数电实验EXP8

刘永鹏 191220070 693901492@qq.com 11.4

1. 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：

1. 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。
2. 当按键松开时，**七段数码管的低四位全灭。**
3. 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的情况。

高级要求：

1. 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。
2. 支持 **Shift 键与字母/数字键同时按下**，相互不冲突
3. 支持输入**大写字符**，显示对应的 ASCII 码。

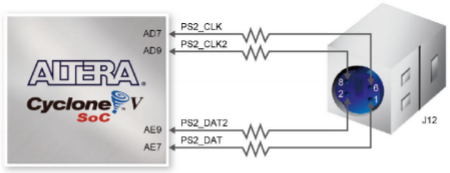
2. 实验原理

1. **有限状态机**

   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。

   有限状态机是数字电路系统中十分重要的电路模块，是一种输出取决于过 去输入和当前输入的时序逻辑电路，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。

2. **PS/2 接口控制器及键盘输入**



(a) PS/2 连线

(b) Y 型转接口

图 8-7: DE10-Standard 开发板上的 PS/2 接口

图 8-8为 DE10-Standard 上的 PS/2 接口引脚列表。

Signal Name	FPGA Pin No.	Description	I/O Standard
PS2_CLK	PIN_AB25	PS/2 Clock	3.3V
PS2_DAT	PIN_AA25	PS/2 Data	3.3V
PS2_CLK2	PIN_AC25	PS/2 Clock (reserved for second PS/2 device)	3.3V
PS2_DAT2	PIN_AB26	PS/2 Data (reserved for second PS/2 device)	3.3V

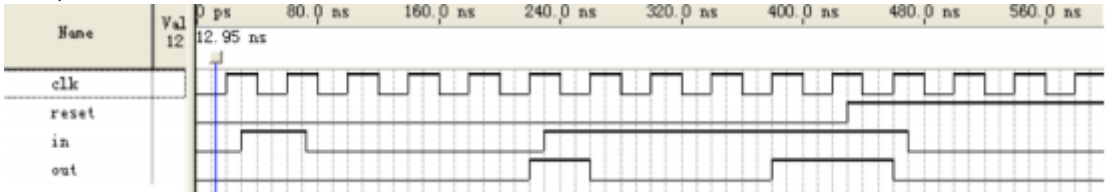
3. **按键处理器状态设计：**
    1. 采用**独热码**设定状态，更好的利用FPGA上充足的寄存器资源。但事实上，个人认为这种方法代码量更大，不便于编写，其实可以采用普通方法编码。
    2. 状态根据实验要求设定，共三种特殊情况：
        1. **shift被按下。**
        2. **ctrl被按下。**
        3. **Caps被打开。**
    3. 上述三种状态自由组合，一共产生8种状态。故state取8位。
4. **模块划分**：
    1. 参考实验pdf实现的**keyboard.v**, 对键盘信号进行初步处理。
    2. **display.v:** 核心文件：处理按键关系，控制8种状态的转移，并作相应的输出（把信号输出到七段数码管上）。
    3. **toASCII.:** 专门读取扫描码->ASCII的mif内存文件。
    4. **keyboardTop.v:** 项目顶层实体，调用以上两个模块。
    5. 具体代码见下文。

3. 实验环境

1. Quartus代码编写+使用实验室PS/2键盘实验。
2. 使用了提供的测试文件，用modelsim初步调试。
3. 上机调试。

4. 实验代码

对与pdf中的小实验，由于其功能与下面键盘的状态机重复，因此不再赘述。



keyboardTop.v

首先是顶层模块：keyboardTop.v

```
module
keyboardTop(clk,ps2_clk,ps2_data,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,spekey,clrn,
testcase,data,ready);
    input clk,ps2_clk,ps2_data;
    input clrn;
    output [3:0]spekey;//SW[9:6],SW[9] is not used
    output [6:0]HEX0,HEX1,HEX2,HEX3,HEX4,HEX5;
    wire [7:0] asdata, kbdata;
    output ready;
    wire overflow; // fifo overflow

    //wire [7:0] times;
reg nextdata_n;
output reg [7:0] data;
output reg [2:0] testcase;

//这里是测试信号，对实验结果无意义
always @ (data)
begin
    case(data)
    8'h12: testcase <= 1;
    8'h14: testcase <= 2;
    8'h58: testcase <= 3;
    8'hf0: testcase <= 4;

    default: testcase <= 0;
    endcase
end

//这里根据键盘信号初步处理的ready，读入有效数据。
always @ (posedge ps2_clk)
begin
    if(ready)
    begin
        data <= kbdata;
        nextdata_n <= 0;
    end
    else begin
        data <= data;
        nextdata_n <= 1;
    end
end

//call modules
    Keyboard k1(.clk(clk),.clrn(clrn),.ps2_clk(ps2_clk),
    .ps2_data(ps2_data),.data(kbdata),.ready(ready),
    .nextdata_n(nextdata_n),.overflow(overflow));/**/

    toASCII t1(.data(data), .asdata(asdata));

    display
d1(.clk(clk),.HEX0(HEX0),.HEX1(HEX1),.HEX2(HEX2),.HEX3(HEX3),.HEX4(HEX4),.H
EX5(HEX5),
    .data(data),.asdata(asdata),.spekey(spekey),.rst(clrn));
endmodule
```

顶层模块的实现较为简单，额外的部分是加入了**[2：0]testcase**测试信号，这一信号位display.v的输出。设定这一信号的目的是方便查看状态机是否进行了正确的状态转移。

此外，**[3:0]spekey** 用来调试特殊键位按下的功能，与testcase功能相近。

toASCII.v

```
//This module receives scan code, and returns corresponding ASCII code.
module toASCII(
    input [7:0]data,
    output [7:0]asdata
);
(* ram_init_file = "ascii.mif" *)reg [7:0] toascii [255:0];

assign asdata = toascii[data];

//ROM
endmodule
```

本文件本来承担了更多功能，但是后期经过调整将这些功能移到了别的模块中，因此本模块只剩下这一个功能了，其实没必要专门用一个文件的。

display.v

接下来着重介绍核心模块：display.v

```
//states
parameter
shiftready = 8'b00000001, //when shift is pressed down, display
corresponding symbols
//this state is on until shift is released
ctrlready = 8'b00000010, //when ctrl is pressed down
//this state is on until shift is released
capsready = 8'b00000100, //when tab is pressed down, corresponding ascii
code adds an offset of -32 or -2 in hex
//this state is on untin tab is pressed again.
nostate = 8'b00001000, //this state does nothing spetial
shiftctrl = 8'b00010000,
shiftcaps = 8'b00100000,
ctrlcaps = 8'b01000000,
allstates = 8'b10000000;
```

状态设计，一共8种状态，采用独热码。事实上不用独热码代码可以得到简化，但这样写可以更好的利用开发板上的寄存器资源。

```
always @ (posedge clk)
begin
    case(state)
    nostate: spekey <= 4'b0000;
    capsready: spekey <= 4'b0001;
    ctrlready: spekey <= 4'b0010;
    shiftready: spekey <= 4'b0100;
    shiftctrl: spekey <= 4'b0110;
```

```
    shiftcaps: spekey <= 4'b0101;
    ctrlcaps: spekey <= 4'b0011;
    allstates: spekey <= 4'b0111;
    default: spekey <= 0;
    endcase
end
```

本段代码为测试信号spekey赋值。

接下来是状态机的设计。我们主要关注的问题是：如何实现组合键？以及如何判断键位松开与否。

```
if(data == curdata)
begin
    state = state;
    curdata = data;
end
else if(data == 8'hf0)
begin
    up = 1;
    state = state;
    curdata = data;
    count = count - 1;
end
```

上面是状态机代码的初始部分，首先**curdata**记录了上一个信号，排除了持续按键持续发射同样信号的情况。如果信号为8'hF0, 也就是断码，我们记录这一信号，再下一次接收信号时再进行处理。

count作为计数记录此时有多少键位被按下，**up**专门记录断码。

```
if(data == 8'h12)
    begin
        if(up == 1)//取消shift
        begin
            case(state)
            shiftready: state = nostate;
            shiftctrl: state = ctrlready;
            shiftcaps: state = capsready;
            allstates: state = ctrlcaps;
            default: state = nostate;
            endcase
            curdata = 8'hf0;
            up = 0;
        end
        else begin
            case(state)
            nostate: state = shiftready;
            ctrlready: state = shiftctrl;
            capsready: state = shiftcaps;
            ctrlcaps: state = allstates;
            default: state = shiftready;
            endcase
            curdata = data;
            times = times + 1;
            count = count + 1;
        end
```

以shift为例, 上面是状态机的转移情况。

对变量up进行单独判断，以确定**此时给出的扫描码是按键按下还是放开。**如果是按键放开（**up == 1**），那么，将curdata记为8'hf0，并将up置零。否则，记录curdata, times(按键计数)++，按下键的个数增加1。

ctrl的判断与之类似，Caps略有不同：因为Caps不需要持续按下。但是基本逻辑相同，下面只给出代码。如果是按下普通按键，不会导致状态机变化。

```
else if(data == 8'h58) //Caps
    begin
        if(up == 0)
        begin
            case(state)
            nostate: state = capsready;
            shiftready: state = shiftcaps;
            ctrlready: state = ctrlcaps;
            shiftctrl: state = allstates;

            capsready: state = nostate;
            shiftcaps: state = shiftready;
            ctrlcaps: state = ctrlready;
            allstates: state = shiftctrl;
            default: state = nostate;
            endcase
            curdata = data;
            times = times + 1;
            count = count + 1;
        end
        else begin
            curdata = 8'hf0;
            state = state;
            up = 0;
        end
    end
    else begin //普通按键
        if(up == 0)
        begin
            curdata = data;
            times = times + 1;
            count = count + 1;
        end
        else begin
            curdata = 8'hf0;
            times = times;
            up = 0;
        end
    end
```

为了处理shift与数字键的联动，以及大写字母的输入，我们还需要如下代码：

```
if(state == shiftready
        || state == shiftcaps
        || state == allstates
        || state == shiftctrl)
    begin
        case(asdata)
```

```
        8'h30: myasdata = 8'h29;
        8'h31: myasdata = 8'h21;
        8'h32: myasdata = 8'h40;
        8'h33: myasdata = 8'h23;
        8'h34: myasdata = 8'h24;
        8'h35: myasdata = 8'h25;
        8'h36: myasdata = 8'h5e;
        8'h37: myasdata = 8'h26;
        8'h38: myasdata = 8'h2a;
        8'h39: myasdata = 8'h28;
        default: myasdata = asdata;
        endcase
    end
    else myasdata = asdata;

    if(state == capsready
        || state == shiftcaps
        || state == ctrlcaps
        || state == allstates)
        offset = 2;
    else offset = 0;
```

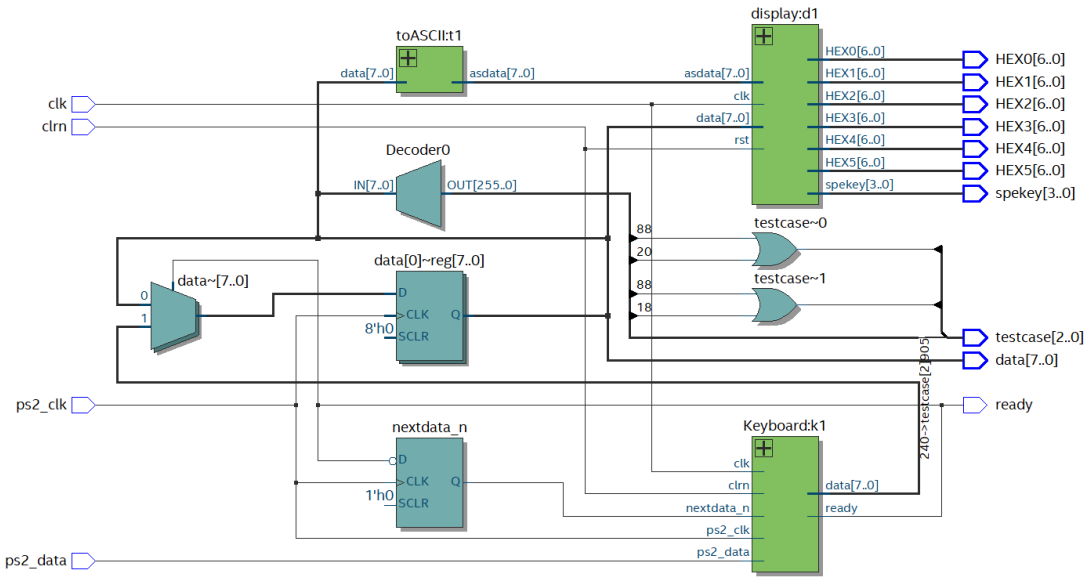
上半部分处理shift+数字键，而下半部份通过**offset**变量完成大写字母显示：**大写字母的ASCII码比对应的小写字母少0x20，也就是高四位少0x20。**因此，在显示时，我们使用如下处理方式：

```
case(myasdata[7:4] - offset)
    ......
```

这样则自动完成了小写<->大写的转换。

至此，所有扩展功能都已经实现。

最后是RTL视图展示。



5. 实验过程

首先modelsim调试，再上机使用键盘调试。

6. 实验结果

已完成验收，实现了**全部扩展功能。**

7. 问题与解决方案/建议

1. 首先是状态机：在RTL实验正常，但在键盘上实验却出现异常。经检查，确认代码的组合逻辑无误，因此一定是时序上出错。
2. 原本使用**always @ (data)**，将data作为激励变量，但总是无法正常运行，直到我将其改为clk，才正常运行，但我无法理解为何不能使用键盘数据作为激励变量。
3. 采用了**阻塞赋值。**使用阻塞赋值的原因是想要防止冲突。原本使用的是非阻塞赋值，但似乎造成了一些时序上的冲突，因此做了一些更改。
4. 状态机事实上还有改进空间：是否能将**0XF0作为状态机的一部分**，这样可以省去一些复杂的if-else判断。
5. **建议：修改给出的测试用例代码。**给出的测试用例代码无法测定边缘情况和特殊键位，因此RTL modelsim验证无误情况下上机烧板总是出bug。我对此代码进行了一些改进，使它可以测试如下情况：
    1. 对相同按键的连续按下。
    2. shift/Caps组合键。