南京大学

数电实验七：

状态机及键盘输入

课程名称：　数字逻辑与计算机组成实验

姓名：　孙文博

学号：　201830210

班级：　数电一班

邮箱：　201830210@smail.nju.edu.cn

实验时间：　2022.4.14 - 4.28

一、 实验目的

1. 学习状态机的工作原理和编码方式；

2. 学习 PS/2 键盘的通码和断码表示；

3. 利用 PS/2 键盘输入实现简单状态机的设计。

二、 实验环境

设计\编译环境：Quartus (Quartus Prime 17.1) Lite Edition

开发平台：DE10-Standard

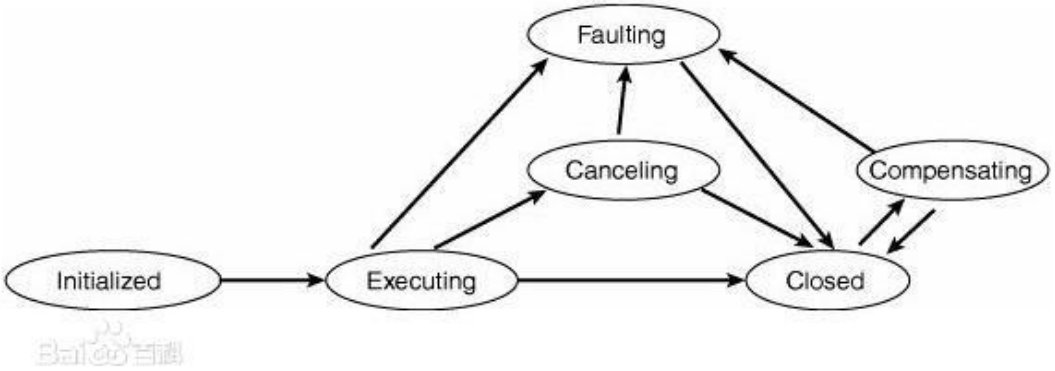
FPGA 芯片：Cyclone II 5CSXFC6D6

三、 实验原理

1. 有限状态机及其编码表示

有限状态机 FSM（Finite State Machine）简称状态机，是一个在有限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。状态机最初从起始状态

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。



有限状态机是数字电路系统中十分重要的电路模块，它是组合逻辑电路和时序逻辑电路的组合。其中组合逻辑分为两个部分，一个是用于产生有限状态机下一个状态的次态逻辑，另一个是用于产生输出信号的输出逻辑，次态逻辑的功能是确定有限状态机的下一个状态；输出逻辑的功能是确定有限状态机的输出。除了输入和输出外，状态机还有一组具有“记忆”功能的寄存器，这些寄存器的功能是记忆有限状态机的内部状态，常被称作状态寄存器。

本实验中的 PS/2 键盘设计就是基于有限状态机的思想实现的，键盘上输入按键和输出按键就对应着不同状态的转换。

2. PS/2 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，PS/2 接口使用两根信号线，一根信号线传输时钟

PS2_CLK，另一根传输数据 PS2_DAT，时钟信号主要用于指示数据线上的比特位在什么时候是有效的。

当按键按下或松开时，键盘会以每帧 11 位的格式串行传送数据给主机，同时在 PS2_CLK 时钟信号上传输对应的时钟。其中第一位是开始位（逻辑 0），后面跟 8 位数据位（低位在前），一个奇偶校验位（奇校验）和一位停止位（逻辑 1）。

3. 键盘扫描码

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。按键被按下时送出的扫描码被称为“通码（Make Code）”，按键被释放时送出的扫描码称为 “断码（Break Code）”。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，当前键盘使用的是第二套扫描码集，内容如下：

在我们的实验中，暂时不考虑拓展键盘及数字键盘的扫描码。

四、 实验过程

1. PS/2 键盘控制器设计

本次实验我们要设计一个能接受键盘输入码的有限状态机，通过七段数码管输出每个按键的通码/断码及其对应的 ASCII 码表示，比如按下键盘 A 按键后，两个数码管显示 1C，同时另外两个数码管显示 a 的 ASCII 码 97。此外，我们记录按键按下的总次数，并通过最后两个数码管显示。

另外实现了选做要求，包括支持 Shift,Ctrl,Caps 等组合按键，当以上三个按键按下时，通过 LED 指示灯提示；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，大写字符的输入方式有两个：按住 Shift 加字符或按下 Caps 后按字符。

下面依次介绍各功能的实现。

首先实验手册中包含了处理键盘数据的代码，它通过设置一个 8 字节的 FIFO 队列处理输入数据，防止发送数据过快造成丢失。其中队列不空时，送出 ready 信号，表示此时有数据要处理；当队列溢出时，送出 overflow 信号。按键处理系统调用该模块时，需要在键盘控制器 ready 信号为 1 的情况下读取键盘数据，确认读取完毕后将 nextdata_n 置零一个周期。这时，键盘控制器模块收到确认读取完毕的信号，将读指针前移，准备提供下一数据。具体代码如下：

```
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
```

在我们的顶层模块中，输入端包括系统时钟 clk，计数清零键 clrn，PS/2 端口的输入数据 ps2_clk 和 ps2_data，输出端包括当前

按键的通码 cur_key，当前按键的 ASCII 码 ascii_key，按键总次数 key_count，特殊按键指示灯 shift_led，ctrl_led，caps_led，以及六个七段数码管。顶层模块声明如下：

```
module exp_7(
    input clk,
    input clrn,
    input ps2_clk,
    input ps2_data,

    output reg [7:0] cur_key,
    output reg [7:0] ascii_key,
    output reg [7:0] key_count,

    output reg [6:0] led0,
    output reg [6:0] led1,
    output reg [6:0] led2,
    output reg [6:0] led3,
    output reg [6:0] led4,
    output reg [6:0] led5,

    output reg shift_led,
    output reg ctrl_led,
    output reg caps_led
    );
```

此外我们设计了一个基于 IP 核的单口 ROM，用于将按键的通码转换成对应的 ASCII 码（对于字母只要考虑小写状态的 ASCII 码，后续判断大小写时会做对应转换）：

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0	0	0	0	0	0	0	0	.....
8	0	0	0	0	0	0	0	0	.....
16	0	0	0	0	0	113	49	0	.....q1.
24	0	0	122	115	97	119	50	0	..zsaw2.
32	0	99	120	100	101	52	51	0	.cxde43.
40	0	32	118	102	116	114	53	0	. vftr5.
48	0	110	98	104	103	121	54	0	.nbhgy6.
56	0	0	109	106	117	55	56	0	..mju78.
64	0	0	107	105	111	48	57	0	..kio09.
72	0	0	0	108	0	112	0	0	...l.p..
80	0	0	0	0	0	0	0	0	.....

接下来是代码的主体部分，这里我们基于状态机的设计思考，键盘的状态大致可以分成按下一个新的按键、按住上次的按键、松开上次的按键三大状态。

当按下一个新的按键时，我们要从键盘中接受通码 XX，将其转换为 ASCII 码，同时在这部分我们还要对 shift，caps，ctrl 三个按键做特殊判断，如果按下了这三个按键中其中一个，通过对应的 state 变量记录这种状态，并将对应的指示灯置为 1：

```
// 按下新的按键
if(out == 0 && keydata != 8'b11110000 && keydata != cur_key)
    begin

    if(keydata == 2'h14)
       ctrl_state <= 1;
    if(keydata == 2'h12)
       shift_state <= 1;
    if(keydata == 2'h58)
       caps_state <= 1;

    cur_key <= keydata;

    if(ctrl_state == 1)
       ctrl_led <= 1;
    if(shift_state == 1)
       shift_led <= 1;
    if(caps_state == 1)
       caps_led <= 1;
```

当按住上次的按键（即接收到的通码依旧等于上周期的键盘码），不做变化，更新当前值即可；当松开了上次的按键，也即此时收到了 0F 加对应的通码的键盘码，则将当前 cur_key 置零，同时判断松开的是否为特殊的三个按键，如果是则修改对应的状态变量 state：

```
// 按任上次的按键
else if(keydata == cur_key)
    cur_key <= keydata;

// 松开上次的按键
else
    if(out == 1 && keydata == 2'h14)
       ctrl_state <= 0;
    if(out == 1 && keydata == 2'h12)
       shift_state <= 0;
    if(out == 1 && keydata == 2'h58)
       caps_state <= 0;

    cur_key <= 0;
```

这里我们需要用一个 out 变量记录输入的断码，以区分断码和通码（因为断码等于 0F 加通码，输入断码后要等一个周期再接受新的通码，否则键盘会将断码误认为是通码）：

```
// out 记录是否是断码
if(keydata == 8'b11110000)
    out <= 1;
else
    out <= 0;
end
```

之后我们将当前按键的通码 cur_key，当前按键的 ASCII 码 ascii_key，按键总次数 key_count 三个变量通过七段数码管以十六进制方式输出：

```
case(cur_key%16)
    0:led0=7'b1000000;
    1:led0=7'b1111001;
    2:led0=7'b0100100;
    3:led0=7'b0110000;
    4:led0=7'b0011001;
    5:led0=7'b0010010;
    6:led0=7'b0000010;
    7:led0=7'b1111000;
    8:led0=7'b0000000;
    9:led0=7'b0010000;
    10:led0=7'b0001000;
    11:led0=7'b0000011;
    12:led0=7'b1000110;
    13:led0=7'b0100001;
    14:led0=7'b0000110;
    15:led0=7'b0001110;
endcase
case(cur_key/16)
    0:led1=7'b1000000;
    1:led1=7'b1111001;
    2:led1=7'b0100100;
    3:led1=7'b0110000;
    4:led1=7'b0011001;
    5:led1=7'b0010010;
    6:led1=7'b0000010;
    7:led1=7'b1111000;
    8:led1=7'b0000000;
    9:led1=7'b0010000;
    10:led1=7'b0001000;
    11:led1=7'b0000011;
    12:led1=7'b1000110;
    13:led1=7'b0100001;
    14:led1=7'b0000110;
    15:led1=7'b0001110;
endcase
```

对于键盘的大写功能，我们检测当前 caps 按键或者 shift 按键是否是按住状态，同时判断按下的按键是否是字符 a-z，如果是大写状态，那么对应的 ASCII 码减去 32：

```
// 判断大写
if(bool == 1 && (caps_state == 1||shift_state == 1))
   ascii_key <= ascii_key - 32;

key_count <= key_count + 1;
end
```

其中 bool 变量判断通过当前键盘码是否是字符，通过 is_char 模块

得到，具体代码如下：

```
module is_char(indata,outdata);
input [7:0] indata;
output reg outdata;
always @(*)
begin
    outdata = 0;
    if(indata == 2'h15    ||
       indata == 2'h1D    ||
       indata == 2'h24    ||
       indata == 2'h2D    ||
       indata == 2'h2C    ||
       indata == 2'h35    ||
       indata == 2'h3C    ||
       indata == 2'h43    ||
       indata == 2'h44    ||
       indata == 2'h4D    ||
       indata == 2'h1C    ||
       indata == 2'h1B    ||
       indata == 2'h23    ||
       indata == 2'h2B    ||
       indata == 2'h34    ||
       indata == 2'h33    ||
       indata == 2'h3B    ||
       indata == 2'h42    ||
       indata == 2'h4B    ||
       indata == 2'h1A    ||
       indata == 2'h22    ||
       indata == 2'h21    ||
       indata == 2'h2A    ||
       indata == 2'h32    ||
       indata == 2'h31    ||
       indata == 2'h3A
       )
          outdata <= 1;
end
endmodule
```

2. 仿真模型检测

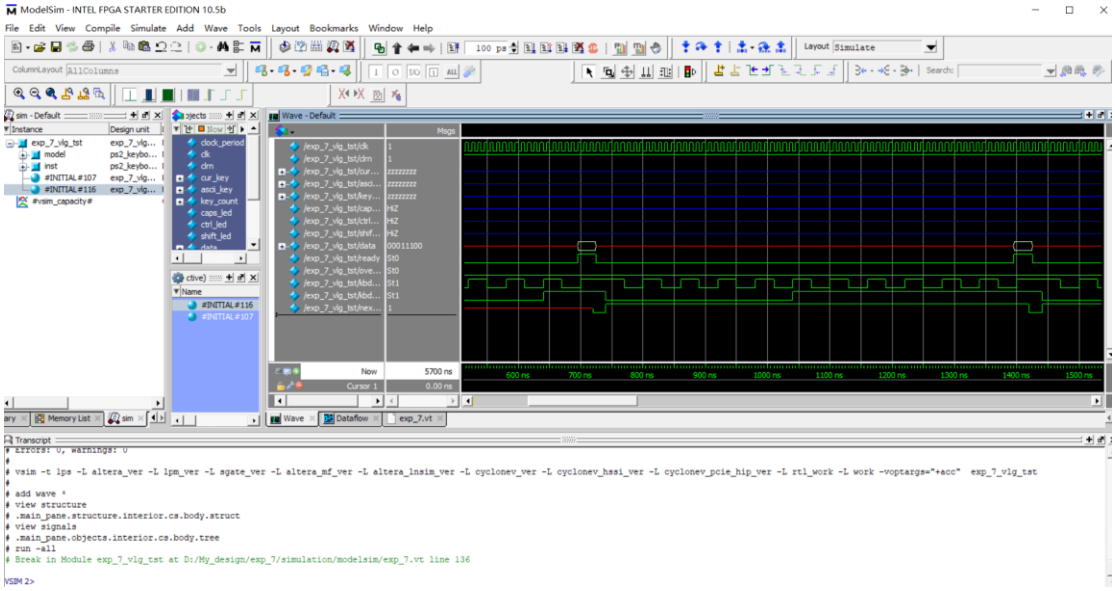
本次实验的仿真模型也需要我们自己编写，好在有助教给的样例，实现方法如下：

```
module ps2_keyboard_model(
    output reg ps2_clk,
    output reg ps2_data
    );
parameter [31:0] kbd_clk_period = 60;
initial ps2_clk = 1'b1;

task kbd_sendcode;
    input [7:0] code; // key to be sent
    integer i;

    reg[10:0] send_buffer;
    begin
        send_buffer[0] = 1'b0; // start bit
        send_buffer[8:1] = code; // code
        send_buffer[9] = ~(^code); // odd parity bit
        send_buffer[10] = 1'b1; // stop bit
        i = 0;
        while( i < 11) begin
        // set kbd_data
            ps2_data = send_buffer[i];
            #(kbd_clk_period/2) ps2_clk = 1'b0;
            #(kbd_clk_period/2) ps2_clk = 1'b1;
            i = i + 1;
        end
    end
endtask

endmodule
```

这里 ps2_keyboard_model 模型即为我们的键盘输入模型，通过输入值给出键盘码。

```
initial begin
   clrn = 1'b0; #20;
   clrn = 1'b1; #20;
   model.kbd_sendcode(8'h1C); // press 'A'
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1;//read data
   model.kbd_sendcode(8'hF0); // break code
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
   model.kbd_sendcode(8'h1C); // release 'A'
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
   model.kbd_sendcode(8'h1B); // press 'S'
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
   #20 model.kbd_sendcode(8'h1B); // keep pressing 'S'
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
   #20 model.kbd_sendcode(8'h1B); // keep pressing 'S'
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
   model.kbd_sendcode(8'hF0); // break code
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
   model.kbd_sendcode(8'h1B); // release 'S'
   #20 nextdata_n =1'b0; #20 nextdata_n =1'b1; //read data
   #20;
   $stop;
end
```

在我们的测试模块中，分别输入 A 的通码，断码，S 的通码，断码，实现按住 A 松开按住 S 松开的模拟过程，仿真结果如下：



其中 data 的值从 00011100 变为 00000000 再变为 00011011，符合预期。接下来进行引脚分配和烧录，等待验收！

五、 实验结果

1. 思考题

本次实验暂无思考题。

2. 上板验收

本次实验验收需要使用到机房的 PS/2 键盘，因此上板验收推迟到 4.28 线下课进行。

六、 总结与反思

本次实验较前几次实验的难度显著提升，无论是从代码量，思维量还是从工程量考虑，都需要花费更多的精力，好在实验时间也是放

到了三周。对于键盘的模拟，我认为最关键的地方是要理解各种键盘码的设计原理，即“为什么会设计成这样”，理解了原理之后，一切便不是那么困难了。当然，本次实验中遇到的 bug 较之前也大幅提升，有一天夜里甚至因为一个分号检查到夜里两点半才发现😭。不过一个个困难解决之后，看着自己的开发板能正确显示键盘内容，还是非常有成就感的，希望接下来的实验可以继续努力把！😉