Digital System Design Lab

Lab 15

A Guessing Game

Student ID: D1166506

Name: 周嘉禾

Date: 2023/12/27

1. Objectives

- To become familiar with Verilog control
- To learn how to build pseudo-random combination with LFSR

2. Theorem

A Linear Feedback Shift Register (LFSR) is a type of shift register used in computing, where the input bit is a linear function of its previous state. The most commonly used linear function of single bits is exclusive-or (XOR). As a result, an LFSR is often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value.

The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state.

LFSRs have a wide range of applications, including generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. They can be implemented in both hardware and software.

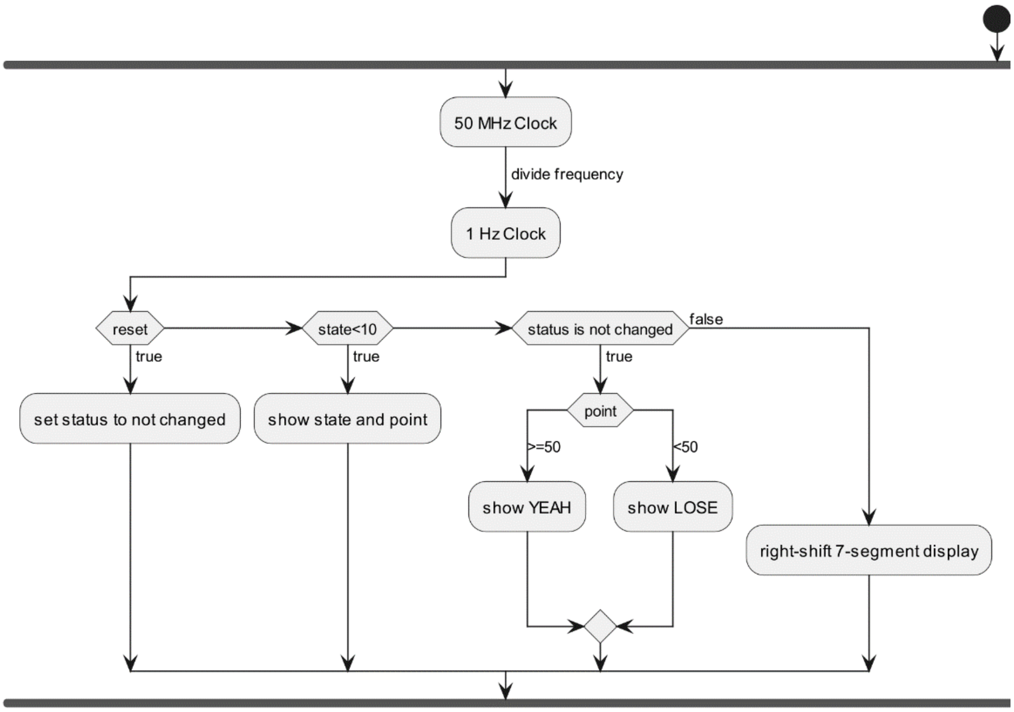
The bit positions that affect the next state are called the taps. In a maximum-length LFSR, it produces an m-sequence (i.e., it cycles through all possible 2^m - 1 states within the shift register except the state where all bits are zero), unless it contains all zeros, in which case it will never change.

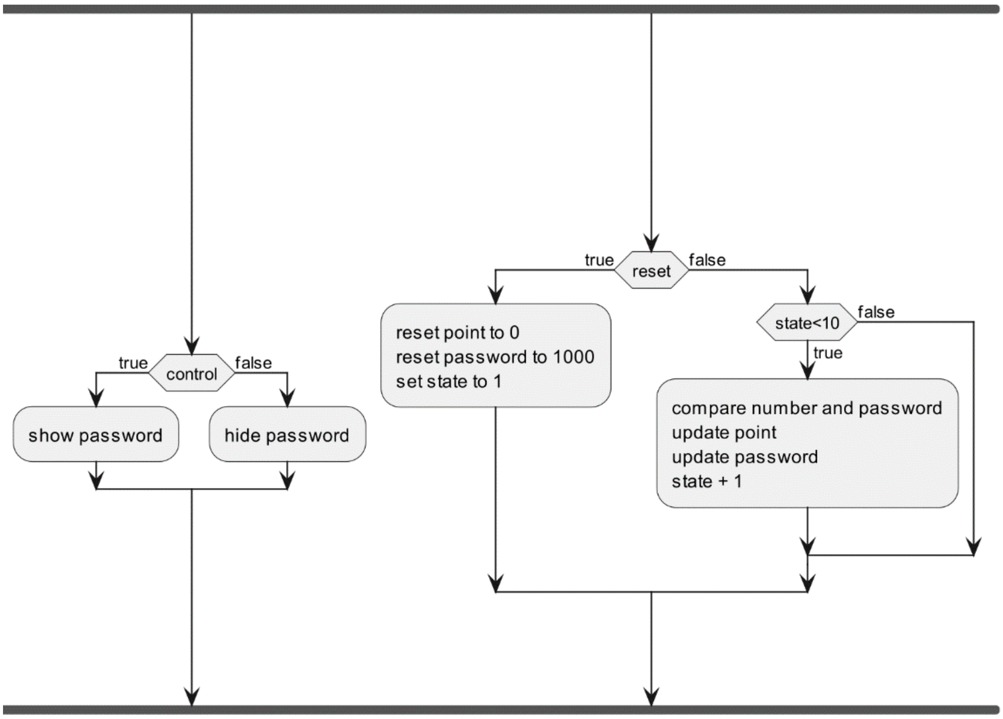
It's worth noting that the mathematics of a cyclic redundancy check, used to provide a quick check against transmission errors, are closely related to those of an LFSR.

In summary, LFSRs are a powerful tool in digital systems for their ability to generate sequences that appear random and have very long cycles, making them useful in a variety of applications.

3. Experimental Results

(1) Flow Chart

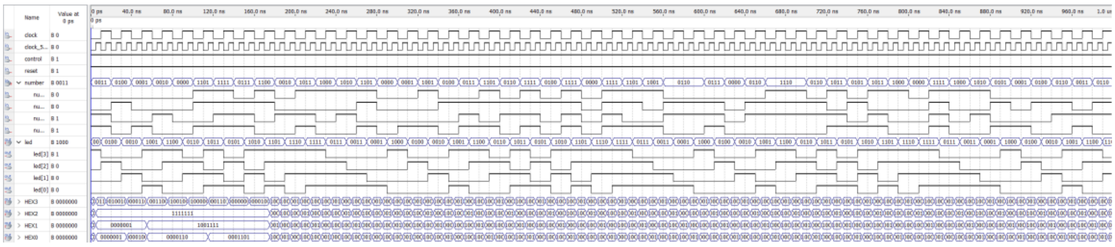




(2) Code

```
module step_4(clock, reset, control, number, led, HEX3, HEX2, HEX1,
HEX0, clock_50M);
        input clock, reset, control, clock_50M;
        input signed[3:0] number;
        output reg[3:0] led;
        output reg[0:6] HEX3, HEX2, HEX1, HEX0;

        reg [7:0] point=0, temp_point=0;
        reg Q=1'b0, last_Q=1'b0;
        reg change=1'b0;
        reg signed[3:0] password;
        integer count=0, state=1;

        initial begin
                password = 4'b1000;
        end

        function [0:6] converted_led;
                input [3:0] number;

                begin
                        case (number)
                                0: converted_led = 7'b0000001; // 0
                                1: converted_led = 7'b1001111; // 1
                                2: converted_led = 7'b0010010; // 2
                                3: converted_led = 7'b0000110; // 3
                                4: converted_led = 7'b1001100; // 4
                                5: converted_led = 7'b0100100; // 5
                                6: converted_led = 7'b0100000; // 6
                                7: converted_led = 7'b0001101; // 7
                                8: converted_led = 7'b0000000; // 8
                                9: converted_led = 7'b0000100; // 9
                                10: converted_led = 7'b0001000;
        // A
                                default: converted_led = 7'b1111111;
        // default
                        endcase
                end
        endfunction

        //      password
        always @(posedge clock or negedge reset) begin
                if (!reset) password <= 4'b1000;
                else password <= {password[1]^password[0],
password[3:1]};
        end

        //      state
        always @(posedge clock or negedge reset) begin
                if (!reset) state = 1;
                else if (state+1<=10) state = state + 1;
        end

        //      point
        always @(posedge clock or negedge reset) begin
                if (!reset) point <= 8'b00000000;
                else if (state<10) begin
                        if (number==password) temp_point = 9;
                        else if (number<password) temp_point = 4;
                        else temp_point = 0;
                        temp_point = (point[3:0]+temp_point>9) ? point
+ temp_point + 6 : point + temp_point;
                        if (temp_point[7:4]>9) temp_point = temp_point
+ 8'b01100000;
                        point <= temp_point;
                end
        end

        always @(control) begin
                if (control) led <= password;
                else led <= 4'b0000;
        end

        always @(posedge clock_50M) begin
                if(!reset) change = 0;
                else if (state<10) begin
                        HEX3 <= converted_led(state);
                        HEX2 <= 7'b1111111;
                        HEX1 <= converted_led(point[7:4]);
                        HEX0 <= converted_led(point[3:0]);
                end
                else if (!change) begin        // 8'b01010000 == 50 in
decimal
                        HEX3 <= (point>=8'b01010000) ? 7'b1000100 :
7'b1110001;
                        HEX2 <= (point>=8'b01010000) ? 7'b0110000 :
7'b1100010;
                        HEX1 <= (point>=8'b01010000) ? 7'b0001000 :
7'b0100100;
                        HEX0 <= (point>=8'b01010000) ? 7'b1001000 :
7'b0110000;
                        change = 1;
                end
                else if (change && (last_Q!=Q && Q)) begin HEX3 <=
HEX0; HEX2 <= HEX3; HEX1 <= HEX2; HEX0 <= HEX1; end
                last_Q <= Q;
        end

        always @(posedge clock_50M) begin
                if (count==24999999) begin
                        Q <= !Q;
                        count <= 0;
                end
                else count <= count + 1;
        end
endmodule
```

(3) Simulation



4. Comments

None

5. Problems & Solutions

None

6. Feedback

None