实验八

状态机及键盘输入

实验报告　　日期：2020 年 11 月 6 日

姓名： 朱嘉琦

学号： 191220185

班级： 数电实验一班

邮箱： 1477194584@qq.com

实验八报告——状态机及键盘输入

191220185 朱嘉琦

一、实验目的

有限状态机 FSM（Finite State Machine）简称状态机，是一个在有限个状态间进行转换和动作的计算模型。有限状态机含有一个起始状态、一个输入列表（列表中包含了所有可能的输入信号序列）、一个状态转移函数和一个输出端，状态机在工作时由状态转移函数根据当前状态和输入信号确定下一个状态和输出。状态机一般都从起始状态开始，根据输入信号由状态转移函数决定状态机的下一个状态。

本实验的目的是学习状态机的工作原理，了解状态机的编码方式，并利用 PS/2 键盘输入实现简单状态机的设计。

二、实验原理

有限状态机

在实际应用中，有限状态机被分为两种：Moore（摩尔）型有限状态机和 Mealy（米里）型有限状态机。Moore 型有限状态机的输出信号只与有限状态机的当前状态有关，与输入信号的当前值无关，输入信号的当前值只会影响到状态机的次态，不会影响状态机当前的输出。即 Moore 型有限状态机的输出信号是直接由状态寄存器译码得到。Moore 型有限状态机在时钟 CLK 信号有效后 经过一段时间的延迟，输出达到稳定值。即使在这个时钟周期内输入信号发生变化，输出也会在这个完整的时钟周期内保持稳定值而不变。输入对输出的影响要到下一个时钟周期才能反映出来。Moore 有限状态机最重要的特点就是将输入与输出信号隔离开来。Mealy 状态机与 Moore 有限状态机不同，Mealy 有限状态机的输出不仅仅与状态机的当前状态有关，而且与输入信号的当前值也有关。Mealy 有限状态机的输出直接受输入信号的当前值影响，而输入信号可能在一个时钟周期内任意时刻变化，这使得 Mealy 有限状态机对输入的响应发生在当前时钟周期，比 Moore 有限状态机对输入信号的响应要早一个周期。因此，输入信号的噪声可能影响到输出的信号。

表 8-1: FSM 的二进制编码

状态	y3	y2	y1	y0
A	0	0	0	0
B	0	0	0	1
C	0	0	1	0
D	0	0	1	1
E	0	1	0	0
F	0	1	0	1
G	0	1	1	0
H	0	1	1	1
I	1	0	0	0

表 8-3: FSM 的 One-Hot 编码

状态	y8	y7	y6	y5	y4	y3	y2	y1	y0
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

状态机的编码方式

One-hot 编码也是状态机设计中常用的编码，在 one-hot 编码中，对于任何 给定的状态，其状态向量中只有 1 位是 “1”，其他所有位的状态都为 “0”，n 个 状态就需要 n 位的状态向量，所以 one-hot 编码最长。one-hot 编码对于状态的 判断非常方便，如果某位为 “1” 就是某状态，“0” 则不是此状态。因此判断状态 输出时非常简单，只要一、两个简单的 “与门” 或者 “或门” 即可。 One-hot 编码的状态机从一个状态到另一个状态的状态跳转速度非常快， 而顺序二进制编码从一个状态跳转到另外一个状态需要较多次跳转，并且随着 状态的增加，速度急剧下降。在芯片受到干扰时，one-hot 编码一般只能跳转到无效状态（如果跳到另一有效状态必须是当前为 “1” 的为变为 “0”，同时另外一位变成由 “0” 变为 “1”，这种可能性很小），因此 one-hot 编码的状态机稳定性高。 格雷码 gray-code 也是状态机设计中常用

一种编码方式，它的优点是 graycode 状态机在发生状态跳转时，状态向量只有 1 位发生变化。 一般而言，顺序二进制编码和 gray-code 的状态机使用了最少的触发器， 较多的组合逻辑，适用于提供更多的组合逻辑的 CPLD 芯片。对于具有更多触发器资源的 FPGA，用 one-hot 编码实现状态机则更加有效。所以 CPLD 多使用 gray-code，而 FPGA 多使用 one-hot 编码。对于触发器资源非常丰富的 FPGA 器 件，使用 one-hot 是常用的。 在表 8 3中，状态机有 9 个状态，我们可以用 9 个触发器来实现这个状态机的电路，这 9 个状态触发器用 y8 y7 y6 y5 y4 y3 y2 y1 y0 表示。该状态机的 one-hot 编码如表 8-3所示。

PS/2 接口控制器及键盘输入

PS/2是个人计算机串行 I/O 接口的一种标准，因其首次在 IBM PS/2（Personal System/2）机器上使用而得名，PS/2 接口可以连接 PS/2 键盘和 PS/2 鼠标。所谓串行接口是指信息是在单根信号线上按序一位一位发送的。

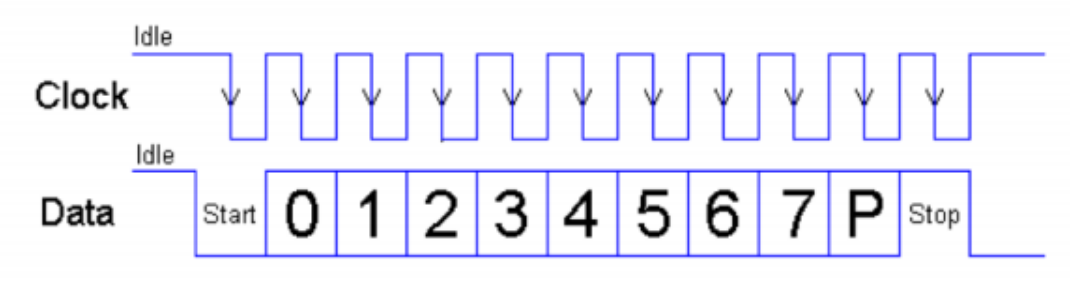


图 8-4: 键盘输出数据时序图

键盘通过 PS2_DAT 引脚发送的信息称为扫描码，每个扫描码可以由单个数据帧或连续多个数据帧构成。当按键被按下时送出的扫描码被称为 “通码 （Make Code）”，当按键被释放时送出的扫描码称为 “断码（Break Code）”。 以 “W” 键为例，“W” 键的通码是 1Dh，如果 “W” 键被按下，则 PS2_DAT 引脚将输出一帧数据，其中的 8 位数据位为 1Dh，如果 “W” 键一直没有释放，则不 断输出扫描码 1Dh 1Dh . . . 1Dh，直到有其他键按下或者 “W” 键被放开。某按键 的断码是 F0h 加此按键的通码，如释放 “W” 键时输出的断码为 F0h 1Dh，分两帧传输。 多个键被同时按下时，将逐个输出扫描码，如：先按左 “Shift” 键（扫描码 为 12h）、再按 “W” 键、放开 “W” 键、再放开左 “Shift” 键，则此过程送出的 全部扫描码为：12h 1Dh F0h 1Dh F0h 12h。

每个键都有唯一的通码和断码。键盘所有键的扫描码组成的集合称为扫描码集。共有三套标准的扫描码集，所有现代的键盘默认使用第二套扫描码。 图 8-5显示了键盘各键的扫描码（以十六进制表示）

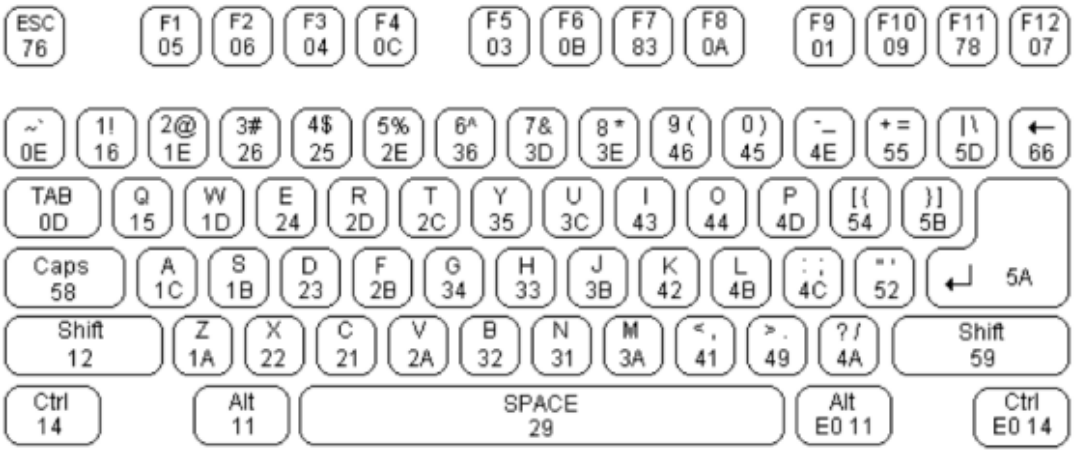


图 8-5: 键盘扫描码

图 8-6是扩展键盘和数字键盘的扫描码。

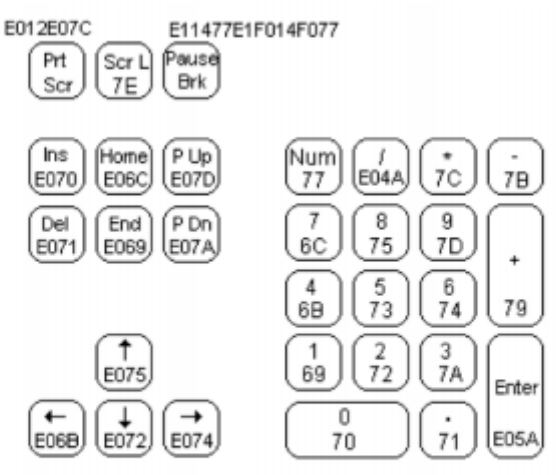


图 8-6: 扩展键盘和数字键盘的扫描码

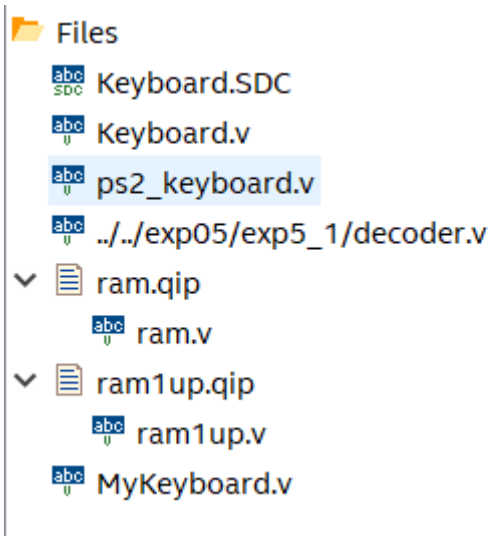
三、实验环境/器材

实验环境是Quarters 17.1 Lite，实验器材是DE10 开发板

四、程序代码或流程图

1. 工程文件

本次实验涉及到多个模块，包括复用之前实验中的分频器和七段编码器等等



2. ps2_keyboard.v

```
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) //read next data
            begin
               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
```

3. MyKeyboard.v

```
module MyKeyboard(input clk,
                  input clrn,
                  input ps2_clk,
                  input ps2_data,
                  //output reg [7:0] data,
                  //output [7:0] ascii,
                  output [6:0] d_digit_low,
                  output [6:0] d_digit_high,
                  output reg [6:0] a_digit_low,
                  output reg [6:0] a_digit_high,
                  output [6:0] c_digit_low,
                  output [6:0] c_digit_high,
                  //output reg[15:0] count,
                  output reg ctrl,
                  output reg shift,
                  output reg up,
                  output reg caps
                  );

   reg [7:0] data;
   reg[15:0] count;
   reg clk_t;
   //reg ctrl_f,shift_f,caps_f;
   integer count_clk;
   reg nextdata_n;
   wire ready;
   wire overflow;
   wire [7:0] temp_data;
   wire [7:0] ascii;
   wire [7:0] ascii_u;
   wire [6:0] tmp_digit_low1,tmp_digit_high1,tmp_digit_low2,tmp_digit_high2;
   reg pre;

   initial
   begin
      clk_t=0;
      count_clk=0;
      nextdata_n=0;
      count=8'b00000000;

      data=8'b00000000;
      pre=1;
      ctrl=0;
      ctrl_f=0;
      up=0;
      shift=0;
      shift_f=0;
      caps=0;
      caps_f=0;
   end

   //divider
   always @ (posedge clk) begin
      if(count_clk>=100) begin
         count_clk<=0;
         clk_t=~clk_t;
      end
      else
         count_clk=count_clk+1;
   end

   //get ps2 info
   ps2_keyboard ps2(
                  .clk(clk),
                  .clrn(clrn),
                  .ps2_clk(ps2_clk),
                  .ps2_data(ps2_data),
                  .data(temp_data),
                  .ready(ready),
                  .nextdata_n(nextdata_n),
                  .overflow(overflow)
               );

   //trans key code to ascii
   ram r1   (
                  .address(data),
                  .clock(clk),
                  .data(8'b00000000),
                  .rden(1'b1),
```

```
                .wren(1'b0),
                .q(ascii)
        );
ram1up r2(
                .address(data),
                .clock(clk),
                .data(8'b00000000),
                .rden(1'b1),
                .wren(1'b0),
                .q(ascii_u)
        );

decoder d1({~pre,data[3:0]},d_digit_low);
decoder d2({~pre,data[7:4]},d_digit_high);

decoder d3({~pre,ascii[3:0]},tmp_digit_low1);
decoder d4({~pre,ascii[7:4]},tmp_digit_high1);
decoder d3u({~pre,ascii_u[3:0]},tmp_digit_low2);
decoder d4u({~pre,ascii_u[7:4]},tmp_digit_high2);

decoder d5({1'b0,count[3:0]},c_digit_low);
decoder d6({1'b0,count[7:4]},c_digit_high);
always @(posedge clk) begin

   if(up==0) begin
      a_digit_low<=tmp_digit_low1;
      a_digit_high<=tmp_digit_high1;
   end
   else begin
      a_digit_low<=tmp_digit_low2;
      a_digit_high<=tmp_digit_high2;
   end

end

always @ (posedge clk_t) begin
   if(ready==1)
   begin
      if(temp_data[7:0]==8'h58) begin //caps

         if((pre==1)&&caps==0) begin
            up=~up;
            caps=1;
         end
         else if(pre==0) begin
            caps=0;
         end
      end

      if(temp_data[7:0]==8'h12||temp_data[7:0]==8'h59) begin   //shift
         if(pre==1&&shift==0) begin
            up=~up;
            shift=1;
         end
         else if(pre==0) begin
            up=~up;
            shift=0;
         end
      end

      if(temp_data[7:0]==8'h14) begin     //ctrl
         if(pre==1&&ctrl==0) begin
            ctrl=1;
         end
         else if(pre==0) begin
            ctrl=0;
         end
      end


      if((temp_data[7:0]!=8'hf0)&&(pre==1)) begin //keep the data
         pre=1;
         data=temp_data;
      end
      else if(temp_data[7:0]==8'hf0) begin
         pre=0;
         count=count+1;
         data=temp_data;
      end
```

```
            else if(pre==0)
               pre=1;

            nextdata_n=0;
         end
         else
            nextdata_n=1;
      end


endmodule
```

4. Keyboard.v

由system builder创建的.v文件，其中的实例化部分

```
//=======================================================
//  REG/WIRE declarations
//=======================================================

MyKeyboard(         .clk(CLOCK_50),
                    .clrn(SW[0]),
                    .ps2_clk(PS2_CLK),
                    .ps2_data(PS2_DAT),
                    //output reg [7:0] data,
                    //output [7:0] ascii,
                    .d_digit_low(HEX0),
                    .d_digit_high(HEX1),
                    .a_digit_low(HEX2),
                    .a_digit_high(HEX3),
                    .c_digit_low(HEX4),
                    .c_digit_high(HEX5),
                    //output reg[15:0] count,
                    .ctrl(LEDR[0]),
                    .shift(LEDR[1]),
                    .up(LEDR[2]),
                    .caps(LEDR[3])
                    );


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

五、实验步骤

- 根据实验讲义复习状态机和键盘输入的相关内容。
- 了解如何利用verilog语言设计 PS/2 键盘控制器

键盘控制器代码：

表 8-4: 键盘控制器

```
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) //read next data
                 begin
                    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
```

- 根据exp07内容设计两个大小为 256*8 的单口存储器，利用ip核生成，并利用.mif文件进行初始化

  ram1：实现从键码转换到ascii码的操作

```
//trans key code to ascii
ram r1   (
                .address(data),
                .clock(clk),
                .data(8'b00000000),
                .rden(1'b1),
                .wren(1'b0),
                .q(ascii)
        );
```

ram1up: 实现在大写情况下从键码转换到ascii码的操作

```
ram1up r2(
                .address(data),
                .clock(clk),
                .data(8'b00000000),
                .rden(1'b1),
                .wren(1'b0),
                .q(ascii_u)
        );
```

- 进行分析与综合，检查是否有语法错误，最终编译通过。
- 进行编译，待编译通过后将二进制文件写入开发板 。
- 在开发板上进行硬件验证。

六、测试方法

1. 运行仿真测试，给输入信号赋予不同的值，观察图中输出与理论输出是否一致 。
2. 将sof文件导入开发板中实际测试电路，观察输出是否符合预期。
3. 在开发板上接入 PS/2 键盘，观察实际功能是否如预期所示。

七、实验结果

键码，ascii码和计数器都正常工作，而且完成了所有高级功能，已经给助教验收。

☞ 高级要求（选做）

- 支持 Shift，CTRL 等组合键，在 LED 上显示组合键是否按下的状态指示
- 支持 Shift 键与字母/数字键同时按下，相互不冲突
- 支持输入大写字符，显示对应的 ASCII 码

八、思考

在思考如何在七段译码器中实现全灭，因为输入只有四位，刚好0~F，没有多余的位数来判断是否要全灭，于是增加了一位输入，而在调用模块的时候需要以这样的格式：

```
(.data({1'b0/1,_data}),.out(output))
```

不知道有没有更方便的办法。

九、实验中遇到的问题及解决方法

在一开始没有完全弄懂ps2_keyboard中各个接口的含义，尤其是nextdata_n，导致在接上键盘实验后没有任何反应，后来通过重新看实验说明发现要在该模块返回一个键码后（即ready=1），要将nextdata_n改为低，而当没有键码输出时，要将nextdata_n改为高，以接收下一个键码。

为了实现高级功能，即多个组合按键，我定义了一个pre变量，用来表示当前是否有按键按下，可以方便实现shift ctrl caps。

十、意见和建议

无