** San Francisco Bay University**

**EE461L - Verilog HDL Lab**

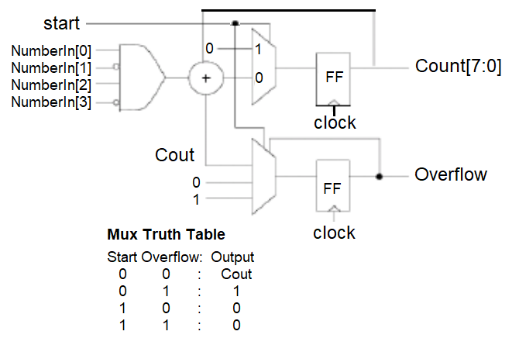
**EE461 Digital Design and HDL**

**Week#5 Sequential Logic**

Name: Melvin Divine Pritchard Student ID: 19857

**V Exercises**

1. Convert the following circuit into Verilog module and write the testbench to verify the design.



module counter(

input clock, /\* clock \*/

input [3:0] in, /\* starting count \*/

input start,

output reg [3:0] Count, /\* high when count down to zero \*/

output wire Overflow

);

reg Cout;

always@(posedge clock) begin

if (start) begin

Count <= in;

end

else begin

{Cout,Count} <= Count + 3'b1;

end

end

assign Overflow = (Cout == 1'b1);

endmodule

1. What does logic statement specify for the hardware in the module?

*`timescale 1ns/100ps*

*module xor3 (input B\_i, D\_i, sel\_i, clock, output E\_o);*

*reg E\_o;*

*always@(posedge clock) begin*

*case(sel\_i)*

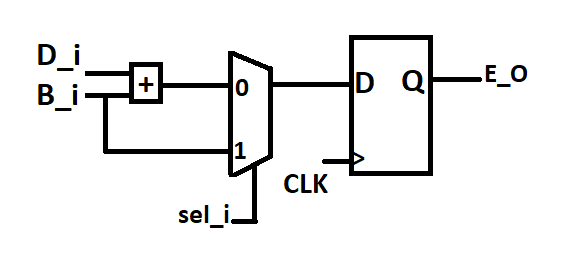
*0: E\_o <= D\_i + B\_i;*

*1: E\_o <= B\_i;*

*endcase*

*end*

*endmodule*



According to the logic statement and schematic above,this is a three-input XOR gate with a MUX that is controlled by the input sel\_o, . B\_i and D\_i are the other two inputs, and E\_O is the only output port. The output port E\_O is given a value based on the selected input using a case statement in an always block, and the statement's clock signal (clock) is used to synchronize the input ports.

The value of the sum of the inputs D\_i and B\_i is allocated to the output E\_O in the case that sel\_i is 0, which is identical to an XOR operation. The value of B i is transmitted to the output E o if sel\_i is 1. This transfers the value of the input B\_i directly to the output E\_O, skipping the XOR procedure.

E\_O register declaration indicates that it is a register variable, which means that its value is updated on the rising edge of the clock signal and is stored in a flip-flop. As a result, the output E o is synchronized with the clock and will only change in value when the clock signal rises.

1. Modify design in **Lab Procedures II** so it is decremented by *two,* and then stops, setting zero flag high, when it reaches 0000 or 0001. After that, verify the design in the testbench.

module counter (clock, in, latch, dec, zero);

input clock; //clock

input [3:0] in; //starting count

input latch; //latch `in’ when high

input dec; //decrement count when dec high

output reg zero; //high when count down to zero

reg [3:0] value; //current count value

always@(posedge clock) begin

if (latch) value <= in;

else if (dec && !zero && value > 2'b01) value <= value - 2'b10;

if (value <= 2'b01) zero <= 1'b1;

else zero <= 1'b0;

end

endmodule //counter

module counter\_tb;

// Inputs

reg clock;

reg [3:0] in;

reg latch;

reg dec;

// Outputs

wire zero;

// Instantiate the Unit Under Test (UUT)

counter uut (

.clock(clock),

.in(in),

.latch(latch),

.dec(dec),

.zero(zero)

);

initial begin

// Initialize inputs

clock = 0;

in = 4'b0000;

latch = 0;

dec = 0;

// Dump waveform to VCD file

$dumpfile("dump.vcd");

$dumpvars;

// Wait 5 clock cycles

#5;

// Test latch functionality

latch = 1;

in = 4'b1010;

#5;

latch = 0;

#5;

// Test decrement functionality

dec = 1;

#5;

dec = 0;

#5;

dec = 1;

#5;

dec = 0;

#5;

// End simulation

$finish;

end

always #1 clock = ~clock;

endmodule