Verilog Assignment 2

-M.Mathy Bala

1. Write Verilog code and relevant testbench that a) D- latch b) d-flipflop with synchronous and synchronous rest c) T-flipflop with synchronous and synchronous rest d) construct 4- bit counter using -T flipflop with reset and 4 bit shift register using synchronous D-flip flop.

1. D- latch

VERILOG

Module dlatch (d,en,rst);

Input(d,en,rst);

Output reg q;

always@(d,en,rst)

begin

if(!rst)

q<=0;

else

if(en)

q<=d;

end

endmodule

TESTBENCH

Module dlatchtb;

Reg d,en,rst;

Wire q;

Integer I;

Dlatch uut(.d(d),.en(en),.rst(rst),.q(q));

Initial

Begin

$dumpfile(“dump.vcd”);

$dumpvars(0,dlatchtb);

D<=0;

En<=0;

Rst<=0;

For(i=0;i<5;i++)

Begin  
#2 en<=~en;

#2 d<=I;

End

End

Endmodule

1. d-flipflop with synchronous and synchronous rest

VERILOG

Module dffrest(d,clk,rst,q);

Input d,clk,rst;

Output reg q;

always@(posedge clk)

if(!rst)

q<=0;

else

q<=d;

endmodule

TESTBENCH

Module dffresttb;

Reg d,clk,rst;

Wire q;

Dff uut (.d(d),.clk(clk),.rst(rst),.q(q));

Initial

Begin

$dumpfile(“dumpdffrest.vcd”);

$dumpvars(0,dffresttb);

Clk=0;

D=0;

Rest=1;

#5 rst = 0;

# 80 rst =1;

$monitor($time, “\tclk=%b\trst=%b\d=%b\tq=%b”,clk,rst,d,q);

#100 $finish;

End

Always #5 clk =~clk;

Always #30 d =~d;

endmodule

T-flipflop with synchronous and synchronous rest

VERILOG

Module tff (d,clk,rst,q);

Input d,clk,rst;

Output q;

always@(posedge clk)

if(~rst) begin

q<=1’b0;

end else if (d)

begin

q<=!q;

end

endmodule

TESTBENCH

Module dffresttb;

Reg d,clk,rst;

Wire q;

Dff uut (.d(d),.clk(clk),.rst(rst),.q(q));

Initial

Begin

$dumpfile(“dumpdffrest.vcd”);

$dumpvars(0,dffresttb);

Clk=0;

D=0;

Rest=1;

#5 rst = 0;

# 80 rst =1;

$monitor($time, “\tclk=%b\trst=%b\d=%b\tq=%b”,clk,rst,d,q);

#100 $finish;

End

Always #5 clk =~clk;

Always #30 d =~d;

Endmodule

2. Write Verilog code and relevant testbench that represents a) an eight-bit Johnson counter b) ring counter

c) one-hot counter and d) Gray code counter

1. an eight-bit Johnson counter

VERILOG

Module johnson(rst,clk,q);

Input rst,clk;

Output q;

Reg q;

always@(posegde clk or negedge rst)

if(!rst)

q<=0;

else

q<={{q[6:0]},~{q[7]}};

endmodule

TESTBENCH

Module johnsontb;

Reg rst,clk;

Wire q;

Johnson son(rst,clk,q);

Initial

Begin

Clk=0;

Rst=0;

$monitor($time, ,”c=%b”,clk, .,”r=%b”,rst,., “q=%b”,q);

#6 rst=1;

End

Always #2 clk=~clk;

Initial #68 $finish;

Endmodule

ring counter

VERILOG

Module ringcounter

{

input clk;

input rst;

output reg [width-1:0]out

};

Integer I;

Always@(posedge clk)

Begin

If(!rst)

Out<=1;

Else begin

Out[width-1]<=out[0];

For(i=0;i<width-1;i++)begin

Out[i]<=out[i+1];

End

End

End

Endmodule

TESTBENCH

Module ringcountertb;

Parameter width=4;

Reg clk,rst;

Wire [width-1:0]out;

Initial begin

$dumpfile(“dump.vcd”);

$dumpvars;

End

Ringcounter u0(.clk(clk),.rst(rst),.out(out));

Always #10 clk=~clk;

Initial begin{clk,rst}<=0;

$monitor ("T=%bt out=%b", $time, out);

repeat (2) @(posedge clk);

rstn <= 1;

repeat (15) @(posedge clk);

$finish;

end

endmodule

c)one-hot counter

VERILOG

Module onehot(out,en,clk,rst);

Input clk,rst,en;

Output out;

always@(posedgeclk)

if (rst)

begin

out<=8’b0000\_0001;

end

else if (en)

begin

out<={out[6],out[5],out[4],out[3],out[2],out[1],out[0],out[7]};

end

endmodule

d) Gray code counter

VERILOG

Module graycode(clk,rst,out);

Input rst;

Input clk;

Output reg [n-1:0]out;

Reg [n-1:0]q;

always @ (posedge clk) begin

if (!rstn) begin

q <= 0;

out <= 0;

end else begin

q <= q + 1;

`if def FOR\_LOOP

for (int i = 0; i < N-1; i= i+1) begin

out[i] <= q[i+1] ^ q[i];

end

out[N-1] <= q[N-1];

`else

out <= {q[N-1], q[N-1:1] ^ q[N-2:0]};

`endif

end

end

endmodule

TESTBENCH

Module graycodetb;

Parameter n=4;

reg clk;

reg rstn;

wire [N-1:0] out;

gray\_ctr u0 ( .clk(clk),

.rstn(rstn),

.out(out));

initial begin

$dumpfile("dump.vcd");

$dumpvars;

end

always #10 clk = ~clk;

initial begin

{clk,rst}<=0;

$monitor ("T=%bt,. rst=%b,. out=bout %b", $time, rst, out);

repeat(2) @ (posedge clk);

rstn <= 1;

repeat(20) @ (posedge clk);

$finish;

end

endmodule

Write Verilog code that represents a modulo-12 up-down counter with synchronous reset.

UP COUNTER

VERILOG

module mod12upcounter(out,rst,clk);

output [3:0]out;

input rst,clk;

reg [3:0]out;

always @(posedge clk)

begin

if(rst|out==4'b1011)

out<=4'b0000;

else

out<=out+1;

end

endmodule

TESTBENCH

module tb;

parameter N=12;

parameter WIDTH =4;

reg clk,rst;

wire [WIDTH -1:0]out;

mod12upcounter uut (.clk(clk),.rst(rst),.out(out));

always #10 clk=~clk;

initial begin

$dumpfile("dump.vcd");

$dumpvars(0,tb);

{clk,rst}<=0;

repeat(2)@(posedge clk);

rst<=1;

repeat(10)@(posedge clk);

$finish;

end

endmodule

DOWNCOUNTER

VERILOG

module down\_counter(input clk, reset, output [3:0] counter );

reg [3:0] counter\_down;

always @(posedge clk or posedge reset)

begin

if(reset)

counter\_down <= 4'hf;

else

counter\_down <= counter\_down - 4'd1;

end

assign counter = counter\_down;

endmodule

TESTBENCH

module downcounter\_testbench();

reg clk, reset;

wire [3:0] counter;

down\_counter dut(clk, reset, counter);

initial begin

clk=0;

forever #5 clk=~clk;

end

initial begin

$dumpfile("dump.vcd");

$dumpvars(0,downcounter\_testbench);

reset=1;

#20;

reset=0;

end

endmodule

4. Write the Verilog code for T flip-flops in all level of abstraction. Design a modulo 12 up/down counter using T flip-flops with synchronous reset. It should include a control input called Up*\_*Down. If Up*/*Down = 0, then the circuit should behave as an up-counter. If Up*/*Down = 1, then the circuit should behave as a down-counter.

VERILOG

Module modcounter(t,q,clk,rst);

Output [3:0]q;

Input clk,rst;

Input [3:0]t;

Tf tf0(q[0],qbar0,t0,clk,rst);

Tf2(q[1],qbar1,t1,qbar0,rst);

Tf3(q[2],qbar2,t2,qbar1,rst);

Tf4(q[3],qbar3,t3,qbar2,rst);

And and0(test,q[2],q[3]);

Or or0(rst,t,test);

Endmodule

5. A sequential circuit has two inputs, *w*1 and *w*2, and an output, *z*. Its function is to compare the input sequences on the two

inputs. If *w*1 = *w*2 during any four consecutive clock cycles, the circuit produces *z* = 1; otherwise, *z* = 0. For example

*w*1 : 0110111000110

*w*2 : 1110101000111

*z* : 0000100001110

Write Verilog code for the FSM described above using

a) Moore and

b) Mealy machine,

c) write the testbench, also

d) submit the simulated waveforms

a) Moore

module moore(clk,rst,din,dout);

input bit clk;

input logic rst,din;

output logic dout;

typedef enum logic [2:0] {S0, S1, S2, S3, S4} state\_t;

state\_t state;

always @(posedge clk or posedge reset) begin

if(reset) begin

dout <= 1'b0;

state <= S0;

end

else begin

case(state)

S0: begin

dout <=1'b0;

if(din)

state <= S1;

end

S1: begin

dout <= 1'b0;

if(din)

state <= S2;

else

state <= S0;

end

S2: begin

dout <= 1'b0;

if(din)

state <= S3;

end

S3: begin

dout <= 1'b0;

if(din)

state <= S4;

else

state <= S0;

end

S4: begin

dout <= 1'b1;

if(din)

state <= S1;

else

state <= S0;

end

endcase

end

end

endmodule

b)Meally Machine

module meally(clk,rst,din,dout);

input bit clk;

input logic rst,din;

output logic dout;

typedef enum logic [1:0] {S0, S1, S2, S3} state\_t;

state\_t state;

always @(posedge clk or posedge reset) begin

if(reset) begin

dout <= 1'b0;

state <= S0;

end

else begin

case(state)

S0: begin

if(din) begin

state <= S1;

dout <=1'b0;

end

else

dout <=1'b0;

end

S1: begin

if(din) begin

state <= S2;

dout <=1'b0;

end

else begin

state <= S0;

dout <=1'b0;

end

end

S2: begin

if(din) begin

state <= S3;

dout <=1'b0;

end

else begin

dout <=1'b0;

end

end

S3: begin

if(din) begin

state <= S3;

dout <=1'b1;

end

else begin

state <= S0;

dout <=1'b0;

end

end

endcase

end

end

endmodule

6. Derive a minimal state table for an FSM that acts as a three-bit parity generator. For every three bits that are observed on the

input *w* during three consecutive clock cycles, the FSM generates the parity bit *p* = 1 if and only if the number of 1s in the

three-bit sequence is odd. Write Verilog code using

a) Moore 2 always and 3 always block.

b) Mealy 2 always and 3 always block.

c) write the testbench, also

d) submit the simulated waveforms.

a)Moore 3 always block

module MOORE\_Verilog(sequence\_in,clock,reset,detector\_out);

input clock;

input reset;

input sequence\_in;

output reg detector\_out;

parameter Zero=3'b000,

One=3'b001,

OneZero=3'b011,

OneZeroOne=3'b010,

OneZeroOneOne=3'b110;

reg [2:0] current\_state, next\_state;

always @(posedge clock, posedge reset)

begin

if(reset==1)

current\_state <= Zero;

else

current\_state <= next\_state;

end

always @(current\_state,sequence\_in)

begin

case(current\_state)

Zero:begin

if(sequence\_in==1)

next\_state = One;

else

next\_state = Zero;

end

One:begin

if(sequence\_in==0)

next\_state = OneZero;

else

next\_state = One;

end

OneZero:begin

if(sequence\_in==0)

next\_state = Zero;

else

next\_state = OneZeroOne;

end

OneZeroOne:begin

if(sequence\_in==0)

next\_state = OneZero;

else

next\_state = OneZeroOneOne;

end

OneZeroOneOne:begin

if(sequence\_in==0)

next\_state = OneZero;

else

next\_state = One;

end

default:next\_state = Zero;

endcase

end

always @(current\_state)

begin

case(current\_state)

Zero: detector\_out = 0;

One: detector\_out = 0;

OneZero: detector\_out = 0;

OneZeroOne: detector\_out = 0;

OneZeroOneOne: detector\_out = 1;

default: detector\_out = 0;

endcase

end

endmodule

TESTBENCH

module tb\_Sequence\_Detector\_Moore\_FSM\_Verilog;

reg sequence\_in;

reg clock;

reg reset;

wire detector\_out;

Sequence\_Detector\_MOORE\_Verilog uut (

.sequence\_in(sequence\_in),

.clock(clock),

.reset(reset),

.detector\_out(detector\_out)

);

initial begin

clock = 0;

forever #5 clock = ~clock;

end

initial begin

sequence\_in = 0;

reset = 1;

#30;

reset = 0;

#40;

sequence\_in = 1;

#10;

sequence\_in = 0;

#10;

sequence\_in = 1;

#20;

sequence\_in = 0;

#20;

sequence\_in = 1;

#20;

sequence\_in = 0;

end

endmodule

b) Mealy 2 always

module seq\_detector(

input x,clk,reset,

output reg z

);

parameter S0 = 0 , S1 = 1 , S2 = 2 , S3 = 3 ;

reg [1:0] PS,NS ;

always@(posedge clk or posedge reset)

begin

if(reset)

PS <= S0;

else

PS <= NS ;

end

always@(PS or x)

begin

case(PS)

S0 : begin

z = 0 ;

NS = x ? S1 : S0 ;

$display(PS);

end

S1 : begin

z = 0 ;

NS = x ? S1 : S2 ;

$display(PS);

end

S2 : begin

z = 0 ;

NS = x ? S3 : S0 ;

$display(PS);

end

S3 : begin

z = x ? 1 : 0 ;

NS = x ? S1 : S2 ;

$display(PS);

end

endcase

end

endmodule

TESTBENCH

module testbench;

reg x, clk, reset;

wire z;

seq\_detector uut (

.x(x),

.clk(clk),

.reset(reset),

.z(z)

);

initial

begin

clk = 1'b0;

reset = 1'b1;

#15 reset = 1'b0;

end

always #5 clk = ~ clk;

initial begin

#12 x = 0;#10 x = 0 ; #10 x = 1 ; #10 x = 0 ;

#12 x = 1;#10 x = 1 ; #10 x = 0 ; #10 x = 1 ;

#12 x = 1;#10 x = 0 ; #10 x = 0 ; #10 x = 1 ;

#12 x = 0;#10 x = 1 ; #10 x = 1 ; #10 x = 0 ;

#10 $finish;

end

endmodule

7. Write verilog to convert a) an 8-bit Serial-in, Parallel-out (SIPO) Conversion and 8-bit Parallel-in, Serial-out (PISO) Conversion. Write the testbench and submit waveform.

a) an 8-bit Serial-in, Parallel-out (SIPO) Conversion

module ShiftRegister\_SIPO(C, SI, PO);

input C,SI;

output [7:0] PO;

reg [7:0] tmp;

always @(posedge C)

begin

tmp = {tmp[6:0], SI};

end

assign PO = tmp;

endmodule

b)8-bit Parallel-in, Serial-out (PISO) Conversion.

module Shiftregister\_PISO(Clk, Parallel\_In,load, Serial\_Out);

input Clk,load;

input [7:0]Parallel\_In;

output reg Serial\_Out;

reg [7:0]tmp;

always @(posedge Clk)

begin

if(load)

tmp<=Parallel\_In;

else

begin

Serial\_Out<=tmp[3];

tmp<={tmp[6:0],1'b0};

end

end

endmodule

11. Design a simple RAM with a data width of 32 bits and a depth of 1024, and write a test bench to verify the module. The testbench must have the read and write tasks to perform the following, a) write down all the locations with random numbers. Read the content of the memory and store all odd numbers in odd.txt file with location and even numbers in a text file with Memory location.

module single\_port\_sync\_ram #(parameter ADDR\_WIDTH =4, parameter DATA\_WIDTH =32,parameter DEPTH=16)

(input clk,

input [ADDR\_WIDTH-1:0] addr,

input [DATA\_WIDTH-1:0] DATA,

input cs,

input we,

input oe);

reg [DATA\_WIDTH-1:0] tmp\_data;

reg[DATA\_WIDTH-1:0] mem [DEPTH];

always @(posedge clk)

begin

if (cs&we)

mem[addr]<=data;

end

always @(posedge clk)

begin

if(cs&!we)

tmp\_data<=mem[addr];

end

assign data=cs & oe & ! we ? tmp\_data: 'hz;

endmodule

12. Write Verilog RTL for generic synchronous FIFO

VERILOG

module FIFObuffer( Clk, dataIn, RD, WR, EN, dataOut, Rst,EMPTY, FULL );

input Clk, RD, WR, EN, Rst;

output EMPTY, FULL;

input [31:0] dataIn;

output reg [31:0] dataOut; // internal registers

reg [2:0] Count = 0;

reg [31:0] FIFO [0:7];

reg [2:0] readCounter = 0,

writeCounter = 0;

assign EMPTY = (Count==0)? 1'b1:1'b0;

assign FULL = (Count==8)? 1'b1:1'b0;

always @ (posedge Clk)

begin

if (EN==0);

else begin

if (Rst) begin

readCounter = 0;

writeCounter = 0;

end

else if (RD ==1'b1 && Count!=0) begin

dataOut = FIFO[readCounter];

readCounter = readCounter+1;

end

else if (WR==1'b1 && Count<8) begin

FIFO[writeCounter] = dataIn;

writeCounter = writeCounter+1;

end

else;

end

if (writeCounter==8)

writeCounter=0;

else if (readCounter==8)

readCounter=0;

else;

if (readCounter > writeCounter) begin

Count=readCounter-writeCounter;

end

else if (writeCounter > readCounter)

Count=writeCounter-readCounter;

else;end;

endmodule

TESTBENCH

module FIFObuffer\_tb;

reg Clk;

reg [31:0] dataIn;

reg RD,WR,EN,Rst;

wire [31:0] dataOut,EMPTY,FULL;

// Instantiate the Unit Under Test (UUT)

FIFObuffer uut (.Clk(Clk), .dataIn(dataIn),.RD(RD), .WR(WR),.EN(EN),

.dataOut(dataOut),

.Rst(Rst),

.EMPTY(EMPTY),.FULL(FULL));

initial begin

$dumpfile("dump.vcd");

$dumpvars(0,FIFObuffer\_tb);

// Initialize Inputs

Clk = 1'b0;

dataIn = 32'h0; RD = 1'b0;

WR = 1'b0;

EN = 1'b0;

Rst = 1'b1;

// Wait 100 ns for global reset to finish

#100;

EN = 1'b1;

Rst = 1'b1;

#20;

Rst = 1'b0;

WR = 1'b1;

dataIn = 32'h0;

#20;

dataIn = 32'h1;

#20;

dataIn = 32'h2;

#20;

dataIn = 32'h3;

#20;

dataIn = 32'h4;

#20;

WR = 1'b0;

RD = 1'b1;

end

always #10 Clk = ~Clk;

endmodule