CSE 352 Laboratory Assignment 4
Designs in FPGAs (Field Programmable Gate Arrays)

Assigned: October 22nd 2013
Due: October 31st 2013 @ 5:20PM

Objectives

In this laboratory assignment you will design and implement an 8-bit accumulator using the FPGA on your board instead of discrete chips. The accumulator is a register which can be cleared to 0 and to which values can be added. The values to be added will be entered using the switches and the value in the accumulator will be shown using the 7-segment displays. The lab assignment will guide you through the design of the components of this circuit; however it will be your job to figure out how to put them together properly and make it work on the FPGA.

Compiling and Downloading Circuits to the FPGA

You will to first learn how to run the Quartus synthesis tool from Altera to compiler your hardware design to a file that can be downloaded and run on the FPGA. Work through this FPGA synthesis tutorial to learn how this is done. You can reuse the schematics you did for Tutorials 1 and 2, but will have to use the built-in yellow gates instead of the lib370 pink gates in order compile to the FPGA. You will be doing this compilation many times during the rest of the course, so you might as well take your time and understand how it works.

Designing with Verilog

In this lab, you will be combining schematics and Verilog modules to build a relatively complex design. Describing this design just with schematics would be tedious. Using Verilog will allow us to design and debug much more complex designs than if we had to use schematics for everything.

You will be using a subset of Verilog for this lab which uses only **assign** statements to describe logic functions and posedge clock blocks for the register. Remember that Verilog is a hardware description language. Although it may look like a programming language, Verilog is used to describe what the hardware does. Verilog programs don't actually get executed by the hardware like in C or Java; instead, they are turned into the hardware circuit described by the Verilog. Your design will be comprised of several modules that will be connected together using schematics in Aldec-HDL. You can refer to Chapter 4.2 of the textbook for a review of using Verilog to describe combinational circuits (if you have the old version of the textbook).

Before you begin

It is strongly suggested that you create a NEW design in your workspace called “lab4”. Making a new design will make debugging easier and reduce the chance for errors.

Task 1: Hexadecimal Display

We will use the 7-segment displays on the board to display the accumulator value as a hexadecimal number. Your first task is to write an “encoder in Verilog to display a 4-bit binary value using a 7-segment display. If you look to the left side of the board above LEDR5-9 you should see 4 hexadecimal displays labeled HEX0, HEX1, HEX2, and HEX3. They are called HEX because they are generally used to display the hexadecimal values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F for a 4-bit number. For example, the binary value 0001 would be displayed as 1, and the binary value 1111 would F.

This requires turning on the appropriate segments for each hex value. The 7 segments in a hexadecimal display can be turned on and off independently using the 7 inputs to the display. A high value (1) to a segment it turns off, and a low value (0) it turns on. (NOTE THE INVERSION!).

Finally you need to know how the 7 segments in the display are numbered; you can go to page 30 and 31 of the DE1 Board Documentation. There is a nice figure that shows how the 7-segments in the display are numbered (Figure 4.7). You will need to get used to using documentation to find important information about the parts you are working with.

You can now design a module that converts a 4-bit input value to a 7-bit output value that drives the 7-segment hexadecimal display. There are many ways to describe this in Verilog. You could describe each of the 7 outputs as a separate function of the 4 inputs. Or you can treat the 7 outputs as a 7-bit bus and assign all bits in parallel, e.g.

```
assign segments[6:0]  = 7'b0110000;
```

Of course the output value depends on the input value. Recall that we use the ? operator in Verilog to implement conditionals. For example, the following statement assigns one of two values to the 7 segments depending on the value of in.

```
assign segments[6:0] = in == 0 ? 7'b1111110 : 7'b0110000;
```

Here is an example Verilog file that implements a full adder using assign statements and conditional operators. You can look in the textbook Chapter 4.2 to get more information on using Verilog for combinational logic. You should use the following module declaration for your hex display module:

```
module toHex (
      output [6:0] segments,
      input [3:0] number
      );

      <Your Verilog code goes here>

endmodule
```

After you have written your Verilog for this module, you should first test it in Active-HDL using the **toHex_tf.v** test fixture. Once you get that to work, you can compile your design to the FPGA. The first step will be to describe to the Quartus compiler how your signals are connected to the pins of the FPGA – this is done by specifiying the pin constraints in the **.qsf** file. For reference, you can refer to this file pinouts.html to see what pins are connected to what inputs/outputs on the FPGA. You should connect the four inputs of your module to the SW[0-3] and 7 outputs to HEX0[0-6].

Now you can compile your design and load it onto the FPGA. After you have downloaded it, test your 4-bit to HEX encoder to make sure it works by going through all the values 0-15 to make sure the HEX displays the correct value.

Check Off Requirement

1. Demonstrate your hex encoder by cycling through the 16 hex digits. Your LED segment display should correctly display the corresponding hex digit.

Task 2: Add Registers

In this part, you will add an 8-bit register (with reset) to your design. Use the following module declaration for this:

```
module reg8 (
      input clk, reset,
      input [7:0] D,
      output reg [7:0] Q
      );

      <Your Verilog code goes here>

endmodule
```

Add this register module to your top-level schematic, connecting the inputs of the register to 8 switches (SW0-7). You will also need inputs for the clock and reset signals – use KEY0 for clock (like you did for Lab 3) and SW8 for reset. Connect the 8-bit input of your register to two hex displays (HEX0/1), and the 8-bit output of your register to the other two hex displays (HEX2/3). Your top-level block diagram should now instantiate the register and the four hex encoders, and provide the input and output ports for the switches and displays.

Now you need to write a pin assignment file (.qsf) for all of your inputs and outputs. When that is completed, synthesize and implement your design onto the FPGA. Test to make sure that it is working correctly. Does the value of the switches display on the HEX display? Does the value

saved inside the 8-bit register show on the HEX display as well? When you feel it is all working call over your TA for the first check-off.

Check Off Requirement

1. Demonstrate your 8 bit register functions correctly by shifting several numbers from the display for HEX0/1 into the register and that they correct appear on the display for HEX2/3.

Task 3: Hexadecimal Accumulator

Your next task is to turn your register into an accumulator by adding an 8-bit adder/subtractor module (call it "alu") to your design. The 8-bit adder/subtractor module has two 8-bit inputs, A[7:0] and B[7:0], a 1-bit control signal that specifies whether the module should add or subtract, and an 8-bit Sum[7:0] result of the add or subtract. Write this 8-bit adder/subtractor module using Verilog. You should use the built-in Verilog add (+) and subtract (-) operators and let the synthesis tool turn into an adder circuit.

Once you have finished the alu module, add it to your schematic so that it adds/subtracts the input value to the output of the register to compute the next value for the register. Synthesize your new circuit and download it to the FPGA, test it to make sure it works, and then have the TA check you off.

Check Off Requirement

1. Demonstrate your functioning hexadecimal accumulator by shifting in a sequence of hexadecimal numbers. Make sure to also demonstrate that it behaves correctly in the event of an overflow.

Task 4: Decimal Accumulator

You have just designed an 8-bit binary accumulator whose value is displayed as 2 hexadecimal "digits". Change the accumulator to be a 2-digit decimal accumulator. For this circuit, the input and output are two 4-bit decimal digits. That is, the values A-F are illegal on the input digits and cannot appear on the output digits. Thus if the accumulator value is 86 and we add the number 45, we will get 31 as the next result (with a lost carry), not the hex value "cb".

Check Off Requirement

1. Shows that your circuit correctly handles the cases where the switch inputs are 0xA, 0xB, 0xC, 0xD, 0xE, or 0xF.
2. Demonstrate your functioning decimal accumulator by shifting in a sequence of decimal numbers. Make sure to also demonstrate that it behaves correctly in the event of an overflow.

Consolidate Check Off Requirements

1. From Task 1: demonstrate your hex encoder by cycling through the 16 hex digits. Your LED segment display should correctly display the corresponding hex digit.
2. From Task 2: demonstrate your 8 bit register functions correctly by shifting several numbers from the display for HEX0/1 into the register and that they correct appear on the display for HEX2/3.
3. From Task 3: demonstrate your functioning hexadecimal accumulator by shifting in a sequence of hexadecimal numbers. Make sure to also demonstrate that it behaves correctly in the event of an overflow.
4. From Task 4: show that your circuit correctly handles the cases where the switch inputs are 0xA, 0xB, 0xC, 0xD, 0xE, or 0xF.
5. From Task 4: demonstrate your functioning decimal accumulator by shifting in a sequence of decimal numbers. Make sure to also demonstrate that it behaves correctly in the event of an overflow.