Rate This Document

Application Note AC333

Connecting User Logic to the SmartFusion Microcontroller Subsystem

Introduction

SmartFusion^TM contains a hard microcontroller subsystem (MSS), programmable analog circuitry, and FPGA fabric, consisting of logic tiles, SRAM, and PLLs. The microcontroller subsystem, or MSS, consists of an ARM®Cortex™-M3 processor, advanced high-performance bus (AHB) matrix, system registers, Ethernet MAC, several peripherals, DMA engine, real-time counter (RTC), embedded nonvolatile memory (eNVM), embedded SRAM (eSRAM), and fabric interface controller (FIC). Refer to the *SmartFusion Intelligent Mixed-SIgnal FPGAs* datasheet for the architectural details of the MSS. The peripherals included in the MSS are communication functions, such as Ethernet MAC, UART, SPI, and I^{2}C, and timers, such as the system timer and watchdog timer, similar to those found in various microcontrollers. The AHB Bus Matrix connects the peripherals within the MSS and also connects to user peripherals in the FPGA fabric through the FIC. The FIC block performs an AHB-Lite or advanced peripheral bus3 (APB3) bridging function between the AHB bus matrix and the FPGA fabric. Note that AHB-Lite and APB3 are defined in the Advanced Microcontroller Bus Architecture-Lite (AMBA®-Lite) specification. FIC allows you to create custom logic that can map into the memory space of the Cortex-M3 and other masters on the AHB bus. Similarly, you can implement an AHB-Lite or APB3 master in the fabric that can access any slave on the AHB bus matrix, including those embedded within the MSS. Since the FIC has an AMBA interface to the fabric, user logic must implement AMBA protocols in order to communicate through the FIC. This application note explains how to create an AHB-Lite or APB3 wrapper on custom logic and how to connect it to the MSS system through the FIC.

Overview of AMBA

The advanced high-performance bus (AMBA) specification defines an on-chip communications standard for designing high-performance embedded microcontrollers. Rev2.0 of the AMBA specification introduced the Advanced High-Performance Bus (AHB) protocol. The AMBA-AHB is used for high-performance, high clock frequency system modules and the AMBA-APB is used for low-power peripherals. An AMBA-based microcontroller typically consists of a high-performance system backbone bus and may also have a bridge to the lower bandwidth APB, where most of the peripheral devices in the system are located. The advanced high-performance bus Lite (AHB-Lite) is a newer version of AMBA specification.

An AHB-Lite transfer consists of two distinct sections:

- Address phase, which lasts only a single cycle
- The data phase, which may require several cycles

The simplest transfer is one with no wait states, so the transfer consists of one address cycle and one data cycle. During the simple transfer, the master drives the address and control signals onto the bus after the rising edge of HCLK. The slave then samples the address and control information on the next rising edge of HCLK. After the slave has sampled the address and control signals, it sends the appropriate HREADY response. This response is sampled by the master on the third rising edge of HCLK.

For write operations, the bus master will hold the data stable throughout the extended cycles. For read transfers, the slave does not have to provide valid data until the transfer is about to complete. A slave can insert wait states into any transfer to enable additional time for completion. Figure 1 on page 2 shows a simple write transfer and a read transfer with two wait states. Note that the control signals define the transfer type, size, etc.

These control signals have exactly the same timing as the address bus. Table 1 shows the AHB-Lite signals. For more information, visit the ARM web site.

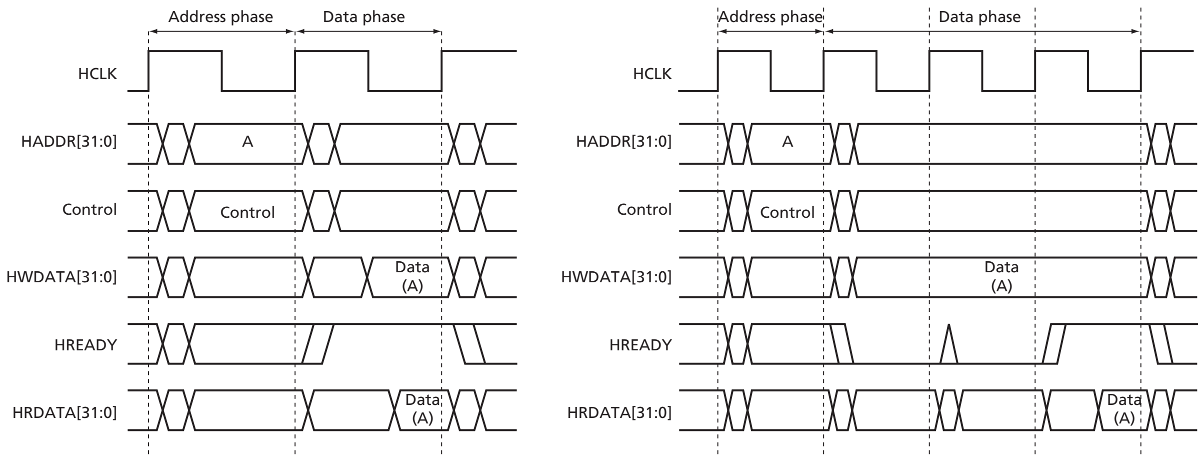


Figure 1 • **AHB-Lite No-Wait Write Transfer (left) and Two Wait State Read Transfer (right)**

Table 1 • **AHB-Lite Signals**

Signal Name	**Description**
HCLK	The bus clock times all bus transfers. All signal timings are related to the rising edge of HCLK.
HRESETn	The bus reset signal is active Low and resets the system and the bus.
HADDR[31:0]	The 32-bit system address bus
HBURST[2:0]	The burst type indicates if the transfer is a single transfer or forms part of a burst. Typically single burst is used that refers to HBURST= 000.
HMASTLOCK	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wants to implement some level of protection.
HSIZE[2:0]	Indicates the size of the transfer; typically byte, halfword, or word are used.
HTRANS[1:0]	Slave. Indicates the transfer type of the current transfer. This can be IDLE, BUSY, NONSEQUENTIAL, or SEQUENTIAL.
HWDATA[31:0]	The write data bus transfers data from the master to the slaves during write operations.
HWRITE	Indicates the transfer direction. When High, this signal indicates a write transfer and when Low, a read transfer.
HRDATA[31:0]	During read operations, the read data bus transfers data from the selected slave.
HREADYOUT	When High, the HREADYOUT signal indicates that a transfer has finished on the bus.
HRESP	The transfer response. When Low, the HRESP signal indicates that the transfer status is OKAY. When High, the HRESP signal indicates that the transfer status is ERROR.
HSELx	Each AHB-Lite slave has its own slave select signal, HSELx, and this signal indicates that the current transfer is intended for the selected slave.
HRDATA[31:0]	Master Read data bus.
HREADY	When High, the HREADY signal indicates to the master and all slaves that the previous transfer is complete.

APB3 has a similar write and read transfer. The APB3 write transfer starts with the address, write data, write signal, and select signal all changing after the rising edge of PCLK. In the next clock edge the enable signal, PENABLE, is asserted, and this indicates that the Access phase is taking place.

The address, data, and control signals all remain valid throughout the Access phase. The transfer completes at the end of this cycle. The enable signal, PENABLE, is de-asserted 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, write, select, and enable 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 2 shows a simple write transfer and a write transfer with two wait states. Table 2 on page 4 shows APB signals.

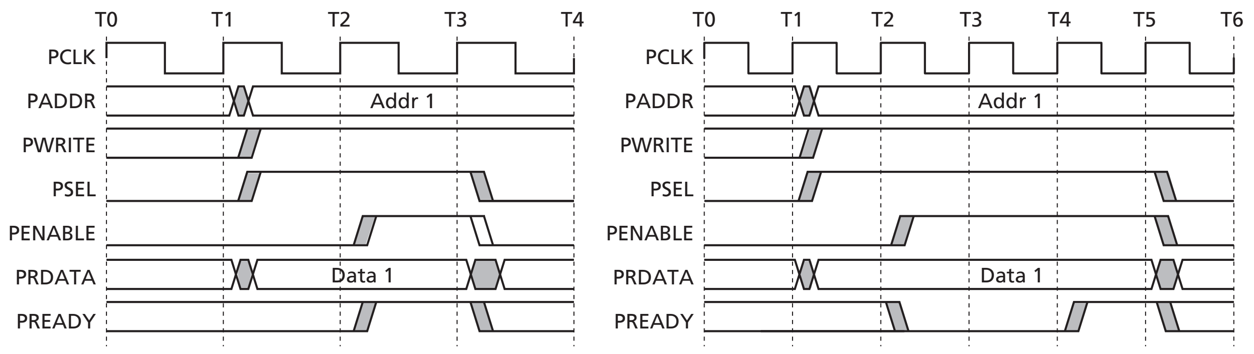


Figure 2 • **APB3 Write Transfer**

Figure 3 shows an APB3 read transfer with and without wait states.

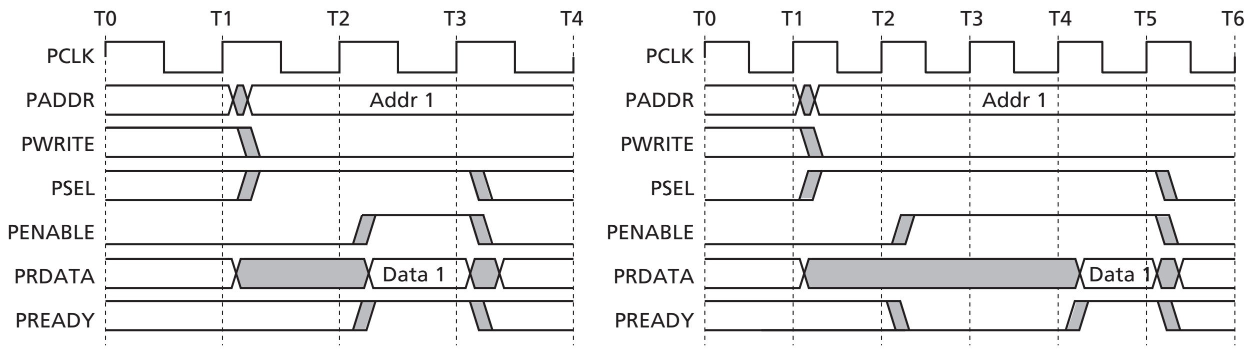


Figure 3 • **APB3 Read Transfer**

Table 2 • **APB3 Signals**

Signal Name	Description
PCLK	Clock. The rising edge of PCLK times all transfers on the APB3.
PRESETn	Reset. The APB3 reset signal is active Low. This signal is normally connected directly to the system bus reset signal.
PADDR	Address. This is the APB3 address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.
PSELx	Select. The APB3 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 APB3 transfer.
PWRITE	Direction. 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.
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 APB3 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	This signal indicates a transfer failure. APB3 peripherals are not required to support the PSLVERR pin. This is true for both existing and new APB3 peripheral designs. Where a peripheral does not include this pin, the appropriate input to the APB bridge is tied Low.

AMBA Interface in the Fabric Interface Controller (FIC)

The fabric interface controller (FIC) is part of the microcontroller subsystem (MSS) and performs an AHB-Lite to AHB-Lite or AHB-Lite to APB3 bridging function between the MSS AHB Bus Matrix and an AHB-Lite or APB3 bus in the FPGA fabric. It provides two buses between the MSS and the fabric. The first is mastered by the MSS and has slaves in the fabric and the second has a master in the fabric and slaves in the MSS.

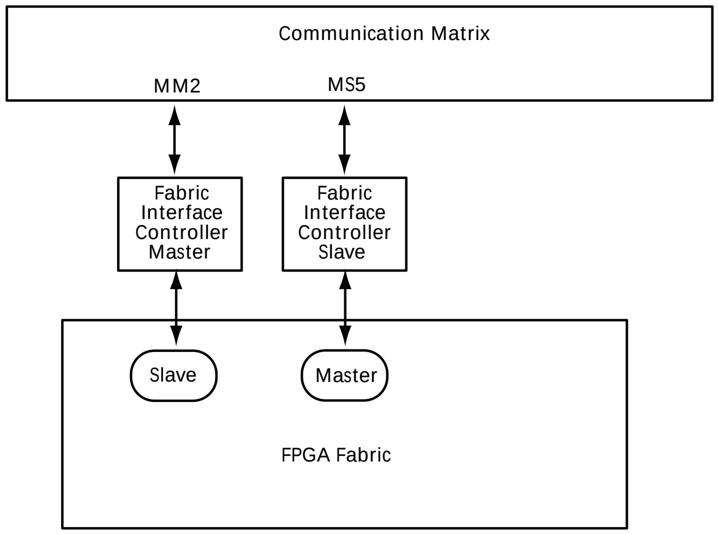

Communication Matrix
MM2
MS5
Fabric Interface Controller Master
Fabric Interface Controller Slave
Slave
Master
FPGA Fabric

Figure 4 • **Connecting User logic to MSS**

The interface to the fabric can be of the 32-bit AHB-Lite or 32-bit APB3 type. However, only one type of interface can be enabled at any given time. Although it is possible to have master and slave at the same time, in general the FIC will allow four possible communication scenarios:

- MSS master to FPGA AHB-Lite slave interface
- MSS master to FPGA APB3 slave interface
- Fabric AHB-Lite master to MSS slave
- Fabric APB3 master to MSS slave

In order to connect to the FIC, the AHB-Lite or APB3 interface must be added to custom user logic. The interface must be an AHB-Lite compliant master or APB3 compliant master. For the slave case, the interface must be an AHB-Lite compliant slave or an APB3 compliant slave interface.

To create an AHB-Lite master interface in the fabric, user logic must drive the address and control signals onto the bus after the rising edge of HCLK. This is followed by data phase. In the data phase the fabric provides data for the write operation and samples data from the FIC for read operation. HREADY is used to insert wait states when a slow peripheral is accessed. To create an AHB-Lite slave in the fabric, user logic samples the address and control and deasserts HREADY if it needs to insert a wait state. For read transfers, the user logic block must make sure that the valid data is available on the bus before asserting HREADY. For a write operation, the slave will sample data while de-asserting HREADY.

Similarly for the APB3 master, the write transfer starts with the address, write data, write signal, and select signal all changing after the rising edge of the clock and asserting PENABLE in the next cycle. If the salve wants to extend the transfer by sending PREADY Low, wait for PREADY to go High and finish the cycle. To create an APB slave, sample the address and control and send the appropriate HREADY response. If the user logic block can accept or send the data in the next cycle, asserting HREADY is not necessary; otherwise insert a wait sate and assert HREADY.

Implementing a Custom APB3 or AHB-Lite Interface

The following sections provide detailed information on creating APB3 and AHB-Lite interfaces, including example RTL source code. After creating the interface wrapper, you must connect to the MSS and then go through the FPGA flow. Refer to the memory map section in the SmartFusion Microcontroller Subsystem User’s Guide for addressing the FPGA fabric from the MSS and addressing various peripherals inside the MSS. This application note has two appendices to guide you on the various FPGA flows and finishing your design using Actel Libero® Integrated Design Environment (IDE):

- "Appendix A: Creating a Subsystem Design with a Custom APB3 or AHB-Lite to MSS"
- "Appendix B: Simulating the User Logic Block"

Custom APB3 Slave

This section shows a design example for creating a custom APB3 wrapper on a register/memory block. You can follow the same technique and create an APB3 interface on counters, registers, or any other custom logic blocks. The example uses a memory block 8 bits wide by 16 bits deep as an APB3 slave. This section describes the APB3 wrapper creation for regular and pipelined mode registers and memory.

Figure 5 shows the memory block diagram. Figure 6 and Figure 7 on page 6 show the timing diagrams for regular and pipelined mode.

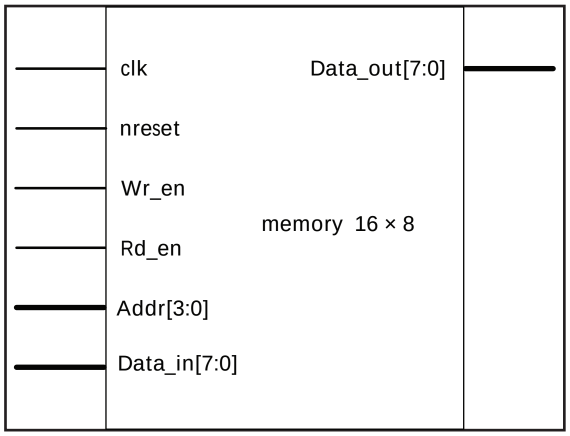


Figure 5 • **Custom Memory Slave Block Diagram**

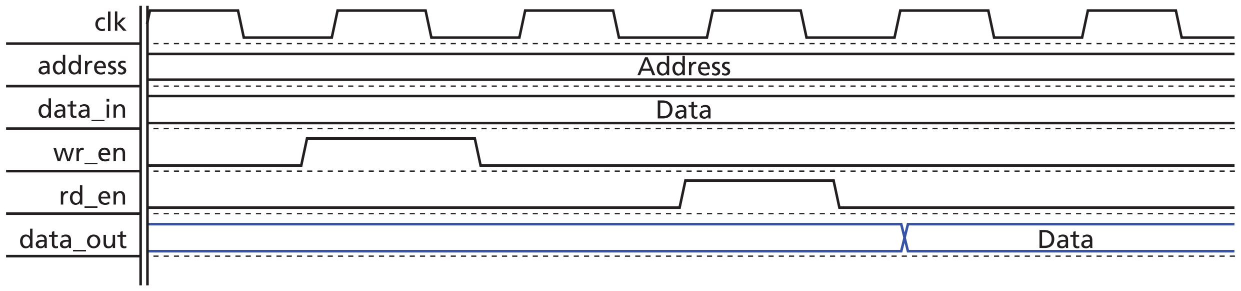


Figure 6 • **Custom Memory Timing Diagram for Regular Mode**

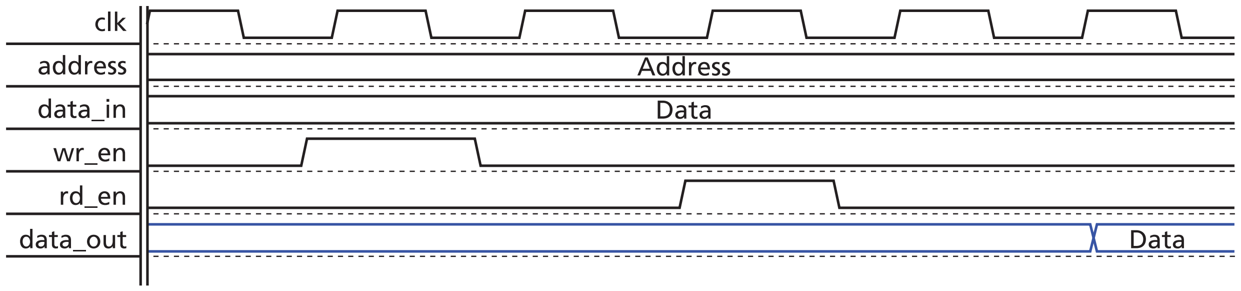


Figure 7 • **Custom Memory Timing Diagram for Pipelined Mode**

Creating an APB3 wrapper for regular mode is simple. User logic must generate the write enable when PSEL, PWRITE, and PENABLE signals are active. For read enable, user logic must generate the signal during the first cycle so that data is ready on the bus during the second cycle. The address and data signals connect directly to the memory blocks.

Figure 8 shows the RTL view for the wrapper. The write enable and read enable signals are generated as shown in the verilog code example below:

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

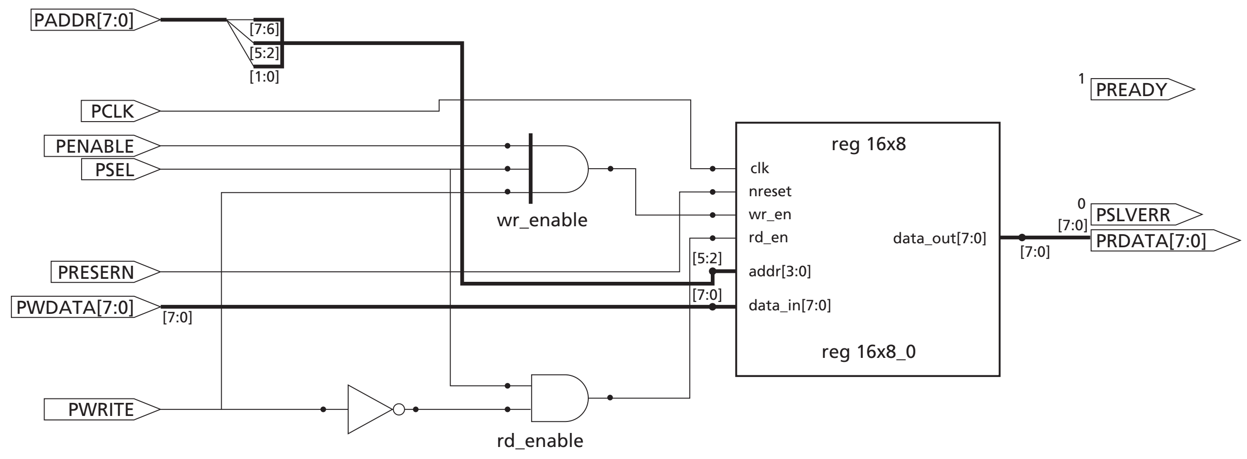


Figure 8 • **RTL View for the APB3 Slave Wrapper with No Wait State**

To create an APB3 wrapper on the pipelined memory, user logic must use the PREADY signal to insert a wait state. User logic must generate the write enable when PSEL, PWRITE, and PENABLE signals are active. For read enable, user logic must generate the signal during the first cycle; however, an extra cycle must be added due to the pipeline option. Figure 9 shows the state diagram.

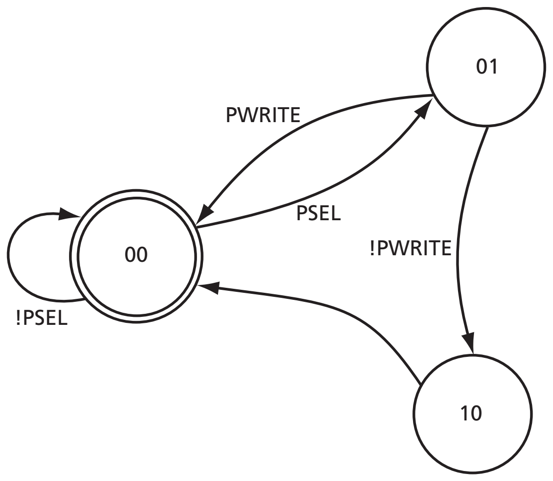


Figure 9 • **State Diagram for APB3 Slave Wrapper with Wait State**

Sample Verilog Code for APB3 Slave Wrapper with Wait State

```
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;
```

The sample RTL source code for custom APB3 slave with and without wait state is available for download on the Actel website: www.actel.com/download/rsc/?f=User_Logic_MSS_DF.

The sample code is in the apb_slave_fabric_nw and apb_with_wait folders within the zip file.

Custom AHB-Lite Slave

This section shows a design example for creating a custom AHB-Lite wrapper on a register/memory block. You can follow the same technique and create an AHB-Lite interface on counters, registers, or any other custom logic blocks. Note that the HSIZE signal in AHB-Lite transfer indicates the size of the transfer, which is typically byte (HSIZE = 000), halfword (HSIZE = 001), or word (HSIZE = 010). The design example uses four memory blocks of 8 bits wide by 16 bits deep as an AHB-Lite slave to show byte-controlled transaction.

Figure 10 shows the byte alignment of the AHB-Lite data bus based on the HSIZE signal.

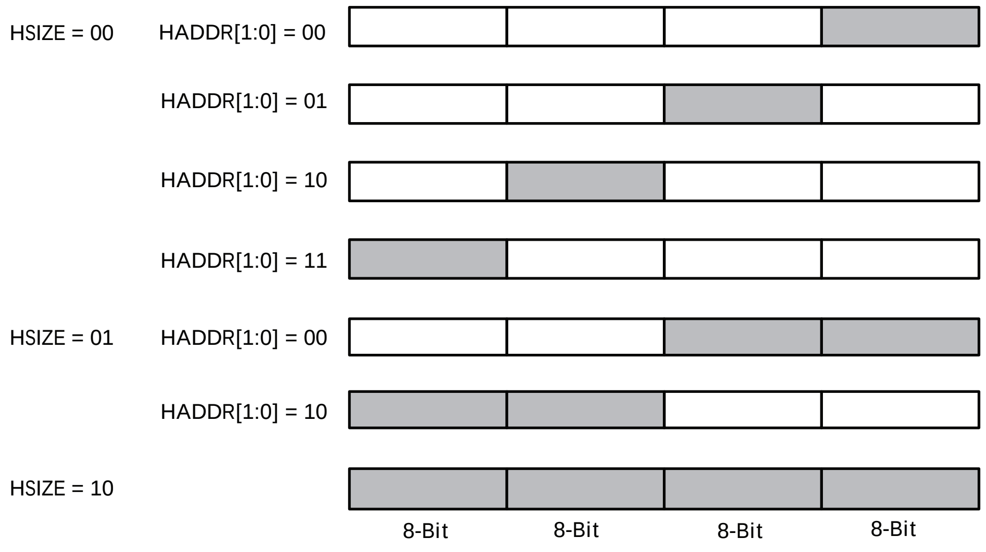


Figure 10 • **Alignment of the APB Bus Data**

To create an AHB-Lite slave, the user logic block must check the address and control signals and generate appropriate read or write enable signals. Depending on HSIZE signal, it will generate only read or write enable signals for the appropriate bytes. During the data read, HREADY_out is used to add an extra cycle. Figure 11 shows the state diagram for the AHB-Lite slave wrapper.

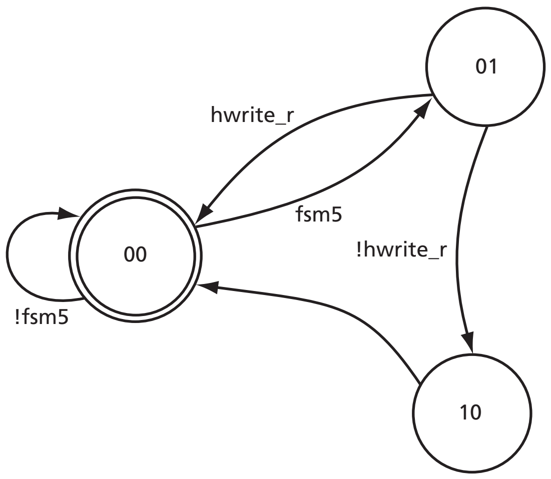


Figure 11 • **State Diagram for AHB-Lite Slave Wrapper**

Sample Verilog Code for AHB-Lite Slave Wrapper

```
case (FSM)
   2'b00 : begin
      if(HTRANS[1] && HREADY && HSEL)
         begin
            fsm <= 2'b01; //move to next state
            mem_addr <= HADDR[5:0]; //store the address value
            hwrite_r <= HWRITE; //store write signal internally

         if(HWRITE)
            begin
               case(HSIZE)
                  3'b000 :
                     begin
                     if (HADDR[1:0] == 2'b00)
                        mem_wen00 <= 1'b1;
                     else if (HADDR[1:0] == 2'b01)
                        mem_wen01 <= 1'b1;
                     else if (HADDR[1:0] == 2'b10)
                        mem_wen10 <= 1'b1;
                     else
                        mem_wen11 <= 1'b1;
                     end
                  3'b001 :
                     begin
                     if (HADDR[1:0] == 2'b00)
                        begin
                        mem_wen00 <= 1'b1;
                        mem_wen01 <= 1'b1;
                        end
                     else
                        begin
                        mem_wen10 <= 1'b1;
                        mem_wen11 <= 1'b1;
                        end
                     end
                  3'b010 :
                     begin
                     mem_wen00 <= 1'b1;
                     mem_wen01 <= 1'b1;
                     mem_wen10 <= 1'b1;
                     mem_wen11 <= 1'b1;
                     end
                  default :
                     begin
                     mem_wen00 <= 1'b0;
                     mem_wen01 <= 1'b0;
                     mem_wen10 <= 1'b0;
                     mem_wen11 <= 1'b0;
                     end
               endcase
            end
         else
            begin
               HREADYOUT_int <= 1'b0; // insert wait-state
               case(HSIZE)
                  3'b000 :
                     begin
                     if (HADDR[1:0] == 2'b00)
                        mem_ren00 <= 1'b1;
                     else if (HADDR[1:0] == 2'b01)
                        mem_ren01 <= 1'b1;
                     else if (HADDR[1:0] == 2'b10)
                        mem_ren10 <= 1'b1;
                     else
                        mem_ren11 <= 1'b1;
                     end
                  3'b001 :
                     begin
                     if (HADDR[1:0] == 2'b00)
                        begin
                        mem_ren00 <= 1'b1;
                        mem_ren01 <= 1'b1;
                        end
                     else
                        begin
                        mem_ren10 <= 1'b1;
                        mem_ren11 <= 1'b1;
                        end
                     end
                  3'b010 :
                     begin
                     mem_ren00 <= 1'b1;
                     mem_ren01 <= 1'b1;
                     mem_ren10 <= 1'b1;
                     mem_ren11 <= 1'b1;
                     end
                  default :
                     begin
                     mem_ren00 <= 1'b0;
                     mem_ren01 <= 1'b0;
                     mem_ren10 <= 1'b0;
                     mem_ren11 <= 1'b0;
                     end
               endcase
            end
         end
      end

   2'b01 : begin
      if (hwrite_r)
         begin
         fsm <= 2'b00; //done
         mem_wen00 <= 1'b0;
         mem_wen01 <= 1'b0;
         mem_wen10 <= 1'b0;
         mem_wen11 <= 1'b0;
         end
      else
         begin
         fsm <= 2'b10; //de-assert read+move to next state
         HREADYOUT_int <= 1'b1;
         mem_ren00 <= 1'b0;
         mem_ren01 <= 1'b0;
         mem_ren10 <= 1'b0;
         mem_ren11 <= 1'b0;
         end

      end


   2'b10 : begin
      fsm <= 2'b00; //change to state 00
      end
   default : fsm <= 2'b00;
endcase
```

Sample RTL for the custom AHB-Lite slave is available for download on the Actel website: www.actel.com/download/rsc/?f=User_Logic_MSS_DF. The sample RTL is placed under the folder labeled AHB_slave_fabric in the zip file.

Custom APB3 Master

This section explains how to hook up a custom APB3 master to the MSS using SmartDesign. It is less likely that a user will write custom logic to create an APB3 master. Instead, Actel recommends you use an Actel IP core—Core8051s or CoreABC—as an APB master. The state diagram for the custom APB3 master is shown here. Figure 12 shows the block diagram for the CoreABC connected to MSS via CoreAPB3.

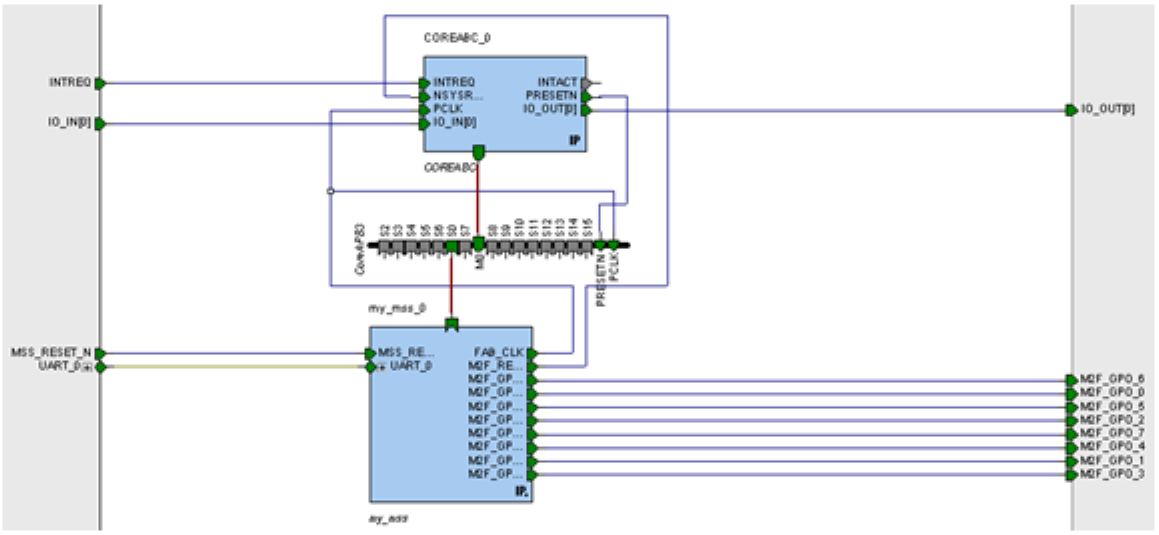


Figure 12 • **CoreABC connected to MSS via CoreAPB3**

CoreAPB3 has two adding modes: direct addressing mode and indirect addressing mode. To connect CoreABC to MSS via CoreAPB3 requires you to select the Indirect Addressing mode. In the Indirect addressing mode, slot 1 is used to store an address register, and slot 0 uses the indirect address register to set the upper bits of a 32-bit PADDR bus to allow a full 32-bit address range for slot 0. Refer to the CoreAPB3 Handbook for details. Also, when CoreABC is used as a master in CoreAPB3, the APB Slot Size configuration option settings should match for both of these cores.

Lets say you want to write to MSS GPIO using CoreABC. To write to MSS GPIO, you need to write to the GPIO_OUT register. The memory map for GPIO_OUT is 0x40013088. In order to do that, you need to configure CoreAPB3 with the indirect addressing mode and set the slot size for both CoreABC and CoreAPB3 to 64k. Then use the following steps in CoreABC:

1. Write 0x40010000 to Slot1 to set IADDR[31:16] = 0x4001. Note that writes to IADDR[15:0] will be ignored since the lower log2(RANGESIZE) = 16 bits of the IADDR register are ignored.
2. Write data word to PADDR[19:0] = 0x03088 (slave 0 address 0x8000), which CoreAPB3 passes to PADDRS0[31:0] = (IADDR[31:16] concatenated with PADDR[15:0]) = 0x40013088.

Here is the sample CoreABc program:

```
APBWRT DAT 1 0x0000 0x40010000
JUMP $MAIN
//-------------------------
// Main loop
//-------------------------
$MAIN
    $LOOP
    APBWRT DAT 0 0x3088 0x00000000
```

A sample design showing CoreABCs connected to the FIC is available for download on the Actel website: www.actel.com/download/rsc/?f=User_Logic_MSS_DF. The sample design is placed under the folder labeled APB_master_fabric in the zip file.

Custom AHB-Lite Master

This section gives a design example for creating a custom AHB-Lite master in the user logic block. To create an AHB-Lite master, the user logic block must drive the address and control signals onto the bus after the rising edge of HCLK. The user logic must also wait if HREADY is Low. Once HREADY is High, go to the data phase. During the data phase, if HREADY is Low, user logic must hold the data stable throughout the extended cycles for a write operation, or read the data only after HREADY is High. Figure 13 shows the state diagram for the AHB-Lite master. Sample verilog code is shown in the "Sample Verilog Code for AHB-Lite Master" section on page 14.

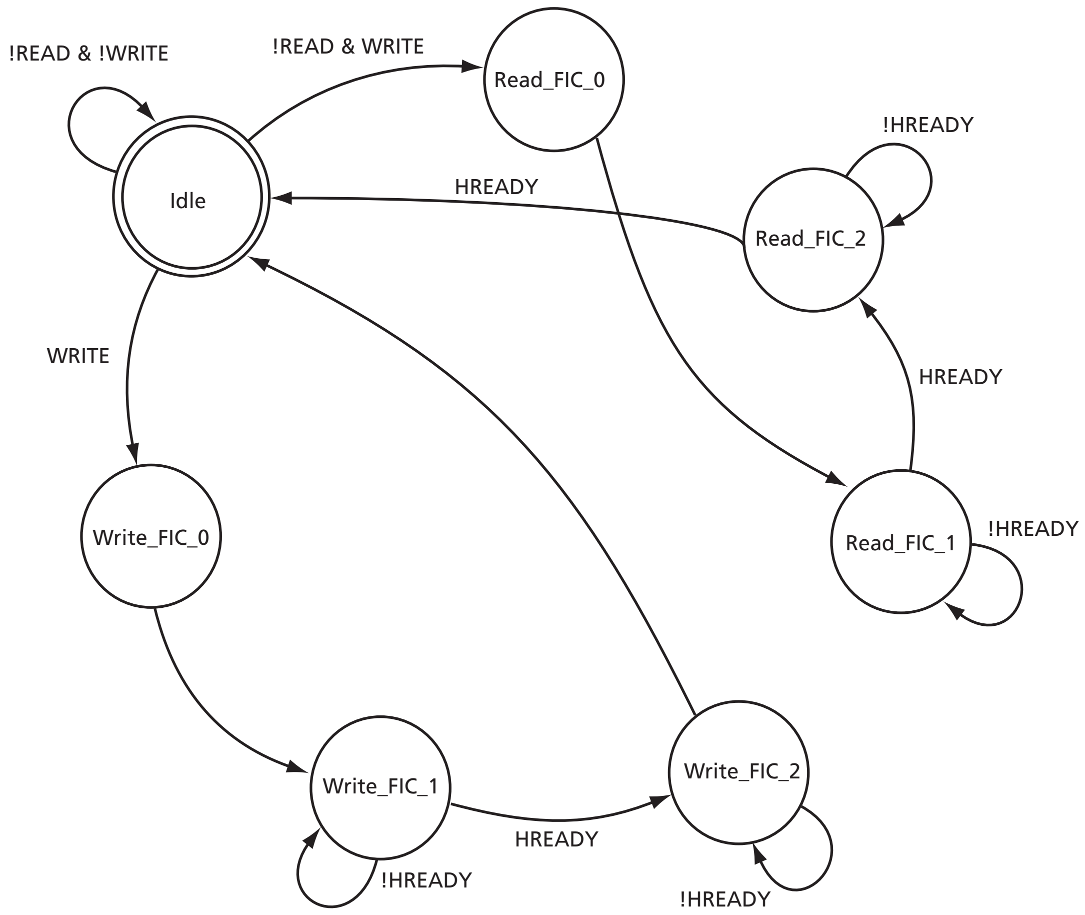


Figure 13 • **AHB-Lite Master State Diagram**

Sample Verilog Code for AHB-Lite Master

```
case (ahb_fsm_current_state)

   Idle: //0x00
      begin
         if ( WRITE == 1'b1)
            begin
            ahb_fsm_current_state       <=  Write_FIC_0;
            HADDR                       <=  ADDR;
            HADDR_int                   <=  ADDR;
            HWDATA_int                  <=  DATAIN;
            ahb_busy                    <=  1'b1;
            end
         else if ( READ  == 1'b1)
            begin
            ahb_fsm_current_state       <=  Read_FIC_0;
            HADDR                       <=  ADDR;
            HADDR_int                   <=  ADDR;
            ahb_busy                    <=  1'b1;
            end
         else
            begin
            ahb_fsm_current_state       <=  Idle;
            end
      end

   Write_FIC_0: //0x01  store the address+control signals and apply to coreahblite
      begin
         HTRANS                      <=  2'b10;
         HSIZE                       <=  HSIZE_int;
         HWRITE                      <=  1'b1;
         ahb_fsm_current_state       <=  Write_FIC_1;
         ahb_busy                    <=  1'b1;
      end

   Write_FIC_1: //0x02
      begin
         if ( HREADY  == 1'b0) //keep the address+control signals when slave is not ready
                              yet
            begin
            HTRANS                  <=  2'b10;
            HSIZE                   <=  HSIZE_int;
            HWRITE                  <=  1'b1;
            HADDR                   <=  HADDR_int;
            ahb_fsm_current_state   <=  Write_FIC_1;
            ahb_busy                <=  1'b1;
            end
         else  //send the data+go to next state, doesn't need to keep the address+other
                   controls active
            begin
            HWDATA                  <=  HWDATA_int;
            HADDR                   <=  32'h00000000;
            HTRANS                  <=  2'b00;
            HWRITE                  <=  1'b0;
            ahb_fsm_current_state   <=  Write_FIC_2;
            ahb_busy                <=  1'b1;
            end
      end
   Write_FIC_2: //0x03
      begin
         if ( HREADY   == 1'b0) //keep the data when slave is not ready yet
            begin
            ahb_fsm_current_state   <=  Write_FIC_2;
            ahb_busy                <=  1'b1;
            end
         else   //finish the write transfer
            begin
            ahb_fsm_current_state  <=  Idle;
            ahb_busy               <=  1'b0;
            end
      end

   Read_FIC_0: //0x04 store the address+control signals and apply to coreahblite
      begin
         HTRANS                  <=  2'b10;
         HSIZE                   <=  HSIZE_int;
         HWRITE                  <=  1'b0;
         ahb_fsm_current_state   <=  Read_FIC_1;
         ahb_busy                <=  1'b1;
      end
   Read_FIC_1: //0x05
      begin
         if ( HREADY  == 1'b1) // go to next state
            begin
            ahb_fsm_current_state    <=  Read_FIC_2;
            end
         else   //keep the address+control signals when slave is not ready yet
            begin
            HTRANS                   <=  2'b10;
            HSIZE                    <=  HSIZE_int;
            HWRITE                   <=  1'b0;
            HADDR                    <=  HADDR_int;
            ahb_fsm_current_state    <=  Read_FIC_1;
            ahb_busy                 <=  1'b1;
            end
      end
   Read_FIC_2: //0x06
      begin
         if ( HREADY  == 1'b1)          //read the data+finish the read transfer
            begin
            DATAOUT                  <=  HRDATA;
            ahb_fsm_current_state    <=  Idle;
            ahb_busy                 <=  1'b0;
            end
         else  //waiting slave to be ready
            begin
            ahb_fsm_current_state    <=  Read_FIC_2;
            ahb_busy                 <=  1'b1;
            end

            HADDR                <=  32'h00000000;  //doesn't need to keep the address +
                                                      other controls any more
            HTRANS                   <=  2'b00;

      end
endcase
```

Sample RTL for the custom AHB-Lite master is available for download on the Actel website: www.actel.com/download/rsc/?f=User_Logic_MSS_DF.

The sample RTL is in the AHB_master_fabric folder in the zip file.

Conclusion

SmartFusion contains a hard microcontroller subsystem (MSS), programmable analog circuitry, and FPGA fabric. The MSS consists of an ARM Cortex-M3 processor, advanced high-performance bus matrix, system registers, Ethernet MAC, several peripherals, DMA engine, real-time counter, embedded nonvolatile memory, embedded SRAM, and fabric interface controller (FIC). The FIC performs an AMBA bus bridging function between MSS and FPGA fabric. In order to connect the FPGA fabric to the MSS through the FIC, the AHB-Lite or APB3 interface must be added to custom user logic. This application note explained how to create the AHB-Lite or APB3 wrapper on custom logic. Instructions for setting up simulation and checking the custom logic using the bus functional model (BFM) flow are included as well.

Appendix A: Creating a Subsystem Design with a Custom APB3 or AHB-Lite to MSS

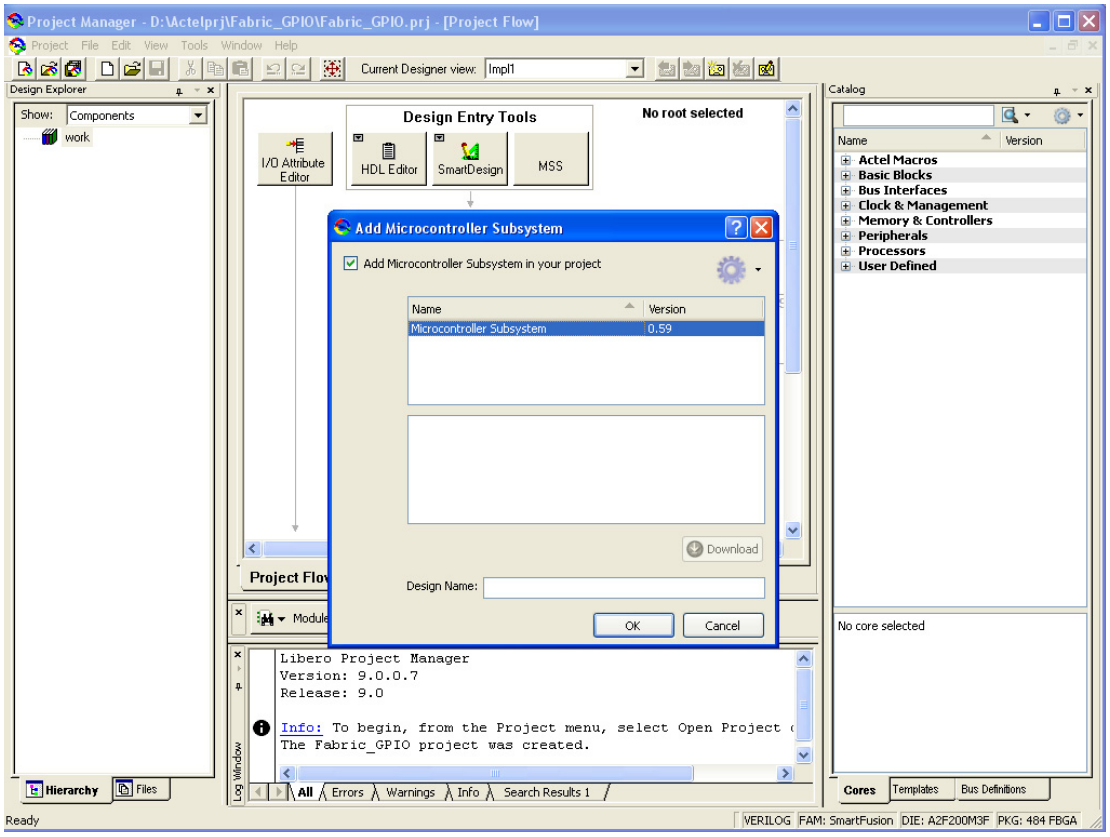
This appendix describes how to connect a custom APB3 slave or AHB-Lite slave interface to the MSS For additional information refer to:

http://coredocs.actel-ip.com/Actel/SmartFusionMSS/MSS_FIC/mss_fic_ext_apb3_ug_1.pdf

http://coredocs.actel-ip.com/Actel/SmartFusionMSS/MSS_FIC/mss_fic_ext_ahb_apb3_ug_1.pdf

Step 1

Once a Libero IDE project is created, the Add Microcontroller Subsystem setup dialog box will appear. After you provide the name and select **OK**, the Microcontroller Subsystem will open, as shown Figure 14.



☑ Add Microcontroller Subsystem in your project

Name	Version
Microcontroller Subsystem	0.59

Design Name:

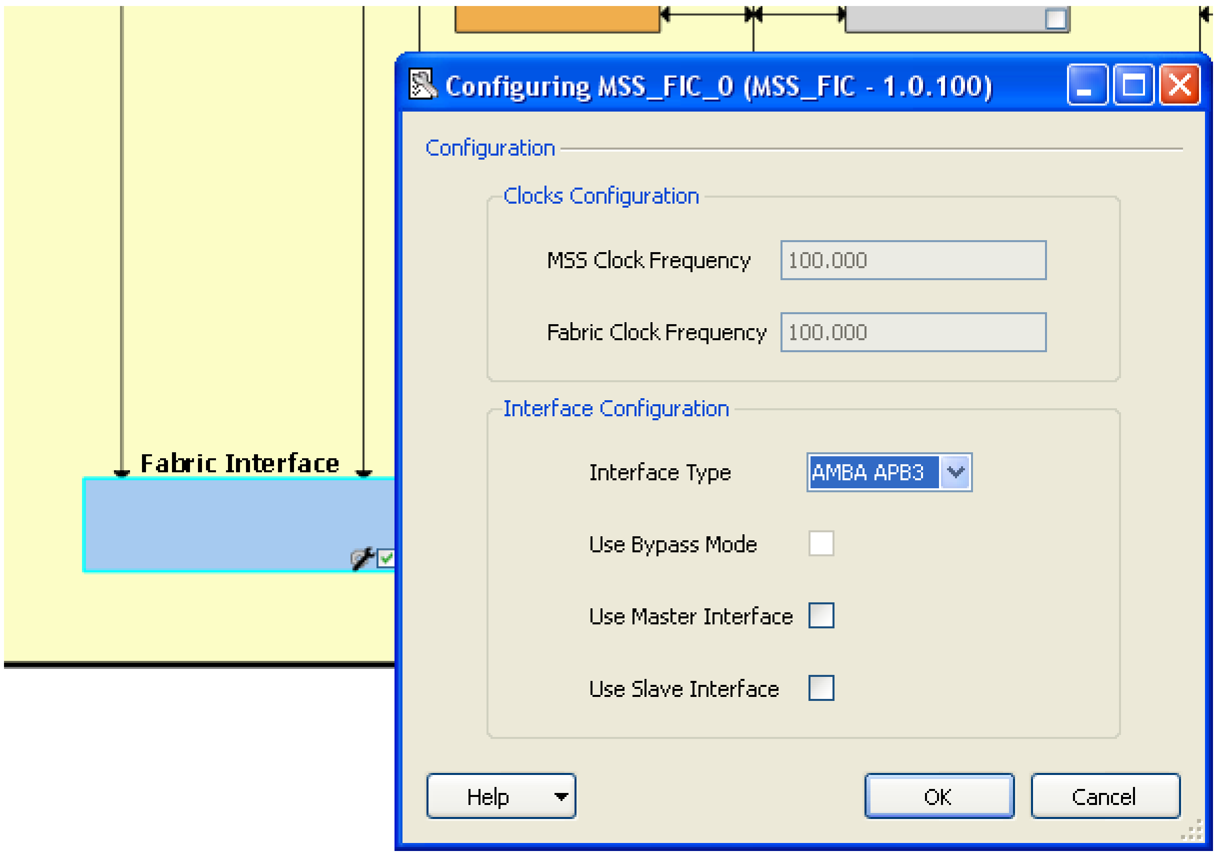
```
Libero Project Manager
Version: 9.0.0.7
Release: 9.0

Info: To begin, from the Project menu, select Open Project (
The Fabric_GPIO project was created.
```

Figure 14 • **Add Microcontroller Subsystem Dialog in New Project**

Step 2

Using the MSS configurator, double-click the **Fabric Interface** sub-configurator (Figure 15) and configure the fabric interface controller with an APB3 or AHB-Lite interface. For an APB3 slave, select **AMBA APB3** and for an AHB slave, select **AHB-Lite**. Also, select the master or slave as needed and then promote the interface to the top level.



MSS Clock Frequency 100.000

Fabric Clock Frequency 100.000

Interface Type AMBA APB3

Use Bypass Mode ☐

Use Master Interface ☐

Use Slave Interface ☐

Figure 15 • **Configuring the Fabric Interface Controller**

Double-click on the clock management block and configure the Clock management dialog as needed (Figure 16). Then enable and promote FAB_CLK.

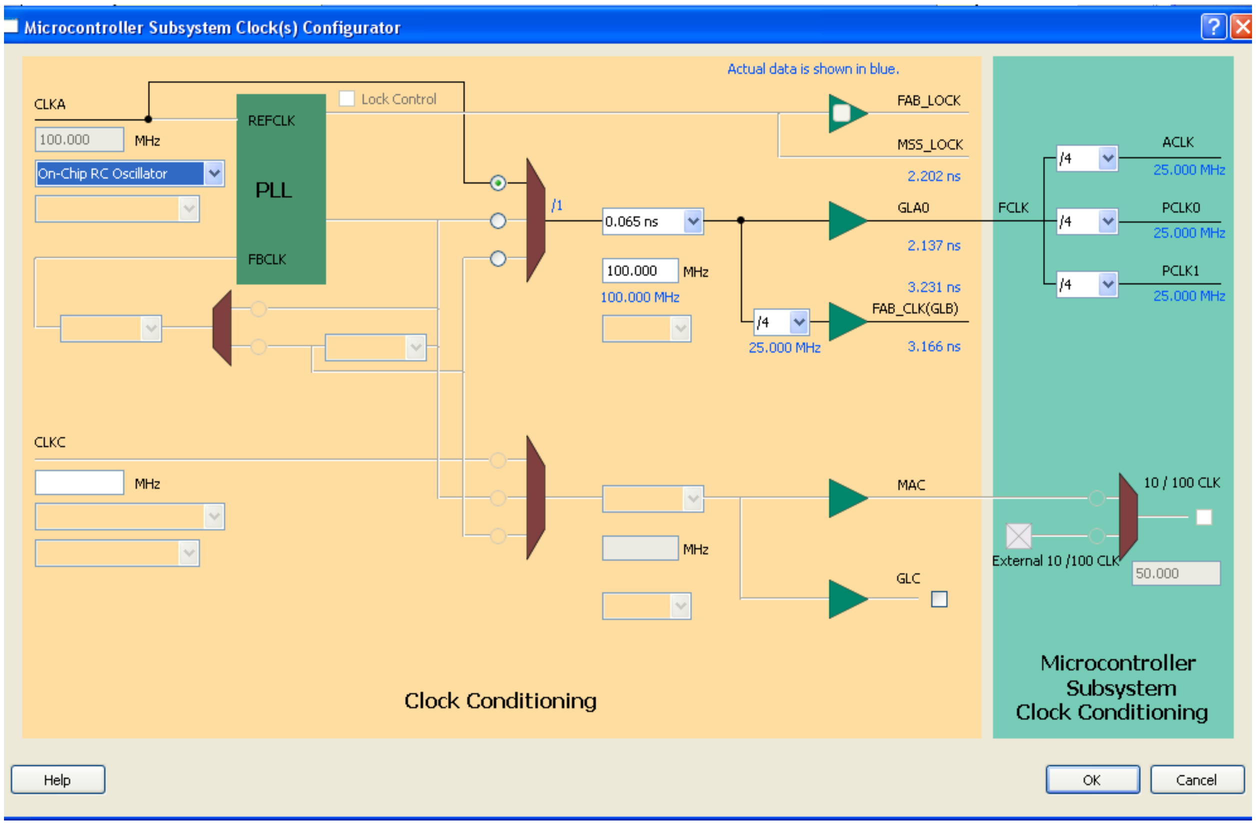


Figure 16 • **Configuring Clock Configurator**

Step 4

Enable M2F_RESET_N from the Reset Management block and promote it to the top level.

Step 5

Configure other MSS blocks as needed for your design and generate the MSS block.

Step 6

Create a new SmartDesign block. Add/create the RTL source code for the custom APB3/AHB-Lite block in Libero IDE.

Step 7

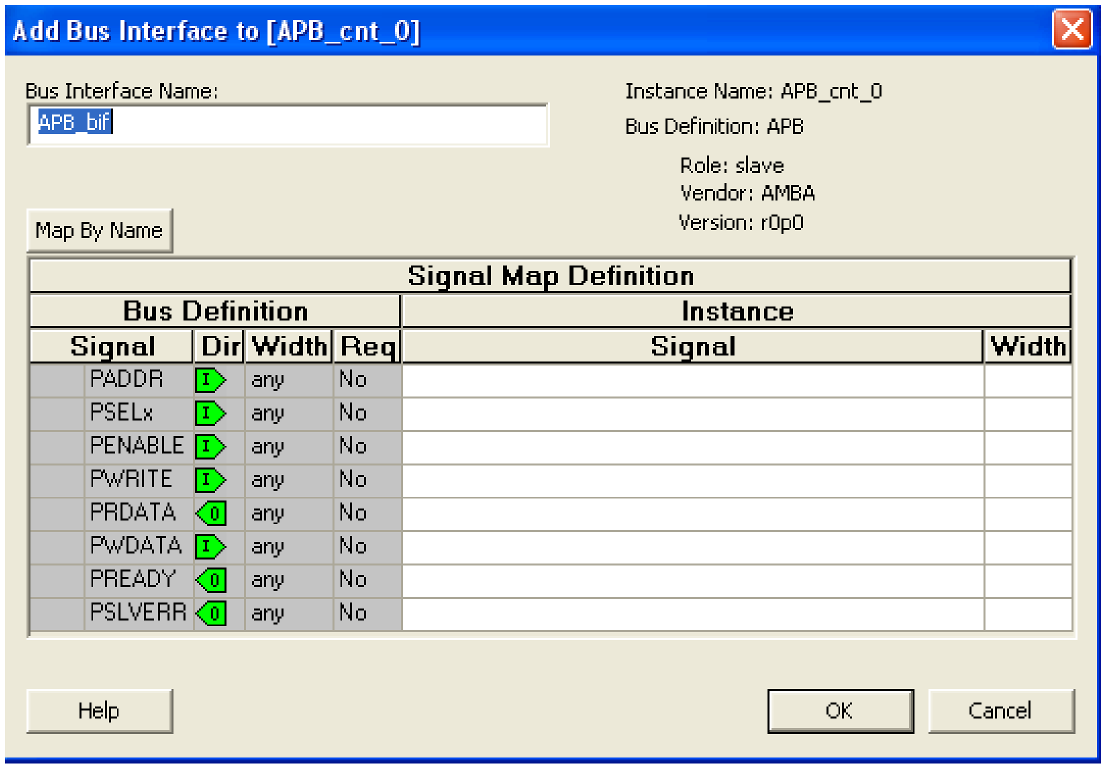
Select the RTL source code for the custom APB3/AHB-Lite wrapper and drag it3 onto the canvas. Similarly drag the MSS block onto the canvas. For the APB3 master example, add the APB3 master block, as shown in Figure 12 on page 12.

Step 8

Expand the Bus Interfaces section in the IP catalog. Select the appropriate bus interface and drag it onto the canvas. For an AHB-Lite slave, select **CoreAHBLite 3.0.103** and drag it onto the canvas. For an APB3 slave, select **CoreAPB3 2.1.101** and drag it onto the canvas. If you have both APB3 slave and AHB-Lite slave, select **CoreAHBLite 3.0.103**, **CoreAHB2APB3 2.0.116,** and **CoreAPB3 2.1.101** and drag them onto the canvas. Make sure to choose the right slot or slots for them when configuring CoreAHBLite or CoreAPB3.

Step 9

Add a bus interface to your APB/AHB wrapper instance. Select a bus definition from the Bus Definition tab in the Catalog and drag it onto your APB/AHB wrapper instance. The Add Bus Interface dialog box opens (Figure 17). Click **Map by Name** to map the signals automatically and the configurator will attempt to map any similar signal names between the bus definition and pin names on the instance. Map other signals manually that are not mapped with Map by Name. Once the Bus interface is added, the custom APB block would similar to shown in Figure 18.


Add Bus Interface to [APB_cnt_0]

Bus Interface Name: APB_bif

Instance Name: APB_cnt_0
Bus Definition: APB
Role: slave
Vendor: AMBA
Version: r0p0

Map By Name

Signal Map Definition

Bus Definition				Instance	
Signal	Dir	Width	Req	Signal	Width
PADDR	I	any	No		
PSELx	I	any	No		
PENABLE	I	any	No		
PWRITE	I	any	No		
PRDATA	O	any	No		
PWDATA	I	any	No		
PREADY	O	any	No		
PSLVERR	O	any	No		

Help | OK | Cancel

Figure 17 • **Add Bus Interface Dialog Box**

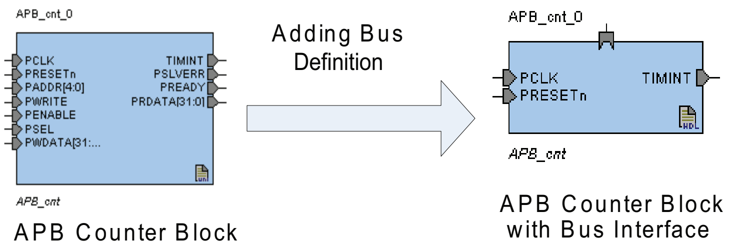


Figure 18 • **Adding Bus Interface to Customer APB Block**

Step 10

Connect all subsystem signals to the FIC bus interface(s) using the SmartDesign auto-connect feature or manually build the AMBA subsystem as shown in Figure 19.

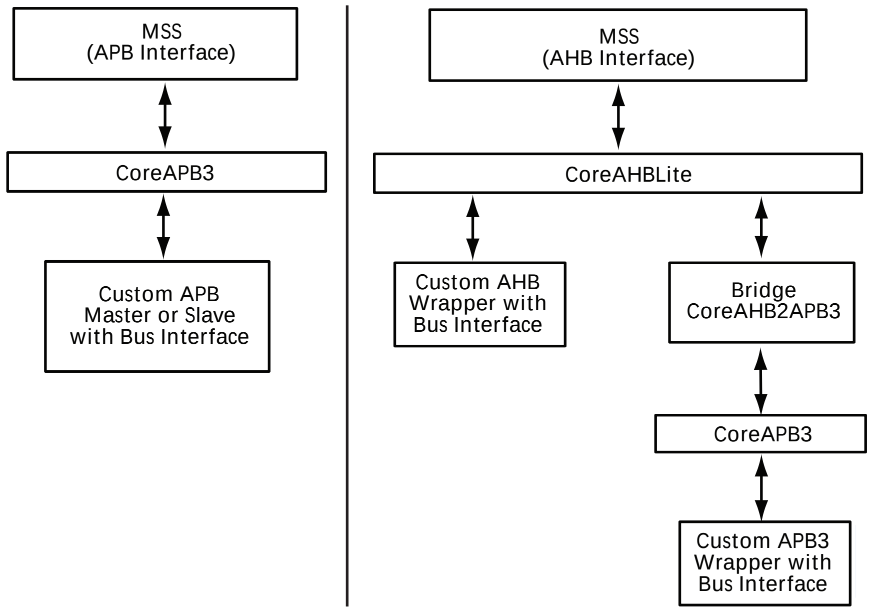

MSS
(APB Interface)
CoreAPB3
Custom APB Master or Slave with Bus Interface
MSS
(AHB Interface)
CoreAHBLite
Custom AHB Wrapper with Bus Interface
Bridge CoreAHB2APB3
CoreAPB3
Custom APB3 Wrapper with Bus Interface

Figure 19 • **Connection to MSS**

Step 11

Connect the FAB_CLK clock signal from the MSS block to the clock signals of the fabric AMBA subsystem.

Step 12

Connect the M2F_RESET_N signal from the MSS block to the reset signals of the fabric AMBA subsystem.

Step 13

Connect other signals present on the fabric AMBA subsystem as needed. Then generate the SmartDesign block.

Appendix B: Simulating the User Logic Block

This section describes how to simulate your user logic block with a bus functional model (BFM). Three BFM files are generated when the top-level SmartDesign block is created: Test.bfm, User.bfm, and Subsystem.bfm. You need customize the User.bfm file to emulate Cortex-M3 transactions in your system. This file contains an include command to subsystem.bfm that needs to be untenanted 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 definitions inside this *.bfm file.

The following section describes the simulation setup using a custom APB interface. You can follow the same for a user AHB interface.

Step 1

Open the User.bfm script file from the Simulation folder. Add or modify the lines shown in bold font below to User.bfm. This BFM exercises only the register block. You need to add BFM for the other blocks in order to see the simulation activity on those blocks.

```
#============================================================
# 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;
#
#============================================================

int i;
int j;
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:
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
```

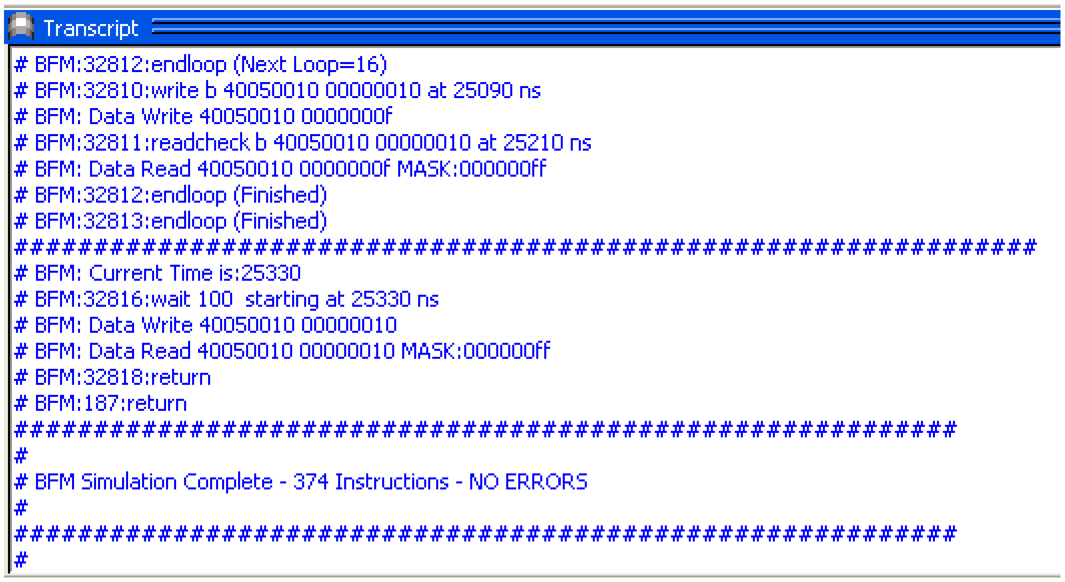
```
header "Testing reg_apb_wrp_0 0x40050004 address in loop mode";
loop i 0 16
write    b   reg_apb_wrp_0    0x04 i;
readcheck b reg_apb_wrp_0    00x04 i; # Expect value i
endloop

header " Current Time is:%0d", $TIME;
waitns 50;
stop 1;
return
```

Please note that the BFM test script will be over written if the core is regenerated. You can copy and paste the edits in user_test.bfm to avoid having to make the edits each time the core is generated.

Step 2

Setup the simulation setting and click the Simulation button in the Libero Design Flow window to run pre-synthesis simulation. Run simulation until all the BFM commands are executed (Figure 20).



```
# BFM:32812:endloop (Next Loop=16)
# BFM:32810:write b 40050010 00000010 at 25090 ns
# BFM: Data Write 40050010 0000000f
# BFM:32811:readcheck b 40050010 00000010 at 25210 ns
# BFM: Data Read 40050010 0000000f MASK:000000ff
# BFM:32812:endloop (Finished)
# BFM:32813:endloop (Finished)
#######################################################################
# BFM: Current Time is:25330
# BFM:32816:wait 100  starting at 25330 ns
# BFM: Data Write 40050010 00000010
# BFM: Data Read 40050010 00000010 MASK:000000ff
# BFM:32818:return
# BFM:187:return
################################################################
#
# BFM Simulation Complete - 374 Instructions - NO ERRORS
#
################################################################
#
```

Figure 20 • **ModelSim® Transcript Window**

You need to add the proper signal in the waveform window to see the simulation activity. Also, note that you can type **bfmtovec** in the ModelSim Transcript window at the VSIM prompt to get information related to any errors in the BFM script.

List of Changes

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

Previous Version	Changes in Current Version* (51900201-0/6.10)	Page
51900201-1/2.10	Links to other SmartFusion documents were corrected and some terminology changes made; AMBA-Lite was changed to AMBA, APB V3 was changed to APB3, and AHB-L was changed to AHB-Lite.	N/A
	The "AMBA Interface in the Fabric Interface Controller (FIC)" section was revised to exclude 16-bit APB as a type of interface to the fabric.	4
	"Appendix A: Creating a Subsystem Design with a Custom APB3 or AHB-Lite to MSS" was revised.	16
	"Appendix B: Simulating the User Logic Block", formerly Appendix C, was revised.	21
	"Creating a Subsystem Design with a Custom APB Master (Core8051s)," formerly Appendix B, was removed.	N/A

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.*

Actel is the leader in low power FPGAs and mixed signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at www.actel.com.

Actel Corporation
2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.
River Court,Meadows Business Park
Station Approach, Blackwater
Camberley Surrey GU17 9AB
United Kingdom
Phone +44 (0) 1276 609 300
Fax +44 (0) 1276 607 540

Actel Japan
EXOS Ebisu Buillding 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668
http://jp.actel.com

Actel Hong Kong
Room 2107, China Resources Building
26 Harbour Road
Wanchai, Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488
www.actel.com.cn

51900201-0/6.10