Signoffs and Grade:

Name:____

Component	Signoff	Date	Time
Working simulation Shows bytes, half words, and words.			
Demonstrate three failures Run all three tests, breaking in between to force failures. Show console failures.			
Demonstrate polling INT LEDs show unique pattern			
Demonstrate function INT LEDs show unique pattern			

==

Component	Received	Possible
Signoffs		100
Penalties • after the first 15 minutes of lab session 8: -10 • after the first 15 minutes of lab session 9: -25 no signoffs after the first 15 minutes of lab session 10:	-	
Total		100

Educational Objective

The educational objective of this lab is to further investigate the use of the component editor to create a custom IP module block that can be instantiated in QSYS, to understand the difference between 8/16/32-bit memory access, and to reinforce concepts of using functions, polling and interrupts.

Technical Objective

The technical objective of this laboratory is to use the Component Editor in the QSYS to add custom IP to a Nios II system. The custom IP being added is a RAM module that will utilize the Cyclone V FPGA block RAM. A RAM confidence test, which can be used for board debug, will be developed. The RAM test will run continuously until the user presses KEY1 on the DE1-SoC board, thus generating an interrupt that will terminate the test.

Documentation

This lab sheet is provided to you as a reference and may not contain every step or configuration will be provided. It is your responsibility as a Design Engineer to collect and understand all the requirements needed to complete the lab. This includes obtaining any additional information from previous labs or from any course lecture material.

It is also your responsibility to fully document the lab requirements and results in your laboratory portfolio so that you have enough information to reference later.

Deliverables

This is a two week lab. A video demo for Parts 1-2 due week1 (lab 6). Parts 3-5 due week 2 (lab 7).

Lab Procedure

In this lab, you will use the QSYS System Builder to create the system in Figure 1 below. The component you will be creating is the **Inferred Memory** IP.

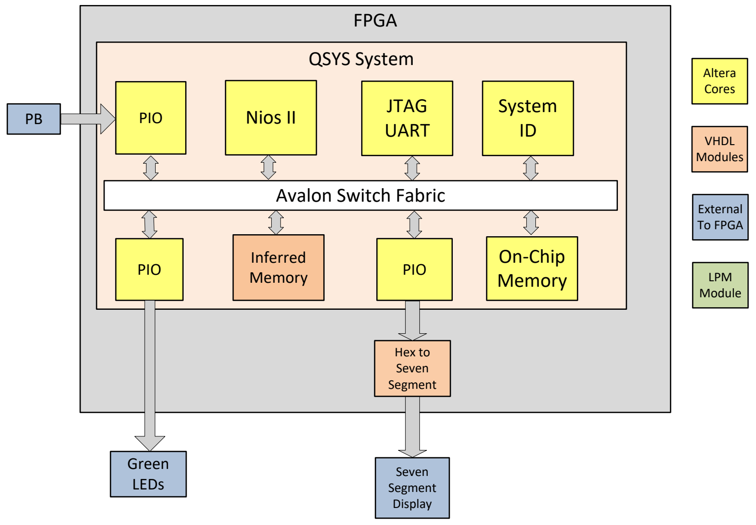

Figure 1: High-level Block Diagram of Design

To create the desired system, start the Quartus software and perform the following steps below.

Please revisit Part2 of Lab6 to refresh your memory on where you left off with RAM verification.

Part 3

Using the memory window to display the memory contents for the RAM, change the data view to display the data as word (4 bytes) by right-clicking in the window and selecting **Format**. Set the column size to 4.

Select an address in the RAM and change the value to 0xABCDEF98. Notice how that memory location's content changes to 0xABCDEF98. This is because the memory window is configured to be addressable as 8 bytes.

Now change the data view to display the data as half-word (column size to 2). Select a RAM address location and change the contents to 0xABCD. Notice that the RAM location changes to 0xABCD but so does the adjacent memory location.

Now change the data view to display the data as byte (column size to 1). Select a RAM address location and change the contents to 0xA5. Notice that the 3 adjacent RAM location changes to 0xA5 with the location we are trying to change.

The display is because the custom RAM implementation is only accessible by word. Any attempt to access the RAM on half-word or byte boundaries causes the data to be replicated on the other byte lanes.

1. You are going to modify the Quartus design you used in Part 1 but before you change anything, make sure you backup your design and then copy your Part 1 design into a new folder called **Lab6/Quartus/part3**.

In this part you are going to create another memory component called **raminfr_be** and add it to your design from Part 1. You can leave the old raminfr in your design.

Create a new file **raminfr_be.vhd** in the directory **ip/ram_be**.

This new RAM component will have a new port call **writebyteenable_n[3:0]** added to it. This port will replace the **write_n** port. These signals will be the port lanes of the memory when you write to the RAM. This means that at least one of and be[*] signals needs to asserted to write into the RAM. For example, to write 32-bit data into the RAM you need writebyteenable_n[3:0] to all be low. To write a byte in the lowest byte line you will need writebyteenable_n[0] to be low. (**VHDL Design Hint**: You want to create 4 independent 4Kx8 RAMs instead of one 4Kx32 RAM). Each 4Kx8 RAM will contribute 8 bits to the 32-bit output.

Open the Component Editor to add your new raminfr_be component.

Compile your new component and verify its operation in ModelSim using a **self-checking** testbench. **Your testbench should verify successful writes for word, half words, and individual byte writes (for a total of 7 tests) to the entire RAM.**

When you have the new RAM module working, add it to your QSYS system.

Select the new component and name it **inferred_ram_be**.

Set the address of the inferred_ram_be to 0x0000_0000. After you set the component address, lock the address.

Make sure the address of the on-chip RAM is locked to address 0x0000_4000.

Generate the system and recompile the design in Quartus

Download your design into the FPGA and verify that new RAM module is addressable by byte, half word and word.

Part 4

1. Open the Nios II Software Build Tools for Eclipse Program and select BSP project and create a new Open the Nios II Software Build Tools for Eclipse Program and create a new NIOS II Application and BSP from Template. Name the application Lab6_Part4_APP.
2. Remember to remove the extra code by right clicking on the BSP project and select **Properties**. Generate the BSP and copy the system.h file from the BSP folder to the APP folder.
3. Create a new file named Lab6_Part4.c

Create a C program that calls a function that will perform a RAM test on the new memory module (raminfr_be). The function call should take three arguments: address, testData and ramSize. Since the RAM test must test word access and byte access, you can write functions for each test. Your final RAM test consists of 4 subtests below:

Test A: Perform an 8-bit accessible test using 0x00 as the test data for the entire RAM memory span.

Test B: Perform a 16-bit accessible test using 0x1234 as the test data for the entire RAM memory span.

Test C: Perform a 32-bit accessible test using 0xABCDEF90 as the test data for the entire RAM memory span.

If at any time the contents read does not match the contents written you should output to the console window.

```
ERROR: Address: 0x????_???? Read:  0x????_???? Expected: 0x????_????
```

NOTE: The read and write data type should be displayed in it native size. For example, if you are reading & writing a unit16 then the display should show the data as uint16.

In addition if at any time the contents read does not match the contents written, the red LEDs should turn on. This program should loop continuously. Be sure to clear the red LEDs between each test.

Your program will continuously loop through the 3 test above. After Test D poll to see if the user pressed the KEY1 button. If the KEY1 was pressed then display a unique pattern on the red LEDs to indicate the test has stopped and display a message RAM TEST DONE to the JTAG UART. Otherwise continue looping through the tests.

Compile and download your program. Put a breakpoint between each write and read cycle. At each breakpoint, check the memory for correctness and manually change the contents of 1 location. Verify that the red LEDs light.

Demonstrate your design and final program testing of your RAM module to receive a signoff.

Part 5

1. You are going to modify the Quartus design you used in Part 3 but before you change anything, make sure you backup your design and then copy your Part 3 design into a new folder called Lab6/Quartus/part5.

Check your QSYS design to make sure the pushbutton interrupt is connected.

Compile the Quartus project and program the Cyclone V FPGA on the DE1-SoC Board to implement the modified system

Create a new Nios II application project named Lab6_Part5_APP and create a new file named Lab6_Part5.c

Copy and modify your Nios II application code from Part 4 to use an interrupt.

When you use the HAL API function to register your ISR function, the size of your program may exceed the 16KB of memory. To get around this we can do one of two things: 1) Add more memory to the design 2) Optimize software to use less memory. For this example, we set the compiler optimization level to size for the BSP project to make the BSP project require less memory. To do this select the BSP project and right click on **Properties**. Then select **Nios II BSP Properties** and set the Optimization Level to **Size**.