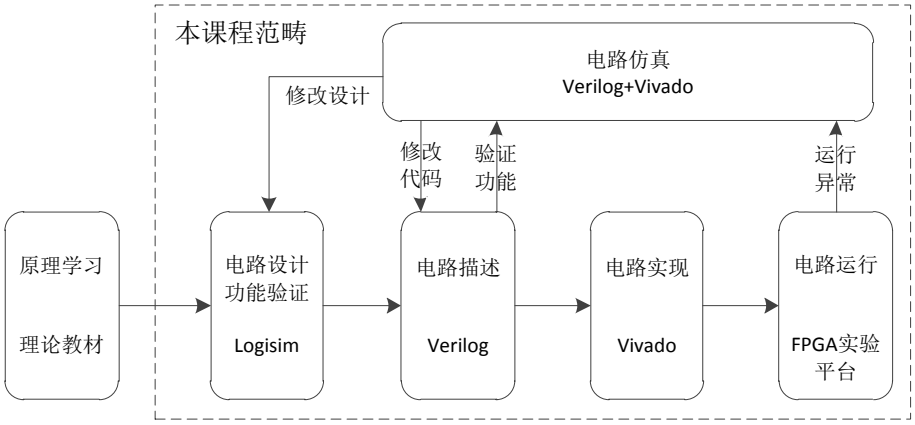
实验 08 信号处理及有限状态机

简介



通过前面实验的学习，我们已经将数字电路实验课程相关的工具、软件、语言、平台等全部介绍完毕，至此读者应当已经基本掌握了使用 FPGA 进行数字电路设计开发全流程的各关键环节。

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

PC 一台

Windows 或 Linux 操作系统

Vivado

FPGA 实验平台（Nexys4 DDR）

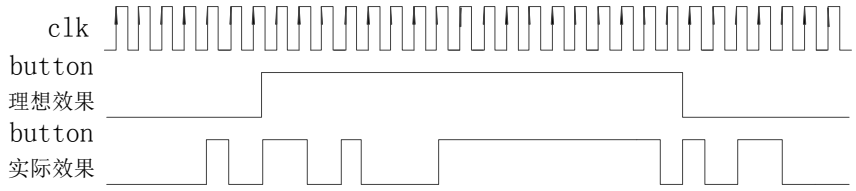
Logisim

vlab.ustc.edu.cn

实验步骤

Step1.信号去抖动

在用 FPGA 开发板的按键或开关作为输入时，由于其机械特性，在电平转换的瞬间会产生一些抖动，这些抖动在用户看来非常短暂，但在 100MHz 的时钟信号下会持续许多个周期。以按键为例，我们希望按键按下后输入信号会直接从 0 变为 1，按键松开时会直接从 1 变为 0，但实际情况却并非这样，如下图所示。

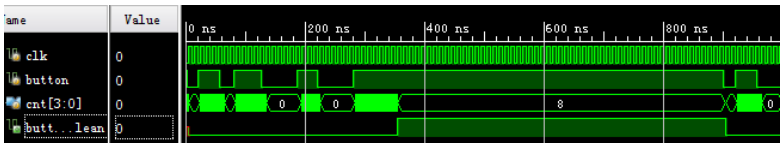


这种结果并不是我们希望看到的，我们希望按键按下后不会出现抖动现象。遗憾的是，对于这种机械式的按键来说，抖动现象很难完全避免，因此我们需要借助额外的去抖动电路来达到消除抖动的目的。

消除抖动的关键在于区分有效的按键输入和按键按下或抬起瞬间的机械抖动。以人的反应速度来看，按键按下再抬起，最快也在毫秒以上量级，而抖动一般都在微秒甚至纳秒量级，因此我们可以通过信号电平持续时间的长短来判定是否为一次有效按键。通过一个计数器对高电平持续时间进行计时，当按键输入信号为 0 时，计数器清零，当输入信号为高电平时，计数器进行累加计数，计数达到阈值后则停止计数，如下面代码所示：

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

通过仿真我们发现上述电路能够将持续时间少于 8 个时钟周期的抖动信号全部滤除，输出变成了一个较为干净的电平信号。我们还可以通过调节计数器的阈值改变该消除抖动电路的精度。



通过去抖动电路，我们能够将信号电平转换过程中的“毛刺”消除掉，使得输入信号变得更加干净、可靠。

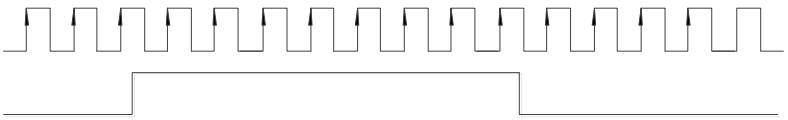
Step2.取信号边沿技巧

很多电子设备都是用按键或开关等外设控制电路状态的跳转，每当按键按下一次，电路切换一种状态，就像是信号的边沿触发电路一样，但我们知道，除了时钟和异步复位信号外，其它信号都不应放在边沿敏感列表内，那到底应到如何实现呢？一般的做法是通过下面的代码来实现状态跳转

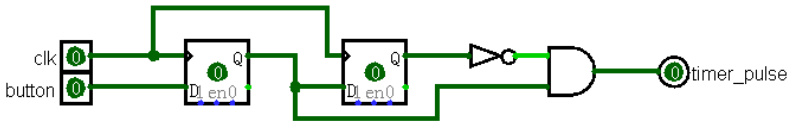
```
always@(posedge clk)
    if(控制信号==1)
        //状态跳转
```

```
else
    //状态保持
```

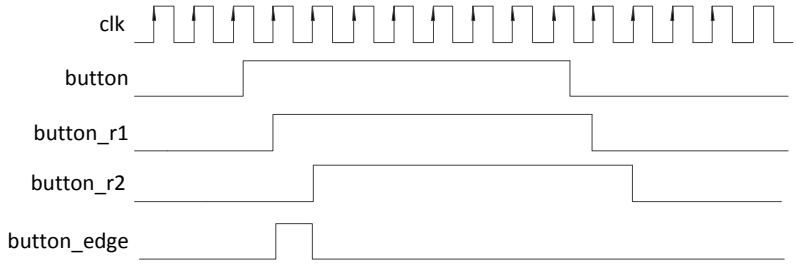
上述代码能够实现电路状态转换的功能，但是要求该信号为持续时间为一个时钟周期的脉冲信号，否则电路状态会一直跳转。在时钟频率较高的情况下，要求用户通过按键、开关这类外设输入这种高精度的信号显然是不现实的，因为用户的反应时间相对于 100MHz 的时钟来说太慢了，一次按键可能会持续许多甚至上千万个周期，如下图所示：



下面我们介绍如何通过该按键信号生成一个时钟周期宽度的脉冲信号。



通过两个寄存器对输入信号进行寄存，寄存器后的信号分别为 button_r1,button_r2，然后将 button_r2 取反并和 button_r1 进行与操作，便得到了一个时钟周期宽度的脉冲信号，该信号在 button 信号的上升沿附近为高电平，其余时间均为低电平。



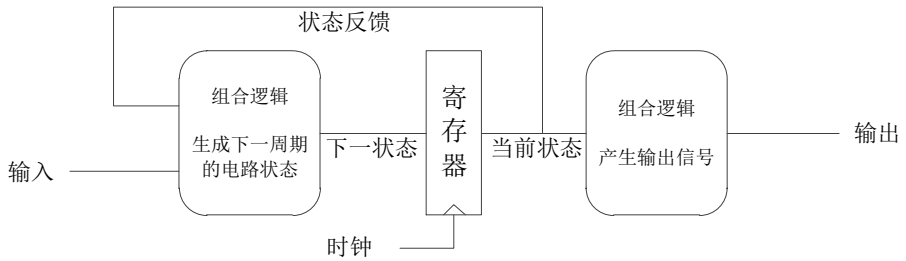
其 Verilog 代码如下：

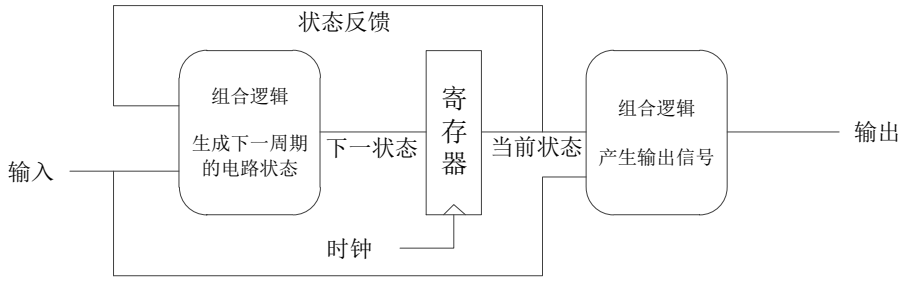
```
module signal_edge(
input clk,
```

```
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
```

Step3. 有限状态机介绍

一般来说，数字电路由组合逻辑和时序逻辑构成。典型的数字电路都可以归结为以下两种基本结构，即便是较为复杂的电路也可以对其进行细化，最终肯定可以对应到其中一种结构上。



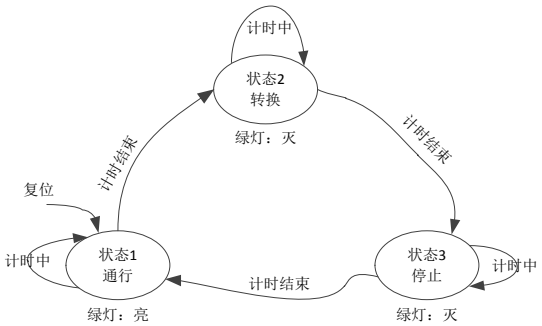


设寄存器位宽为 n，则该电路的状态数量不会超过 2^n，即其状态数量是有限的，因此这种电路结构称为有限状态机。

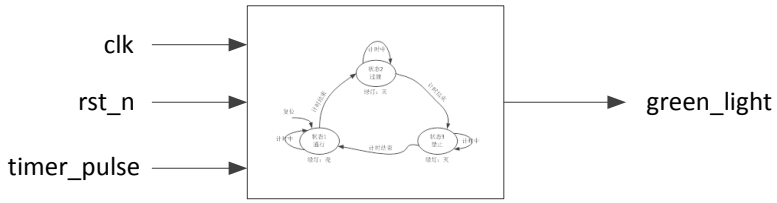
有限状态机可分为两种。第一种输出信号只与当前状态有关，输入信号不会直接影响到输出信号，而是与当前状态（简称现态）信号一起生成下一状态（简称次态）信号，在时钟的上升沿之后次态转换为现态，才能影响到输出，这种结构称为摩尔型（Moore）有限状态

机；另外一种其输出信号由现态与输入信号共同生成，输入信号可立刻对输出信号产生影响，这种结构称为米莉（Mealy）型有限状态机。

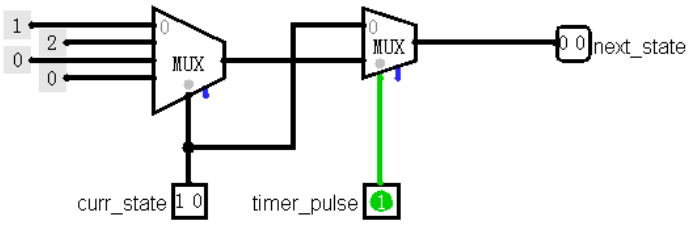
两种结构的有限状态机各有优缺点：Moore 型时序更好，但响应要慢一拍，Mealy 型响应最快，但时序上要差一些。一般来说，如果对电路相应速度要求不是非常苛刻的话，推荐使用 Moore 型有限状态机。我们以交通灯为例来讲解有限状态机的工作原理。



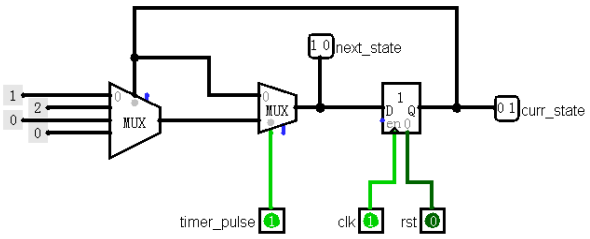
交通灯有红黄绿三种颜色，分别代表停止、转换、通行三种状态，交通灯通过计时信号控制状态在这三者之间不断转换。计时结束前，将保持原状态，计时信号出现后，会跳转到下一状态，如从“通行”状态跳转到“转换”状态。绿灯在“通行”状态下灯亮，另外两个状态下灯灭。如果用电路实现该交通灯的功能，其输入信号为计时信号（timer_pulse），该信号为持续一个时钟周期的脉冲信号，输出信号为绿灯 (green_light)。电路包含 3 种状态(PASS、TRANS、STOP，分别用 2'b00,2'b01,2'b10 表示)，复位状态为“PASS”。电路端口为：



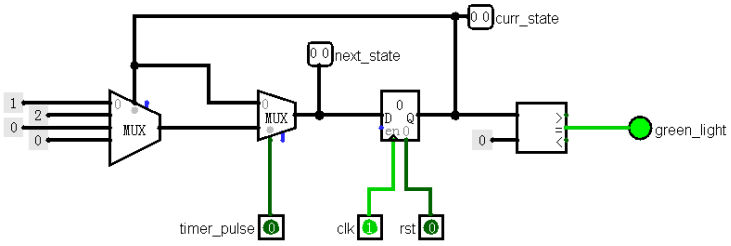
次态生成逻辑如下图所示



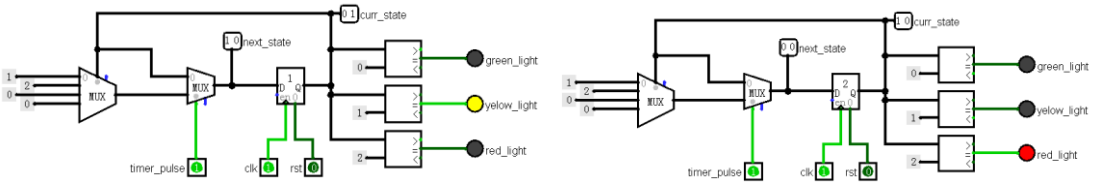
在上述电路的基础上，添加寄存器，在每个时钟的上升沿将 next_state 信号赋值给 curr_state 信号。



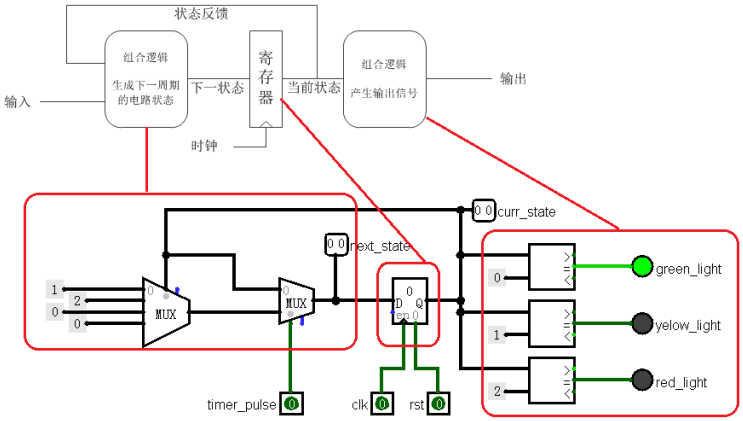
最后，我们通过 curr_state 信号控制绿色交通灯的亮灭



实际上，通过同样的方式还可以控制其它输出信号，如“转换”状态时黄灯亮，“停止”状态时红灯亮。



至此，我们完成了用有限状态机实现交通灯控制的电路设计。该电路是一个典型的摩尔型有限状态机，电路各部分与有限状态机结构对应关系如下图所示。



Step4. 有限状态机 Verilog 实现

通过分析有限状态机的结构图，我们可以发现其包含三个部分，第一部分为纯组合逻辑，通过现态和输入信号生成次态信号，第二部分为时序逻辑，该时序逻辑非常简单，只包含一个带有复位功能的寄存器单元，复位时现态信号变为初始值，否则在每个时钟的上升沿将次态信号赋值给现态信号。第三部分为组合逻辑，该部分通过现态信号生成各输出信号。对于上述电路，其 Verilog 代码实现为：

```
module traffic_ctrl(
input   clk,
input   rst,
input   timer_pulse,
output  green_light
);
parameter   C_PASS  = 2'b00;
parameter   C_TRANS = 2'b01;
parameter   C_STOP  = 2'b10;
reg [1:0]   curr_state;
reg [1:0]   next_state;
//有限状态机第一部分
always@(*)
begin
    if(timer_pulse)
    begin
        case(curr_state)
            C_PASS: next_state = C_TRANS;
            C_TRANS:next_state = C_STOP;
```

```
            C_STOP: next_state = C_PASS;
            default:next_state = C_PASS;
        endcase
    end
    else
        next_state = curr_state;
end
//有限状态机第二部分
always@(posedge clk or posedge rst)
begin
    if(rst)
        curr_state  <= C_PASS;
    else
        curr_state  <= next_state;
end
//有限状态机第三部分,各输出信号的赋值都应放在此部分
assign green_light = (curr_state==C_PASS)? 1'b1 : 1'b0;
//...
endmodule
```

通过阅读代码可以发现，代码主体可划分为三段，分别对应有限状态机结构图中的三个部分，这种代码结构称为有限状态机的三段式写法。除此之外，还有两段式和一段式写法，从功能上来说都可以正常工作，但代码层次结构和可读性要比三段式写法差一些，因此我们推荐使用三段式写法。对两段式和一段式写法感兴趣的读者可自行调研，此处不再介绍。此外，建议读者将代码与前面的电路图对照学习，以加深对有限状态机的理解。

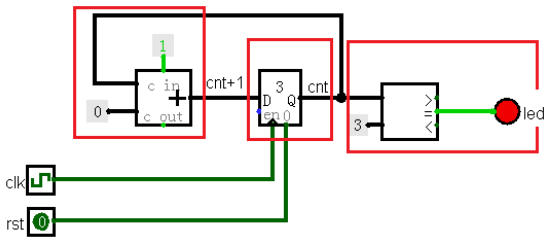
Step5. 有限状态机的另一形式

虽然前面的几次实验没有介绍有限状态机，但实际上我们已经在不自觉中多次用到，请看下面的代码

```
module test(
input  clk,rst,
output led);
reg [1:0] cnt;
always@(posedge clk or posedge rst_n)
```

```
begin
    if(rst)
        cnt <= 2'b0;
    else
        cnt <= cnt + 1'b1;
end
assign led = (cnt==2'b11) ? 1'b1 : 1'b0;
endmodule
```

此段代码的电路图如下所示



可以看出，该电路包含了 4 个状态（0,1，2，3），电路在这 4 个状态之间循环跳转，其中加法器用于生成次态（cnt+1），寄存器将次态信号传递给现态（cnt），最后的比较器通过现态信号生成输出信号（led）。它虽然没有按着有限状态机的三段式格式来写，但确确实实是一个摩尔型有限状态机。

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。

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。

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，通过按键控制计数模式，一个按键控制累加，每按键一次计数器加一，另一按键控制递减，每按下一次计数器减一，按键按下的瞬间触发操作，一个按键按下未抬起，不会影响到另一按键触发计数器。计数值用数码管显示，其复位值为“1F”。

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，用开关表示待输入的数值，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，最近输入数值显示“0011”，状态机编码则与具体实现有关。

总结与思考

1. 请总结本次实验的收获

2. 请评价本次实验的难易程度

3. 请评价本次实验的任务量

4. 请为本次实验提供改进建议