Ryan Writz

Priyam Shah

ECE 4/581

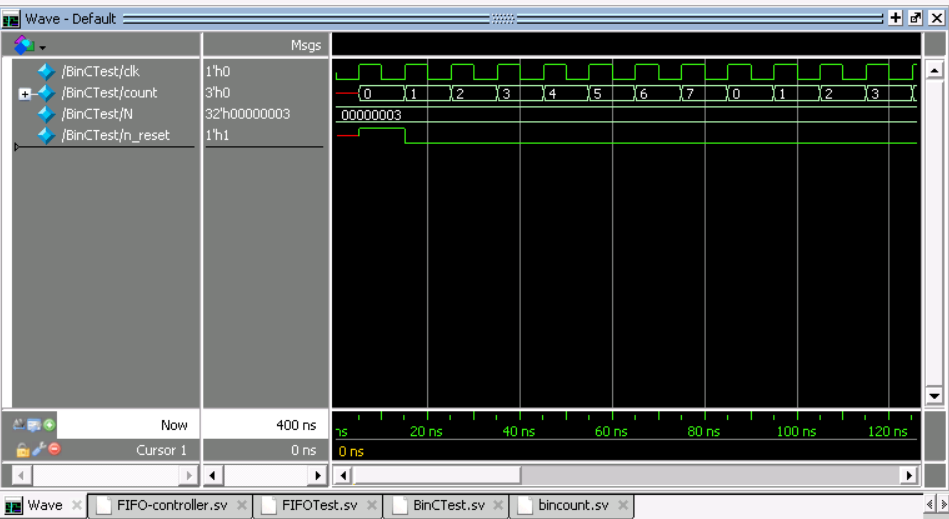
Project 2

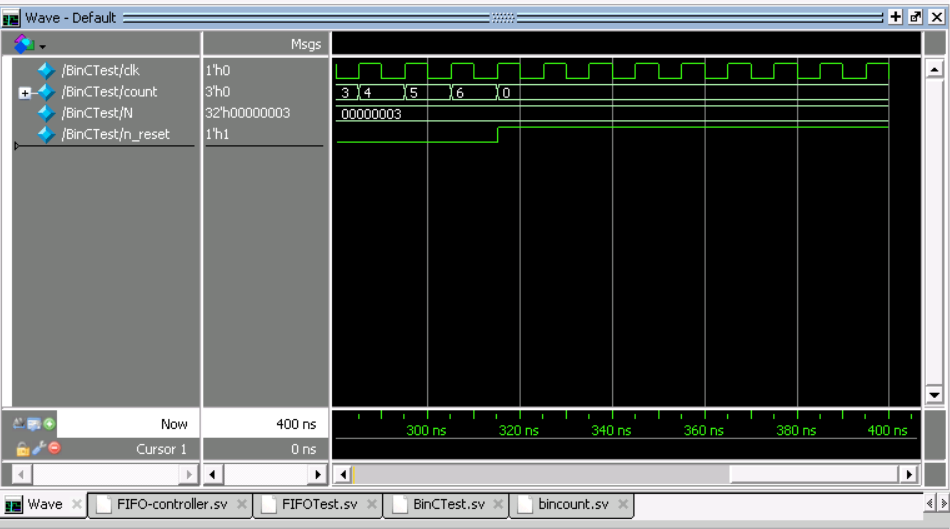
7/22/19

1:

Binary counter counts up simply by one bit ach clock tick and upon reaching maximum loops to 0.

Binary Counter Testing for 3 bits





FIFO-controller.sv

//Ryan Writz and Priyam Shah (c) July 2019

//ECE 4/581 Project 2 Question 1

//SystemVerilog model of a FIFO controller where a FIFO,

//is a type of memory that stores data serially, where the

//first word read is the first word that was stored.

//The module controls the reading and writing of data from/into a FIFO.

//The FIFO is a two-port RAM array having separate read and write data

//buses, separate read and write address buses, a write signal and

//a read signal. The size of the RAM array is 32 x 8 bits.

//Data is read from and written into the FIFO at the same rate

//(a very trivial case of the FIFO).

module FIFOControl #(parameter N = 4)

(output logic [N-1:0] rd\_ptr, wr\_ptr,

output logic wr\_en, rd\_en, emp, full,

input logic clk, wr, rd,f\_reset);

logic [N:0] count\_rd = 0;

logic [N:0] count\_wr = 0;

logic rd\_reset,wr\_reset;

bincount #(N) BCRD(count\_rd,rd\_reset,clk); //binary counter of rd\_ptr to maintain empty/full conditions

bincount #(N) BCWR(count\_wr,wr\_reset,clk); //binarhy counter of wr\_ptr to maintain empty/full conditions

//Intializing the FIFO pointers

always\_ff @(posedge f\_reset)

begin

rd\_reset <= f\_reset; //makes f\_reset serve as master reset so that all resets on same page

wr\_reset <= f\_reset;

end

always\_ff @ (posedge clk, posedge f\_reset)

begin

if(f\_reset) //At reset, both pointers are initialized to point to the first location of the FIFO

begin //emp is made high and full is made low.

rd\_ptr <= 0;

wr\_ptr <= 0;

emp <= 1;

full <= 0;

end

if(rd & ~emp) //If an external device wishes to read data from the FIFO by

begin

rd\_en <= 1; //asserting rd, then the controller asserts rd\_en only if emp is deasserted.

end

if(wr & full) //A similar logic exists for the write operation. The crux of this design

wr\_en <= 1; //is in determining the conditions which lead to the assertion/deassertion

//of the emp and full signals.

if(rd\_en) //The read pointer rd\_ptr contains the address of the next FIFO location to be read

begin

//rd operation here

rd\_ptr <= rd\_ptr + 1;

end

if(wr\_en) //write pointer wr\_ptr contains the address of the next FIFO location to be written

begin

//wr operation here

wr\_ptr <= wr\_ptr + 1;

end

if(~f\_reset & count\_rd[N] != count\_wr[N])

full <= 1;

if(~f\_reset & count\_rd[N] == count\_wr[N])

emp <= 1;

if(full)

wr\_en <= 0;

else if(emp)

rd\_en <= 0;

end

endmodule

FIFO-Test.sv

module FIFOTest();

parameter N = 4; //Number of initial bits for pointers

logic clk, n\_reset;

logic [N:0]count;

FIFOControl #(N) FIFO(rd\_ptr, wr\_ptr, wr\_en,

rd\_en, emp, full, clk, wr, rd,f\_reset);

initial

begin

clk = 0;

forever #5 clk = ~clk;

end

initial

#5 f\_reset = 1; //Start two pointers at same location with reset

#5 f\_reset = 0; rd = 1; //Tries to read but will fail since empty

#5 wr = 1; //writes for 5 cycles

#25 wr = 0; rd = 1; //read for 5 cycles

end

endmodule

bincount.sv

//Simple N bit binary counter with

//active high asynchronous reset

module bincount #(parameter N = 4)

(output logic [N:0]count,

input logic n\_reset, clk);

always\_ff @ (posedge clk, negedge n\_reset)

begin

if(n\_reset)

count <= 0;

else

count <= count + 1;

end

endmodule

BinTest.sv

module BinCTest();

parameter N = 3;

logic clk, n\_reset;

logic [N - 1:0]count;

bincount #(N) BC(count, n\_reset, clk);

initial

begin

clk = 0;

forever #5 clk = ~clk;

end

initial

begin

#5 n\_reset = 1;

#10 n\_reset = 0;

#300 n\_reset = 1;

end

endmodule

2:

typedef enum { OFF, //power off

RED, //red state

YELLOW, //yellow state

GREEN, //green state

PRE\_GREEN //state before green

}

lights\_t;

module trafficlight

(

output lights\_t ns\_light, //North-South light status, main road

output lights\_t ew\_light, //East-West light status

input ew\_sensor, //East-West sensor for new car

input emgcy\_sensor, //emergency sensor

input reset\_n, //synchronous reset

input clk //master clock

); //khensu/Home02/pshah/Downloads/P2 Ryan Edit.sv

timeunit 1ns;

timeprecision 100ps;

parameter FAIL =1'b0;

logic [1:0] ns\_green\_timer; //timer for NS light in GREEN//khensu/Home02/pshah/Downloads/P2 Ryan Edit.sv

logic [1:0] ew\_green\_timer; //timer for EW light in GREEN

logic reset\_s;

int i = 0;

int j = 0; //khensu/Home02/pshah/Downloads/P2 Ryan Edit.sv

//Creates synchronous reset

always\_comb

begin

reset\_s = reset\_n && clk;

end

//khensu/Home02/pshah/Downloads/P2 Ryan Edit.sv

//If not in reset position start NS green timer for 3 seconds following this time

//otherwise make NS light yellow and reset timer

always\_ff @(posedge clk)

begin

if(reset\_s == 0)

begin

if(i!=3)

begin

for(i = 0; i < 3; i++)

begin

ns\_green\_timer <= i;

end

end

else if (i == 3)

begin

i = 0;

ns\_green\_timer <= 0;

end

end

else if(reset\_s == 1 || ns\_light == YELLOW)

begin

ns\_green\_timer <= 0;

end

end

//If not in reset position start EW green timer for 3 seconds following this time

//otherwise make EW light yellow and reset timer

always\_ff @(posedge clk)

begin

if (reset\_s == 0)

begin

if (j != 3)

begin

for (j = 0; j < 3; j++)

begin

ew\_green\_timer <= j;

end

end

else if (j == 3)

begin

j = 0;

ew\_green\_timer <= j;

end

end

else if(reset\_s == 1 || ew\_light == YELLOW)

begin

ew\_green\_timer <= 0;

end

end

always\_ff @(posedge clk)

begin

//If both sensors for EW low then make EW light red and NS light green

if(ew\_sensor == 0 && emgcy\_sensor ==0)

begin

ns\_light = GREEN;

ew\_light = RED;

/\*ns\_light red <= 0;

ns\_light pre\_greem <= 0;

ns\_light yellow <= 0;

ns\_light off <= 0;

ns\_light green <= 1;

ew\_light red <= 1;

ew\_light pre\_greem <= 0;

ew\_light yellow <= 0;

ew\_light off <= 0;

ew\_light green <= 0; \*/

end

else if(ns\_light == GREEN && emgcy\_sensor==1)

begin

ns\_light = YELLOW;

ew\_light = RED;

/\*ns\_light red <= 0;

ns\_light pre\_greem <= 0;

ns\_light yellow <= 1

ns\_light off <= 0;

ns\_light green <=0;

ew\_light red <= 1;

ew\_light pre\_greem <= 0;

ew\_light yellow <= 0;

ew\_light off <= 0;

ew\_light green <= 0; \*/

end

else if(ew\_light == PRE\_GREEN)

begin

//for(ew\_green\_timer = ew\_green\_timer + 1)

begin

ew\_light <= GREEN

end

end

else if (ew\_light == YELLOW)

begin

//ew\_light yellow = 0;

ew\_light = RED;

end

end

3:

Sequence-Detect.sv

//Ryan Writz and Priyam Shah (c) July 2019

//ECE 4/581 Project 2 Question 3

//Systm Verilog module of a sequencer detector. This detector compares the input sequence

//with its own built-in sequence bit by bit. If the input sequence is identical to the

//built-in sequence, it will display a message "matched", otherwise it will display

//a message "not-matched". When the mismatched bit occurs, the detector will return to the

//initial state and process the next input bit as the beginning of a new sequence.

//Your built-in sequence is a 8-bit BCD code created by converting the last two

//digits of the PSU ID number of one member of your group. Implement the detector

//by an FSM-based model in Systemverilog.

module SequenceDetect(output logic [11:0] Match, //Match output to contain a string

input logic [7:0]Input,

input logic [3:0] Num1, Num2,

input logic clk,reset);

logic [7:0] BuiltIn; //Built in 8 digit BCD

logic [7:0] TestVec; //Test vector for checking bit by if

logic [7:0] AllOnes = 8'b1111\_1111;

[7:4]BuiltIn = [3:0] Num1;

[3:0]BuiltIn = [3:0] Num2;

int i;

always\_ff @(posedge clk, posedge reset)

begin

for(i = 0; i < 7; i++)

begin

if(BuiltIn[i] == Input[i]) //Fills out a vector by comparing each

TestVec[i] <= 1; //bit for a match between bulit in vs. given

else //Vector then later used to check matching case

TestVec[i] <= 0;

end

end

always\_ff @(posedge clk, posedge reset)

begin

for(i = 0; i < 7; i++)

begin

if(TestVec & AllOnes == 1)

Match = "matched";

else

Match = "not-matched";

end

end

endmodule

SeqTest.sv

module SeqTest();

logic [11:0] Match;

logic [7:0] Num1;

logic [7:0] Num2;

SequenceDetect SeqDet(Match,Input,Num1, Num2,clk,reset);

initial

begin

clk = 0;

forever #5 clk = ~clk;

end

initial

begin

#5 $display("%s",Match);

end

endmodule