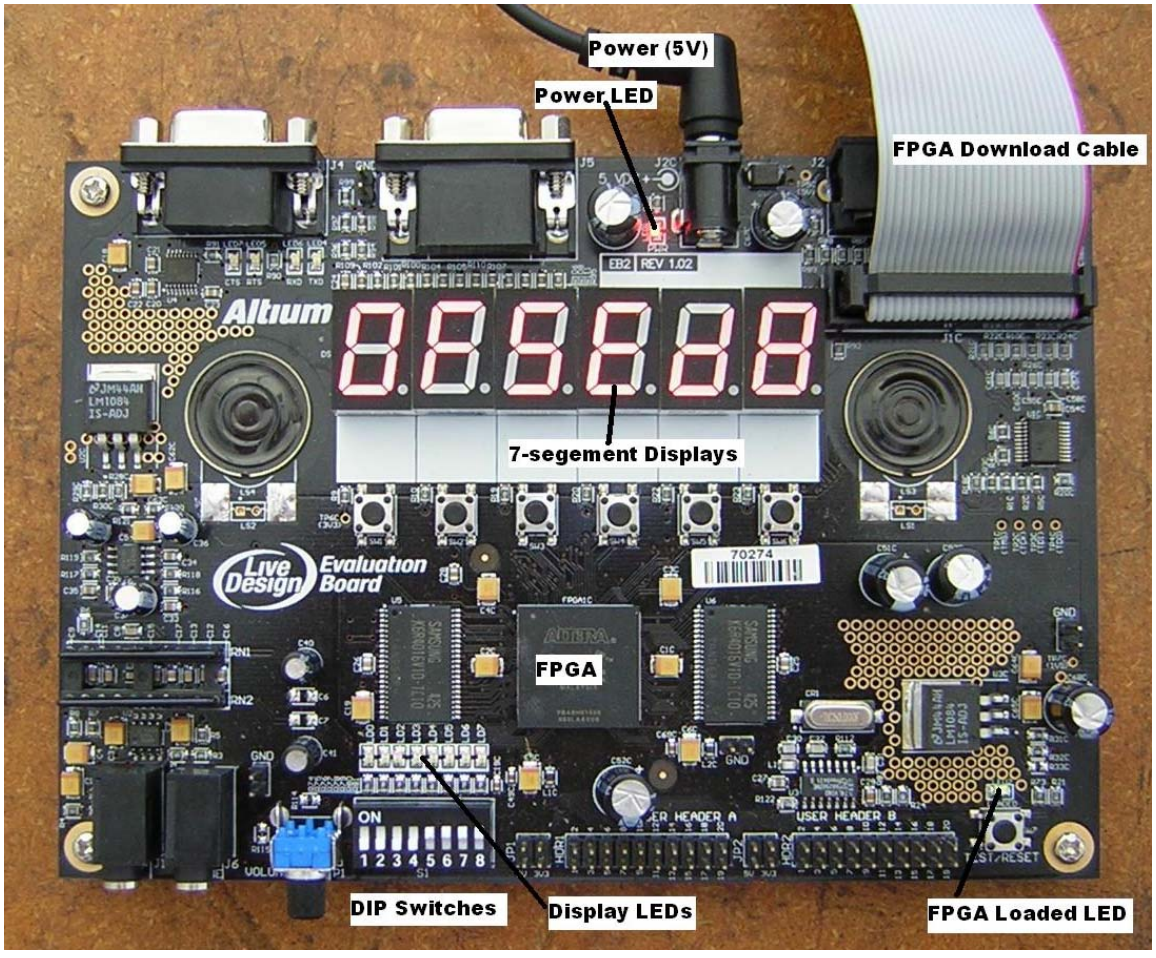
FPGA Lab 1 & 2

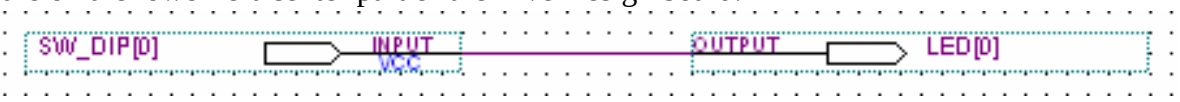
Purpose: In this lab we will be getting acquainted with the FPGA software and hardware with some simple design problems. You will learn how to edit and save designs, compile them, and then program the code into the FPGA.

1) Plug in the power connection to the "Live Design" Board (upper center of board). The red LED near the power connector should light up. Plug in the ribbon cable from the parallel port to the connector on the upper right of the board.
2) Please create your own folder on your computer. You will use this folder for the entire 127B class. Copy the folders for Lab 1 and Lab 2 ("Lab1.1" to "Lab1.2", "Lab2.1" to "Lab2.5") from the lab server file to your folder as well as these instructions.



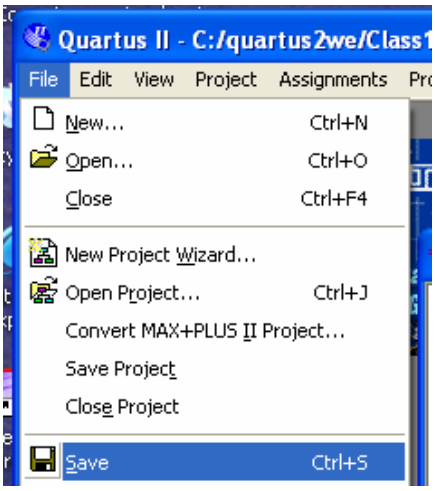
3) **<u>Lab1.1</u> : LED display of dip switch settings.** Open the Lab1.1 folder and double click on the "LiveDesign" icon with the double sheet of paper and blue symbol. This is the project file for the first hardware design, and will start up the program with the proper compiler and programmer settings. The schematic design should be visible under the name "LEDtest".

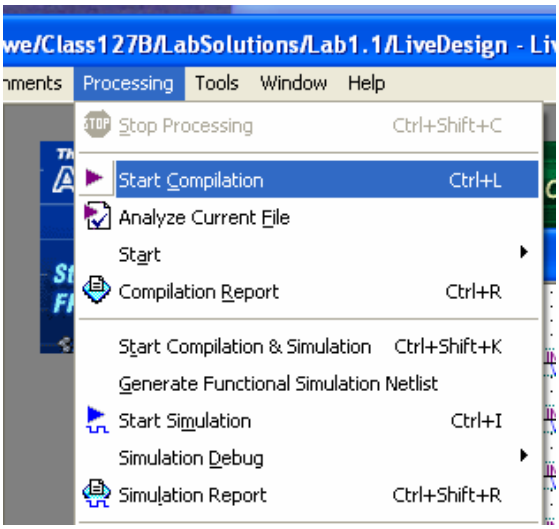
4) The input pins to the FPGA are the symbols at the left (SW_DIP[7..0]); each of the eight pins are connected to a 5K Ohm pull up resistor and a switch to ground. When you slide the DIP switches to the ON position, these inputs then go low (ground). The output pins on the right side (LED[7..0]) are each connected to light emitting diodes through a 270 Ohm resistor to ground. The LEDs produce light when the output pin is high. The DIP switch and the LEDs are on the lower left-center part of the Live Design board.



5) Connect each input pin to each output pin with wires. Click the wire tool button on the left hand side bar (thin wire). Click and hold on the right hand side of the input symbol, then DRAG the wire across to the output symbol. Repeat for each set of inputs and outputs. Wires now connect the inputs and outputs.

6) Save your schematic "LEDtest" with the "File/Save" command.





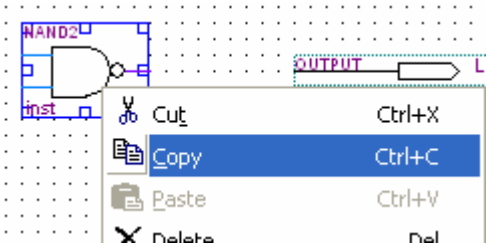
7) Compile your design with the "Processing/StartCompilation" command. After a minute the compilation will finish. Please look for any errors being displayed in the message box at the bottom of the screen. Warnings are ok – there may be many of them because I have defined a lot of input and output pins that we are not using in this schematic.
8) Load the compiled program into the FPGA by using the "Tools/Programmer" command. 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.
9) When the program loads successfully, the green LED at the lower right hand side of the LiveDesign board should light up.

10) Now test your program. Change the DIP switch settings, and check that all the LEDs light up when the dip switch is not in the ON position. Why does ON for the DIP switch correspond to the LED off?

11) **Lab 1.2 : LED display using bus wiring.** In this lab you will see the advantage of using buses, as connecting many wires in a bus is as much work as one wire. The basic project is the same as last time. Open the LiveDesign project in the folder Lab1.2.
12) Note that now there is only one input symbol that represents all eight input pins SW_DIP[7..0]. Similarly, the eight output pins are represented by the one output symbol LED[7..0]. Wire all eight lines together with a bus, created by selecting the thick wire on the left hand side bar. Click and drag the heavy wire from the input to output symbols

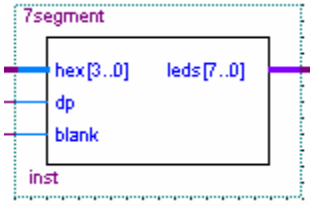
13) Compile, program, and test as in last project.

14) **Lab 1.3 : Gates.** In this lab you will design simple gate circuits using NANDs. Load the LiveDesign project in folder "Lab1.3". Note: all gates of this lab can be made using one schematic, 8 DIP switch inputs and 7 LED outputs.
15) Wire two DIP switch pins to the input of the NAND gate, and its output to an LED. Compile and test for proper operation of the NAND gate.
16) Make an OR gate from multiple NAND gates. You may copy the NAND gate by right clicking the gate, and using the copy and paste commands. You may also move the gates and input/output pins by dragging, or box-selecting and dragging the objects.

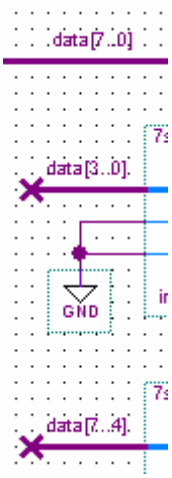


17) Make an XOR gate from multiple NAND gates.
18) Make a DECODE circuit from multiple NAND gates. This circuit takes 2 input bits "inbits[1..0]", and sets high one of 4 output bits "outbits[3..0]" according to
    outbits[3..0]=B"0001" when inbits[1..0]=B"00"
    outbits[3..0]=B"0010" when inbits[1..0]=B"01"
    outbits[3..0]=B"0100" when inbits[1..0]=B"10"
    outbits[3..0]=B"1000" when inbits[1..0]=B"11".
    Connect inbits to two bits of the DIP switch, and outbits to 4 LEDs. Compile and test for proper operation.

19) **<u>Lab 2.1</u> : 7-segment LED display.** This project will take the 8 input bits from the DIP switch and display these 8 bits as two hexadecimal digits on the two rightmost 7-segment displays. Because hexadecimal 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. Load the LiveDesign project in folder "Lab2.1".



20) 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 LED 7 segment display (plus one bit driving a decimal point). 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 DIG5_SEG[7..0], where the 5 in DIG5 represent the rightmost (5th) 7-segment display. Do the same for lower submodule and the DIG4_SEG[7..0] output pins.



21) Connect the dp and blank input pins to ground since we don't want to display a decimal point, and don't want to blank the display.
22) 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].
23) Compile and program the FPGA. Cycle through the numbers 0 through F on the DIP 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.
24) When 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. 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 I have 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 defines input-output relation. 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. Note that after the truth table there is a text definition for the logic to display the decimal point, which also gets blanked.

```
SUBDESIGN 7segment %converts
(
    hex[3..0]    : INPUT; %4 b
    dp           : INPUT; %8th
    blank        : INPUT; %bla
    leds[7..0]   : OUTPUT; %co
)
```

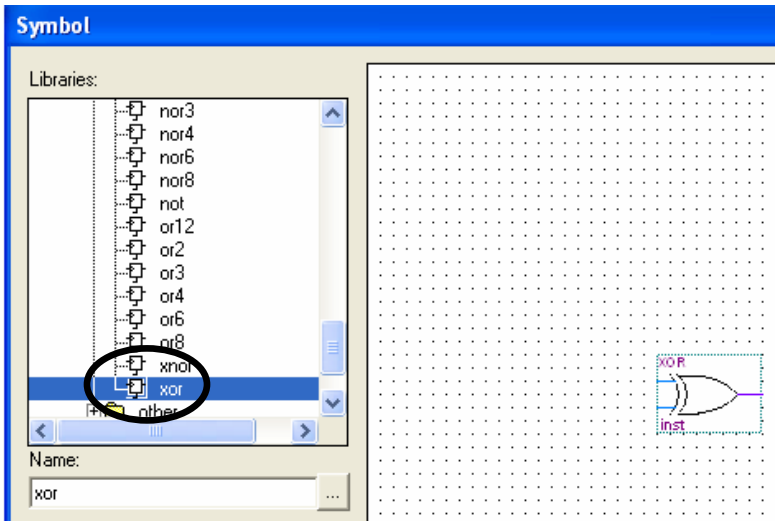
```
TABLE
blank,hex[3..0] => leds[6..0];   %7

0,    H"0"      => B"0111111";   %7
0,    H"1"      => B"0000110";
0,    H"2"      => B"1011011";
0,    H"3"      => B"1001111";
```

25) Fix the 3 errors in the truth table. Save the file using "File/Save", then recompile, program, and test.
26) Try connecting an input "dp" or "blank" to one of the input bits, eg. data[7]. Verify proper operation of the dp and blank logic.

27) **Lab2.2 : Gray code translator.** In this lab we will convert a 4 bit Gray code into hexadecimal. Open Lab2.2.

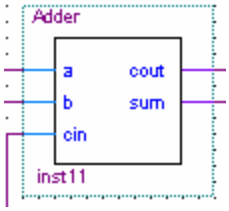
28) We want to make the circuit in Fig. 8.7B, page 480 of Horowitz and Hill. The XOR gate may be found by clicking on the device icon (looks like an AND gate), on the left tool bar. When the symbol menu pops up, type xor (circled at right) and a list of gates will appear in the box, with the XOR gate selected. Click ok, and the symbol may 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.



29) Wire up the gates according to Fig. 8.7B. 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 we saw in the last lab. Also note how the raw input bits are grouped into a bus and connected to the LED[3..0].

30) 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 point of the Gray code!

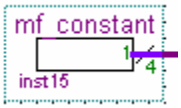
31) **Lab2.3 : Adder.** In this lab we will make a 4 bit adder, using logic we defined in class. Open Lab2.3



32) First, double click on the "Adder" submodule. The truth table for adding a, b, and cin is given for 2 out of the 8 possible input states. Complete the truth table.

33) 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 dipswitch, 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 3rd display is b[3..0], and rightmost is sum[3..0].

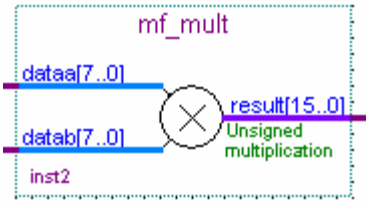
34) Test the adder stage for various inputs. What happens when the output is greater than 15?

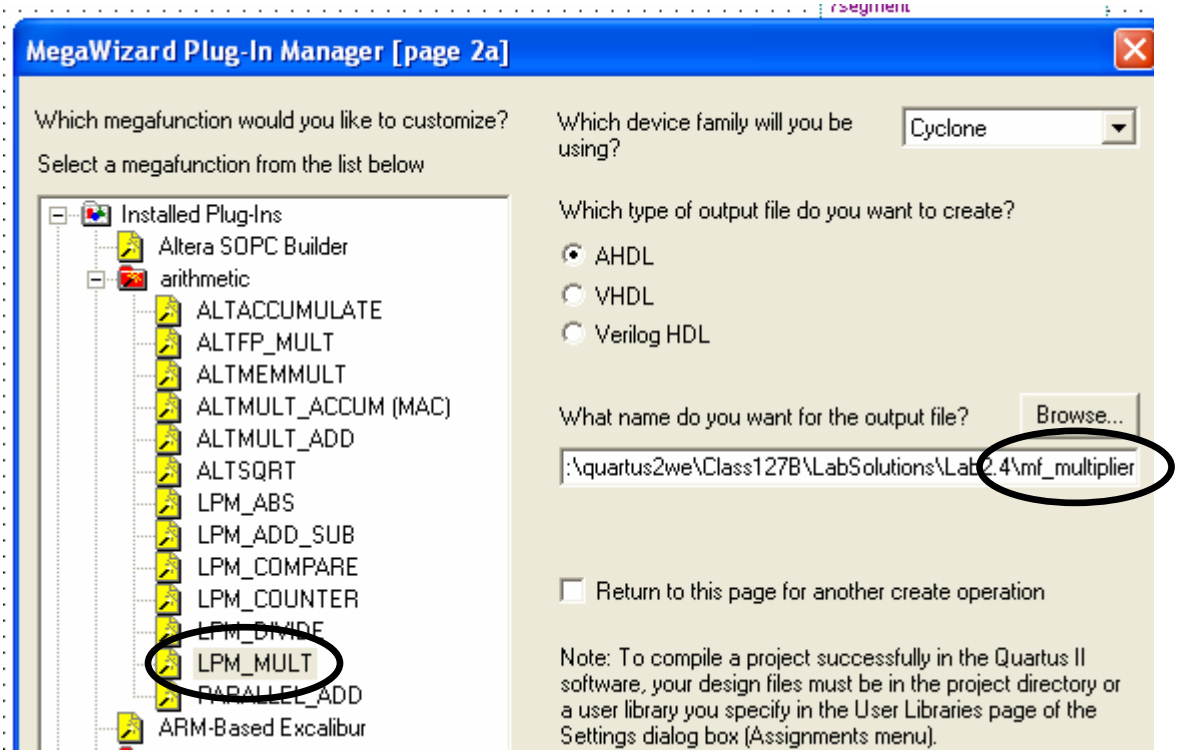


35) Explain how the carry output display works. Note that the submodule named "mf_constant" produces a 4 bit bus with the (constant) value B"0001".

36) **<u>Lab2.4</u> : Multiplier.** In class we discussed how, in principle, to make a multiplier. But why should you have to make one of these when engineers have already figured out how to do it really well? In this lab we will show how very powerful functions can be generated by the FPGA software using megafunctions. Open up Lab2.4

37) We want to build a circuit to square an 8 bit number set by the dipswitch. We will build the multiplier with the megafunction tool, which can be found by selecting the symbol tool (AND gate symbol on left tool bar). In the symbol dialog box, press the megawizard plug-in manager button. Then select next since we want to <u>create</u> a function. In the left box, click arithmetic, then LPM_MULT to choose the multiplier function. Enter a name of the function, say "mf_multiplier" in the text box on the middle right hand side, and 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 do flip-flops). Finally, click finish and insert the function onto your schematic.





38) Each input to the multiplier is 8 bits, so connect the input dipswitch 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.

39) Compile, program, and test your module.

40) 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?

41) **<u>Lab2.5</u> : Why a Hexidecimal Display?** In the previous labs 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 is takes some more coding, it's not all that bad because of divider megafunctions. (Try thinking about building a divider yourself!)

42) Use lab2.4 and construct a squaring circuit with the input and output now displayed as decimal numbers. Conversion to decimal may be accomplished by 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.