Verilog Session Assignment [2 hours Every Thursday]

All assignments are behavioural unless specified otherwise.

Session-1: Structural ADDER.

Make half-adder, full adder using gate descriptions. Show waveform output and RTL design generated by Xilinx. Write test-bench for each and show output on waveform. Also try "$monitor" command to generate values of output from the testbench. Complete 4-bit ripple carry and show at the beginning of next lab.

Session-2: Ripple-carry – delay analysis

Implement a 4-bit ripple carry adder with gate delays:
output sum and cout are generated after 2ns each.
Thus answer for first bit comes after 2ns.
Answer for second bit comes after 4ns and so on.

A 4-bit ripple carry must give output after 8ns.
Try various testcases and use monitor stmt to check the timing of final output generation.
Find the testcase which gives the worst case delay.

Ripple carry adders are slow due to the ripple effect. Therefore, Carry Look Ahead adders are better. Write the code for CLA adder for 4-bit and check the timings. Use equations for CLA logic that give out Pi and Gi for propagate and generate inputs. Each level of logic has 1ns delay. For example, if you have out= (a&b) | (c&d), then you have 2-level logic that uses AND gates followed by OR. So the delay for this will be 2ns.

Comment on which type of adder is good and when and why. Demonstrate your answer.

Session-3:

MUX

=====

Design a 4x1 MUx using following type of stmts:

Output = equations for AND-gates that are inside the mux. Inputs are i0,i1,i2,i3 and select=s1,s0.

Also write output = conditional assignment

Cond ? : stmt-1 : stmt-2

Synthesise both types are check the difference in RTL schematic. Also read the design summary generated and see if the final designs are different or not.

ALU

====

Design 4-bit ALU:

The 4-bit ALU has the following inputs:

A: 4-bit, B: 4-bit , Cin: 1-bit, Control: 3-bit control input

Output: ANS: 4-bit, Cout: 1-bit

Control Instruction Operation

000 ADD Output <= A + B + Cin; Cout contains the carry

001 SUB Output <= A – B - Cin; Cout contains the borrow

010 OR Output <= A or B

011 AND Output <= A and B

100 SHL Output <= A[2:0] & ‘0’

101 SHR Output <= ‘0’ & A[3:1]

110 ROL (Rotate left) Output <= A[2:0] & A[3]

111 ROR (Rotate right) Output <= A[0] & A [3:1]

Use 'case' statement. Design is purely combinational. Output should change as soon as any input or control combination changes.

Subtraction function must give the correct answer and indicate overflows.

Make sure there are no latches in the synthesised design (RTL schematic).

Try removing one case from the ‘case’ stmt and see the RTL. Compare the RTL with all case options. Identify the difference.

How many MUX are there in your RTL? 1 or 2? What is their purpose?

Session-4:

(1)

Design a 3-bit register using a serial input “a”. Input goes to first flip-flop (d0). Output of do which is q0 goes as input to second flip-flop d1-q1. Output of second flip-flop goes to third one d3-q3.

take inputs as d0,d1,d2 and outputs as q0,q1,q2. Don't take multibit registers.

Design using an always @clk statement.

(i) Change the order of your assignments within the always blocks.

(ii) Also implement the block statements using “<=” as assignment operator instead of “=”.

(ii) For both type of assignment operators and all possible reordering of stmts in always block check the RTL generated. Is it same or different?

(iv) Also check the “synthesis report” in the “design summary” and find out delay of our circuit under “HDL synthesis”.

(2)

Take multi-bit inputs and outputs d0-d3 and q0-q3. Design 4-bit register. that has LOAD and CLEAR as control inputs. CLEAR input has priority over LOAD.

Use parallel load feature.

Write appropriate testbench and show that CLEAR has priority.

Testbench mush generate CLK as a continuous waveform.

Session-5:

First make a 4-bit register with LOAD, CLEAR features.

Test it by loading and clearing at random time intervals.

Testcase will have CLK generated using for-loop.

Use the above register and change it to a up-counter. There is additional input INCR. If counter is enabled then it increments.

Write elaborate testcases to demonstrate all features.

Cascade the above counter with another one to generate a 8-bit up-counter. You have to use the 4-bit counter as a module for this design.

Make necessary changes in the 4-bit version if required.

Write proper testcases.

[BONUS]

BCD up-down counter

counter parameters:

Input = Enable, Load, Up, CLR (= async clear), Data-in

Carry-out, Data-out

counter works only when Enable=1.

LOAD=1, Data-in gets loaded in the counter

CLR=1, Counter is reset to 0, asynchronously

LOAD=0 & UP=1 : increment. when counter reaches 9, it makes carry-out=1 and value=0

LOAD=0 & UP=0 : decrement. when counter reaches 0, it makes carry-out=0

use clock period = 10 ns. Use for-loop in testbench to generate clock signal. Try different delay gaps between clk and load/clr and verify that the count responds only to the clock even when the load/clear are given earlier.

!!! MID-SEM !!!

Session-6: Sequence Detector – Parity Checker [Deadline – 13 April]

Draw the FSM for parity checker. Draw Moore as well as Mealy FSMs. Write Verilog code for the FSMs.

What is the difference between Moore and Mealy FSMs ?

Moore Code will have 3 always blocks:

Always block @clk to change state.

Always block @ input and state to compute next-state

Always block @ state to compute output.

Mealy code will have 2 always blocks:

Always block @clk to change state.

Always block @ input and state to compute next-state and output.

Synthesise the circuits. Show waveform outputs. Use forever-loop to generate clock with period=10ns. Change input at various time intervals using delay stmt in test-file.

Identify the difference in output of Moore vs Mealy.

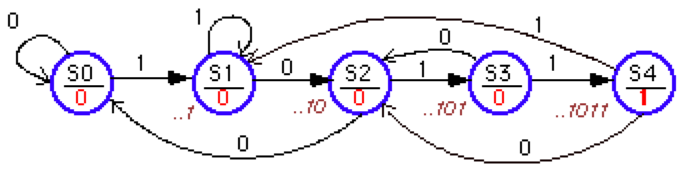
Note the time when the output is generated by both machines.

(insert a minor delay for Mealy machine output to observe changes in state and output timings.)

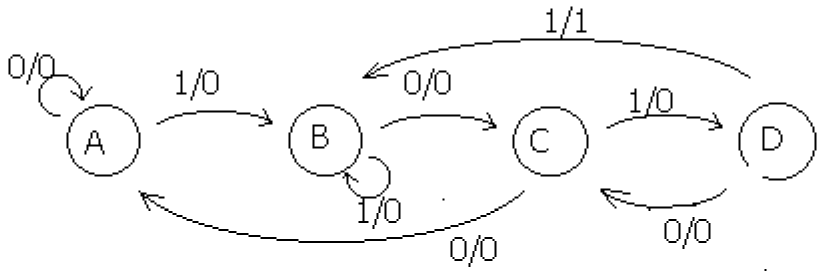
Session-7: Sequence Detector – for sequence 1011 [Deadline – 13 April]

Similar to the above problem, write a state machine based Verilog code to detect the sequence of 1011. Input comes as single bits one by one and the output is also one bit. The output becomes 1 when the sequence 1011 is seen at the input. In all other cases output remains 0. Design the machine to detect overlapping sequences. Design both Moore and Mealy machines.

Following is FSM for Moore machine:



Following is FSM for Mealy machine:



State diagram of 1011 sequence recognizer

Session-8: Booth Multiplier [Deadline – 20 April]

You have studied Booth Multiplication algorithm in the digital design theory course. Implement the same algorithm in Verilog on 4-bit input data.

Session-9: Sequential Multiplier using ASM charts [Deadline – 27 April]

In the chapter on ASM – Algorithmic State Machines, you must have studied the sequential multiplier. Implement the same in Verilog. Write separate modules for controller design, and datapath components.

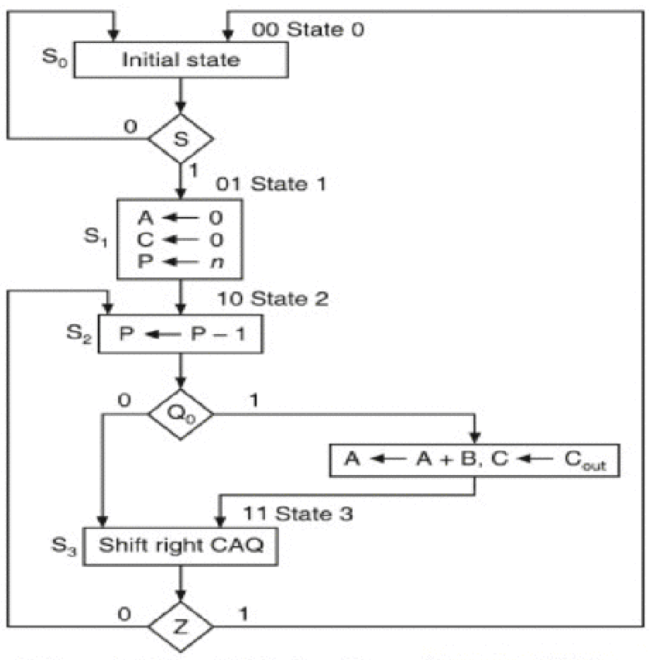


Figure 15.21 ASM chart for a binary multiplier.

Session-10: [Deadline – 4 May]

To be announced

END-SEM