Digital System Design Lab

Lab 14

Digital Combination Lock with Pseudo-Random Combination

Student ID: D1166506

Name: 周嘉禾

Date: 2023/12/27

1. **Objectives**
   * To become familiar with finite state machine
   * 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 2m - 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.

1. **Experimental Results**
   1. **Flow Chart**

一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述

一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述

* 1. **State Diagram**一張含有 行, 圖表 的圖片

     自動產生的描述
  2. **Code**

module step\_3(enable, control, clock, reset, data, relock, status, led, clock\_50M, HEX3, HEX2, HEX1, HEX0);

input enable, control, clock, reset, data, relock;

input clock\_50M;

output reg[1:0] status=2'b00;

output reg[5:0] led=6'b111111;

output reg[0:6] HEX3, HEX2, HEX1, HEX0;

reg change=1'b0;

reg Q=1'b0, last\_Q=1'b0;

reg [5:0] password=6'b111111;

reg [2:0] state=0, last\_state=0;

integer count=0;

parameter open=6, alarm=7;

always @(posedge clock or negedge reset) begin

if (!reset) state = 0;

else if (relock && state==6) state = 0;

else begin

case (state)

0: state = (data==password[5]) ? 1 : 7;

1: state = (data==password[4]) ? 2 : 7;

2: state = (data==password[3]) ? 3 : 7;

3: state = (data==password[2]) ? 4 : 7;

4: state = (data==password[1]) ? 5 : 7;

5: state = (data==password[0]) ? 6 : 7;

endcase

end

end

always @(posedge clock\_50M) begin

if (!change || last\_state!=state) begin

case (state)

6: begin status = 2'b10; change = 1; end

7: begin status = 2'b01; change = 1; end

default: begin status = 2'b00; change = 0; end

endcase

case (state)

0: begin HEX3 <= 7'b0000001; HEX2 <= 7'b1111111; HEX1 <= 7'b1111111; HEX0 <= 7'b1111111; end // 0

1: begin HEX3 <= 7'b1001111; HEX2 <= 7'b1111111; HEX1 <= 7'b1111111; HEX0 <= 7'b1111111; end // 1

2: begin HEX3 <= 7'b0010010; HEX2 <= 7'b1111111; HEX1 <= 7'b1111111; HEX0 <= 7'b1111111; end // 2

3: begin HEX3 <= 7'b0000110; HEX2 <= 7'b1111111; HEX1 <= 7'b1111111; HEX0 <= 7'b1111111; end // 3

4: begin HEX3 <= 7'b1001100; HEX2 <= 7'b1111111; HEX1 <= 7'b1111111; HEX0 <= 7'b1111111; end // 4

5: begin HEX3 <= 7'b0100100; HEX2 <= 7'b1111111; HEX1 <= 7'b1111111; HEX0 <= 7'b1111111; end // 5

6: begin HEX3 <= 7'b0000001; HEX2 <= 7'b0011000; HEX1 <= 7'b0110000; HEX0 <= 7'b1101010; end // OPEN

7: begin HEX3 <= 7'b0110000; HEX2 <= 7'b1111010; HEX1 <= 7'b1111010; HEX0 <= 7'b1111110; end // alarm

default: begin HEX3 <= 7'b1111111; HEX2 <= 7'b1111111; HEX1 <= 7'b1111111; HEX0 <= 7'b1111111; end // default

endcase

last\_state <= state;

end

if (change && (last\_Q!=Q && Q)) begin HEX3 <= HEX0; HEX2 <= HEX3; HEX1 <= HEX2; HEX0 <= HEX1; end

if (change && (last\_Q!=Q && Q) && state==alarm) status[0] = !status[0];

last\_Q <= Q;

end

always @(posedge Q or negedge reset) begin

if (!reset) password <= 6'b100000;

else if (enable) password <= {password[0]^password[2], password[5:1]};

end

always @(control) begin

if (control) led <= password;

else led <= 6'b000000;

end

always @(posedge clock\_50M) begin

if (count==24999999) begin

Q <= !Q;

count <= 0;

end

else count <= count + 1;

end

endmodule

* 1. **Simulation**

一張含有 螢幕擷取畫面, 行, 文字, 數字 的圖片

自動產生的描述

1. **Comments**

None

1. **Problems & Solutions**

None

1. **Feedback**

None