Intro to Digital Logic, Lab 1
An Introduction to Verilog and Digital Components

Lab Objectives

Read the whole lab first before starting on any work. The first lab for EE 271 will introduce you to the Altera's Terasic DE1-SoC Development board and our primary design tool this quarter, Altera's Quartus Prime edition version 17. Both of these components are very important for all future labs so please pay attention and do not rush through this first lab.

If you have any questions at any point in the lab, make sure you have read the entire lab thoroughly. If you still cannot find the answer you should ask your TA. And as always, there are no dumb questions, please ask them; you don't want to damage any of the expensive hardware that is provided for you.

Laboratory Design Kits

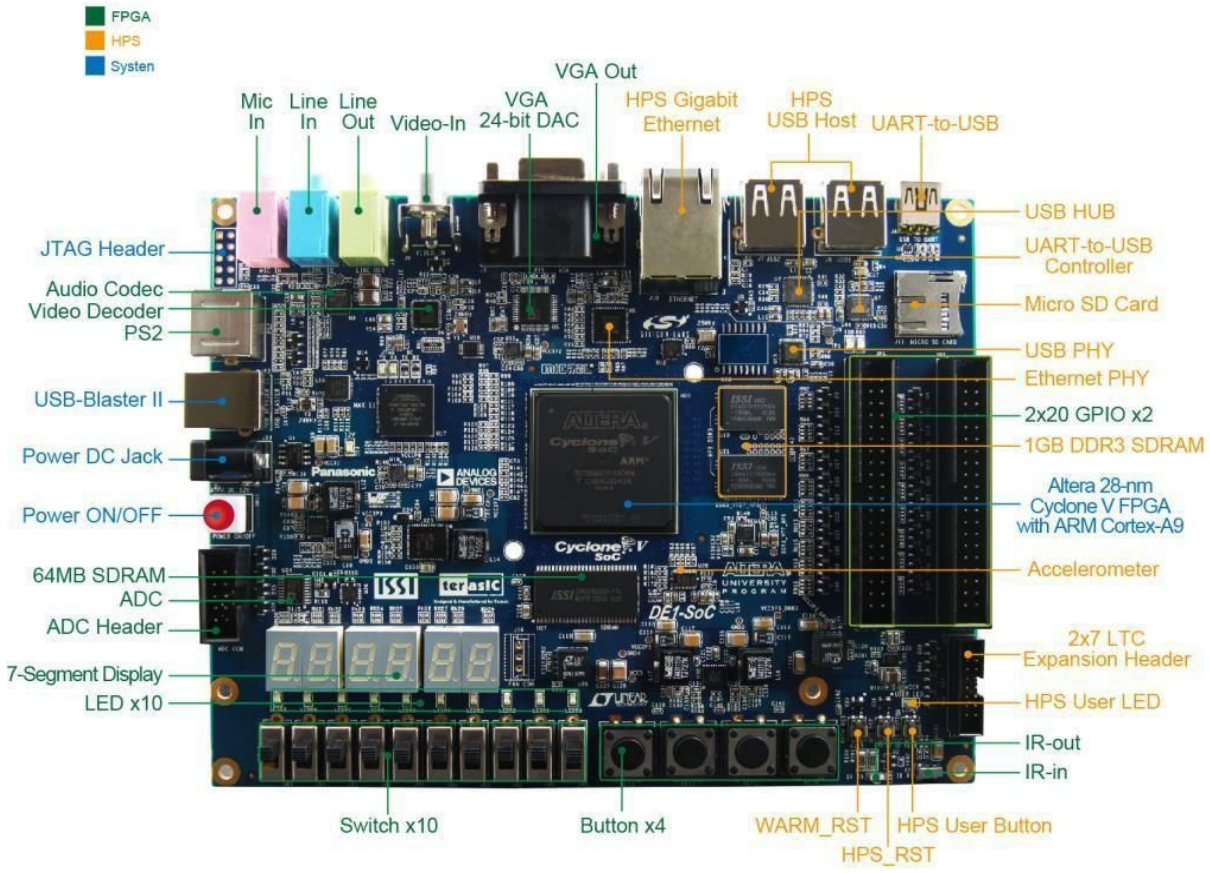
The design kits contain the resources you need to construct most of the circuits assigned to you over the course of the quarter. You are responsible for your lab kit and we expect that you will return the kit in good working order with all pieces intact. This especially applies to the chips, which have very delicate pins. Use care when extracting them from the solder-less breadboard.

- The Altera/Terasic DE1-SoC Development Board (the "DE1") with UW Proto board attached on the top. The green Proto board has a white solder-less breadboard on it.
- A black power cord for powering the DE1.
- A grey USB cable for hooking the DE1 to a host computer.
- A set of TTL logic chips. All of these chips are Dual-In-Line Packages compatible with your breadboard.

Please make sure all of the above are provided for you in your kit. For the final project, you may decide you need additional chips, switches, lights, and other devices. If so, you can purchase this at EE stores, Fry's in Renton, or on-line.

We are aware that accidents can happen and the pins on chips may have already been weakened from years of use. Typically if you break a pin the chip will be replaced and the damage will be forgiven; however, in cases of gross negligence, you will have to pay for a replacement. Take care not to short out your board by connecting Ground to Power. This results in a large current which can quickly burn all the components of the board, including the board itself. Finally, be careful not to snap off wires inside the I/O connectors or the holes in the solder-less breadboard.

Altera/Terasic DE1-SoC Development Board

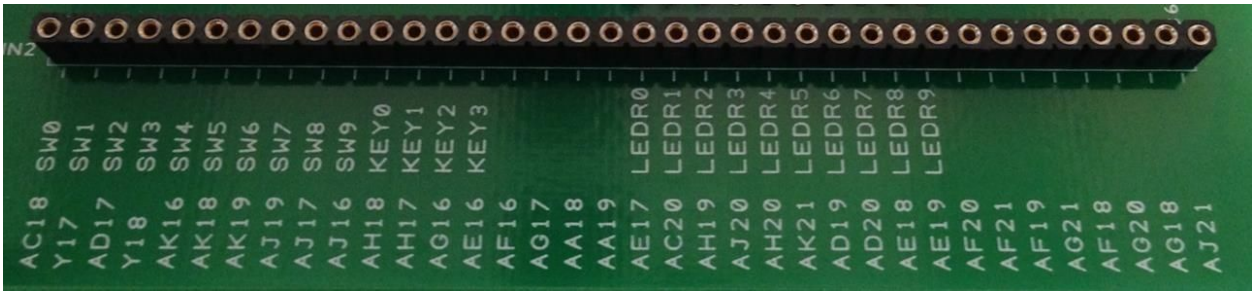


The picture above is a diagram of the DE1 with most of the major components highlighted. For the first few labs, you will be using the various input/output devices located directly on the board such as the switches and LEDs. Further details will be provided in each lab. Take note of the large FPGA (Field Programmable Gate Array) that is in the middle of the board. Later on, in the quarter, you will be programming this and directly interfacing with devices on the board. Think of it like a universal logic unit that all the devices can talk to. For now, there is a program loaded into the FPGA to allow you to use the input/output connectors on the board to make the earlier labs easier.

The FPGA

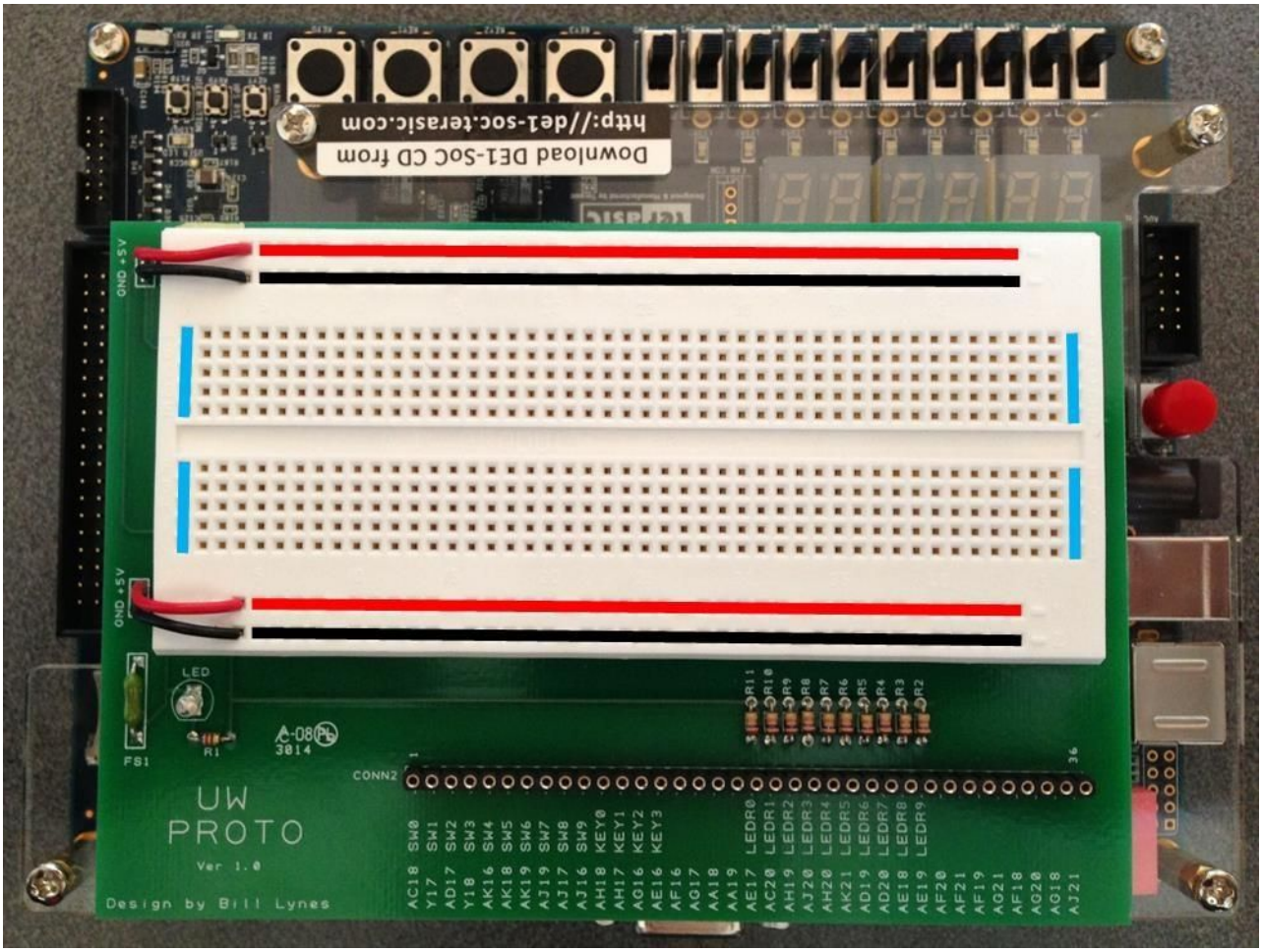
"FPGA" is an acronym for Field Programmable Gate Array. Essentially it is a large array of logical elements that have been connected together. However, in an FPGA the connections between these logical elements can be programmed and reprogrammed, which means the FPGA can be used to build many different kinds of hardware all on the same chip. If you look at the lights when you turn on the board, you will see that it has a premade display running. This is a program that we have written and programmed into your DE1's non-volatile memory. By programming into the non-volatile memory, each time the board is turned on this startup program will be loaded, regardless of what you did previously.

Switches, Keys, and LEDs



The image is a zoomed in picture of the prototyping board. As you can see, each little hole has a corresponding pin mapping (more on this in later labs) and a pre-programmed functionality. Remember the program we put in the non-volatile memory? Well in addition to having the visual display, the program also maps those holes to the indicated switches, keys, or LEDs. So for example, if I hooked up the hole labeled "AJ16 SW9" to the hole labeled "AE19 LEDR9" with a wire, I would be controlling the red LED labeled LEDR9 on the DE1 board with the switch labeled SW9.

The Solder-Less Breadboard



The white solder-less breadboard attached to your DE1 is where you will be building your first few circuits. Notice the red and black wires going to each of the rows at the top and bottom of the board. The red wire denotes VDD (+5 Volts) and the black wire denotes Ground (0 Volts) Never directly connect these two rows together in any way. You should have both a VDD and Ground at the top of your board and a set at the bottom. Even though there are spaces between the rows all of the holes are directly connected to each other. This means you have rails of VDD and Ground both at the top and bottom of your board.

In addition to the rails of VDD and Ground provided for you on the board there are two 48column grids of holes separated by an indented divider. All 5 holes in one column are connected to each other; however, the column on the top of the divider is not connected to the column below the divider. All of these connections are underneath the board so you cannot see them. So remember, if you connect VDD into one of the five holes in a column, all five holes now have VDD running across them.

The red lines mark where VDD runs across the board, and the black lines mark were GND runs across the board. The holes on the board are interconnected as the blue line shows, up and-down for each 5-hole segment.

Wiring your Solder-less Breadboard

Wires should be inserted as perpendicular as possible to the breadboard and should slide in and out with just a little force. If you find that there is a lot of resistance in either direction, first review the wiring guidelines below, and if that doesn't work, then please talk to the TAs.

Do not try to force anything larger than stripped wires into the holes, because this could damage the breadboard (at great cost).

Before doing any work on the breadboard such as wiring and inserting/removing chips, be sure the power is OFF. That is, unplug the power connector while you are constructing the circuit. After you have finished wiring up your design and before you turn on the power, double check the power and ground connections.

Wiring Guidelines

Wiring your circuit together can often feel tedious, especially in the beginning. However, if you are patient and wire your circuit nicely, you will find that you will spend a lot less time tracking down wiring errors. To aid you in this, here are a few tips to consider while wiring up your circuit. If anything is unclear, ask your TA for an example.

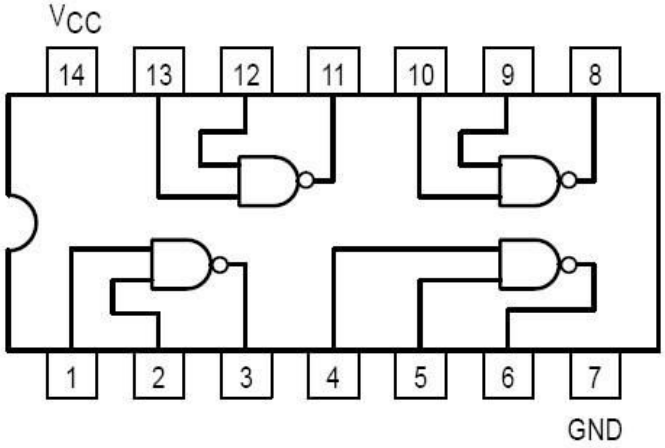
- Make sure your wires are stripped carefully. This means that when you put the wire in your breadboard, there shouldn't be any un-insulated wire visible and the wire shouldn't crunch against the bottom of the board.
- Your wires should always be nice and straight. There should be no twists or kinks in them, as they can cause your board to short out when you insert them in your breadboard.
- Arrange the chips on the breadboard so that only short wire connections are needed. Put tightly connected chips closer together. Chips will only fit with one set of pins on one side of the center divider and another set of pins on the other side of the divider.
- Do not make a jungle of wires. Long looping wires that go way into the air are easy to pull out (a hard bug to find later when the circuit doesn't work as intended).
- Try to maintain a low wiring profile so that you can reach the pins of the chips and so the chips can be replaced if necessary. The best connections are those that lie flat on the board. Try to avoid wiring over any chips so that chips can be removed easily and replaced.

Chips

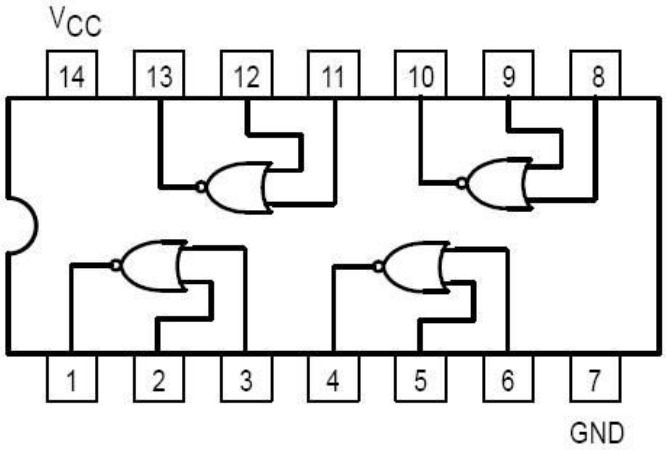
Several of these parts can be found in your kit. The chips are designed to straddle the trough in the breadboard with two rows of pins. One row is lined up on one side of the trough on column "e" and the other row lines up with column "f" on the other side of the trough. Don’t insert the chips with both rows of pins on one side of the trough as this will connect corresponding pins across the chip and will lead to non-functional designs and quite possibly damage the parts. All the chips have their catalog part number stamped on their top (when

oriented so that you can read it, pin #1 will be on the bottom left and the numbering will proceed counter-clockwise from there) something like 74LSXX, where the XX will be a two digit number that will differ on each chip type. The XX specifies the function of the chip, while the rest of the code specifies the implementation technology. The figures below use the number XX to define the pins of each chip. The ground pin is usually labeled GND and is the rightmost pin on the bottom row. The power pin is usually labeled VCC or VDD and is the leftmost pin on the top row. Note that some portions of the code might be different - for example, LS might be something else for a different vendor. However, for our purposes the only thing important is the value in the XX.

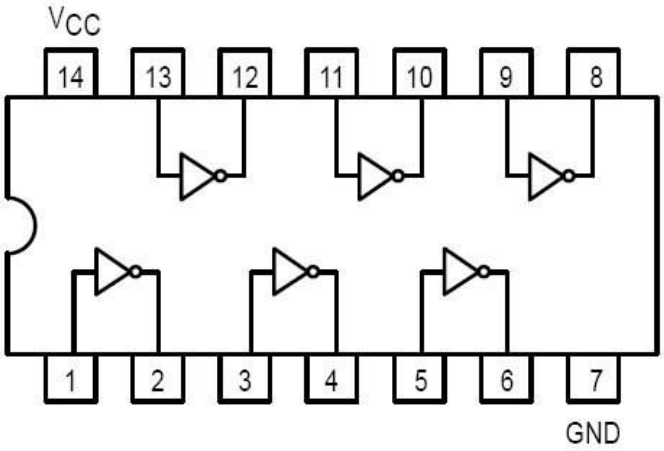
’00 – 4 2-input NAND gates:



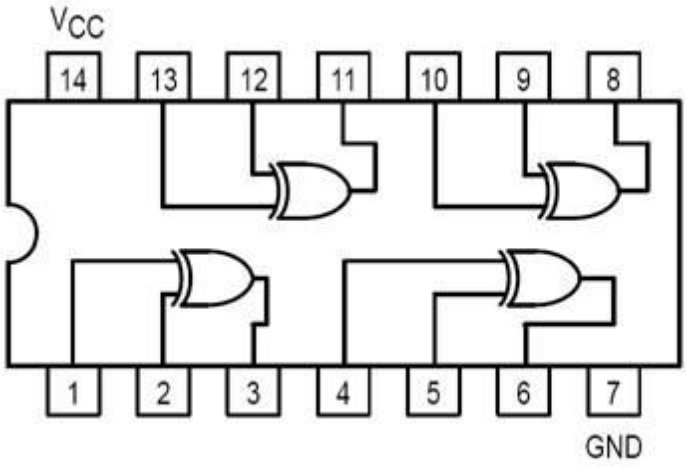
’02 – 4 2-input NOR gates:



’04 – 6 inverters:



’86 – Quad XOR gates:



Using Quartus II Software

Most of the designs in this class will be done through the Altera Quartus II software. The Quartus II web edition release 17.0 is preloaded on the machines in the department, and you can do all the work on these PCs. However, if you have a PC of your own that you would like to use, you can install the software for free. To install the software, go to http://dl.altera.com/?edition=lite , download the free web edition and install it. Note that you will have to register to be able to download the software.

First, make sure Select Edition is “Lite” and Select Release is “17.0”, then download the 5.8 GB tar file under the “Combined files” tab. Extract the tar file using a program like 7-zip and run the QuartusLiteSetup-17.0-windows.exe file. When it asks for the components to install, make sure you select each of these:

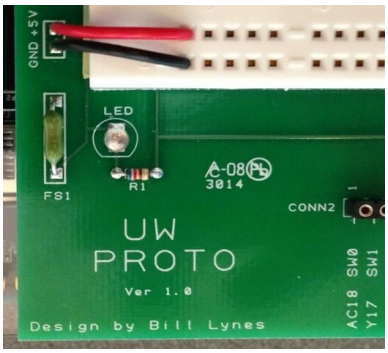
- Quartus Prime Lite Edition (Free)
- Devices: Cyclone V
- ModelSim: Intel FPGA Starter Edition (Free)

When the software is done, make sure to install the USB blaster driver. Run Quartus next, and if asked about licensing just run the software (we use the free version, so no license required).

Warm-Up Task – Using Wires, Switches, Pushbuttons, Gates, LEDs

Let's explore the elements of the DE1 board and build some basic circuits.

You will need to use the black power cord to power the board. Plug the cord into an outlet and into the "Power DC Jack" socket just above the red on/off button. The grey USB cable is used to connect the "USB Blaster II" to the computer. For each of the next steps, **make sure the board is off when you insert or remove wires or chips**. This will help avoid accidental shortcircuits when you move elements around. **Be careful that you never let a wire connected to VDD touch a wire connected to Ground or you will short out your board. If that ever happens, or you smell odor/see smoke, turn off your DE1 board immediately and pull the power.**



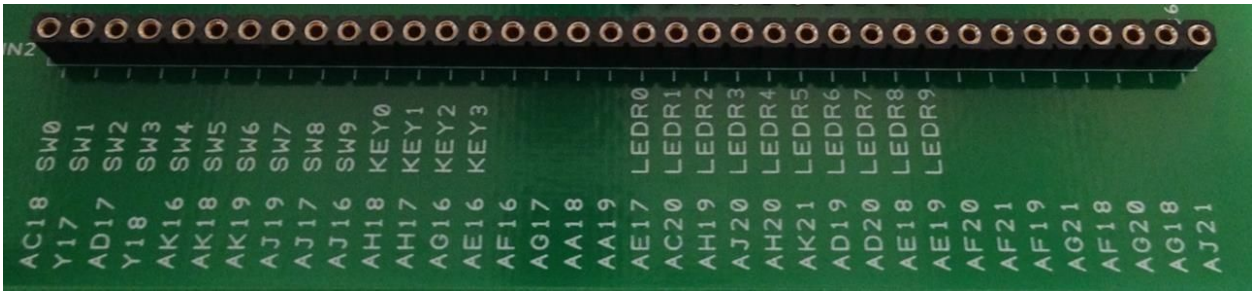
There is a fuse (FS1) and an indicator light (the blue LED) on your prototyping board. If you do accidentally wire up GND to VDD, you will mostly likely blow the fuse "F1" and the LED light will go off.

LEDs as a Logic Probe

Throughout the class, we will use the switches, pushbuttons, and red LEDs to act as the inputs and outputs of our circuits. However, first thing we need to do is figure out the logic

values of various signals in the design. The best starting point is to figure out what values turn on the red LEDs.

Turn off the power before making any connections. Hook a wire from the GND rail to the hole on the Proto board going to LEDR0, and another wire from the VDD rail to the hole on the Proto board going to LEDR1.



Turn on the power, and see which light turned on – LEDR0 or LEDR1? From this, you can figure out whether a TRUE (VDD) or FALSE (GND) value sent to the LEDR's turn on those LEDs.

Note that this use of LEDs can be very helpful as you develop your circuits – if you ever wonder what the value is of a given wire, just hook it up to an unused LEDR and whether the LED lights up or not will tell you the value of that wire.

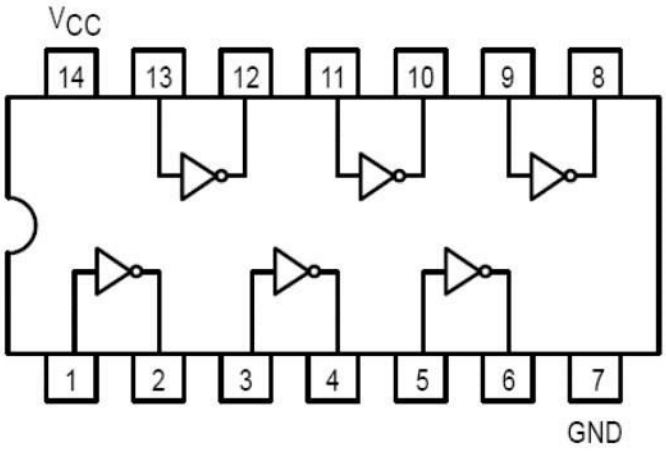
The logic values of switches and pushbuttons

Once again, turn off the power before making any connections. **We won't repeat this again, but you should always remove power before making any wiring changes and only reconnect it after you've double-checked that you've wired things correctly.**

On the Protoboard, use a wire to hook SW9 (the leftmost slider switch) to LEDR9 (the red LED just above it), and another wire to hook KEY3 (the leftmost pushbutton) to LEDR0 (the rightmost LED). Turn on the power, and experiment with SW9 and KEY3 on the DE1. Since we know what kind of value lights up the LEDs from the previous section, figure out which position of the slider switch outputs a TRUE – up or down. Also, figure out which position of the pushbutton outputs a TRUE – pressed or unpressed.

Place an inverter (NOT gate) between the switch and LED.

This is your first chip. Insert the '04 inverter chip into the empty region straddling the divider on the breadboard. Each pin of the chip is now connected to a 5-hole column on the breadboard. Be sure to wire up VDD and Ground - chips need power in order to function, and if you don't provide them with a ground they won't function correctly. Make sure you connect the correct pins: Ground is pin 7 and VDD is pin 14 on the '04 chip.



The best way is to look at the chip maps to determine the orientation and chip inputs and outputs for any of the chips we have provided for you. This '04 Chip has 6 inverters in one package - pick one of them to use. Hook up a slider switch to the inverter's input, and an LED to the inverter's output. Now turn on the power and flip the switch a few times – does it work correctly?

Now hook the output of that first inverter to the input of another and hook that other inverter's output to an LEDR. Turn on the power, and again play with the switch – is the system working properly? Since there are now two inverters in series (a double negative, which cancels out), the output of the second inverter should be identical to the input of the first inverter, which is the value from the switch.

Connect a two-input NOR gate to two switches and one LED.

Use two switches and one LED to explore the functions of a NOR gate. We won't give you detailed explanations on how to do this final task since you should be able to apply your knowledge from the previous tasks to complete this one. Think about how a NOR gate works and see if you can find all four combinations.

General Notes before working on this Lab

1. **For Task 1, some chips provided to you may be malfunctioning. Therefore, if you firmly believe that your circuit design is correct, please replace the chip and retest your solution. (We've provided you with two identical chips for each type of logical gate).**
2. **For task 1, some wires may also be malfunctioning. You can test the connection and check the wires using a multimeter.**
3. **Read over the lab handout CAREFULLY so that you don't make "dangerous errors" including short-circuiting the breadboard, etc.**
4. **When working on Task 2 and Task 3, make sure to follow the commenting guideline. A significant amount of grade will depend on the commenting style.**
5. **For the lab report, make sure to follow our lab report guideline.**

Assigned Tasks

Task 1: Wiring digital components and LEDs on a breadboard

Design a full adder that has 3 inputs A, B, Cin, and 2 outputs: sum and Cout. Create a truth table from which you should derive the output equations in the minimal form. Draw a schematic diagram of the full adder and wire up the circuit on your breadboard using the chips provided to you in the lab kit. Test your circuit and provide a truth table that shows the output obtained during testing.

Task 2: Implementation of the full adder on Quartus Prime

For this task, you should follow along a series of video tutorials that will walk you through the steps of creating your first design using SystemVerilog and simulating the design in ModelSim. For your reference, the source code used in the tutorials is in the appendix of this document.

Please follow the tutorials in the following order:

1. Launch the Quartus Prime software.
2. Create a project from scratch. Please follow the steps in the following videos and use the same project name as in the video https://youtu.be/iLbmSTG7bpA
3. Implement the full adder using SystemVerilog and simulate it using ModelSim. Please follow the following video: https://youtu.be/BcvclrqZ2fc

   Please note that in the video when it refers to compiling the project for the first time, it may give you a compilation error. If you run the video for few more seconds it 'll tell you about setting the top level modules so that the program compiles.

   - After you successfully simulate the full adder, save a screenshot of the simulation. You can do that by going to File->Export->image and then save the image. You will need to include this image in your lab report as a proof that you went over the tutorials.
4. Mapping a SystemVerilog design to an FPGA. Please follow the steps in the following video and save an image of the simulation. https://youtu.be/mnZt2iNNfp4
5. Loading the design onto the FPGA. Please follow the steps in the following video and test your full adder on the DE1_SoC board and verify that it matches the truth table.

   https://youtu.be/uMxA3VcS3f8

After you are done with this task, close the current project before moving to the next task. You may quit Quartus at this point.

Task 3: Design a 4-bits adder

In the previous tasks, we created an adder that can add 3 bits. However, in reality, the numbers consist of multiple bits and therefore we want to extend our design accordingly. Your task is to use the "fullAdder"

SystemVerilog module to extend the design to 4 bits numbers as shown in the following figure

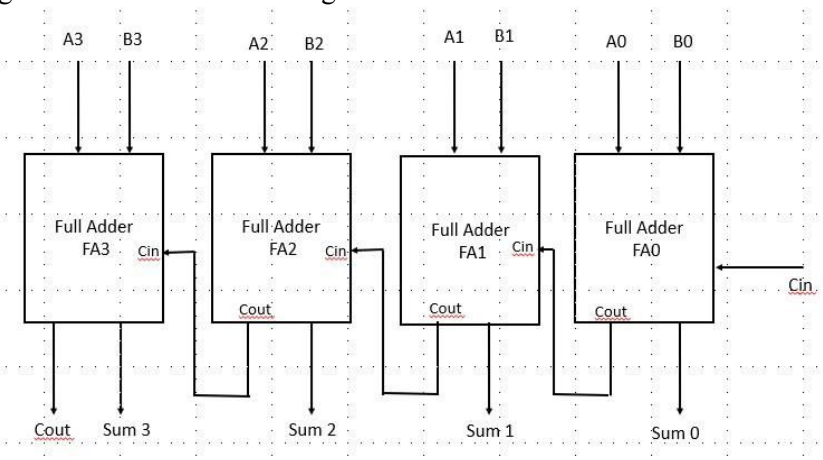


Figure 1. 4-bits Adder

To get started, you need to do the following:

1. Instead of creating a project from scratch, make a copy of the lab1 folder that was created in task 2 and call it "lab1a". Launch quartus by double clicking on DE1_Soc.qpf file under the lab1a folder.

   If you followed all the steps of task 2, you should have two .sv files: DE1_SoC.sv and fullAdder.sv

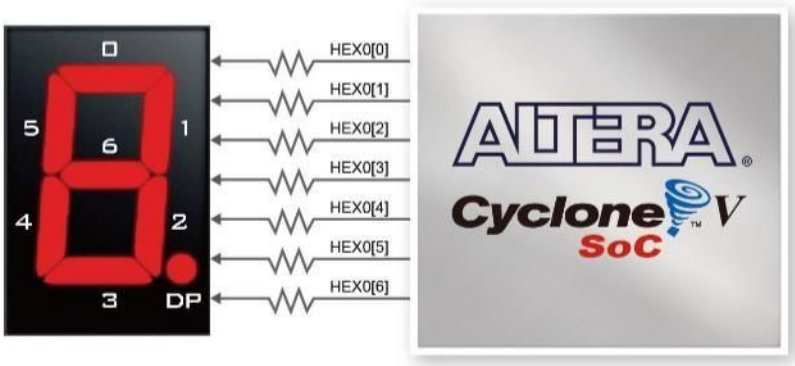
```
module fullAdder4 (A, B, cin, sum, cout);

    input logic A [3:0];
    input logic B[3:0];
    input logic cin;

    output logic sum [3:0];
    output logic cout;

    logic c0;

    fullAdder FA0 (.A(A[0]), .B(B[0]), .cin(cin), .sum(sum[0]), .cout(c0));
    fullAdder FA1 (.A(A[1]), .B(B[1]), .cin(c0), .sum(sum[1]), .cout(cout));

    //continue instantiating the fullAdder module and add any necessary logic to make it 4-bits

endmodule
```

2. Create a new SystemVerilog file and call it fullAdder4. This module will instantiate the fullAdder module created in task 2. To help you get started, here's a skeleton of fullAdder4 please modify it accordingly. Make the fullAdder4 module your top level entity and simulate it in ModelSim and save a picture of the simulation for your lab report.

3. Modify the DE1_SoC.sv such that you use 9 switches and 5 LEDs. SW0 is for Cin, SW4-SW1 for the 4-bits A and SW8-SW5 for B. The 5 LEDs should show Sum3-Sum0 and Cout. For better readability, show the output in the same order shown in Figure 1 from left to right (i.e Cout should be the left most LED and Sum0 is the right most LED).

4. Additionally, modify the assign statements for the HEX displays in the DE1_SoC.sv such that it displays the word "Adding". As an example, to make HEX5 shows the letter 'A', all the segments need to be turned on except segment 3 (HEX5[3]). Figure 2 is from the DE1_SoC datasheet and shows the 7-segments display connections. The segments are active low which means that a value of 0 is needed to turn on any segment.

Figure 2. HEX0 connections on the DE1_SoC board



Accordingly, to display the letter 'A' on HEX5, the following statement can be used

```
assign HEX5 = 7'b0001000; //displays 'A' where all segments except 3 are turned on.
```

5. Make DE1_SoC.sv your top level entity and simulate it in ModelSim and save a picture of the simulation for your lab report.
6. Load the program to the FPGA and demonstrate to the TA.

Lab Demonstration and Submission Requirements

- Demonstrate the full adder circuit of task 1 to the TA.
- Demonstrate the 4-bits adder of task 3 to the TA.
- Submit a short lab report (about 3 pages) that should include 3 main sections, detailed below.

  Procedure
  - Describe how you approached the problem and include any truth tables or schematics.
  - Include screenshots of the code for task 3.

  Results
  - Include screenshots of ModelSim results
  - Describe what you tested in the simulation, and what the results in the screenshot show
  - Give a brief overview of the finished project, compared to what was asked

  Problems Faced & Feedback
  - Describe any issues you had while completing the lab, and how/if you were able to overcome them.
  - Include tips and tricks that you found out about that may be useful to you in future labs.
  - Give any other positive or negative feedback you have on the lab

- Submit the SystemVerilog files (files with extension .sv) of task 3. Make sure to follow the commenting guide provided
- Submit your report and programs to Canvas. No hard copies.

.

Appendix

Source code used in the videos

1. FullAdder module

```
module fullAdder (A,B, cin, sum, cout);

   input logic A,B, cin;
   output logic sum, cout;

   assign sum = A ^ B ^ cin;
   assign cout = A&B | cin & (A^B);

endmodule

module fullAdder_testbench();

      logic A, B, cin, sum, cout;

      fullAdder dut (A, B, cin, sum, cout);

      initial begin

         A = 0; B= 0; cin =0; #10;
                      cin =1; #10;
                B= 1; cin =0; #10;
                      cin =1; #10;
         A = 1; B= 0; cin =0; #10;
                      cin =1; #10;
                B= 1; cin =0; #10;
                      cin =1; #10;

      $stop;

      end //initial

endmodule
```

2. DE1_Soc module

```
module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);

   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   input logic [3:0] KEY;
   input logic [9:0] SW;

   fullAdder FA (.A(SW[2]), .B(SW[1]), .cin(SW[0]), .sum(LEDR[0]), .cout(LEDR[1]));

   assign HEX0 = 7'b1111111;
   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

endmodule

module DE1_SoC_testbench();

   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [9:0] LEDR;
   logic [3:0] KEY;
   logic [9:0] SW;

   DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW);

   integer i;
   initial begin
   SW[9] = 1'b0;
   SW[8] = 1'b0;
   for (i=0; i<2**8; i++) begin
      SW[7:0] = i; #10;
   end
end

endmodule
```