Laboratory Exercise 5

Bus Communication

The purpose of this exercise is to learn how to communicate using a bus. In the designs generated by using Altera's SOPC Builder, the Nios II processor connects to peripheral devices by means of the Avalon Switch Fabric. To connect to the switch fabric, an SOPC Builder *component* is required. To make it possible to investigate the bus communication, without requiring the creation of an SOPC Builder component, this exercise will use the *Avalon to External Bus Bridge* SOPC component. The bridge allows the designer to create a peripheral device and connect it to the Nios II system in the Quartus II software. It creates a bus-like interface to which one or more "slave" peripherals can be connected. Figure 1 shows the bus signals and timing information for the external bus.
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.

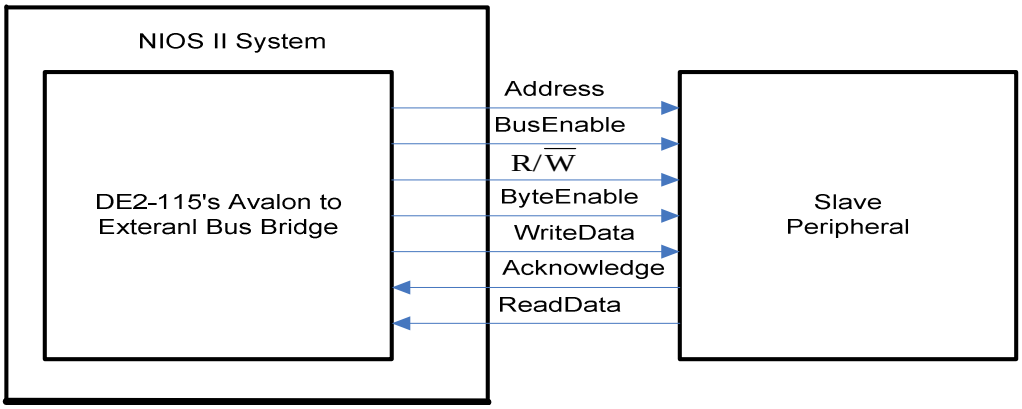
- $R/\overline{W}$ – 1 `bit. Indicates whether the data transfer is a Read (1) or a Write (0) operation.、
- ByteEnable – 16, 8, 4, 2 or 1 bits. Each bit indicates whether or not the corresponding byte should be read or written. These signals are active high.
- WriteData – 128, 64, 32, 16 or 8 bits. The 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.

The bus is synchronous – all bus signals to the peripheral device must be read on the rising edge of the clock. To initiate a transfer the *Address*, $R/\overline{W}$, *ByteEnable* and possibly *WriteData* signals are set to the appropriate values. Then, the *BusEnable* signal is set to 1.

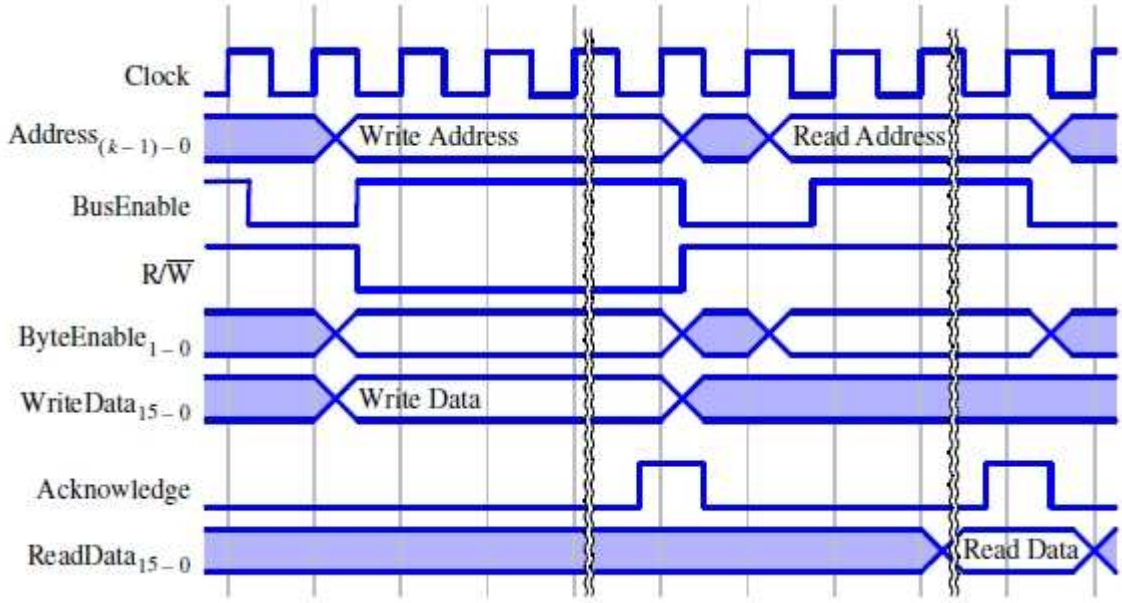
If the $R/\overline{W}$ *signal* is 1, then the transfer is a Read operation and the peripheral device must set the *ReadData* signals to the appropriate values and set the *Acknowledge* signal to 1. The *Acknowledge* signal must remain at 1 for only one clock cycle. The *ReadData* signals must be constant 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 $R/\overline{W}$ 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.



(*a*) External Bus Signals



(b) External Bus Timing Diagram

Figure 1. The Avalon to External Bus Bridge signals.

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 SOPC Builder. This should produce the system given in Figure 3. The slave peripheral will be a Verilog/VHDL module which you will design. The Avalon to External Bus Bridge connects the previously discussed external bus to the Avalon Switch Fabric of the Nios II system. The switch fabric is the main interconnection network for peripherals in a system generated by the SOPC Builder.

The slave peripheral comprises just four 16-bit registers plus the circuitry needed to display the contents of these registers on the 7-segment displays on the DE2_115 board. The registers are accessible as memory locations, so that the Nios II processor can write data into them.

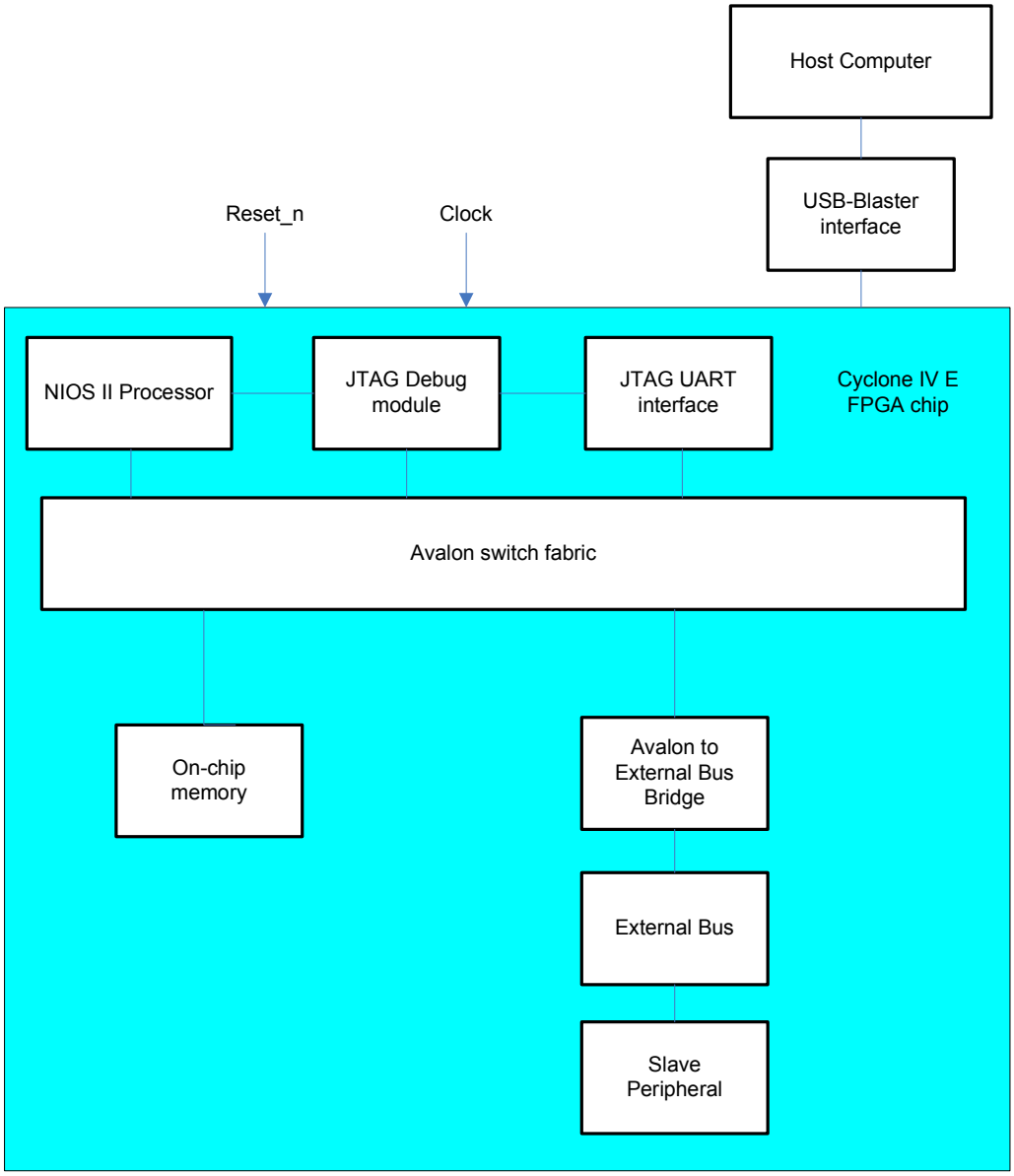


Figure 2. The desired Nios II system

To facilitate the implementation of the desired peripheral, three modules defined in Verilog/VHDL are provided. One of these modules are provided in full. The other two are given in a skeleton form and have to be completed as a part of this exercise. These modules are:

• Lab5_Part1 (which is 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 on the Altera University Program site:

ftp://ftp.altera.com/up/pub/Laboratory_Exercises/Comp_Org/comporg_lab5_design_files.zip

Once the system in Figure 3 has been generated, modify the Verilog/VHDL module, Peripheral on External Bus to connect it to the Avalon to External Bus Bridge signals. Also, modify this module such that it contains 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 SOPC Builder. Use the 7-segment displays *HEX*3 − 0 to display the contents of these registers. Since only one register can be displayed at one time on the 7-segment displays, use two toggle switches, *SW*1 and *SW*0, to choose which register to display. These modules, in addition to the modules generated by the SOPC Builder, should implement the desired system in Figure 2.

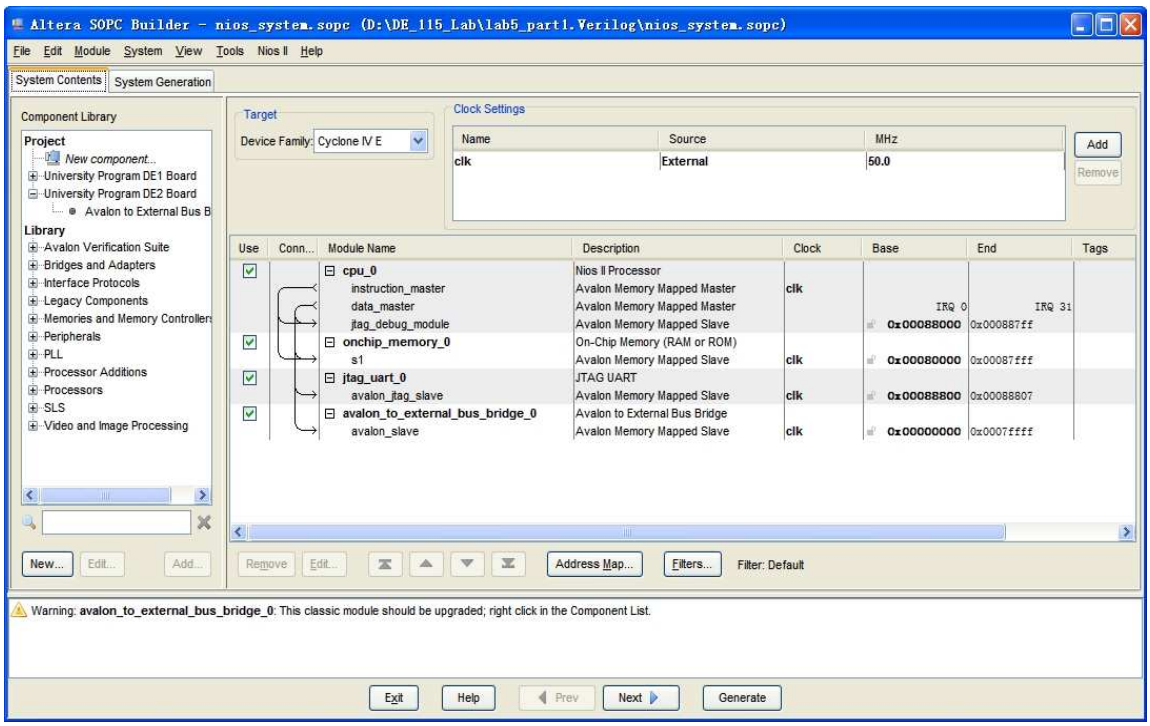


Figure 3. The Nios II system specified in the SOPC Builder.

To use the Bridge component, you have to copy the directory *altera up avalon to external bus bridge* into your project directory, before you start the SOPC Builder software. This directory is available on the Altera University Program site:

ftp://ftp.altera.com/up/pub/University_Program_IP_Cores/avalon_to_external_bus_bridge.zip

Implement the required system as follows:

1. Create a new Quartus II project called *Lab5_Part1*. Select Cyclone IV E EP4CE115F29C7 as the target chip, which is the FPGA chip on the Altera DE2_115 board.

2. Use the SOPC Builder to create a system named *nios_system*, which includes the following components:
   - Nios II/s processor with JTAG Debug Module Level 1
   - On-chip memory - select the RAM mode and the size of 32 Kbytes
   - JTAG UART - use the default settings

- Avalon to External Bus Bridge - choose the 16-bit data width and the address range of 512 Kbytes. These parameters are chosen to simplify the connection to the SRAM chip in Part II of this exercise.In the SOPC Builder window, the desired bridge is found by selecting **Avalon Component**>**University Program** > **Bridges**>**Avalon to External Bus Bridge**.

Note: The choice of the address range of 512 Kbytes implies that k = 19 address lines will be implemented in the bus.

3. Connect the Avalon to External Bus Bridge to Nios II's data master port and not to the instruction master port. You can remove the connection between the bridge and the instruction master port by clicking on the connecting path in the SOPC Builder window.

4. From the **System** menu, select **Auto-Assign Base Addresses**. You should now have the system shown in Figure 3.

5. Generate the system, exit the SOPC Builder and return to Quartus II software.

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

7. Check that the generated Nios II system is instantiated correctly within the given Verilog/VHDL module Lab5_Part1.

8. Modify the Verilog/VHDL module Peripheral on External Bus to implement the bus protocol needed to connect the four 16-bit registers to the Avalon Switch.

9. Modify the Verilog/VHDL module Seven Segment Display to display the four registers on the 7-segment displays in hexadecimal format.

10. Import the pin assignments using the file *DE2-115 pin assignments.csv*.

11. Compile your Quartus II Project.

12. Program and configure the Cyclone IV E FPGA on the DE2_115 board to implement the generated system.

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

14. Use the Altera Monitor Program to compile, download and run your program. Verify that the registers can be selected for display by using the toggle switches *SW*1−0.

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 in the DE2_115 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. The addresses of the 16-bit data words.
- *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 4 demonstrates the timing for the SRAM signals. Notice 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.

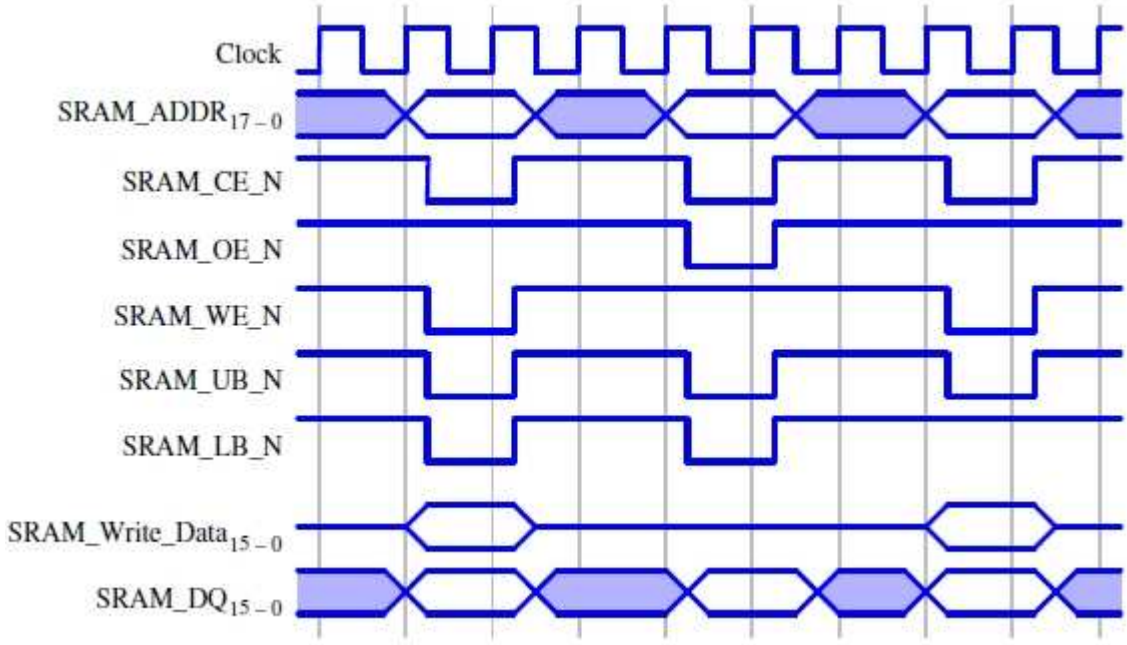


Figure 4. The SRAM signals timing diagram

The SRAM chip can read and write one 16-bit value within one 50-MHz clock cycle, so 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 *Lab5_Part2*. Use the same SOPC Builder project settings as in Part 1 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 files, *Lab5_ Part2* and *SRAM Controller* are provided (in skeleton form) to help you get started. They can be found on the the same Altera web page as the files used in Part I.

3. Compile your project and configure the Cyclone IV E chip to implement the generated system.

4. 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.

5. 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 just one Avalon to External Bus Bridge 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.

Perform the following steps:

1. Create a new project called *Lab5_Part3*.

2. Use the SOPC Builder to generate the system in Figure 3, but choose the address range of 1024 Kbytes for the Avalon to External Bus Bridge component. Note that this will change the base addresses that the SOPC Builder will auto-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
   - 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 and configure the Cyclone IV E chip to implement the generated system.

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. Run your program to show that the sample data is correctly displayed on the 7-segment displays.