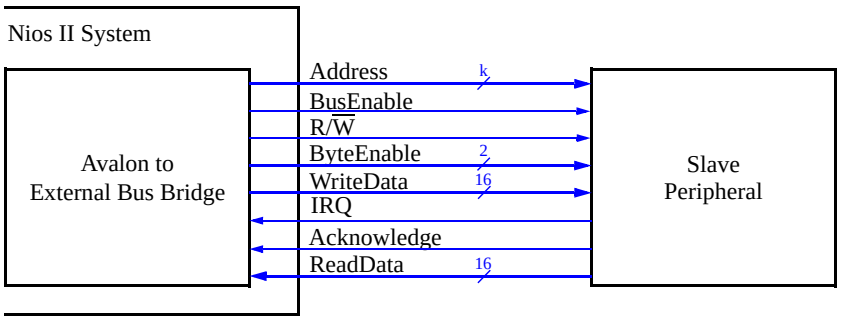
Laboratory Exercise 10

Bus Communication

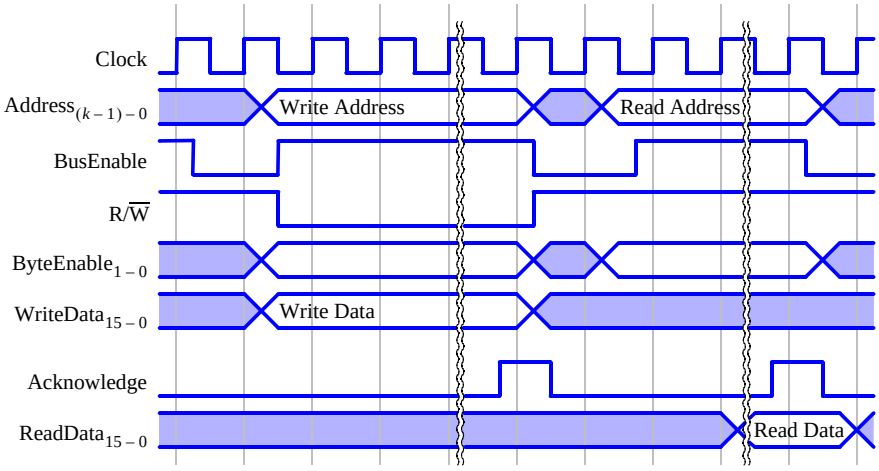
This exercise is intended for Altera's DE1, DE2, and DE2-115 boards only.

The purpose of the exercise is to learn how to communicate using a bus. In the designs generated by using Altera's Qsys tool, the Nios II processor is connected to peripheral devices by means of the Avalon Switch Fabric. To connect a device to the switch fabric, a Qsys *component* is required. Instead of using a device-specific component, we will use a more general *Avalon to External Bus Bridge* Qsys component that provides a bus-like interface to which one or more "slave" peripherals can be connected. The bridge allows the designer to create a peripheral-device interface that can be connected to the bus, and thus to the Nios II system.

Figure 1 shows the bus signals and timing information for the external bus when 16-bit data width is used. In general, the bus can be configured to use data widths of 8, 16, 32, 64, or 128 bits.



(*a*) External Bus Signals



(b) External Bus Timing Diagram

Figure 1: The External Bus for 16-bit data width.

The required signals are:

- Address – k bits (up to 32). The address of the data to be transferred. The address is aligned to the data size. For 32-bit data, the address bits $Address_{1-0}$ are equal to 0. The byte-enable signals can be used to transfer less than 4 bytes.
- BusEnable – 1 bit. Indicates that all other signals are valid, and a data transfer should occur.
- RW – 1 bit. Indicates whether the data transfer is a Read (1) or a Write (0) operation.
- ByteEnable – 16, 8, 4, 2 or 1 bits. Each bit indicates whether or not the corresponding byte should be accessed (read or written). These signals are active high.
- WriteData – 128, 64, 32, 16 or 8 bits. The data to be written to the peripheral device during a Write transfer.
- Acknowledge – 1 bit. Used by the peripheral device to indicate that it has completed the data transfer.
- ReadData – 128, 64, 32, 16 or 8 bits. The data that is read from the peripheral device during a Read transfer.
- IRQ – 1 bit. Used by the peripheral device to interrupt the Nios II processor. This is an optional signal, which is not shown in the figure.

All bus signals to the peripheral device must be read on the rising edge of the clock. To initiate a transfer the *Address*, *RW*, *ByteEnable*, and possibly *WriteData* signals are set to the appropriate values. Then, the *BusEnable* signal is set to 1.

If the *RW signal* is 1, then the transfer is a Read operation and the peripheral device must set the *ReadData* signals to the appropriate values and set the *Acknowledge* signal to 1. The *Acknowledge* signal must remain at 1 for only one clock cycle. The *ReadData* signals must be constant while the *Acknowledge* signal is being asserted. Note that the reason why the *Acknowledge* signal must be high for exactly one clock cycle is that if this signal spans two or more cycles it may be interpreted by the Avalon Switch Fabric as corresponding to another transaction.

If the *RW* signal is 0, then the transfer is a Write operation and the peripheral device should write the value on the *WriteData* lines to the appropriate location. Once the peripheral device has completed the Write transfer, it must assert the *Acknowledge* signal for one clock cycle.

Part I

Figure 2 indicates the system that we wish to design and implement. The part of the system that consists of a Nios II/s processor, a JTAG UART, an on-chip memory block and an Avalon to External Bus Bridge can be generated using the Qsys tool. The slave peripheral will be a Verilog/VHDL module that you will design. It comprises just four 16-bit registers plus some circuitry needed to display the contents of these registers on the 7-segment displays on the DE-series board. The registers are accessible as memory locations, so that the Nios II processor can write data into them.

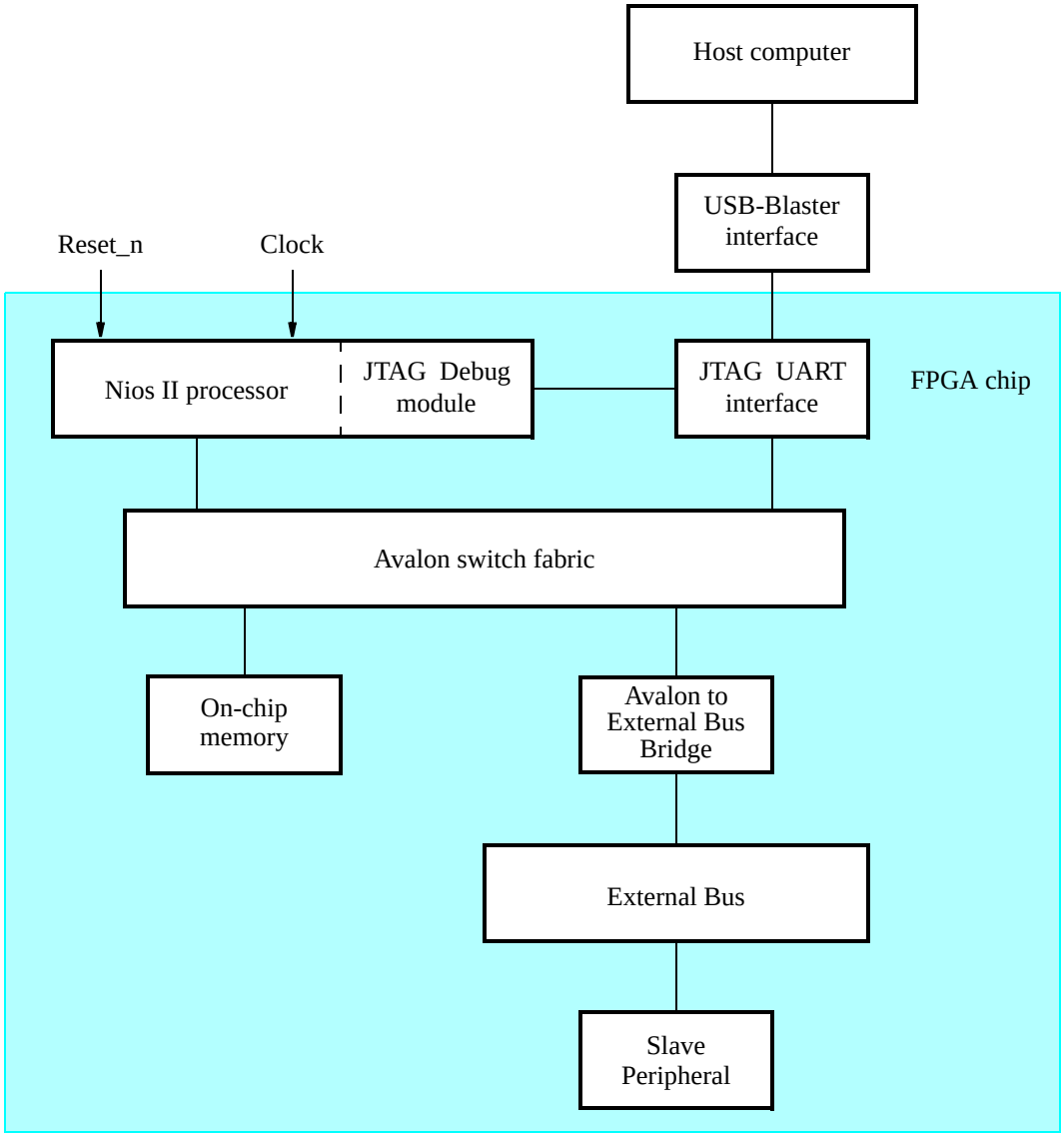


Figure 2: The desired system.

To simplify your task of implementing the desired peripheral, three modules defined in Verilog/VHDL are provided. One of these modules is provided in full. The other two are given in a skeleton form and you will have to complete them as a part of this exercise. These modules are:

- Lab10_Part1 (provided in full)
- Peripheral_on_External_Bus (skeleton is provided)
- Seven_Segment_Display (skeleton is provided)

The Verilog/VHDL code for these modules is provided as design files for this exercise on the Altera University Program website.

Examine the Lab10_Part1 module and observe that it instantiates three other modules. You will use the Qsys tool to generate the nios_system module. You have to modify the Peripheral_on_External_Bus module to connect it to the External Bus signals. Also, this module has to specify four 16-bit registers. Each of these registers should be mapped to one quarter of the address space assigned to the Avalon to External Bus Bridge by the Qsys tool. Use the 7-segment displays $HEX3 - 0$ to display the contents of these registers. (Hint: In the module Seven_Segment_Display use a separate module to convert a four-bit hexadecimal number into 7-segment bit patterns.) Since only one register can be displayed at one time on the 7-segment displays, use two slider switches, SW_1 and SW_0, to choose which register to display. These modules, in addition to the modules generated by the Qsys tool, should produce the desired system in Figure 2.

Implement the required system as follows:

1. Create a new Quartus II project called *Lab10_Part1*. From the list of available devices, choose the appropriate device name for your DE-series board, as listed in Table 1.

Board	Device Name
DE1	Cyclone II EP2C20F484C7
DE2	Cyclone II EP2C35F672C6
DE2-115	Cyclone IVE EP4CE115F29C7

Table 1: DE-series FPGA device names.

2. Use the Qsys tool to create a system named *nios_system*, which includes the following components:
   - Nios II/s processor with JTAG Debug Module Level 1 - do not select the Hardware Multiply and Hardware Divide options
   - On-chip memory - select the RAM mode, 32-bit data width, and the size of 8 Kbytes
   - JTAG UART - located in the component library Interface Protocols $>$ Serial; use the default settings
   - Avalon to External Bus Bridge - located in the components library University Program $>$ Bridges; choose the 16-bit data width and the address range of 512 Kbytes for the DE1 and DE2 boards, or 2 Mbytes for the DE2-115 board. These parameters are chosen to simplify the connection to the SRAM chip in Part II of this exercise.

     Note: The choice of the address range of 512 Kbytes implies that $k = 19$ address lines will be implemented in the bus. For the DE2-115 board, the address range of 2 Mbytes implies that $k = 21$ address lines will be implemented in the bus.

3. Make the necessary connections between the components. Connect the Avalon to External Bus Bridge to Nios II's data_master port and not to the instruction_master port.

4. From the System menu, select Assign Base Addresses followed by Assign Interrupt Numbers. This will remove any error messages about the address assignment that may be displayed. You should now have the system shown in Figure 3 (the image pertains to the DE2-115 board). Rename the components to the names shown in the figure to match the names used in the design modules provided for this exercise.

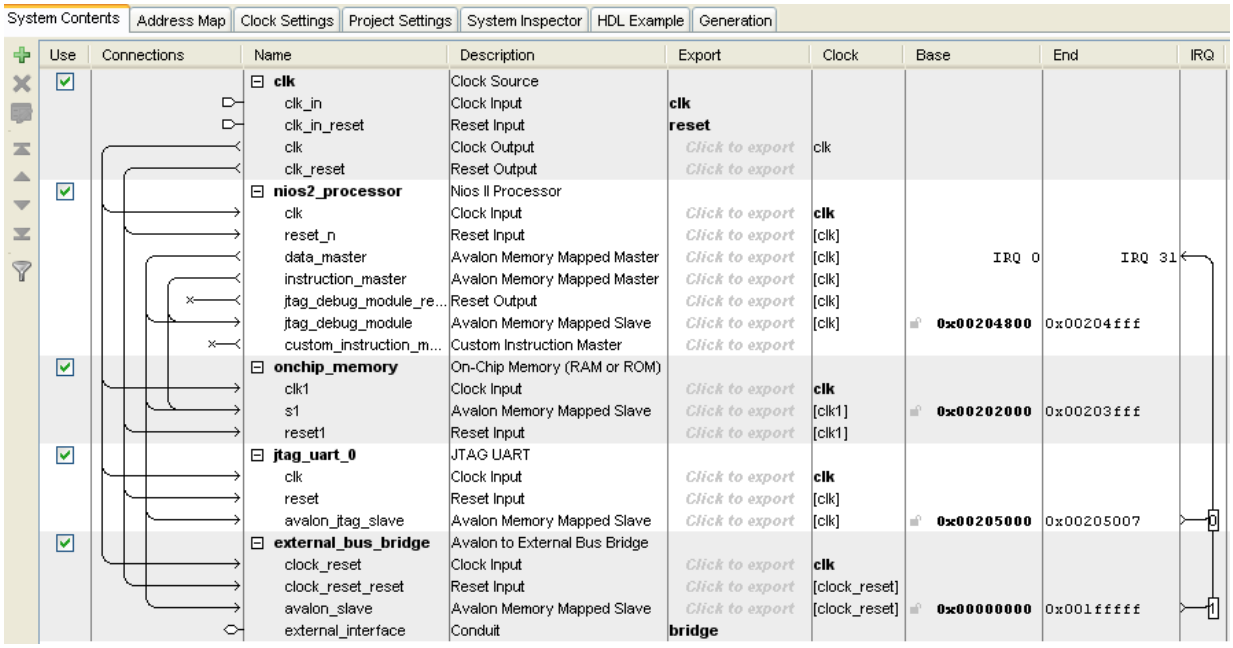


Figure 3: The Nios II system specified in the Qsys tool.

5. Edit the *nios2_processor* setting so both the reset vector and exception vector use onchip_memory. The processor should now be specified as shown in Figure 4.

6. Generate the system, exit the Qsys tool and return to Quartus II software.

7. Check that the generated Nios II system is instantiated correctly within the given Verilog/VHDL module Lab10_Part1.

8. Add the three Verilog/VHDL modules mentioned above to your Quartus II project.

9. Import the pin assignments file for the DE-series board. The file for a specific DE-series board can be found on the Altera's DE-series web pages.

10. Add the *nios_system.qip* file to your project.

11. Compile your Quartus II Project.

12. Create a Nios II assembly-language program to write a different 16-bit number into each of the four registers.

13. Use the Altera Monitor Program to download the generated system into the DE-series board. Then, compile, download and run your program. Verify that the registers can be selected for display by using the slider switches SW_{1-0}.

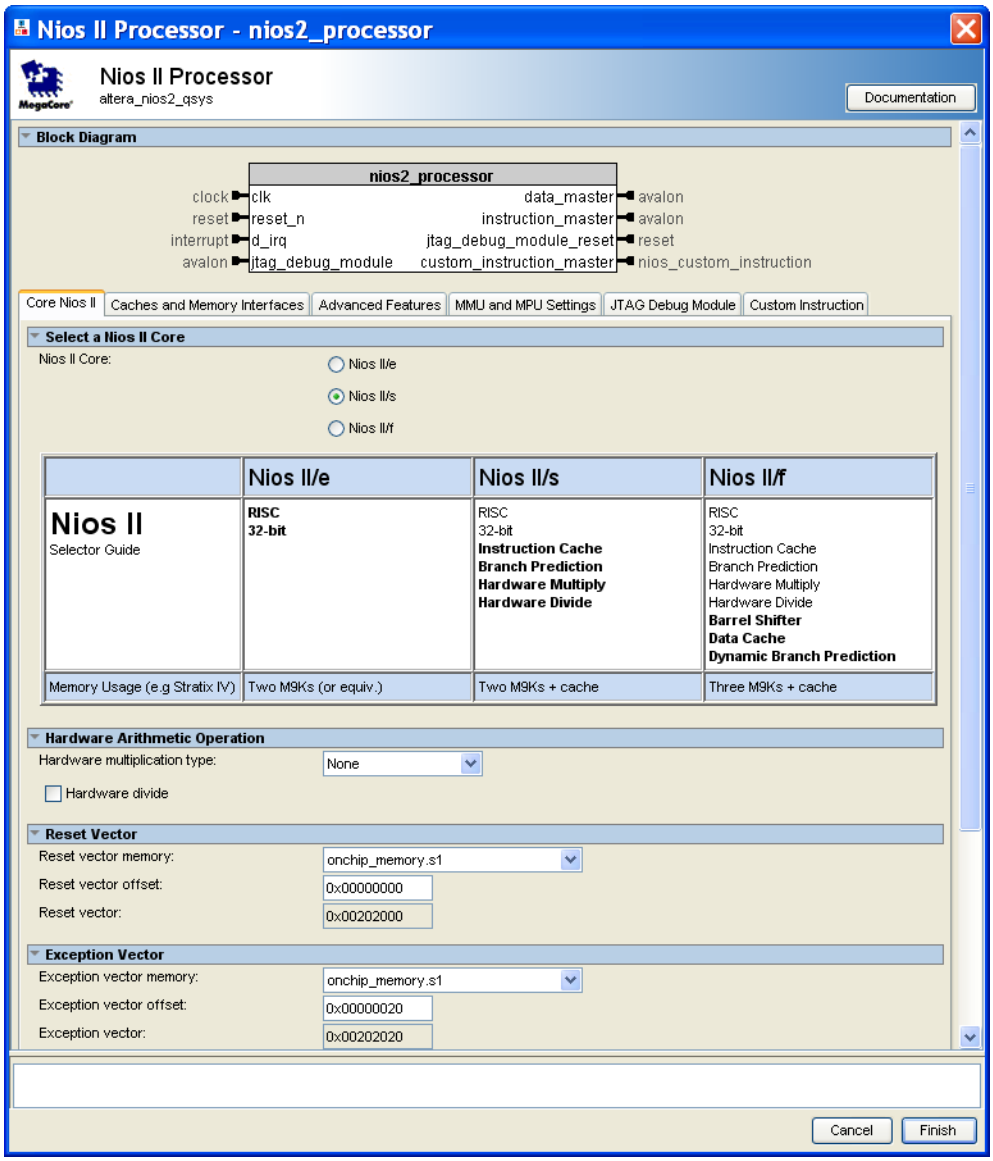


Figure 4: The Nios II processor specified in the Qsys tool.

Part II

In this part of the exercise we wish to implement a different slave peripheral. The peripheral is to serve as a controller that provides access to the SRAM chip on the DE-series board – it has to connect the SRAM chip to the Avalon Switch Fabric.

The SRAM chip uses the following signals:

- SRAM_ADDR$_{17-0}$ – 18 bits, input. Address of a 16-bit data word if the DE1 or DE2 board is being used.
  - OR -
  SRAM_ADDR$_{19-0}$ – 20 bits, input. Address of a 16-bit data word if the DE2-115 board is being used.
- SRAM_CE_N – 1 bit, input. Indicates that all other signals are valid. (Chip Enable)

- SRAM_WE_N – 1 bit, input. Indicates that the bus transfer is a write operation. (Write Enable)
- SRAM_OE_N – 1 bit, input. Indicates that the bus transfer is a read operation. (Output/Read Enable)
- SRAM_UB_N – 1 bit, input. Indicates that the upper byte should be read or written. (Upper Byte Enable)
- SRAM_LB_N – 1 bit, input. Indicates that the lower byte should be read or written. (Lower Byte Enable)
- SRAM_DQ$_{15-0}$ – 16 bits, bidirectional. These lines carry the data being transferred. They are driven by the SRAM chip during read operations and by your controller during write operations.

Figure 5 demonstrates the timing for the SRAM signals. Note that the SRAM chip completes data transfers within one cycle. Also, notice that all control signals are active low. The signal SRAM_Write_Data is an internal signal equivalent to SRAM_DQ, but it is used in write transfers only. This signal must be set to high impedance, except during a write transfer, when it is set to the data to be written.

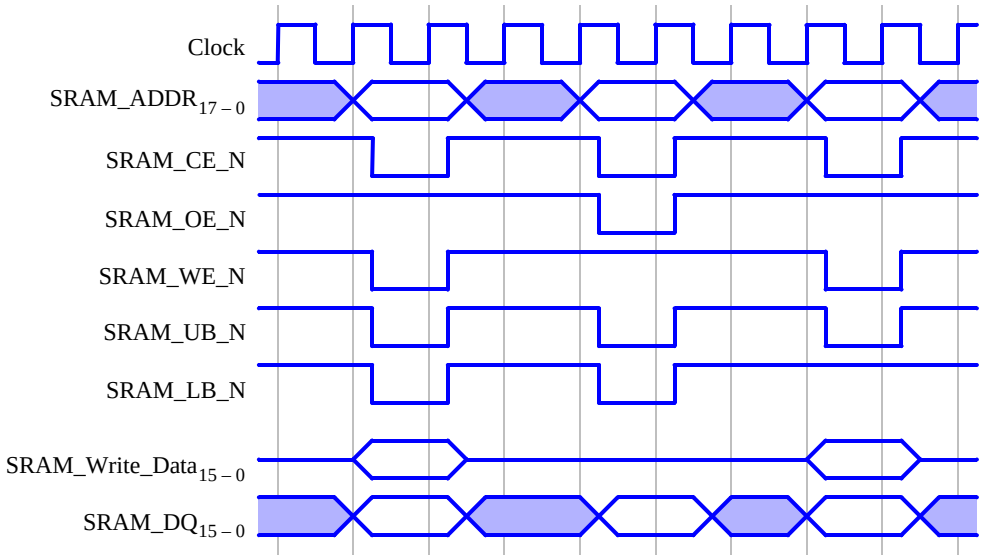


Figure 5: The SRAM signals timing diagram.

The SRAM chip can read and write one 16-bit value within one 50-MHz clock cycle, hence all signals to the SRAM chip need only be asserted for that one clock cycle for each transfer. The SRAM controller will act as the slave peripheral in Figure 1.

Perform the following steps:

1. Create a new project called *Lab10_Part2*. Use the same Qsys project settings as in Part I of this exercise, and generate a Nios II system that corresponds to Figure 3.
2. Instantiate the generated system into your project to produce the desired controller. Two Verilog/VHDL modules, Lab10_Part2 and SRAM_Controller are provided. The latter is given in skeleton form, which you will have to modify. They can be found on the same Altera web page as the modules used in Part I. Include these files in your project.
3. Import the pin assignments file for the DE-series board.

4. Add the *nios_system.qip* file to your project.
5. Compile your Quartus II Project.
6. Write a Nios II assembly-language program to write some sample data to the SRAM chip and then read this data back into the processor registers.
7. Use the Altera Monitor Program to download the generated system, and then compile and load your program.
8. Run your program to verify the correctness of your design. You can also test the SRAM controller by using the memory window in the Altera Monitor Program.

Part III

In this part we will combine the designs in Parts I and II. We want to use an External Bus and connect two slave peripherals to it. One peripheral will be the SRAM controller and the other peripheral will be the four registers with the associated display circuitry from Part I.

Perform the following steps:

1. Create a new project called *Lab10_Part3*.
2. Use the Qsys tool to generate the system in Figure 3, but choose the address range of 1024 Kbytes for the DE1 and DE2 boards, or 4 Mbytes for the DE2-115 board, for the Avalon to External Bus Bridge component. Note that this will change the base addresses that the Qsys tool will assign to the components of the system.
3. Prepare a Verilog/VHDL file that instantiates the Nios II system and implements the two slave peripherals such that
   - The SRAM controller uses the low-order 512 Kbytes of the 1024-Kbyte address space for the DE1 and DE2 boards
     - OR -
     The SRAM controller uses the low order 2 Mbytes of the 4-Mbyte address for the DE2-115 board
   - The four registers use the remaining address space
4. Modify the other Verilog/VHDL files used in Parts I and II accordingly.
5. Compile your project.
6. Write a Nios II assembly-language program that writes some sample data to the SRAM chip and then reads this data into the four registers in the slave peripheral.
7. Download the generated system, and then compile and load your program.
8. Run your program to show that the sample data is correctly displayed on the 7-segment displays.

Copyright ©2011 Altera Corporation.