Lab #3: FPGA gates, adder and multiplier

Physics 127BL   Winter 2024

Lab report due **Wednesday, January 31, at 11:55 P.M.**

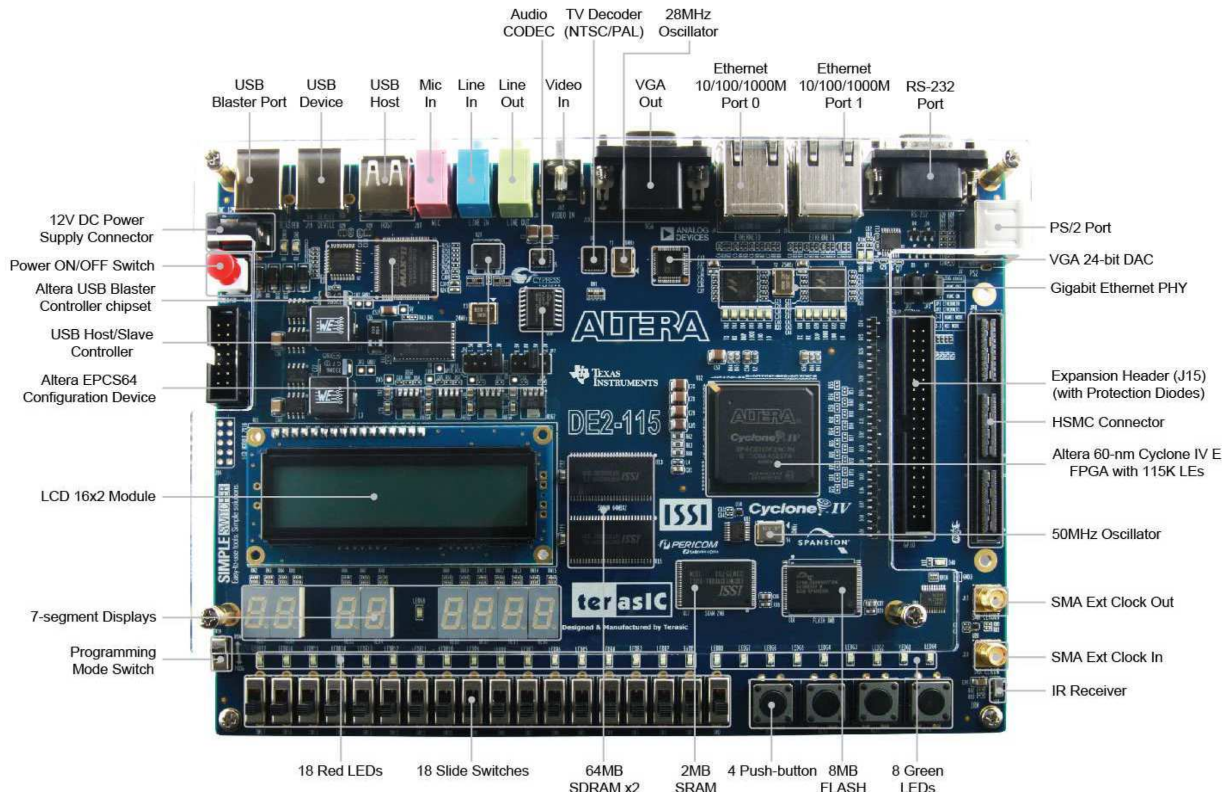
Please read the lab report and homework guidelines handout on the course web page.

Introduction

In this exercise, you will be getting acquainted with the FPGA software and hardware with some simple design problems. You will learn how to edit and save designs, compile them, and then download the code into the FPGA.

1. Create a directory (folder) on your computer that you will use for the entire 127B class. Download the `.zip` files for Lab 3 from the course web page and unpack them in the directory you created.

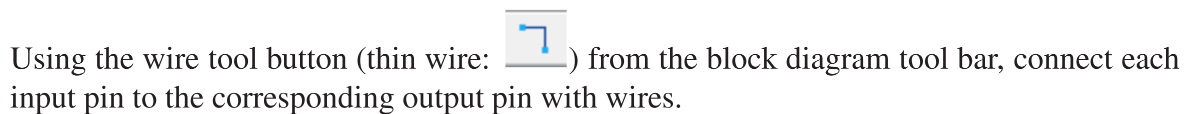
2. Set the red power switch on your DE2-115 board to the off (up) position. Plug one end of the USB cable into the USB Blaster port at the left end of the upper edge of the board, and the other end into your computer. Be careful to use the correct USB port (not "USB Device"). Plug the 12 V power supply into the power connector at the top of the left edge of the board. Turn on power to the board. The blue LED near the power connector should turn on.



1 LED display of slide switch settings

In Quartus, go to `File → Open Project` and navigate to the directory containing your lab files. In the `lab3_part1` directory, there will be a file with a `.qpf` extension. Select and open this file. This is the project file for the first hardware design A dialog box may pop up as Quartus loads the project, asking: "Do you want to overwrite the database for revision..." Select "Yes" to continue. To see the schematic, double click on the the name `LEDtest` in the Project Navigator on the left side of the Quartus window.

1. The input pins to the FPGA are the symbols at the left (SW[7..0]). The output pins on the right side (LEDR[7..0]) are each connected to light emitting diodes. The slide switches and the red LEDs are on the lower left part of the Altera board.

   Using the wire tool button (thin wire: ) from the block diagram tool bar, connect each input pin to the corresponding output pin with wires.

   Click and hold on the right hand side of the input symbol, then DRAG the wire across to the output. Repeat for each set of inputs/outputs. Wires should now connect the inputs and outputs.

   Save your schematic `LEDtest` with `File → Save` from the menu.

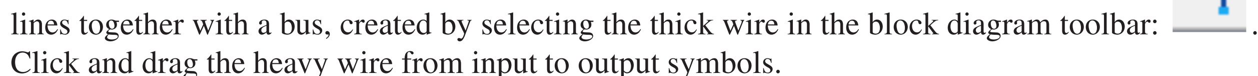
2. Compile your design with `Processing → Start Compilation` from the menu, by pressing the "Play" button from the uppermost toolbar, or by double clicking on the "Play" button next to `Compile Design` under Tasks on the left-hand side of the Quartus window.

3. Load the compiled program into the FPGA using `Tools → Programmer` from the menu. The file to be programmed should be `LiveDesign.sof`, and the program/configure box should be checked. Click on the start button (upper left), and the program will download onto the board. The taskbar on the right hand side should show the progress of the download. Look for any error messages in the message box.

4. Now, test your program! Change the slide switch settings, and check that all 8 LEDs light up when the switches are in the `ON` positions. Are the input pins `HIGH` or `LOW` when the switches are in the `ON` positions? How can you check this? Hint: any pin can be connected to voltage or ground using the **vcc** or **gnd** symbols located under "Primitives" in the Symbol explorer. You can insert symbols by right-clicking and choosing "Insert", or by clicking the Symbol tool (looks like an `AND` gate) in the block diagram toolbar.

2 LED display using bus wiring

We will now see the advantage of using buses, as connecting many wires in a bus is the same amount of work as connecting one wire. The basic project is the same as last time.

Select File → Close Project from the menu. Open the project file from lab3_part2. Note that now there is only one input symbol that represents all eight input pins SW[7..0]. Similarly, the eight output pins are represented by the one output symbol LEDR[7..0]. Wire all eight lines together with a bus, created by selecting the thick wire in the block diagram toolbar: . Click and drag the heavy wire from input to output symbols.

Compile, download to the FPGA, and test as you did earlier.

3 Universal Gates

In this part you will design simple circuits using NAND gates.

1. Load the project from lab3_part3. Note that all gates for this lab can be made using one schematic, 8 slide switch inputs, and 7 LED outputs.

2. Wire two slide switch pins to the inputs of the NAND gate, and its output to an LED. Compile and test for proper operation of the NAND gate. The green LEDs are at the lower right on the FPGA board.

3. Make an OR gate from multiple NAND gates. You can copy the NAND gate by selecting the cursor (arrow) in the block diagram tool bar, right clicking the gate, and using the copy and paste commands. You can also move the gates and input/output pins by dragging, or box-selecting and dragging the objects.

4. Make an XOR gate from multiple NAND gates.

5. Make a DECODE circuit from multiple NAND gates. This circuit takes 2 input bits inbits[1..0], and sets one of 4 output bits outbits[3..0] HIGH according to

   outbits[3..0]=B0001 when inbits[1..0]=B00
   outbits[3..0]=B0010 when inbits[1..0]=B01
   outbits[3..0]=B0100 when inbits[1..0]=B10
   outbits[3..0]=B1000 when inbits[1..0]=B11

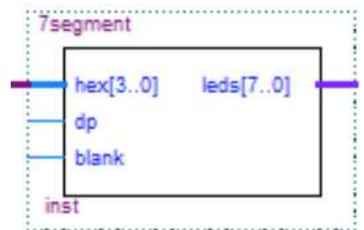
   Connect inbits to two bits of the slide switch, and outbits to 4 LEDs. Compile and test for proper operation.

4 7-segment LED display

This project will take the 8 input bits from the slide switch and display these 8 bits as two hexadecimal digits on the two rightmost 7-segment displays. Because one hexadecimal digit displays 4 bits, you will have to practice wiring to parts of buses, that is, breaking up an 8-bit bus to two 4-bit buses.

1. Load the project from lab3_part4. In the LEDtest module you see a submodule named 7segment. This module takes the 4-bit hex code and converts it into 7 bits driving the 7-segment LED display (the bit that should drive a decimal point, “dp”, does not work, ignore

it). We'll look at this more closely in a moment, but for now wire up the output bus of the top submodule to the output pins of the 7-segment display `HEX0[7..0]`, where the `0` in `HEX0` represents the rightmost 7-segment display. Do the same for the lower submodule and the `HEX1[7..0]` output pins.



2. Connect the `blank` input pin to ground since we don't want to blank the display. Ground `dp` as well. We will just use the gap between `HEX3` and `HEX4` to signify our decimal point later, whenever possible.

3. The upper submodule has its 4 bits of input connected to the lowest 4 input bits, which are labeled as `data[3..0]`. Note that *wires with the same names are connected together within each sheet of a schematic.* Connect the lower submodule to the upper 4 bits of the input `data[7..4]` by naming this bus `data[7..4]`. Do this by clicking this bus and typing "data[7..4]".

4. Compile and program the FPGA. Cycle through the numbers `0` through `F` on the slide switches, and test for proper decoding of the hex into the LED display segments. Note that you will find 3 errors in the coding, which we will now fix.

5. While displaying the `LEDtest` schematic, double click on the submodule `7segment`. A window will open showing text code that defines the logic for the 7-segment decoder using a truth table:

```
1   SUBDESIGN 7segment %converts
2   (
3       hex[3..0]    : INPUT; %4 b
4       dp          : INPUT; %8th se
5       blank       : INPUT; %blanks
6       leds[7..0]   : OUTPUT; %co
7   )

TABLE
blank,hex[3..0]    => leds[6..0]; %T

0,    H"0"        => B"1000000";     %I
0,    H"1"        => B"1111001";
0,    H"2"        => B"0100100";
0,    H"3"        => B"0110001";
```

The beginning lines define the name of the submodule and the input and output connections of the submodule. The logic is defined between the `BEGIN` and `END` commands. Note that there is comment text defining the 7-segment bus number for each segment of the display. The truth table defines what the output should be for each possible input. The first line defines the input and output variables, and the following lines define the input-output relations. The `X` symbol means any number, and is used for the last entry because when `blank` is `HIGH` the display is turned off no matter what hex number is at the input.

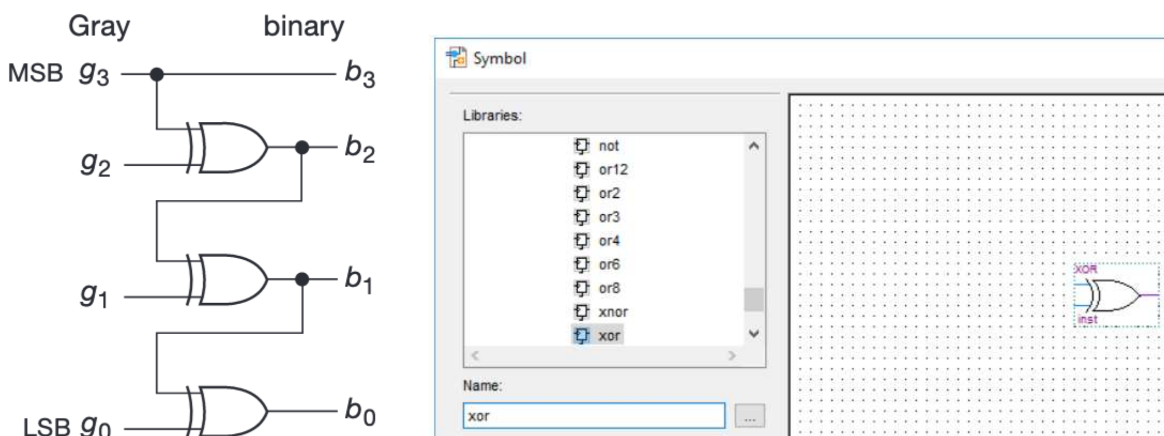
6. Fix the 3 errors in the truth table. Save the file using `File` → `Save`, then recompile, program, and test.

7. Try connecting an input `dp` or `blank` to one of the input bits, for example `data[7]`. Verify proper operation of the `blank`s, and non-operation of `dp`.

5 Gray code translator

In this lab we will convert a 4-bit Gray code into hexadecimal. Load the project from `lab3_part5`.

1. We want to make the circuit on the *left* of the figure below (this is Fig. 10.10B in *Art of Electronics*, 3rd ed., p. 711).



The `XOR` gate may be found by clicking on the device icon (looks like an `AND` gate), on the upper toolbar. When the symbol menu pops up, type "xor" and a list of gates will appear in the box, with the `XOR` gate selected. Click "ok", and the symbol can be inserted 3 times into your schematic. Try clicking on this again, and note the many possible gates that are available in this library. As we will see later, the use of these prefabricated modules makes FPGA programming very powerful.

2. Wire up the gates according to the aforementioned circuit. Note that we have to break out the individual bits of the inputs to connect them to the `XOR` gates, but we have to get them grouped back into a bus to use the 7-segment display. This is done by naming wires and buses as you did previously. Also note how the raw input bits are grouped into a bus and connected to `LEDR[3..0]`.

3. Cycle through all inputs of the Gray code. Note how much easier it is to check all 16 possible inputs now that you only have to change 1 switch at a time. That's the whole point of the Gray code ordering!

6 Adder

In this part, we will make a 4-bit adder by cascading one-bit adders with carry inputs and outputs. Load the project from `lab3_part6`.

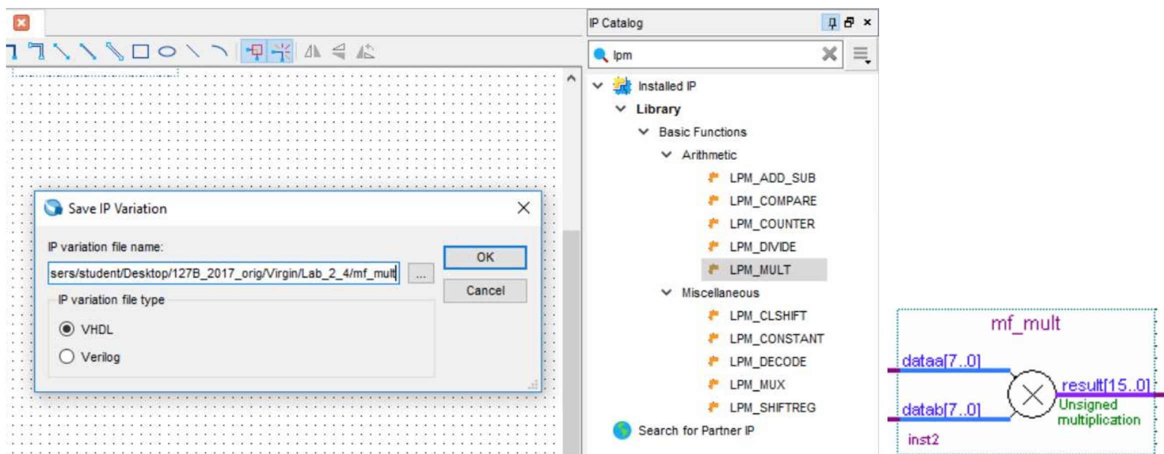
1. First, double click on the `Adder` submodule. The truth table for adding `a`, `b`, and `cin` is given for two out of the eight possible input states. Complete the truth table.
2. Complete the wiring for the 4 bits of the adder stage. Note how the `a[3..0]` and `b[3..0]` inputs are set by the slide switch, and are broken out into the individual bit stages of the adder by naming wires. Also note the output 7-segment displays: the leftmost 7-segment display is `a[3..0]`, the third display is `b[3..0]`, and the rightmost is `sum[3..0]`.
3. Test the adder stage for various inputs. What happens when the output is greater than 15?
4. Explain how the carry output display works. Note that the submodule named `mf_constant` produces a 4-bit bus with the (constant) value B"0001".

7 Multiplier

In principle, one can combine multiplexers and adders to make a multiplier. But when it comes to complex digital circuitry, it is not necessary to reinvent the wheel. In this part we will see how powerful functions can be generated by the FPGA software using "megafunctions." Load the project from `lab3_part7`.

1. We want to build a circuit to square an 8-bit number set by the slide switches. We will build a multiplier with the megafunction tool using the IP Catalog viewer on the right side of the window ("IP" stands for "Intellectual Property"). If the IP Catalog is not visible, open it with `Tools` → `IP Catalog`.

   Under `Basic Functions` → `Arithmetic`, find `LPM_MULT` (or search for it in the search bar). This is the function we will modify to make our multiplier. Double click `LPM_MULT` to start the MegaWizard (see figure below). A text box will appear asking for a name for your new module. Type "mf_multiplier", then click "next".



   The next few dialog boxes define your multiplier. As you want an 8-bit by 8-bit multiplier, just click "next". Click "next" since you want unsigned multiplication. Click "next" again because you don't need it to be pipelined (to be explained later when we learn about flip-flops). Be sure to check *all* the boxes for which files to create. Finally, click "finish" and insert the function — looking like the one on the figure above (bottom right) — onto your schematic by locating it in the Symbol navigator under "Project".

2. Each input to the multiplier is 8 bits, so connect the input slide switches to both inputs of the multiplier to get a squaring function. Name the output bus of the multiplier, and connect this 16-bit bus, 4 bits at a time, to the top four 7-segment display modules. Connect the 8 input bits to the two 7-segment displays at the bottom of the schematic.

3. Compile, download, and test your module.

4. How "big" is this multiplier hardware? Compile the module again, but now in the compilation report dialog box note the number of total logic elements that were used by your design. This is larger than you could breadboard easily, yet how much of the FPGA have you used?

8 Why a Hexadecimal Display?

In previous parts we have been displaying bus data as hexadecimal numbers, not as decimal numbers. This is the easiest way to display bus data, but takes a while to get comfortable with. In a digital computer, binary or hexadecimal numbers are usually converted to decimal in software display routines, so the conversion is "transparent" to the user. Let's try to do it here. We want to show that although it takes some more coding, it's not all that bad because of divider megafunctions. (Try thinking about building a divider yourself!)

Use the project from `lab3_part7` and construct a squaring circuit with the input and output now displayed as decimal numbers. Conversion to decimal can be accomplished using the megafunction divider module that is displayed just above the megafunction multiplier. With a divisor of 10 created from the `constant` megafunction, use the quotient and remainder outputs to compute all of the decimal digits.