

UNIVERSITY OF CALIFORNIA AT BERKELEY
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

| | |
|-----------|--------------------------------------------------------------------------|
| ASSIGNED: | Week of 1/28 |
| DUE: | Week of 2/4, 10 minutes after start of <i>your assigned</i> lab section. |

Lab 2

Designing with Verilog

1.0 Motivation

In this lab you will learn how to express a circuit design in a Hardware Description Language (HDL), namely Verilog. By creating two similar designs in behavioral and structural verilog, you will get a chance to not only familiarize yourself with different styles of Verilog, but also with the ModelSim simulator.

2.0 Introduction

To support the creation of very efficient circuits, and as an effort to ease entry of old designs, and importing designs from other environments, Verilog supports the use of universally accessible primitive gates like `and`, `or` and `not`. In addition, it supports dataflow and behavioral modeling, which allow the design of more efficient control.

In this lab, you will investigate the use of the two primary kinds of Verilog: structural and behavioral. You will create two similar, simple modules. One module will be specified in behavioral Verilog, where you will describe the function of the circuit succinctly and rely on the modeling and synthesis CAD tools to determine the exact circuit. The other circuit will be written in structural Verilog, where you will describe the circuit down to the gate level. The final piece of Verilog in this lab will be the testbench, which, though also written in behavioral Verilog, is a little different in that it **cannot** be synthesized. This is because it describes an artificial set of test stimulus rather than an actual circuit.

3.0 Prelab

Please make sure to complete the prelab before you attend your lab section. **You will not be able to finish this lab in 3 hrs otherwise!**

1. **Read this handout thoroughly.** Pay particular attention to Section **4.0 Lab Procedure** as it describes in detail the circuits you must create.
2. **Examine the Verilog** provided for this week's lab.
 - a. Make sure you understand exactly how the **Lab2Testbench works**.
3. **Write all of your Verilog ahead of time.**
 - a. You should **add more test cases** to **Lab2Testbench.v**. You will not get checked off for this lab without a more thorough testing of your modules.
4. You will need the **entire 3 hr lab** to **test and debug** your Verilog!

4.0 Lab Procedure

Since we expect you to write your Verilog ahead of time, and Verilog is nothing more than a bunch of standard text in a file with a *.v extension, **you can do this part of the lab entirely from home** in your favorite text editor (we recommend **notepad**). Or you can come into the lab and use the tools there. For those of you who like maintaining a single Xilinx Project Navigator project for each lab, you can even **create the project ahead of time and write your Verilog from within Project Navigator**.

Whatever editor you use to create your Verilog files, at some point you will need to **add all of the Verilog files** (including **Lab2Top** and **Lab2Testbench**) to a new **Xilinx Project Navigator project**, or to a new **ModelSim project**, to test your code and get it checked off.

Remember to **manage your Verilog, projects and folders well**. Doing a poor job of managing your files can cost you **hours of rewriting code**, if you accidentally delete your files.

4.1 Lab2Top

In this lab you will be building the two sub-modules of Lab2Top. Lab2Top takes a string of 8-bit inputs and maintains both a running sum of the inputs using an Accumulator and the largest of the inputs using a PeakDetector. The PeakSelect input to Lab2Top can be used to select which of these values appears at the output Out.

Below are: a high-level block diagram of Lab2Top, a table specifying its ports and their purposes and a hierarchy of the modules that you will need to build to complete this lab.

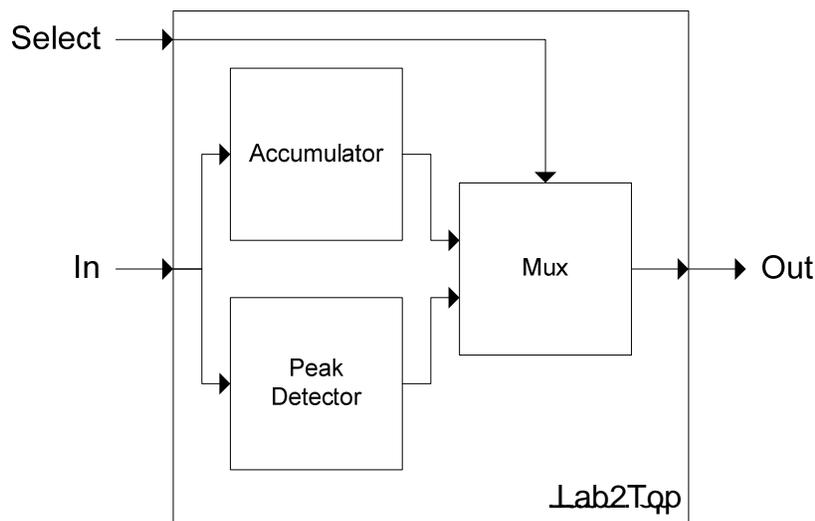


Figure 1: Lab2Top High Level Block Diagram

| Signal | Width | Dir | Description |
|--------|-------|-----|--------------------------------------------------------------|
| In | 8 | I | The input value |
| Out | 8 | O | The output value from either the PeakDetector or Accumulator |

| | | | |
|------------|---|---|----------------------------------------------------|
| Enable | 1 | I | Notify Lab2Top to accept the current input at In |
| PeakSelect | 1 | I | Selects the output form the PeakDetector when 1'b1 |
| Clock | 1 | I | The Clock signal |
| Reset | 1 | I | Reset both the Accumulator and the PeakDetector |

Table 1: Port Specification for Lab2Top

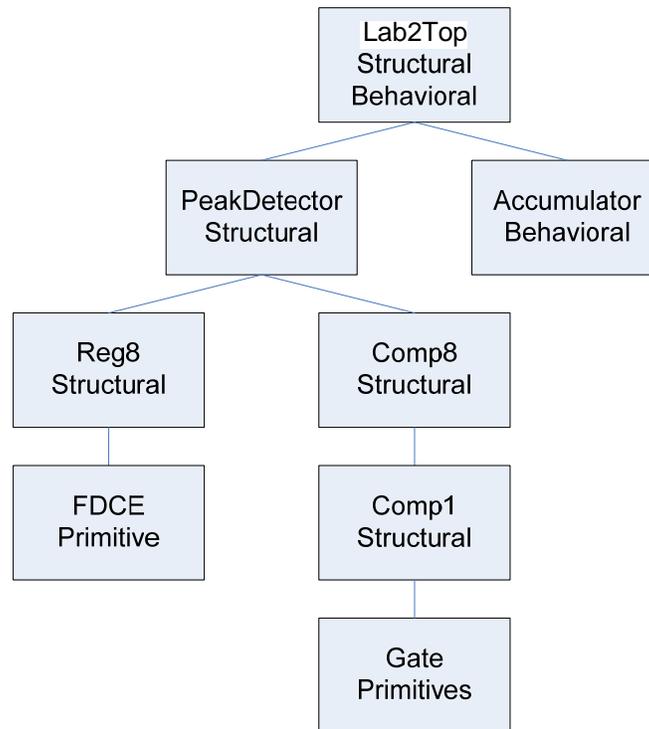


Figure 2: Hierarchy of Modules

4.2 Behavioral Accumulator

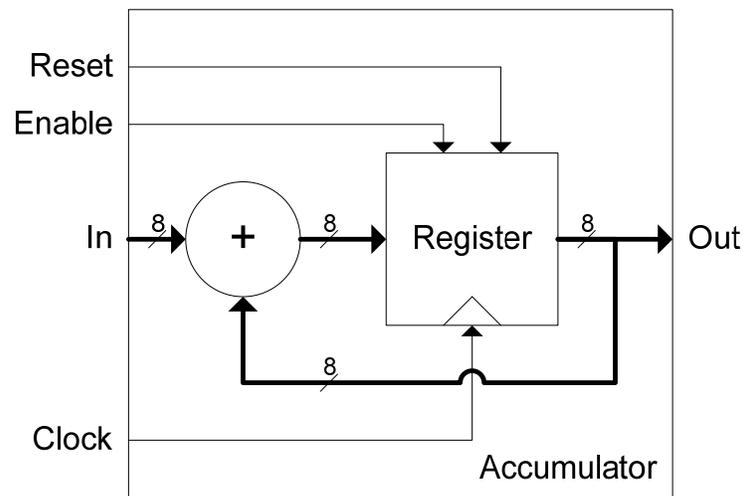


Figure 3: The Accumulator

1. In the text editor of your choice (we recommend notepad) create a file called **Accumulator.v** in which you will define a single module called `Accumulator`.
 - a. This module should have the ports listed in Table 2.
 - b. Be sure to use good Verilog style. You'll be graded on this!

| Signal | Width | Dir | Description |
|--------|-------|-----|-----------------------------------------------------------|
| In | 8 | I | The input value to be added to the total output |
| Out | 8 | O | The sum of all previous input values |
| Enable | 1 | I | Notify the accumulator to add In to Out |
| Clock | 1 | I | The Clock signal, the output changes only on rising edges |
| Reset | 1 | I | Reset the output to 8'h00. |

Table 2: Port Specification for Accumulator

2. Specify your module entirely in **behavioral Verilog**.
 - a. Our solution takes 4 lines of behavioral Verilog (not counting declarations).
 - b. You may not instantiate any other module within your accumulator.
 - c. You may wish to refer to the lab lecture slides for examples of behavioral Verilog.
3. **Types of Assignment**
 - a. A **blocking** assignment is written as “=” and should be used in combinational logic.
 - b. A **non-blocking** assignment is written as “<=” and should be used for sequential logic, namely anywhere you have an `always @ (posedge Clock)`
 - c. Note that these kinds of assignments refer only to `always` blocks. The `assign` keyword always requires a simple “=”.
4. **Reg vs. Register**
 - a. The keyword `reg` in Verilog names a location into which an `always` block can place a value. It does not create any kind of register, it is merely a poorly named artifact of days past.
 - b. `always @ (posedge Clock)` implies a register.
 - i. Any `reg` assigned using Non-blocking assignment (<=) in an `always @ (posedge Clock)` block will become a register during synthesis.
5. **Test this module individually**
 - a. You should be able to **modify the testbench** that we have given you to test `Accumulator` instead of `Lab2Top`.
 - b. The `Lab1Circuit` you dealt with last week was also an accumulator.

4.3 Structural Peak Detector

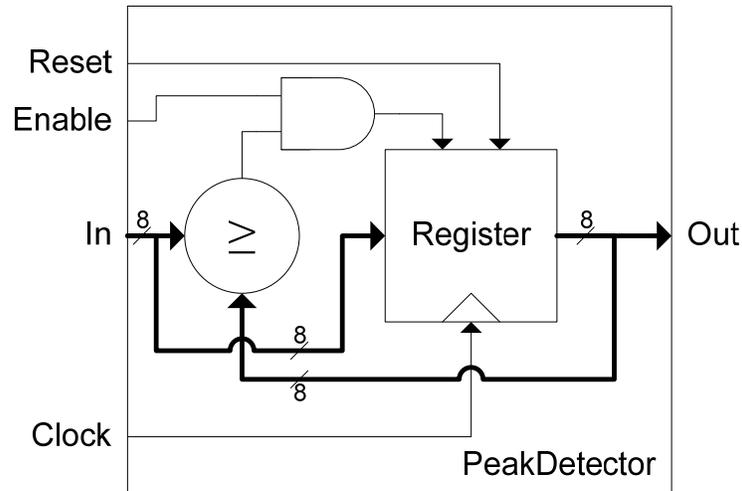


Figure 4: The Peak Detector

This will most certainly be the more time consuming part of the lab to write. Please note that in the future we will not expect you to use purely structural Verilog, as it is generally less efficient (in your time as well as in implementation quality).

For this part of the lab you must follow these rules:

1. You **may not use the assign keyword** at all.
2. You **may not use the always keyword** at all.
3. You may only use the primitives that we specify in the following sections.

4.3.1 Structural Register

1. In the text editor of your choice (we recommend notepad) create a file called **Register.v** in which you will define a single module called **Reg8**.

- a. This module should have the ports listed in Table 3.

| Signal | Width | Dir | Description |
|--------|-------|-----|------------------------------------------------------------|
| D | 8 | I | The input value to be loaded into the register |
| Q | 8 | O | The output value from the register |
| Enable | 1 | I | When high at a rising clock edge, load D into the register |
| Clock | 1 | I | The Clock signal, the output changes only on rising edges |
| Reset | 1 | I | Reset Q to 8'h00 |

Table 3: Port Specification for Reg8

2. **In this module you may only instantiate the FDCE primitive.**
 - a. You may instantiate as many of these as you need.
 - b. The FDCE primitive is a 1 bit D type Flip-Flop with the ports listed in Table 4 below.

| Signal | Width | Dir | Description |
|--------|-------|-----|-----------------------------------------------------------|
| Q | 1 | O | The output value from this Flip-Flop |
| C | 1 | I | The Clock signal, the output changes only on rising edges |
| CE | 1 | I | Clock Enable, when 1'b1, D will be copied to Q |
| CLR | 1 | I | When 1'b1, will force Q back to 1'b0 |

| | | | |
|---|---|---|---------------------------------------------|
| D | 1 | I | The data input, copied to Q when CE is 1'b1 |
|---|---|---|---------------------------------------------|

Table 4: Port Specification for FDCE

3. You may wish to test this module individually.

4. Simulating with FDCE

- This primitive is part of the Xilinx **unisims_ver** library. See the ModelSim tutorial on the EECS150 course website at <http://www-inst.eecs.berkeley.edu/~cs150/fa06/Documents.php> for more information.
- The FDCE primitive will only exist on Xilinx parts. Other manufacturers might call it something else.
- During simulation: **the FDCE primitive will not function during the first 100ns of simulation time.** This is because it is a realistic model of the Flip-Flops in a Xilinx chip, which are held in reset for the first 100ns.
- The reset on the FDCE primitives is asynchronous.**

4.3.2 Structural Comparator

1. In the text editor of your choice (we recommend notepad) create a file called **Comparator.v** in which you will define **two modules** called **Comp8** and **Comp1**.

2. The **Comp8** module should have ports as listed in Table 5.

| Signal | Width | Dir | Description |
|--------|-------|-----|----------------------------------------|
| In0 | 8 | I | The first input |
| In1 | 8 | I | The second input |
| Out | 1 | O | Should be 1'b1 whenever $In1 \geq In0$ |

Table 5: Port Specification for Comp8

- The **Comp8** module should be built from **gate level primitives** as described below **and Comp1 modules**.
- You should be sure to **make good use of bit vectors (busses) in this module** rather than declaring a lot of unrelated one bit wires.

3. The **Comp1** module should have ports as listed in table 6.

| Signal | Width | Dir | Description |
|------------|-------|-----|----------------------------------------------------------------------------------|
| In0 | 1 | I | The first input |
| In1 | 1 | I | The second input |
| GreaterIn | 1 | I | The GreaterOut from the next higher bit |
| EqualIn | 1 | I | The EqualOut from the next higher bit |
| GreaterOut | 1 | O | Should be 1'b1 whenever $In1 > In0$ and EqualIn is 1'b1, or if GreaterIn is 1'b1 |
| EqualOut | 1 | O | Should be 1'b1 whenever $In1 = In0$ and EqualIn is 1'b1, but GreaterIn is 1'b0. |

Table 6: Port Specification for Comp1

- The **Comp1** module **should be built from the and, or, not and xor gate level primitives**.
 - These primitive gates can have **any number of inputs**.
 - They all have one output

- iii. They do not have pin names (Like In0 or Clock)
 - iv. Here is an **example of xor-ing three signals together**:

```
xor xor1(OutputWire, Signal1, Signal2, Signal3);
```

Notice the name of the primitive is xor and the name of this instance is xor1.
 - v. **For more examples of using primitive gates see Lab1Circuit.v from last week's lab.**
- b. Notice that each Comp1 module is responsible for **comparing one bit** of In0 to one bit of In1.
- i. To generate a useful output however it needs to know the relationship between the **higher order bits** of In1 and In0, hence the GreaterIn and EqualIn inputs.
 - ii. The GreaterOut and EqualOut outputs from the least significant bit (bit 0), will yield the correct information for the comparison of **all of the bits of In0 and In1**.
 - iii. GreaterOut and EqualOut of the highest bit, will feed to GreaterIn and EqualIn of the second highest bit and so on.
 - iv. You should use the GreaterOut and EqualOut of the least significant bit to form your Comp8 output.
- 4. You may wish to test this module individually**
- a. If you decide to write a testbench for this module, you do not need a clock. However to see waveforms, **you will need to let some time pass**. The amount of time is immaterial.

4.3.3 Structural Peak Detector

5. In the text editor of your choice (we recommend notepad) create a file called **PeakDetector.v** in which you will define a single modules called PeakDetector. Refer to Figure 4 for more information.
- a. This module should have the ports listed in Table 7.

| Signal | Width | Dir | Description |
|--------|-------|-----|-----------------------------------------------------------|
| In | 8 | I | The input value |
| Out | 8 | O | The output value, the largest input since the last reset |
| Enable | 1 | I | When high at a rising clock edge, accept a new input |
| Clock | 1 | I | The Clock signal, the output changes only on rising edges |
| Reset | 1 | I | Reset Out to 8'h00 |

Table 7: Port Specification for PeakDetector

- b. You may wish to refer to Figure 3 for assistance.

5.0 Lab 2 Checkoff

ASSIGNED: | Week of 1/28
 DUE: | Week of 2/4, 10 minutes after start of *your assigned* lab section.

Name: _____ SID: _____
 Section: _____

I Behavioral Accumulator _____ (15%)
 II Structural Peak Detector
 1 Register _____ (15%)
 2 Comparator _____ (20%)
 III Quality of Verilog _____ (25%)
 IV Complete Simulation _____ (25%)

V Hours Spent: _____

VI Total: _____

VII TA: _____

| | | |
|------------------|---------------|-------------------------------------------------------------------------|
| RevD – 1/27/06 | Brian Gawalt | Fixed Comparator description (Table 6) |
| RevC – 8/14/2005 | Randy Katz | Various minor edits and grammatical fixes |
| RevB – 1/17/2005 | Greg Gibeling | Updated to become Lab3, Working to incorporate the errata |
| RevA – 7/7/2004 | Greg Gibeling | New Lab Designed to replace old Lab3 covering Verilog and Simulation |