ECE 156A, 2016
Homework 3: Making Verilog Behave

Due: 11:59pm, 21 October 2016

Introduction

The purpose of this assignment is to continue your re-familiarization with Verilog. We will be examining Behavioral Verilog; what it is, and how to use it. This homework will also present an opportunity to refresh your memory on some aspects of digital design.

The tasks assigned in this homework will be used for future homeworks. Therefore it is critical that you complete these tasks. If there is something, in this homework, that is not clear to you, you should review the subject in the resource materials.

Resources

ECE156A Course Website:
https://groups.google.com/d/forum/forum/ucsb_ece156a_f16
Verilog Cheat Sheet:
https://www.cl.cam.ac.uk/teaching/0910/ECAD+Arch/files/verilogcheatsheet.pdf
ASIC WORLD:
http://www.asic-world.com/verilog/index.html
Testbench.in:
http://www.testbench.in/

Behavioral Verilog

The style of Behavioral Verilog is similar to a programming language, such as C or Python. There will be, virtually, no need of the gate primitives and for ECE156A you should not use the **assign** statement to encode a Boolean formula. Behavioral Verilog is more abstract than Structural Verilog, the feeling you should have is less like wiring gates and more like writing a software program. The abstraction of Behavioral Verilog will allow us to create larger projects, make modifications easier and speed up the development cycle.

Much like Homework 2 we have included two examples. They are the same two examples: RS Latch and D Latch. There are a few things to note about these examples. We have introduced the keyword **reg**. This just tells Verilog that this variable will only change values when explicitly assigned a value using the '=' or the '<=' assignment operators. The other thing to note is the concatenation operator ({}). This allows us to treat multiple wires as if they were a bus, if only temporarily. The concatenation operator can be used for reading or for assignment and is super useful for constructing case statements.

There is usually some confusion about the **reg** keyword. It does not necessarily imply a memory element. In Listing 1 we do use the **reg** to store information, but we could have easily used it to construct a combinational circuit as we did in Listing 4.

Listing 1: *Behavioral Verilog RS Latch*

```
module rs_latch_B(q, q_bar, r, s);
output q, q_bar;
input r, s;
reg q; // This allows us to assign values to q

assign q_bar = ~q; // One of the few allowed uses
                   // of the assign keyword for this class

always @ (r or s) // When ever r or s change,
begin             // run this block of code
  case({r, s})
    2'b00 : q <= ~q;   // Model the race condition
    2'b01 : q <= 1'b1; // Set
    2'b10 : q <= 1'b0; // Reset
  endcase
end

endmodule
```

Listing 2: *Behavioral Verilog D Latch*

```
module d_latch_B(clk, q, q_bar, d);
output q, q_bar;
input clk, d;
reg q;

assign q_bar = ~q;

always @ (clk or d)
begin
  if(clk) // True is any non-zero value
  begin
    q <= d;
  end
end

endmodule
```

Behavioral D Flip-Flop

Design

Using Behavioral Verilog and prototype in Listing 3 implement a D Flip-flop. The flip-flop should be positive edge triggered and there should be a synchronous reset. Essentially this flip-flop's behavior should be identical to the one you wrote in Homework 2. Not everything in this prototype is correct, you may even want to write it for yourself.

Listing 3: *D Flip-Flop Prototype*

```
module dff_B(clk, q, q_bar, d);
output q, q_bar;
input clk, d;

always @ (posedge clk)
begin
  // Your code here
end

endmodule
```

Testing

Create a test module and verify the functionality of the flip-flop. Keep in mind the basic behavior should be identical to the one you wrote before.

Behavioral Upcounter

Design

Using Behavioral Verilog implement a 4-bit counter. Like the structural version from homework 2 this one should count from 0 to 15 with wrap around. There should also be reset and enable signals. The design should be holistic. You should not build this from smaller components (e.g. flip-flops, gate primitives). Also there is no need to use the assign keyword.

Testing

Create a test module and verify the functionality of the counter circuit. Keep in mind the basic behavior should be identical to the one you wrote before.

Behavioral 7 Segment Decoder

Design

Using Behavioral Verilog implement and test a 7 segment decoder. See Homework 2 for details.

Note on Combinational Logic

The 7 segment decoder is different from the other two modules. The previous two modules are examples of sequential logic, where as the 7 segment decoder is purely combinational. When we

write combinational logic in Behavioral Verilog there are usually a few 'guidelines' that are used. First we use the '=' assignment operator. This is the so called blocking assignment operator. It sets up a definite ordering and hierarchy. The other 'guideline' is that we always have a default assignment. Leaving out a default assignment could result in there being a 'hidden' memory element. An example can be seen in Listing 4.

Listing 4: *Behavioral Combinational Logic*

```
module comb(y, a, b, c);
output reg y;
input a, b, c;

always @ (a or b or c)
begin
  y = 0;  // Without this statement the previous value would be stored in a memory element
  case ({a, b, c})
    3'd0 : y = 1; // We use '=' not '<='
    3'd2 : y = 0;
    3'd3 : y = 1;
  endcase
end
endmodule
```

Testing

Create a test module and verify the functionality of the 7 segment decoder. Keep in mind the basic behavior should be identical to the one you wrote before.

Report

Write everything you have done. What was created, the testing performed, and the results should appear in the report. This need not be a formal report. You should at least include an introduction, a procedure section, a results section and some conclusions.

Where appropriate you should include a waveform in your report. The waveform should be well annotated. The annotations should be good enough, that it is clear what has occurred. Each signal on the waveform should be labeled. Waveforms without labels or annotations will not be given full credit.

Homework Submission

Zip your report and Verilog files together, in one file, and email them to your TA. Please, do not use Rar or tar files.

Email Format:

Subject: ECE156A HW#3 ⟨your name⟩
Attachments: One zip file containing a clearly named report and all Verilog files

Late Policy

Homework, not turned in on time, will incur a 10 point loss for every day late. Submissions more than 5 days late will not be accepted. So, a homework with an initial score of 90/100, would be

given a score of 70/100 if it was two days late.

TA E-Mails

Matt: mrnero@umail.ucsb.edu
KK: kuokai@umail.ucsb.edu
Joel: jdick@umail.ucsb.edu
Bharath: bharathvishwanath@umail.ucsb.edu