**Course Code: EE461**

**Bonus Assignment**

**PREPARED BY**

**Khandoker Samiul Hoque**

**Student Id: 19837**

**Github Url:** <https://github.com/KhandokerSamiulHoque/EE461assignment6-bonus.git>

**1 No Answer:**

**Design:**

`define kBitW 6

`define kBitS `kBitW-1:0

module upDownLFSR (

input clk, rst,en\_i,upDown\_i,

output [`kBitS] cnt\_o,bitStr\_o

);

reg [`kBitS] cnt\_o;

assign bitStr\_o = (!en\_i) ? 1'b0 :

(upDown\_i) ? cnt\_o[0] : cnt\_o[5];

always @(posedge clk) begin

if (rst) begin

cnt\_o <= 6'b000000;

end

else if (en\_i) begin

if (upDown\_i) begin

cnt\_o[0] <= cnt\_o[1];

cnt\_o[1] <= cnt\_o[2];

cnt\_o[2] <= cnt\_o[3];

cnt\_o[3] <= cnt\_o[4];

cnt\_o[4] <= cnt\_o[5];

cnt\_o[5] <= cnt\_o[0] ~^ cnt\_o[3];

end

else

begin

cnt\_o[0] <= cnt\_o[5] ~^ cnt\_o[4]~^ cnt\_o[3] ~^ cnt\_o[2];

cnt\_o[1] <= cnt\_o[0];

cnt\_o[2] <= cnt\_o[1];

cnt\_o[3] <= cnt\_o[2];

cnt\_o[4] <= cnt\_o[3];

cnt\_o[5] <= cnt\_o[4];

end

end

else

begin

cnt\_o <= 6'b000000;

end

end

endmodule

**Testbench:**

module upDownLFSR\_tb;

parameter kBitW = 6;

reg clk = 0;

reg rst = 1;

reg en\_i = 0;

reg upDown\_i = 0;

wire [`kBitW-1:0] cnt\_o;wire bitStr\_o;

upDownLFSR ee461prob1 (.clk(clk),.rst(rst),.en\_i(en\_i),.upDown\_i(upDown\_i),.cnt\_o(cnt\_o),.bitStr\_o(bitStr\_o));

initial begin

$dumpfile("upDownLFSR\_tb.vcd");

$dumpvars(0, upDownLFSR\_tb);

#5;

// Test case 1

rst = 1;

en\_i = 1;

upDown\_i = 1;

#10;

rst = 0;

#50;

rst = 1;

#10;

// Test case 2

rst = 1;

en\_i = 1;

upDown\_i = 0;

#10;

rst = 0;

#50;

rst = 1;

#10;

// Test case 3

rst = 1;

en\_i = 1;

upDown\_i = 1;

#10;

rst = 0;

#100;

en\_i = 0;

// Test case 4

rst = 1;

en\_i = 1;

upDown\_i = 0;

#10;

rst = 0;

#100;

en\_i = 0;

#10

$finish;

end

always

#5

clk = ~clk;

always @(posedge clk) begin

$display("[%t] cnt\_o = %b, bitStr\_o = %b", $time, cnt\_o, bitStr\_o);

end

endmodule

**2 No Answer:**

**Design:**

module Dividerby7(clk,rst,clkOut\_o);

input clk;

input rst;

output clkOut\_o;

reg[2:0] rEdgeCnt\_r;

reg[2:0] fEdgeCnt\_r;

assign clkOut\_o = (rEdgeCnt\_r < 4 && fEdgeCnt\_r < 3)?1'b1: 1'b0;

always@(posedge clk)

begin

if(rst) rEdgeCnt\_r <= 0;

else if(rEdgeCnt\_r == 6) rEdgeCnt\_r <= 0;

else rEdgeCnt\_r <= rEdgeCnt\_r + 1;

end

always@(negedge clk)

begin

if(rst) fEdgeCnt\_r <= 0;

else if(fEdgeCnt\_r == 6) fEdgeCnt\_r <= 0;

else fEdgeCnt\_r <= fEdgeCnt\_r + 1;

end

endmodule

**Testbench:**

module Dividerby7\_tb;

reg clk, rst;

wire clkOut\_out;

Dividerby7 ee461prob2(.clk(clk),.rst(rst),.clkOut\_o(clkOut\_out));

initial begin

$dumpfile("Dividerby7\_tb.vcd");

$dumpvars(0,Dividerby7\_tb);

clk = 0;

rst = 1;

#10

rst = 0;

end

always

#5

clk = ~clk;

initial begin

#100

$finish;

end

always @(posedge clk) begin

$display("Time %0d: clk=%b rst=%b clkOut\_out=%b", $time, clk, rst,clkOut\_out);

end

endmodule

**3 No Answer:**

**Design:**

module crc4\_decoder(input clk, rst,en\_i,

input [3:0]dIn\_i,output [3:0] dOut\_o);

reg [3:0] lfsr\_r;

assign dOut\_o = lfsr\_r;

always @(posedge clk) begin

if (rst) lfsr\_r <= 3'b000;

else if (en\_i)

begin

lfsr\_r[2] <= lfsr\_r[1];

lfsr\_r[1] <= lfsr\_r[0] ^ lfsr\_r[2] ^ dIn\_i[3];

lfsr\_r[0] <= lfsr\_r[3] ^ dIn\_i[2];

lfsr\_r[3] <= dIn\_i[1];

end

end

endmodule

**Testbench:**

module crc4\_decoder\_tb();

reg reset, clock, en\_reg;reg [4:0] dIn;wire [3:0] out;

crc4\_decoder ee461prob3(.clk(clock),.rst(reset),.en\_i(en\_reg),.dIn\_i(dIn[4:1]),.dOut\_o(out));

initial begin

$dumpfile("crc4\_decoder\_tb.vcd");

$dumpvars(0,crc4\_decoder\_tb);

end

always

#5

clock = ~clock;

initial begin

reset = 1'b1;

en\_reg = 1'b0;

clock = 1'b0;

dIn = 5'b10010;

#10

reset = 1'b0;

#10

en\_reg = 1'b1;

#10

dIn[0] = 1'b0;

#10

dIn[0] = 1'b1;

#10

dIn[0] = 1'b0;

#10

dIn[0] = 1'b0;

#10

dIn[0] = 1'b1;

#10

dIn[0] = 1'b1;

#10

dIn[0] = 1'b1;

#10

dIn[0] = 1'b0;

#10

$finish;

end

endmodule