21BDS0340

Abhinav Dinesh Srivatsa

Digital Systems Design Lab

Assignment – V

1. Question 1

Aim:
To implement a universal shift register model

Truth Table:

Case	Operation
0	Retain register information
1	Shift register information right
2	Shift register information left
3	Overwrite register information

Components Required:
4:1 multiplexer
2 registers (input and output)

Verilog Program:

```
module shift_reg (
    input [3:0]i,
    output reg [3:0]o,
    input [1:0]s,
    input MSB, LSB, clk, clear
);

always @(posedge clk or negedge clear)
    if (clear == 0)
        o <= 4'b0000;
    else
        case (s)
            2'b00: o <= o;
            2'b01: o <= {MSB, o[3:1]};
            2'b10: o <= {o[2:0], LSB};
            2'b11: o <= i;
        endcase

endmodule
```

Testbench:

```
module test_reg;

reg [3:0]i;
reg [1:0]s;
wire [3:0]f;
reg clk, clear, msb, lsb;

shift_reg sr1(i, f, s, msb, lsb, clk, clear);

always
    #5
    clk = ~clk;

initial begin
    i = 4'b0110;
    msb = 1; lsb = 1;
    s = 2'b00; #10
    s = 2'b01; #10
    s = 2'b10; #10
    s = 2'b11; #10
    s = 2'b11;
end

initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
end

endmodule
```

Result:
The universal shift register has been implemented in Verilog code

2. Question 2

Aim:
To implement a universal shift register structural model

Truth Table:

Case	Operation
0	Retain register information
1	Shift register information right
2	Shift register information left
3	Overwrite register information

Components Required:
4:1 multiplexer
2 registers (input and output)
4 D flipflops

Verilog Program:

```
module shift_reg_struct (
    input [3:0]i,
    output reg [3:0]o,
    input [1:0]s,
    input MSB, LSB, clk, clear
);

stage s0(o[0], o[1], LSB, i[0], s, clk, clear);
stage s1(o[1], o[2], o[0], i[1], s, clk, clear);
stage s2(o[2], o[3], o[1], i[2], s, clk, clear);
stage s3(o[3], MSB, o[2], i[3], s, clk, clear);

endmodule

module stage (
    input [0:3]i,
    input [0:1]s,
    output q,
    input clk, clear
);

wire mux_out;
mux4to1 m(i, s, mux_out);
dflipflop d(mux_out, q, clk, clear);

endmodule

module mux4to1 (
    input [0:3]i,
    input [0:1]s,
    output reg o,
    input clk,
);

always @(s)
    case(s)
        2'b00: o = i[0];
        2'b01: o = i[1];
        2'b10: o = i[2];
        2'b11: o = i[3];
    endcase

endmodule

module dflipflop (
```

```
    input d,
    output reg q,
    input clk, clear
);

always @(posedge clk or negedge clear)
    if(clear == 0)
        q <= 1'b0;
    else
        q <= d;

endmodule
```

Testbench:

```
module test_reg;

reg [3:0]i;
reg [1:0]s;
wire [3:0]f;
reg clk, clear, msb, lsb;

shift_reg_struct sr1(i, f, s, msb, lsb, clk, clear);

always
    #5
    clk = ~clk;

initial begin
    i = 4'b0110;
    msb = 1; lsb = 1;
    s = 2'b00; #10
    s = 2'b01; #10
    s = 2'b10; #10
    s = 2'b11; #10
    s = 2'b11;
end

initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1);
end

endmodule
```

Result:

The universal shift register has been implemented in Verilog code

3. Question 3

Aim:
To implement a Johnson ring counter model

Truth Table:

Clock Cycle	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Components Required:
1 register

Verilog Program:

```
module ring_counter (
    output reg [3:0]o,
    input clk, clear
);

always @(posedge clk or posedge clear)
    if(clear == 1)
        o <= 4'b0000;
    else
        o <= {~o[0], o[3:1]};

endmodule
```

Testbench:

```
module test_ring;

wire [3:0]counter;
reg clk, clear;

ring_counter rc(counter, clk, clear)

always
    #5
    clk = ~clk;

initial begin
    clear = 1; clk = 0;
    #10
    clear = 0;
    counter = 4'b0000;
```

```
end

endmodule
```

Result:
The Johnson ring counter has been implemented in Verilog code

4. Question 4

Aim:
To implement a 4-bit up/down counter model

Truth Table:

	Current	Up	Down
0	0000	0001	1111
1	0001	0010	0000
2	0010	0011	0001
3	0011	0100	0010
4	0100	0101	0011
5	0101	0110	0100
6	0110	0111	0101
7	0111	1000	0110
8	1000	1001	0111
9	1001	1010	1000
10	1010	1011	1001
11	1011	1100	1010
12	1100	1101	1011
13	1101	1110	1100
14	1110	1111	1101
15	1111	0000	1110

Components Required:
1 register

Verilog Program:

```
module counter(
    input [3:0]i,
    output reg [3:0]o,
    input load, up, down,
    input clk, reset
);

always @(posedge clk)
    if (reset == 1)
        o <= 4'0000;
    if (load == 1)
        o <= i;
    if (up == 1)
        o <= o + 1'b1;
    if (down == 1)
```

```
        o <= o - 1'b1;

endmodule

module counter_mux(
    input [3:0]i,
    output reg [3:0]o,
    input [0:1]s,
    input clk, reset
);

always @(posedge clk)
    if (reset == 1)
        o <= 4'0000;
    case (s)
        2'b00: o <= i;
        2'b01: o <= o + 1'b1;
        2'b10: o <= o - 1'b1;
        2'b11: o <= o;

endmodule
```

Testbench:

```
module test_updowncounter;

reg [3:0]in;
wire [3:0]out;
reg load, up, down, clk, reset;

counter rc(in, out, load, up, down, clk, reset);

always
    #5
    count = ~count;

initial begin
    reset = 1; clk = 0; up = 1;
    #20
    down = 1;
    #20
    in = 4'b1000; load = 1;
    #10
    up = 1;
end

endmodule
```

Result:

The up and down counter has been implemented in Verilog code

5. Question 5

Aim:
To implement a ripple counter model

Truth Table:

Clock State	Q0	Q1	Q2	Q3
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Components Required:
1 register

Verilog Program:

```
module ripple_counter (
    output [0:3]o,
    input count, reset
);

comp_dflipflop d1(o[0], count, reset);
comp_dflipflop d2(o[1], o[0], reset);
comp_dflipflop d2(o[2], o[1], reset);
comp_dflipflop d2(o[3], o[2], reset);

endmodule

module comp_dflipflop (
    input d,
    output reg q,
    input clk, reset
);

always @(negedge clk or posedge reset)
    if(clear == 0)
        q <= 1'b0;
    else
        q <= #2 ~q;

endmodule
```

Techbench:

```
module test_ripple_counter;
```

```
reg count;
reg reset;
wire [0:3]out;
ripple_counter rc(out, count, reset);
always
    #5
    count = ~count;
initial begin
    count = 1'b0;
    reset = 1'b1;
    #4
    reset = 1'b0;
end

endmodule
```

Result:
The ripple counter has been implemented in Verilog