Homework #2

Assigned: February 4, 2020

Due: February 11, 2020 at 11:59pm

This assignment is to be completed individually. Submit all files in a single tar-file.

This homework is intended to make you familiar with Verilog coding and simulation in preparation for the first phase of the course project. Refer to the posted Verilog tutorial for proper setup to run Verilog tools in this course. Read and follow the instructions very carefully.

A. (10 points) Basic Verilog Concepts

1. Give expected output of following operations for A= 4'b1101in binary format
   - **(a)** &A
   - **(b)** ~&A
   - **(c)** |A
   - **(d)** ~|A
   - **(e)** ^A
   - **(f)** ~^A

2. What will be binary output of the following comparisons of data1 = 4'b11x1 and data2 = 4'bx101?
   - **(a)** Case Comparison: data1 === data2
   - **(b)** Logical Comparison: data1 == data2

3. Give the expected binary output for the following operations for A=4'b1010 and B=4'b1x10
   - **(a)** A|B
   - **(b)** A+B
   - **(c)** A>B
   - **(d)** {{2{A}}, B}
   - **(e)** A && B
   - **(f)** A || B

4. How will the following constants get interpreted?
   For each of the constants, state whether the declaration is legal or illegal and provide a binary interpretation for each of the legal declarations.
   - **(a)** 4'x11
   - **(b)** 'h3C
   - **(c)** 12'HABC
   - **(d)** 4'b111011
   - **(e)** 4'b11??
   - **(f)** 8'b1100_1001

Submission: question1.txt

B. (10 points) Blocking/Non-blocking statement

Design five different behavioral Verilog models for a **2-input OR** gate and simulate them to identify and explain which models result in inertial delays and which result in transport delays and explain the reason. (Recall transport delays occur when all input changes "transport" through to the output; for inertial delays, input changes that are not stable for as long as the delay specified for the assignment get filtered out and do not propagate to the output.) You may include your input stimulus in the same file with the models or in a separate test-bench file. The same stimulus must be used for all models and is as follows:

@ time 0, A = 1, B = 1
@ time 4, A = 1, B = 0
@ time 7, A = 0, B = 0
@ time 11, A = 0, B = 1
@ time 14, A = 0, B = 0
End the simulation at time 20. The five models are:

Model 1: Procedural blocking assignment with inter-statement delay
#4 OUT1 = A | B;
Model 2: Procedural non-blocking assignment with inter-statement delay
#4 OUT2 <= A | B;
Model 3: Procedural blocking assignment with intra-statement delay
OUT3 = #4 A | B;
Model 4: Procedural non-blocking assignment with intra-statement delay
OUT4 <= #4 A | B;
Model 5: Continuous assignment
assign #4 OUT5 = A | B;

Use monitor and file I/O to store all states of the model in "delay.out" file. The format is as follow:

```
    At time XX ns, A =X, B=X, OUT1=X, OUT2=X, …
```

Submission: delay.v, tb_delay.v, delay.out, delay.pdf for explanations telling which model manifested which kind of delay and waveforms of your testbench signals

C. (20 points) D Flip-Flop Design

Design FOUR 1-bit NEGATIVE edge triggered D flip-flop models using RTL-style Verilog code with the constraints as specified in Table 1. The flip-flop has inputs called "clk" for clock signal, "rst" for reset signal, "en" for data enable for the flip-flop, and "d" is the actual data input for the flip-flop. The output of the flip-flop is "q". The input data is to be accepted by the flip-flop only when the data enable signal is active.

For all models, write a test-bench to verify its operations for all possible input combinations. Use a 100 MHz clock in your testbench and assert reset for 5-clock cycles at the beginning of simulations. Print the results in dff_<model_name>.out. Your output format should look like (where X replaced by actual value)

```
At time XX ns, clk =X, rst=X, en=X, d=X, q=X
```

```
module dff_<module_name>  (clk, rst, en, d, q);
    input clk, rst, en, d;
    output q;
    // your code
endmodule
```

Table 1: D flip-flop specifications and control mechanisms for enable and reset signals

Module name	Reset mechanism	Data enable
alpha	synchronous active HIGH	active HIGH
beta	asynchronous active HIGH	active LOW
gamma	synchronous active LOW	active HIGH
delta	asynchronous active LOW	active LOW

Submission: dff_apha.v, dff_beta.v, dff_gamma.v, dff_delta.v, tb_dff.v
(You may combine all the D-flip flop models with a single test-bench or name individual test benches as tb_dff_alpha.v, tb_dff_beta.v, tb_dff_gamma.v, tb_dff_delta.v)

D. (30 points) Sequence Detector

1. (10 points) Design an 8-bit shift register with a synchronous load, serial input and reset control mechanism operating at the POSITIVE edge of the clock signal "clk". It performs logical shift to left/right based on the value of direction "dir" signal. The shift register has the following interface:
Module Name: shift_register
Input: clk, rst, shift, load, dir, data [7:0], ser_in
Output: q [7:0]

Reset signal "rst" is a synchronous active HIGH signal. "load" is a synchronous active HIGH control signal used to control latching of new input data into the shift register. "shift" is a synchronous active HIGH control signal used to shift the data 1- bit either to the left or right depending on the "dir" signal value. Direction or "dir" signal controls the direction of the shift. When "dir" is HIGH the shift-RIGHT operations occurs. Similarly, when "dir" is LOW, shift-LEFT operation occurs.

The shift registers should conform to following specifications -
A. Order of priority for input signal to be processed is "rst", "load", and "shift".
B. When "rst" is active, It resets (clears) the shift register contents(zero).
C. When "load" is active, the value of "data" is written to the shift register.
D. When "shift" is active, depending on the value of "dir" signal, the data in the shift register (Q [7:0]) shifts left or right by one-bit. "ser_in" bit overwrites LSB or MSB in the left-shift or Right-shift respectively.

E. When “rst”, “load”, “shift” signals are inactive, the current value in shift register Q [7:0] remains unchanged.

Test your design using your test bench that covers all the cases explained above.

Submission shift_reg.v, tb_shift_reg.v

2. (20 points) Implement a sequence detector to detect pattern **“11010101”** (With respect to time, the pattern is read is specified from left to right, i.e., the leftmost bit is the earliest occurring bit of the pattern and the rightmost bit is the most recently occurring bit in the stream.) using the shift register designed above. You should instantiate the shift-register module in your sequence detector. The sequence detector uses the following module declarations:

```
module seq_detect (D_IN, CLK, RST, MATCH);
    input CLK, RST, D_IN;
    output MATCH;
    // your code
endmodule
```

CLK is clock signal. All events would occur at the POSITIVE edge of the CLK signal. The system reset signal is RST. It is an active HIGH synchronous reset signal. D_IN is the input data-stream. Design a test-bench to verify your design. Assert RST signal for 4-cycles at the beginning of the simulations. Your test-bench should read one bit at a time from an input file called “pattern.in” which contains a stream of 0’s and 1’s to be used as your input pattern. When the sequence detector detects the pattern, output signal “MATCH”, should go HIGH in the same clock cycle and remains high for 1-clock cycle. Use a 500 MHz clock in your test-bench. Your output format should look like (where X represents a binary value for that timestamp). Write your output to file “seq.out”.

```
    At time XX ns, CLK=X, RST=X, D_IN=X, MATCH=X
```

Submission: seq_detect.v, tb_seq.v, seq.out

E. (30 points) JTAG Test Access Port Controller State Machine

Design a JTAG Test Access Port (TAP) controller’s state machine as shown in following Figure 1. It has three 1-bit inputs TCK, TRST, TMS. Its output is STATE, a 4-bit number associated with the state of TAP. TCK is the clock of the TAP controller state machine. TRST is the synchronous active HIGH reset signal when active causes the machine to return to STATE#0 (test logic reset). TMS is the 1-bit signal controlling the complete state machine. Use the following state assignment shown in Table2. Assign a number to each state using the PARAMETER command.

Table 2: TAP Controller State Machine state assignments

State	STATE[3:0]	State	STATE[3:0]
Test logic reset	0000	Update DR	1000
Run test idle	0001	Select IR scan	1001
Select DR scan	0010	Capture IR	1010
Capture DR	0011	Shift IR	1011
Shift DR	0100	Exit 1 IR	1100
Exit1DR	0101	Pause IR	1101
Pause DR	0110	Exit 2 IR	1110
Exit2 DR	0111	Update IR	1111

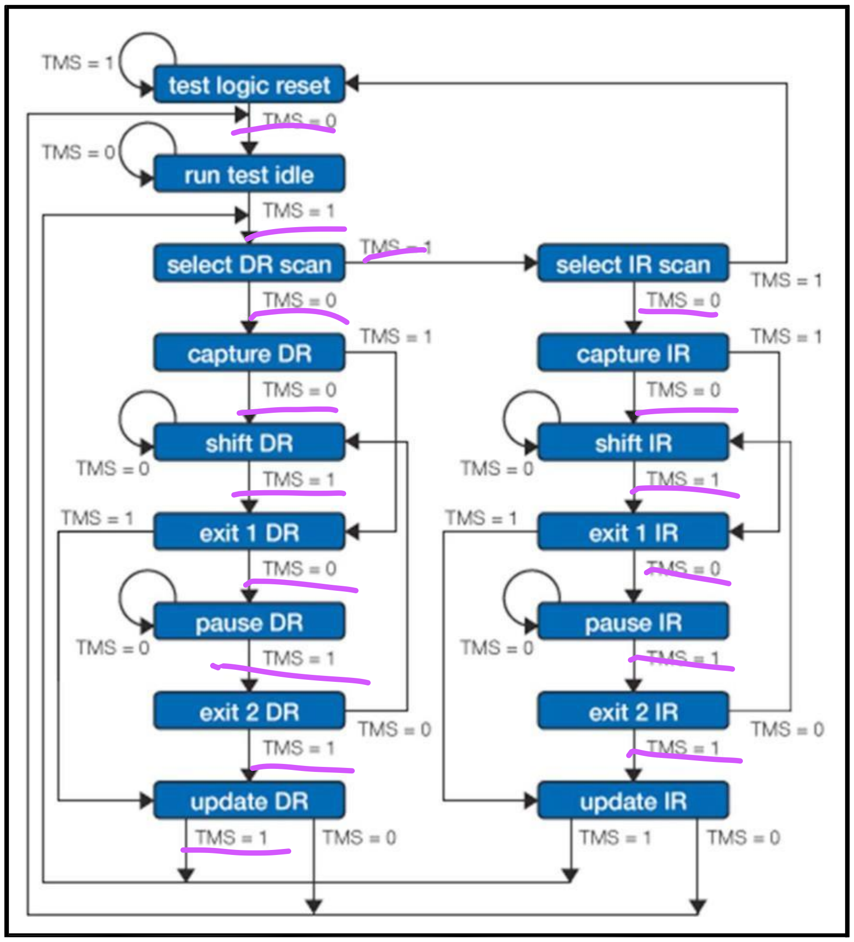

Figure 1: TAP Controller State Machine [1]

State Machine copied from http://www.xjtag.com/support-jtag/jtag-technical-guide.php

Name your TAP controller state machine module – "tap_controller". Write a test-bench to verify correctness of your state machine. Print the results in tap_controller.out. Your output format should look like (where X replaced by actual value). Use a 500MHz clock signal in your test bench. All operations happen at the POSITIVE edge of the clock.

```
At time XX ns, TMS=X, STATE=X
```

Use the following module declaration for your module “tap_controller” -

```
module tap_controller (TCLK, TRST, TMS, STATE);
     input TCLK, TRST, TMS;
     output STATE[3:0];
     // your code
endmodule
```

Submission: tap_contoller.v, tb_tap_controller.v, tap_controller.out

Submission Process

“tar” all the required files for HW#2 for electronic submission:

(1) Submit the “tar” file on BlackBoard-Assignment-HW2 submission for electronic submission