ECE318 Laboratory #3

Lab Date: Jan. 29th   Report Due:Feb. 5th

Synchronous Design and Hierarchy Prototyping on the DE2 Board

1. Introduction

This lab reinforces Quartus skills and introduces hierarchy and component libraries. Sequential circuits will be modeled and prototyped on the DE2 board.

2. Prelab

In this lab we are going to set up a shift register from D flip-flop components. The D Flip Flop (DFF) can be found in Quartus by opening a new Block Diagram file and clicking on "Insert Symbol" . The DFF block is found under primitives/storage/dff. In figure 1 below, the DFF component is shown:

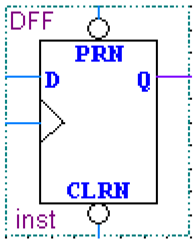


Figure 1. Quartus DFF Component

Task 1:

The first task in the Prelab is to construct an 8 bit shift register by stringing together 8 of these elements. The shift register should have a **preset** input that sets the outputs to all ones, a **clear** input that sets the outputs to all zeros, and a **data_in** input to the **rightmost** flip-flop (i.e., data is shifted into bit 0 of the shift register). Make the diagram in Quartus which will save some time in lab.

```
COMPONENT DFF
PORT (d    : IN STD_LOGIC;
clk  : IN STD_LOGIC;
clrn : IN STD_LOGIC;
prn  : IN STD_LOGIC;
q    : OUT STD_LOGIC );
END COMPONENT;
```

Figure 2. DFF Component Declaration

Above is the VHDL component declaration for the DFF in the Quartus library. You can find information on the DFF by looking in the Help for Quartus (see figure 3). Be sure that you look at the VHDL code as otherwise you will be very confused!

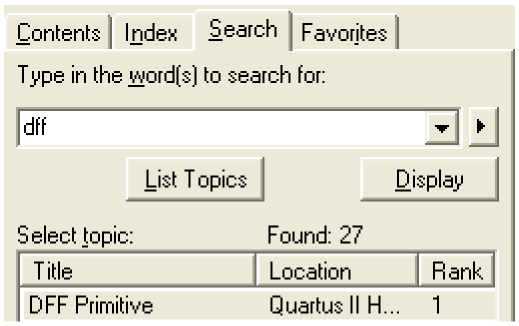


Figure 3. Results of a search in Help for DFF

Instead of drawing each of the DFFs and stringing them together to form the shift register, we can code all this using structural VHDL code.

Task 2:

Using the component from figure 2, write a structural VHDL model of an **8-bit shift register**. As in task 1, the shift register should have a **preset** input that sets the outputs to all ones, a **clear** input that sets the outputs to all zeros, and a **data_in** input to the **rightmost** flip-flop (i.e., data is shifted into bit 0 of the shift register). All 8 flip-flop outputs should be available. You will need to use port type "buffer" for the shift register outputs since they are also inputs (to other flip-flops) in the model.

Think carefully of the entity statement for the shift register before writing the architecture statement which will employ the components.

Be sure to bring your lab manual (I will be giving you a DVD with the files you will need).

3. Procedure

3.1 Chapter 4: Tutorial II

Initial Test

Follow the instructions in Chapter 4 starting on page 73: Tutorial II: Sequential Design and Hierarchy in your lab text. The tutorial files required are on the lab manual CDROM. Copy this entire folder to your own directory (such as E:\118\mydesigns) before opening Quartus.

Complete laboratory exercise 1 in section 4.9 on page 83 which requires you to simulate the circuit.

Debounce the Switch

Follow the instructions starting on page 81 in the lab textbook. Compile, download, and test your design.

Modify the circuit

- Complete exercises 2 and 3 from section 4.9 on page 83 of the lab textbook.

3.2. Shift Register Implementation

- Create a new project and take the block circuit diagram that you have derived in the prelab and insert this in a block diagram. Compile the project and simulate with appropriate inputs to make sure that the shift register is functioning correctly. When you set the clock period, set it to a reasonable value, e.g., 100 ns. Make sure that you experiment with the preset and clear to obtain the correct output.
- Create another new project. Now code a shift register using VHDL structural code and the DFF component as outlined in Task 2 of the prelab. Once you have successfully compiled this, simulate again (you can use the same vector waveform file as the last simulation.)
- You can create a symbol for your shift register by using “File – Create/Update – Create Symbol File for Current File”.

Replace the 8-bit counter in the **tutor2** schematic with your shift register. (note - your shift register must be compiled in a known subdirectory before you will find a symbol for it. If you run into compile problems, you may need to add the vhdl file into your project by using “Add files to Project”.)

Connect the preset input to the pushbutton switch KEY2, connect the clear input to the Vdd component, and set the clock input to the output of the debounce circuit. Use switches SW1 to SW8 for the data_in input. Use switch SW0 for data_in.

To connect the outputs of the shift register to the decoders you will have to rename the bus. You can eliminate the outputs that were connected to the counter. You will draw a bus output from your shift register and label it something like SR[7..0]. Then you can label the input buses of the decoders SR[7..4] and SR[3..0]. (Note: you must use two periods between the numbers to compile correctly – see section 4.4 (pg. 78) of the text for more information on this.)

Compile the new design and download it to the board. Shift in values on data_in to display "d0" and "FA". Once you have completed this, show your lab instructor the result so that it can be verified that your design works.

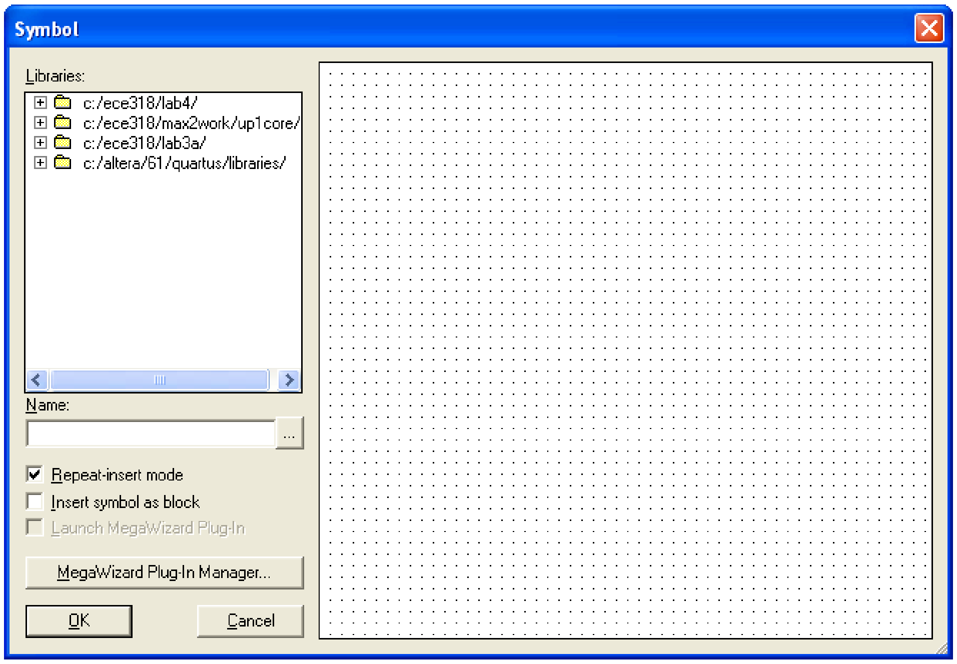
3.3 Laboratory Exercises

In these exercises, we will work with the megafunctions which are already available in MaxPlus. First we will create a ROM module and then an LPM _COMPARE module.

ROM Module

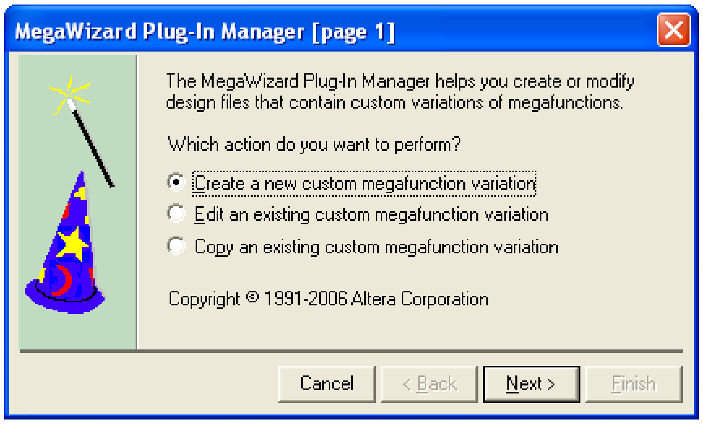
Block Diagram

In the newest version of Quartus, there is a Megawizard which can be used to create the different functions that we need. You can create the megafunction by opening first a Block Diagram File and then clicking on Insert Symbol. This brings up the screen shown in figure 6 below. Click on the “Megawizard Plug-In Manager” and this will bring up the panel shown in figure 7.


Symbol
Libraries:
c:/ece318/lab4/
c:/ece318/max2work/up1core/
c:/ece318/lab3a/
c:/altera/61/quartus/libraries/
Name:
Repeat-insert mode
Insert symbol as block
Launch MegaWizard Plug-In
MegaWizard Plug-In Manager...
OK
Cancel

Repeat-insert mode [✓]

Figure 6. Insert Symbol Screen



The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

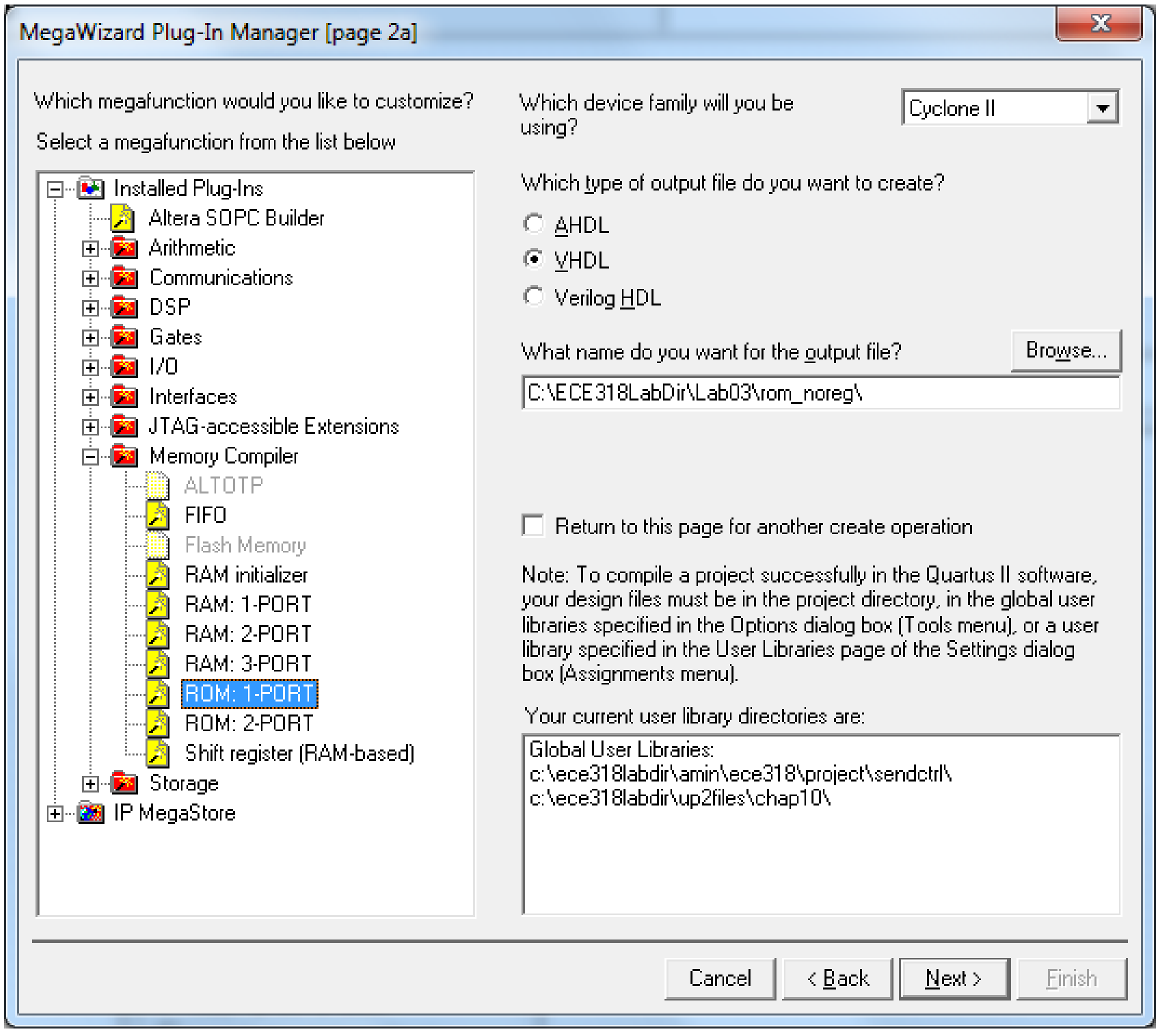
Which action do you want to perform?

[✓ Create a new custom megafunction variation]
Edit an existing custom megafunction variation
Copy an existing custom megafunction variation

Copyright © 1991-2006 Altera Corporation

Figure 7. Megawizard Manager

Next you get to choose a component and for the ROM, the ROM: 1-PORT is chosen as shown in figure 8.


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?

Select a megafunction from the list below

- Installed Plug-Ins
  - Altera SOPC Builder
  - Arithmetic
  - Communications
  - DSP
  - Gates
  - I/O
  - Interfaces
  - JTAG-accessible Extensions
  - Memory Compiler
    - ALTOTP
    - FIFO
    - Flash Memory
    - RAM initializer
    - RAM: 1-PORT
    - RAM: 2-PORT
    - RAM: 3-PORT
    - ROM: 1-PORT
    - ROM: 2-PORT
    - Shift register (RAM-based)
  - Storage
- IP MegaStore

Which device family will you be using? Cyclone II

Which type of output file do you want to create?

( ) AHDL
(●) VHDL
( ) Verilog HDL

What name do you want for the output file? C:\ECE318LabDir\Lab03\rom_noreg\

[ ] Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the Options dialog box (Tools menu), or a user library specified in the User Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

Global User Libraries:
c:\ece318labdir\amin\ece318\project\sendctrl\
c:\ece318labdir\up2files\chap10\

Figure 8. Insertion of a ROM module

You will be prompted to specify the parameters of your ROM module. Since the smallest available memory module is 32 words of memory, we will use this and make the width 8 bits. To find a more detailed description of the function, enter in the function name in “Help – Index” and this will pull up a description of the megafunction.

You will need to specify a “mif” file for the ROM module. You can create this under File – New – Other Files – Memory Initialization File. Fill in the entries so that the data is equal to 4 times the address. The easiest way to do this is to right click on the entries and use “Custom Fill Cells”.

Simulation

Once the symbol is inserted, create input and output ports and simulate the performance. Include a screenshot in your report to verify that the output is as expected. Use hex format for inputs and outputs to make it easier to read.

Synthesis

Assign dip switches to the address ports and use dec7seg components to display the results. Download your circuit to the board and show the working circuit to your lab instructor.

LPM_COMPARE Module

You will be on your own for the creation of this one – the LPM_COMPARE function can be found under the Arithmetic folder in the Megawizard plug in manager.

Block Diagram

Follow the same procedure as above and compare 2 4-bit numbers. You can choose equal to, greater than or less than as an output.

Simulation

Assign input and output ports and simulate the circuit to ensure that it is working correctly.

Synthesis

Assign dip switches to the address ports and use an LED for the output (perhaps the DP LED would be appropriate but any one is fine). Download your circuit to the board and show the working circuit to your lab instructor.

4. Lab Report

Write up your lab report using the usual format. Be sure to include printouts of your circuits and document any problems you had with the lab.

Lab3 updated 2013