



## San Francisco Bay University

### EE461 Verilog-HDL Homework #6

Due day: 12/4/2024

#### Instruction:

- a. Push answer sheets/source code to Github
- b. Please follow the code style rule like programs on handout.
- c. Overdue homework submission could not be accepted.
- d. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

1. Write LFSR up & down count module to create 6-bits pseudo random number in Verilog module

```
`define kBitW 6
`define kBitS `kBitW-1:0

module upDownLFSR(
    clk,
    rst,
    en_i,
    upDown_i,    //upDown_i = 1 -> up; othrwise down
    cnt_o,
    bitStr_o     //Overflow bit
);
    ... ...;
Endmodule
```

#### MODULE

```
`timescale 1ns/1ps
```

```
module tb_upDownLFSR;
```

```
    // Inputs
    reg clk;
    reg rst;
    reg en_i;
    reg upDown_i;
```

```
    // Outputs
    wire [5:0] cnt_o;
    wire bitStr_o;
```

```
    // Instantiate the LFSR module
    upDownLFSR uut (
        .clk(clk),
```

```
.rst(rst),
.en_i(en_i),
.upDown_i(upDown_i),
.cnt_o(cnt_o),
.bitStr_o(bitStr_o)
);

// Clock generation
initial clk = 0;
always #5 clk = ~clk; // Adjusted to 10ns clock period

// Test sequence
initial begin
    // Initialize inputs
    rst = 1; en_i = 0; upDown_i = 1;
    #10 rst = 0; // Deassert reset

    // Enable counting and count up
    en_i = 1; upDown_i = 1;
    #50;

    // Change direction to count down
    upDown_i = 0;
    #50;

    // Disable counting
    en_i = 0;
    #50;

    $finish; // End simulation cleanly
end

// Monitor signals
initial begin
    $monitor("Time: %0d, Count: %b, Overflow: %b", $time, cnt_o, bitStr_o);
end

endmodule

TESTBENCH
`timescale 1ns/1ps

module tb_upDownLFSR;

    // Inputs
    reg clk;
    reg rst;
    reg en_i;
    reg upDown_i;

    // Outputs
    wire [5:0] cnt_o;
    wire bitStr_o;
```

```
// Instantiate the LFSR module
upDownLFSR uut (
    .clk(clk),
    .rst(rst),
    .en_i(en_i),
    .upDown_i(upDown_i),
    .cnt_o(cnt_o),
    .bitStr_o(bitStr_o)
);

// Clock generation
initial clk = 0;
always #5 clk = ~clk; // Adjusted to 10ns clock period

// Test sequence
initial begin
    // Initialize inputs
    rst = 1; en_i = 0; upDown_i = 1;
    #10 rst = 0; // Deassert reset

    // Enable counting and count up
    en_i = 1; upDown_i = 1;
    #50;

    // Change direction to count down
    upDown_i = 0;
    #50;

    // Disable counting
    en_i = 0;
    #50;

    $finish; // End simulation cleanly
end

// Monitor signals
initial begin
    $monitor("Time: %0d, Count: %b, Overflow: %b", $time, cnt_o, bitStr_o);
end

endmodule
```

## SIMULATION OUTPUT

```

3 module tb_upDownLFSR;
4
5     // Inputs
6     reg clk;
7     reg rst;
8     reg en_i;
9     reg upDown_i;
10
11     // Outputs
12     wire [5:0] cnt_o;
13     wire bitStr_o;
14
15     // Instantiate the LFSR module
16     upDownLFSR uut (
17         .clk(clk),
18         .rst(rst),
19         .en_i(en_i),
20         .upDown_i(upDown_i),
21         .cnt_o(cnt_o),
22         .bitStr_o(bitStr_o)
23     );
24
25     // Clock generation
26     initial clk = 0;
27     always #5 clk = ~clk; // Adjusted to 10ns clock period
28
29     // Test sequence
30     initial begin
31         // Initialize inputs
32         rst = 1; en_i = 0; upDown_i = 1;
33         #10 rst = 0; // Deassert reset
34
35         // Enable counting and count up
36         en_i = 1; upDown_i = 1;
37         #50;
38
39         // Change direction to count down
40         upDown_i = 0;
41         #50;
42
43         // Disable counting
44         en_i = 0;
45         #50;
46
47         $finish; // End simulation cleanly
48     end
49
50     // Monitor signals
51     initial begin
52         $monitor("Time: %d, Count: %b, Overflow: %b",
53             $time, cnt_o, bitStr_o);
54     end
55 endmodule

```

```

3 `define kBitw 6
4 `define kBitS `kBitw-1:0
5
6 module upDownLFSR (
7     input clk,
8     input rst,
9     input en_i, // Enable signal
10    input upDown_i, // Direction: 1 for up, 0 for down
11    output reg [`kBitS] cnt_o, // Current count value
12    output reg bitStr_o // Overflow bit
13);
14
15 reg [`kBitS] lfsr; // LFSR register
16 wire feedback;
17
18 assign feedback = lfsr[5] ^ lfsr[3]; // Feedback taps for pseudo-randomness
19
20 always @(posedge clk or posedge rst) begin
21     if (rst) begin
22         lfsr <= 6'b1; // Initialize LFSR to non-zero value
23         cnt_o <= 6'b0;
24         bitStr_o <= 1'b0;
25     end else if (en_i) begin
26         if (upDown_i) begin
27             // Count up
28             lfsr <= {lfsr[4:0], feedback};
29         end else begin
30             // Count down
31             lfsr <= {feedback, lfsr[5:1]};
32         end
33         cnt_o <= lfsr; // Output the current LFSR value
34         bitStr_o <= lfsr[5]; // Overflow bit is the MSB
35     end
36 end
37 endmodule
38

```

[2024-12-16 12:16:24 UTC] iverilog '-wall' '-g2012' design.vv testbench.vv && unbuffer vvp a.out  
 Time: 0, Count: 000000, Overflow: 0  
 Time: 15, Count: 000001, Overflow: 0  
 Time: 25, Count: 000010, Overflow: 0  
 Time: 35, Count: 000100, Overflow: 0  
 Time: 45, Count: 001000, Overflow: 0  
 Time: 55, Count: 010001, Overflow: 0  
 Time: 65, Count: 100010, Overflow: 1  
 Time: 75, Count: 110001, Overflow: 1  
 Time: 85, Count: 111000, Overflow: 1  
 Time: 95, Count: 011100, Overflow: 0  
 Time: 105, Count: 101110, Overflow: 1  
 testbench.vv:47: \$finish called at 160000 (tps)  
 Done

## 2. Design frequency divider by 7 in Verilog module

```
MODULE
module frequencyDividerBy7(
    input clk,    // Input clock
    input rst,    // Reset signal
    output reg clk_out // Divided clock output
);

    reg [2:0] counter; // 3-bit counter to count up to 7

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            counter <= 3'b0;
            clk_out <= 0;
        end else begin
            if (counter == 3'b110) begin // Counter reaches 6 (7 cycles)
                counter <= 3'b0;    // Reset counter
                clk_out <= ~clk_out; // Toggle output clock
            end else begin
                counter <= counter + 1; // Increment counter
            end
        end
    end
endmodule

TESTBENCH
`timescale 1ns/1ps

module tb_frequencyDividerBy7;

    // Inputs
    reg clk;
    reg rst;

    // Outputs
    wire clk_out;

    // Instantiate the Frequency Divider module
    frequencyDividerBy7 uut (
        .clk(clk),
        .rst(rst),
        .clk_out(clk_out)
    );
endmodule
```

Anika Haque, 20283

```
// Clock generation
initial clk = 0;
always #5 clk = ~clk; // 10ns clock period for the input clock

// Test sequence
initial begin
    // Initialize inputs
    rst = 1; // Apply reset
    #10 rst = 0; // Release reset after 10ns

    // Observe the output for multiple cycles
    #500; // Allow simulation to run for 500ns

    $finish; // End simulation
end

// Monitor signals
initial begin
    $monitor("Time: %0d, Input Clock: %b, Output Clock: %b", $time, clk, clk_out);
end

endmodule
```

SCEENSHOT

```

1 `timescale 1ns/1ps
2
3 module tb_frequencyDividerBy7;
4
5 // Inputs
6 reg clk;
7 reg rst;
8
9 // Outputs
10 wire clk_out;
11
12 // Instantiate the Frequency Divider module
13 frequencyDividerBy7 uut (
14     .clk(clk),
15     .rst(rst),
16     .clk_out(clk_out)
17 );
18
19 // Clock generation
20 initial clk = 0;
21 always #5 clk = ~clk; // 10ns clock period for the

```

```

3 input rst; // Reset signal
4 output reg clk_out; // Divided clock output
5
6
7 reg [2:0] counter; // 3-bit counter to count up to 7
8
9 always @(posedge clk or posedge rst) begin
10     if (rst) begin
11         counter <= 3'b0;
12         clk_out <= 0;
13     end else begin
14         if (counter == 3'b110) begin // Counter reaches 6 (7 cycles)
15             counter <= 3'b0; // Reset counter
16             clk_out <= ~clk_out; // Toggle output clock
17         end else begin
18             counter <= counter + 1; // Increment counter
19         end
20     end
21 end
22 endmodule
23

```

```

[2024-12-16 13:15:45 UTC] iverilog -Wall -g2012 design.v testbench.v && unbuffer vvp a.out
warning: Some design elements have no explicit time unit and/or
: time precision. This may cause confusing timing results.
: Affected design elements are:
:   -- module frequencyDividerBy7 declared here: design.v:1
Time: 0, Input Clock: 0, Output Clock: 0
Time: 5, Input Clock: 1, Output Clock: 0
Time: 10, Input Clock: 0, Output Clock: 0
Time: 15, Input Clock: 1, Output Clock: 0
Time: 20, Input Clock: 0, Output Clock: 0
Time: 25, Input Clock: 1, Output Clock: 0
Time: 30, Input Clock: 0, Output Clock: 0
Time: 35, Input Clock: 1, Output Clock: 0
Time: 40, Input Clock: 0, Output Clock: 0
Time: 45, Input Clock: 1, Output Clock: 0
Time: 50, Input Clock: 0, Output Clock: 0
Time: 55, Input Clock: 1, Output Clock: 0
Time: 60, Input Clock: 0, Output Clock: 0
Time: 65, Input Clock: 1, Output Clock: 0
Time: 70, Input Clock: 0, Output Clock: 0
Time: 75, Input Clock: 1, Output Clock: 1
Time: 80, Input Clock: 0, Output Clock: 1
Time: 85, Input Clock: 1, Output Clock: 1
Time: 90, Input Clock: 0, Output Clock: 1
Time: 95, Input Clock: 1, Output Clock: 1
Time: 100, Input Clock: 0, Output Clock: 1
Time: 105, Input Clock: 1, Output Clock: 1
Time: 110, Input Clock: 0, Output Clock: 1
Time: 115, Input Clock: 1, Output Clock: 1
Time: 120, Input Clock: 0, Output Clock: 1
Time: 125, Input Clock: 1, Output Clock: 1
Time: 130, Input Clock: 0, Output Clock: 1
Time: 135, Input Clock: 1, Output Clock: 1
Time: 140, Input Clock: 0, Output Clock: 1
Time: 145, Input Clock: 1, Output Clock: 0
Time: 150, Input Clock: 0, Output Clock: 0
Time: 155, Input Clock: 1, Output Clock: 0
Time: 160, Input Clock: 0, Output Clock: 0
Time: 165, Input Clock: 1, Output Clock: 0

```

(Output too big to put in one screenshot)

## SCREENSHOT - END OF OUTPUT

The screenshot displays a Verilog simulation environment with two main panels: 'testbench.sv' on the left and 'design.sv' on the right. Below these panels is a 'Log' window showing the simulation results.

**testbench.sv:**

```

1 `timescale 1ns/1ps
2
3 module tb_frequencyDividerBy7;
4
5     // Inputs
6     reg clk;
7     reg rst;
8
9     // Outputs
10    wire clk_out;
11
12    // Instantiate the Frequency Divider module
13    frequencyDividerBy7 uut (
14        .clk(clk),
15        .rst(rst),
16        .clk_out(clk_out)
17    );
18
19    // Clock generation
20    initial clk = 0;
21    always #5 clk = ~clk; // 10ns clock period for the
22    input clock;
23
24    // Test sequence
25    initial begin
26        // Initialize inputs
27        rst = 1; // Apply reset
28        #10 rst = 0; // Release reset after 10ns
29    end
30 endmodule

```

**design.sv:**

```

3 input rst; // Reset signal
4 output reg clk_out; // Divided clock output
5
6
7 reg [2:0] counter; // 3-bit counter to count up to 7
8
9 always @(posedge clk or posedge rst) begin
10     if (rst) begin
11         counter <= 3'b0;
12         clk_out <= 0;
13     end else begin
14         if (counter == 3'b110) begin // Counter reaches 6 (7 cycles)
15             counter <= 3'b0; // Reset counter
16             clk_out <= ~clk_out; // Toggle output clock
17         end else begin
18             counter <= counter + 1; // Increment counter
19         end
20     end
21 end
22 endmodule
23

```

**Log:**

```

@Log
Time: 350, Input Clock: 0, Output Clock: 0
Time: 355, Input Clock: 1, Output Clock: 1
Time: 360, Input Clock: 0, Output Clock: 1
Time: 365, Input Clock: 1, Output Clock: 1
Time: 370, Input Clock: 0, Output Clock: 1
Time: 375, Input Clock: 1, Output Clock: 1
Time: 380, Input Clock: 0, Output Clock: 1
Time: 385, Input Clock: 1, Output Clock: 1
Time: 390, Input Clock: 0, Output Clock: 1
Time: 395, Input Clock: 1, Output Clock: 1
Time: 400, Input Clock: 0, Output Clock: 1
Time: 405, Input Clock: 1, Output Clock: 1
Time: 410, Input Clock: 0, Output Clock: 1
Time: 415, Input Clock: 1, Output Clock: 1
Time: 420, Input Clock: 0, Output Clock: 1
Time: 425, Input Clock: 1, Output Clock: 0
Time: 430, Input Clock: 0, Output Clock: 0
Time: 435, Input Clock: 1, Output Clock: 0
Time: 440, Input Clock: 0, Output Clock: 0
Time: 445, Input Clock: 1, Output Clock: 0
Time: 450, Input Clock: 0, Output Clock: 0
Time: 455, Input Clock: 1, Output Clock: 0
Time: 460, Input Clock: 0, Output Clock: 0
Time: 465, Input Clock: 1, Output Clock: 0
Time: 470, Input Clock: 0, Output Clock: 0
Time: 475, Input Clock: 1, Output Clock: 0
Time: 480, Input Clock: 0, Output Clock: 0
Time: 485, Input Clock: 1, Output Clock: 0
Time: 490, Input Clock: 0, Output Clock: 0
Time: 495, Input Clock: 1, Output Clock: 1
Time: 500, Input Clock: 0, Output Clock: 1
Time: 505, Input Clock: 1, Output Clock: 1
testbench.sv:32: $finish called at $10000 (1ps)
Time: 510, Input Clock: 0, Output Clock: 1

```



3. Design CRC-4 decoder in Verilog module showing the hardware circuit

MODULE

```
`timescale 1ns/1ps
```

```
module CRC4Decoder(
```

```
    input clk,
```

```
    input rst,
```

```
    input [7:0] data_in,
```

```
    output reg [3:0] crc_out
```

```
);
```

```
    reg [7:0] shift_reg;      // Shift register to process data
```

```
    reg [4:0] polynomial = 5'b10011; // CRC-4 polynomial (5 bits)
```

```
    integer i;
```

```
    reg processing;          // Flag to indicate computation is in progress
```

```
    always @(posedge clk or posedge rst) begin
```

```
        if (rst) begin
```

```
            shift_reg <= 8'b0;    // Reset the shift register
```

```
            crc_out <= 4'b0;    // Reset CRC output
```

```
            processing <= 0;    // Ensure processing flag is reset
```

```
        end else if (!processing) begin
```

```
            shift_reg = data_in; // Load input data into the shift register
```

```
            processing = 1;    // Set processing flag
```

```
            // Perform polynomial division for 8 bits
```

```
            for (i = 0; i < 8; i = i + 1) begin
```

```
                if (shift_reg[7]) begin
```

```
                    // XOR the polynomial when MSB is 1
```

```
                    shift_reg = (shift_reg << 1) ^ polynomial;
```

```
                end else begin
```

```
                    // Otherwise, just shift left
```

```
                    shift_reg = shift_reg << 1;
```

```
                end
```

```
            end
```

```
            crc_out = shift_reg[7:4]; // Extract the top 4 bits as CRC
```

```
            processing = 0;    // Reset processing flag when done
```

```
        end
```

```
    end
```

```
endmodule
```

TESTBENCH

```
`timescale 1ns/1ps
```

```
module tb_CRC4Decoder;
```

```
// Inputs
reg clk;
reg rst;
reg [7:0] data_in;

// Outputs
wire [3:0] crc_out;

// Instantiate the CRC4Decoder module
CRC4Decoder uut (
    .clk(clk),
    .rst(rst),
    .data_in(data_in),
    .crc_out(crc_out)
);

// Clock generation
initial clk = 0;
always #5 clk = ~clk; // 10ns clock period

// Test sequence
initial begin
    // Apply reset
    rst = 1;
    data_in = 8'b0;
    #10 rst = 0; // Release reset

    // Apply test inputs with delays for stabilization
    #20 data_in = 8'b11010101;
    #40 data_in = 8'b10101010;
    #40 data_in = 8'b11110000;

    // Allow time for observation
    #50;

    $finish; // End simulation
end

// Monitor signals
initial begin
    $monitor("Time: %0d, Data In: %b, CRC Out: %b", $time, data_in, crc_out);
end

endmodule
```

## SCREENSHOT

```

1 `timescale 1ns/1ps
2
3 module tb_CRC4Decoder;
4
5 // Inputs
6 reg clk;
7 reg rst;
8 reg [7:0] data_in;
9
10 // Outputs
11 wire [3:0] crc_out;
12
13 // Instantiate the CRC4Decoder module
14 CRC4Decoder uut (
15     .clk(clk),
16     .rst(rst),
17     .data_in(data_in),
18     .crc_out(crc_out)
19 );
20
21 // Clock generation
22 initial clk = 0;
23 always #5 clk = ~clk; // 10ns clock period
24
25 // Test sequence
26 initial begin
27     // Apply reset
28     rst = 1;
29     data_in = 8'b0;
30     #10 rst = 0; // Release reset
31
32     // Apply test inputs with delays for stabilization
33     #20 data_in = 8'b11010101;
34     #40 data_in = 8'b10101010;
35     #40 data_in = 8'b11110000;
36
37     // Allow time for observation
38     #50;
39
40     $finish; // End simulation
41 end
42
43 // Monitor signals
44 initial begin
45     $monitor("Time: %0d, Data In: %b, CRC Out: %b",
46         $time, data_in, crc_out);
47 end
48 endmodule
49

```

```

1 `timescale 1ns/1ps
2
3 module CRC4Decoder(
4     input clk,
5     input rst,
6     input [7:0] data_in,
7     output reg [3:0] crc_out
8 );
9
10 reg [7:0] shift_reg; // Shift register to process data
11 reg [4:0] polynomial = 5'b10011; // CRC-4 polynomial (5 bits)
12 integer i;
13 reg processing; // Flag to indicate computation is in progress
14
15 always @(posedge clk or posedge rst) begin
16     if (rst) begin
17         shift_reg <= 8'b0; // Reset the shift register
18         crc_out <= 4'b0; // Reset CRC output
19         processing <= 0; // Ensure processing flag is reset
20     end else if (processing) begin
21         shift_reg = data_in; // Load input data into the shift register
22         processing = 1; // Set processing flag
23         // Perform polynomial division for 8 bits
24         for (i = 0; i < 8; i = i + 1) begin
25             if (shift_reg[7]) begin
26                 // XOR the polynomial when MSB is 1
27                 shift_reg = (shift_reg << 1) ^ polynomial;
28             end else begin
29                 // Otherwise, just shift left
30                 shift_reg = shift_reg << 1;
31             end
32         end
33         crc_out = shift_reg[7:4]; // Extract the top 4 bits as CRC
34         processing = 0; // Reset processing flag when done
35     end
36 end
37 endmodule
38

```

eLog   <Share  
 [2024-12-16 13:43:00 UTC] iverilog '-wall' '-g2012' design.v testbench.v && unbuffer vvp a.out  
 Time: 0, Data In: 00000000, CRC Out: 0000  
 Time: 30, Data In: 11010101, CRC Out: 0000  
 Time: 35, Data In: 11010101, CRC Out: 1111  
 Time: 70, Data In: 10101010, CRC Out: 1111  
 Time: 110, Data In: 11110000, CRC Out: 1111  
 Time: 115, Data In: 11110000, CRC Out: 1110  
 testbench.v:40: \$finish called at 160000 (1ps)  
 Done