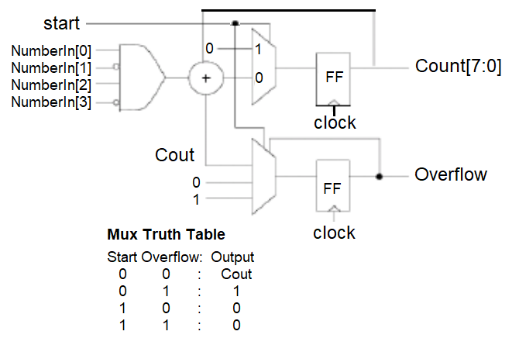
**-Convert the following circuit into Verilog module and write the testbench to verify the design.**



module Circuit (

input [3:0] NumberIn,

input start,

input clock,

output reg [7:0] Count,

output reg Overflow,

output Cout // Make Cout an output so it can be monitored in the testbench

);

wire [3:0] add\_out;

reg next\_overflow;

// 4-bit Adder: Adds NumberIn and start (start as LSB)

assign {Cout, add\_out} = NumberIn + {3'b000, start};

// Determine next value of Overflow based on the truth table

always @(\*) begin

if (start == 0 && Overflow == 0)

next\_overflow = Cout;

else if (start == 0 && Overflow == 1)

next\_overflow = 1;

else

next\_overflow = 0;

end

// Initialize Count and Overflow to avoid 'x' state

initial begin

Count = 8'b00000000;

Overflow = 0;

end

// Update Count and Overflow on each clock edge

always @(posedge clock) begin

Count <= {Count[6:0], add\_out[0]};

Overflow <= next\_overflow;

end

endmodule

//tb

module Testbench;

reg [3:0] NumberIn;

reg start;

reg clock;

wire [7:0] Count;

wire Overflow;

wire Cout;

// Instantiate the Circuit module

Circuit uut (

.NumberIn(NumberIn),

.start(start),

.clock(clock),

.Count(Count),

.Overflow(Overflow),

.Cout(Cout)

);

// Clock generation

initial begin

clock = 0;

forever #5 clock = ~clock; // 10 ns clock period

end

// Test procedure

initial begin

// Initialize inputs

NumberIn = 4'b0000;

start = 0;

// Case 1: Start = 0, Initial Overflow = 0, Cout from addition

#10 start = 0; NumberIn = 4'b0011; // Expect Overflow to follow Cout

#10 $display("Case 1 -> Time = %0t, Start = %b, NumberIn = %b, Cout = %b, Count = %b, Overflow = %b",

$time, start, NumberIn, Cout, Count, Overflow);

// Case 2: Start = 0, Set Initial Overflow = 1

#10 start = 0; NumberIn = 4'b0101;

uut.Overflow = 1; // Manually set initial Overflow to 1

#10 $display("Case 2 -> Time = %0t, Start = %b, NumberIn = %b, Cout = %b, Count = %b, Overflow = %b",

$time, start, NumberIn, Cout, Count, Overflow);

// Case 3: Start = 1, Overflow = 0

#10 start = 1; NumberIn = 4'b1111;

#10 $display("Case 3 -> Time = %0t, Start = %b, NumberIn = %b, Cout = %b, Count = %b, Overflow = %b",

$time, start, NumberIn, Cout, Count, Overflow);

// Case 4: Start = 1, Set Initial Overflow = 1

#10 start = 1; NumberIn = 4'b1001;

uut.Overflow = 1; // Manually set initial Overflow to 1

#10 $display("Case 4 -> Time = %0t, Start = %b, NumberIn = %b, Cout = %b, Count = %b, Overflow = %b",

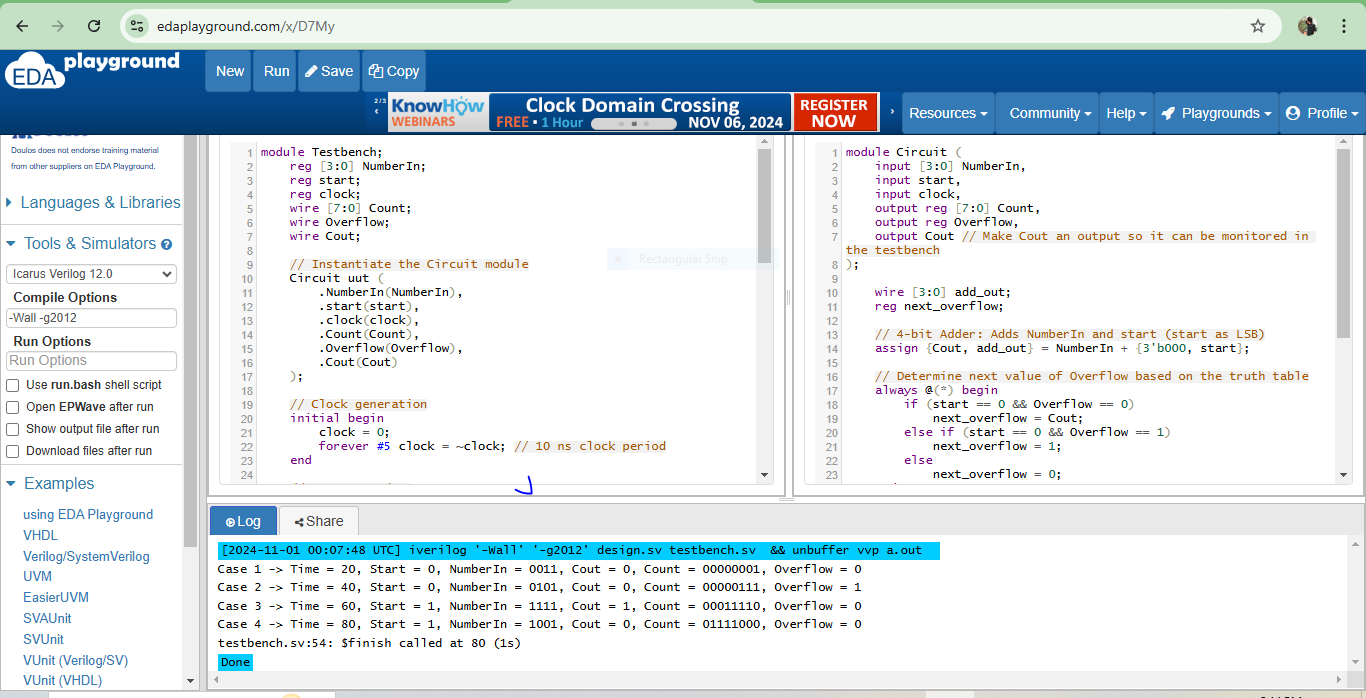
$time, start, NumberIn, Cout, Count, Overflow);

// Finish simulation without additional messages

$finish;

end

endmodule



**- 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***

The logic statement in this xor3 module specifies the behavior of the output E\_o based on the input signals B\_i, D\_i, sel\_i, and clock.

**Code Breakdown**

1. **timescale 1ns/100ps**:

-This specifies the time unit and time precision for simulation.

-1ns means that the time unit in the simulation is 1 nanosecond, and 100ps means the simulation precision is up to 100 picoseconds.

1. **Module Definition**:

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

-The module has four inputs: B\_i, D\_i, sel\_i, and clock, and one output E\_o.

Note: This module is named xor3, which may be misleading because it doesn't perform an XOR operation. It actually implements a multiplexer-like logic based on the sel\_i input.

1. **always Block**:

-always@(posedge clock) triggers on the rising edge of the clock signal. This means E\_o is updated only on the positive edge of the clock.

-Inside the always block, a case statement controls the behavior of E\_o based on the value of sel\_i.

1. **Behavior of E\_o Based on sel\_i**:

-case(sel\_i) checks the value of sel\_i.

-When sel\_i is 0: E\_o <= D\_i + B\_i;

-This adds D\_i and B\_i and assigns the result to E\_o. This is equivalent to an OR operation in Verilog because D\_i and B\_i are treated as 1-bit values.

-When sel\_i is 1: E\_o <= B\_i;

-This assigns the value of B\_i directly to E\_o.

**-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 high

output zero; // high when count reaches 0000 or 0001

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

// Always block triggered on positive edge of clock

always @(posedge clock) begin

if (latch)

value <= in; // Load value with 'in' if latch is high

else if (dec && !zero)

value <= value - 2'b10; // Decrement by 2 if dec is high and zero flag is not set

end

// Set zero flag high if value is 0000 or 0001

assign zero = (value == 4'b0000) || (value == 4'b0001);

endmodule

**//tb**

module Testbench;

reg clock;

reg [3:0] in;

reg latch;

reg dec;

wire zero;

wire [3:0] value;

// Instantiate the counter module

counter uut (

.clock(clock),

.in(in),

.latch(latch),

.dec(dec),

.zero(zero)

);

// Monitor the value for debugging

assign value = uut.value;

// Clock generation

initial begin

clock = 0;

forever #5 clock = ~clock; // 10 ns clock period

end

// Test procedure

initial begin

// Initial setup

in = 4'b0101; // Load with 5 (0101)

latch = 1;

dec = 0;

#10 latch = 0; dec = 1; // Start decrementing by 2

// Display output at each clock cycle up to 40 ns

repeat (4) begin

#10 $display("Time = %0t, Value = %b, Zero = %b", $time, value, zero);

end

$finish;

end

endmodule

