Lab 1 – Introduction to Verilog HDL & the Lab Design Flow

In this lab you will be introduced to the Verilog Hardware Descriptive Language (HDL) and the design flow we will be using in the labs throughout the course. You will receive a message from Canvas when your sign in process is working. Do not try to sign in until you receive such a message.

Connecting to electro9

When you receive the message that your sign in process is available, follow the instructions on the Digital Design course web page:

https://sites.google.com/a/temple.edu/ece2612/

In particular you should reference the following links on the left of the page:

- ***Cloud9 Setup/Sign Out***
- ***Cloud9 – Vivado Simulation***
- ***Cloud9 – Vivado Synthesis***.

If you have trouble connecting to *electro9*, see the TA or the instructor.

You will also be needing a waveform viewer that you will use on your local computer. Information on this will be available in the link: ***Cloud9 – Debugging***, which is currently under construction. It should be available for Lab 2.

Basys3 Hardware

The hardware you will be using is *Basys3*, which you will have purchased from Digilent. This board contains an FPGA device manufactured by *Xilinx*. You also need: a) a micro USB-A cable for powering the board, b) a USB memory stick for storing your design.

See the Appendix section for instructions on loading designs into this board.

Basic Design Flow for Lab Exercises

The Design Flow for the labs in this course consists of the following steps:

- Enter your design using Verilog HDL into the ***Cloud9 Integrated Development Environment (IDE)*** tool on *electro9*.
- Simulate your design and verify that it is functionally correct (*Vivado command scripts from Cloud9*).

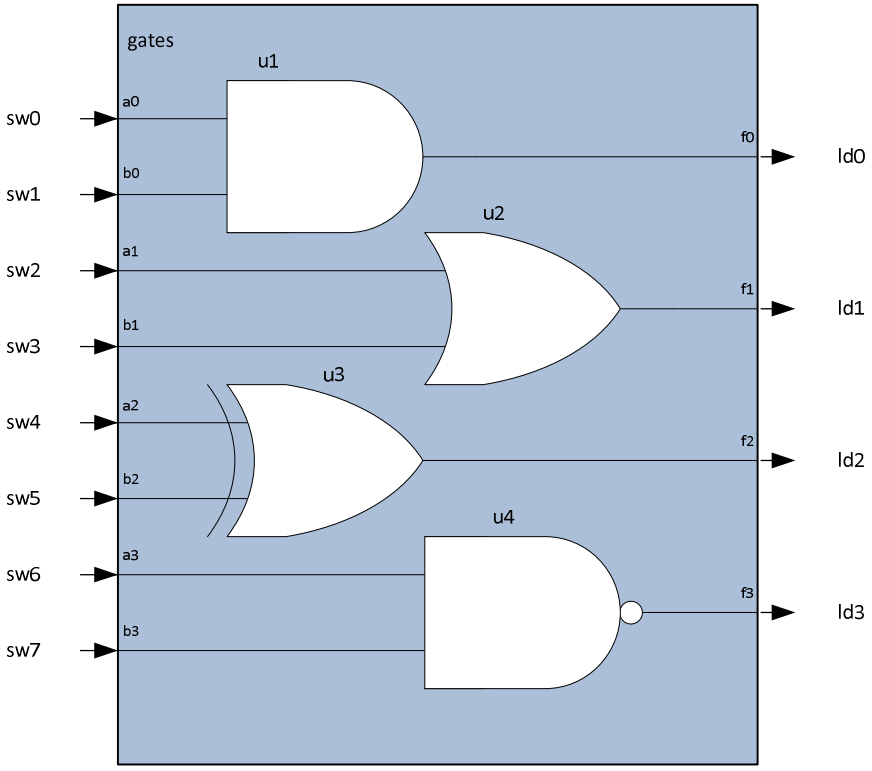
- Create a bit file that can be loaded into the FPGA hardware (*Vivado command scripts from Cloud9*).
- Copy/Download the bit file to your local laptop (*Cloud9*).
- Copy the bit file from your local laptop to your USB memory stick.
- Insert the USB memory stick and run your design on your hardware.

The Design

For this lab you need to design four different gates by ***instantiating*** the appropriate ***primitives***. You will use the Verilog gate primitives as discussed in class, videos and the textbook (examples of gate primitives: http://www.asic-world.com/verilog/gate1.html). The four gates for this design are:

- AND, OR, XOR, NAND

The inputs of the gates will be connected to switches and outputs to LED's. A block diagram/schematic of the *gates* design (module) is shown below.



Complete the truth table for the gates in this design.

a0, 1, 2 or 3	b0, 1, 2 or 3	f0	f1	f2	f3
0	0				
0	1				
1	0				
1	1				

Design Entry

Many of these steps are also covered in the web pages, *Cloud9 – Vivado Simulation* pages:

https://sites.google.com/a/temple.edu/ece2612/

Start by opening the *Cloud9 IDE* tool on electro9. Open the ***lab1*** folder and double click on the *gates.v* file. This file contains a skeleton of your design module. You are now ready to enter your design.

Within *gates.v* instantiate each of the four primitive gates, connecting them as shown in the above diagram. The instance names are u1, u2, u3 and u4 as shown above.

Save your edited result when you are finished editing your design (ctrl-s keystroke).

To check the syntax, use the Run command by right clicking on either the tab or the filename. A window will pop up in the bottom of the screen to inform you of any syntax errors it found.

Simulation

This is an operation where it again checks your design's syntax and if ok, it simulates the design – to make sure it matches your design objective or specifications. The design specification that is tested for this lab is contained in the truth table above.

In your hardware implementation, you will be testing this by toggling switches and observing the outputs of the LED's. One way to see if your design meets your specifications before committing it to hardware is to provide all of the possible input combinations to the design module in a wrapper module, run the simulator and check by viewing a timing diagram to check that each output is correct. Such a wrapper is called a verification ***testbench***.

A better way is to create a ***self checking testbench***. This module also *knows* the expected values and compares the simulated outputs to the expected outputs. Then any mismatches are printed to the console display.

In your directory you are provided a *self checking testbench*, called ***tb_gates.v***. The details of this module will be covered in future lectures and labs. For now it is important that you know that it reads a text file called ***tb_gates.txt***. You can open this file and compare it to your truth table. These data are read by the *self checking testbench*. When you run the simulation (*tb_gates.v*), each of these inputs will be applied to your design every 20 nsec. If your design has errors, there will be mismatches. **You must correct these mismatches before continuing to the hardware implementation.**

To simulate your design, find the file ***gates.sim*** in your Environment, right click on it and select ***Run***. Another Run configuration window will appear in the bottom area. This will contain messages from your simulation. The messages are also copied to a file: ***xsim.log***. Open this file by double clicking on it.

You are looking for a message: Simulation complete – no mismatches!!! You need to take a screen shot of this window to put into your lab report.

If you have mismatches, the inputs, expected outputs and your design outputs will be displayed. You should be able to find your design error using this information to direct you.

Synthesis

This step creates a *bit file* that must be copied to your local PC, then to a USB memory stick, and finally loaded into your Basys3 board through the memory stick. The design can then be tested in hardware.

A top level input/output wrapper is now substituted for the testbench in your design. For this lab it is called: *lab1_top_io_wrapper.v.* You can view this file if you like, but do not edit it. There is a related file called *lab1_top_io_wrapper.tcl.* To synthesize the design, right click on this TCL file and select *Run*. This will take a little time for the tool to optimize your design and map it to gates for your hardware.

When it completes without errors, it should have built a *bit* file called: *lab1_top_io_wrapper.bit*. This is the file that you need to load into your hardware. **DO NOT TRY TO VIEW THIS DATA BY DOUBLE CLICKING ON IT**. It is a large binary file and will almost certainly lock up your Cloud9 environment for a time. To download it, right click on the filename and select *Download*. The file will be in your Downloads path on your local PC.

Next copy or move the bit file to the top level of your USB flash drive. Then unmount your drive and plug it into the Basys3's USB port with the power off. Make sure a) that there is only one bit file at the top level directory of the memory stick and b) that the Basys3 jumpers are configured to load the FPGA with a bit file on the USB memory stick (see Appendix below).

Verify your design by toggling the switches and observing the LED's. Is the behavior what you expect? Does it meet the design specifications?

Shutting Down/Logging Off

When you are finished, close all of the open tabs inside the *Cloud9 IDE*. Then close the browser IDE tab. Finally go the AWS tab and sign out of your account.

Appendix

From the Basys3 reference manual on booting from a USB memory device:

You can program the FPGA from a pen drive attached to the USB-HID port (J2) by doing the following:

1. Format the storage device (USB flash drive) with a FAT32 file system if it is not already done.
2. ***Place a single .bit configuration file in the root directory of the storage device***.
3. Attach the storage device to the Basys3.
4. Set the JP1 Programming Mode jumper on the Basys3 to "USB" (see Figure below).
5. Push the PROG button or power-cycle the Basys3 (see Figure below).

The FPGA will automatically be configured with the *.bit* file on the selected storage device. Any *.bit* files that are not built for the proper Artix-7 device will be rejected by the FPGA.

The Auxiliary Function Status, or "BUSY" LED (LD16), gives visual feedback on the state of the configuration process when the FPGA is not yet programmed:

- When steadily lit, the auxiliary microcontroller is either booting up or currently reading the configuration medium (pen drive) and downloading a bitstream to the FPGA.
- A slow pulse means the microcontroller is waiting for a configuration medium to be plugged in.
- In case of an error during configuration, the LED will blink rapidly.