实验名称：实验八 状态机及键盘输入

姓名：张涵之

学号：191220154

班级：周一 5-6

邮箱：191220154@smail.nju.edu.cn

实验时间：2020/11/7

8.1.2 简单状态机

请查阅相关资料，研究米里型状态机的设计与摩尔型有何不同？

在米里型状态机中，输出信号由输入信号和存储电路的状态共同决定；在摩尔状态机中，输入信号只能影响存储电路的状态，输出信号只由存储电路的状态直接决定。

设计实例中也能看出，输出信号 out 是直接由状态寄存器译码得到的，与存储状态 state 有关而与输入信号 in 无关，输入信号 in 改变的是存储状态 state 即将成为的下一个状态。

8.4 实验内容

自行设计状态机，实现单个按键的 ASCII 码显示。

基本要求：七段数码管低两位显示当前按键的键码，中间两位显示对应的 ASCII 码（转换可以考虑自行设计一个 ROM 并初始化）。只需完成字符和数字键的输入，不需要实现组合键和小键盘。当按键松开时，七段数码管的低四位全灭。七段数码管的高两位显示按键的总次数。按住不放只算一次按键。只考虑顺序按下和放开的情况，不考虑同时按多个键。

实验目的：设计状态机实现单个按键的 ASCII 码显示。

程序代码或流程图：

```
module ps2_keyboard(clk,clrn,ps2_clk,ps2_data,data,ready,nextdata_n,overflow);

    input clk,clrn,ps2_clk,ps2_data;
    input nextdata_n;
    output [7:0] data;
    output reg ready;
    output reg overflow; // fifo overflow
    // internal signal, for test
    reg [9:0] buffer; // ps2_data bits
    reg [7:0] fifo[7:0]; // data fifo
    reg [2:0] w_ptr,r_ptr; // fifo write and read pointers
    reg [3:0] count; // count ps2_data bits
    // detect falling edge of ps2_clk
    reg [2:0] ps2_clk_sync;

    always @(posedge clk) begin
        ps2_clk_sync <= {ps2_clk_sync[1:0],ps2_clk};
    end

    wire sampling = ps2_clk_sync[2] & ~ps2_clk_sync[1];

    always @(posedge clk) begin
        if (clrn == 0) begin // reset
            count <= 0; w_ptr <= 0; r_ptr <= 0; overflow <= 0; ready<= 0;
        end
        else begin
            if (ready) begin // read to output next data
                if(nextdata_n == 1'b0) begin //read next data
                    r_ptr <= r_ptr + 3'b1;
                    if(w_ptr==(r_ptr+1'b1)) //empty
                        ready <= 1'b0;
                end
            end
            if (sampling) begin
                if (count == 4'd10) begin
                    if ((buffer[0] == 0) && // start bit
                        (ps2_data) && // stop bit
                        (^buffer[9:1])) begin // odd parity
                        fifo[w_ptr] <= buffer[8:1]; // kbd scan code
                        w_ptr <= w_ptr+3'b1;
                        ready <= 1'b1;
                        overflow <= overflow | (r_ptr == (w_ptr + 3'b1));
                    end
                    count <= 0; // for next
                end
                else begin
                    buffer[count] <= ps2_data; // store ps2_data
                    count <= count + 3'b1;
                end
            end
        end
    end

    assign data = fifo[r_ptr]; //always set output data

endmodule
```

//键盘控制器模块，用于从键盘获取输入（键码）。

```
module ascii(scancode,caps,ascii);

   input [7:0] scancode;
   input caps;
   output reg [7:0] ascii;
   reg [7:0] code [255:0];

   initial begin
      $readmemh("code2ascii.txt", code, 0, 255);
   end

   always @ (*) begin
      if (caps && (code[scancode] >= 97 && code[scancode] <= 122))
         ascii <= code[scancode] - 32;
      else ascii <= code[scancode];
   end

endmodule
```

//转译模块，从文件读入构建存储器，用于通过键码获得对应的 ascii 码。

```
module hex(in,en,out);

   input [3:0] in;
   input en;
   output reg [6:0] out;

   always @ (*) begin
      if (en) begin
         case (in)
            0: out = 7'b1000000;
            1: out = 7'b1111001;
            2: out = 7'b0100100;
            3: out = 7'b0110000;
            4: out = 7'b0011001;
            5: out = 7'b0010010;
            6: out = 7'b0000010;
            7: out = 7'b1111000;
            8: out = 7'b0000000;
            9: out = 7'b0010000;
            10: out = 7'b0001000;
            11: out = 7'b0000011;
            12: out = 7'b1000110;
            13: out = 7'b0100001;
            14: out = 7'b0000110;
            15: out = 7'b0001110;
            default: out = 7'b1111111;
         endcase
      end
      else out = 7'b1111111;
   end
endmodule
```

//显示模块，将键码、ascii 码、按键次数在七段数码管上显示。

```
module clk_1s(clk,clk_1s);

   input clk;
   output reg clk_1s = 0;
   reg [6:0] count_clk = 0;

   always @ (posedge clk) begin
      if(count_clk == 50) begin
         clk_1s <= ~clk_1s;
         count_clk <= 0;
      end
      else count_clk <= count_clk + 1;
   end

endmodule
```

//时钟模块，得到频率合适的时钟信号用于主模块刷新。

```
   reg [7:0] scancode;
   reg [7:0] times = 0;
   reg caps = 0;
   reg nextdata = 0;
   reg en = 1;
   reg flag = 1;
   reg clk_1s = 0;
   wire [7:0] code,ascii;
   wire clr,ready,overflow;
   assign clr = SW[0];
```

//scancode 用于暂存从键盘获得的键码，times 用于计数
//caps 用于判断 CapsLock 键是否按下
//nextdata 即键盘控制器中的 nextdata_n，在顶层模块中更新。
//en 为七段数码管是否显示的使能控制端，在顶层模块中判断更改。
//flag 为调节 en 的控制信号，在按键松开（接受 0xf0 后一个键值）时 en 置零。
//clk_1s 为顶层模块刷新频率的时钟信号。
//clr 为清零信号，由开关 SW[0]控制。
//ready 和 overflow 即键盘控制器中的 ready 和 overflow。

```
    ps2_keyboard p(CLOCK_50,clr,PS2_CLK,PS2_DAT,code,ready,nextdata,overflow);
    ascii r(scancode,caps,ascii);
    hex h0(scancode[3:0],en,HEX0);
    hex h1(scancode[7:4],en,HEX1);
    hex h2(ascii[3:0],en,HEX2);
    hex h3(ascii[7:4],en,HEX3);
    hex h4(times[3:0],1'b1,HEX4);
    hex h5(times[7:4],1'b1,HEX5);
    clk_1s c(CLOCK_50,clk_1s);

//=======================================================
//  Structural coding
//=======================================================

    always @ (posedge clk_1s) begin
        if(ready && nextdata) begin
            nextdata <= 0;
            scancode <= code;
            if(code == 8'hf0) begin
                times <= times + 1;
                flag <= 1;
            end
            else begin
                if(flag) begin
                    en <= 0;
                    flag <= 0;
                    if(code == 8'h58)
                        caps = ~caps;
                end
                else en <= 1;
            end
        end
        else nextdata <= 1;
    end
```

//顶层综合各个模块，判断键盘的按键/松开，进行按键的计数和显示。

实验环境/器材：实验箱一个，笔记本电脑一台，键盘一个。

实验原理/步骤/过程：
仿照课件设计，完成键盘控制器模块 ps2_keyboard；
将键码与 ascii 码的对应写在文本文件中，仿照实验七存储器设计完成键码转译模块 ascii；
　　*此处考虑配合 CapsLock 键进行字母的大小写判断。
完成处理时钟信号的模块 clk_1s 和将数据转化成七段数码管输入的模块 hex；
分别写测试代码测试上述几个模块，确保功能能够正确实现。
在顶层模块 exp8 中调用上述模块，实现接受键盘输入、译码、按键计数、输出等功能。

测试方法：按下键盘，在开发板上观察数码管输出。
实验结果：通过观察，开发板上按键的输出均符合预期。
实验中遇到的问题及解决办法：感到提供的 ps2_keyboard 代码十分难懂。
　　解决办法：多读几遍，思考每个变量的含义和作用，在顶层模块中该如何调取。
实验得到的启示：无。
意见和建议：无。