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 II. You will also learn about how to implement a complex design on the breadboard and will complete the flow of mapping designs from Verilog to the FPGA inside the DE1 SoC. All 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

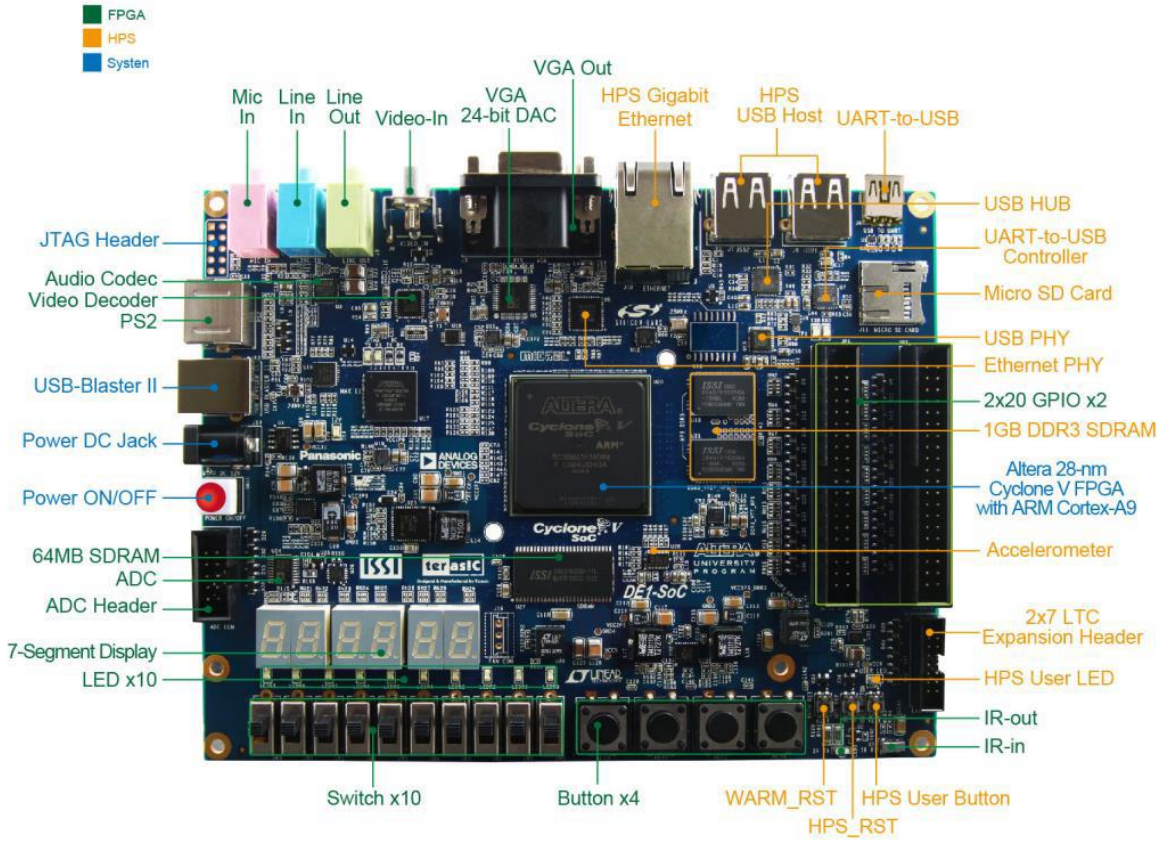
The design kits contain the resources you need to construct most of the circuits assigned to you over the course of the quarter. You are responsible for your lab kit and we expect that you will return the kit in good working order with all pieces intact. This especially applies to the chips, which have 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 (labs 2+).
- 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, Radio Shack on the Ave, Fry's in Renton, or on-line.

We are aware that accidents can happen and the pins on chips may 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

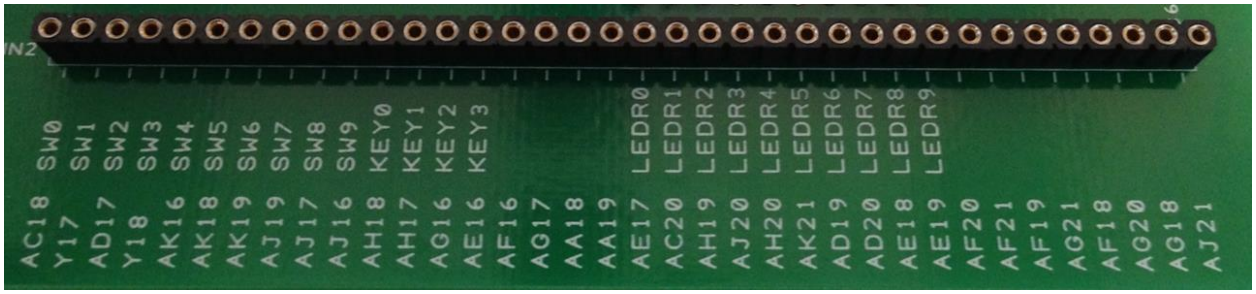


The picture above is a diagram of the DE1 with most of the major components highlighted. For the first few labs you will be using the various input/output devices located directly on the board such as the switches and LEDs. Further details will be provided in each lab. Take note of the large FPGA (Field 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

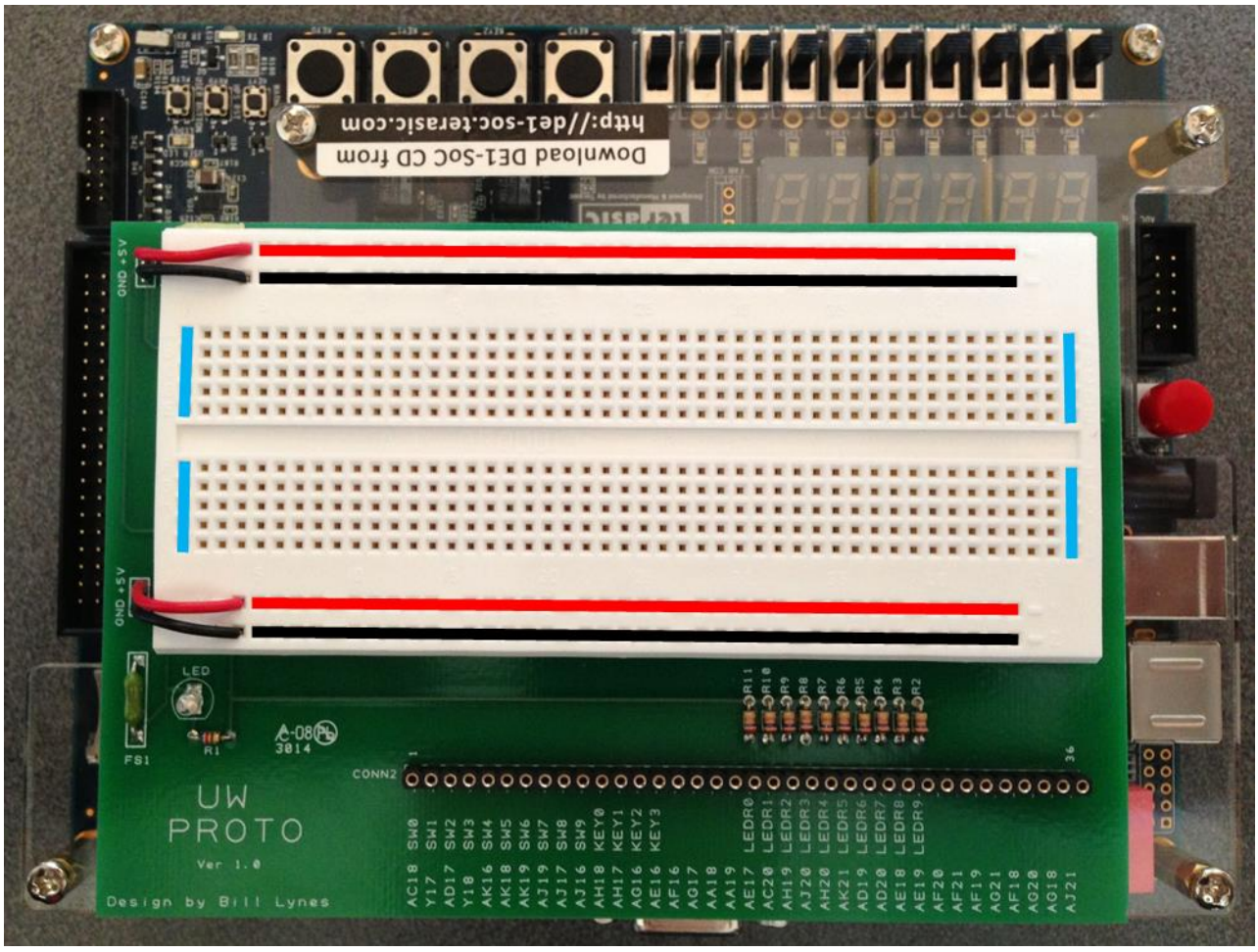
"FPGA" is an acronym for Field Programmable Gate Array. Essentially it is a large array of logical elements that have been connected together. However, in an FPGA the connections between these logical elements can be programmed and reprogrammed, which means the FPGA can be used to build many different 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 48-column 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.

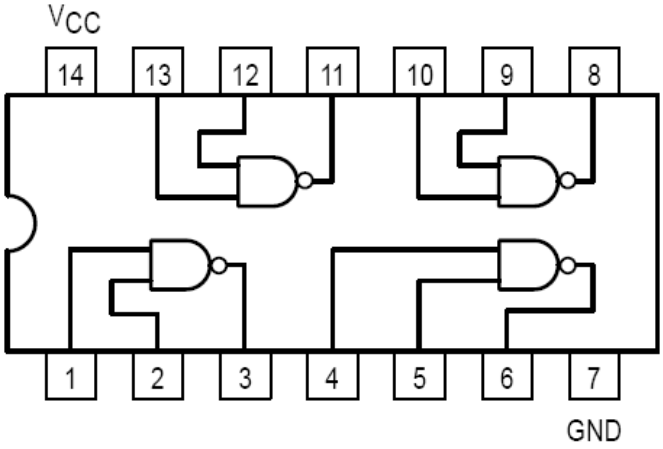
- Make sure your wires are stripped carefully. This means that when you put the wire in your bread board, there shouldn't be any un-insulated wire visible and the wire shouldn't crunch against the bottom of the board.
- Your wires should always be nice and straight. There should be no twists or kinks in them, as they can cause your board to short out when you insert them in your bread board.
- 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.

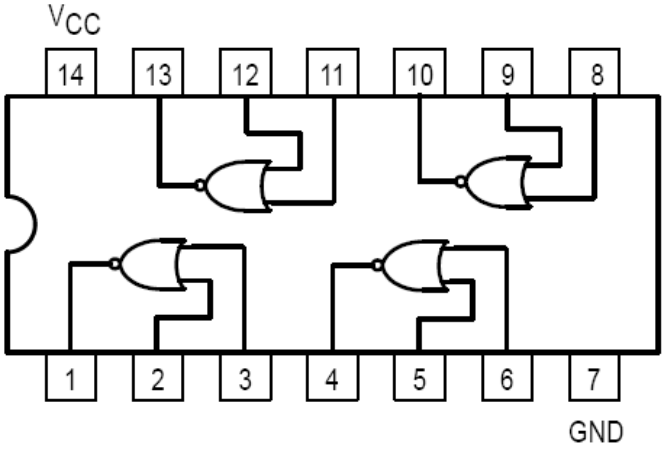
Chips

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.

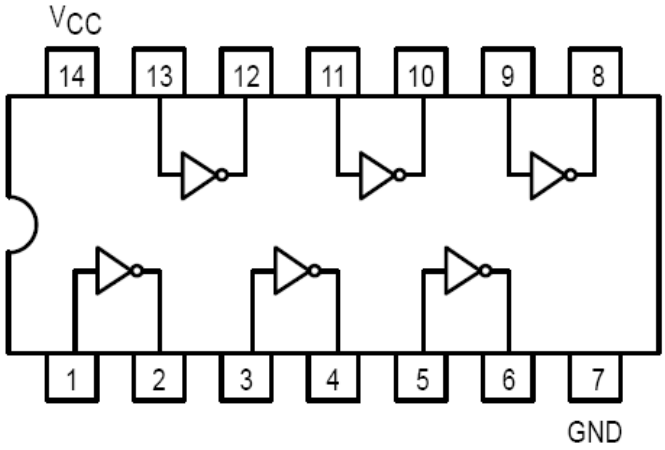
'00 – 4 2-input NAND gates:



'02 – 4 2-input NOR gates:



’04 – 6 inverters:



Using Quartus II Software

Most of the designs in this class will be done through the Altera Quartus II software. This is preloaded on machines in the department, and you are free to 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 there as well.

All of the files you need are available on the class website, including a tutorial that will guide you through the process of creating your first few designs.

Assigned Task – Verilog Design and Simulation

Note that you will reuse parts of this task in the second part of this lab, so keep your files somewhere safe.

You will also be performing steps from the tutorial MANY times in subsequent labs, so taking notes on how to do important tasks will save you lots of time later.

Get the Quartus II tutorial from the class website and perform the steps in the tutorial all the way through section 7. This will have you simulate two designs, mux2_1 and mux4_1. Be sure to do all the steps, and ask the TAs in the lab for help if you get stuck. During your lab demo time you will need to show your TA the simulation of the mux4_1 in Quartus II, and give a simple explanation of what the mux4_1 actually does (Note: we do NOT want a written description of the circuit gates, we want a description of what it actually does – if you’re not sure, see how we described how the mux2_1 works).

Assigned Task – Using Wires, Switches, Pushbuttons, Gates, LEDs

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

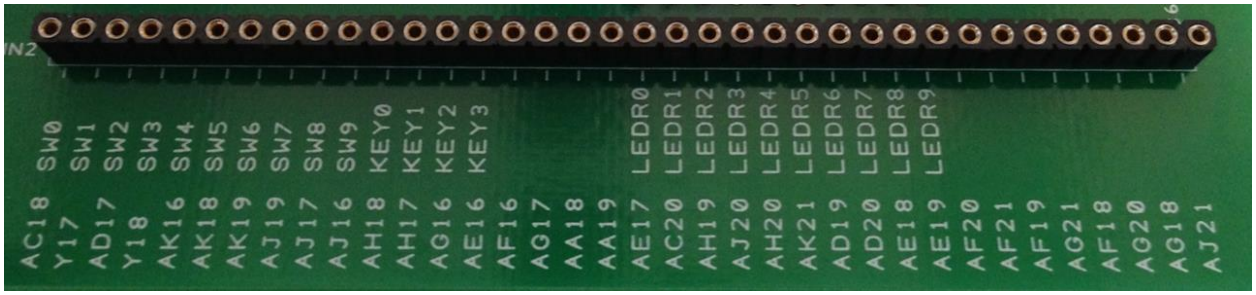
You will need to use the black power cord to power the board - we won't use the grey USB cable until the next lab. Plug the cord into an outlet, and into the "Power DC Jack" socket just above the red on/off button. For each of the next steps, **make sure the board is off when you insert or remove wires or chips**. This will help avoid accidental short-circuits 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 LEDRs 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 LEDR lights up or not will tell you the value of that wire.

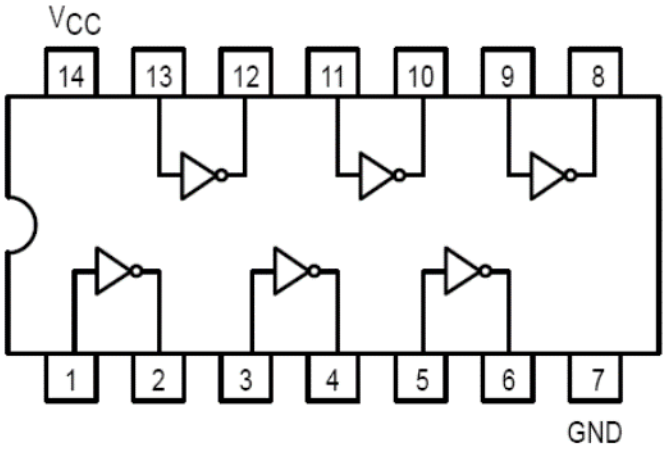
The logic values of switches and pushbuttons

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 Proto board, 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 LEDR 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.

Design Problem – Multi-level Logic on the Breadboard.

Electronics can be very cold and impersonal. So let's change that by having your DE1 recognize you and only you (assuming you only have 9 other friends with carefully chosen student ID numbers…). We want a circuit that will light up an LED when it sees the last digit of your student ID # and no other (including invalid codes such as 1111). The list of all possible codes is given below – find your student ID number under "meaning", and design your circuit to match just that pattern. Your goal is to design the circuit out of Inverter, NAND, and NOR gates, and use the fewest number of gates as possible. Note that to make it fair to those with more difficult student ID #'s, inverters hooked directly to a switch output are free.

First, design the circuit by hand, optimizing with Boolean logic as needed. Then, implement the design on the breadboard. You may use '00, '02, and '04 chips as needed, but cannot use any other types of gates.

Note that for all design problems, you will be graded 100 points on correctness, style, testing, etc. Most labs will also have a bonus category, with which you can get up to 20 bonus points, though only through effort above and beyond just getting it working (i.e. not everyone will get bonus points, and on some labs no-one will get all 20). For this lab, the bonus goal is to use as FEW logic gates as possible. Each gate (an individual Inverter, an individual NAND, and an individual NOR) cost the same, though any inverters connected directly to a switch input are free. The fewer the number of gates, regardless of the number of chips, the better the grade.

SW3	SW2	SW1	SW0	Meaning
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Design Problem – Multi-Digit Recognizer in the FPGA.

Read the Verilog Tutorial on the website up to (but not including) Register Transfer Level (RTL) Code.

The circuit you developed for the previous section is good, but recognizing only one digit is a bit simplistic. We'd like to scale it up to handle more digits, though wiring it all on the breadboard will be way too much work. Instead, we'll develop a Verilog version and load that into the FPGA.

First, complete the Quartus II tutorial you started in lab #1. This will involve loading the mux2_1 design into the DE1 FPGA and testing it with the switches on the board. This will show you the complete flow for creating FPGA-based designs.

Next, develop the logic for a circuit that will recognize the bottom two digits of your student ID number. SW3-SW0 will be the bottom digit, encoded like in the previous section. SW7-SW4 will be the next digit up, using the similar code (i.e. when SW7==1 and SW6, SW5, and SW4 are each 0, the digit encoded is 8). To make things easier I have provided a structure for the file below that will help you get started, and hook things up to the proper inputs and outputs. Note that we do NOT care about the number of gates in the Verilog version – we'll start worrying about FPGA efficiency in later labs.

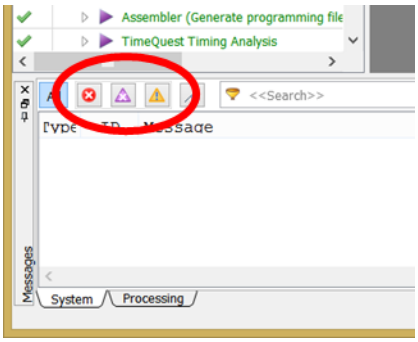
```
// Top-level module that defines the I/Os for the DE-1 SoC board

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;

   // Default values, turns off the HEX displays
   assign HEX0 = 7'b1111111;
   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

   // Logic to check if SW[3]..SW[0] match your bottom digit,
   // and SW[7]..SW[4] match the next.
   // Result should drive LEDR[0].

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);

   // Try all combinations of inputs.
   integer i;
   initial begin
      SW[9] = 1'b0;
      SW[8] = 1'b0;
      for(i = 0; i <256; i++) begin
         SW[7:0] = i; #10;
      end
   end
endmodule
```

Note that the testbench uses an “initial” statement to generate the input patterns, with #10 delays, for loops, etc. These are great for quickly writing testbenches but should not appear in the code that actually gets put into the FPGA.

Make sure to test your design in simulation BEFORE mapping it to the FPGA. This will speed up your development time for this lab and will be critical as your designs get larger.

Quartus II Tip: Many of the “warning” messages that Quartus prints are actually early warnings of errors in your design. You should fix your Verilog to remove all warnings before simulating your design, and your final project should compile without warnings.

To quickly find errors or warnings, the upper-right corner of the message window has filtering icons that will just show errors, critical warnings, or warnings (left to right) in the message window:



Note however that Quartus II does have some useless warning messages. We have set up the tool to filter out many of them, but if you find a warning message that you don't think anyone should see, please show your TA. We will augment the system to ignore these or explain why that actually is a real problem with your code.

During simulation you'll likely notice two new kinds of waveforms – red and blue. Red (or the value X) represents an unknown value – since our testbench doesn't specify the value of the KEY inputs, their value is unknown. Blue (or the value Z) represents a signal that is disconnected – since our circuit doesn't make any connection to the LEDRs other than LEDR[0], the others are disconnected. Red and Blue lines are a way for the simulator to get your attention and have you think about whether those values are what you actually want.

Note: do NOT remove the 1-digit design or the Double Inverter and NOR circuits from the breadboard when you develop the 2-digit design in Verilog, you'll need to demo them all to the TAs. When you load your design into the FPGA, the default design that connects the breadboard to the DE1 sliders and LEDs is overwritten, isolating the breadboard signals from the rest of the DE1. If you turn the DE1 off and then on again, the default configuration will be back.

Lab Demonstration/Turn-In Requirements

You will be graded 100 points on correctness, style, testing, test coverage, etc. Your bonus is using as few logic gates as possible for the 1-digit ID recognizer. Inverters on the inputs DO NOT count towards your total; all other gates on your schematic will count as 1 gate. Note that you can only use standard gates (AND, OR, NAND, NOR, NOT, XOR, XNOR) – no AND gates with one input inverted, etc.

A TA needs to "Check You Off" for each of the tasks listed below.

- Demonstrate to the TA your working simulation of the mux4_1.sv circuit.
- Demonstrate to the TA your working Double Inverter and NOR circuits.
- Demonstrate to the TA your working 1-digit and 2-digit ID recognizers working in hardware.
- Simulate in ModelSim your 2-digit recognizer for the TA
- Submit on canvas a pdf that lists the following:
  a. Your name and the class number.
  b. A simple, correct explanation of what the mux4_1 circuit does.

c. The value of the signal (TRUE or FALSE) that makes the LEDR's light up.
d. The position of the slider switches that cause them to output TRUE.
e. The position of the pushbuttons that cause them to output TRUE.
f. The circuit diagram for the 1-digit recognizer breadboard circuit
g. The Verilog printout for the 2-digit recognizer design.

- Tell the TA how many hours (estimated) it took to complete this lab, including reading, planning, design, coding, debugging, testing, etc. Everything related to the lab (in total).