中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：信号处理及有限状态机

学生姓名：_____徐奥_____

学生学号：_____PB20061343_____

完成日期：__2021 年 12 月 8 日__

计算机实验教学中心制

2020 年 09 月

【实验题目】

本次实验中，我们将介绍几种常用的数字信号处理技巧并学习一种数字电路开发中非常重要的设计方法：有限状态机（FSM：Finite State Machine）

【实验练习】

题目 1. 在不改变电路功能和行为的前提下，将前面 Step5 中的代码改写成三段式有限状态机的形式，写出完整的 Verilog 代码

代码如下：

```
module test(
    input clk,rst,
    output led
);
reg [1:0] cnt;
reg [1:0] nxt;

always @(*)
    begin
        case(cnt)
            2'b00:  nxt<=2'b01;
            2'b01:  nxt<=2'b10;
            2'b10:  nxt<=2'b11;
            2'b11:  nxt<=2'b00;
            default: nxt<=2'b00;
        endcase
    end

always@(posedge clk or posedge rst)
    begin
        if(rst)
            cnt <= 2'b0
        else
            cnt <= nxt;
    end

assign led = (cnt==2'b11) ? 1'b1 : 1'b0;

endmodule
```

图 1

题目 2. 请在 Logisim 中设计一个 4bit 位宽的计数器电路，clk 信号为计数器时钟，复位时（rst==1）计数值为 0，在输入信号 sw 电平发生变化时，计数值 cnt 加 1，即在 sw 信号上升沿时刻和下降沿时刻各触发一次计数操作，其余时刻计数器保持不变

首先搭建一个电路，实现通过按键信号生成一个时钟周期宽度的脉冲信号，即 sw 发生变化时，会产生一个时钟周期宽度的脉冲信号，电路如下图：

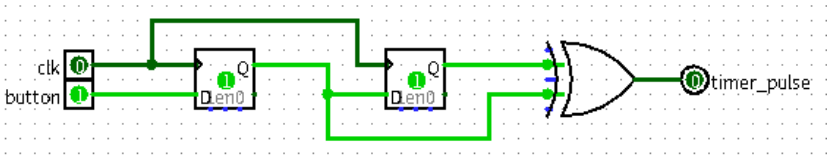

图 2

然后搭建 4bit 位宽的计数器，使用 Logisim 自带的计数器，调整其为加一计数器，并且 rst==1 时计数值为 0，如下图：

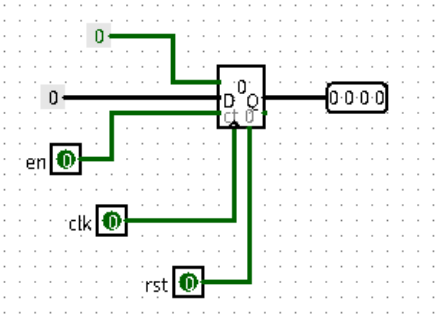

图 3

将上述两个电路封装，搭建题目要求的计数器，如下图：

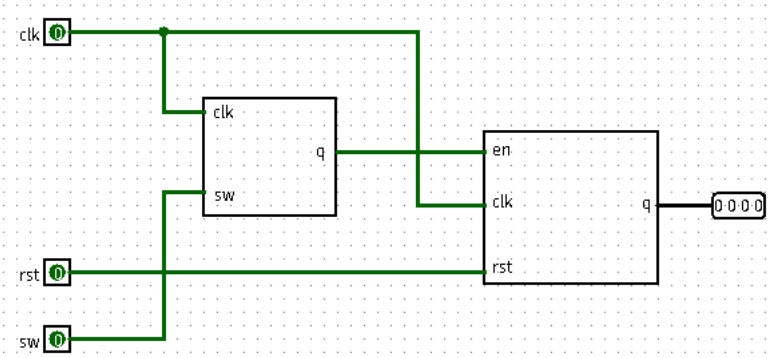

图 4

最后将其封装，在 1KHz 的时钟频率下实现题目要求，如下图：

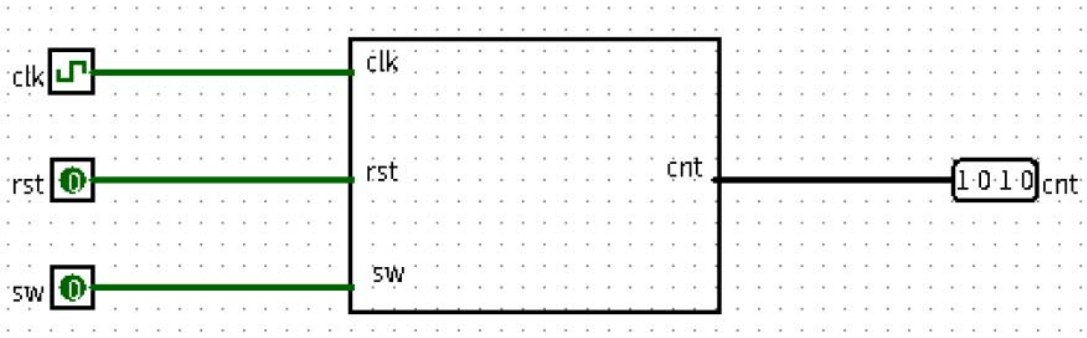

图 5

题目 3. 设计一个 8 位的十六进制计数器，时钟采用板载的 100MHz 时钟，通过 sw[0]控制计数模式，开关为 1 时为累加模式，为 0 时为递减模式，按键控制计数，按下的瞬间根据开关的状态进行累加或递减计数。计数值用数码管显示，其复位值为“1F”。

首先，编写 Verilog 代码。

通过按键 button 产生一个时钟周期宽度的脉冲信号，使得该信号在 button 信号的上升沿附近为高电平，其余时间为低电平，代码如下：

```
module signal_edge(
    input clk,
    input button,
    output button_edge
    );

    reg button_r1,button_r2;

    always@(posedge clk)
        button_r1 <= button;
    always@(posedge clk)
        button_r2 <= button_r1;

    assign button_edge = button_r1 & (~button_r2);

endmodule
```

图 6

该模块将作为子模块，引用代码如下：

```
wire button_edge;
signal_edge signal_edge_1(clk,button,button_edge);
```

图 7

对 8 位十六进制数进行计数，如果标志位（add）为 1，那么为加一计数，否则为减一。并且只有当 button==1 时，才进行计数操作，代码如下：

```
reg [31:0] number;
always@(posedge clk)
begin
    if(rst == 1) number <= 32'h1f;
    else
        begin
            if(button_edge == 1)
            begin
                if (add) number <= number + 1;
                else number <= number - 1;
            end
            else number <= number;
        end
end
```

图 8

利用视觉暂留效应实现同时输出 8 位十六进制数，代码如下：

```
    reg [31:0] cnt;
    always@(posedge clk)
    begin
        if (cnt >= 1000000) cnt <= 0;
        else cnt <= cnt + 1;
    end

    always@(posedge clk)
    begin
        if (cnt == 0)
        begin
            if (hexplay_an == 7) hexplay_an <= 0;
            else hexplay_an <= hexplay_an + 1;
        end
    end

    always@(*) begin
        case(hexplay_an)
            3'b000: hexplay_data = number[3:0];
            3'b001: hexplay_data = number[7:4];
            3'b010: hexplay_data = number[11:8];
            3'b011: hexplay_data = number[15:12];
            3'b100: hexplay_data = number[19:16];
            3'b101: hexplay_data = number[23:20];
            3'b110: hexplay_data = number[27:24];
            3'b111: hexplay_data = number[31:28];
        endcase
    end

endmodule
```

图 9

然后，修改引脚约束文件，使用到如下几个引脚：

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_3
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
```

图 10

```
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { add }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { rst }];
```

图 11

```
set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[0] }];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[1] }];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[2] }];
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[3] }];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[0] }];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[1] }];
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[2] }];

## FPGAOL BUTTON & SOFT_CLOCK

set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { button }];
```

图 12

完整代码如下：

```
module signal_edge(
    input clk,
    input button,
    output button_edge
    );

    reg button_r1,button_r2;

    always@(posedge clk)
        button_r1 <= button;
    always@(posedge clk)
        button_r2 <= button_r1;

    assign button_edge = button_r1 & (~button_r2);

endmodule


module q3_v(
    input clk,
    input rst,
    input add,
    input button,
    output reg [3:0] hexplay_data,
    output reg [2:0] hexplay_an
    );

    wire button_edge;
    signal_edge signal_edge_1(clk,button,button_edge);

    reg [31:0] number;
    always@(posedge clk)
    begin
        if(rst == 1) number <= 32'h1f;
        else
            begin
                if(button_edge == 1)
                begin
                    if (add) number <= number + 1;
                    else number <= number - 1;
                end
```

图 13

```
                else number <= number;
            end
    end

    reg [31:0] cnt;
    always@(posedge clk)
    begin
        if (cnt >= 1000000) cnt <= 0;
        else cnt <= cnt + 1;
    end

    always@(posedge clk)
    begin
        if (cnt == 0)
        begin
            if (hexplay_an == 7) hexplay_an <= 0;
            else hexplay_an <= hexplay_an + 1;
        end
    end

    always@(*) begin
        case(hexplay_an)
            3'b000: hexplay_data = number[3:0];
            3'b001: hexplay_data = number[7:4];
            3'b010: hexplay_data = number[11:8];
            3'b011: hexplay_data = number[15:12];
            3'b100: hexplay_data = number[19:16];
            3'b101: hexplay_data = number[23:20];
            3'b110: hexplay_data = number[27:24];
            3'b111: hexplay_data = number[31:28];
        endcase
    end

endmodule
```

图 14

题目 4. 使用有限状态机设计一个序列检测电路，并进行计数，当检测到输入序列为“1100”时，计数器加一，用一个数码管显示当前状态编码，一个数码管显示检测到目标序列的个数，用 4 个数码管显示最近输入的 4 个数值，用 sw[0]进行数据的串行输入，按键每按下一次将输入一次开关状态，时钟采用板载的 100MHz 时钟。要求画出状态跳转图，并在 FPGA 开发板上实现电路

首先，画出状态跳转图：

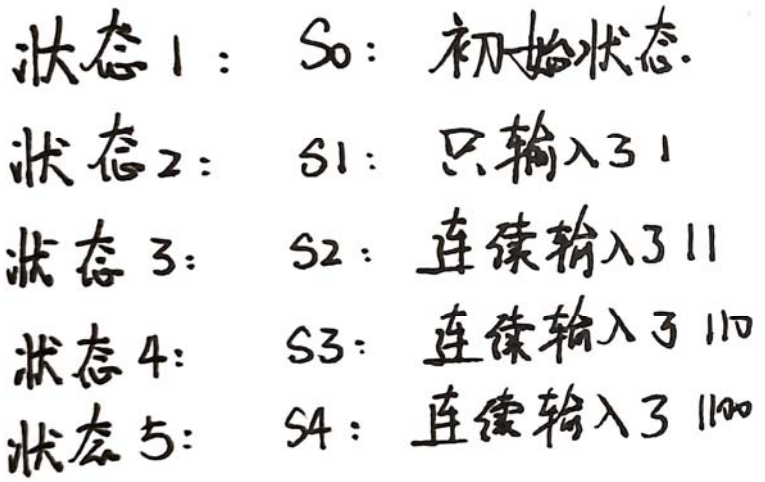

图 15

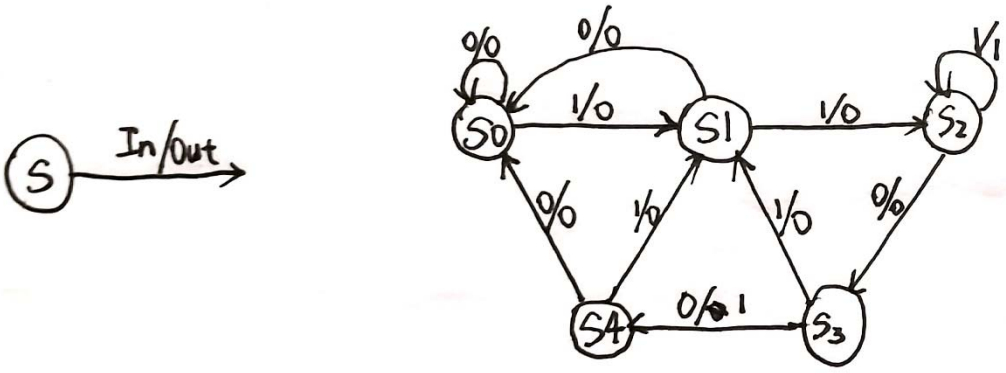

图 16

通过按键 button 产生一个时钟周期宽度的脉冲信号，使得该信号在 button 信号的上升沿附近为高电平，其余时间为低电平，代码如下：

```
module signal_edge(
    input clk,
    input button,
    output button_edge
    );
    reg button_r1,button_r2;
    always@(posedge clk)
        button_r1 <= button;
    always@(posedge clk)
        button_r2 <= button_r1;
    assign button_edge = button_r1 & (~button_r2);
endmodule
```

图 17

保存最近读入的四个数值：

```
reg [3:0] last_in;
always @(posedge clk) begin
    if(button_edge) begin
        last_in[3:1]=last_in[2:0];
        last_in[0]=innum;
    end
end
```

图 18

并准备将其以十六进制格式输出

```
//准备输出最近输入的四个值，需要转化为16进制
reg [15:0] hex_in;
initial begin
    hex_in<=16'b0;
end

always @(*) begin
    hex_in[12]<=last_in[3];
    hex_in[8]<=last_in[2];
    hex_in[4]<=last_in[1];
    hex_in[0]<=last_in[0];
end
```

图 19

将五个状态编码：

```
parameter S0=3'b000;
parameter S1=3'b001;
parameter S2=3'b010;
parameter S3=3'b011;
parameter S4=3'b100;
```

图 20

求出次态：

```
//状态转移
always @(*) begin
    if(button_edge) begin
        case(cur)
            S0: begin
                if(innum) nxt<=S1;
                else nxt<=S0;
            end
            S1: begin
                if(innum) nxt<=S2;
                else nxt<=S0;
            end
            S2: begin
                if(!innum) nxt<=S3;
                else nxt<=S2;
            end
            S3: begin
                if(!innum) nxt<=S4;
                else nxt<=S1;
            end
            S4: begin
                if(innum) nxt<=S1;
                else nxt<=S0;
            end
            default: begin
                nxt<=3'b0;
            end
        endcase
    end
end
```

图 21

在时钟有效边沿状态转移：

```
//更新现态
always @(posedge clk) begin
    if(button_edge) begin
        cur = nxt;
        if(cur==S4) count = count+1;
    end
end
```

图 22

为输出到 FPGA 上，设计刷新与输出：

```
//为更新到FPGA上做准备
reg [31:0] cnt;
always @(posedge clk) begin
    if(cnt>=1000000) cnt<=0;
    else cnt<=cnt+1;
end

always @(posedge clk) begin
    if(cnt==0) begin
        if(hexplay_an>=5) hexplay_an<=0;
        else hexplay_an<=hexplay_an+1;
    end
end
```

图 23

```
//将状态转化为十六进制输出
reg [3:0] status;
always @(*) begin
    case(cur)
        S0:status[3:0]<=3'b000;
        S1:status[3:0]<=3'b001;
        S2:status[3:0]<=3'b010;
        S3:status[3:0]<=3'b011;
        S4:status[3:0]<=3'b100;
        default: status[3:0]<=3'b000;
    endcase
end

//数码管0~3表示最近输入的四个数值
//数码管4表示当前状态编码
//数码管5表示目标序列出现的次数
always @(*) begin
    case(hexplay_an)
        0: hexplay_data=hex_in[3:0];
        1: hexplay_data=hex_in[7:4];
        2: hexplay_data=hex_in[11:8];
        3: hexplay_data=hex_in[15:12];
        4: hexplay_data=status;
        5: hexplay_data=count;
        default: hexplay_data=3'b0;
    endcase
end
```

图 24

完整代码为：

```
module signal_edge(
    input clk,
    input button,
    output button_edge
    );
    reg button_r1,button_r2;
    always@(posedge clk)
        button_r1 <= button;
    always@(posedge clk)
        button_r2 <= button_r1;
    assign button_edge = button_r1 & (~button_r2);
endmodule

//数码管0~3表示最近输入的四个数值
//数码管4表示当前状态编码
//数码管5表示目标序列出现的次数
module q4_v (
    input clk,
    input innum,//串行输入的数据
    input button,//按键每按下一次将输入一次开关状态
    output reg [3:0] hexplay_data,
    output reg [2:0] hexplay_an
);

    wire button_edge;
    signal_edge signal_edge(clk,button,button_edge);

    reg [3:0] count;//计数目标序列出现的次数
    initial begin
        count=0;
    end

    reg [3:0] last_in;
    always @(posedge clk) begin
        if(button_edge) begin
            last_in[3:1]=last_in[2:0];
            last_in[0]=innum;
        end
    end
```

图 25

```
//准备输出最近输入的四个值，需要转化为16进制
reg [15:0] hex_in;
initial begin
    hex_in<=16'b0;
end

always @(*) begin
    hex_in[12]<=last_in[3];
    hex_in[8]<=last_in[2];
    hex_in[4]<=last_in[1];
    hex_in[0]<=last_in[0];
end

parameter S0=3'b000;
parameter S1=3'b001;
parameter S2=3'b010;
parameter S3=3'b011;
parameter S4=3'b100;

reg [2:0] cur;
reg [2:0] nxt;

//状态转移
always @(*) begin
    if(button_edge) begin
        case(cur)
            S0: begin
                if(innum) nxt<=S1;
                else nxt<=S0;
            end
            S1: begin
                if(innum) nxt<=S2;
                else nxt<=S0;
            end
            S2: begin
                if(!innum) nxt<=S3;
                else nxt<=S2;
            end
```

图 26

```
            S3: begin
                if(!innum) nxt<=S4;
                else nxt<=S1;
            end
            S4: begin
                if(innum) nxt<=S1;
                else nxt<=S0;
            end
            default: begin
                nxt<=3'b0;
            end
        endcase
    end
end

//更新现态
always @(posedge clk) begin
    if(button_edge) begin
        cur = nxt;
        if(cur==S4) count = count+1;
    end
end

//为更新到FPGA上做准备
reg [31:0] cnt;
always @(posedge clk) begin
    if(cnt>=1000000) cnt<=0;
    else cnt<=cnt+1;
end

always @(posedge clk) begin
    if(cnt==0) begin
        if(hexplay_an>=5) hexplay_an<=0;
        else hexplay_an<=hexplay_an+1;
    end
end
```

图 27

```
    //将状态转化为十六进制输出
    reg [3:0] status;
    always @(*) begin
        case(cur)
            S0:status[3:0]<=3'b000;
            S1:status[3:0]<=3'b001;
            S2:status[3:0]<=3'b010;
            S3:status[3:0]<=3'b011;
            S4:status[3:0]<=3'b100;
            default: status[3:0]<=3'b000;
        endcase
    end

    //数码管0~3表示最近输入的四个数值
    //数码管4表示当前状态编码
    //数码管5表示目标序列出现的次数
    always @(*) begin
        case(hexplay_an)
            0: hexplay_data=hex_in[3:0];
            1: hexplay_data=hex_in[7:4];
            2: hexplay_data=hex_in[11:8];
            3: hexplay_data=hex_in[15:12];
            4: hexplay_data=status;
            5: hexplay_data=count;
            default: hexplay_data=3'b0;
        endcase
    end

endmodule //q4_v
```

图 28

引脚约束文件为：

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100M
```

图 29

```
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { innum }];
```

图 30

```
set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[0] }];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[1] }];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[2] }];
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports { hexplay_data[3] }];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[0] }];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[1] }];
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports { hexplay_an[2] }];

## FPGAOL BUTTON & SOFT_CLOCK

set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { button }];
```

图 31

【总结与思考】

1. 本次实验前三题相对比较简单，第四题在思维量和代码量上都有较大的难度提升
2. 我在第一次写第四题的 Verilog 代码时，没有在写之前就想清楚各模块作用，导致最后写得比较混乱，无法生成有效的比特流文件，最终是重新写了一遍，并且在第二遍时加上了很多注释，写起来更有条理一些
3. 本次实验量中等偏多
4. 本次实验对于 Verilog 代码能力有充分的练习