实验六：有限状态机

实验时间： 2017年12月4日 第十三周 星期一
实验者：16307130194 陈中钰 16级 计算机科学技术学院
座位号：30
指导老师：唐志强

1 实验目的

- 了解并掌握有限状态机的概念和设计
- 设计交通灯控制器，实现东南西北四个方向的红绿灯正常交替变化
- 加深对 BCD 7段显示译码器的理解，了解其实际应用
- 了解并掌握扫描显示技术，清楚其作用及优点
- 了解并设计显示时分的数字钟

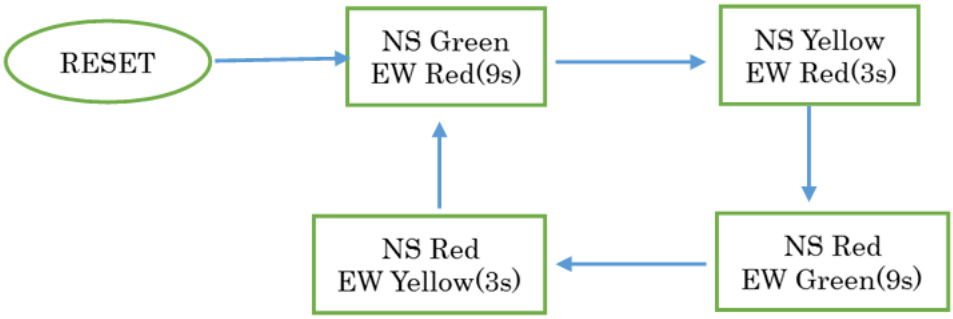
2 实验原理

2.1 交通灯控制器 1

- 使用3个 LED 灯分别代表一个路口的红、黄、绿灯，每种情况下仅只有一个灯亮（如：亮第一个灯代表红灯），四个路口用4组 LED 灯，共12个 LED 灯

C	B	A	状态	持续时间
0	0	1	Green	9s
0	1	0	Yellow	3s
1	0	0	Red	12s

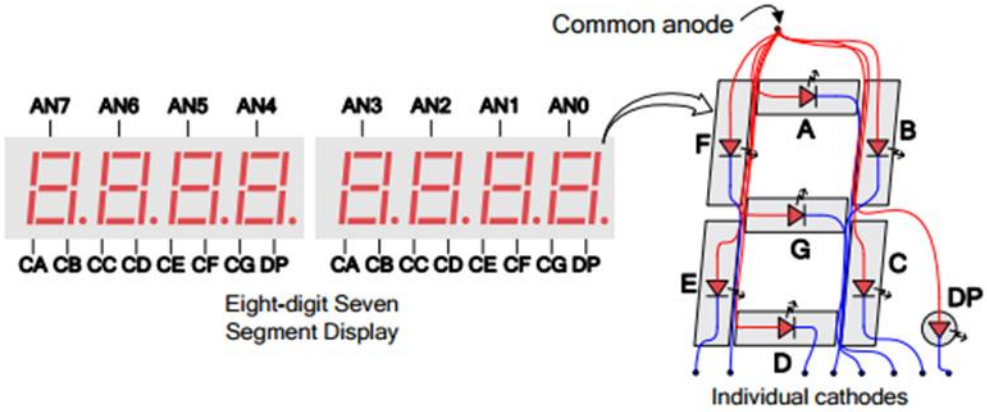
- 南北和东西向的在同一时间亮同样的灯，也就是状态相同
- 每个路口的灯的变化次序为：绿、黄、红，并循环变化，其中绿灯持续约9s，黄灯持续约3s，红灯持续约12s，可以通过使用一个6位二进制计数器进行计时，每个时钟周期增加1，当值为0~9为状态一，9~12为状态二，12~21为状态三，21~24为状态四，在状态交接处转换状态并正常计数，其他时候只计数，状态不变，而且在值为24时还同时把计数器置0
- 通过异步 RESET 开关可以设置状态为“NS 方向为绿灯”的开始
- 通过以上要求，可以得到状态图



2.2 显示时分的数字钟

- BCD 7段显示模块的 CA~CG、AN 都是0（低）有效的
- 显示时、分，时间范围00:00~23:59，每个数字分别用一个 BCD 码表示，也就是用1个 BCD 7段显示译码器（和实验二中的一致）来显示，一共4*4=16位二进制

- 输入时钟约1Hz，通过时钟分频获得
- 由于 board 上的8个 BCD 7段显示器是共用 CA~CG 引脚的，故需要采用扫描显示的方式进行多位数字的显示
- 扫描显示：利用人的0.1s~0.4s 视觉残留，每次只显示一位数字，并在一个周期里轮流显示4个数字，频率控制合适，能同时、稳定看到4位数字，在本次实验中采用约3kHz 的频率（如果频率过高，数字交替过快，每个数字显示的时间过短，视觉感官来不及感受灯光，而只能看见部分数字，而且在快速闪烁，不稳定，或甚至全部数字都看不到；如果频率过低，数字交替过慢，不足以形成视觉残留，会看见数字轮流点亮，而不是“同时亮”）
- 扫描显示实现：独立计算数字钟的四个数字，轮流给 CA~CG 赋值，并在对应时段使 AN 中该数字对应位置值为0，点亮该数字对应位置的7段显示器
- 数字计算的实现：使用约1Hz 的时钟，每过一个周期使时钟计数器增加1.但是与一般计数器不同的是，分的低位、时的低位为10进制，分的高位为6进制，时的高位只有0、1、2，因此要注意对进位情况的判断及赋值
- 轮流赋值的实现：使用约3kHz 的时钟，使用2位计数器的原理，使2位计数器从00~11不断循环计数，在对应时刻给 CA~CG 赋上对应数字的值，并使对应 AN 对应位置为0，其他 AN 为1，那么可以在这一个时钟周期在相应位置显示出数字钟的1位，在合适的频率下不断轮流显示后，便能“同时”看到4位数字显示
- 器件图



3 实验分析

3.1 交通灯控制器 1

- 通过计数器计数，用 case()语句判断是哪一种状态转换，以及是否要转换状态或只是单纯计数
- 通过 RESET 可以异步设置为初始状态，应该把该控制开关纳入 always@()的敏感表中，并可以通过 if 语句控制，而且该异步操作优先级最高
- 时钟分频取用的是 q[26]，约为 1Hz

3.2 显示时分的数字钟

- 使用到了实验二中的 BCD 7段显示译码器，用于把数字钟的4个数字显示出来，而由于真值表相同，此处不再展示
- 由于时间计数的特殊性，时间需要用特殊的计数器进行计数，每位数字用4位 BCD 码计数，并设置特殊的进位机制

优先级	当前时间	下一时间
high	23:59	00:00
	X9:59	(X+1)0:00
	XX:59	X(X+1):00
	XX:X9	XX:(X+1)0
low	XX:XX	XX:X(X+1)

- 添加异步置0功能（优先级最高）：当 reset 开关为1时，把4位 BCD 数字均置0000，且 reset 也在计数器部分的 always@()的敏感表中，且优先级最高
- 添加异步置数功能（优先级仅次于异步置0）：使用 myset 开关控制是否加载输入的时间，用 XX:XX <- sw[13:12] sw[11:8]:sw[7:4] sw[3:0]代表时间，由于时的高位只有0、1、2三个数，故只需要2个开关即可，一共需要2+4+4+4=14个开关（同时可以剩下2个开关，一个作为重置开关 reset，另一个作为置数开关 myset），而为了使这个数字的7段显示也能套用 BCD 7段显示译码器的模块，可以把数字输入向量补为{0, 0, sw[13], sw[12]}，而另外的三个数字则可以正常使用 BCD 段显示模块，当 myset 为1时，会把 sw[13:0]输入的时间加载到 BCD 7段显示译码器的显示上，并且从该时间开始继续计数
- 添加 DP 闪烁：设置时钟频率也是约为1Hz，在每次时钟上升沿来临时，对 DP 的值取反，那么就能看到闪烁的效果了，每次亮的时间约为1s
- 控制开关真值表

Reset	myset	功能
0	0	正常计数
0	1	异步置数
1	X	异步置0

4 实验内容

4.1 交通灯控制器 1

4.1.1 Verilog 代码

```
module traffic_lights_1(input clk, input reset, output reg [2:0] N,
output reg [2:0] S, output reg [2:0] E, output reg [2:0] W);
    wire myclk;
    reg [5:0] counter;
    clkdiv c1(clk,myclk);
    always@(posedge myclk, posedge reset)
    begin
    if(reset)
    begin N<=3'b001; S<=3'b001; E<=3'b100; W<=3'b100; end
    else
    begin
        case(counter)
        6'b001001:
    begin N<=3'b010; S<=3'b010; E<=3'b100; W<=3'b100;
          counter<=counter+1; end
        6'b001100:
    begin N<=3'b100; S<=3'b100; E<=3'b001; W<=3'b001;
```

```
          counter<=counter+1; end
         6'b010101:
    begin N<=3'b100; S<=3'b100; E<=3'b010; W<=3'b010;
          counter<=counter+1; end
         6'b011000:
    begin N<=3'b001; S<=3'b001; E<=3'b100; W<=3'b100;
          counter<=6'b000000; end
        default
    begin N<=N; S<=S; E<=E; W<=W; counter<=counter+1; end
        endcase
    end
    end
endmodule

module clkdiv(input inclk, output outclk);
    reg [35:0] q;
    always@(posedge inclk)
         q<=q+1;
    assign outclk=q[26];//around 0.75Hz
endmodule
```

4.1.2 操作

- 当 reset 为0时，LED 按照状态图中状态不断循环显示
- 当 reset 设为1时，LED 马上变为 NS 为 Green，EW 为 Red 的状态开始

4.1.3 改进

- 计数器最大计数为24（11000），故只需要5位，而不需要6位
- 使用 parameter 定义3种不同的状态，每种状态代表一种灯（如：设置 parameter yellow=3'b010，代表黄灯），可以避免直接的向量赋值，而且更加直观易懂
- 设置 state 指示4种状态（0，1，2，3），通过计数器设置 state，再通过 state 设置状态转换和显示，这样虽然在计数器和显示之间插入了 state，但是会使逻辑结构更清晰，而且可以把计数部分与状态转换部分的代码分别分开到2个 always@() 语句中，功能更独立明确，一个负责计数，另一个负责状态转换，代码会更规整而简洁易懂

4.1.4 改进结果（省略 clkdiv 模块）

```
module traffic_lights_1(input clk, input reset, output reg [2:0] N,
output reg [2:0] S, output reg [2:0] E, output reg [2:0] W);
    wire myclk;
    reg [5:0] counter; reg [2:0] state;
    clkdiv c1(clk,myclk);
    parameter yellow=3'b010; parameter green=3'b001; parameter
red=3'b100;
    always@(posedge myclk, posedge reset)
    begin
        if(reset)begin counter<=5'b00000;state<=0;end
```

```
        else
        begin counter<=counter+1;
            if(counter==5'b01001) state<=1;
            else if(counter==5'b01100) state<=2;
            else if(counter==5'b10101) state<=3;
            else if(counter==5'b11000) begin
                  state<=0;counter<=5'b00000;end
        end
    end
    always@(state)
    case(state)
    0: begin N<=green; S<=green; E<=red; W<=red; end
    1: begin N<=yellow; S<=yellow; E<=red; W<=red; end
    2: begin N<=red; S<=red; E<=green; W<=green; end
    3: begin N<=red; S<=red; E<=yellow; W<=yellow; end
    endcase
endmodule
```

4.2 显示时分的数字钟

4.2.1 设计

- 使用 BCD 7段显示译码器模块，显示4位数字
- 用分频后约1Hz 的时钟，在该4位数字上实行时钟式计数，可以通过 if else 语句实现
- 扫描显示单个数字，使用分频后约3kHz 的时钟
- 一般来说，数字钟的时和分之间会有一个冒号在闪烁，因此原本想设置中间的冒号进行闪烁。但是引脚文件中没有找到，只找到数字右下角的 DP 点，所以最后改为设置 DP 点在闪烁，使用约为1Hz 的时钟

4.2.2 Verilog 代码

```
module digital_clock(input clk, input [13:0] swt, input reset, input
myset, output reg [6:0] C, output reg DP, output reg [7:0] AN);
    wire [6:0] C1; wire [6:0] C2; wire [6:0] C3; wire [6:0] C4;
    reg [3:0] T1; reg [3:0] T2; reg [3:0] T3; reg [3:0] T4; reg [1:0]
cnt;
    wire myclk1,myclk2;
    clkdiv1 c1(clk,myclk1);//time
    clkdiv2 c2(clk,myclk2);//display
    always@(posedge myclk1)//beep
        DP<=~DP;
    always@(posedge myclk1, posedge reset, posedge myset)//set time
    begin
    if(reset)
    begin T4<=4'b0000; T3<=4'b0000; T2<=4'b0000; T1<=4'b0000; end
    else if(myset)
    begin T4<={0,0,swt[13],swt[12]}; T3<=swt[11:8]; T2<=swt[7:4];
```

```
T1<=swt[3:0];
    end
    else if(T4==4'b0010&T3==4'b0011&T2==4'b0101&T1==4'b1001)//23:59
    begin T4<=4'b0000; T3<=4'b0000; T2<=4'b0000; T1<=4'b0000; end
    else if(T3==4'b1001&T2==4'b0101&T1==4'b1001)//x9:59
    begin T4<=T4+1; T3<=4'b0000; T2<=4'b0000; T1<=4'b0000; end
    else if(T2==4'b0101&T1==4'b1001)//xx:59
    begin T4<=T4; T3<=T3+1; T2<=4'b0000; T1<=4'b0000; end
    else if(T1==4'b1001)//xx:x9
    begin T4<=T4; T3<=T3; T2<=T2+1; T1<=4'b0000; end
    else//xx:xx
    begin T4<=T4; T3<=T3; T2<=T2; T1<=T1+1; end
    end
    BCD_7_segment_display b1(T1[3:0],C1[6:0]);
    BCD_7_segment_display b2(T2[3:0],C2[6:0]);
    BCD_7_segment_display b3(T3[3:0],C3[6:0]);
    BCD_7_segment_display b4(T4[3:0],C4[6:0]);
    always@(posedge myclk2)
    begin
        case(cnt)
        2'b00: begin AN<=8'b11111110; C<=C1; cnt<=cnt+1; end
        2'b01: begin AN<=8'b11111101; C<=C2; cnt<=cnt+1; end
        2'b10: begin AN<=8'b11111011; C<=C3; cnt<=cnt+1; end
        2'b11: begin AN<=8'b11110111; C<=C4; cnt<=cnt+1; end
        endcase
    end
endmodule

module BCD_7_segment_display(input [3:0] swt, output reg [6:0] C);
    always@(swt)
    begin
    case(swt)
    4'b0000:C=7'b0000001;
    4'b0001:C=7'b1001111;
    4'b0010:C=7'b0010010;
    4'b0011:C=7'b0000110;
    4'b0100:C=7'b1001100;
    4'b0101:C=7'b0100100;
    4'b0110:C=7'b0100000;
    4'b0111:C=7'b0001111;
    4'b1000:C=7'b0000000;
    4'b1001:C=7'b0000100;
    default:C=7'b0000001;//consider it 0
    endcase
```

```
    end
endmodule

module clkdiv1(input inclk, output outclk);
    reg [35:0]q;
    always@(posedge inclk)
         q<=q+1;
    assign outclk=q[26];//around 0.75Hz
endmodule

module clkdiv2(input inclk,output outclk);
    reg [35:0]q;
    always@(posedge inclk)
         q<=q+1;
    assign outclk=q[16];//around 3kHz
endmodule
```

4.2.3 操作

- 当 myset，reset 均为0时，数字钟从00:00~23:59，正常变化，并跳回00:00
- 当 reset 为1时，数字钟马上变回00:00，并从这个时间开始计时
- 当 reset 为0，myset 为1时，数字钟设计为 sw[13:0]的输入时间，并从这个时间开始计时

4.2.4 改进

- 由于该时钟有异步置数的功能，因此输入的数字可能非法。BCD 7段显示译码器模块中已经有一个安全机制，当时的低位和分的低位输入值超过9时，会设置为0。但是，时的高位为3，或者分的高位在5~9，此时非法，但是没有安全设置，应该补全这个安全机制，使得非法时间输入统一改为某个合法时间输入，以避免时钟异步跳进一个非法时间进行计时，而导致之后的奇怪显示

	时的高位	时的低位	分的高位	分的低位
开关输入	>2	>9	>5	>9
安全设置	0	0	0	0

5 实验结论

- reset 为0时，交通信号等能正常按照状态图变化；reset 为1时重置
- 数字钟能正常计时；reset 为1时重置，myset 为1且 reset 不为1时，置数

6 实验感想

- 针对有限状态机，可以使用 parameter 对状态进行定义，并加入 state，next_state 等设置，会简化 case()或 if else 判断过程，使得代码更简洁明了，而不必要像非时序电路那样单纯依靠判断、赋值来进行设计，反而会更繁琐
- 有限状态机在交通信号灯、自动售卖机等均有所应用，因此了解并掌握有限状态机的设计及应用是十分重要的，也会是将来进一步学习时序电路的重要基础