Building an APB3 Core for SmartFusion cSoC FPGAs

Table of Contents

Introduction . . . 1
Overview of APB, APB3, AHB, and AHB-Lite . . . 1
APB3 Slave in the AMBA System . . . 2
Implementing an APB3 Interface for a Custom Logic Block . . . 5
Appendix A: Creating a Subsystem Design with a Custom APB3 Slave . . . 10
Appendix B: Simulating the User Logic Block . . . 13
List of Changes . . . 14

Introduction

The advanced microcontroller bus architecture (AMBA®) specification defines an on-chip communications standard for designing high-performance embedded microcontrollers. Several distinct buses are defined within the AMBA specification, including advanced high-performance bus (AHB) and advanced peripheral bus (APB). The AMBA-AHB is used to connect high-performance modules. They are also used for low power and low speed peripherals.

The SmartFusion® customizable system-on-chip (cSoC) devices include a hard embedded microcontroller subsystem (MSS) with FPGA fabric and high-performance analog block. The MSS is composed of a 100 MHz ARM® Cortex™-M3 processor and integrated peripherals, which are interconnected via a multi-layer AHB bus matrix (ABM). The MSS can be connected to the FPGA fabric through a configurable fabric interface controller (FIC) that allows either an AHB to AHB or AHB to APB3 (also known as the APB v3) bridging function between the ABM and an AHB or APB3 bus implemented in the FPGA fabric. APB3 is much simpler than AHB and the user logic in the FPGA normally communicates with the MSS via APB3 register mapping. This document describes how to create an APB3 wrapper interface for the user's logic or IP and how to connect it to the MSS through the FIC.

Overview of APB, APB3, AHB, and AHB-Lite

The AMBA bus specification is an open standard introduced by ARM Ltd. and details a strategy for the interconnection and management of functional blocks in an embedded microcontrollers or system-on-chip (SoC). Several distinct buses are defined within the AMBA specification. The AMBA-AHB is for high-performance, high clock frequency system modules to support efficient connection of processors, on-chip memories, and off-chip external memory interfaces. It allows high performance, pipelined operation, multiple bus masters, burst transfers, and split transactions. AHB-Lite, defined in the AMBA 3 protocol (third generation of the AMBA specification), is a subset of the full AHB specification for use in designs where only a single bus master is used. AMBA-AHB is used to interface with any peripherals that are low bandwidth and do not require the high performance of a pipelined bus interface. This bus has an address and data phase similar to AHB, but a much reduced low complexity signal list; for example, no bursts. APB3, defined in the AMBA 3 protocol, allows extending an APB transfer and transferring failure information.

APB3 Slave in the AMBA System

Figure 1 shows a typical AMBA system. It shows the ARM processor connected to the AHB-Lite bus and to the APB3 bus via a bridge. There are several peripherals connected to the AHB-Lite and APB3 buses. The peripherals on the AHB-lite bus are in general more complex, high performance, high throughput devices requiring an important amount of the bus bandwidth.

In comparison, APB3 allows the connection of low throughput peripherals requiring lesser bandwidth. APB3 is much simpler than AHB-Lite and requires fewer gates.

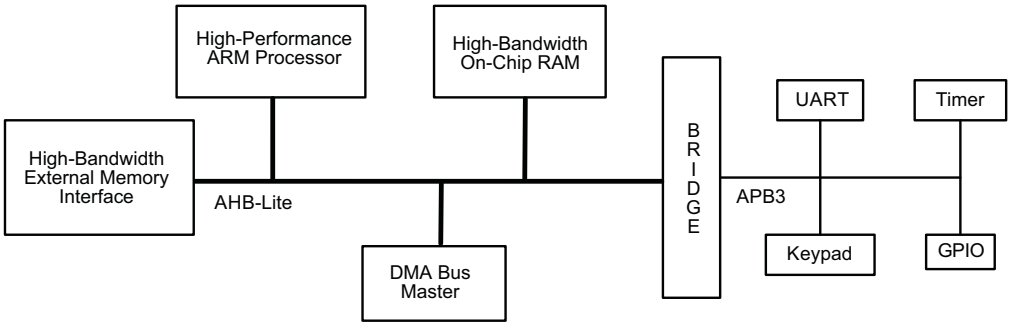

***Figure 1* • APB3 Slave in the AMBA System**

As stated earlier, the FIC in SmartFusion cSoC devices allows bridging between the MSS and either an AHB-Lite or an APB3 interface. In most SmartFusion cSoC based applications, the FPGA fabric is used for low bandwidth peripherals or blocks. You need to create an APB3 interface to the custom peripherals or blocks and then connect them to the MSS via the FIC bridge.

Figure 2 shows a custom logic block with APB3 interface connected to the MSS through CoreAPB3 in a SmartFusion cSoC device.

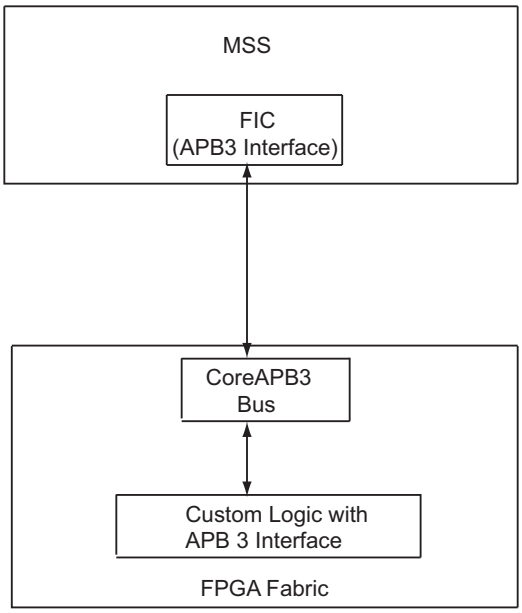

***Figure 2* • Custom Logic with APB3 Interface Connected to MSS in SmartFusion cSoC**

An APB3 interface is needed on custom low bandwidth peripherals in order to connect to an APB3 bus. The APB3 slave interface acts as a bridge between the APB3 bus and the peripheral device to which the bus is connected. It receives the APB3 bus signals and converts them to a form understood by the connected peripheral. The most common application of the APB3 interface is to read and write registers associated with the connected peripheral. The APB3 slave interface would be implemented to interface with the APB3 bus at one end and registers at the other end.

It receives the control signals from the APB3 bus and uses them to generate the read and write enable signals for the registers. The write data and address signals of the APB3 bus are forwarded by the APB3 interface to the registers during write operations. During read operations, the data read from the registers is transmitted onto the APB3 bus.

The following sections describe APB3 in detail and provide examples of how to create an APB3 slave wrapper interface on the user custom logic or IP.

APB3 Bus Operation

The AMBA specification defines an on-chip communications standard for designing high-performance embedded microcontrollers. AMBA-Lite APB, also known as APB v3 or APB3, is used to interface to any peripherals that are low bandwidth and do not require the high-performance of a pipelined bus interface.

Table 1 gives the APB3 signals.

Table 1 • **APB3 Signals**

Signal Name	Description
PCLK	Clock. The rising edge of PCLK times all transfers on the APB.
PRESETn	Reset. The APB reset signal is active Low. This signal is usually connected directly to the system bus reset signal.
PADDR	Address. This is the APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.
PSELx	Select. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a PSELx signal for each slave.
PENABLE	Enable. This signal indicates the second and subsequent cycles of an APB transfer.
PWRITE	Write/read. This bus does a write to slave when PWRITE is High. It reads slave when PWRITE is Low. This is 1 bit.
PWDATA	Write data. This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is High. This bus can be up to 32 bits wide.
PREADY	Ready. The slave uses this signal to extend an APB transfer.
PRDATA	Read data. The selected slave drives this bus during read cycles when PWRITE is Low. This bus can be up to 32 bits wide.
PSLVERR	Slave error. This signal indicates a transfer failure. APB peripherals are not required to support the PSLVERR pin. This is true for both existing and new APB peripheral designs. Where a peripheral does not include this pin, the appropriate input to the APB bridge is tied Low.

Figure 3 on page 4 shows the state diagram for APB3 bus specification. It has three states as explained below:

- IDLE: This is the default state for the peripheral bus.
- SETUP: When a transfer is required, the bus moves to this state where the appropriate select signal PSELx is asserted. The bus remains in this state for one clock cycle only and always moves to the ACCESS state on the next rising edge of the clock.

- ACCESS: In this state, the enable signal PENABLE is asserted. The address, write, and select signals should be stable during the transition from SETUP to ACCESS state. The transition from the ACCESS state is controlled by the PREADY signal from the slave:
  - If PREADY is held Low by the slave then the peripheral bus remains in the ACCESS state.
  - If PREADY is held High by the slave and no more transfers are required, the bus transitions from the ACCESS state to the IDLE state. Alternatively, if another transfer follows, the bus moves directly to the SETUP state.

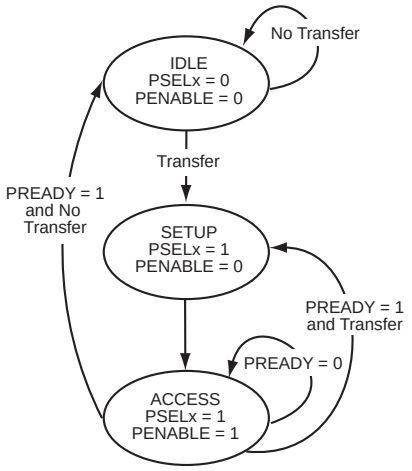

***Figure 3* • APB3 State Diagram**

Figure 4 and Figure 5 on page 5 show the APB3 timing diagrams. The APB write transfer starts with the address, write data, write signal, and select signal—all changing after the rising edge of the PCLK. In the next clock edge, the enable signal is asserted. PENABLE indicates that the Access phase is taking place. The address, data, and control signals all remain valid throughout this Access phase. The transfer completes at the end of this cycle. The enable signal, PENABLE, is deasserted at the end of the transfer. The select signal, PSELx (or PSEL), also goes Low unless the transfer is to be followed immediately by another transfer to the same peripheral. During an Access phase, when PENABLE is High, the transfer can be extended by driving PREADY Low. During a read transfer, the timing of the address (PADDR), write (PWRITE), select (PSEL), and enable (PENABLE) signals are as described in Write transfers. The slave must provide the data before the end of the read transfer. The transfer is extended if PREADY is driven Low during an Access phase.

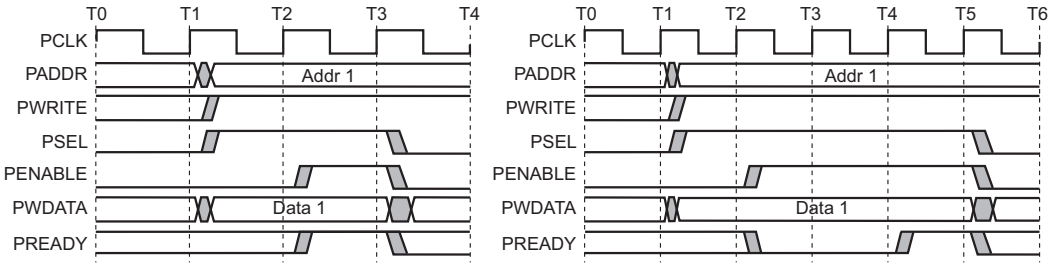

***Figure 4* • APB3 Write Transfer With and Without Wait State**

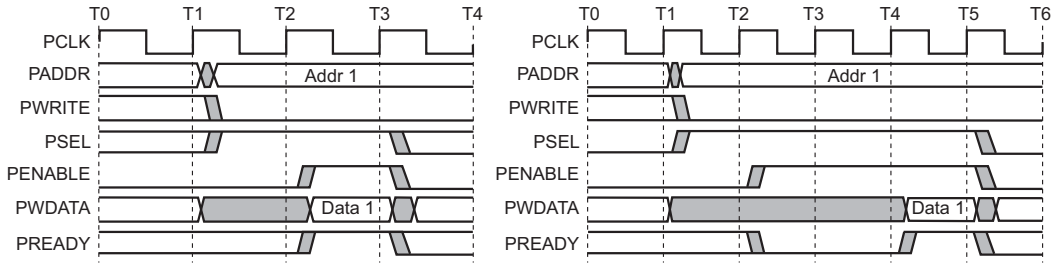

***Figure 5* • APB3 Read Transfer With and Without Wait State**

Implementing an APB3 Interface for a Custom Logic Block

This section introduces two design examples of how to create a custom APB3 wrapper for a user logic block. You can follow the same process and create an APB3 interface for your custom logic blocks implemented in the FPGA fabric. After creating the interface wrapper, connect it to the MSS and run the FPGA flow. The two appendices at the end of this document guide you through the FPGA flow using the Libero® System-on-Chip (SoC) software:

- "Appendix A: Creating a Subsystem Design with a Custom APB3 Slave"
- "Appendix B: Simulating the User Logic Block"

To create an APB3 slave, sample the address and control and send the appropriate PREADY response. If the user logic block can accept or send the data in the next cycle, asserting PREADY is not necessary; otherwise insert wait state and assert PREADY. The logic block must have a dedicated register/memory interface that communicates with the MSS via an APB3 bus. This also helps in avoiding any timing problems. When writing RTL, define the specific register that communicates with the APB3 bus.

This document presents two design examples:

- "Example 1: Memory Block with APB3 Wrapper"
- "Example 2: Counter with APB3 Wrapper"

Example 1: Memory Block with APB3 Wrapper

The example design uses a memory block of 8 bits wide by 16 bits deep, it is memory mapped to the APB3 system. It acts like an APB3 slave and is used in regular and pipelined mode. Figure 6 shows the block diagram. Figure 7 and Figure 8 show the timing diagrams for regular and pipelined mode.

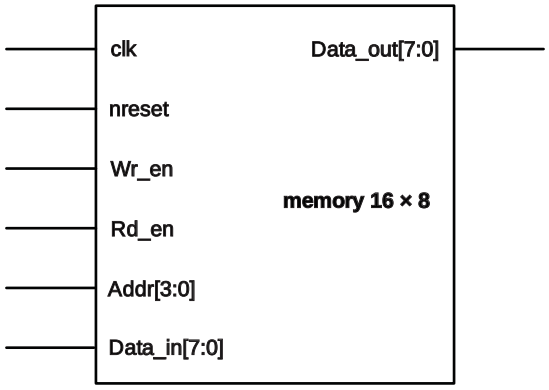

***Figure 6* • Block Diagram – Memory Block**

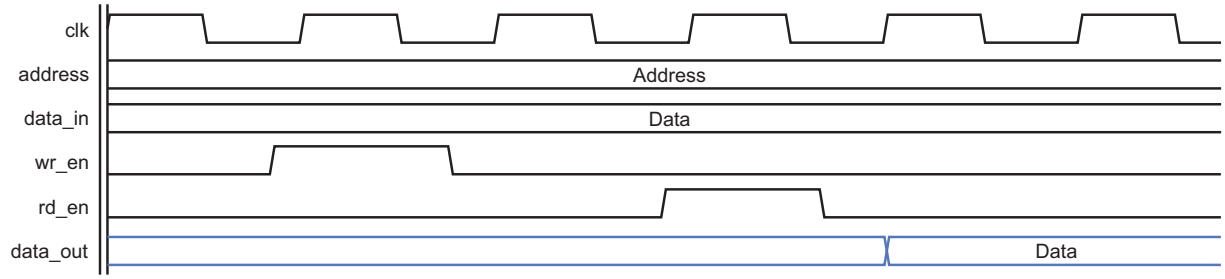

***Figure 7* • Timing Diagram for Regular Mode**

***Figure 8* • Timing Diagram for Pipelined Mode**

Creating an APB3 slave memory module is very simple. Create a wrapper that interfaces between the APB3 signal and memory block. The wrapper logic must generate the write enable when the PSEL, PWRITE, and PENABLE signals are active. For read enable, user logic must generate the signal during the first cycle so that the data is ready on the bus during the second cycle. The address and data signals connect directly to the memory block address and data ports. Figure 9 shows the RTL view for the wrapper. The write enable and read enable signals are generated as shown in the following verilog code example:

```
assign wr_enable = (PENABLE &&  PWRITE && PSEL);
assign rd_enable = (!PWRITE && PSEL);
```

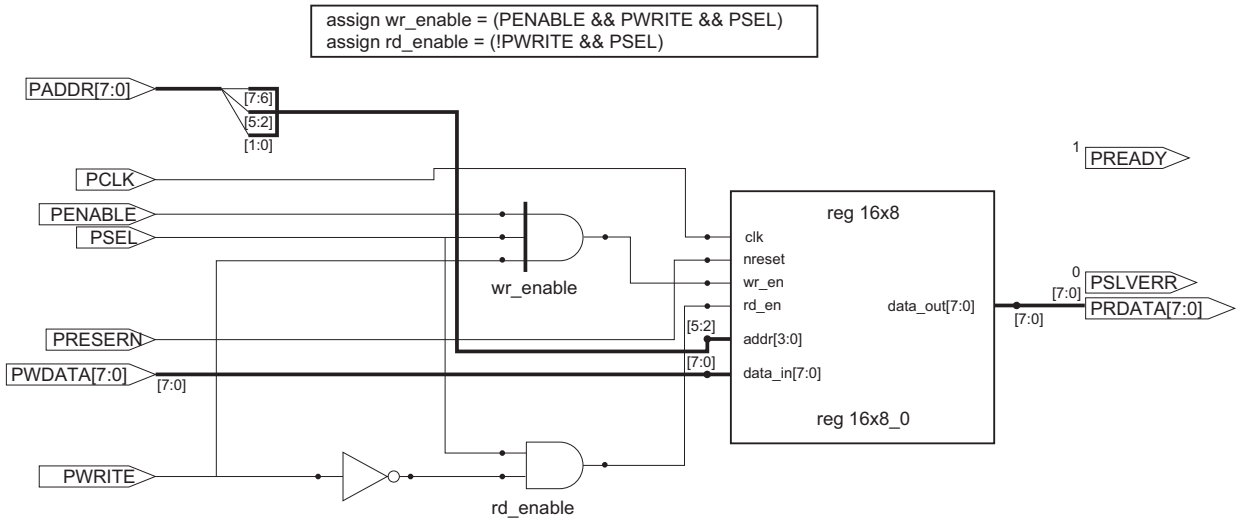

***Figure 9* • RTL View for the APB Slave Wrapper**

To create an APB3 wrapper on the pipelined memory, user logic must use the PREADY signal to insert a wait state. Additionally, the user logic must generate the write enable when the PSEL, PWRITE, and PENABLE signals are active. For read enable, the user logic must generate the signal during the first cycle. However, there is a need to add an extra cycle due to the pipeline option. Figure 10 on page 8 shows the state diagram and sample Verilog code for this wrapper.

```
case (fsm)
     2'b00 :  begin
                  if (~PSEL)
                        begin
                          fsm <= 2'b00;
                        end
                  else
                        begin
                          fsm <= 2'b01;
                          if (PWRITE)
                                begin
                                    rd_enable <= 1'b0;
                                    wr_enable <= 1'b1;
                                    PREADY <= 1'b1;
                                end
                          else
                                begin
                                    rd_enable <= 1'b1;
                                    wr_enable <= 1'b0;
                                    PREADY <= 1'b0;
                                end
                        end
               end
     2'b01 :  begin
                          if (PWRITE)
                            begin
                                rd_enable <= 1'b0;
                                wr_enable <= 1'b0;
                                PREADY <= 1'b1;
                                fsm <= 2'b00;
                            end
                          else
                            begin
                                rd_enable <= 1'b0;
                                wr_enable <= 1'b0;
                                PREADY <= 1'b0;
                                fsm <= 2'b10;
                            end
              end
     2'b10 :  begin
                          fsm <= 2'b00;
                          PREADY <= 1'b1;
              end
default : fsm <= 2'b00;
```

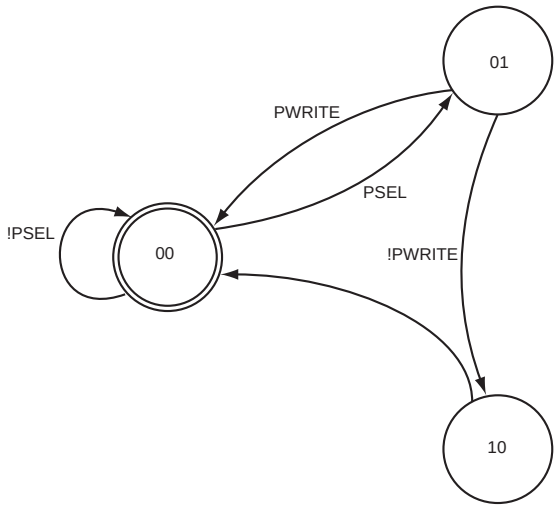

Figure 10 • **State Diagram and Sample Verilog Code for APB3 Slave Wrapper with Wait State**

The memory block with APB3 wrapper can be connected to the MSS as described in "Appendix A: Creating a Subsystem Design with a Custom APB3 Slave".

The sample Libero SoC project is available for download as a part of the design files from www.microsemi.com/soc/download/rsc/?f=A2F_AC335_DF.

Example 2: Counter with APB3 Wrapper

This design example uses a simple 32-bit counter. A wrapper is used to interface the counter to the APB3 bus. For illustration purposes, the master on the APB3 bus can load the counter value and can also enable the counter. The master can also read the counter value and check the status. Three registers are used in the wrapper to interface with the APB3 bus. Following is the sample VHDL code for this wrapper:

```
    constant COUNTERLOADA     : std_logic_vector(4 downto 0) := "00000";
    constant COUNTERVALUEA    : std_logic_vector(4 downto 0) := "00100";
    constant COUNTERCONTROLA  : std_logic_vector(4 downto 0) := "01000";
……..
    p_reg_seq : process (PRESETn, PCLK)
    begin
        if (PRESETn = '0') then
                TimerEn     <= '0';
                Load        <= (others => '0');
                LoadEnReg   <= '0';
        elsif (PCLK'event and PCLK = '1') then
          if (PWRITE = '1' and PSEL = '1' and PENABLE = '1' and PADDR = COUNTERCONTROLA)
then
                TimerEn    <= PWDATA (0);
                LoadEnReg  <= '0';
          elsif (PWRITE = '1' and PSEL = '1' and PENABLE = '1' and PADDR = COUNTERLOADA)
then
                Load        <= PWDATA(31 downto 0);
                LoadEnReg  <= '1';
            end if;
        end if;
    end process p_reg_seq;

my_counter: Count32
    port map(Clock    => PCLK,
       Q            => Count,
```

```
    Aclr         => PRESETn,
    Enable       => TimerEn,
    Sload        => LoadEnReg,
    Data         => Load,
    Tcnt         => TIMINT);

-------------------------------------------------------------------------------
-- Output data generation
-------------------------------------------------------------------------------
p_data_out : process (PWRITE, PSEL, PADDR, Load, Count, TimerEn)
begin
    DataOut <= (others => '0'); -- Drive zeros by default
    if (PWRITE = '0' and PSEL = '1') then
        case PADDR is
            when COUNTERLOADA =>
                DataOut(31 downto 0) <= Load;
            when COUNTERVALUEA =>
                DataOut(31 downto 0) <= Count;
            when COUNTERCONTROLA =>
                DataOut(0) <= TimerEn;
            when others =>
                DataOut <= (others => '0');
        end case;
    else
        DataOut <= (others => '0');
    end if;
end process p_data_out;

-- Generate PRDATA on falling edge
p_PRDATA : process (PRESETn, PCLK)
begin
    if (PRESETn = '0') then
        DataOut_int <= (others => '0');
    elsif (PCLK'event and PCLK = '1') then
        if (PWRITE = '0' and PSEL = '1') then
            DataOut_int <= DataOut;
        end if;
    end if;
end process p_PRDATA;

PRDATA <= DataOut_int;
```

The counter with APB3 wrapper can be connected to the MSS as described in "Appendix A: Creating a Subsystem Design with a Custom APB3 Slave".

The sample Libero SoC is available for download as a part of the design files from www.microsemi.com/soc/download/rsc/?f=A2F_AC335_DF.

Appendix A: Creating a Subsystem Design with a Custom APB3 Slave

This appendix describes how to connect the custom APB slave interface to the MSS.

1. In the SmartDesign framework, configure the fabric interface inside the MSS configurator by double-clicking the fabric interface sub-configurator (Figure 11) and configuring the FIC with an APB3 interface.

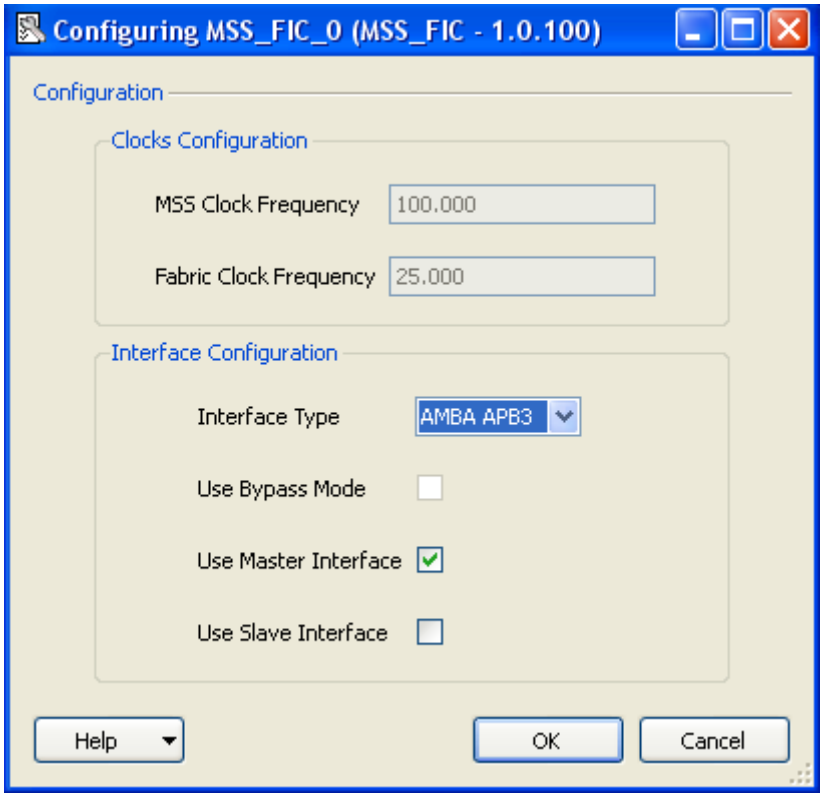

***Figure 11* • Configuring the FIC**

2. Select **MSS_Master_APB** bus interface on the fabric interface and promote to the top level, as shown in Figure 12.

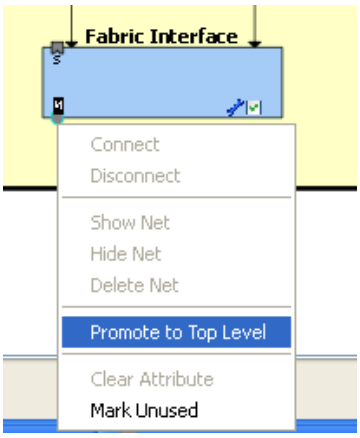

Figure 12 • **Promoting MSS_Master_APB Bus Interface to Top Level**

3. Configure the Clock Management block and promote FAB_CLK to the top level. Similarly, configure the Reset Management block and promote M2F_RESET_N to the top level.
4. Configure the other blocks inside the MSS configurator as needed and generate the MSS.
5. Expand the Bus Interfaces section in the catalog. Select **CoreAPB3 3.0.103** and drag it into the canvas.
6. Import or create the RTL code for the custom APB3 slave and right-click the RTL source code file in the **Design Hierarchy** tab and select **Create Core From HDL**.
7. Select **Add Bus Interface** in the Edit Core Definition window and then select a bus definition from the **Select Bus Definition** window.

   The Edit Core Definition window is displayed as shown in Figure 13. Select **Map by Name** to map the signals automatically. The tool attempts to map any similar signal names between the bus definition and pin names on the instance. Map other signals manually that are not mapped by **Map by Name**.

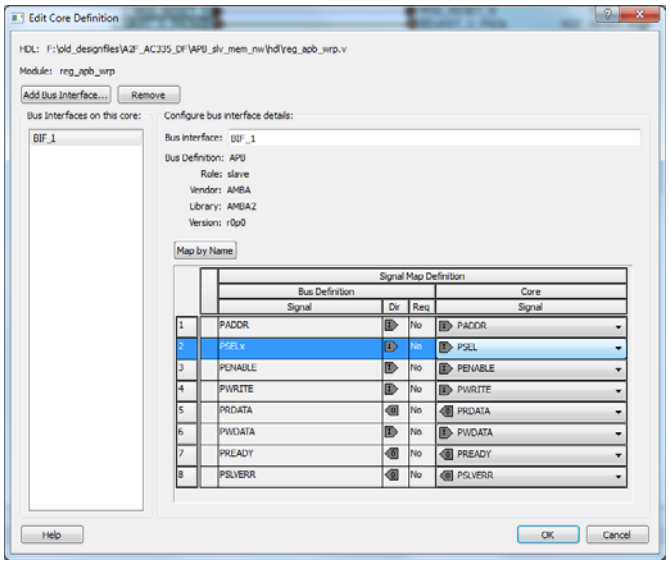

Figure 13 • **Add Bus Interface Dialog Box**

8. Select the RTL source code for the custom APB3/AHB-Lite wrapper in the **Design Hierarchy** tab and drag it onto the canvas.

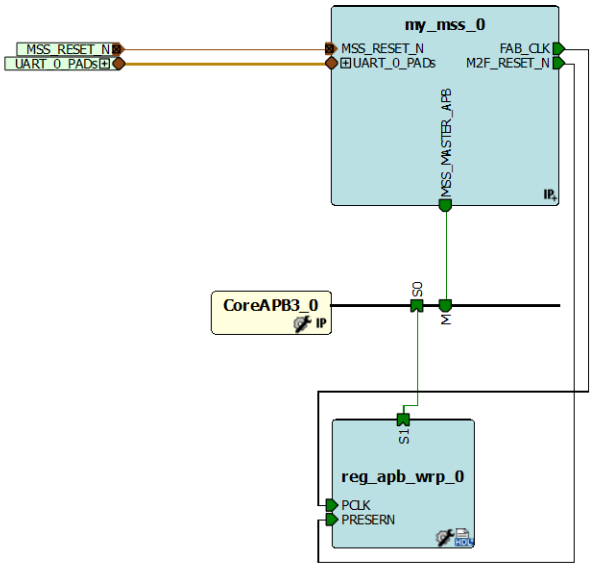

Figure 14 • **Connection Between Custom APB3 Slave and MSS**

9. Connect the FAB_CLK clock signal to the PCLK clock signal of the MSS.
10. Connect the M2F_RESET_N signal from FIC to the PRESETN signals of the fabric APB3 slave.
11. Connect other signals present on the fabric AMBA subsystem as needed, and generate the SmartDesign block.

Appendix B: Simulating the User Logic Block

This section describes how to simulate user logic or an IP block using the BFM models. When the RTL code is generated by SmartDesign, the tool also creates three BFM files: test.bfm, user.bfm, and Subysystem.bfm. You need a customized User.bfm file to emulate Cortex-M3 processor transactions in the system. The following section describes the simulation setup for the memory block with APB3 wrapper design. Follow the same process for a user APB3 interface.

1. Open the BFM test script file (user.bfm) in the Libero SoC editor or any text editor. The script file appears under Simulation on the **Libero Files** tab.

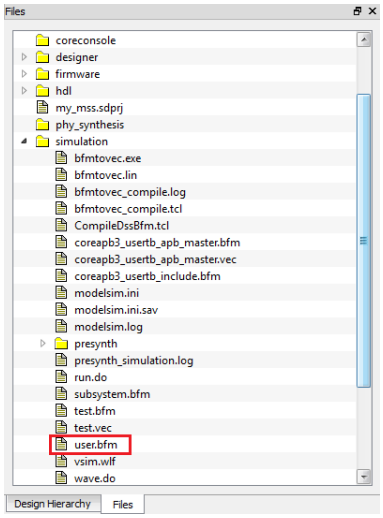

***Figure 15* • test.bfm Script File**

2. Add or modify the lines shown in bold font below to test.bfm. Note that the BFM command should map the address and register setting for your design.

```
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:
print "                                                                ";
print "                                                                ";
print "****************************************************************";
print "Testing custom APB slave";
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
return
```

3. Save the file user.bfm.

Important: Note that the BFM test script as test.bfm will be overwritten if the core is regenerated. Microsemi recommends that you keep a backup copy.

4. Change simulation runtime to meet your needs and click the **Simulation** button under Verify Pre-Synthesized Design in the Libero SoC Design Flow to run pre-synthesis simulation. You must add the correct waveform to see the signal toggling.

List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 3 (February 2012)	Removed the '.zip' from URL in the application note (SAR 36763).	
Revision 2 (January 2012)	Modified points 5, 6, 7, and 8 in "Appendix A: Creating a Subsystem Design with a Custom APB3 Slave" section (SAR 35798).	11
	Placed updated images for Figure 13, Figure 14 and Figure 15 (SAR 35798).	11, 12, 13
	Modified points 1 and 3 in "Appendix B: Simulating the User Logic Block" section (SAR 35798).	13
Revision 1 (October 2010)	Modified Figure 2.	2

*Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.*

Microsemi Corporate Headquarters
One Enterprise Drive, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Outside the USA: +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**.

51900203-3/02.12