姓名： Epi　　实验日期： 2024 年 11 月 1 日

1、实验名称：实验三 时序逻辑实验（一）

2、实验目的：

本实验的目的是学习时序逻辑模块（状态机）在数字系统中的应用；掌握实验平台的外部功能模块在数字系统设计中的应用。

3、实验内容：

(1) **实验 3.1——流水灯**

a) 了解分频模块的功能；

b) 实现流水灯的流动功能；

c) 修改代码，实现流水灯的**变速**流动功能。

(2) **实验 3.2——智能报警器**

a) 了解消抖模块的功能；

b) 完成智能报警器。

(3) **实验 3.3——密码锁**

a) 按照要求实现密码锁；

b) 本实验无需上版，只需仿真即可。

实验步骤：

4、**实验 3.1 的实现及仿真验证**

核心代码：

1. module counter 的部分实现

```
always @(*) begin
    case (speed)
        // 3'b000 : begin maxcount = 14'd10000; rst = 1; end
        3'b000: maxcount = 14'd10000; // 状态 0
        3'b001: maxcount = 14'd08714; // 状态 1
        3'b010: maxcount = 14'd07428; // 状态 2
        3'b011: maxcount = 14'd06142; // 状态 3
        3'b100: maxcount = 14'd04856; // 状态 4
        3'b101: maxcount = 14'd03570; // 状态 5
        3'b110: maxcount = 14'd02284; // 状态 6
        3'b111: maxcount = 14'd01000; // 状态 7
```

```
            default: maxcount = 14'd10000;
        endcase
        //  maxcount  =  maxcount/100;    //this  line  is  for
simulation, to make the counter run faster for purposes
    end

    always @(posedge clk) begin
        if (rst)
            cnt_first <= 14'd0;
        else if (cnt_first == maxcount)
            cnt_first <= 14'd0;
        else
            cnt_first <= cnt_first + 1'b1;
    end

    always @(posedge clk) begin
        if (rst)
            cnt_second <= 14'd0;
        else if (cnt_second == maxcount)
            cnt_second <= 14'd0;
        else if (cnt_first == maxcount)
            cnt_second <= cnt_second + 1'b1;
    end

    assign clk_bps = (cnt_second == maxcount) ? 1'b1 : 1'b0;
endmodule
```

2. 顶层模块的实现

```
module epi_flash_led_top(
    input clk,
    input [7:0] speed_ctrl,
    output wire [15:0] led
);
    wire clk_bps;
    wire [2:0] speedgrade;
    reg [2:0] grade;
    reg rst;
    integer i;

    always @(*) begin
        grade = 3'd0;
        for (i = 0; i < 8; i = i + 1) begin
            if (speed_ctrl[i]) begin
                grade = grade + 1;
```

```
            end
        end

        if (grade == 3'd0) begin
            rst = 1'b1;
        end
        else begin
            rst = 1'b0;
            // $display("grade: %d, rst: %b", grade, rst);
        end
    end

    counter counter_inst(
        .clk(clk),
        .rst(rst),
        .speed(grade),
        .clk_bps(clk_bps)
    );

    flash_led_ctl flash_led_ctl_inst(
        .clk(clk),
        .rst(rst),
        .dir(speed_ctrl[7]),
        .clk_bps(clk_bps),
        .led(led)
    );
endmodule
```

3. 测试激励代码

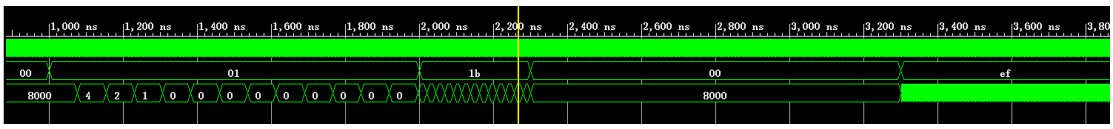
```
module epi_flash_led_top_tb;
        reg clk;
         reg [7:0] speed_ctrl;
        wire [15:0] led;
        initial begin
            clk = 1'b0;
            speed_ctrl = 8'b00000000;
            #1000000 speed_ctrl = 8'b00000001;
            #1000000 speed_ctrl = 8'b00011011;
            #300000 speed_ctrl = 8'b00000000;
            #1000000 speed_ctrl = 8'b11101111;
        end
        always #5 clk <= ~clk;
        epi_flash_led_top epi_flash_led_top(
            .clk( clk ),
```

```
            .speed_ctrl( speed_ctrl ),
            .led( led )
        );
endmodule
```

功能声明：把输入的 8 位二进制数作为补码，正数右移，负数左移，0 表示复位。二进制中 1 的数目表示移动速率。

（1）流水灯仿真结果截图，对波形进行简要解释

仿真代码见 **counter.v**

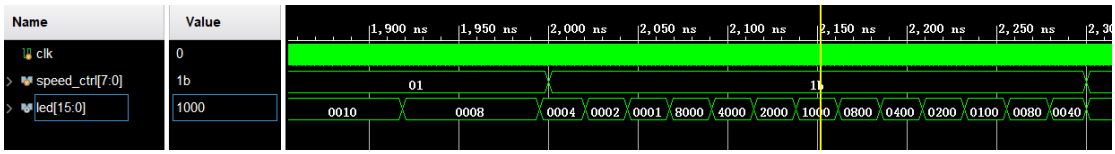


仿真总览

通过总览，可以看出信号变化的频率有明显的改变，下面具体分析各段。

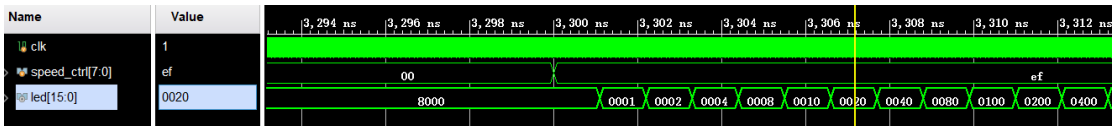
Part1

当 **speed_ctrl** 输入为 **8'b00000001(8'h01)**时，可以看到 **led** 输出在不断右移，间隔约为 76ns。



Part2

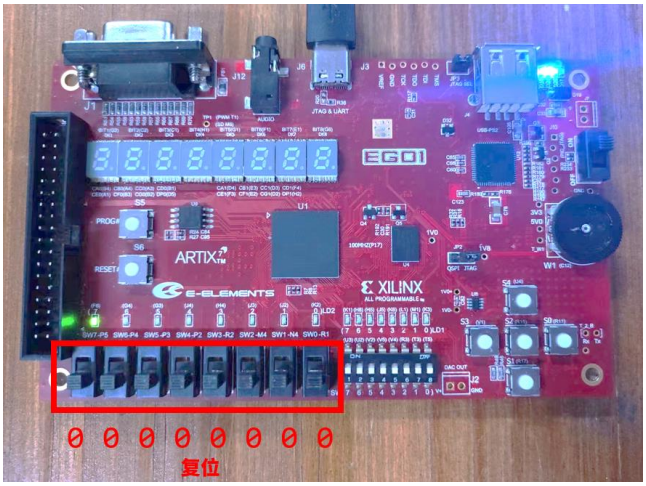
当 **speed_ctrl** 输入为 **8'b00011011 (8'h1b)**时，可以看到 **led** 输出在不断右移，间隔约为 25ns。



Part3

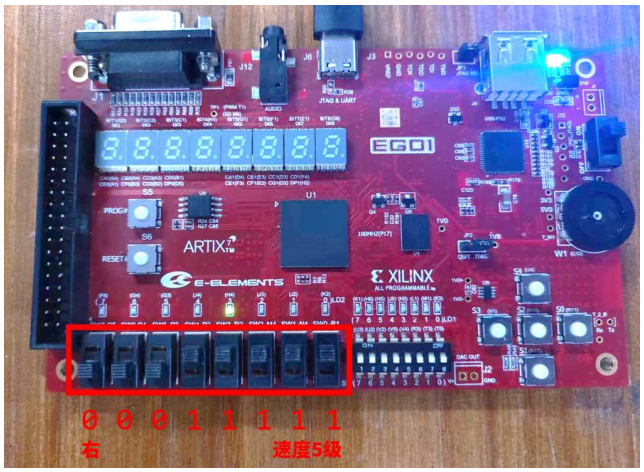
当 **speed_ctrl** 输入为 **8'b11101111 (8'hef)**时，可以看到 **led** 输出在不断右移，间隔约为 1ns。

（2）给出板子运行结果照片，以及操作过程



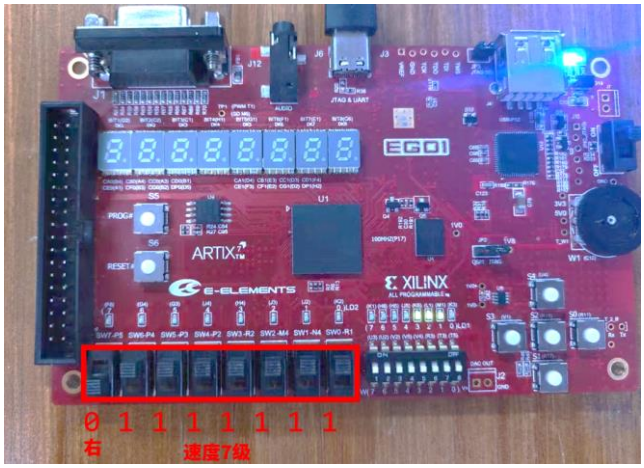
复位测试

输入 **00000000**，进入复位态，最左侧的指示灯亮起，流水灯不工作。



5 级速度

输入 **00011111**，流水灯向右侧以中等速度流动，图片抓拍了其中的一瞬间。



7 级速度

输入 **01111111**，流水灯向右侧高速流动，图片抓拍了其中的一瞬间，由于 7 级速度频闪较快，拍照时有残留影像故能看到有三个灯亮着。

（3）counter 模块的作用是什么？

在代码中，counter.v 对时钟信号进行分频，以得到（不同的）更低的时钟速率，从而控制流水灯的速度。

5、实验 3.2 的实现及验证

（1）请说明你对消抖模块功能的理解

当我们按下一个按钮时，按钮的机械部件可能会由于弹性、接触不良或外界干扰等因素，导致按钮的状态在短时间内多次发生变化。这种抖动现象可能会被错误地认为是多次按下或释放。所以我们消抖功能的设计是**必要的**。

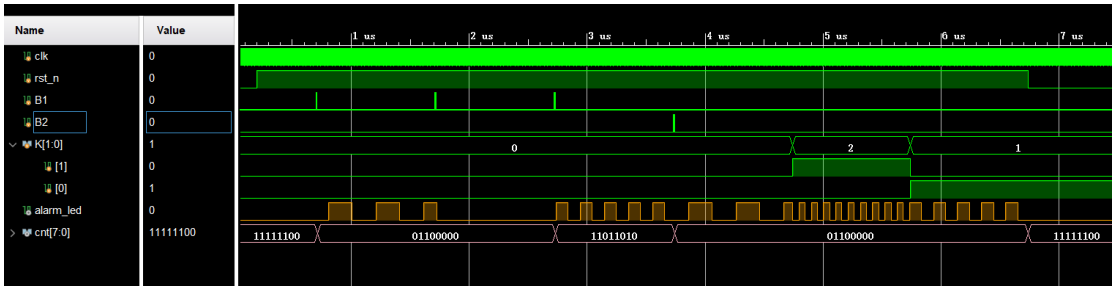
朴素的，我们连续在若干个上升周期取样，如果连续三个状态为 1 且下一个状态为 0，考虑为按键稳定。这一功能可以借助状态机实现。

```
always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            key_state <= 3'b000;
            key_flag <= 1'b0;
            key_last <= 1'b1;
        end else begin
            key_last <= key_in;

            if (key_in == key_last) begin
                key_state <= {key_state[1:0], key_in};
            end
            else begin
                key_state <= 3'b000;
            end
            if (key_state == 3'b111 && !key_in) begin
                key_flag <= 1'b1;
            end
            else begin
                key_flag <= 1'b0;
            end
        end
    end
```

（2）该模块的仿真验证波形及说明

声明：为仿真方便，对时钟分频模块的代码进行了部分修改。

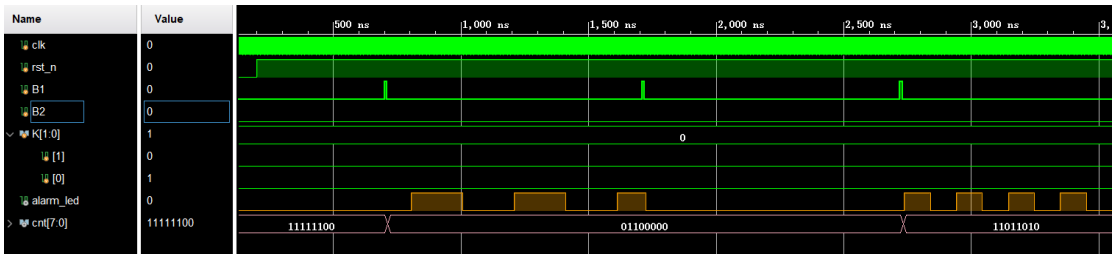
仿真总体波形图见下：



自动报警系统的仿真波形图总览

下面是每块的仿真波形分析：

➢ 500-3000ns 是对 B1 功能的测试,，观察波形图：



500-3000ns 的仿真波形

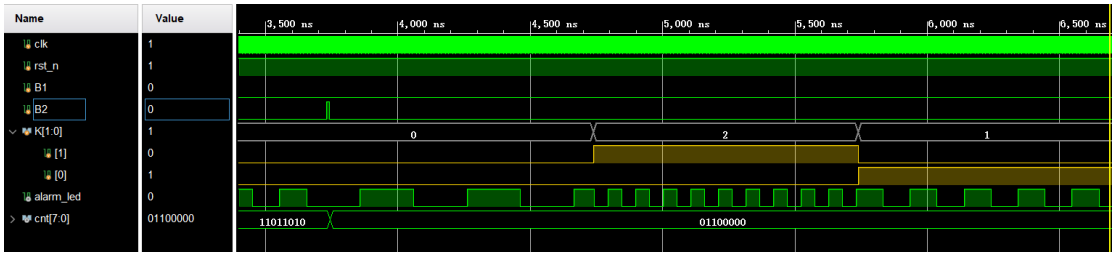
✧ 当 B1 第一次输入后，alarm_led 开始按频率波动（经测量约 200ns），此时的 cnt 变为七段数码管的数字“1”，表示触发了一次报警。

✧ 当 B1 第二次输入后，alarm_led 不再波动，报警停止。

✧ 当 B1 第三次输入后，alarm_led 开始按频率波动（经测量约 100ns），此时的 cnt 变为七段数码管的数字“2”，表示触发了两次报警。

至此，我们完成了对触发警报，记录次数，按次数决定频闪频率的功能验证。

➢ 3500-6500ns 是对 B2，K1，K2 功能的测试,，观察波形图：



3500-6500ns 的仿真波形

✧ 当 B2 输入后，cnt 立刻变为七段数码管的数字“1”，重置次数，alarm_led 也还原为 200ns 的频率波动。

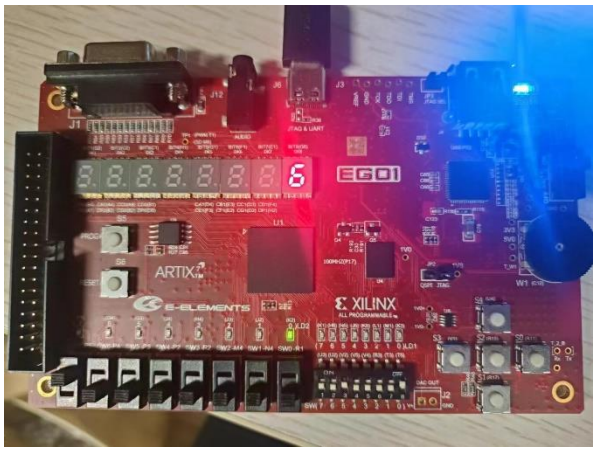
✧ K2（即图中的 K[1]）输入，可以明显的观察到 alarm_led 频率波动加快，经测量为 50ns。

✧ K1（即图中的 K[0]）输入，可以明显的观察到 alarm_led 频率波动减慢，但仍比原来的 200ns 快，经测量为 100ns。

至此，我们完成了对计数复原，K 开关频率控制的功能验证。

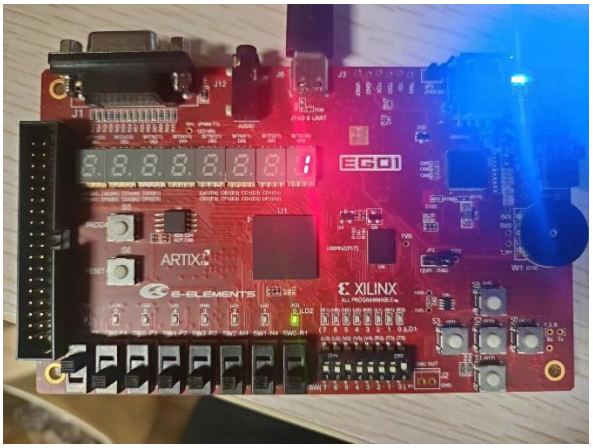
（3）板子运行结果照片

由于照片拍不出来运行的实际效果，所以此处简略的说明操作过程，具体的待验收时展示。



警报与计数功能

反复按下 S0 按钮，数码管显示数字不断增加，报警灯闪烁。



还原计数功能

按下 S3 按钮，数码管计数还原为 1，报警灯依旧闪烁。

主模块及测试激励代码：

```
module main(
    input wire clk, input wire rst_n, input wire B1, input wire B2,
    input wire [1:0] K, output wire alarm_led,
    output wire [3:0] pos, output wire [7:0] cnt
);
    wire B1_filtered; wire B2_filtered;
    reg alarm_flag; reg [3:0] num_cnt; reg [4:0] rate; wire clk_bps;
    assign pos = 4'b0001;
    key_filter B1_filter_inst(
        .clk(clk),.rst_n(rst_n),.key_in(B1),.key_flag(B1_filtered)
    );
    key_filter B2_filter_inst(
        .clk(clk),.rst_n(rst_n),.key_in(B2),.key_flag(B2_filtered)
    );

    always @(posedge clk or negedge rst_n) begin
```

```
        if (!rst_n) begin
            alarm_flag <= 1'b0;
            num_cnt <= 4'b0000;
            rate <= 5'b00000;
        end
        else begin
            if (B1_filtered) begin
                alarm_flag <= !alarm_flag;
                if (!alarm_flag) begin
                    num_cnt <= num_cnt + 1;
                end
            end

            if (B2_filtered) begin
                num_cnt <= 4'b0001;
            end
            case (K)
                2'b01: rate <= num_cnt*2;
                2'b10: rate <= num_cnt*4;
                default: rate <= num_cnt;
            endcase
        end
    end
    segmsg segmsg_inst(.seg(num_cnt),.led(cnt)
    );

    HzDown HzDown_inst(
        .clk(clk),.rst_n(rst_n),.rate(rate),.clk_bps(clk_bps)
    );

    assign alarm_led = alarm_flag && clk_bps;
endmodule
module main_tb;
    //此处略去变量声明及实例化
    initial begin
        clk = 0;
        forever #1 clk = ~clk;
    end
    initial begin
        rst_n = 0; B1 = 0; B2 = 0; K = 2'b00;
        #200 rst_n = 1;
        #500 B1 = 1; #10 B1 = 0;
        #1000 B1 = 1; #10 B1 = 0;
        #1000 B1 = 1; #10 B1 = 0;
```

```
        #1000 B2 = 1; #10 B2 = 0;
        #1000 K = 2'b10;
        #1000 K = 2'b01;
        #1000 rst_n = 0;
        $stop;
    end
endmodule
```

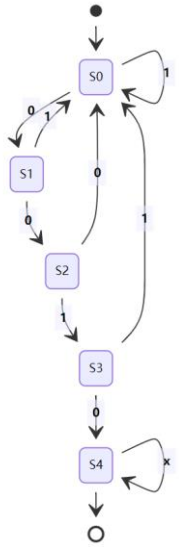
6、实验 3.3 的实现及仿真验证

（1）密码锁模块的代码，及简要解释

```
module lock(
    input clk,
    input rst,
    input in,
    output reg unlock
);
parameter S0 = 3'b000;
parameter S1 = 3'b001;
parameter S2 = 3'b010;
parameter S3 = 3'b011;
parameter S4 = 3'b100;

reg [2:0] current_state, next_state;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        current_state <= S0;
        unlock <= 0;
    end else begin
        current_state <= next_state;
    end
end

always @(*) begin
    case (current_state)
        S0: next_state = (in == 1'b0) ? S1 : S0;
        S1: next_state = (in == 1'b0) ? S2 : S0;
        S2: next_state = (in == 1'b1) ? S3 : S0;
        S3: next_state = (in == 1'b0) ? S4 : S0;
        S4: next_state = S4;
        default: next_state = S0;
    endcase
end
```

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        unlock <= 0;
    end else if (current_state == S4) begin
        unlock <= 1;
    end
end
endmodule
```

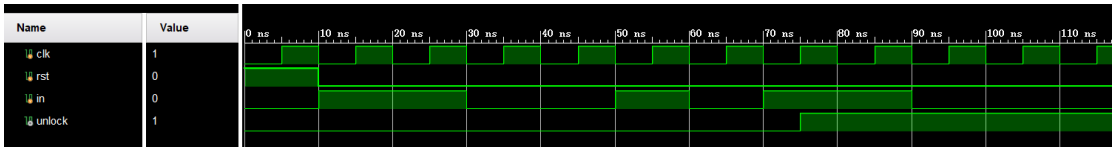
这是一个 Mealy 状态机，简化的状态转移图如下所示。S0-S4 代表了逐步解锁的状态。S4 为解锁态（unlock==1）



状态转移图

（2）密码锁的仿真截图，并简要说明波形的正确性

仿真截图：



分析与说明见（3）

（3）测试激励的设计

下面回答上述两问题，我设计了测试输入“110010110”（具体见代码），可以看出在其中的第 6 位出现了密码（110**0**1**0**110），观察波形图可以看出确实在输入 0 的下一个时钟后，密码锁一直保持解锁状态，由此验证功能的正确性。

测试激励代码如下

```
module lock_tb;
    //此处略去变量声明
```

```
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        rst = 1;
        in = 0;
        #10 rst = 0;

        //"11001"
        in = 1; #10;
        in = 1; #10;
        in = 0; #10;
        in = 0; #10;
        in = 1; #10;
        //"0"->"0010"
        in = 0; #10;

        in = 1; #10;
        in = 1; #10;
        in = 0; #10;
        #50;
        $stop;
    end
endmodule
```

7、实验中遇到的问题、现象及解决方法

问题 1：关于 step1 流水灯，仿真的时候看不到波形变化？

问题原因：分频太慢了，波形变化需要相当长的时间。

解决方法：让分频出的 clk_bps 加快即可

问题 2：怎么写消抖模块，可以直接写成下面的样子吗？

```
///wrong code///
module key_filter
(input wire key_in, output wire key_flag);
reg key_flag_reg;
always@(*) begin
    if(key_in == 1'b1) begin
        #10 key_flag_reg = 1'b1;
    end
    else key_flag_reg = 1'b0;
```

```
end
assign key_flag = key_flag_reg;
endmodule
```

解决方法：不可以这样写！具体的写法见上。

9、本次实验心得体会

深入理解了时序逻辑在代码中的应用，体会到了阻塞赋值与非阻塞赋值在时钟周期上的不同表现。学会了初步使用状态机。

10、关于本次实验课程的改进建议

有关于仿真的说明可以更详细一点，可以像 step3 密码锁一样给出说明。