ASIC Design and Modeling

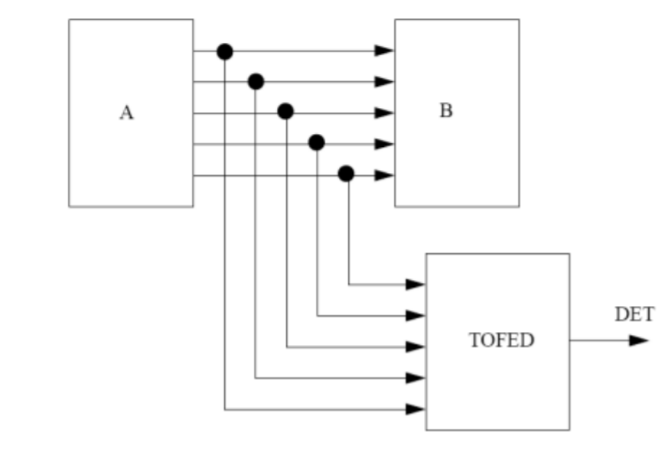
ECE – 581

Project – 1

Engineers: 1] Yadnesh Samant (ysamant@pdx.edu)

2] Hiral Barot (bhiral@pdx.edu)

Task 1: In a Two-out-of-Five code word, exactly two bits are logic high and exactly three bits are logic low. A block diagram of a typical system is shown below. Component A produces an output coded in the Two-out of-Five format which is given as an input to component B. The Two out-of-Five code error detector (TOFED) observes the five bits flowing from component A to component B and outputs logic high when an error is detected.



* 1. • Write a SV sequential (always\_comb) model of TOFED having a delay of 10ns. The input should be named CODE. The output should be named DET and must be of type bit.

• Write a testbench and simulate the design using an HDL simulator.

**Task 1.1 Description:**

1. I have implemented task using always\_comb block, at start of block all internal variables are set to zero to avoid data corruption between two successive inputs.
2. Logic is implemented using two counters for counting number of zero’s and one’s. Based upon the number of one’s and zero’s the output signal DET is asserted low or high. Below is code snippet to elaborate the logic design.

|  |
| --- |
|  |

c)Test bench results along with console output are as below:

|  |
| --- |
| https://raw.githubusercontent.com/baba256/ECE581_proj/master/Task1.PNG |

|  |
| --- |
| https://raw.githubusercontent.com/baba256/ECE581_proj/master/Task1_Monitor.PNG |

* 1. • Write a SV dataflow (continuous assignments) model of TOFED having the same nomenclature and type for the input and output as in task 1 above.

• Using the same testbench as in task 1 above, simulate the design using an HDL simulator.

Task 1.2 Description:

1. The continuous assignment model for TOFED having same input and output is implemented by adding the number of bits in each CODE and verifying the addition of bits is equal to 3’b010(3’d2).
2. If the addition of bits is 3’d2 then the output DET is held low or else assigned high using ternary operator.

|  |
| --- |
|  |

* 1. • Write a SV structural model of TOFED having the same nomenclature and type for the input and output as in task 1 above. Use only NAND and NOT gates. You have to create behavioral (data flow or sequential) models for these gates. Assign a 5ns delay for each NAND gate and a 2ns delay for each NOT gate.

• Using the same testbench as in task 1 above, simulate the design using an HDL simulator.

**Task 1.3 Description:**

1. The SV Structural model of TOFED using NAND and NOT gates is implemented using below equation.

Assumption: CODE [4:0] corresponds to a=CODE [0], b=CODE [1], c=CODE [2], d=CODE [3], e=CODE [4]

For 5-bit input, there would 5C2=10 combinations since there needs to be only 2 bits high(1b’1).

|  |
| --- |
| *Output Equation = abc’d’e’+acb’d’e’+adb’c’e’+aeb’c’d’+bca’d’e’+bea’c’d’+bda’c’e’+cda’b’e’+cea’b’d’+dea’b’c’* |

1. AND logic is implemented using two NAND gates as follows:

|  |
| --- |
| a c=ab a  b b c=ab |

1. OR Logic is implemented using NOT gate followed by NAND gate.

|  |
| --- |
| a c= a+ b a' a  b a |

1. Code snippet is provided below:

|  |
| --- |
|  |

Common testbench for all above task is as below:

|  |
| --- |
|  |

Task 2: For the following problems, implement them in SV with testbench and simulate  
the design using an HDL simulator.

**2.1)** A full adder has the truth table for its sum (S) and carry (Co) outputs, in terms of its inputs, A,  
B and carry in (Ci).

1] Derive expressions for S and Co using only AND and OR operators. Hence write a SystemVerilog description of a full adder as a netlist of AND and OR gates and inverters. Do not include any gate delays in your models.

**Answer]** Module Name: hw\_sum\_tb …. (Please refer to the code with module name)

Code Snippet:

always\_comb

begin

Co = (A & B) | (B & C) | (A & C);

S = (~A & ~B & C) | (~A & B & ~C) | (A & ~B & ~C) | (A & B & C);

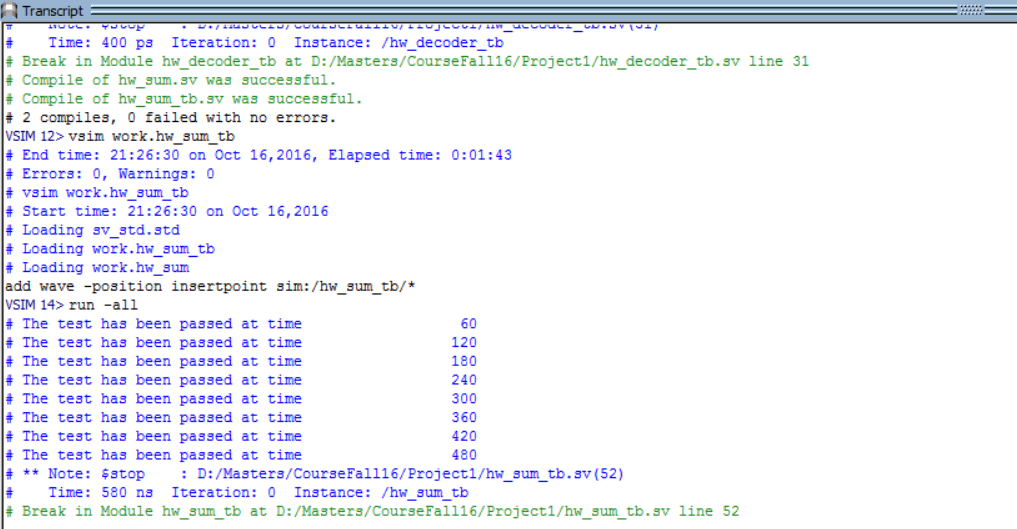
end

2] Write a SystemVerilog testbench to test all combinations of inputs to the full adder of 1). Verify the correctness of your full adder and of the testbench using a SystemVerilog simulator.

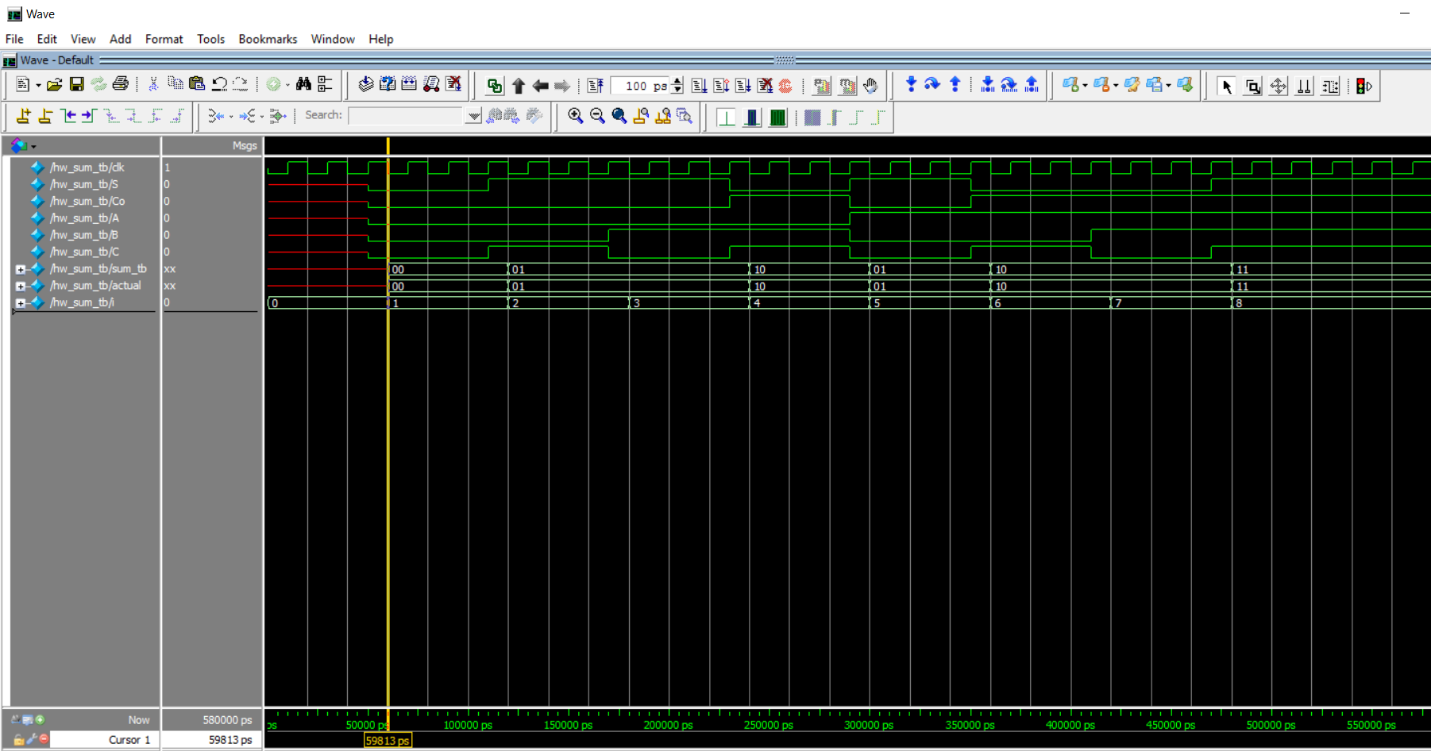
**Answer]** Module Name: hw\_sum\_tb …. (Please refer to the code with module name)

**Description:** This is a self-checking test bench which will shell out the error if mis-match is found. A function is created which will generate the “expected sum” which is used to compare with the actual sum and carry generated by the Design under test. If the mismatch is found, error will be displayed in the console.

**Console output:**

****

**Waveforms:**

****

**2.2)** Write SystemVerilog models of a 3 to 8 decoder using (a) Boolean operators, (b) a conditional operator and (c) a shift operator. Write a testbench to compare the three versions.

**A] Boolean Operator**

Module name: hw\_3\_8\_decoder …. (Please refer to the code with module name)

Description: Boolean operators are also called as Logical operators.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **INPUT** | | | **OUTPUT** | | | | | | | |
| **s2** | **s1** | **s0** | **q0** | **q1** | **q2** | **q3** | **q4** | **q5** | **q6** | **q7** |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Boolean Equations: (~ : NOT)

Q0 = ~s2~s1~s0 Q4 = s2~s1~s0

Q1 = ~s2~s1 s0 Q5 = s2~s1 s0

Q2 = ~s2 s1~s0 Q6 = s2 s1~s0

Q3 = ~s2 s1 s0 Q7 = s2 s1 s0

Code snippet:

always\_comb

begin

if(EN == 1)begin

q[0] = (~s[2] && ~s[1] && ~s[0]);

q[1] = (~s[2] && ~s[1] && s[0]);

q[2] = (~s[2] && s[1] && ~s[0]);

q[3] = (~s[2] && s[1] && s[0]);

q[4] = ( s[2] && ~s[1] && ~s[0]);

q[5] = ( s[2] && ~s[1] && s[0]);

q[6] = ( s[2] && s[1] && ~s[0]);

q[7] = ( s[2] && s[1] && s[0]);

end

else

q=8'b0;

end

**B] Conditional Operator**

**Answer]** Module Name: hw\_3\_8\_decoder\_cond …. (Please refer to the code with module name)

Description: Conditional operators are also called as Ternary operators, which when synthesized form a MUX structure.

Code Snippet:

always\_comb

begin

if(EN == 1)begin

q[0] = (s == 3'd0) ? 1'b1 : 1'b0;

q[1] = (s == 3'd1) ? 1'b1 : 1'b0;

q[2] = (s == 3'd2) ? 1'b1 : 1'b0;

q[3] = (s == 3'd3) ? 1'b1 : 1'b0;

q[4] = (s == 3'd4) ? 1'b1 : 1'b0;

q[5] = (s == 3'd5) ? 1'b1 : 1'b0;

q[6] = (s == 3'd6) ? 1'b1 : 1'b0;

q[7] = (s == 3'd7) ? 1'b1 : 1'b0;

end

else

q=8'b0;

end

**C] Shift Operator:**

Answer] Module Name: hw\_3\_8\_decoder\_shift …. (Please refer to the code with module name)

Description: Q0 bit s loaded with 1 and is Left-shifted based on the input given to the module.

Code Snippet:

//Temporary value to hold q[0]

logic [7:0] q\_temp = 8'b1;

always\_comb

begin

if(EN == 1) begin

q = q\_temp << s;

end

else

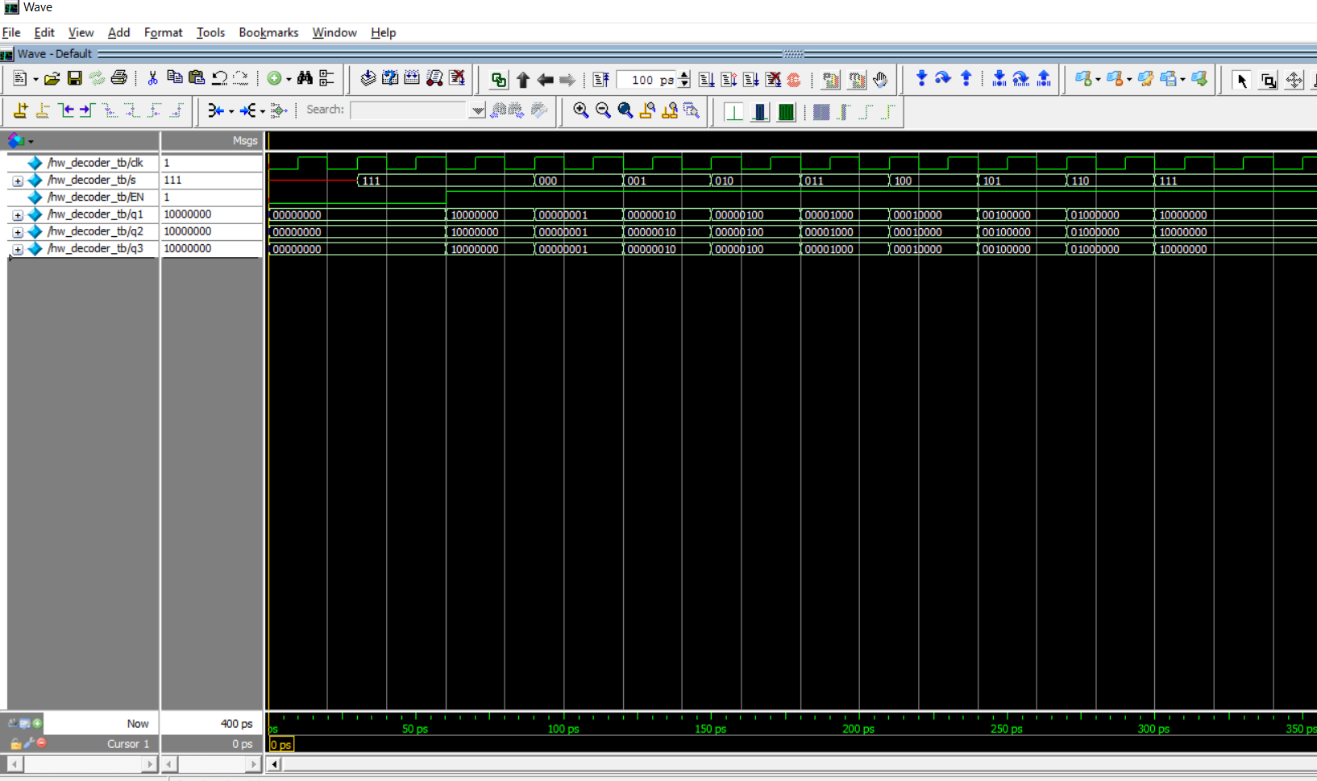
q=8'b0;

end

**Test Bench:**

**Module Name:** hw\_decoder\_tb

**Waveform:**

****

**2.3]** Write a SystemVerilog model of a 2n to n priority encoder.

Module name: hw\_priority\_encoder …. (Please refer to the code with module name)

Description: The priority encoder is designed such that if MSB bit has a SET value of 1, all the bits below i.e LSBs will be ignored. A parameter is created, which takes a value from the instantiation. For eg. If n=3, the from 2^(3)=8 inputs, bit 7 will have higher priority over bit 0.

Eg. If input =>enc\_in[7:0] = 8’b01100000, as bit 6 i.e enc\_in[6]=1, it will be given higher priority over bit 5. Thus the output => enc\_op = 3’b110

**Code Snippet:**

always\_comb

begin

if(enc\_in == 0)

enc\_op= 'x;

else

for(i=0;i<2\*\*n;i=i+1) begin

if(enc\_in[i] == 1'b1)

enc\_op = i;

end

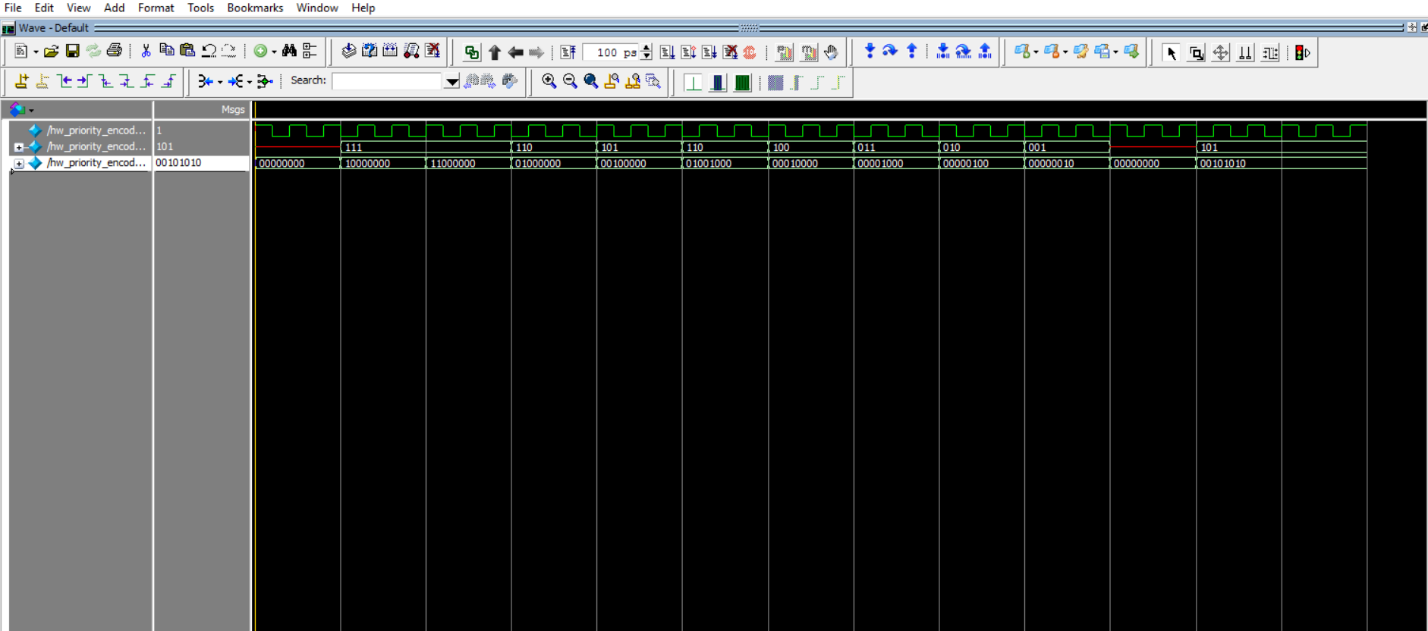
end

**TestBench:**

Module name: hw\_priority\_encoder\_tb … (Please refer to the code with module name)

Description: Various 8 bits inputs were given to the design under test, and the behavior was validated using the waveform.

**WaveForm:**

****

2.4) A comparator is used to to determine whether two signals have equal values. A one-bit comparator is described by

eqo = ˜(x ˆ y) & eqi;

where eqi is the result of the comparison of other bits and eqo is passed to the next comparison operation. Write a model of an n-bit iterative comparator.

**Task 2.4 Description:** N-bit iterative comparator is implemented using for loop for N bits and parameter N is declared while defining module. Code snippet is provided below:

|  |
| --- |
|  |

**Test bench for N-bit comparator is as below:**

|  |
| --- |
|  |

**Waveform for the above simulation:**

|  |
| --- |
|  |