Lab Assignment #3

Guideline

This lab can be done with a partner.

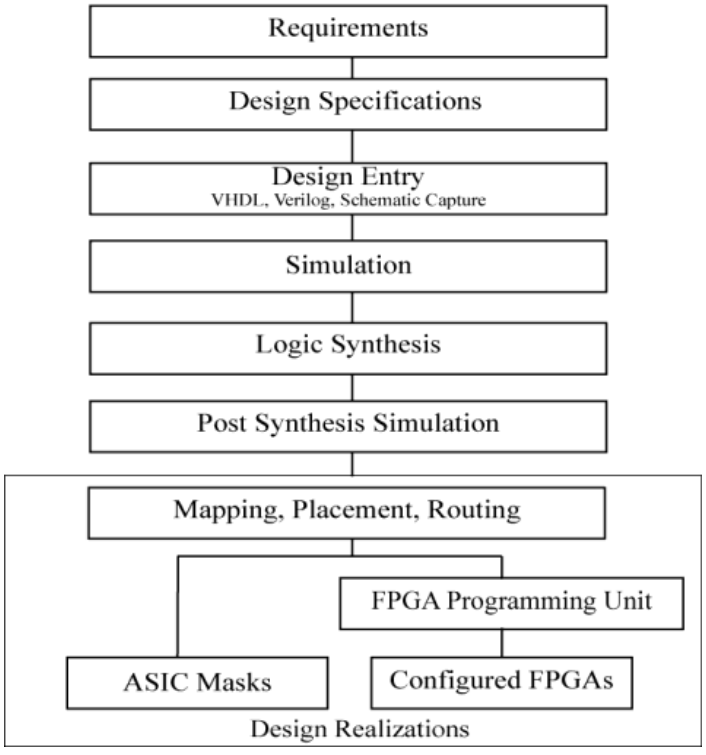
Objective

1. Understanding the ASIC/FPGA design flow
2. More digital design - sequential and combinational circuits.
3. Learn writing and using testbenches in Verilog
4. Implementing circuits on FPGA

Problem 1: ASIC/FPGA Design Flow

The following figure shows the design flow as described in chapter 2 of the text. Annotate each box in this figure with the answers to the following questions:

a. What is the function of each box (answer in one line)?
b. Which tool do you use in the lab to perform this step? If a step is not performed in the lab, mention this.
c. What inputs are needed at each stage and what outputs are delivered at each stage?



Problem 2: Package Sorter (simulation only – using a testbench)

Design a package sorter to classify packages based on their weights and to keep track of packages of different categories. The sorter has an active high asynchronous reset and will keep track of packages since the last reset. Packages should be classified into 6 groups:

Fall 2016: Use configuration 1

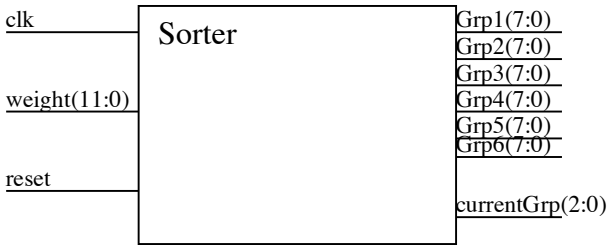
Configuration #1

i) between 1 and 200 grams
ii) between 201 and 500 grams
iii) between 501 and 800 grams
iv) between 801 and 1000 grams
v) between 1001 and 2000 grams
vi) greater than 2000

Configuration #2

i) between 1 and 250 grams
ii) between 251 and 500 grams
iii) between 501 and 750 grams
iv) between 751 and 1500 grams
v) between 1501 and 2000 grams
vi) greater than 2000

You need to decode weight measurements and classify them into various groups. The input to the circuit will be a 12-bit *unsigned* binary number (indicating the weight of the package), a clock signal, and a reset. One of the outputs will be *currentGrp*, a 3-bit unsigned number representing the current group number. There will also be six 8-bit *unsigned* outputs *Grp1-Grp6* representing the number of items weighed in each category since the last reset. The reset line is provided as input to allow these counts to be cleared.



The output lines have the following functionality:
currentGrp[2:0]: Outputs the group number for the weight currently being applied to the sorter. When a weight of zero is applied, it should output a zero. This should update as soon as a package weight changes and may not necessarily reflect the last group that a package was assigned to. **Assume there is nothing on the package sorter when simulation begins, i.e. *currentGrp* is initially 3'b000. Also assume that reset does NOT affect the value of *currentGrp[2:0]***.

Grp1-Grp6[7:0]: Outputs the number of objects that have been weighed in each group since the last reset. These outputs should be zero when reset='1'.

Notice that the functionality of the two outputs is such that the description of *currentGrp* will be purely combinational since it does not depend on any previous inputs. But the description of *Grp1-Grp6* will be sequential since it depends not just on the current input but also on the previous inputs.

Any sequential output should change on the falling edge of the clock. Notice that the *clk* signal will be significantly faster than the duration of the weight signal. As such, you must ensure that the count is only updated once for a given input weight. Secondly, new objects can only be detected and sorted if the weight is allowed to go to zero. This is to ensure that any fluctuations in the weight after it has been sampled are not considered new items. Only the first weight after 0 updates a group count.

Test your design by using **a Verilog testbench** similar to Fig. 2-64 in the text. The testbench should use arrays (to set the inputs, to store the expected group counts and currentGrp values). Do not just use the example input. It is for illustrating the desired functionality. You are responsible for adequately testing your design, so make sure you test everything described for this problem.

Example input sequence

For Configuration 1

Reset → Put 250grams on → Take off → Put on 300 grams → Take off → Put on 501grams → Put 512 grams more
[In your waveforms, this input sequence will look like this: reset -> 250 -> 0 -> 300 -> 0 -> 501 -> 1013]

At the end of this sequence, the outputs should be:

grp1 = grp4 = grp5 = grp6 = 0x00
grp2 = 0x02
grp3 = 0x01
currentGrp = 0x5

Note that after 501 grams is sampled in grp3, adding 512 grams only updates the current group and not the grp5 count.

For Configuration 2

Reset → Put 270grams on → Take off → Put on 300 grams → Take off → Put on 501grams → Put 512 grams more
[In your waveforms, this input sequence will look like this: reset -> 270 -> 0 -> 300 -> 0 -> 501 -> 1013]

At the end of this sequence, the outputs should be:

grp1 = grp4 = grp5 = grp6 = 0x00
grp2 = 0x02
grp3 = 0x01
currentGrp = 0x04

Note that after 501 grams is sampled in grp3, adding 512 grams only updates the current group and not the grp4 count.

Problem 3: Traffic Light Controller (implementation – on an FPGA)

Design a traffic light controller for an intersection with a main street, a side street, and a pedestrian crossing.

Traffic light A consists of three lights: Green (Ga), Yellow (Ya), and Red (Ra).
Similarly, traffic light B consists of three lights: Green (Gb), Yellow (Yb), and Red (Rb).
Lastly, the walk indicator consists of two lights: Green (Gw) and Red (Rw).

The normal sequence of operation is as follows: Ga Rb Rw, Ya Rb Rw, Ra Gb Rw, Ra Yb Rw, Ra Rb Gw, Ra Rb Rw (flash), Ga Rb Rw… (repeat). The timings are given below:

Fall 2016 : Use configuration 1

Configuration 1

Main (A) Street:

- Green: lasts 4 seconds.
- Yellow: lasts 2 seconds.
- Red: lasts 10 seconds.

Side (B) Street:

- Red: lasts 6 seconds
- Green: lasts 3 seconds.
- Yellow: lasts 1 seconds.
- Red: lasts 6 more seconds.

Pedestrian Crossing:

- Red: lasts for 10 seconds.
- Green: lasts 2 seconds.
- Red: Flashes 4 seconds at 1Hz.

Maintenance mode:

- RST=1: Ra, Rb, and Rw all flash at 1Hz
- RST=0: Traffic lights should resume operation with Ga,Rb,Rw as initial state

Configuration 2

Main (A) Street:

- Green: lasts 3 seconds.
- Yellow: lasts 2 seconds.
- Red: lasts 8 seconds.

Side (B) Street:

- Red: lasts 5 seconds.
- Green: lasts 3 seconds.
- Yellow: lasts 1 seconds.
- Red: lasts 4 more seconds.

Pedestrian Crossing:

- Red: lasts for 9 seconds.
- Green: lasts 2 second.
- Red: Flashes 2 seconds at 2Hz.

Maintenance mode:

- RST=1: Ra, Rb, and Rw all flash at 1Hz
- RST=0: Traffic lights should resume operation with Ga,Rb,Rw as initial state

The above mentioned delays can be obtained through the use of counters, just like you divided 100MHz clock to generate a 1Hz (1 sec period) in Lab#2. **Note that if something is to flash at 4Hz, that means there should be 4 highs and 4 lows in one second, not 2 highs and 2 lows.** If you are still unsure, there is a spreadsheet on Canvas under the lab documentation page that presents the timing in a more precise manner.

Your design steps are listed below:

1. Start by designing a state graph for the controller. You do not need to derive any equations, since you can model the state graph using behavioral Verilog code. Note that on designing your state graph, you will transition from one state to the other when the appropriate time has elapsed.

2. Write behavioral Verilog code that represents your state graph. For purposes of checking the functionality of your code, reduce the counter time to a small number during simulation, otherwise you may have to simulate your code through several simulation pages.

3. Once your code simulates properly, proceed to synthesizing it and implementing it on the FPGA. Read through the FAQs at the beginning of this manual to understand and clarify doubts about how to use always statement to make combinational logic and sequential logic, and how to avoid latches. The following table gives the IO connections for implementing the traffic light controller:

Green Light street A:	LED2
Yellow Light street A:	LED1
Red Light street A:	LED0
Green Light street B:	LED7
Yellow Light street B:	LED6
Red Light street B:	LED5
Green Light Ped Xing:	LED3
Red Light Ped Xing:	LED4
Rst (Maintenance mode)	SW0

There is something else that you need to do as well for this part of the lab. You need to generate and analyze five reports while implementing your design:

1. The Synthesis report – to find out the digital elements used by your design and to verify there are no latches
2. The Utilization report – to find out the number of slices of the FPGA used by your design
3. The IO report – to verify the IO pins you mapped your design to are correct
4. The Route Status report – to verify there are no nets in your design with routing errors
5. The Timing Summary report – to find out the critical path in your design

To view these reports, go to the "Reports" tab in the bottom window. The Vivado Synthesis report can be seen by expanding "Synth Design" after synthesizing your design. Ensure that there are no latches in your design. Look for "cell usage" in the Synthesis report and there should not be any cells having names starting with L but "LUT".

After running the "Implement Design" step, you should be able to access the post-implementation Utilization report under "Place Design" (not the one under "Synth Design"). You can locate the digital elements (like Look-Up Tables, flip-flops and latches) used by your design in the Utilization report. In the Utilization report, you can also verify there are no latches in your design by looking for "Slice Logic" and see if any latches are listed.

The IO Report should be under "Place Design", and the Route Status and Timing Summary reports should be under "Route Design". To find the critical path delay, which is defined as the longest path delay, look under "Max Delay Paths" in the Timing Summary report. Make sure that you uncomment all three lines related to the clock pin in the .xdc file or else you will get an invalid Timing Summary Report.

In the Synthesis report, verify that there are no latches. In the IO Report, verify that the pins are correctly mapped to your design. In the Utilization report, highlight or otherwise note the number of slices used by your design. In the Route Status report, highlight or otherwise note the number of nets with routing errors. In the Timing Summary report, highlight or otherwise note the critical path delay of your design. Include these reports in your lab submission.

Useful Information

1. For the reports, just copy the entire reports into text/doc files and highlight the parts which contain relevant information (like cell count, path delays, etc)

Submission Details

All parts of this lab are to be submitted on Canvas. No hard-copy submission is needed.

- Problem 1
  - Text file/Word document containing the answers
- Problem 2
  - Typed Verilog Code (.v file)
  - Typed Testbench Code (.v file)
- Problem 3
  - Typed Verilog Code (.v file)
  - Synthesis report (.txt or .doc file)
  - IO report (.txt or .doc file)
  - Utilization report with number of slices noted (.txt or .doc file)
  - Route Status report with number of nets with routing errors noted (.txt or .doc file)
  - Timing Summary report with critical path delay noted (.txt or .doc file)
  - [filename].bit file from compilation
  - XDC file

Checkout Details

During your checkout you will be expected to demonstrate each of the problems (both simulation and implementation if required for the problem) in the assignment and answer verbal questions about the assignment.