Educational Objective

The purpose of this exercise is to investigate the use of a bus bridge to interface the NIOS II with an external component, the use of Altera's MegaWizard Plug-in Manager to create a RAM module and the creation of an arbitration scheme to manage two components writing to the same RAM location.

Technical Objective

The technical objective of this demonstration is to learn how to communicate using a bus. In the designs generated by Altera's QSYS system builder, the Nios II processor connects to peripheral devices by means of the Avalon Switch Fabric. To connect to the switch fabric, previous demonstrations and labs required the creation of a QSYS custom component. To make it possible to investigate the bus communication, without requiring the creation of a custom component, this exercise will use the Avalon to External Bus Bridge component. The bridge allows the designer to create a peripheral device and connects it to the Nios II system in the Quartus software. The Avalon to External Bus Bridge creates a bus-like interface to which one or more "slave" peripherals can be connected.

Overview

This demonstration is broken into multiple parts. Each part builds on the previous part so you must do all the steps in order. Read the instructions carefully and do not skip any steps.

The external bus signals and timing information of the Avalon to External Bus Bridge is shown in Figure 1 below. The required signals are:

Signal Name	Size	Description
Address	k-bit (up to 32)	This is the address where the data is to be transferred to or from. 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 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, 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.

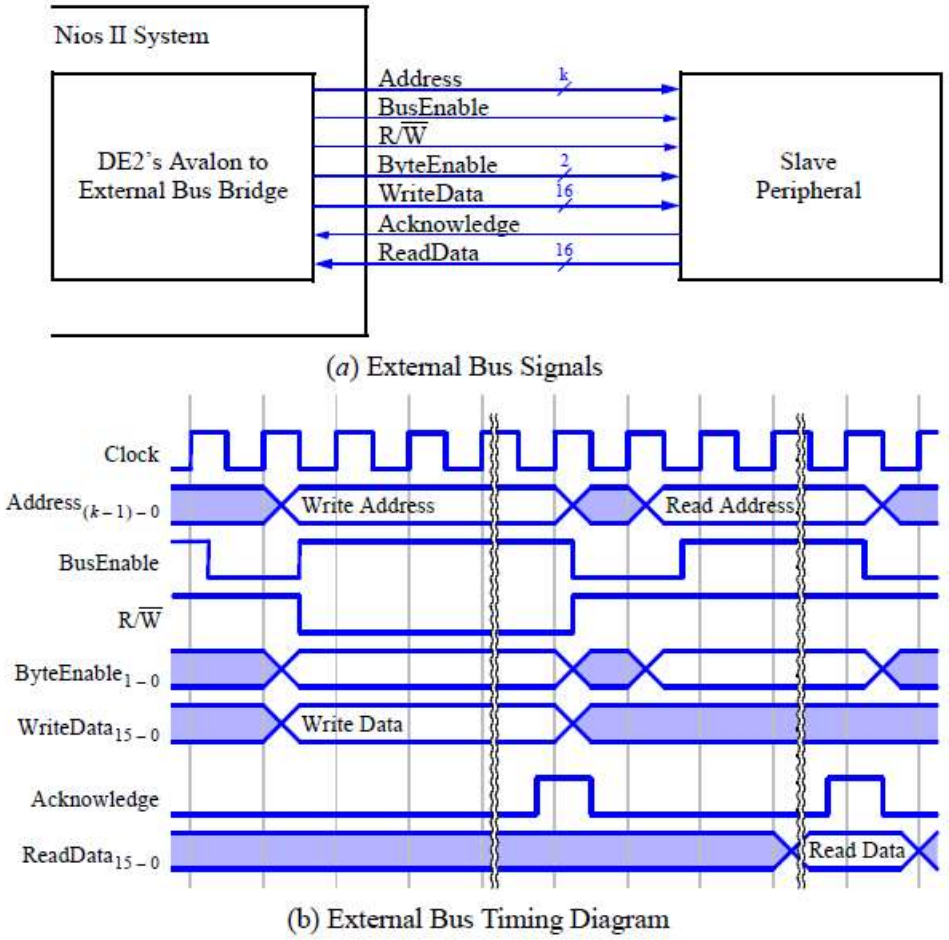

Figure 1: Avalon to External Bus Bridge signals

PART 1: Bus Bridge Component and Peripheral on External Bus

Figure 2 indicates the system that we will design and implement. The part of the system consisting of a Nios II/e processor, a JTAG UART, an on-chip memory block and an Avalon to External Bus Bridge can be generated using QSYS. This should produce the system given in Figure 3. The slave peripheral will be a VHDL module that functions as an RAM controller to interface with an Altera LPM RAM module. 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 QSYS.

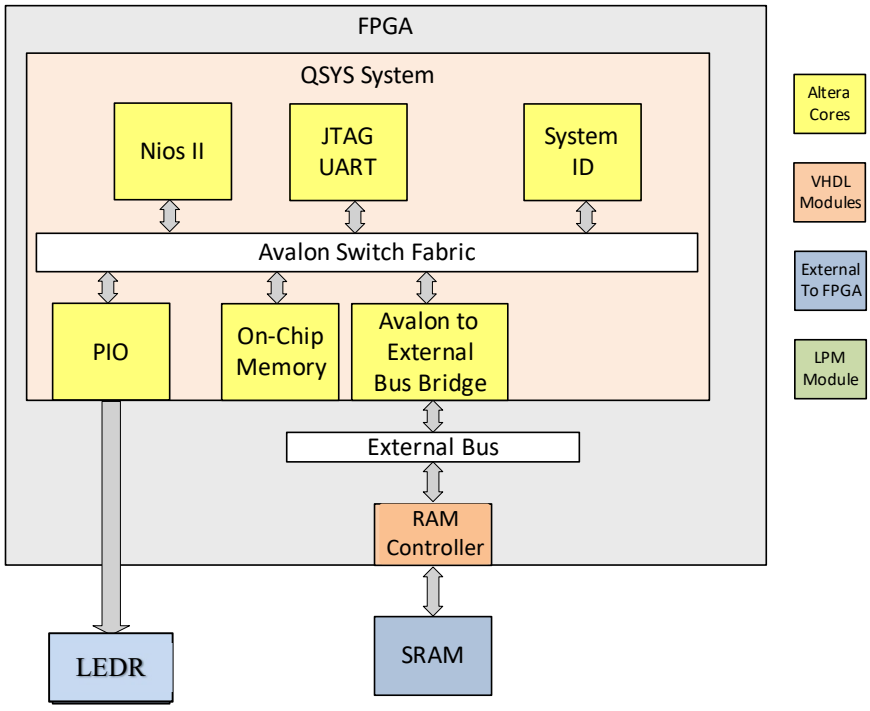

Figure 2: High-level Block Diagram of Design

Realize the required hardware by implementing a Nios II system on the DE1-SoC board, as follows:

1. Create a new Quartus project that is stored in the directory **Arbitration_Demo/Quartus/part1**.
2. Use QSYS to generate the desired circuit, called **nios_system**, which comprises:
    - Nios II/e processor
    - On-chip memory - select the RAM mode and the size of 32 Kbytes
    - A JTAG UART - use the default settings
    - System ID Peripheral with unique system ID
    - A ten-bit output PIO named ledr to connect to the leds on the DE1-SoC board
3. Add the Avalon to External Bus Bridge. In the QSYS window, the desired bridge is found by selecting **University Program > Bridges > Avalon to External Bus Bridge**. Choose the 16-bit data width and the address range of 2 Kbytes. These parameters are chosen to simplify the connection to the RAM IP in the next part of this exercise. Note: The choice of the address range of 2 Kbytes implies that k = 11 address lines will be implemented in the bus.
4. Rename the Avalon to External Bus Bridge component to **avalon_bridge**.
5. Connect the Avalon to External Bus Bridge to Nios II's data master port but not to the instruction master port.
6. Select the external_interface for the avalon_bridge. Click on export column and name the export **avalon_bridge**.
7. Lock the address of the Avalon_brige at 0x0000_0000. From the System menu, select Auto-Assign Base Addresses.

8. Assign the IRQ for the JTAG UART and the Avalon_bridge.
9. You should now have the system similar to Figure 3.

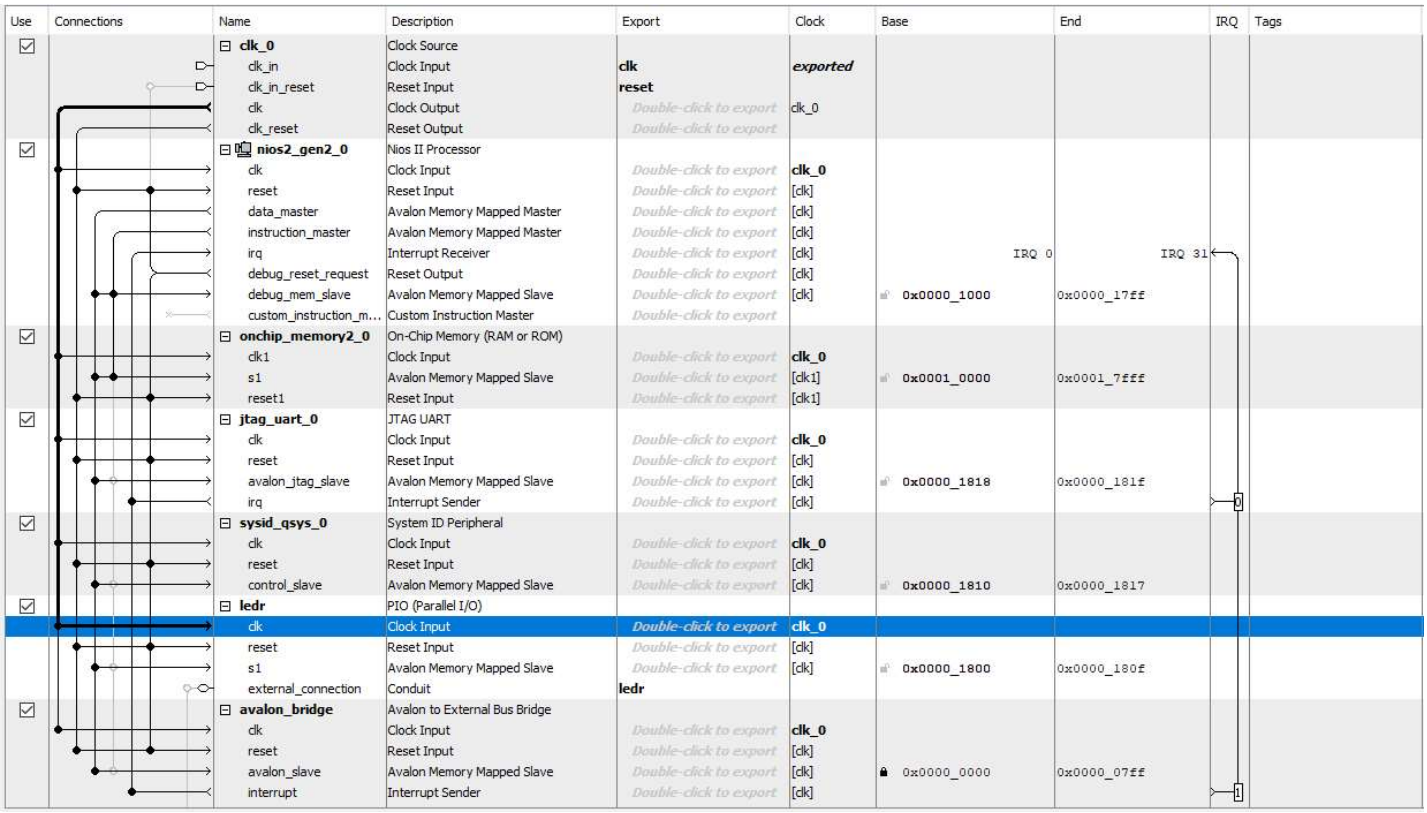

Figure 3: Nios-II System in QSYS

10. Save and Generate the VHDL

Now we need a memory for the bus bridge to connect to. Create a RAM IP in Quartus as follows:

1. In your Quartus project look at the **IP Catalog** on the right side of the window.

2. Navigate to following IP component:

   Installed IP->Library->Basic Functions->On Chip Memory->RAM: 1-Port

3. Double click on this component to open the wizard and create one. Complete the pop-up windows using these steps
   a. Choose VHDL and store it in your current project with the name **external_RAM**
   b. Make the 'q' output bus 16 bits and the number of words, 2048. This matches the Avalon_bridge created in the previous section. Keep Auto for memory block type and single clock for clocking method.
   c. Accept the defaults on all remaining pop-ups.

The next step is to put everything together in a VHDL file

11. Add the **arbitration_part1.vhd** and **RAM_controller.vhd** files to your project. Also add the nios_system.qip and verify the external_RAM.qip is in the project.

12. Import the pin assignments, compile and program the DE1-SoC board

The final step is to test the system using software.

13. Open the NIOS II Software Build Tools for Eclipse
14. Create a new Nios II Application and BSP from template and name it Arbitration_part1_app. Remember to choose Blank Project for the template.
15. Generate BSP and copy system.h from the bsp folder to the app folder.
16. Move the **Arbitration_part1.c** file to the app folder and build project.
17. Choose Debug As > NIOS II hardware. From the Debug perspective, run the program.
18. Using the memory viewer (at address 0), verify that the memory is filled appropriately.
19. Now insert a breakpoint inside the readback loop and rerun the program. When the program stops, manually change one of the memory locations. Remove the breakpoint and click on resume. Verify that the LEDs come on.

Demonstration: Submit a video to the dropbox of step #19, be sure to include an explanation of what is happening.