EDA3 报告

刘鸣霄　自 12　2021010584

一、预习报告

1．各个模块的功能

键盘输入模块：扫描检测，将按键情况转化为数字。

数码管显示模块：扫描显示，在四个数码管上显示相应的数字。

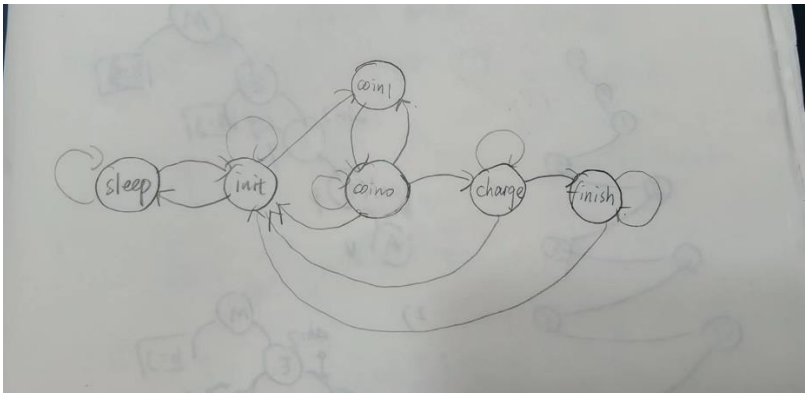
控制单元:整个电路的状态机,根据其他模块的输入决定状态,并给出其他模块的输出。

蜂鸣器：在充电结束时，播放音乐。

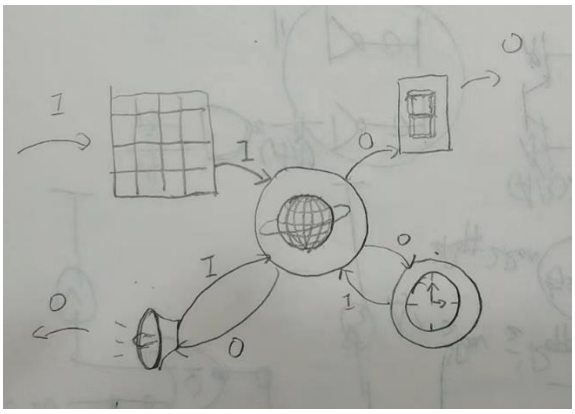
计时器：计算剩余的时间和 10 秒自动熄灭的时间。

2．状态转换图

如下图所示。



二、电路设计

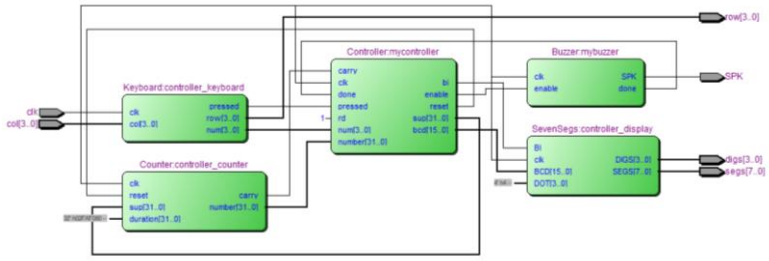


设计思路：将电路拆解成若干个模块：键盘输入模块负责处理输入。数码管显示模块负责显示投币数额和剩余时间。蜂鸣器模块负责充电结束时鸣响提示音。计时器模块负责计算剩余充电时间和处在“开始阶段”的时间。控制器负责电路的状态转换。

1、顶层文件。

```
module Assignment3(clk,col,row,digs,segs,SPK);
```

顶层文件负责连接所有的模块，如下方的 RTL 图所示



2、键盘输入模块

```
module Keyboard(clk,col,row,num,pressed,toggle);
    input clk;
    input [3:0] col;
    output [3:0] row;
    output [3:0] num;
    output pressed;//按下标志，按下为高电平
    output toggle;//按下时会给出一个时钟周期宽度的正脉冲

    wire [15:0] key1;
    wire [15:0] key2;
     wire pressed_temp;

    KeyboardScanner myScanner(
        .clk(clk),
        .col(col),
        .row(row),
        .key(key1),
          .pressed(pressed_temp)
    );//扫描模块
    KeyboardDebouncer myDebouncer(
        .clk(clk),
        .key_in(key1),
        .key_out(key2),
          .pressed_in(pressed_temp),
          .pressed_out(pressed),
          .toggle(toggle)
    );//防抖模块
    Coder_4bits myCoder(
        .onehot(key2),
        .binary(num)
    );//编码器
endmodule
```

输入和输出的意义就是字面意义。特殊的输入和输出端会在注释说明。

1) 扫描模块：

用分频器得到 1kHz 频率时钟，不断的扫描每一行（切换行值），根据不同的列输入得到

对应的按键（独热码，便于消抖）。

```
module KeyboardScanner(clk,col,row,key,pressed);
```

```
    always @(posedge scan_clk) begin//状态机：扫描四行

        case(state)
            s0:state=s1;
            s1:state=s2;
            s2:state=s3;
            s3:state=s0;
            default:state=s0;
        endcase
    end
```

2) 消抖模块：

间隔 20ms 取样一次按键电平，保存 3 次的值（移位寄存），根据 3 次电平的值来确定高低电平

```
module
KeyboardDebouncer(clk,key_in,key_out,pressed_in,pressed_out,toggle);
```

```
    always@(posedge debounce_clk) begin//移位寄存，取到最新电平，其他的依次后移
        key_reg0 = key_in;
        key_reg1 = key_reg0;
          key_reg2 = key_reg1;

          pressed_reg0 = pressed_in;
        pressed_reg1 = pressed_reg0;
          pressed_reg2 = pressed_reg1;
    end
```

```
    assign key_out = (~key_reg0 & ~key_reg1 & ~key_reg2) | (~key_reg0 & ~key_reg1 & key_reg2);
    assign pressed_out = (pressed_reg0 & pressed_reg1 & pressed_reg2) | (pressed_reg0 & pressed_reg1 & ~pressed_reg2);
     //低低低或高低低为高电平，这里把低电平有效转换成了高电平有效
```

3) 编码器：将独热码转换为二进制编码。

实际上，我这里处理麻烦了。以我这种键值策略其实只需要对 pressed 消抖，所以 key 值并不用转化为独热码。

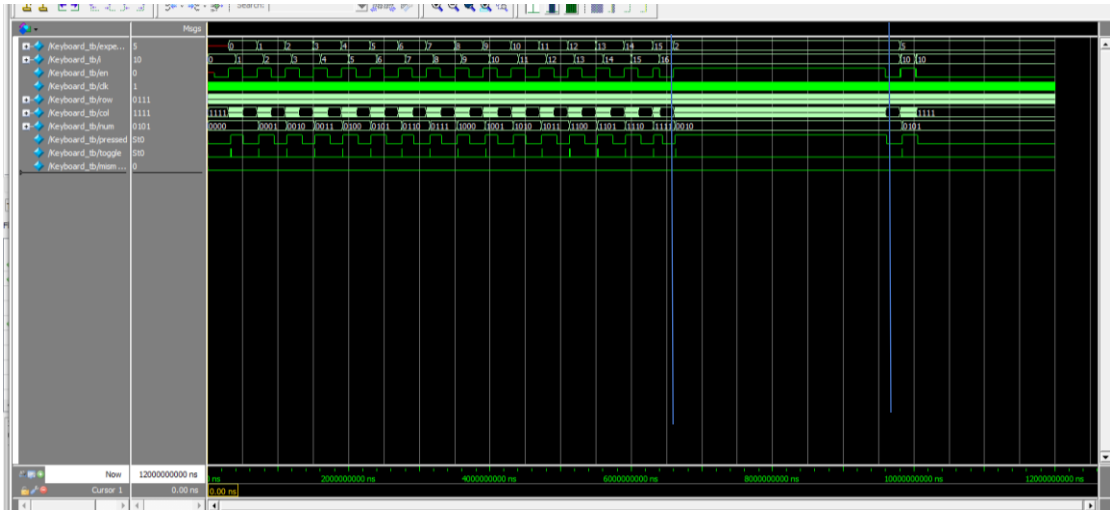
仿真：

为提高仿真速度，我对键盘模块的代码略作了一些修改：去掉了分频器，testbench 给 500Hz 的时钟。后面模块的仿真也做了类似处理。

仿真分三部分：0~15 键逐一按下（100ms），长按键（3s，100ms 在实质上也是长按键，因为扫描周期是 2ms），抖动按下（用宽度 1ms 连续 10 次的脉冲来模拟）。

```
: [PCDPC] - Port size (4 or 4) does not match connection size (32) for port 'num'. The p
ort definition is at: C:/Users/hw/Desktop/Quartus_File/Assignment3/Keyboard_tb.v(63).
#
#         Region: /Keyboard_tb/myn2c
# Loading timing data from Assignment3_v.sdo
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /Keyboard_tb File: C:/Users/hw/Desktop/Quartus_
File/Assignment3/Keyboard_tb.v
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# normal press...
#
# output:0, shouldbe:0
#
# output:1, shouldbe:1
#
# output:2, shouldbe:2
#
# output:3, shouldbe:3
#
# output:4, shouldbe:4
#
# output:5, shouldbe:5
#
# output:6, shouldbe:6
#
# output:7, shouldbe:7
#
# output:8, shouldbe:8
#
# output:9, shouldbe:9
#
# output:a, shouldbe:a
#
# output:b, shouldbe:b
#
# output:c, shouldbe:c
#
# output:d, shouldbe:d
#
# output:e, shouldbe:e
#
# output:f, shouldbe:f
#
# long press...
#
# debounce...
#
# ** Note: $finish    : C:/Users/hw/Desktop/Quartus_File/Assignment3/Keyboard_tb.v(58)
#    Time: 12 sec  Iteration: 0  Instance: /Keyboard_tb
```

调试台的输出正常



波形正常。长按键和抖动按键（仔细看抖动边沿是粗的）也能正常工作。抖动按键的输出信号（如 pressed 和 toggle）并不出现抖动现象。

3、数码管显示模块：

```
module SevenSegs(clk,BCD,DOT,BI,DIGS,SEGS);
    input clk;
    input[15:0] BCD;//输入的 8421 码，4 位为 1 组
    input[3:0] DOT;//控制 4 个小数点
    input BI;//灭灯输入端

    output reg[3:0] DIGS;//使能输出
    output reg[7:0] SEGS;//段输出
```

用分频器得到 1kHz 频率时钟，逐个给 4 个数码管使能并给相应的数据，通过人眼的视觉暂留效应实现四段数码管同时显示的效果。

验证：下载到 FPGA 板上。顶层模块是这个：

```
module Test_KeyboardDisplay(clk,col,segs,digs,row,pressed,toggle);
```

可以同时仿真键盘和数码管。左面两位是按键次数，右面两位是按键码。

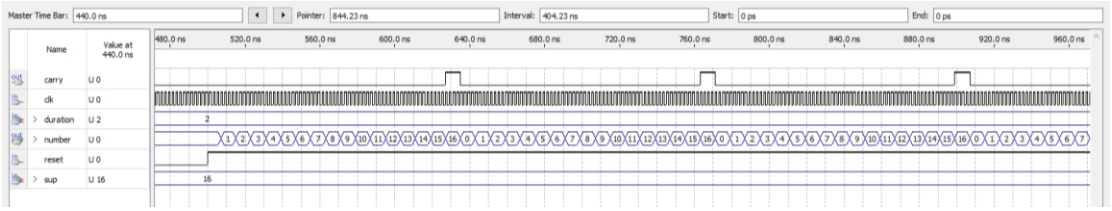
4、 计时器

计时器就是计数器。计时器计充电时间和灭灯时间。计数器模块在蜂鸣器模块中也会用到：

```
module Counter(clk,sup,duration,reset,number,carry);
    input clk;
    input[31:0] sup;//计的最大数
    input[31:0] duration;//分频周期，决定计数速度
    input reset;//归零端

    output reg[31:0] number;//计的数
    output reg carry;//进位
```

计时器的原理较简单：每个时钟上升沿将数字加 1，记满后归 0。即将计满的时候给出进位输出 carry。

仿真：



工作正常

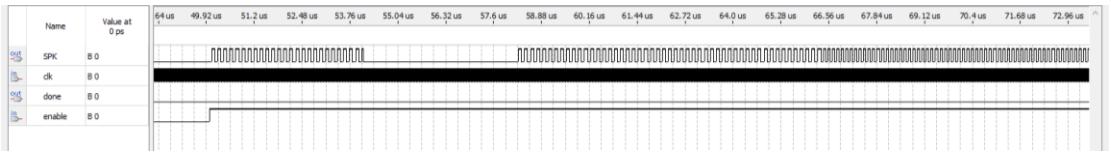
5、 蜂鸣器模块：

```
module Buzzer(clk,enable,done,SPK);
    input clk;
    input enable;//使能端：每次使能都重新播放曲子，高电平有效

    output done;//播放完后输出 1
    output SPK;//音频信号

    wire[4:0] note;
    ROM myROM(.clock(clk),.address(counter_num),.q(note));//ROM

    wire[31:0] counter_num;
    Counter
mycounter(.clk(clk),.sup(255),.duration(3000000),.reset(enable),
        .number(counter_num),.carry(done));//计数器

    NoteToSignal
mysig(.clk(clk),.note(note),.SPK(SPK),.mute(enable));//转换为音频信号
```

```
endmodule
```

1) 计数器：将 ROM 中的音符逐个读出。计数总数由音乐的长度决定，计数速度取决于音乐的速度。
2) ROM：储存乐谱，每个地址存储 1/4 拍的音符。曲子是泉水叮咚响(我家洗衣机的音乐)。
工程文件里还有一个曲子(车尔尼 599 钢琴练习曲 No.60，清华的上下课铃)，但是效果不好，所以没用这个。
3) 信号转换器：将音符转换为音频信号 SPK，由分频器来实现。

仿真：这是其中一段。为便于仿真，各个模块都做了降频处理。

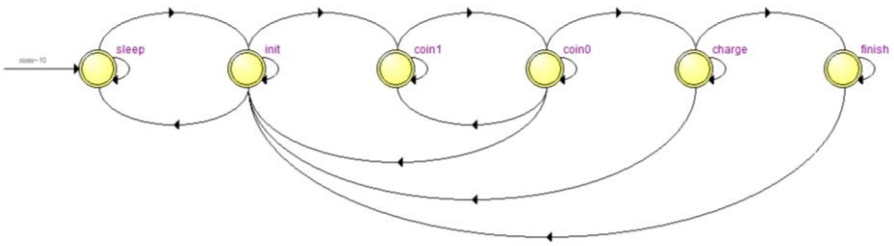


6、 控制模块：

```
module Controller(clk,rd,num,pressed,sup,reset,number,
                  carry,bcd,bi,enable,done);
    always@(posedge controller_clk or negedge rd)begin
     if(!rd) state=sleep;
     else
        case(state)
            sleep:
                if(pressed&&num==10)state=init;//按开始键：开始
                else state=sleep;
            init:
                if(carry)state=sleep;//10s 灭灯
                else if(pressed&&num>=0&&num<=8)state=coin1;
                                    //按一个数字：输入
                else state=init;
            coin0:
                    //if(carry)state=sleep;//也可以设置投币状态的灭灯
                if(pressed&&num==11)state=init;//清零键：清零
                else if(pressed&&num==12)state=charge;//确认键：开始充电
                else if(pressed&&num>=0&&num<=9)state=coin1;//继续输入
                else state=coin0;
            coin1:
                if(!pressed)state=coin0;//按键松开：回到 coin0
                    else state=coin1;
            charge:
                if(carry)state=finish;//充电时间结束：蜂鸣器响
                else if(pressed&&num==11)state=init;//清零键：清零
                    else state=charge;
            finish:
```

```
                if(done||(pressed&&num==11))state=init;
                        //蜂鸣器结束：开始状态
                else state=finish;
            default:state=sleep;
        endcase
    end
```

控制模块的任务是控制整个电路的状态。接受所有模块的输入然后跳转到相应的状态，并给出正确的输出。实际上控制模块只是一个状态机。

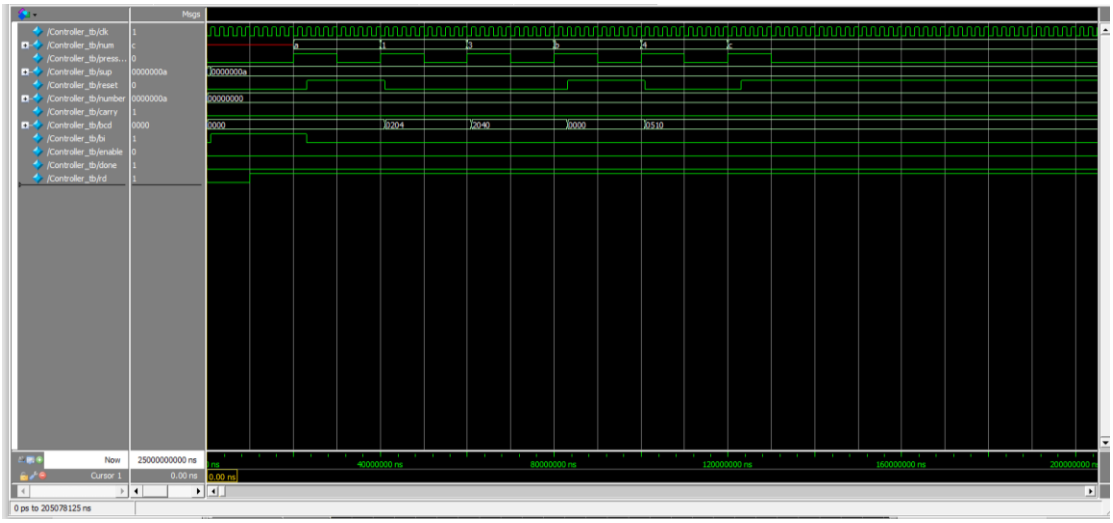


6 个状态的意义：
sleep：初始状态
init：开始状态
coin0：输入状态，并且按键没有按下
coin1：输入状态，并且按键正在按下
charge：正在充电
finish：播放音乐
状态之间的转换关系见上面程序的注释

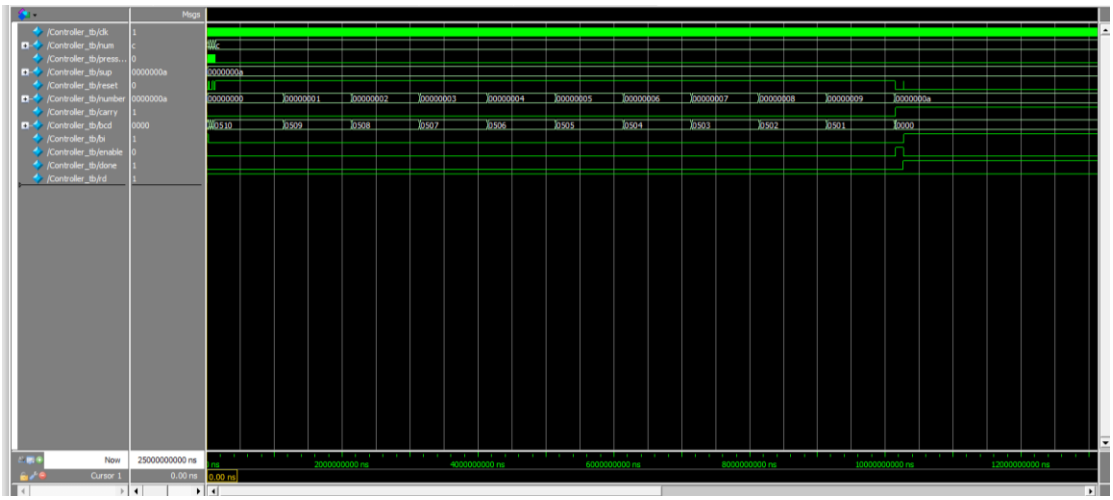
控制模块的仿真：
控制台输出：

```
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# init:bcd=0000
#
# input1:bcd=2040
#
# clear:bcd=0000
#
# input2:bcd=0510
#
# timeover:buzzer=1
#
# sleep:rbi=1
#
# ** Note: $finish    : C:/Users/hw/Desktop/Quartus_File/Assignment3/Contro
ller_tb.v(56)
#    Time: 25 sec  Iteration: 0  Instance: /Controller_tb
# 1
```

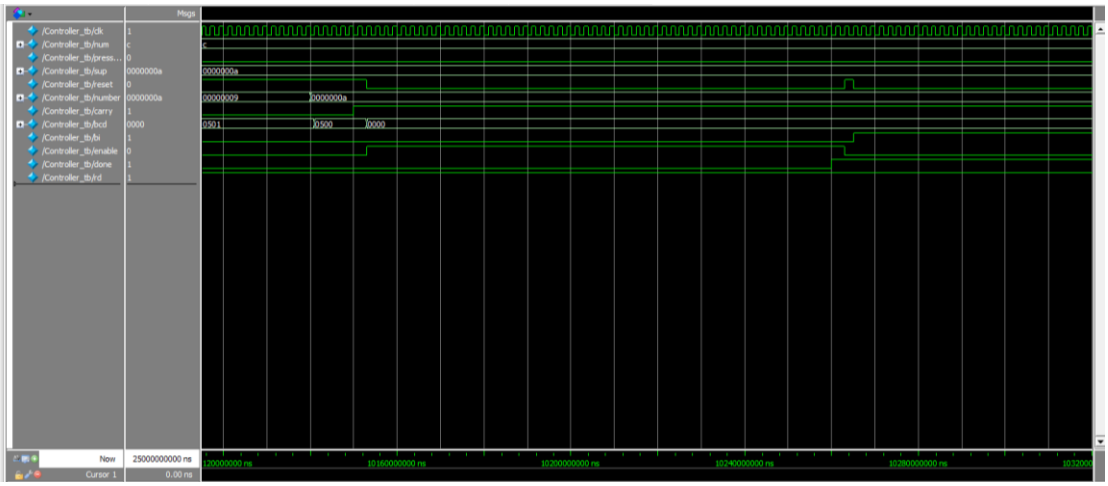
输入时的波形：数码管数值 bcd 正确
数码管的数值为 bcd，灭灯信号为 bi；蜂鸣器的使能信号为 enable，完成信号为 done；计数器的计数上限为 sup，归零信号为 reset，数字为 number，进位为 carry；键盘的数字为 num，按下信号为 pressed。下面波形是 16 进制。



充电时的波形：计时器时间 number 正确，数码管 bcd 正确



播放音乐时的波形：enable 信号正确。并且灭灯功能能工作，bi 信号正确。



7、 分频器：原理如下述代码所示

```
module Filter(clk_in,cnt,clk_out,reset);//分频器
input clk_in;
input[31:0] cnt;
```

```
input reset;

output reg clk_out;

integer i;

initial begin
    i=0;
     clk_out=0;
end

always @(posedge clk_in or negedge reset) begin
    if(reset==0)begin
        i=0;
        clk_out=0;
    end
    else if(i>=cnt/2) begin
        i=0;
        clk_out=~clk_out;
    end
    else i=i+1;
end


endmodule
```

三、遇到的问题

做这个大作业的过程中遇到了很多问题，这里简述一下遇到的最棘手的一个问题。

当时我混淆了控制电路和顶层模块。控制电路我写了 40 多个状态（对应 20 个数字），然后在控制模块中就其他模块，将控制模块作为顶层模块。最后没有生成状态机，因为当时我把状态变量作为了输出，在这种情况下 Quartus II 不会生成状态机。后来我把控制电路和顶层模块拆分开，控制模块只控制状态转换，且不调用器件，只提供接口，并且简化状态数。最后解决了问题。

最初的代码就是 Controller 模块下面一大段被注释掉的代码。