# Educational Objective

After completing this lab, you will

1. Learn how to use the component editor to create a custom IP module block that can be instantiated in QSYS.

2. Understand the difference between 8/16/32-bit memory access and what it means to have an access aligned by performing a memory self-test on the memory.

3. Concepts of using functions and interrupts are reinforced.

# Technical Objective

The technical objective of this laboratory is to use the Component Editor in QSYS, to add custom IP (Intellectual Property) to a Nios II system. The custom IP is essentially a device implemented in HDL, and which integrates into your QSYS system through the Avalon bus. The custom IP in this case is a RAM module, that will utilize the Cyclone V FPGA block RAM. A RAM confidence test will be written in C, and used to validate your VHDL RAM module. You can add your custom component (which you will write in VHDL), to your QSYS system in basically the same way you added the GPIO, JTAG serial port, RAM, etc.

In the first part of the lab we will implement the RAM module in VHDL without using the Byte Enable signals from the Avalon bus. When we test this module with our confidence test it will fail. We will also observe the behavior in the memory editor in the Eclipse debugger, and see how it does not work properly.

In the second part of the lab, we will “fix” the VHDL RAM module so that it properly utilizes the Byte Enable signals from the Avalon bus. We will then see that our confidence test passes, and the behavior in the debugger is correct.

# Part 1 – Implement Custom Component That Ignores Byte Selects

## Create Quartus Project and Nios System

Realize the required hardware by implementing a Nios II system on the DE2 board, as follows:

1. Create a new Quartus project using the same method presented in lab one. The Quartus project should be in directory cpet561/lab3\_part1/quartus. As in lab 2, please also create a directory for your source code cpet561/lab3\_part1/src.
2. Create a QSYS system in the same manner you did in lab 1. The only difference at this point is the PIO peripherals. You will need one 8 bit PIO port to drive the LEDs, these will light up when the RAM confidence test you will write indicates the memory is functioning properly. You will also need a one bit PIO input for KEY1, which will start the confidence test.
3. Create directory nios\_system/ip/ram in which you will store the VHDL for the custom RAM component. The VHDL code to implement this ip can be found in the file ram\_ip.vhd on MyCourses, place this code in the nios\_system/ip/ram.
4. Open the component editor by clicking on create new component in the Component Library window in QSYS. After reading the introduction, click on the HDL files tab. Add your raminfr.vhd file.
5. Next click on the Signals tab. You should see all of your port signals listed. The signal types need to be changed to match the Figure 3 below. Clicking on the signal type will bring up a pull-down menu where you will find the appropriate type.
6. As an Avalon Memory Mapped Slave, the Nios II processor will control all of the reads and writes to the RAM. Click on Interfaces tab and set the reset of Avalon interface to the reset signal as shown in Figure 4 below. The reset signal is required to satisfy the Avalon interface but since the component is memory, the reset performs no action on the memory.
7. In the Component Wizard tab you can accept the defaults and then click on Finish.
8. The new component will appear in the system contents window. Add it to your Nios system and rename it to **inferred\_ram**.
9. Set the address of the inferred\_ram to 0x0000\_0000. After you set the component address, lock the address.
10. Set the address of the on-chip RAM to 0x0000\_4000. After you set the component address, lock the address.
11. Select System > Auto‐Assign Base Addresses and note the resultant memory map.
12. Generate the system.
13. Instantiate the generated Nios II system within your VHDL top module along with any other required components. You need to use the **synchronizer** and **debounce** circuits from lab 2 in order to maintain reliable operation.
14. Add the necessary files to the project. Remember that the raminf component is part of the nios\_system so it does not have to be instantiated in the top\_level VHDL nor does the file need to be added to the Quartus II project.
15. Assign the pins needed to make the necessary connections, by importing the lights\_import.qsf.
16. Compile the Quartus II project.
17. Program and configure the Cyclone II FPGA on the DE2 Board to implement the generated system.

## Test RAM IP Component Using 32 Bit Access

In this part you will write the function:

bool ramConfidenceTest(unsigned long \*ramLocation\_ptr, numBytesToCheck).

The input parameters to this function are the location and size in bytes of the RAM you want to check. The return parameter is a boolean which will indicate true if the memory passes the test, and false if any memory location fails.

The function will check the RAM by writing a pattern to the entire RAM, and then reading the entire RAM back to verify that it matches the pattern written. As soon as any location fails to return the correct value, the function exits and returns false. If the entire RAM checks out ok then it returns a true.

Write this function, and verify that indeed the memory functions as intended.

# Part 2 Implement the Byte Selects and Verify That Byte and Half Word Access Now Work

## Test RAM IP Using Byte and Half Word Access

Although the RAM works for 32 bit word access, in this section we will see that it does not work for byte or half word access. We will fix this by implementing the Byte Enable signals, and then verify the RAM IP is fully functional.