Adding Custom Peripherals to the AMBA Host and Peripheral Buses

Introduction

The Actel CoreMP7 microprocessor is a soft-core implementation of the industry-standard ARM7TDMI-S™ and is optimized for maximum speed and minimum size in Actel flash-based FPGAs. The combination of the ARM7TDMI-S microprocessor and FPGA enables designers to quickly implement ARM7-based systems while also adding new features to existing systems, without the high cost of an ASIC.

Overview

CoreConsole, the Actel IP Deployment Platform, gives designers the ability to easily construct a system around CoreMP7. CoreConsole is used in conjunction with the Actel IP vault to rapidly assemble a processor-based system. The microprocessor's peripherals are selected from the IP vault and stitched to the system bus within the CoreConsole environment. The resulting processor-based system is imported into the Libero® Integrated Development Environment (IDE), where the designer can include additional functionality. Additionally, CoreConsole provides the flexibility to add custom peripherals to either the AMBA High-performance Bus (AHB) or the AMBA Peripheral Bus (APB). This application note focuses on the procedures required to add custom APB peripherals.

Design Creation

Creating a CoreMP7 Design within CoreConsole

The CoreConsole IP Deployment Platform tool is a standalone microprocessor system builder that allows the designer to choose from among CoreMP7 variants, along with the peripherals connected to the AHB and APB interfaces. Once the CoreConsole design has been imported into Libero IDE, the remaining unused logic can be utilized for implementing additional processor peripherals and other desired digital functions.

Add Existing Components from CoreConsole

Start by launching the CoreConsole tool and using the existing library components available in the IP vault to build your CoreMP7-based system (see the *CoreConsole Users Guide* for details on the tool's operation). The system constructed in this example (Figure 1 on page 2), is comprised of the following components:

- CoreMP7/CoreMP7Bridge – ARM7TDMI-S/ARM native bus to AHB master
- CoreAHBLite – Single master AMBA High-performance bus interface
- CoreMemCtrl – Memory controller (AHB peripheral)
- CoreAHB2APB – AHB-to-APB bridge (AHB slave, APB master peripheral)
- CoreAPB – AMBA peripheral bus interface
- CoreGPIO – General purpose I/O interface (APB peripheral)
- CoreUARTapb – Universal Asynchronous Receiver Transmitter interface (APB peripheral)

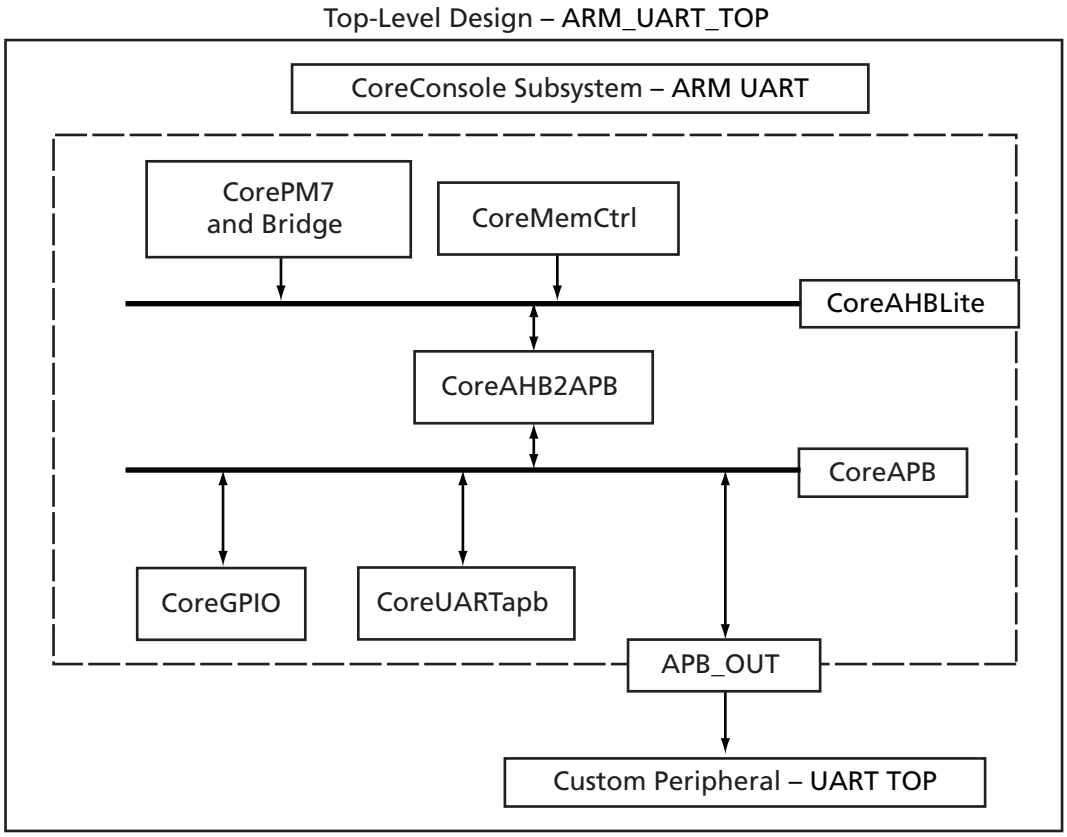

Figure 1 • **Processor-Based Subsystem Block Diagram**

CoreConsole automatically stitches the internal signals of the user-selected components to the AHB and APB. Only signals that connect to (or from) the top-level to the processor-based subsystem and the desired processor/peripheral interrupt must be manually stitched. Any custom peripherals that are not in the IP vault must be manually added. CoreConsole allows connection of the APB to the top level of the processor-based subsystem, where it can be manually connected to other APB peripherals.

Add a Custom Advanced Peripheral Bus (APB) Peripheral

For this example, a size-optimized UART is added to the APB. Figure 1 shows the completed block diagram.

Provisions for Adding Custom Peripherals

The designer must first select an unused APB slave address for the custom peripheral and connect the APB and slave select to the subsystem top level.

Using the **Configuring Connection** dialog box shown in Figure 2, the APB signals are connected to the top level of the processor-based subsystem. In this example, APB slave address 7 is connected to the top level and given the name is APB_OUT.

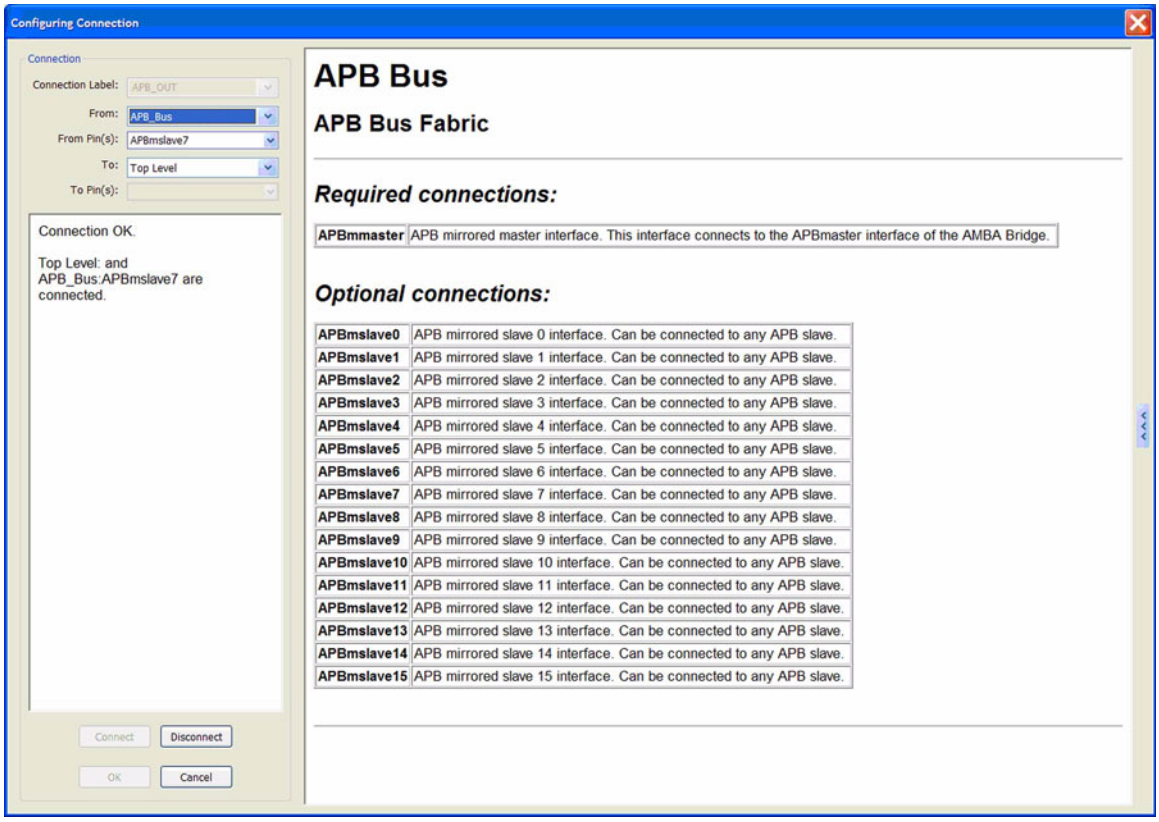

Figure 2 • **CoreConsole – Connection Configuration Dialog Box**

Figure 3 shows the completed subsystem. The APB signals interfacing with the custom APB peripheral are included in the bussed signal named APB_OUT, which is found on the top level. The IP vault containing the remaining components can be seen in the pane on the left hand side of the window, shown in Figure 3.

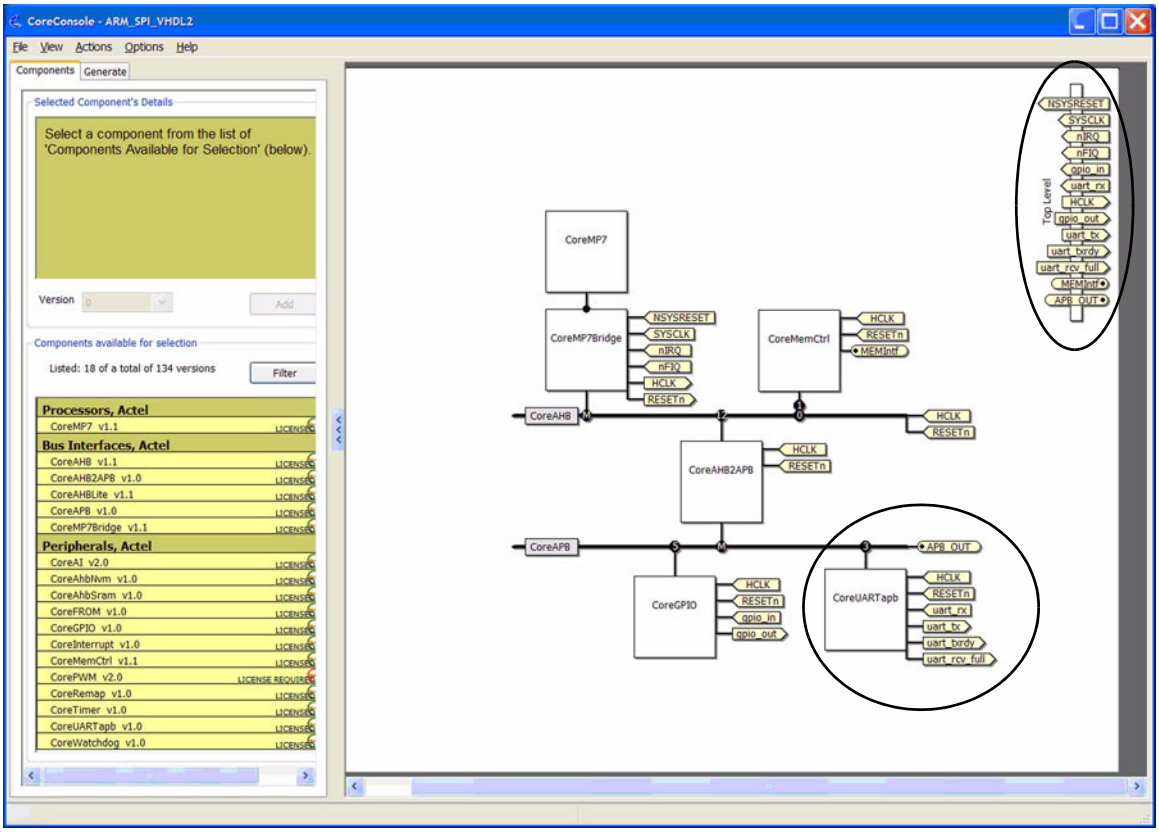

Figure 3 • **Completed Subsystem**

Once stitching has been completed, generation of the processor-based subsystem can begin for importation into Libero IDE. Selecting the **Generate** tab in the CoreConsole window and clicking the **Save & Generate** button starts this process. CoreConsole can also generate a memory map of the system. To generate the memory map, select **View** on the menu bar and then choose **Memory Map**. The memory map includes the base address and individual register information for each peripheral that was added from the component IP vault. Figure 4 on page 5 shows the memory map for the example subsystem. The base addresses of the CoreUARTapb and CoreGPIO components are shown, as well as the specific register information for CoreUARTapb. Note custom peripherals that will be outside of CoreConsole are not reported in the memory map, as CoreConsole is not aware of the components specifics. This information must be managed by the designer.

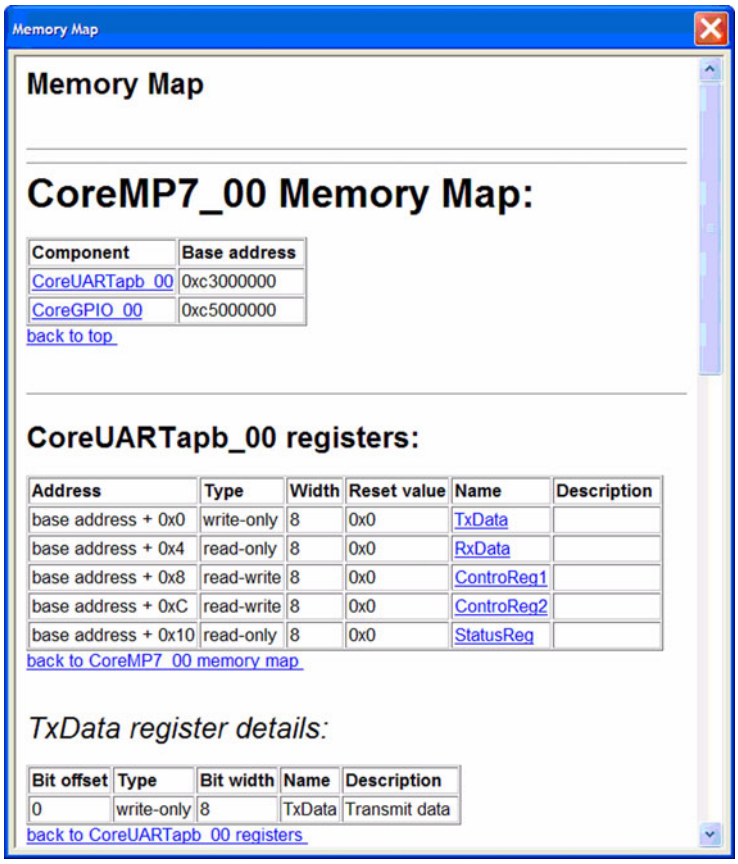

Figure 4 • **Memory Map**

Import the Processor-Based Subsystem into Libero IDE

Next, import the generated CoreConsole design containing the files for synthesis and simulation into Libero IDE. Import the CoreConsole_DesignName.ccp file located in *<CoreConsole Installation Directory>\LiberoExport\DesignName*. Figure 5 on page 6 shows the design files that are imported into Libero IDE. The top level of the imported design is ARM_UART, with the CoreConsole components instantiated as sub-blocks of the top level. Also visible in this view is the overall top level of the design, ARM_UART_TOP, which instantiates the CoreMP7-based subsystem from CoreConsole and the custom peripheral, UART_TOP.

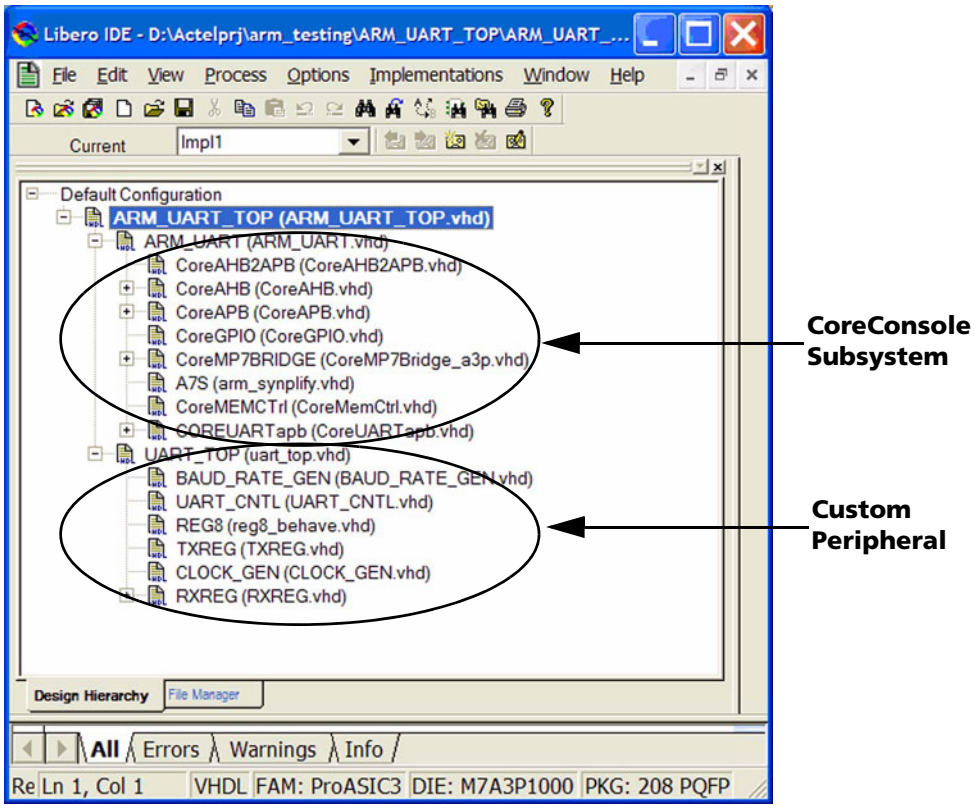

Figure 5 • **Design Hierarchy in Libero IDE**

After the CoreConsole design is imported into Libero IDE, the top-level ports of the CoreConsole subsystem are accessible at the top level of the imported CoreConsole design. The entity and port definitions for this example system are shown below. The signals containing the APB_OUT prefix are the APB signals connected to the top level for connection to the custom peripheral.

```
entity ARM_UART  is
    -- Port list
    port(
        APB_OUT_PADDR : out std_logic_vector(23 downto 0);
        APB_OUT_PENABLE : out std_logic;
        APB_OUT_PRDATA : in std_logic_vector(31 downto 0);
        APB_OUT_PSELx : out std_logic;
        APB_OUT_PWDATA : out std_logic_vector(31 downto 0);
        APB_OUT_PWRITE : out std_logic;
        MEMIntf_FlashCSN : out std_logic;
        MEMIntf_FlashOEnN : out std_logic;
        MEMIntf_FlashWEnN : out std_logic;
        MEMIntf_MemAddr : out std_logic_vector(27 downto 0);
        MEMIntf_MemDataIn : in std_logic_vector(31 downto 0);
        MEMIntf_MemDataOEnN : out std_logic;
        MEMIntf_MemDataOut : out std_logic_vector(31 downto 0);
```

```
        MEMIntf_MemReadN : out std_logic;
        MEMIntf_MemWriteN : out std_logic;
        MEMIntf_SramByte0N : out std_logic;
        MEMIntf_SramByte1N : out std_logic;
        MEMIntf_SramByte2N : out std_logic;
        MEMIntf_SramByte3N : out std_logic;
        MEMIntf_SramCSN : out std_logic;
        MEMIntf_SramOEnN : out std_logic;
        MEMIntf_SramWEnN : out std_logic;
        NSYSRESET : in std_logic;
        SYSCLK : out std_logic;
        gpio_in : in std_logic_vector(31 downto 0);
        gpio_out : out std_logic_vector(31 downto 0);
        nFIQ : in std_logic;
        nIRQ : in std_logic;
        uart_rcv_full : out std_logic;
        uart_rx : in std_logic;
        uart_tx : out std_logic;
        uart_txrdy : out std_logic
    );
end ARM_UART;
```

Attach the Custom Peripheral to the CoreConsole Subsystem

Connections to Peripheral

The block diagram showing the connections (marked with an asterisk) between the custom peripheral and the CoreMP7-based subsystem is shown in Figure 6. Table 1 lists the descriptions of the signals.

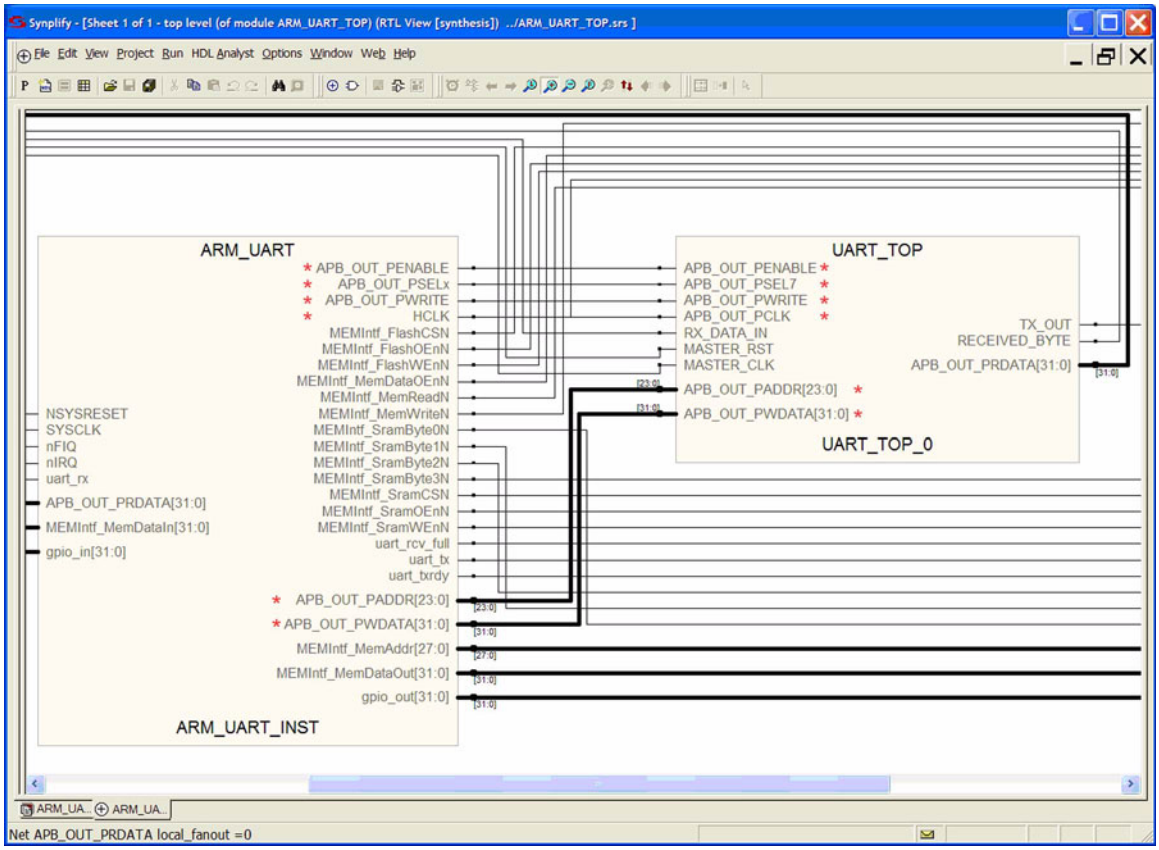

Figure 6 • **Signal Connections Between the CoreMP7-Based Subsystem and the Custom Peripheral**

Table 1 • **Signal Descriptions in CoreMP7-Based Subsystem**

Signal Name	Description
APB_OUT_PWRITE	Active high write / active low read signal for APB slaves. Connect this signal to the enable input of all APB peripheral slaves.
APB_OUT_PENABLE	Active high data valid strobe for all slaves. Connect this signal to the enable input of all APB peripheral slaves.
APB_OUT_PSELx	Individual active high select signal for each slave to indicate host access. In this example, the internal select signal is labeled APB_OUT_PSELx and is connected to the APB slave peripheral in slot 7.
APB_OUT_PADDR[23:0]	APB Slave address. Connect to all APB slave peripheral address inputs.
APB_OUT_PWDATA[31:0]	Write data output of the APB fabric; becomes write input to peripheral. Connect this signal to the data input of all APB peripheral slaves.
APB_OUT_PRDATA[31:0]	Peripheral data output, which becomes the input to the APB Bus block. Connect this signal to the read output of all APB peripheral slaves.
HCLK	Internal subsystem clock. Connect this to the clock input of all APB peripheral slaves.

Design Verification

Simulation

After the top-level design is created and all custom peripherals have been added to the processor-based subsystem, the next step is simulation. Simulating a design containing a CoreConsole subsystem consists of verifying the register interface between the processor core and attached peripherals. This is accomplished by using the CoreConsole-generated testbench along with the following modifications.

If the CoreMP7-based subsystem is instantiated as a hierarchical block in a larger design, the CoreConsole-generated testbench, located in the Libero IDE project CoreConsole directory, must be copied to the stimulus directory of the top-level project. Figure 7 on page 12 highlights the original testbench location and the copied location in the project's hierarchy.

The file, *testbench.vhd*, is located in the following directory:

<Libero Project>\coreconsole\coreconsole_project_name.

The following changes need to be made to the original testbench so the simulation will run correctly:

1. The CoreMP7 top-level entity description in the testbench must be replaced by the new top-level entity description. The "Original Testbench" section and the "Modified Testbench" section on page 11 show the entity descriptions from the original and modified testbenches. The differences are as follows:
   - Component name change from processor subsystem to the top-level design name
   - Removal of APB bus signals from the top-level interface
   - Addition of the clock and serial transmit and receive data for the custom peripheral
2. The component port mapping must also be modified to reflect the changes to the top level.

Original Testbench

```
-- Component to test
    component ARM_UART
        -- Port list
        port(
            -- Inputs
            APB_OUT_PRDATA: in  std_logic_vector(31 downto 0);
            NSYSRESET : in  std_logic;
            SYSCLK : in  std_logic;
            uart_rx : in  std_logic;
            -- Outputs
            APB_OUT_PADDR : out std_logic_vector(23 downto 0);
            APB_OUT_PENABLE : out std_logic;
            APB_OUT_PSELx : out std_logic;
            APB_OUT_PWDATA: out std_logic_vector(31 downto 0);
            APB_OUT_PWRITE : out std_logic;
            HCLK : out std_logic;
            -- Memory signals
            MEMIntf_FlashCSN : out std_logic;
            MEMIntf_FlashOEnN : out std_logic;
            MEMIntf_FlashWEnN: out std_logic;
            MEMIntf_MemAddr : out std_logic_vector(27 downto 0);
            MEMIntf_MemDataIn : in  std_logic_vector(31 downto 0);
```

```
            MEMIntf_MemDataOEnN : out std_logic;
            MEMIntf_MemDataOut : out std_logic_vector(31 downto 0);
            MEMIntf_MemReadN : out std_logic;
            MEMIntf_MemWriteN: out std_logic;
            MEMIntf_SramByte0N : out std_logic;
            MEMIntf_SramByte1N : out std_logic;
            MEMIntf_SramByte2N : out std_logic;
            MEMIntf_SramByte3N : out std_logic;
            MEMIntf_SramCSN : out std_logic;
            MEMIntf_SramOEnN : out std_logic;
            MEMIntf_SramWEnN : out std_logic;
--GPIO signals
            gpio_in : in  std_logic_vector(31 downto 0);
            gpio_out : out std_logic_vector(31 downto 0);
           -- Interrupts
            Nfiq: in  std_logic;
            nIRQ : in  std_logic;
           -- UART signals
            uart_rcv_full : out std_logic;
            uart_tx : out std_logic;
            uart_txrdy : out std_logic
        );
    end component;
```

Modified Testbench

```
-- Component to test
    component ARM_UART_TOP
        -- Port list
        port(
                NSYSRESET           : in  std_logic;
                SYSCLK              : in  std_logic;
      HCLK               : out std_logic;
     -- memory signals
         MEMIntf_FlashCSN: out std_logic;
     MEMIntf_FlashOEnN    : out std_logic;
        MEMIntf_FlashWEnN     : out std_logic;
          MEMIntf_MemAddr       : out std_logic_vector(27 downto 0);
     MEMIntf_MemDataIn     : in  std_logic_vector(31 downto 0);
      MEMIntf_MemDataOEnN : out std_logic;
         MEMIntf_MemDataOut: out std_logic_vector(31 downto 0);
         MEMIntf_MemReadN      : out std_logic;
         MEMIntf_MemWriteN     : out std_logic;
         MEMIntf_SramByte0N    : out std_logic;
         MEMIntf_SramByte1N    : out std_logic;
         MEMIntf_SramByte2N    : out std_logic;
        MEMIntf_SramByte3N    : out std_logic;
         MEMIntf_SramCSN       : out std_logic;
        MEMIntf_SramOEnN      : out std_logic;
         MEMIntf_SramWEnN      : out std_logic;
         -- GPIO signals
         gpio_in               : in  std_logic_vector(31 downto 0);
         gpio_out              : out std_logic_vector(31 downto 0);
         -- Interrupts
         nFIQ                  : in  std_logic;
         nIRQ                  : in  std_logic;
         -- UART signals
         uart_rcv_full         : out std_logic;
         uart_rx               : in  std_logic;
         uart_tx               : out std_logic;
         uart_txrdy            : out std_logic;
                -- tiny UART signals
                MASTER_CLK          : in  std_logic; -- global clock signal
                RX_DATA_IN          : in  std_logic; -- serial receive input
                TX_OUT              : out std_logic  -- serial data out
        );
    end component;
```

In the default testbench, the only signals displayed in the wave window by default are NSYSRESET and SYSCLK. To add additional signals, the testbench could be modified further to accommodate additional signals or the designer could navigate through the hierarchy within Model*Sim*® and add the desired signals.

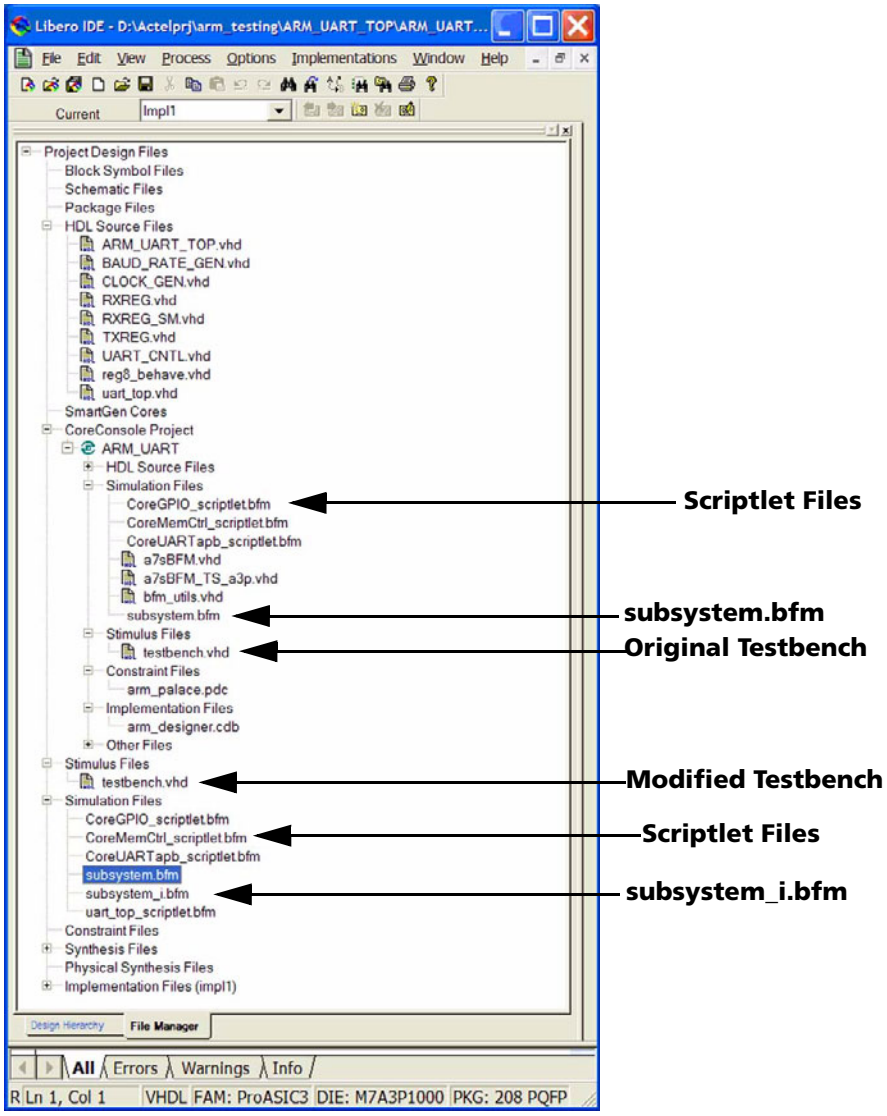

Figure 7 • **Libero IDE File Manager View**

Bus Functional Simulation Model

The Bus Functional Model (BFM) is a cycle-accurate simulation model of the CoreMP7 ARM7TDMI-S processor core being implemented. A BFM is generated from the CoreConsole tool, which also generates generic test scriptlets to correctly model and test the peripheral components selected within CoreConsole (excluding the manually added custom peripherals). The Bus Functional simulation file structure is composed of the BFM, *subsystem.bfm*, and individual test scriptlets that are associated with each peripheral component. The naming convention for the test scriptlets is *peripheral_name_scriptlet.bfm*. Each time Model*Sim* is invoked for a new simulation, the *subsystem_i.bfm* file is automatically created from the peripheral scriptlets. For this reason, edits should not be made to the *subsystem_i.bfm* file, as they will be overwritten. The *subsystem_i.bfm* file acts as the top-level script file, managing the memory map of the peripherals and including each of the peripheral scriptlets to be executed. Each of the scriptlet

files contain read and write commands to be performed on the peripheral under test. When the generic scriptlets are created, they consist of sample register accesses and are not exhaustive. Editing the peripheral scriptlet files is necessary to exhaustively test the CoreMP7-based subsystem and to mimic external events that may drive the embedded microprocessor. To include a custom peripheral, the following changes are required:

1. The *subsystem.bfm* file must be modified to include the memory map of the custom peripheral and its corresponding scriptlet file.
2. The custom peripheral's scriptlet file must be created to include commands to verify the CoreMP7 peripheral interface and any other tests that may be required.

The easiest way to create a scriptlet for the custom peripheral is to navigate to the *<Libero Project>\simulation* directory, copy and rename one of the existing scriptlet files, and then modify its contents. The code below represents a portion of the *subsystem.bfm* file that was generated by CoreConsole. The bold italic lines represent the modifications required to add the custom peripheral into the simulation. The specific instance of a peripheral is mapped to a base address and then linked to its component. This format allows for multiple instantiations of the same component to be memory mapped.

```
#------------------------------------------------------------
# Memory Map
# Define name and base address of each resource.
# memmapresource_namebase address
#------------------------------------------------------------

memmap CoreUARTapb_000xc3000000;
memmap CoreGPIO_00       0xc5000000;
memmap UART_TOP_0 0xc7000000;

#------------------------------------------------------------
# Include resource scriptlets
# includecomponent_nameinstance_name
#------------------------------------------------------------

include CoreUARTapb CoreUARTapb_00;
include CoreGPIO        CoreGPIO_00;
include UART_TOP uart_top_0;
```

Scriptlet Files

For this example, a test scriptlet was created to test the register interface to the custom UART peripheral. The register definition of the UART is given in Table 2:

Table 2 • **Register Definition of the UART**

Register	Offset
Control	0
Transmit	4
Receive	8
Status	C

The scriptlet consists of read and write operations to the peripheral's control register to test connectivity. The UART is then configured for loopback mode, data is written to the transmit register, the status register is polled for a received data indication, after which the receive register is read and compared against the expected transmitted value. The script is as follows:

```
Function     width    resource        offset   data
readcheck    b        VAR_resource    0x00     0x00;    # Expect value 00
write        b        VAR_resource    0x00     0x05;
readcheck    b        VAR_resource    0x00     0x05;    # Expect value 05
write        b        VAR_resource    0x00     0x01;
readcheck    b        VAR_resource    0x00     0x01;    # Expect value 01
write        b        VAR_resource    0x00     0x02;
readcheck    b        VAR_resource    0x00     0x02;    # Expect value 02
write        b        VAR_resource    0x00     0x04;
readcheck    b        VAR_resource    0x00     0x04;    # Expect value 04
write        b        VAR_resource    0x04     0x05;    # Write 0x05 to transmit data
                                                        register
write        b        VAR_resource    0x00     0x05;    # Write 0x05 to control
                                                        register, enable UART and loop
                                                        back mode write
write        b        VAR_resource    0x00     0x0D;    # Write 0x0d to control
                                                        register, initial settings,
                                                        enable write
wait 10;                                                #  Wait 10 clock cycles
write        b        VAR_resource    0x00     0x05;    # Write 0x0d to control
                                                        register, initial settings,
                                                        disable write indication
poll         b        VAR_resource    0x0c     0x01;    # Poll status register
readcheck    b        VAR_resource    0x08     0x05;    # Read receive register, expect
                                                        05 to match transmit data
```

Simulation

Once all modifications to the testbench and *subsystem.bfm* are completed and the scriptlet created, the design can be simulated. Invoking Model*Sim* from within Libero IDE compiles and executes all the necessary files. The waveform in Figure 8 on page 15 shows the activity on the APB that corresponds to the custom scriptlet created in the previous section, up to and including the polling activity. The UART is being accessed when the apb_out_psel7 signal is high. Figure 9 on page 15 shows the end of the polling activity and completion with a successful read of the UART receive register. These UART accesses, along with the accesses to the other peripheral components in the design, are also displayed in the Model*Sim* transcript window, shown in Figure 10 on page 16.

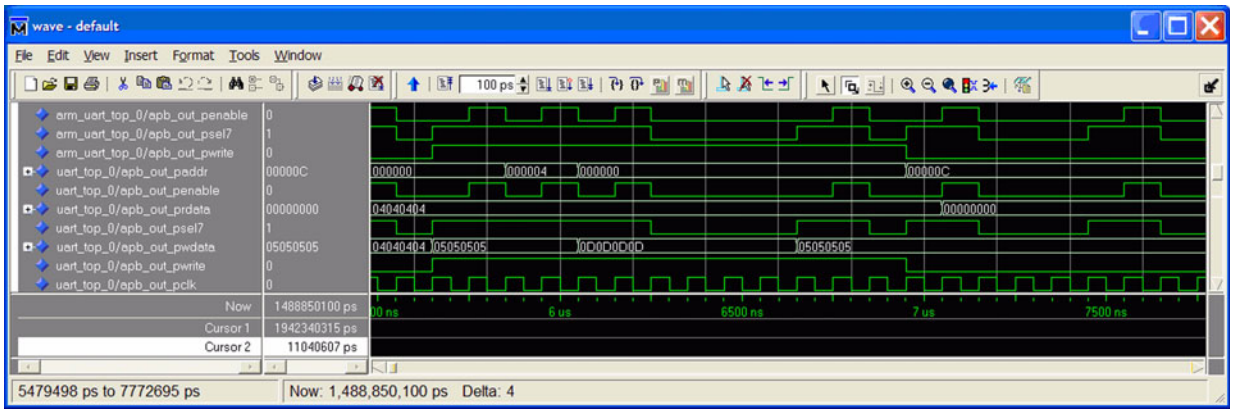

Figure 8 • **UART Register Initialization**

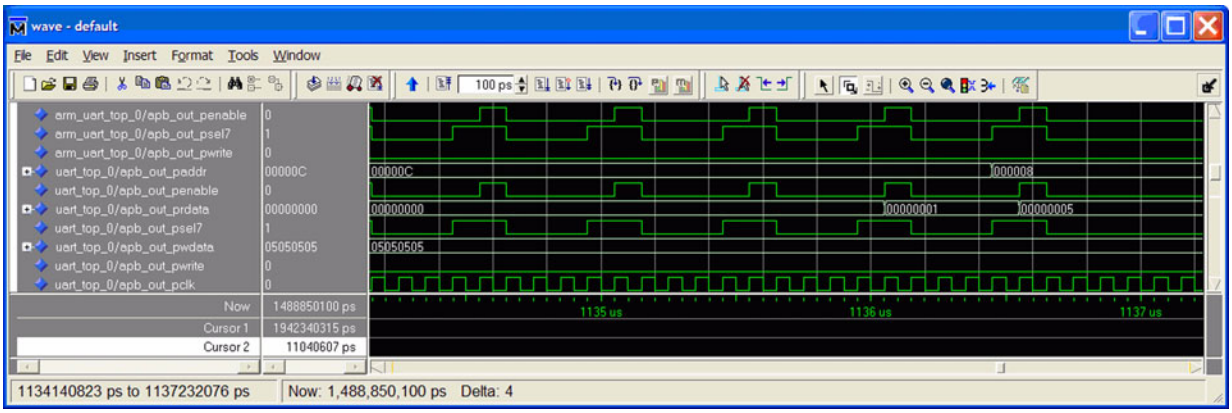

Figure 9 • **UART Receive Register Read**

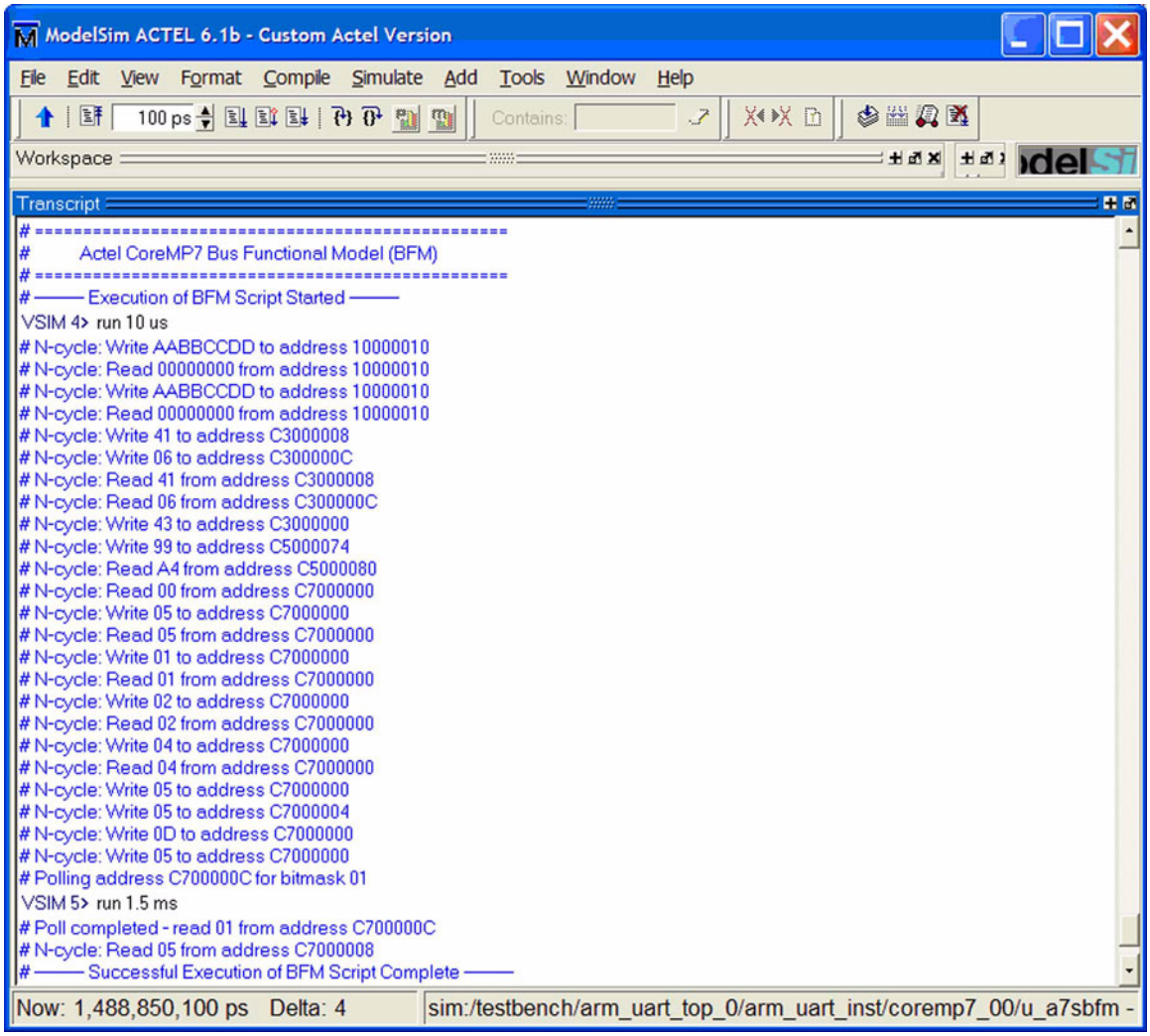

Figure 10 • **Model*Sim* Transcript Display**

Conclusion

Using CoreConsole in conjunction with Libero IDE, a CoreMP7-based design can be constructed quickly and easily. The techniques described in this document allow peripherals currently not present in the CoreConsole IP Vault to quickly and easily be added to an existing CoreMP7 system and their verification through simulation.

Related Documents

User's Guide

CoreConsole Users Guide

http://www.actel.com/documents/CoreConsole_ug.pdf

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 Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668
www.jp.actel.com

Actel Hong Kong
Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488
www.actel.com.cn

51900162-0/6.07