SmartFusion Bus Functional Model

User's Guide

Microsemi

Table of Contents

Introduction

This document describes the Bus Functional Model (BFM) solution for the SmartFusion® device. BFM models microcontroller subsystem (MSS) bus cycles without modelling other Cortex-M3 processors and other MSS peripherals behavior and replaces the Cortex-M3 processor core when simulating design. This BFM simulation enables the SmartFusion device customers to verify and simulate communication between MSS and fabric logic.

Note: The complete verification of MSS, fabric and their interfaces and protocol compliance is outside the scope of the BFM solution.

This document describes the following:

- Overview of SmartFusion BFM
- BFM usage flow
- SmartFusion BFM commands
- BFM examples

1 – Overview of SmartFusion BFM

The BFM acts as a pin-for-pin replacement for the MSS in the project system. It initiates cycle-accurate bus transactions on the MSS Advanced Microcontroller Bus Architecture (AMBA®) High-Performance Bus (AHB) matrix and thus allows customers to verify and simulate communication between SmartFusion MSS and fabric logic.

To minimize simulation time, certain peripherals in the SmartFusion MSS do not have full behavioral models. Instead they are replaced with memory models that output a message indicating when the memory locations inside the peripheral have been accessed. This means that the peripheral signals do not toggle based on any writes to registers, or react to any signal inputs on the protocol pins. The peripherals that fall into this group include:

- Universal asynchronous receiver/transmitter (UART)
- Serial peripheral interface bus (SPI)
- Integrated circuits (I^2C)
- Media access control (MAC)
- Peripheral direct memory access (PDMA)
- WatchDog Timer
- Real-time counter (RTC)

The peripherals that have full behavioral models include:

- Clock management
- Embedded nonvolatile memory (eNVM)
- External memory controller
- Analog compute engine (ACE)
- General purpose input/output (GPIO)
- Fabric interface controller
- Embedded FlashROM (eFROM)
- AMBA AHB bus matrix

The eNVM simulation model is not initialized with data storage or initialization client data. Similarly, eFROM simulation model is not initialized with region configuration data. You can write and read to both peripherals as memory elements.

2 – BFM Usage Flow

Figure 2-1 shows a typical BFM simulation. The BFM is scripted through a text file containing a list of bus cycles. 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® System-on-Chip (SoC) is configured to automatically compile the BFM script when simulation using Modlesim™ is invoked.

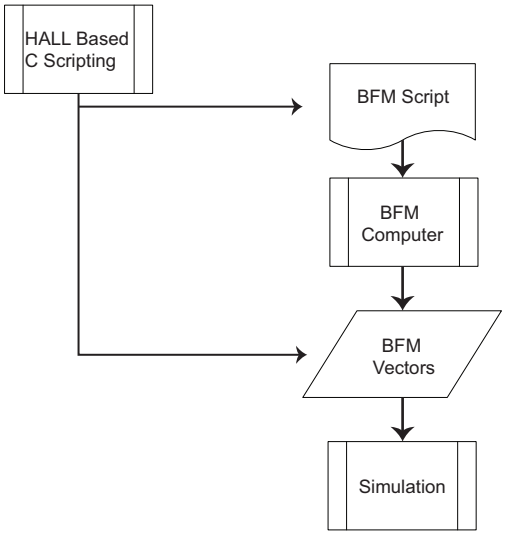

***Figure 2-1* • BFM Tool Flow**

The BFM files are generated in Libero when the MSS is created. The SmartDesign generates the following files when the MSS is generated:

1. Test.bfm: This contains the BFM commands to initialize the simulation model. The BFM commands in this file are generated based upon your MSS configuration. This file is analogous to the system boot code, as it initializes the MSS and calls your user application. Do not modify this file.
2. User.bfm: You can customize this file to emulate Cortex-M3 processor transactions in your system. This contains an include directive to the subsystem.bfm that needs to be uncommented if you have any fabric peripherals that you wish to simulate. The memory map of the fabric peripherals is specified inside subsystem.bfm, you can refer to those defines inside this BFM file. This file is analogous to your user application code.
3. Subysystem.bfm: Contains the fabric memory map. Do not modify this file.

These files are automatically passed to ModelSim during simulation in the Libero SoC. The default user.bfm does not have any script for the user system. So, you need to modify the user.bfm script and add your own script to test the system before running the ModelSim.

The user.bfm script can be accessed through the File Hierarchy, below your MSS component in the Simulation Files node (as shown in Figure 2-2).

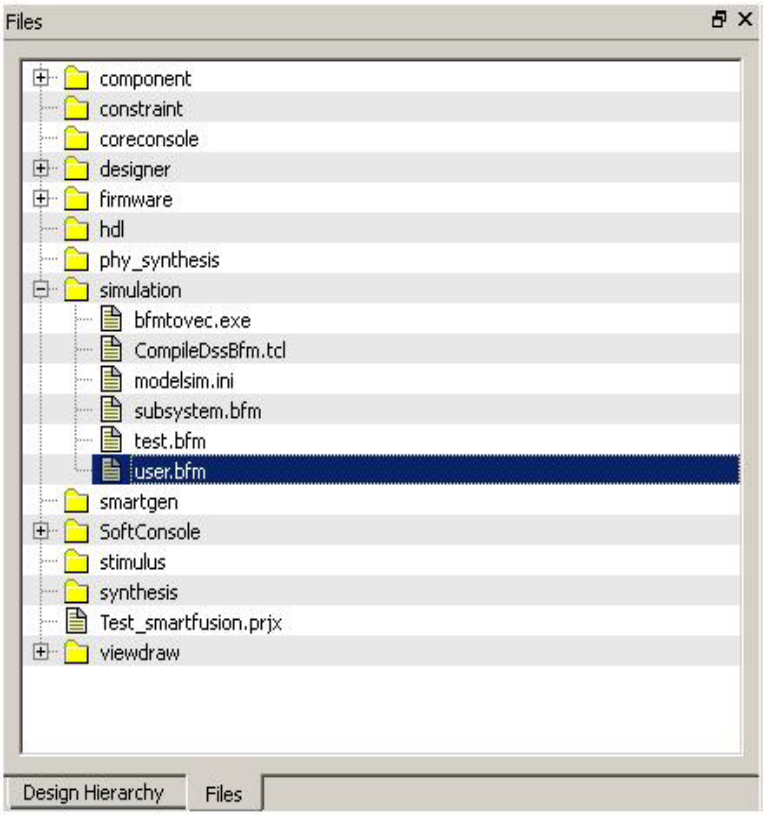

***Figure 2-2* • BFM Tool Flow**

The BFM source files are automatically recompiled every time the simulation is invoked from the Libero SoC by the bfmtovec.exe executable, if running on a Windows® platform, or by the bfmtovec.lin executable, if running on a Linux® platform.

Refer to www.microsemi.com/ipdocs/CoreAMBA_BFM_UG.pdf for more information.

3 – SmartFusion BFM Commands

The following commands are supported by the SmartFusion BFM. The Clocks column indicates the number of clock cycles an instruction can accommodate; V indicates that it is variable based on the instruction parameters or AHB/Advanced Peripheral Bus (APB) response times.

Basic Read and Write Commands

The commands mentioned in Table 3-1 provide basic read and write functions.

***Table 3-1* • Basic Read and Write Functions**

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*	Sets the base address of the associated with the resource.	V
read *width resource address*	Performs a read cycle and echoes the read data to the simulation log.	V
readcheck *width resource address data*	Performs a read cycle and checks the read data.	V
poll *width resource address data*	Performs a read cycle until the read data matches the specified value. In this case a match is read_data & data=data.	V
wait *cycles*	Waits for the specified number of clock cycles. The allowed value for clock cycle is 1 to 510. Use multiple wait statements for implementing a wait statement with a value greater the upper limit, that is, divide the wait cycles and write multiple wait statements	V

Enhanced Read and Write Commands

The commands mentioned in Table 3-2 provide enhanced read and write functions.

***Table 3-2* • Enhanced Read and Write Functions**

Enhanced Read and Write Commands	Description	Clocks
readstore *width resource address variable*	Performs a read cycle and stores the data in the specified variable.	V
Readmask *width resource address data mask*	Performs a read cycle and checks the read data. The data is masked as follows: read_data & mask=data & mask.	V
pollmask *width resource address data mask*	Performs 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*	Performs a read cycle until the specified bit matches the specified value.	V

***Table 3-2* • Enhanced Read and Write Functions (continued)**

Enhanced Read and Write Commands	Description	Clocks
memtest *resource addr size align cycles seed*	Performs 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]* 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]* 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
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*	Performs 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, 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 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

These commands allow AMBA burst instructions to be created. They also simplify filling of memory and creation of data tables as shown in Table 3-3.

***Table 3-3* • Burst Support**

Burst Support	Description	Clocks
writemult *width resource address data1 data2 data3 … data4*	Writes 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
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. During burst transfers the address increments based on the required transfer width. Thus, if a byte transfer is requested the address increments by 1.
3. A table may only contain 255 values.
4. Arrays are declared using int blah[100] instruction. In the read and writearray 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

Flow Control

***Table 3-4* • Flow Control**

Flow Control	Description	Clocks
label *labelid*	Sets 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*	Sets a label in the BFM script for a call and name its parameters.	0
jump *labelid*	Jumps to the specified label within the current procedure.	0
jumpz *labelid data*	Jumps if the specified data value is zero.	0
jumpnz *labelid data*	Jumps 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	Returns from the routine.	0
return data	Returns from the routine returning the data value or variable. Return value is accessed using the $RETVALUE variable.	0
loop *para1 start end inc*	Repeats 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
endloop	End of loop.	0
if *variable*	The instructions between if and the following endif are performed if *variable* is non zero. If/endif can be nested.	0
endif	End of If.	0
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*	Pauses the BFM script operation for N clock cycles.	V
waitns *N*	Pauses 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.*	V

***Table 3-4* • Flow Control (continued)**

Flow Control	Description	Clocks
waitus *N*	Pauses 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.*	V
flush N	Wait sfor any pending read or write cycles to complete, and then wait for N additional clock cycles.	V
quit	Terminates the BFM and assert the FINISHED output.	1

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, mentioned in Table 3-5, 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 3-5* • Variables**

Parameters	Description	Clocks
int *para1 … paran*	Declare variable.	0
int *array*[*N*]	Declares 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
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

The supported operators for the set command are shown in Table 3-6; these are evaluated during run time by the BFM.

***Table 3-6* • The Supported Operators for the Set Command**

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
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)

BFM Control

The commands mentioned in Table 3-7 are extended control functions for corner testing etc.

***Table 3-7* • BFM Control**

BFM Control	Description	Clocks
version	Prints versioning information for the BFM in the simulation and log file, allows later verification.	0
timeout *N*	Sets an internal timeout value in clock cycles that triggers if the BFM stalls. Default timeout is 512 clocks.	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 3-7* • BFM Control (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
lock N	Lock 1 asserts LOCK on the next AHB cycle and stay on until a LOCK 0 command is executed, typical operation is Lock 1 Read w ahbslave Write w ahbslave Lock 0	0
Echo *D0 D1 D2 .. D7*	Simply list 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*	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 takes 17 cycles. It can also be used to check the how long a poll, waintint, waitirq, waitfiq, iowait or extwait instruction took to complete. An error is recorded if the check fails.	1
starttimer	Starts an internal timer (clock cycles).	0
checktimer *min max*	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. An error is recorded if the check fails.	1
setfail	Sets the BFM FAILED output.	1
Setrand para	Sets the internal random number seed.	0

BFM Compiler Directives

Table 3-8 shows the instructions used by the compiler rather than used in the vector files.

***Table 3-8* • BFM Compiler Directives**

BFM Compiler Directives	Description	Clocks
include *filename* include "*filename*"	Includes another BFM file, include files may also contain include files. The filename should be double quoted when filenames are case sensitive C header files may also be included that have been generated by the IP core packaging system.	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
;	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

Note: MEMMAP, SET and CONSTANT all declare symbol values. Once declared, the symbol can be used for any integer like parameter. The same SYMBOL name cannot be used with memmap, set or constant commands.

Parameter Formats

Table 3-9 describes the parameter formats used in the above command description.

***Table 3-9* • Parameter Formats**

Parameters	Type/Values	Description
Integer		Integer value using the following syntax, **0x1245ABCD**: hexadecimal **0d12343456**: decimal **0b10101010**1: 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 additional parenthesis may be used. **Mnopq[100]:** An array element, the index may be variable, but not another array, that is, only single dimensional arrays are supported.
Symbol		ASCII string staring with a letter.

***Table 3-9* • Parameter Formats (continued)**

Parameters	Type/Values	Description
Resource	Integer	Integer specifying the base address of a resource. Microsemi recommends that the base address is declared using the memap command and symbol is used for the resource parameter.
Address	Integer	Specifies the address offset from its base address.
Width	b, h, w	Specifies whether byte, half word, or word access.
Cycles	Integer	-
Mask	Integer	-
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 " ".

Note: All commands, labels and parameters are case insensitive.

$ Variables

The BFM supports some special integer values that may be specified rather than immediate or variables. The supported $ variables are mentioned in Table 3-10.

***Table 3-10* • $ Variables**

$ Variable	Description
$RETVALUE	Is the value from the last executed return instruction.
$TIME	Current simulation time in ns.
$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.
$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.
$RAND*n*	As above but the random number is limited to *n* bits.
$RANDSET*n*	Same as $RANDRESET, but the random number is limited to *n* bits
$RANDRESET*n*	Same as $RANDRESET, 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. In CoreConsole Soft IP environments the actual core generic values as set by the CoreConsole GUI can be passed to the script if the values are set in the coreparameters.v/vhd files are used to set the ARGVALUE parameters on the BFM in the testbench.

Note: The random function is simple CRC implementation.

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.

4 – BFM Example

The following section describes an example BFM script that is used for testing a custom advanced peripheral bus (APB) core from MSS.

Writing and Verifying Fabric

Figure 4-1 shows a simple design with a soft register block (16x8 bit) connected to the fabric. After creating the design using SmartDesign, click “Generate” to create RTL and all other required files. During the generation, the subsystem.bfm is automatically generated by the system and contains the memory map of the custom APB core. The user.bfm is also automatically generated, but does not have any BFM command. You can add BFM command shown in this example and run simulation.

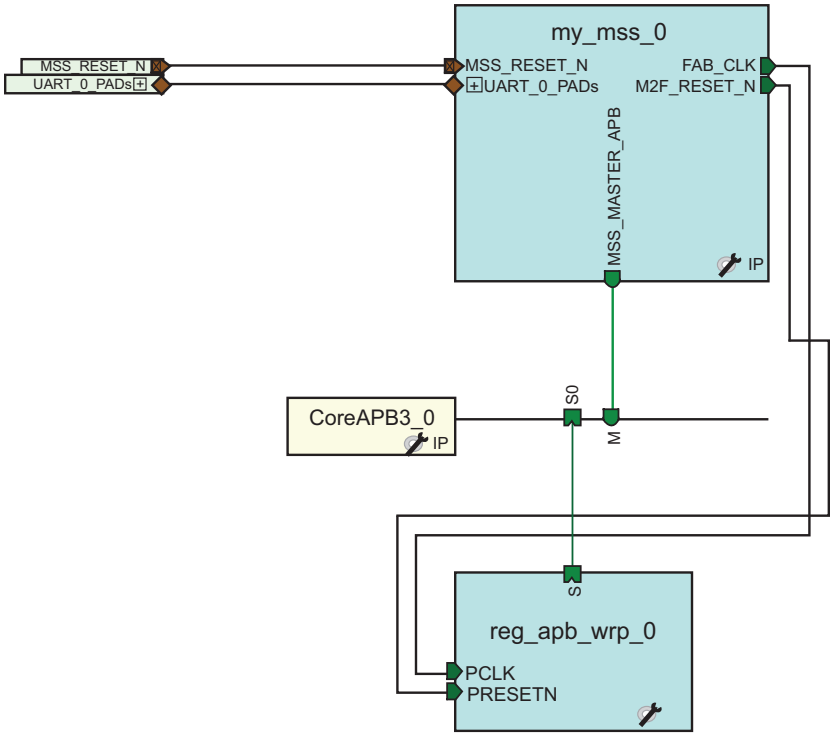

Figure 4-1 • **ISmartFusion design with a custom APB block connected to MSS**

```
subsystem.bfm:
#============================================================
# Created by Microsemi SmartDesign Tue Jan 03 15:04:43 2012
#
# Syntax:
# -------
#
# memmap resource_name base_address;
#
# write width resource_name byte_offset data;
# read width resource_name byte_offset;
# readcheck width resource_name byte_offset data;
#
#============================================================
```

```
#-----------------------------------------------------------
# Memory Map
# Define name and base address of each resource.
#-----------------------------------------------------------
memmap reg_apb_wrp_0 0x40050000;
```

The subsystem.bfm file is automatically generated and you do not need to modify it.

user.bfm:

```
#===========================================================
# Enter your BFM commands in this file.
#
# Syntax:
# -------
#
# memmap resource_name base_address;
#
# write width resource_name byte_offset data;
# read width resource_name byte_offset;
# readcheck width resource_name byte_offset data;
#
#===========================================================
procedure user_main;
# uncomment the following include if you have soft peripherals in the fabric
# that you want to simulate. The subsystem.bfm file contains the memory map
# of the soft peripherals.
include "subsystem.bfm"
# add your BFM commands below:

int i;
int j;
wait 5;

print "*****************************************************************";
print "Testing Custom APB slave block";
print "*****************************************************************";
write b reg_apb_wrp_0 0x00 0x01;
write b reg_apb_wrp_0 0x04 0x05;
write b reg_apb_wrp_0 0x08 0x09;
wait 5;
readcheck b reg_apb_wrp_0 0x00 0x01; # Expect value 01
readcheck b reg_apb_wrp_0 0x04 0x05; # Expect value 05
readcheck b reg_apb_wrp_0 0x08 0x09; # Expect value 09
wait 10;

header "Testing reg_apb_wrp_0 in loop mode";
loop j 0 16 4
loop i 0 16
write b reg_apb_wrp_0 j i;
readcheck b reg_apb_wrp_0 j i; # Expect value i
endloop
endloop

header " Current Time is:%0d", $TIME;
wait 100;

return
#===========================================================
# Created by Microsemi SmartDesign Tue Jan 03 15:04:43 2012
#
# Syntax:
# -------
#
# memmap resource_name base_address;
#
```

```
# write width resource_name byte_offset data;
# read width resource_name byte_offset;
# readcheck width resource_name byte_offset data;
#
#============================================================
#------------------------------------------------------------
# Memory Map
# Define name and base address of each resource.
#------------------------------------------------------------
memmap reg_apb_wrp_0 0x40050000;
```

The subsystem.bfm file is automatically generated and you do not need to modify it.

user.bfm:

```
#============================================================
# Enter your BFM commands in this file.
#
# Syntax:
# -------
#
# memmap resource_name base_address;
#
# write width resource_name byte_offset data;
# read width resource_name byte_offset;
# readcheck width resource_name byte_offset data;
#
#============================================================
procedure user_main;
# uncomment the following include if you have soft peripherals in the fabric
# that you want to simulate. The subsystem.bfm file contains the memory map
# of the soft peripherals.
include "subsystem.bfm"
# add your BFM commands below:

int i;
int j;
wait 5;

print "************************************************************";
print "Testing Custom APB slave block";
print "************************************************************";
write b reg_apb_wrp_0 0x00 0x01;
write b reg_apb_wrp_0 0x04 0x05;
write b reg_apb_wrp_0 0x08 0x09;
wait 5;
readcheck b reg_apb_wrp_0 0x00 0x01; # Expect value 01
readcheck b reg_apb_wrp_0 0x04 0x05; # Expect value 05
readcheck b reg_apb_wrp_0 0x08 0x09; # Expect value 09
wait 10;

header "Testing reg_apb_wrp_0 in loop mode";
loop j 0 16 4
loop i 0 16
write b reg_apb_wrp_0 j i;
readcheck b reg_apb_wrp_0 j i; # Expect value i
endloop
endloop

header " Current Time is:%0d", $TIME;
wait 100;

return
```

A – 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**.

© 2012 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

50200321.0/7.12