实验 08 信号处理及有限状态机

实验目的

- 进一步熟悉 FPGA 开发的整体流程
- 掌握几种常见的信号处理技巧
- 掌握有限状态机的设计方法
- 能够使用有限状态机设计功能电路

实验环境

- VLAB: vlab.ustc.eud.cn
- FPGAOL: fpgaol.ustc.edu.cn
- Logisim
- Vivado

实验题目

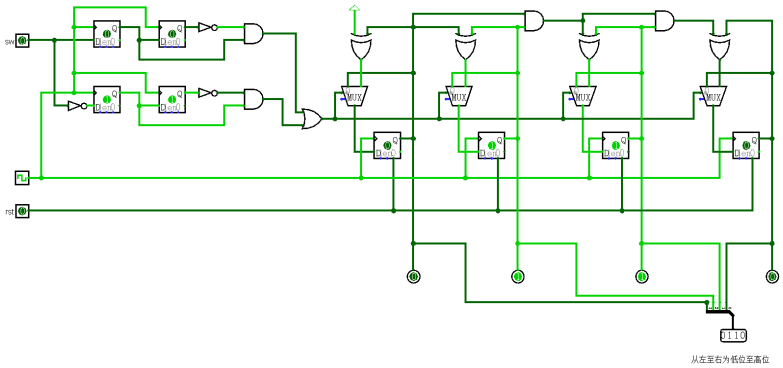
> 题目 1. 在不改变电路功能和行为的前提下， 将前面 Step5 中的代码改写成三段式有限状态机的形式， 写出完整的 Verilog 代码。

代码如下：

```
module t1 (
    input clk,rst,
    output led
);
    reg [1:0] cnt;
    reg [1:0] cntnextstate;
    always@(*)
    begin
        cntnextstate=cnt+2'b1;
    end
    always@(posedge clk or posedge rst)
    begin
        if(rst)cnt<=2'b0;
        else cnt<=cntnextstate;
    end
    assign led=(cnt==2'b11)?1'b1:1'b0;
endmodule
```

> 题目 2. 请在 Logisim 中设计一个 4bit 位宽的计数器电路， 如下图所示， 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。

按照实验文档说明中已经给出了取信号边沿（上升沿为例）得到一个周期的脉冲信号的技巧，对输入信号取反，再应用该技巧，即可得到上升沿和下降沿两个脉冲信号。两个信号或后即可得到状态
转换的输入信号。据此原理，即可得到最终设计的电路图：



> 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”.

verilog 代码：

```
module counter (
    input clk,button,
    input [2:0] sw,
    output reg [3:0] out,
    output reg [2:0] selsct
);
    wire clk_n;
    wire button_n;
    wire button_edge;
    reg enable;
    reg [7:0] result;
    //信号处理
    jtrclr clr(.clk(clk),.button(button),.button_n(button_n));
    signal_edge cedge(.clk(clk),.button(button_n),.button_edge(button_edge
    clk_wiz_0 clk_wiz_0_insrt(.clk_in1(clk),.clk_out1(clk_n),.reset(sw[1])
    //计数模块
```

```
    always@(posedge clk or posedge sw[1])
    begin
        if(sw[1])
        begin
            result<=8'h1f;
        end
        else
        begin
            if(button_edge)
            begin
                if(sw[0]) result<=result+8'b1;
                else result<=result-8'b1;
            end
        end
    end
    //输出信号选择模块
    always@(posedge clk_n or posedge sw[1])
    begin
        if(sw[1]) enable<=1'b0;
        else enable<=enable+1'b1;
    end
    always@(*)
    begin
        case(enable)
        1'b0:begin selsct=3'b0; out=result[3:0]; end
        1'b1:begin selsct=3'b1; out=result[7:4]; end
        default:begin selsct=3'b0; out=result[3:0]; end
        endcase
    end
endmodule



module jtrclr (//去毛刺
    input clk,button,
    output button_n
);
    reg [3:0] cnt;
    always@(posedge clk)
    begin
        if(button==1'b0) cnt<=4'h0;
        else if(cnt<4'h8)cnt<=cnt+4'b1;
    end
    assign button_n=cnt[3];
endmodule

module signal_edge(//取上升沿
    input clk,button,

    output button_edge
```

```
);
    reg button_r1,button_r2;
    always@(posedge clk)
    begin
        button_r1<=button;
    end
    always@(posedge clk)
    begin
        button_r2<=button_r1;
    end
    assign button_edge=button_r1&(~button_r2);
endmodule
```

需要用到的脚管约束文件如下：

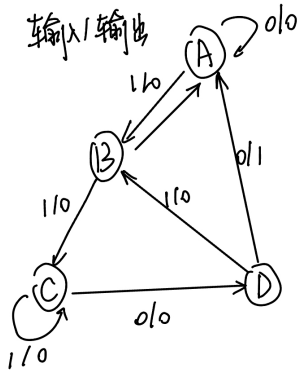
```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports {

set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports {

set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports {
```

> 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，例如当输入“0011001110011”时，目标序列个数应为 2，最近输入数值显“0011”，状态机编码则与具体实现有关。

由题意画出状态图如下 (Moore)：



有限状态机的 verilog 代码：

```
module statemachine(input clk,
            input sw,
            input btn,
            output reg [2:0] hexplay_an,
            output reg [3:0] hexplay_data);
    reg [2:0] curr_state;
    reg [2:0] next_state;

    // 取按钮边沿
    reg button_r1,button_r2;
    always@(posedge clk) button_r1 <= btn;
    always@(posedge clk) button_r2 <= button_r1;
    assign btn_pulse = button_r1 & (~button_r2);

    // 取状态边沿（因为是 Moore 型，需要取边沿来统计计数）
    reg state_r1,state_r2;
    always@(posedge clk) state_r1 <= curr_state[2];
    always@(posedge clk) state_r2 <= state_r1;
    assign state_pulse = state_r1 & (~state_r2);

    // 状态转换
    always@(*) begin
        if (sw)
            case (curr_state)
                3'b000: next_state  = 3'b001;
                3'b001: next_state  = 3'b010;
                3'b010: next_state  = 3'b010;
                3'b011: next_state  = 3'b001;
                3'b100: next_state  = 3'b001;
                default: next_state = 3'b000;
            endcase
        else
            case (curr_state)
                3'b010: next_state  = 3'b011;
                3'b011: next_state  = 3'b100;
                default: next_state = 3'b000;
```

```
            endcase
    end

    // 时序逻辑（检测按钮）
    reg [3:0] recent_num;
    always@(posedge clk) begin
        if (btn_pulse) begin
            curr_state    <= next_state;
            recent_num    <= recent_num << 1;
            recent_num[0] <= sw;
        end
        else begin
            curr_state <= curr_state;
            recent_num <= recent_num;
        end
    end

    // 输出部分，需要输出 6 个数字，分时复用输出
    reg [3:0] success_count;
    always@(posedge clk) begin
        if (state_pulse) success_count <= success_count + 1;
        else success_count             <= success_count;
    end
    reg [4:0] time_count;
    always @(posedge clk) begin
        if (time_count == 5'b10111)
            time_count <= 5'b00000;
        else
            time_count <= time_count + 1;
    end

    always@(posedge clk) begin
        hexplay_an <= time_count[4:2];
        case (time_count[4:2])
            3'b000 : hexplay_data <= {1'b0, curr_state};
            3'b001 : hexplay_data <= success_count;
            3'b010 : hexplay_data <= {3'b000, recent_num[0]};
            3'b011 : hexplay_data <= {3'b000, recent_num[1]};
            3'b100 : hexplay_data <= {3'b000, recent_num[2]};
            3'b101 : hexplay_data <= {3'b000, recent_num[3]};
            default: hexplay_data <= 4'b0000;
        endcase
    end
endmodule
```

脚管约束文件：

```
## This file is a general .xdc for FPGAOL_BOARD (adopted from Nexys4 DDR R
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to th

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports {
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_por
## FPGAOL BUTTON & SOFT_CLOCK
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports {
## FPGAOL LED (signle-digit-SEGPLAY)
# set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports

## FPGAOL SWITCH
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports {
# set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports
# set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports


## FPGAOL HEXPLAY

set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports {
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports {



##USB-RS232 Interface

#set_property -dict { PACKAGE_PIN C4    IOSTANDARD LVCMOS33 } [get_ports {
#set_property -dict { PACKAGE_PIN D4    IOSTANDARD LVCMOS33 } [get_ports {
#set_property -dict { PACKAGE_PIN D3    IOSTANDARD LVCMOS33 } [get_ports {
#set_property -dict { PACKAGE_PIN E5    IOSTANDARD LVCMOS33 } [get_ports {
```

```
#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk];
#set_property CFGBVS VCCO [current_design]
#where value1 is either VCCO or GND
#set_property CONFIG_VOLTAGE 3.3 [current_design]
#where value2 is the voltage provided to configuration bank 0
```

总结与思考

1. 请总结本次实验的收获

   - 掌握了几种常用的数字信号处理技巧。
   - 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。
   - 进一步巩固熟悉了使用 FPGA 进行数字电路设计开发全流程的各关键环节。

2. 请评价本次实验的难易程度

   - 较难，需要学习信号整形及去毛刺，取信号边沿技巧等一系列很难的新知识，脑子要栈溢出了。

3. 请评价本次实验的任务量

   - 接上条，较大。

4. 请为本次实验提供改进建议

   - 实验渐进尾声，建议以后把学分提高（因为 1 学分和这个工作量真的不匹配啊），或者把实验练习与实验手册里的理论知识做分离，否则一周花费时间有点多。仅为个人建议。