 microseconds causes a simulation error. Instead, use multiple waits with less wait time.*	V

***Table 6-6* • Flow Control (continued)**

Flow Control	Description	Clocks
flush *N*	Wait for any pending read or write cycles to complete, and then wait for N additional clock cycles. The BFM is pipelined and it can start processing following instructions before the current one has completed, especially when AMBA read cycles are in progress.	V
quit	Terminate the BFM and assert the FINISHED output.	1

If statements can be use a single variable or comparison

```
          If y                -- will be true I y is non zero
              nop
          else
              flush
          endif
or
          If x >= 6        -- will be true if x>=6  (note spaces)
              nop
          else
              flush
          endif
```

Case statements are of the form

```
case i
when 1
set y  101
when 2
set y  102
when 3
set y  103
when 1
set y  10000
default
set y 67677
endcase
```

When processing case statements the BFM compares the case value to EVERY when value and if equal execute the following set of statements until it finds the next when. In the above example when i=1 both the first and last set of when statements are executed. If no match is found then the default is executed.

A subroutine is declared using the procedure command then the passed parameters may be referred to by the declared name.

```
procedure example address data;
write b UART address data
return
```

Up to eight parameters may be passed.

Variables

The commands in Table 6-7 allow a BFM script to use variables, etc. If variables are declared within a procedure they are local to the procedure, if declared outside a procedure then they are global. Variables may only be assigned (set) within a procedure.

Table 6-7 • **Variables**

Parameters	Description	Clocks
int *para1 … paran*	Declare variable.	0
int *array[N]*	Declare an array variable of N elements. The maximum supported array size is 8192.	0
set *para1 value*	Sets a variable to have an integer value. The parameter must have been declared with the int command.	0
set *paraS paraA op paraB* set *paraS paraA op paraB op paraC op parad …*	The BFM sets paraS to a function of paraA and paraB. The function is specified by op. 1. paraA and paraB can be integer values, or another declared parameter within the procedure. 2. There is no precedence; the function is simply evaluated left to right. 3. Bracketed expressions may only contain expressions that can be evaluated at compile time that is, they must not contain any variables declared using the int command. 4. THERE MUST BE A SPACE ON EITHER SIDE OF THE OPERATOR.	0
compare *variable data mask*	Compares variable to the specified data value. The mask value is optional. If the compare fails then an error is recorded.	0
cmprange *variable datalow datahigh*	Checks the variable is in the data range specified. If not an error is recorded.	0

Table 6-8 lists the supported operators for the set command; these are evaluated during run time by the BFM.

Table 6-8 • **Operators**

Operator	Function
None	
+	A+B
-	A-B
*	A * B
/	A / B (integer division)
MOD	Modulus (remainder)
**	A ** B
AND	A and B
OR	A or B
XOR	A xor B
&	A and B
|	A or B
^	A xor B

***Table 6-8* • Operators (continued)**

Operator	Function
CMP	A == B (uses XOR operator - result is zero if A==B)
<<	A shifted left by B bits (infill is 0)
>>	A shifted right by B bits (infill is 0)
==	Equal (result is 1 if true else 0)
!=	Not Equal (result is 1 if true else 0)
>	Greater than (result is 1 if true else 0)
<	Less than (result is 1 if true else 0)
>=	Greater than or equal (result is 1 if true else 0)
<=	Less than or equal (result is 1 if true else 0)
SETB	Sets bit B in A
CLRB	Clears bit B in A
INVB	Inverts bit B in A
TSTB	Tests bit B in A (result is 1 if bit set else 0)

BFM Control

Table 6-9 lists extended control functions for corner testing, etc.

***Table 6-9* • BFM Extended Control Functions**

BFM Control	Description	Clocks
version	Prints versioning information for the BFM in the simulation.	0
setup N X Y	Allows advanced configuration options and corner case settings, see Table 6-14 on page 32.	0
reset N	Asserts HRESETN for N clock cycles. If N is not specified then HRESETN is asserted for a single clock cycle. The script continues to execute whilst HRESET is asserted allowing reset conditions to be checked.	N
stopclk *N*	Stopclk 1 stops HCLK, the clock is held high after its rising edge. Stopclk 0 restarts HCLK.	1
timeout N	Sets an internal timeout value in clock cycles which trigger if the BFM stalls. Default timeout is 512 clocks.	0
drivex N	Forces the AHB/APB signals to an X condition. Bit 3: Sets HCLK/PCLK to X Bit 2: Sets HRESETN/PRESETN to X Bit 1: Sets AHB/APB write data to X Bit 0: Sets AHB/APB address and control lines to X Setting back to 0 makes the BFM act as normal.	0
print "string"	Prints the string in the simulation log, max string length is 256 characters.	0
header "string"	Prints a separating line off hash's in the simulation log followed by the string, max string length is 256 characters.	0

***Table 6-9* • BFM Extended Control Functions (continued)**

BFM Control	Description	Clocks
print “string %d %08x” para1 para2 ….	Print with support of print formatting. Up to 7 parameters may be used.	0
header “string %d %08x” para1 para2 …..	Header with support of print formatting. Up to 7 parameters may be used.	0
debug N	Controls the verbosity of the simulation trace 0: No Simulation log 1: Only text strings printed 2: Instructions logged 3: All Read and Write Transfers logged 4: Full debug trace	0
hresp N	AHB Error Response Handling. N=0 Stop simulation if error is asserted. N=1 Ignore response error. N=2 Check that the previous cycle caused an error response, and revert to mode 0. The AHB-APB bridge translates the PSLVERR to a AHB error response.	1
hprot protvalue	The following cycles use the specified HPROT value. The BFM defaults HPROT to b0011 AHB Spec 3.7.	0
lock N	Lock 1 asserts LOCK on the next AHB cycle and stays on until a LOCK 0 command is executed, typical operation is Lock 1 Read w ahbslave Write w ahbslave Lock 0	0
burst N	The following cycles use the specified HBURST value.	0
echo *D0 D1 D2... D7*	This simply lists the parameter values in the simulation log window. The command can help in the debug of bfm scripts using calls etc.	0
checktime *min max*	This allows the number of clock cycles that the previous instruction took to be executed, the two parameters specify the allowed min and max values (in clock cycles). The instruction waits for any internal pipelined activity to complete. The command can be used to verify the number of clock cycles that an AHB cycle took to complete, and includes both the address and data phases. A 16-word burst with zero wait states take 17 cycles. It can also be used to check the how long a poll, waintint, waitirq, waitfiq, iowait or extwait instruction took to complete. If the check fails an error is recorded.	V
starttimer	Start an internal timer (clock cycles)	0
checktimer *min max*	This allows the number of clock cycles since the starttimer instruction was executed to be checked. The two parameters specify the allowed min and max values (in clock cycles). The instruction waits for any internal pipelined activity to complete. If the check fails an error is recorded.	1

***Table 6-9* • BFM Extended Control Functions (continued)**

BFM Control	Description	Clocks
setfail	Sets the BFM FAILED output.	1
setrand para	Sets the internal random number seed.	0

BFM Compiler Directives

Table 6-10 lists instructions used by the compiler rather than used in the vector files.

***Table 6-10* • Compiler Directives**

Compiler Directives etc.	Description	Clocks
include *filename* include "*filename*"	Include another BFM file, include files may also contain include files. The filename should be double quoted when filenames are case sensitive.	0
memmap *resource address*	Enumerates the base address of a resource. If a resource is declared within a procedure then its scope is limited to that procedure. Once declared the resource cannot be changed.	0
constant *symbol value*	Sets a symbol to have an integer value. If a constant is declared within a procedure then its scope is limited to that procedure. Once declared the constant cannot be changed.	0
#setpoint m0 m1 m2 …	Sets the multiplier value to use when points are found in an integer. By default all values are 256, thus the following integer values become "3.5" => 0x0305 "3.4.5" = 0x030405 "1.3.4.5" = 0x01030405 Using "#setpoint 2048,1024,32,1" allows 1553B command word mapping "3.1.4.6" => 0x1c86.	0
;	Commands may be terminated by a semicolon.	0
#	Comment, may also be in the middle of a line, must be followed by a space.	0
--	Comment, may also be in the middle of a line.	0
//	Comment, may also be in the middle of a line.	0
/* */	All code between these symbols is commented out.	0
\	Instruction continued on next line, useful for table commands.	0

Supported C Syntax in Header Files

The BFM Compiler reads in C header files (Table 6-11) created by the SmartDesign system for cores with predefined register maps. These header files typically include all the register address and bit definitions.

Table 6-11 • **C Syntax in Header Files**

Compiler Directives, etc.	Description
include *filename* include "*filename*"	Include the C header files. The filename should be double quoted when filenames are case sensitive.
#define *symbol value*	Define a constant value. Value should be simple integer value typically 1234 or 0x1234.
#define *symbol*	Define a constant and default its value to 1.
#ifndef *symbol*	If the symbol has not already been defined include following lines until #endif statement.
#ifdef *symbol*	If the symbol has already been defined include following lines until #endif statement.
#endif	Restart including.

Parameter Formats

Table 6-12 describes the parameter formats used in the preceding Command Descriptions.

Table 6-12 • **Parameter Formats**

Parameter	Type/Values	Description
Integer		Integer value using the following syntax: **0x1245ABCD**: hexadecimal **0d12343456**: decimal **0b101010101**: binary **12345678**: decimal **$XYZ**: Special value. **Mnopq**: symbol, procedure parameter or declared variable. **(12+67)**: If a value is contained in parenthesis the compiler attempts to evaluate the expression. The expression evaluator supports the same set of operators as the set command. In this case parenthesis may be used. **1.3.4.5**: See the #setpoint compiler directive. **Mnopq[100]**: An array element, the index may be variable, but not another array, which means only single dimensional arrays are supported.
Symbol		ASCII string staring with a letter.
Resource	Integer	Integer specifying the base address of a resource. Microsemi recommends that you declare the base address using the memap command.
Address	Integer	Specifies the address offset from its base address.
Width	b,h,w, x	Specifies whether byte, half word or word access. X specifies a special Transfer mode see the setup commands (Table 6-14 on page 32).
Cycles	Integer	
Mask	Integer	

***Table 6-12* • Parameter Formats (continued)**

Parameter	Type/Values	Description
Bit	0 to 31	Integer that specifies the bit number.
Tableid	Symbol	Label used to identify a table.
Val01	0 or 1	Integer, only 0 and 1 are legal.
labelid	Symbol	Label used by flow control instructions.
String		A string must be enclosed in " ".

$ Variables

The BFM supports some special integer values that may be specified rather than immediate data or variables. The supported $ variables are listed in Table 6-13.

***Table 6-13* • $ Variables**

$ Variable	Description
$RETVALUE	Is the value from the last executed return instruction.
$ARGVALUEn	Is the value of the ARGVALUEn generic, n is 0 to 99. For example, $ARGVALUE4.
$TIME	Current Simulation time in ns.
$DEBUG	Current DEBUG level.
$LINENO	Current script line number.
$ERRORS	Current internal error counter value.
$TIMER	Returns the current timer value, see starttimer instruction.
$LASTTIMER	Returns the timer value from the last checktimer instruction.
$LASTCYCLES	Returns the number of clocks from the last checktime instruction.
$RAND	Returns a pseudo random number. The number is a 32-bit value; the random function is a simple CRC implementation.
$RANDSET	Returns a pseudo random number, and remembers the seed value.
$RANDRESET	Returns a pseudo random number after first resetting the seed value to that when the $RANDSET variable was used. This causes the same random sequence to be regenerated.
$RANDn	As above but the random number is limited to n bits.
$RANDSETn	As above but the random number is limited to n bits.
$RANDRESETn	As above but the random number is limited to n bits.

These variables are can be used as below

```
write b resource address $ARGVALUE5
compare $RETVALUE 0x5677
set variable $RETVALUE

fill w ahbslave 0x30 4 $RANDSET   $RAND
fillcheck w ahbslave 0x30 2 $RANDRESET $RAND
```

The multiple ARGVALUES can be used to pass core configuration information to the script to allow the test script to modify its behavior based on the core configuration.

If $RAND is specified for the data increment field of the fill and fillcheck instructions then the data sequence increments by the same random value for each word.

Setup Commands

Table 6-14 details Setup mode commands.

Table 6-14 • **Setup Commands**

Command	N	X Y	Description
widthX	1	*Size Ainc*	Sets the behavior of the 'X' memory access mode. Size specifies the HSIZE value 0:B 1:H 2:W ainc specifies the Address increment on bursts.
autoflush	2	*0|1*	When set the BFM stops pipelining AHB transactions, IDLE cycles is inserted until the current read/write completes.
xrate	3	*Rate*	Sets the transfer rate using during bursts, when non-zero the specified number of clock cycles are inserted as BUSY cycles between data transfers by controlling HTRANS.
noburst	4	*0|1*	When set all burst operations are converted into consecutive NONSEQ transactions on the AHB bus.
align	5	*Mode*	Sets the AHB data alignment mode. 0: Data is correctly aligned for a 32-bit AHB bus based on the address and size specified 1: Reserved 2: Reserved 8: No data alignment is performed. The data is written/read from the bus as provided
endsim	6	*0 to 4*	Controls what the BFM does on completion (VHDL Only). 0: Stops in an IDLE state with simulation running (default) 1: Executes an assert with severity NOTE 2: Executes an assert with severity WARNING 3: Executes an assert with severity ERROR 4: Executes an assert with severity FAILURE
endsim	7	*0 to 2*	Controls what the BFM does on completion (Verilog Only). 0: Stops in an IDLE state with simulation running (default) 1: Executes a $stop 2: Executes a $finish

Note: It is recommended that constants are use for the N value to enhance readability that is,

```
Constant  C_WidthX 1
Constant  C_Xrate 3
Setup C_WidthX 0 4
Setup 2 1      -- enable autoflush
```

HSEL and PSEL Generation

The Master models generate HSEL and PSEL as shown in Table 6-15.

***Table 6-15* • HSEL and PSEL Generation**

HSEL	Address Range
0	0x00000000 to 0x0FFFFFFF
1	0x10000000 to 0x1FFFFFFF
2	0x20000000 to 0x2FFFFFFF
3	0x30000000 to 0x3FFFFFFF
4	0x40000000 to 0x4FFFFFFF
5	0x50000000 to 0x5FFFFFFF
6	0x60000000 to 0x6FFFFFFF
7	0x70000000 to 0x7FFFFFFF
8	0x80000000 to 0x8FFFFFFF
9	0x90000000 to 0x9FFFFFFF
10	0xA0000000 to 0xAFFFFFFF
11	0xB0000000 to 0xBFFFFFFF
12	0xC0000000 to 0xCFFFFFFF
13	0xD0000000 to 0xDFFFFFFF
14	0xE0000000 to 0xEFFFFFFF
15	0xF0000000 to 0xFFFFFFFF

PSEL	Address Range
0	0x10000000 to 0x10FFFFFF
1	0x11000000 to 0x11FFFFFF
2	0x12000000 to 0x12FFFFFF
3	0x13000000 to 0x13FFFFFF
4	0x14000000 to 0x14FFFFFF
5	0x15000000 to 0x15FFFFFF
6	0x16000000 to 0x16FFFFFF
7	0x17000000 to 0x17FFFFFF
8	0x18000000 to 0x18FFFFFF
9	0x19000000 to 0x19FFFFFF
10	0x1A000000 to 0x1AFFFFFF
11	0x1B000000 to 0x1BFFFFFF
12	0x1C000000 to 0x1CFFFFFF
13	0x1D000000 to 0x1DFFFFFF
14	0x1E000000 to 0x1EFFFFFF
15	0x1F000000 to 0x1FFFFFFF

This decoding is a simple decode of the upper eight address lines. If different decoding is required then separate HSEL and PSEL decode logic can be created leaving the HSEL and PSEL outputs unconnected. Internally HSEL(1) "0x1xxxxxxx" is used to select the internal APB bridge on the APB and AHBLAPB models.

On the APB only BFM Model the complete address range is mapped to APB. The default PSEL decoding ignores the upper four address bits.

7 – BFM Commands - Slave Cores

The slave cores (AHB and APB) by default respond with zero wait state cycles. When ENFUNC is greater than 0 the slave core allows its behavior to be varied for corner case testing. The slave model catches AHB or APB writes to a 256 byte address space located at the address specified by ENFUNC and uses the written data to alter its behavior, as shown in Table 7-1.

Table 7-1 • **BFM Commands - Slave Cores**

Address	Description
ENFUNC+0x00	Set the HRESP ERROR or PSLVERR response on the Nth access after this one.
ENFUNC+0x04	Bits [7:0]: Set the number of wait cycles, that is, HREADY/PREADY timing, values 0 to 255 may be used. Bit 8: If this bit is set then the number of inserted wait cycles is random up to the value specified in bits [7:0], these bits must be a power of 2 that is, 1,2,4,8,16….,128.
ENFUNC+0x08	Sets the debug level.
ENFUNC+0x0C (12)	Zero the memory.
ENFUNC+0x10 (16)	Write test pattern to memory. Pattern is a decrementing sequence starting at 255, that is byte values 0xff 0xfe etc., or viewed as words 0xFCFDFEFF 0xF8F9FAFB etc.
ENFUNC+0x14 (20)	The return data value contains the HTRANS, HSIZE, HPROT and HBURST values from the previous access cycle. [1:0] HTRANS[1:0] [6:4] HSIZE[2:0] [11:8] HPROT[3:0] [14:12] HBURST[2:0] [16] HWRITE [17] HMASTLOCK AHB Slave Only is Reserved on APB slave.
ENFUNC+0x18 (24)	Sets the slaves response to misaligned transfers. The default mode is "0001", that is, the AHBSLAVE causes a simulation ERROR on a misaligned transfer occurring Bit 0: Generate Simulation ERROR. Bit 1: Generate a HRESP error. Bit 2: Make the device read only, writes are treated as errors. Bit 3: Allow write on misalignment. AHB Slave Only is Reserved on APB slave.
ENFUNC+0x1C (28)	Data writes to this address is delayed by the number off clocks specified at ENFUNC+0x20.
ENFUNC+0x20 (32)	Clock cycle delay until reads from ENFUNC+0x1c contains the last written value.
ENFUNC+0x24 (36)	Reinitialize the memory from the vector file. A FLUSH 2 command should be used after this command is issued.
ENFUNC+0x28 (40)	Dump the memory file to a vector file in a format that can reloaded. Log file is “imageX.txt” where X is ID generic value.
ENFUNC+0x2C (44)	Last Read or Write address, can be used to check that another master accessed as expected. Address is word aligned.

***Table 7-1* • BFM Commands - Slave Cores (continued)**

Address	Description
ENFUNC+0x30 (48)	Last Read or Write data value, can be used to check that another master accessed as expected.
ENFUNC+0x34 (52)	Special Mode Enables. Bit 0: When 0 the slave behaves in AMBA compliant way returning 0's on the databus when not being read. When 1 X values are provided on the RDATA bus when not being read.
ENFUNC+0x38 to ENFUNC+0x7C	Reserved.
ENFUNC+0x80 (128)	Transmit FIFO Data In port. (written by AHB).
ENFUNC+0x84 (132)	Transmit FIFO Data Out port (read by external).
ENFUNC+0x88 (136)	Transmit FIFO count. Reads provide current count. Writes 0x00000000 resets the FIFO to zero count.
ENFUNC+0x8C (140)	Transmit FIFO HREADY to TXREADY latency time (LATCNT). When 0 the TXREADY is de-asserted immediately after the data cycle. When >0 that is N then TXREADY is de-asserted N clock cycles after the data cycle. This is to model latency on the TXREADY deassertion.
ENFUNC+0x90 (144)	Transmit FIFO force FULL. (FEMCNT) When non zero this forces the FIFO to signal that it is full for N clock cycles after data cycle. This value must be greater than 0x8c.
ENFUNC+0x94 (148)	Swap TXREADY and RXREADY outputs Bit [0:0]: Normal Operation Bit [0:1]: RXREADY and TXREADY swapped
ENFUNC+0x98 to ENFUNC+0xBC	Reserved.
ENFUNC+0xA0 (160)	Receive FIFO Data In port. (written by external).
ENFUNC+0xA4 (164)	Receive FIFO Data Out port (read by AHB).
ENFUNC+0xA8 (168)	Receive FIFO count. Reads provide current count. Writes 0x00000000 reset the FIFO to zero count.
ENFUNC+0xAC (172)	Receive FIFO HREADY to RXREADY latency time (LATCNT). When 0 the RXREADY is de-asserted immediately after the data cycle. When >0 that is, N then RXREADY is de-asserted N clock cycles after the data cycle.
ENFUNC+0xB0 (176)	Receive FIFO force EMPTY (FEMCNT). When non zero this forces the FIFO to signal that it is empty for N clock cycles after data cycle. This value must be greater than 0xAC. This is to model latency on the RXREADY de-assertion.
ENFUNC+0xB4 to ENFUNC+0xFC	Reserved.

The control space above can only be read or written through the main AHB/APB interface. The external interface backdoor access supports read and write cycles to the main memory array only. Should an AHB/APB write access to the same location occur at the same time the external backdoor access takes precedence.

A – Simple BFM Script

The following shows an example BFM script that for testing a APB core.

```
##############################################
# Core1553BRT APB Test Harness
##############################################

#setpoint 2048,1024,32,1   # 1553B CW formatting 31.1.31.31

# Global Variables
# These are inherited from the Parameters set in CC
int  FAMILY
int  CLKSPD
int  CLKSYNC
int  LOCKRT
int  BCASTEN
int  LEGMODE
int  SA30LOOP
int  INTENBBR
int  TESTTXTOUTEN
int  INT_POLARITY
int  VERIF
int  TBNRTS

# APB base address of each RT
memmap base_RT0 0x10000000
memmap base_RT1 0x11000000
memmap base_RT2 0x12000000
memmap base_RT3 0x13000000
memmap base_RT4 0x14000000
memmap base_RT5 0x15000000
memmap base_RT6 0x16000000
memmap base_RT7 0x17000000

# Registers
constant R_CONTROL   0x1F80
constant R_INTERRUPT 0x1F84
constant R_VWORD     0x1F88
constant R_LEGREG0   0x1FC0

constant NRTS  4
int  READBACK
int CWORD

--------------------------------------------------------------------------------

procedure main
  int doall
  int vw;
  int RT;
  int SA;
  int WC;
  int base;
  int isrtl

  # get parameter settings
  set  VERIF        $ARGVALUE0
  set  TBNRTS       $ARGVALUE1
  set  FAMILY       $ARGVALUE2
  set  CLKSPD       $ARGVALUE3
```

```
set  CLKSYNC      $ARGVALUE4
set  LOCKRT       $ARGVALUE5
set  BCASTEN      $ARGVALUE6
set  LEGMODE      $ARGVALUE7
set  SA30LOOP     $ARGVALUE8
set  INTENBBR     $ARGVALUE9
set  TESTTXTOUTEN $ARGVALUE10
set  INT_POLARITY $ARGVALUE11

set  READBACK  FAMILY >= 16           # AX/APA versions do not allow RAM readback
set  ISRTL     FAMILY == 0
if ISRTL                              # Also if RTL code the readback allowed
  set READBACK 1
endif
set CWORD 1                           # Initial transmissions on Bus A

header "Core1553BRT APB Test Harness"
print "  CLKSPD:%0d", CLKSPD
print "  CLKSYNC:%0d", CLKSYNC
print "  LEGMODE:%0d",LEGMODE
print "  LOCKRT:%0d",LOCKRT
print "  TESTTXTOUTEN:%0d",TESTTXTOUTEN
print "  INT_POLARITY:%0d",INT_POLARITY
print "   "

if READBACK
  print "  RAM Readback Allowed "
endif
ifnot READBACK
  print "  RAM Readback Not Allowed "
endif

setup 1 1 4            -- Set the BFM to operate bursts using Half Words
                              -- with an address increment of 4
timeout (50*20*50)     -- increase default timeout to allow >50 1553B Words
debug 1

#-------------------------------------------------------------------------------
# Simple Test Example using RT1

# Check core Version
readmask  x base_rt1 R_INTERRUPT 0x4800 0xF900  # Check correct core version

# Enable all sub addresses on RT1
call setup_legal_mode base_RT1

# create two messages and set up data patterns
#        Addr   Control   CW     CW2     SW     SW     DP     Status Reserved
extwrite 0x0100 0x0002  1.0.1.4  0.0.0.0 0x0000 0x0000 0x0200 0       0   # BC to RT
extwrite 0x0108 0x0002  1.1.1.3  0.0.0.0 0x0000 0x0000 0x0300 0       0   # RT to BC
extwrite 0x0110 0x0000
extwrite 0x0200  1 2 3 4 5 6 7 8 9 10        # Test Data in BC
fill x base_RT1 0x1080 32 0x1000 1          # Test data in the RT SA 1 for TX command

# Cause the 1553B BC BFM to transmit the message and wait for completion
extwrite 0 0x0001 0x0100
extwait

# Check returned data
extcheck 0x0300 0x1000
extcheck 0x0301 0x1001
extcheck 0x0302 0x1002

#-------------------------------------------------------------------------------
# Verification Tests
```

```
  if verif
    header "Running More Complex Set of Verification Tests"
    call setup_legal_mode base_RT0
    call setup_legal_mode base_RT1
    call setup_legal_mode base_RT2
    call setup_legal_mode base_RT3
    call testmemory base_RT1 0000 64

    call test_rxtx;
    call test_control
    call test_interrupt

    call test_legality_mode
    call test_bcbfm

    header "Verification Tests Complete"
    print "  CLKSPD:%0d", CLKSPD
    print "  CLKSYNC:%0d", CLKSYNC
    print "  LEGMODE:%0d",LEGMODE
    print "  LOCKRT:%0d",LOCKRT
    print "  TESTTXTOUTEN:%0d",TESTTXTOUTEN
    print "  INT_POLARITY:%0d",INT_POLARITY

  endif;

  print "End of tests"

return


###############################################################################
# Test 1553B message Support in BC BFM

procedure test_bcbfm

#Set up Message Tables
#        Addr      Control     CW     CW2   SW     SW     DP     Status Reserved
extwrite 0x0100    0x0002  1.0.1.4  0.0.0.0 0x0000 0x0000 0x0200 0      0       ! BC to RT
extwrite 0x0108    0x0002  1.1.1.3  0.0.0.0 0x0000 0x0000 0x0300 0      0       ! RT to BC
extwrite 0x0110    0x0012  2.0.2.4  1.1.1.4 0x0000 0x0000 0x0400 0      0       ! RT1 to RT2
extwrite 0x0118    0x0002 31.0.1.3  0.0.0.0 0x0000 0x0000 0x0500 0      0       ! BC to
RT BCast
extwrite 0x0120    0x0012 31.0.2.5  1.1.1.5 0x0000 0x0000 0x0600 0      0       ! RT to
RT BCast
extwrite 0x0128    0x0002  1.1.0.1  0.0.0.0 0x0000 0x0000 0x0800 0      0       ! Mode
Code NODATA
extwrite 0x0130    0x0002  1.1.0.16 0.0.0.0 0x0000 0x0000 0x0000 0      0       ! Mode
Code RT TX VW
extwrite 0x0138    0x0002  1.0.0.17 0.0.0.0 0x0000 0x0000 0x1234 0      0       ! Mode
Code RT RX
extwrite 0x0140    0x0002 31.1.0.1  0.0.0.0 0x0000 0x0000 0x0000 0      0       ! Bcast
Mode Code NODATA
extwrite 0x0148    0x0002 31.0.0.17 0.0.0.0 0x0000 0x0000 0x5678 0      0       ! Bcast
Mode Code RT RX
extwrite 0x0150    0x0000

extwrite 0x0200  1 2 3 4 5 6 7 8 9 10
extwrite 0x0500  9 8 7 6 5 4

# Put known data in RT 1 SA 1 and Vector Word
fill x  base_RT1 0x1080 32 0x1000 1
write x base_RT1 R_VWORD 0xCAFE

#Now do the messages one by one
extwrite 0 0x0003 0x0100         # BC to RT
```

```
extwait
extwrite 0 0x0003 0x0108         # RT to BC
extwait
extcheck 0x0300 0x1000           # and check returned data
extcheck 0x0301 0x1001
extcheck 0x0302 0x1002
extwrite 0 0x0003 0x0110         # RT1 to RT2
extwait
extwrite 0 0x0003 0x0118         # BC to RT BCast
extwait
extwrite 0 0x0003 0x0120         ! RT to RT BCast
extwait
extwrite 0 0x0003 0x0128         ! Mode Code NODATA
extwait
extwrite 0 0x0003 0x0130         ! Mode Code RT TX VW
extwait
extcheck 0x0135 0xCAFE           ! check VW
extwrite 0 0x0003 0x0138         ! Mode Code RT RX
extwait
extwrite 0 0x0003 0x0140         ! Bcast Mode Code NODATA
extwait
extwrite 0 0x0003 0x0148         ! Bcast Mode Code RT RX
extwait

header "All messages as a burst"
timeout (50*20*10*10)
extwrite 0 0x0001 0x0100         # Repeat all messages
extwait
wait 100
return


##############################################################################
# Test Control Register

procedure test_control
int sw;
int bit;

  #No Bits
  write x base_rt0 R_CONTROL 0x8000
  call rx_message 0 8 1 0x0000 1
  set sw $RETVALUE;
  compare sw 0x0000

  #SREQUEST
  write x base_rt0 R_CONTROL 0x8001
  call rx_message 0 8 1 0x0000 1
  set sw $RETVALUE;
  compare sw 0x0100

  #RTBUSY
  write x base_rt0 R_CONTROL 0x8002
  call rx_message 0 8 1 0x0000 1
  set sw $RETVALUE;
  compare sw 0x0008

  #SSFLAG
  write x base_rt0 R_CONTROL 0x8004
  call rx_message 0 8 1 0x0000 1
  set sw $RETVALUE;
  compare sw 0x0004

  #TFLAG
  write x base_rt0 R_CONTROL 0x8008
```

```
  call rx_message 0 8 1 0x0000 1
  set sw $RETVALUE;
  compare sw 0x0001

  #TESTORUN
  if TESTTXTOUTEN                ! If this is disabled then we cant test
    call use_busA
    write x base_rt0 R_CONTROL 0x8010
    call rx_message 0 8 32 0x0000 1
    call tx_message_nochk 0 8 30 0x0000 1
    write x base_rt0 R_CONTROL 0x8000
    call use_busB
    call get_bit 0
    compare $RETVALUE 0xA989     # Expected Return BIT Value
    call use_busA
    call get_bit 0
    compare $RETVALUE 0x2189     # Expected Return BIT Value now
  endif

  #TESTORUN
  ifnot TESTTXTOUTEN                 ! tests will not overflow!
    call use_busA
    write x base_rt0 R_CONTROL 0x8010
    call rx_message 0 8 32 0x0000 1
    call tx_message_nochk 0 8 30 0x0000 1
    write x base_rt0 R_CONTROL 0x8000
    call use_busB
    call get_bit 0
    compare $RETVALUE 0x8009     # Expected Return BIT Value
    call use_busA
    call get_bit 0
    compare $RETVALUE 0x0009     # Expected Return BIT Value now
  endif


  #CLRERR - since bits are set
  write x base_rt0 R_INTERRUPT 0x0400
  wait 4
  write x base_rt0 R_INTERRUPT 0x0000
  call get_bit 0
  compare $RETVALUE 0x0009     # Expected Return BIT Value now
  call get_bit 0

  #BUSY bit
  readmask x base_rt0 R_CONTROL 0x0000 0x0080     # should be non busy
  call get_bit_nowait 0                           # Should cause RT to busy
  waitus 30                                       # after CW received
  readmask x base_rt0 R_CONTROL 0x0080 0x0080     # should be busy now
  pollbit  x base_rt0 R_CONTROL 7 0               # wait for bit to zero
  readmask x base_rt0 R_CONTROL 0x0000 0x0080     # should be non busy now
  waitus 60                                       # Allow BC to recover

  if LOCKRT
    #RTADDR Bits RT0 has LOCK generic set
   write    x base_rt0 R_CONTROL 0x0908        # Try and change to RT9 plus terminal flag
    readcheck x base_rt0 R_CONTROL 0xA008          # Should fail to set, bit 15 always
set + parity
    call sync_nodata 0                                          # Returns SW
    compare $RETVALUE 0x0001                      # Should Return SW from RT 0
    call sync_nodata 9                                          # Returns SW
    compare $RETVALUE 0xFFFF                      # No SW as no RT9

    iosetbit 0
                                                  # make sure RT legalizes CW
    #RTADDR Bits RT1 has LOCK  set
```

```
    write     x base_rt1 R_CONTROL 0x8800          # Try and change to RT8 with lock on
    readcheck x base_rt1 R_CONTROL 0x8100          # Should fail to set, bit 15 always
set + parity
    call sync_nodata 1                                          # Returns SW
    compare $RETVALUE 0x0800                     # Should Return SW from RT 1
    call sync_nodata 8                                        # Returns SW
    compare $RETVALUE 0xFFFF                     # No SW as no RT8
  endif

  ifnot LOCKRT
    #RTADDR Bits RT1 has LOCK not set
    write     x base_rt1 R_CONTROL 0x0800          # Try and change to RT8 with lock on
    readcheck x base_rt1 R_CONTROL 0x0800          # Should now be set
    call sync_nodata 1                             # Returns SW
    compare $RETVALUE 0xFFFF                       # Should not Return SW from RT 1
    call sync_nodata 8                             # Returns SW
    compare $RETVALUE 0x4000                       # RT8 returns status

    #RTADDR Bits RT1 has LOCK not set
    write     x base_rt1 R_CONTROL 0x1000          # Try and change to RT16
    readcheck x base_rt1 R_CONTROL 0x1000          # Should now be set
    call sync_nodata 16                            # Returns SW
    compare $RETVALUE 0x8000                       # RT16 returns status

    #RTADDR Bits RT1 has LOCK not set
    write     x base_rt1 R_CONTROL 0x3100          # Try and change to RT17
    readcheck x base_rt1 R_CONTROL 0x3100          # Should now be set
    call sync_nodata 17                            # Returns SW
    compare $RETVALUE 0x8800                       # RT16 returns status

    #RTADDR Bits RT1 has LOCK not set
    write     x base_rt1 R_CONTROL 0x3200          # Try and change to RT18
    readcheck x base_rt1 R_CONTROL 0x3200          # Should now be set
    call sync_nodata 18                            # Returns SW
    compare $RETVALUE 0x9000                       # RT16 returns status

    #RTADDR Bits RT1 has LOCK not set
    write     x base_rt1 R_CONTROL 0x3400          # Try and change to RT20
    readcheck x base_rt1 R_CONTROL 0x3400          # Should now be set
    call sync_nodata 20                            # Returns SW
    compare $RETVALUE 0xA000                       # RT16 returns status

    #RTADDR Bits RT1 has LOCK not set
    write     x base_rt1 R_CONTROL 0x3800          # Try and change to RT24
    readcheck x base_rt1 R_CONTROL 0x3800          # Should now be set
    call sync_nodata 24                            # Returns SW
    compare $RETVALUE 0xC000                       # RT16 returns status

    #RTADDR Bits RT1 has LOCK not set
   write    x base_rt1 R_CONTROL 0x1800        # Try and change to RT24 incorrect parity
    wait 10                                   # Allow for setting to cross clock domains
    readcheck x base_rt1 R_CONTROL 0x5800          # Incorrect Parity set

    write     x base_rt1 R_CONTROL 0x8000          # Back to RT1 external set
  endif

return

#############################################################################
# Test Interrupt Register

procedure test_interrupt

  iosetbit 0                                       # make sure RT1 legalized
```

```
  write     x base_rt1 R_INTERRUPT 0xFFFF          # Clear Interrupt Register
  write     x base_rt1 R_INTERRUPT 0x0000          # make sure enables clear
  readmask  x base_rt1 R_INTERRUPT 0x0000 0x0780   # Should be clear

  call rx_message 1 1 1 0x5060 3                   # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x00C1 0x07FF   # Interrupt Received
  iotstbit 1 0                                     # No external interrupt
  write     x base_rt1 R_INTERRUPT 0x0080          # Clear Interrupt Register
  readmask  x base_rt1 R_INTERRUPT 0x0041 0x07FF   # Interrupt Cleared

  call rx_message 1 2 1 0x5060 3                   # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x00C2 0x07FF   # Interrupt Received
  iotstbit 1 0                                     # No external interrupt
  write     x base_rt1 R_INTERRUPT 0x0080          # Clear Interrupt Register
  readmask  x base_rt1 R_INTERRUPT 0x0042 0x07FF   # Interrupt Cleared

  call rx_message 1 4 1 0x5060 3                   # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x00C4 0x07FF   # Interrupt Received
  iotstbit 1 0                                     # No external interrupt
  write     x base_rt1 R_INTERRUPT 0x0080          # Clear Interrupt Register
  readmask  x base_rt1 R_INTERRUPT 0x0044 0x07FF   # Interrupt Cleared

  call rx_message 1 8 1 0x5060 3                   # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x00C8 0x07FF   # Interrupt Received
  iotstbit 1 0                                     # No external interrupt
  write     x base_rt1 R_INTERRUPT 0x0080          # Clear Interrupt Register
  readmask  x base_rt1 R_INTERRUPT 0x0048 0x07FF   # Interrupt Cleared

  call rx_message 1 16 1 0x5060 3                  # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x00D0 0x07FF   # Interrupt Received
  iotstbit 1 0                                     # No external interrupt
  write     x base_rt1 R_INTERRUPT 0x0080          # Clear Interrupt Register
  readmask  x base_rt1 R_INTERRUPT 0x0050 0x07FF   # Interrupt Cleared

  call tx_message 1 16 1 0x5060 3                  # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x00E0 0x07FF   # Interrupt Received
  iotstbit 1 0                                     # No external interrupt
  write     x base_rt1 R_INTERRUPT 0x0080          # Clear Interrupt Register
  readmask  x base_rt1 R_INTERRUPT 0x0070 0x07FF   # Interrupt Cleared

  write     x base_rt1 R_INTERRUPT 0x0100          # Enable External Interrupt
  call rx_message 1 1 1 0x5060 3                   # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x01C1 0x07FF   # Interrupt Received
  iotstbit 1 1                                     # external interrupt
  write     x base_rt1 R_INTERRUPT 0x0180          # Clear Interrupt Register
  flush  4
  iotstbit 1 0                                     # No external interrupt
  readmask  x base_rt1 R_INTERRUPT 0x0141 0x07FF   # Interrupt Cleared

  call rx_message 1 1 1 0x5060 3                   # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x01C1 0x07FF   # Interrupt Received
  iotstbit 1 1                                     # external interrupt
  call tx_message 1 2 1 0x5060 3                   # rt sa len dstart dinc
  readmask  x base_rt1 R_INTERRUPT 0x03E2 0x07FF   # Interrupt overrun
  write     x base_rt1 R_INTERRUPT 0x0200          # Clear Interrupt overrun
  readmask  x base_rt1 R_INTERRUPT 0x02E2 0x07FF   # Interrupt Cleared
  write     x base_rt1 R_INTERRUPT 0x0080          # Clear Interrupt Register
  readmask  x base_rt1 R_INTERRUPT 0x0062 0x07FF   # Interrupt Cleared

return


##########################################################################

procedure testmemory RTbase addr size
```

```
 if readback
   memtest RTbase 0x0100 256 2 500 0x23494579
 endif

return

##############################################################################

procedure test_vword base rt
int exp_vword
int got_vword
int ok

   set exp_vword rt * 256
   write w base R_VWORD exp_vword
   call get_vw rt
   compare $RETVALUE exp_vword
return

##############################################################################

procedure test_rxtx;
 int RT;
 int SA;
 int WC;
 int base;

    wait 6
    header "Testing RX TX"
    # Set up RTs
    loop RT 0 (NRTS-1)
      set base RT * 0x01000000 + 0x10000000
      write h base R_CONTROL   0x8000
      write h base R_INTERRUPT 0x0000
    endloop

    # Fetch the vector Word from each RT
    loop RT 0 (NRTS-1)
      set base RT * 0x01000000 + 0x10000000
      call test_vword base RT
    endloop

    # Test RX/TX varying RT
    loop RT 0 (NRTS-1)
      call rx_message RT 1 10 0x5060 RT
      call tx_message RT 1  6 0x5060 RT
    endloop

    # Test RX/TX varying SA
    loop SA 1 22
      call rx_message 1 SA 10 0x5060 SA
      call tx_message 1 SA  6 0x5060 SA
    endloop

    # Test RX/TX varying WC
    loop WC 1 32
      call rx_message 1 1 WC 0x5060 WC
      call tx_message 1 1 WC 0x5060 WC
    endloop

return
```

```
#########################################################################
# Set up legalization registers

-- 0: Internal to RT core
-- 1: External Input
-- 2: APB Registers
-- 3: APB RAM block

-- reset legalization table  0 enables message
table LEGALISATION  0x0000 0x0000 0x0000 0x0000 \
                    0x0000 0x0000 0xffff 0xffff \
                    0xffff 0xfffd 0xfe01 0xfff2 \
                    0xffff 0xfffd 0xfe05 0xffff



procedure setup_legal_mode baseaddr
 int cmp;

  set cmp LEGMODE == 0
  if cmp
    call  setup_legal_mode0 baseaddr
  endif
  set cmp LEGMODE == 1
  if cmp
    call  setup_legal_mode1 baseaddr
  endif
  set cmp LEGMODE == 2
  if cmp
    call  setup_legal_mode2 baseaddr
  endif
  set cmp LEGMODE == 3
  if cmp
    call  setup_legal_mode3 baseaddr
  endif
return

procedure setup_legal_mode0 baseaddr
 # no need to do anything
return

procedure setup_legal_mode1 baseaddr
 iosetbit 0   -- CMDOK is driven by GPIO bit 0
return


procedure setup_legal_mode2 baseaddr

 readtable x baseaddr 0x1fc0 LEGALISATION 16      -- check reset values
 fill       x  baseaddr 0x1fc0 16 0x0003 0x1234   -- Verify writeable and readable
 fillcheck  x  baseaddr 0x1fc0 16 0x0003 0x1234   -- fill with pattern

 writetable x baseaddr 0x1fc0 LEGALISATION 16     -- write back reset values
 readtable  x baseaddr 0x1fc0 LEGALISATION 16

return

procedure setup_legal_mode3 baseaddr

 writetable x baseaddr 0x1fc0 LEGALISATION 16
 if readback
   readtable  x baseaddr 0x1fc0 LEGALISATION 16
 endif
return
```

```
##############################################################################
# test Legality logic

# Mode 0


procedure test_legality_mode
 int cmp;

  set cmp LEGMODE == 0
  if cmp
    call  test_legality_mode0
  endif
  set cmp LEGMODE == 1
  if cmp
    call  test_legality_mode1
  endif
  set cmp LEGMODE == 2
  if cmp
    call  test_legality_mode2
  endif
  set cmp LEGMODE == 3
  if cmp
    call  test_legality_mode3
  endif

return


procedure test_legality_mode0
int sw;

 write w base_RT0 R_CONTROL 0x8000       -- Reset State
 ioclrbit 0                              -- This should have no effect
 call rx_message_nochk 0 8 10 0x0000 1
 set sw $RETVALUE
 compare sw 0x0000                       -- should be Legal message
 iosetbit 0

return


procedure test_legality_mode1
int sw;

 iosetbit 0                              -- CMDOK is driven by GPIO bit 0
 call rx_message 1 8 10 0x0000 1
 set sw $RETVALUE
 compare sw 0x0800

 ioclrbit 0                              -- CMDOK is driven by GPIO bit 0
 call rx_message_nochk 1 8 10 0x0000 1
 set sw $RETVALUE
 compare sw 0x0c00                       -- Message Error
 iosetbit 0
 iomask 0x010A0000 0x0FFF0000            -- Check the CMDVAL value as well

return


procedure test_legality_mode2
int sw;

 write w base_RT2 R_LEGREG0 0xFEFF       -- Enable SA 8 Receive
 call rx_message 2 8 10 0x0000 1
```

```
 set sw $RETVALUE
 compare sw 0x1000

 write w base_RT2 R_LEGREG0 0x0100       -- Disable SA 8 Receive
 call rx_message_nochk 2 8 10 0x0000 1
 set sw $RETVALUE
 compare sw 0x1400                       -- Message Error
 write w base_RT2 R_LEGREG0 0x0000       -- Reenable

return

procedure test_legality_mode3
int sw;

 write w base_RT3 R_LEGREG0 0xFEFF       -- Enable SA 8 Receive
 call rx_message 3 8 10 0x0000 1
 set sw $RETVALUE
 compare sw 0x1800

 write w base_RT3 R_LEGREG0 0x0100       -- Disable SA 8 Receive
 call rx_message_nochk 3 8 10 0x0000 1
 set sw $RETVALUE
 compare sw 0x1C00                       -- Message Error

 write w base_RT3 R_LEGREG0 0x0000       -- Reenable

return


##########################################################################
# THESE ARE THE LOW LEVEL DRIVERS TO THE 1553B BUS CONTROLLER TEST MODULE
--------------------------------------------------------------------------
-- Extension Bus Register Set for 1553B BC

-- Addr 0 : Bit 0 = Start
--          Bit 1 = Busy
-- Addr 1 : Block Pointer  (BP)
--
-- Block Address Mapping - pointed to by BP
-- BP+0 : Blk control
--        bit  0: RTRT
--        bit  1: Do next message
--        bit 16: okay
-- BP+1 : CW1
-- BP+2 : CW2
-- BP+3 : SW1
-- BP+4 : SW2
-- BP+5 : DataPtr or Data
-- BP+6 : Num DW received
-- BP+7 : Pointer to next message

--------------------------------------------------------------------------


procedure use_busA
 set CWORD 0x0001
return

procedure use_busB
 set CWORD 0x0021
return

procedure sync_nodata RT
int sw;
int cw;
```

```
 print "Sync No Data"
 set cw RT << 11 + 0x0401

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16
 extwrite 1 8
 extwrite 0 1
 waitus 10
 extwait
 extread 11 sw

return sw;

procedure get_lastsw RT
int sw;
int cw;

 print "Get Last SW"
 set cw RT << 11 + 0x0402

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16
 extwrite 1 8
 extwrite 0 1
 waitus 10
 extwait
 extread 11 sw

return sw;

procedure get_vw RT
int vw;
int cw;

 print "Transmit Vector Word"
 set cw RT << 11 + 0x0410

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16
 extwrite 1 8
 extwrite 0 1
 waitus 10
 extwait
 extread 13 vw

return vw;

procedure get_bit RT
int bit;
int cw;

 print "Transmit Bit"
 set cw RT << 11 + 0x0413

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 0
 extwrite 1 8
 extwrite 0 1
 extwait
 extread 13 bit
```

```
return bit;


procedure get_bit_nowait RT
int vw;
int cw;

 print "Transmit Bit"
 set cw RT << 11 + 0x0413

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16
 extwrite 1 8
 extwrite 0 1
return;

procedure rx_message rt sa len dstart dinc
int cw;
int sw;
int len5;
int eaddr;
int edata;
int i;
int base;
int addr;

 print "Receive Message"
 set len5 len and 0x1F
 set cw RT << 1 + 0 << 5
 set cw cw + SA << 5
 set cw cw + len5

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16

 set eaddr 16       -- write pattern to external device
 set edata dstart
 loop i 1 len 1
   extwrite eaddr edata
   set eaddr eaddr + 1
   set edata edata + dinc
 endloop

 extwrite 1 8
 extwrite 0 3
 waitus 10
 extwait
 extread 11 sw

 # Check the data
 set base RT * 0x01000000 + 0x10000000
 set addr SA * 0x80
 fillcheck x base addr len dstart dinc

return sw;


procedure rx_message_nochk rt sa len dstart dinc
int cw;
int sw;
int len5;
int eaddr;
```

```
int edata;
int i;
int base;
int addr;

 print "Receive Message"
 set len5 len and 0x1F
 set cw RT << 1 + 0 << 5
 set cw cw + SA << 5
 set cw cw + len5

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16

 set eaddr 16       -- write pattern to external device
 set edata dstart
 loop i 1 len 1
   extwrite eaddr edata
   set eaddr eaddr + 1
   set edata edata + dinc
 endloop

 extwrite 1 8
 extwrite 0 3
 waitus 10
 extwait
 extread 11 sw

return sw;

procedure tx_message rt sa len dstart dinc
int cw;
int sw;
int len5;
int eaddr;
int edata;
int i;
int base;
int addr;

 print "Transmit Message"

 # Create the data
 set base RT * 0x01000000 + 0x10000000
 set addr SA * 0x80 + 0x1000
 fill x base addr len dstart dinc

 set len5 len and 0x1F
 set cw RT << 1 + 1 << 5
 set cw cw + SA << 5
 set cw cw + len5

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16

 extwrite 1 8
 extwrite 0 3
 waitus 10
 extwait
 extread 11 sw

 # Check the data
 set eaddr 16       -- read and check pattern from external device
```

```
 set edata dstart
 loop i 1 len 1
   extcheck eaddr edata
   set eaddr eaddr + 1
   set edata edata + dinc
 endloop

return sw;

procedure tx_message_nochk rt sa len dstart dinc
int cw;
int sw;
int len5;
int eaddr;
int edata;
int i;
int base;
int addr;

 print "Transmit Message"

 # Create the data
 set base RT * 0x01000000 + 0x10000000
 set addr SA * 0x80 + 0x1000
 fill x base addr len dstart dinc

 set len5 len and 0x1F
 set cw RT << 1 + 1 << 5
 set cw cw + SA << 5
 set cw cw + len5

 extwrite 8 CWORD
 extwrite 9 CW
 extwrite 13 16

 extwrite 1 8
 extwrite 0 3
 waitus 10
 extwait
 extread 11 sw

return sw;
```

B – Known Issues

- When using the set command:
  - Spaces must be left either side of all the operators, including the operators in the set command
  - Bracketed expressions may only contain expressions that can be evaluated at compile time that is, they must not contain any variables declared using the int command
  - The set command (excluding bracketed expressions) is evaluated left to right.
- It is possible to replace integer values with basic equations in brackets. The equation is evaluated at compile time.

  ```
  Write w ahbslave 0x000 (6+8)
  ```

  There is no need for spaces here, the supported C operators are +,-,*,/,>>,<<,>,<,--,!-,>=,<=. The equations can use brackets

  ```
  Constant xxxx (512<<8)
  Write w ahbslave 0x000 (6+8*(4+XXXX))
  ```

  Setting the -eval switch on the compiler reports how it has interpreted the values
- When using # as a comment character it must be followed by a space character.

  The maximum number of global variables that may be used is 8192. Also each subroutine may declare an additional 8191 variables (including the subroutine parameters). An array with 100 elements counts as 100 variables.
- When using array no spaces are allowed in the [ ] index value

C – List of Changes

The following table lists critical changes that were made in the current version of the chapter.

Date	Changes	Page
Revision 1 (August 2012)	Added reference in "BFM Commands - Master Cores" (SAR 34556)	19
	Modified Table 6-6 (SAR 34556)	23

*Note: *The part number is located on the last page of the document. The digits following the slash indicate the month and year of publication.*

D – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call 800.262.1060
From the rest of the world, call 650.318.4460
Fax, from anywhere in the world, 408.643.6913

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Customer Support website (www.microsemi.com/soc/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the website.

Website

You can browse a variety of technical and non-technical information on the SoC home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Index

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

50200141-1/ 8.12