Laboratory Exercise 11

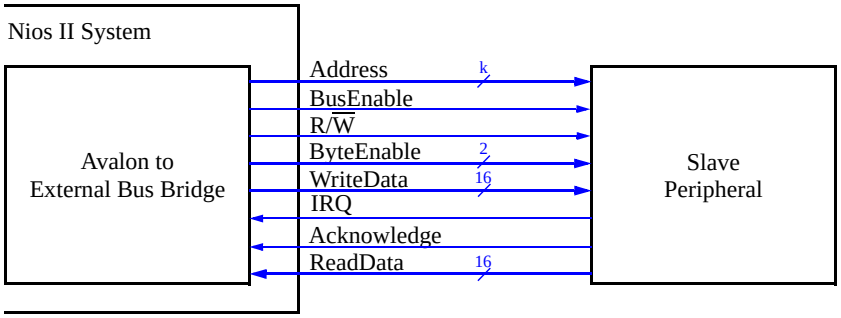
Multiple Processors and Bus Arbitration

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

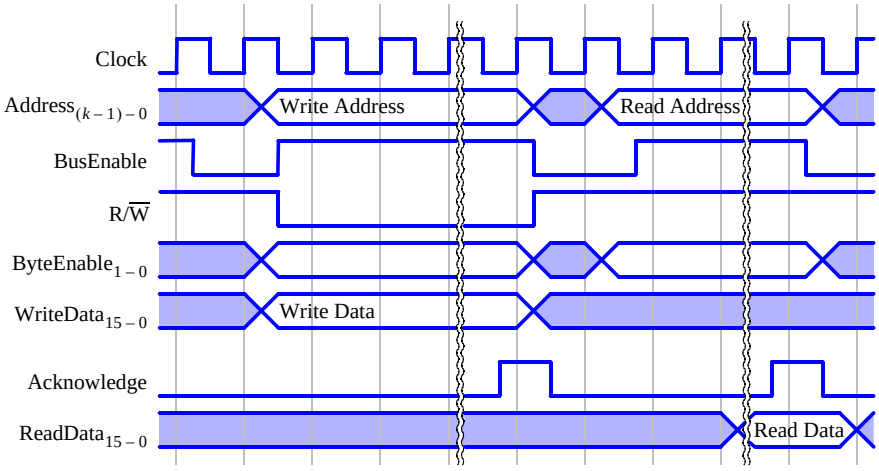
The purpose of the exercise is to investigate the concept of bus arbitration and its use when two processors are included in a system. The exercise follows naturally the Laboratory Exercise 10, but it can also be done independently.

Altera's Qsys tool can use the *Avalon to External Bus Bridge* component to implement a bus-like structure to which the user may connect a variety of peripheral devices. The bridge allows the designer to connect a peripheral device to a Nios II system in the Quartus II software. It provides a bus interface to which one or more "slave" peripherals can be connected.

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 Avalon to External Bus Bridge signals.

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

In this part of the exercise, we want to connect the SRAM chip on the DE-series board to a Nios II processor. Figure 2 indicates the system that we wish to design and implement. The part of the system that consists of a Nios II processor, a JTAG UART, an on-chip memory block and an Avalon to External Bus Bridge can be generated using the Qsys tool. The SRAM chip will be added as a slave peripheral, by creating an *SRAM Controller* Verilog/VHDL module that 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 Qsys tool.

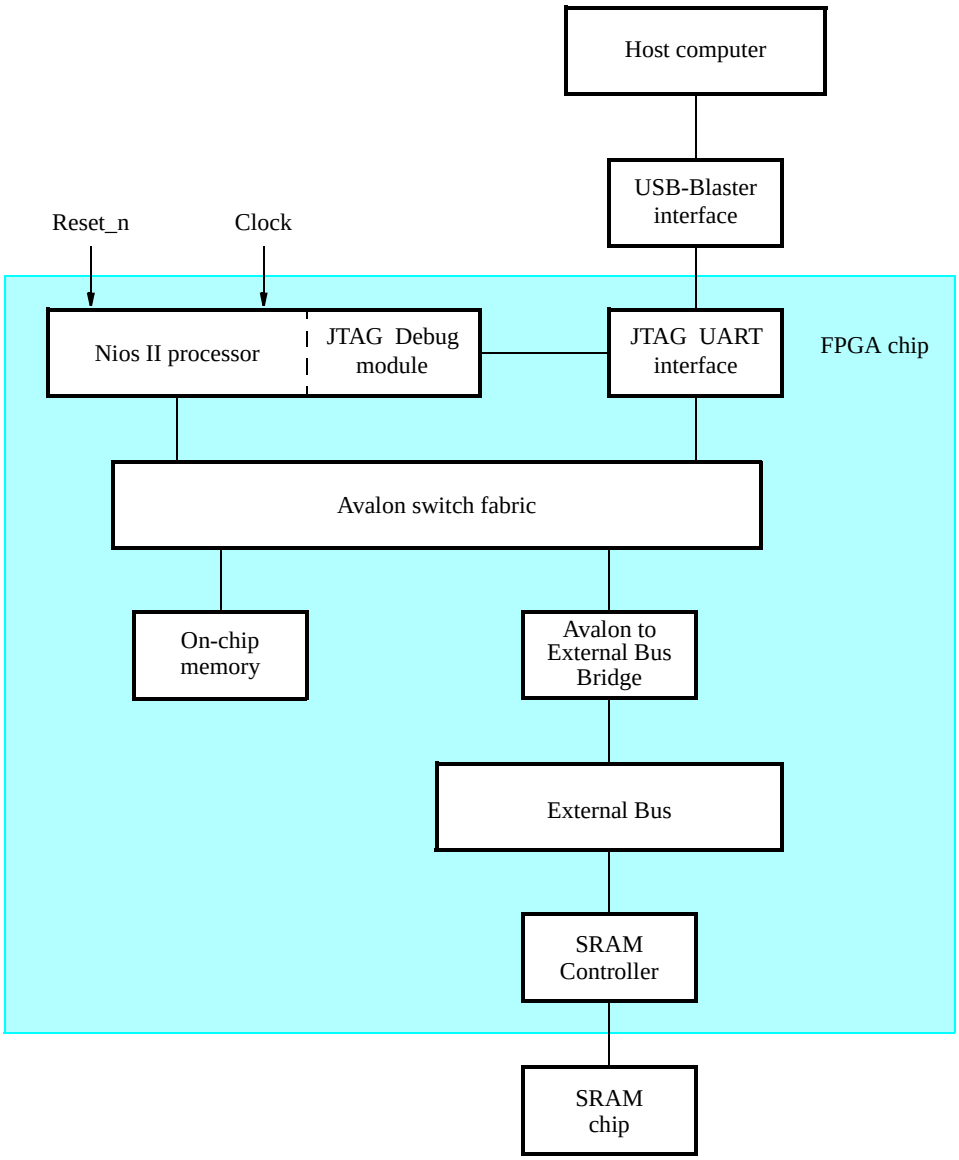


Figure 2: The desired system

The SRAM chip is organized as a 256K $\times$ 16 bits memory. Each 16-bit word is accessed by specifying an 18-bit address. Individual bytes are accessed by specifying the UB (upper byte) and LB (lower byte) control signals. 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 3 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.

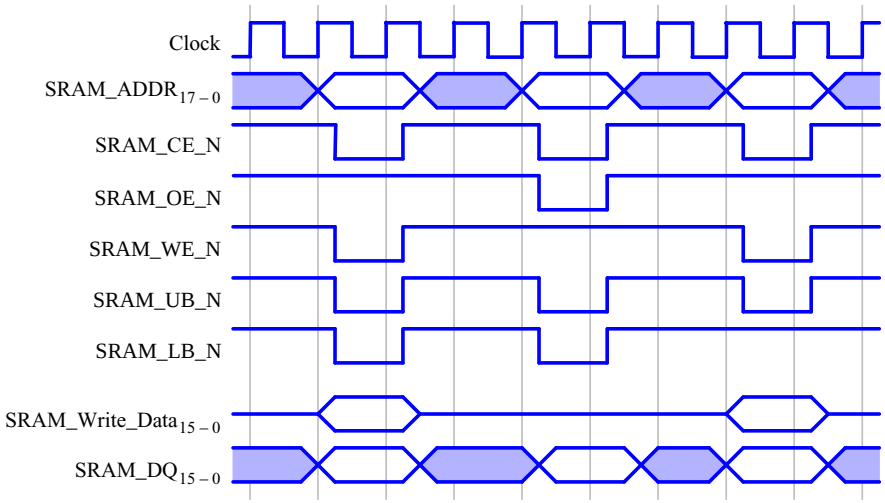


Figure 3: The SRAM signals timing diagram

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 16-bit transfer. The SRAM controller will act as a slave peripheral in Figure 2

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

- Lab11_Part1 (provided in full)
- SRAM_Controller (skeleton is provided)

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

1. Create a Quartus II project called Lab11_Part1. From the list of available devices, choose the appropriate device name for your DE-series board, as listed in Table 1. Note that the DE0 and DE2-70 boards are not listed because this lab document does not apply to them.

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
   - On-chip memory - select the RAM mode, 32-bit data width, and the size of 4K bytes
   - JTAG UART - use the default settings
   - Avalon to External Bus Bridge - 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. In the Qsys tool window, the desired bridge is found by selecting 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. 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 4 (the image pertains to the DE-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.

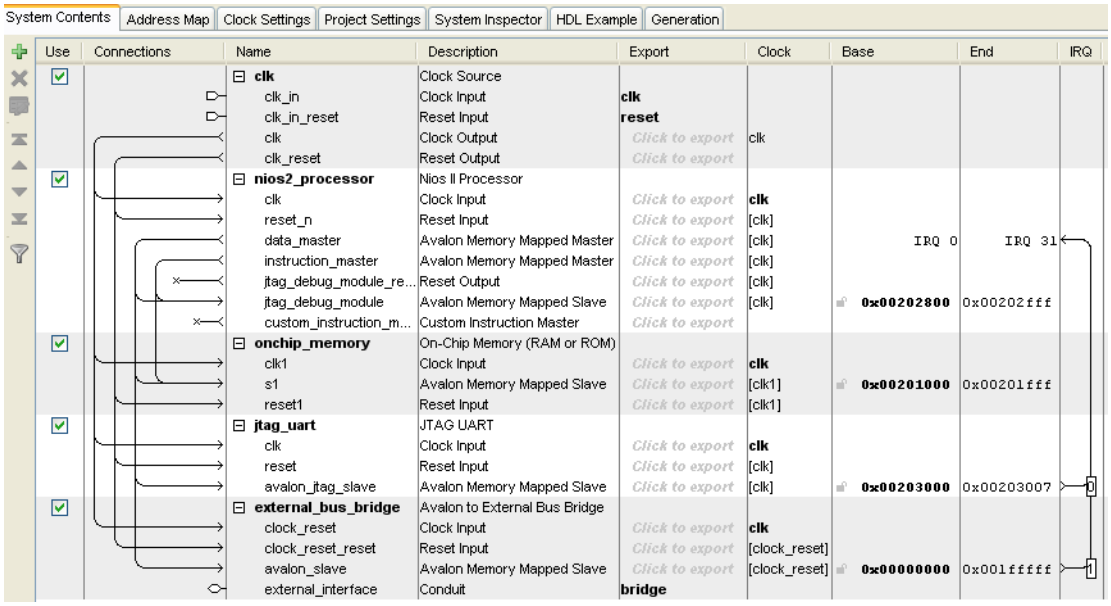


Figure 4: The Nios II system for Part I specified in the Qsys tool.

5. Edit the *nios2_processor* setting so both the reset vector and exception vector use onchip_memory.
6. Generate the system, exit the Qsys tool and return to the Quartus II software.
7. Add the two Verilog/VHDL modules mentioned above to your Quartus II project.
8. Check that the generated Nios II system is instantiated correctly within the given Verilog/VHDL module Lab11_Part1.
9. Complete the design of the SRAM Controller – start with the skeleton design in the Verilog/VHDL file SRAM_Controller.
10. 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.
11. Add the *nios_system.qip* file to your project.
12. Compile your Quartus II Project.
13. Write a Nios II assembly-language program to write some data into a few locations in the SRAM chip and then read this data back into processor registers.
14. Use the Altera Monitor Program to download the generated system into the DE-series board. Then, compile, download and run your program. Verify that your system works correctly.

Part II

In Part I we examined a system where a single master, the Nios II Data Master, communicates with a slave connected to the bus. In this part we will consider the case where two masters are connected to one slave. Since both masters may request access to the slave at the same time, it is necessary to provide an arbitration circuit which will grant access to the slave peripheral to only one master at a time.

Consider a system that includes two Nios II processors which have to communicate with each other by exchanging messages. One way of providing the communication capability is to use a memory module that is accessible by both processors. Each processor can write a message into the memory module and the other processor can read it from this module. We will use the SRAM chip on the DE-series board as the common memory. The desired system is given in Figure 5. Each processor can access the SRAM memory by means of an Avalon to External Bus Bridge, as in Part I. Since there are two processors, there will be two instances of the Avalon switch fabric and two Avalon to External Bus Bridges as shown in the figure. The Arbiter circuit must handle the Read and Write requests from the processors, allowing one request to be serviced at any time. The SRAM Controller is the circuit used in Part I. A connection to the host computer is provided by incorporating a separate JTAG UART interface for each processor. These interfaces are connected through a JTAG Hub to the USB-Blaster interface.

As a simple application of our system, we wish to implement a dialog capability that resembles a "chat room". A separate version of the Altera Monitor Program will be activated for each processor. An application program will allow the user to type a message using one Monitor Program and this message will appear in the terminal window of the other Monitor Program.

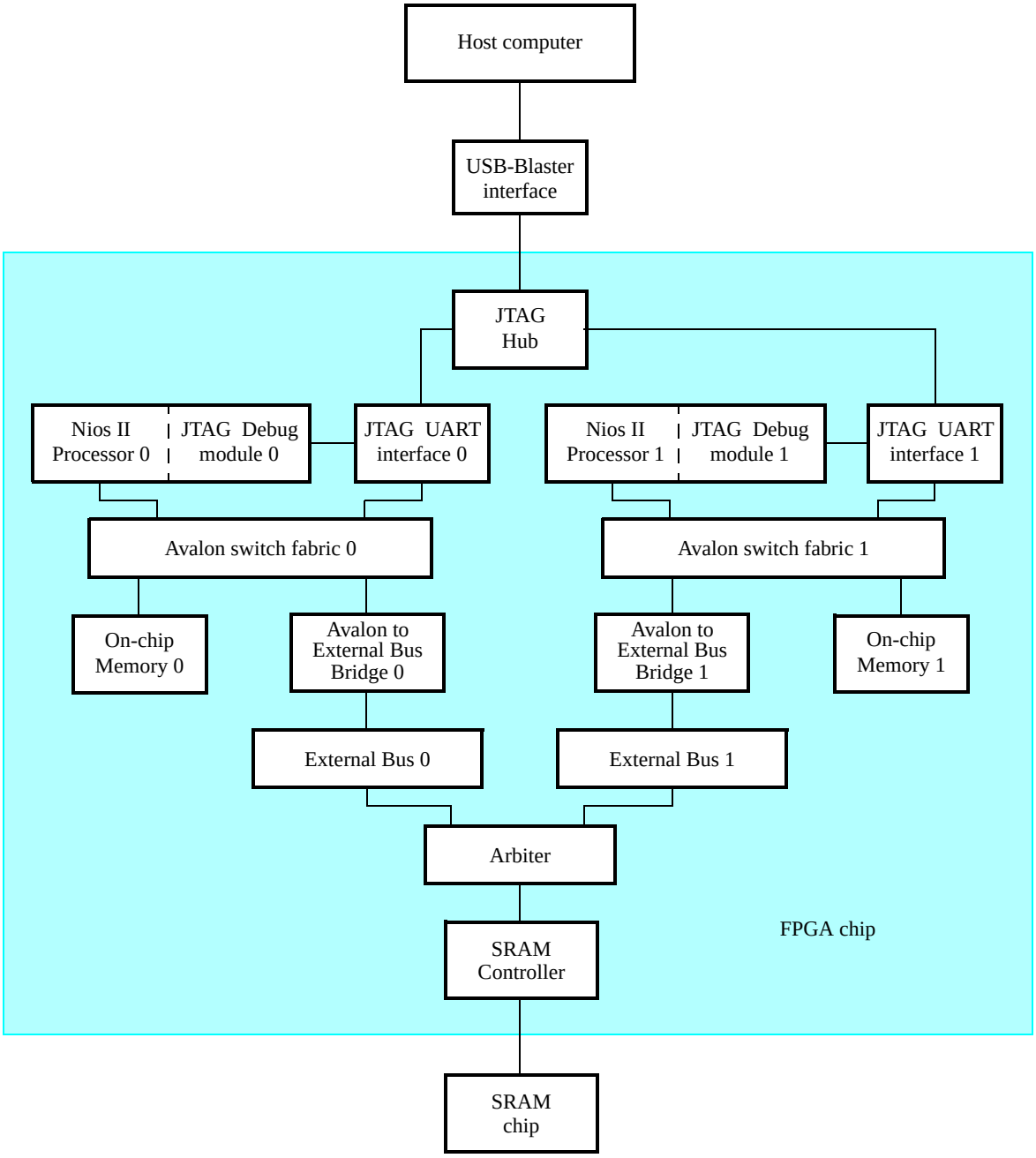


Figure 5: The system for Part II.

Perform the following steps:

1. Create a new project called Lab11_Part2.
2. Use the Qsys tool to generate the system in Figure 6, which implements the modules given in Figure 5. The system comprises two subsystems, each of which includes:
   - Nios II/s processor with JTAG Debug Module Level 1
   - On-chip memory - select the RAM mode and the size of 4 Kbytes

- JTAG UART - use the default settings
- Avalon to External Bus Bridge - select 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. Connect the bridge to the Nios II's data_master port.

3. From the System menu, select Assign Base Addresses followed by Assign Interrupt Numbers. You should now have the system shown in Figure 6. Rename the components to the names shown in the figure to match the names used in the design modules provided for this exercise.

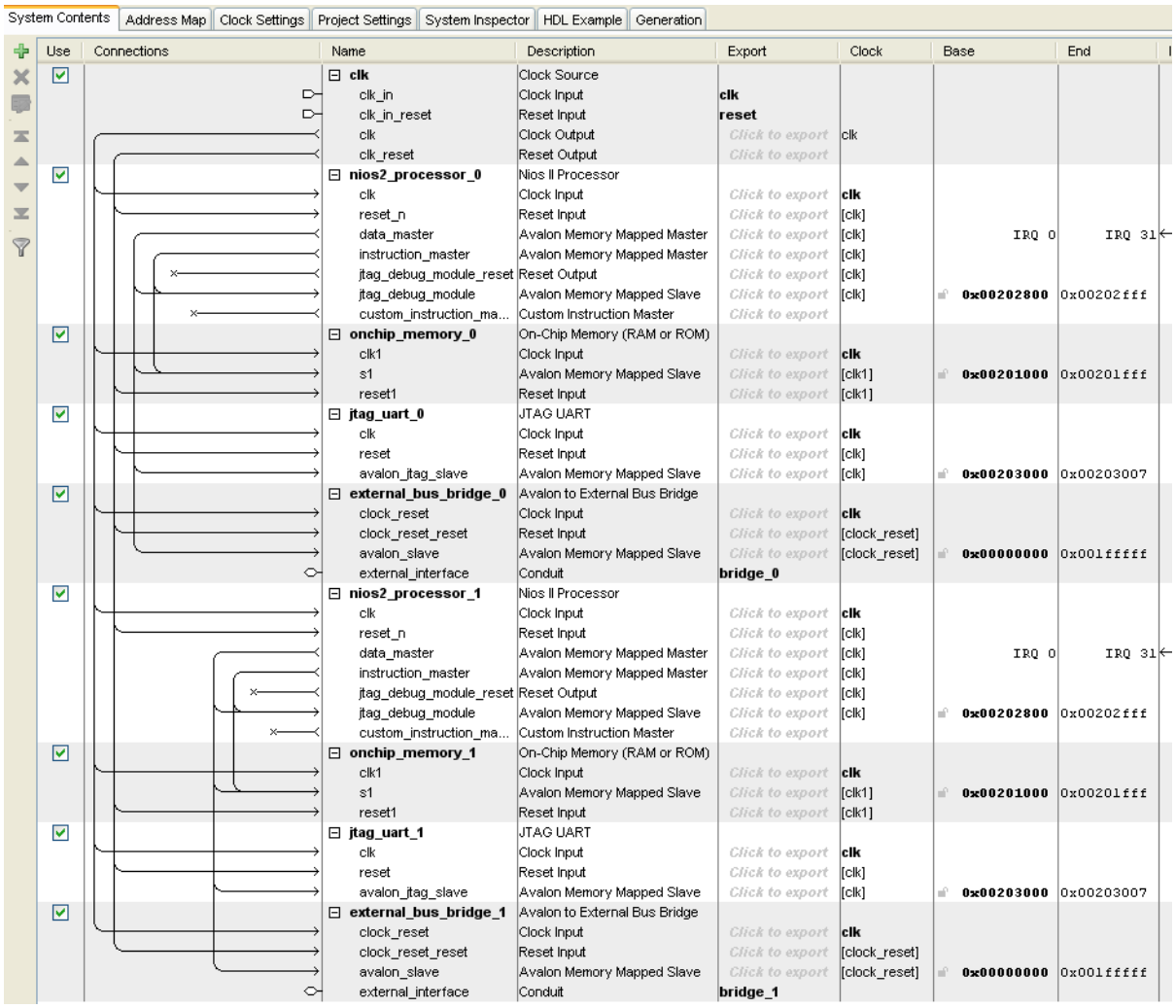


Figure 6: The Nios II system for Part II specified in the Qsys tool.

4. Edit the settings for *nios2_processor_0* and *nios2_processor_1*, so their reset vector and exception vector use onchip_memory_0 and onchip_memory_1, respectively.

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

6. Develop the Arbiter module which handles transfers from both External Buses. Design a reasonable scheme for choosing which request to respond to first if both buses request transfers in the same clock cycle. The Verilog/VHDL file Bus_Arbiter, which gives a skeleton of the required design, is provided to help you get started.

7. Make sure that the generated *nios_system* and your Arbiter are properly instantiated in the Verilog/VHDL file Lab11_Part2.

8. Add the Verilog/VHDL files to your Quartus II project and specify the pin assignments by importing the file for the DE-series board.

9. Compile your Quartus II Project.

10. Write two application programs that implement the functionality that resembles a chat room. Each program is to be executed using a separate Monitor Program. Program A has to perform the following functions:

    - It reads a character (typed on the keyboard of the host computer) via its JTAG UART.
    - It echoes the read character back to the terminal window of its Monitor Program.
    - It places the read character into an SRAM location *LOC*, and also sets to 1 a status flag in the SRAM location *LOC+4* to indicate that a new character is available. These locations will be read by Program B which is run by the other processor (in the other Monitor Program).
    - It checks the SRAM location *LOC+12*, which holds the status flag that indicates whether there is an incoming character from program B. If the status flag is set to 1, then the character in the SRAM location *LOC+8* must be read and transmitted to the terminal window of the Monitor Program.

    Program B is the same as Program A, except that it uses the SRAM locations *LOC+8* and *LOC+12* for output purposes and *LOC* and *LOC+4* for input purposes.

11. Open the Monitor Program and create a project that uses nios2_processor_0 as the processor. Download the generated system into the FPGA chip. Since the system includes two JTAG UART components, one per processor, it is necessary to provide their identification numbers to the Monitor Program. This information is available in the *.jdi* file produced by the Quartus II compiler. Include this file, along with the *.qsys* and *.sof* files, as shown in Figure 7. Then, compile and download Program A. Make sure that the .text and .data section pointers point to the on-chip memory.

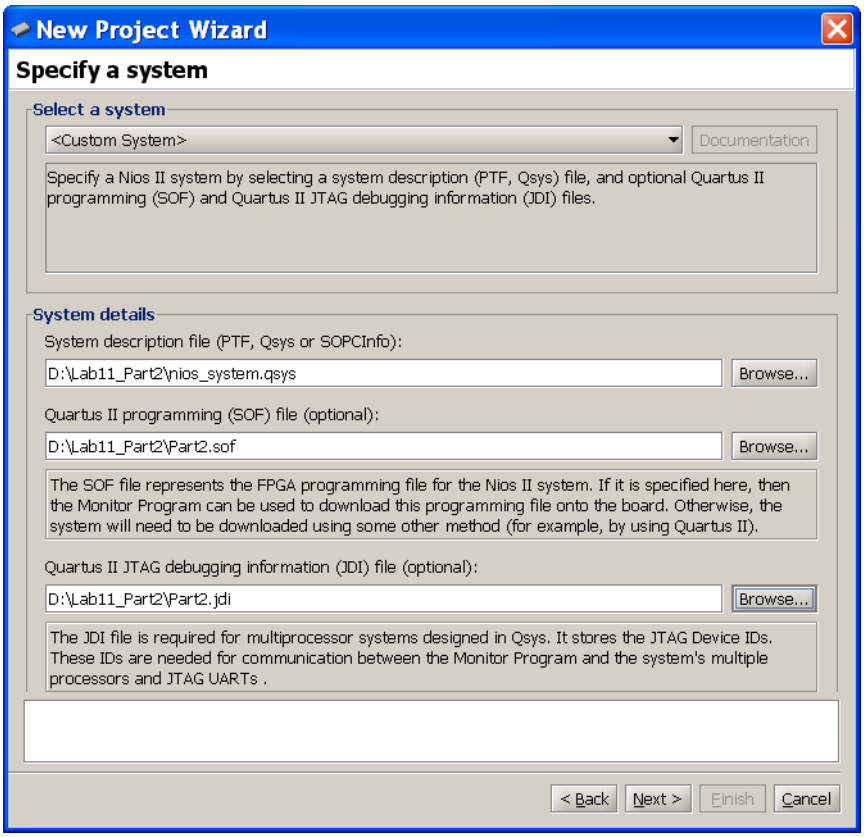


Figure 7: Specifying the files that define the system.

12. Open a second instance of the Monitor Program. Create a new project and choose nios2_processor_1 as the processor, as indicated in Figure 8. Compile and download Program B, without downloading the generated two-processor circuit again.

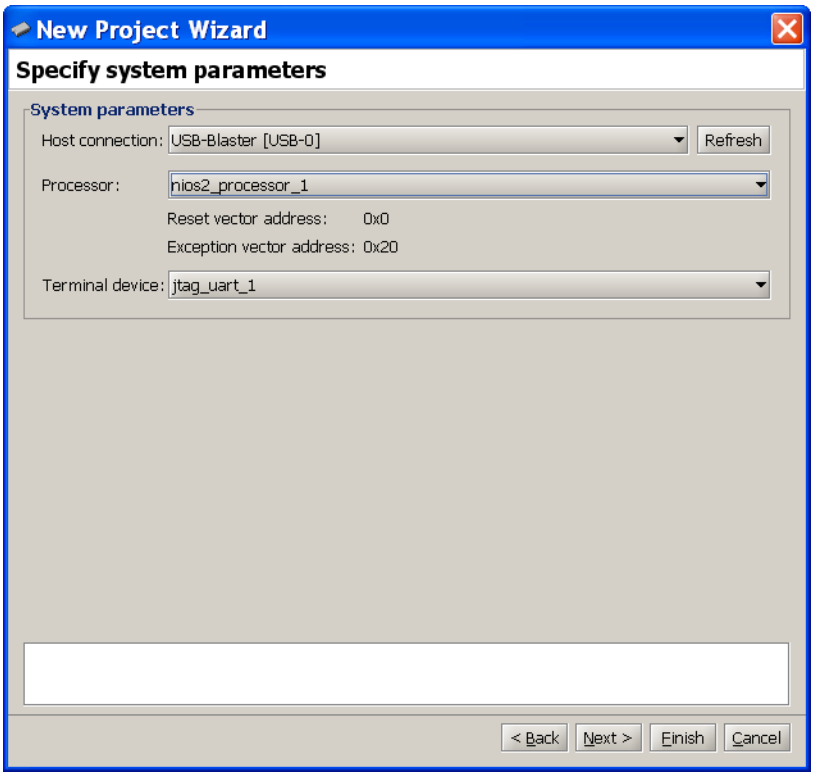


Figure 8: Selection of nios2_processor_1 in the second instance of Monitor Program.

13. Run both programs and demonstrate the chat room capability. Note that the terminal window in a Monitor Program is activated by clicking on it.