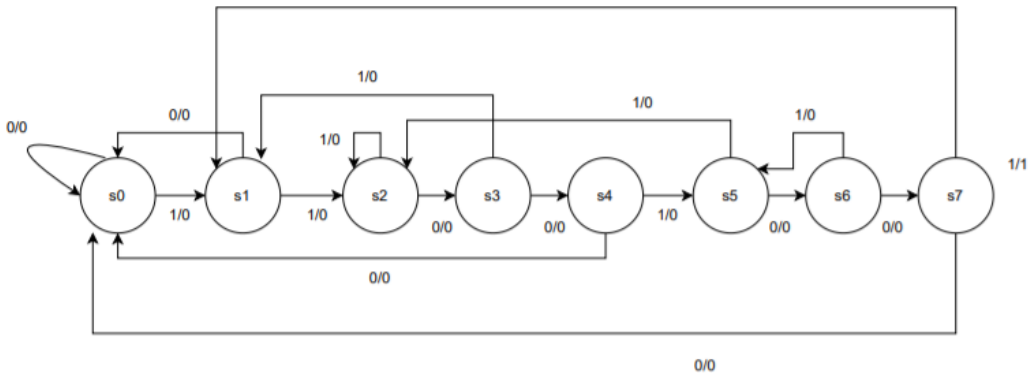
Lab 5 Team 20 Report

109062222 徐嘉徽

109062119 李佳栩

Advance_1:

這是我們的 state_diagram。

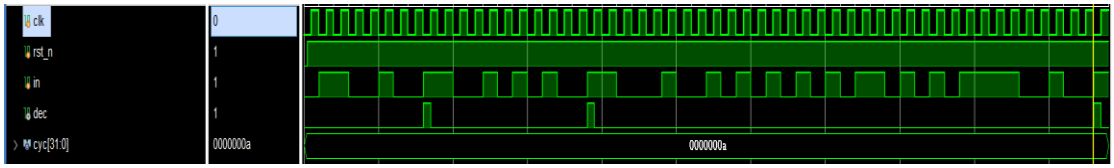


這題是上次 lab 的進階版，這題是一個 mealy machine 也就是 output 要跟著 state 改變，需要注意的地方就是當我 state 等於 s2 的時候，再接收到 1 是要回到 s2，因為整串數字就是 111，便從最右邊兩個 1 繼續讀取，相同的地方在 s5 和 s6 也有，因為讀進來可以是 1100(10)+01，(10)可以很多次，所以 s6 如果 input 是 1 則回到 s5。

Testbench:

我測了 6 種測資，分別是 11001001、110010101001、110001001、10101010、1011010101、1111001001。

答案分別是 1,1,0,0,0,1



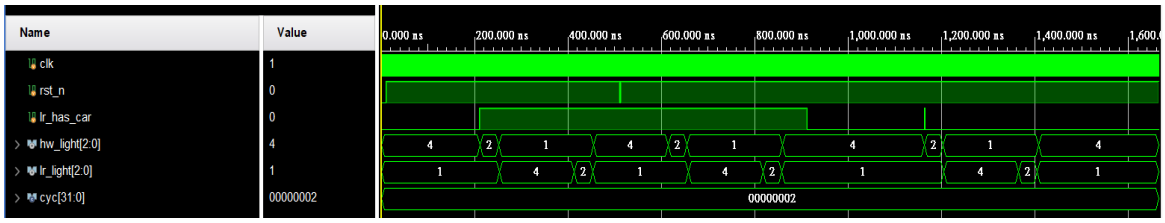
Advance_2:

這題是要做紅綠燈，總共有 6 種，用 FSM 蠻好寫的，只要把每個 state 的狀態都寫好，也記得要寫 default，一開始以為這題沒有甚麼陷阱，後來才發現如果等待的時間超過 80 秒很多的話，我用來記住時間的 counter 會爆掉，所以才改成

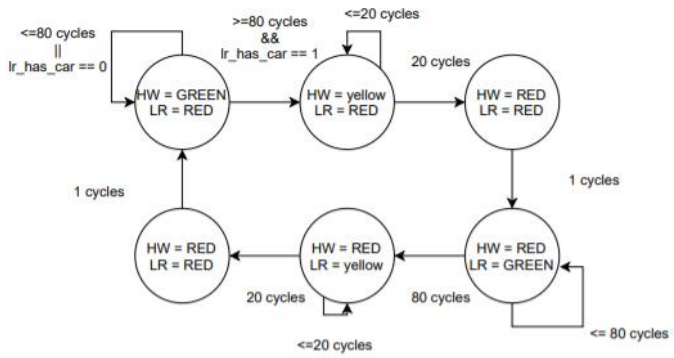
```
if(counter >= 7'd80) begin
    HW_eighty = 1'b1;
end
else HW_eighty = HW_eighty + 1'b0;
if(lr_has_car && HW_eighty) begin
    state_next = s1;
    counter_next = 1'b1;
    hw_light_next = 3'b010;
    lr_light_next = 3'b001;
end
else begin
    counter_next = counter + 1'b1;
    state_next = state;
    hw_light_next = 3'b100;
    lr_light_next = 3'b001;
end
```

這樣即使 overflow 也沒關係。

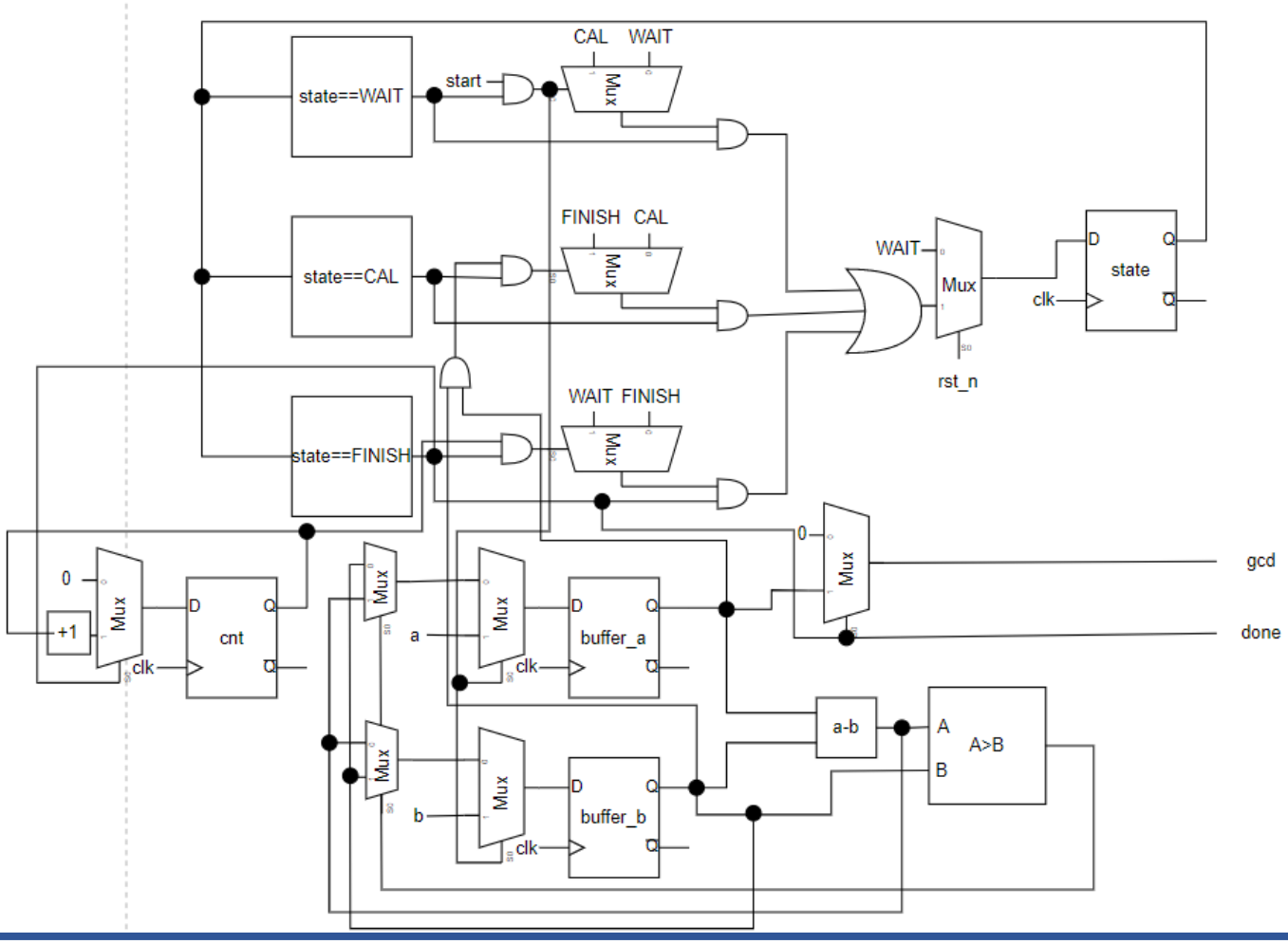
Testbench:



前面就照著 80 秒、20 秒、1 秒、80 秒、20 秒、1 秒來測試，也有測試 reset，最後測試如果一開始綠燈的時間超過 80 秒且 overflow 的情況。



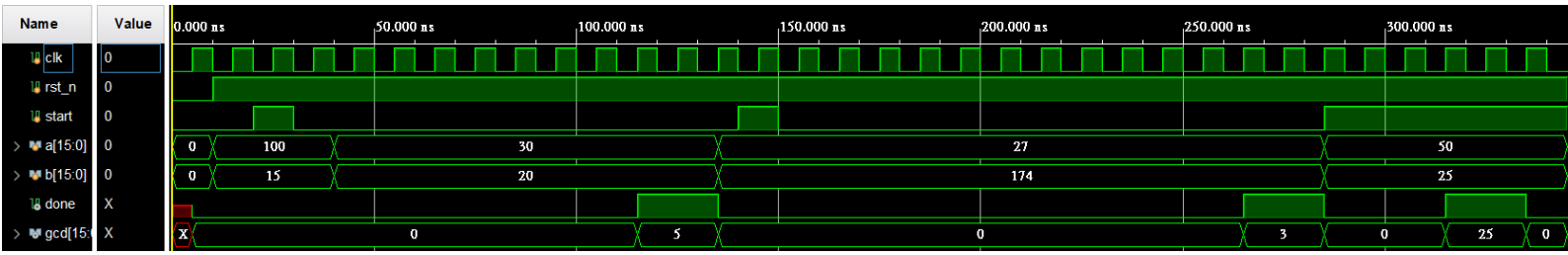
Advanced 3:



我們這題使用的 state diagram 如 pdf 上所附，所以就不再附上。首先，我們設定了 3 個 state : WAIT、CAL、FINISH。當 state==WAIT，代表 state machine 在初始狀態，gcd 跟 done 這兩個 output 都是 0，並且在 start==1 時轉換成下個 state CAL，也就是 next_state=CAL。當 state==CAL，便會執行找出最大公因數的功能，而為了完成此功能，我們利用輾轉相除法。由於每個 clock cycle 只能執行一次，所以設了 5 種 16 bit 的 reg : buffer_a,next_buffer_a,buffer_b, next_buffer_b，以及一個運算元 t。當 state 從 WAIT 變成 CAL 時，把 a、b 值存到 next_buffer_a 及 next_buffer_b 中，這樣下個 cycle 就會把 buffer_a 跟 buffer_b 更新成我們抓到的 a 跟 b。接下來就每個 cycle 做一次輾轉相除 `t=buffer_a % buffer_b; next_buffer_a=buffer_b; next_buffer_b=t;`

且如果 buffer_b==0 時，就代表輾轉相除法做完並找出 gcd，於是讓 next_state==FINISH，結束運算。當 state==FINISH，代表我們已經找出最大公因數，所以讓 output gcd=剛剛找出的答案，也就是 buffer_a，且讓 done=1。由於題目要求要讓 FINISH state 維持兩個 cycle，所以利用 counter 完成，讓 counter 在 FINISH 之外都為 0，並在 FINISH 時每個 cycle 加 1，數到 1 就讓 next_state=WAIT，就會讓 FINISH 在 2 個 cycle 後變成 WAIT，完成題目的要求。

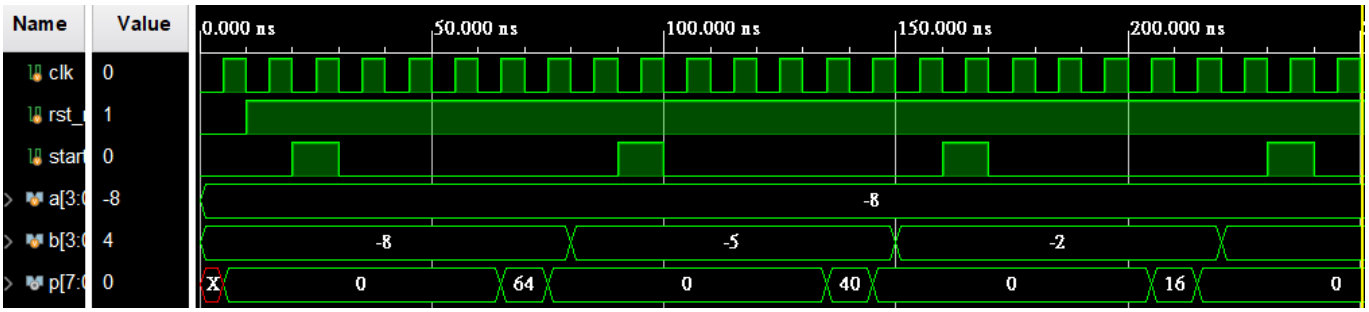
我們測試的測資有 a=16'd100,b=16'd15、a=16'd27,b=16'd174、a=16'd50,b=16'd25。並且觀察一些當不是我們要偵測 input 時且 input 改變會不會出現錯誤，waveform 如下:



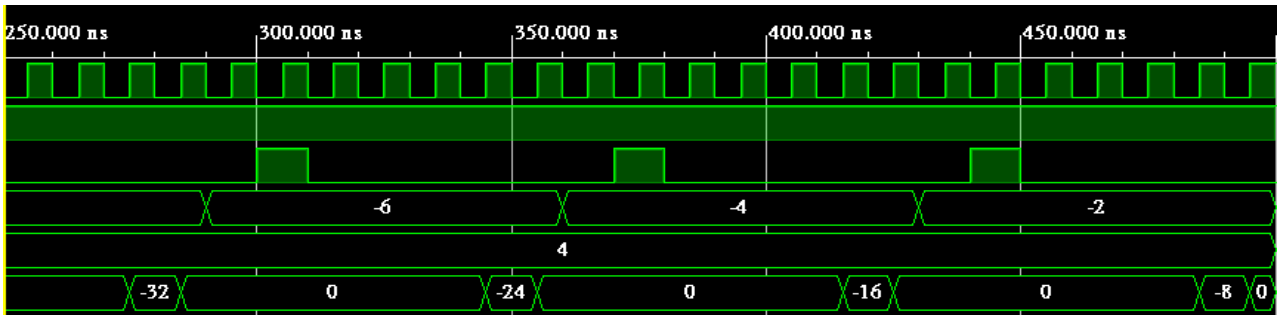
Advanced 4 Bonus:

我們這題使用的 state diagram 如 pdf 上所附，所以就不再附上。首先，我們設定了 3 個 state : WAIT、CAL、FINISH。當 state==WAIT，代表 state machine 在初始狀態，output p=0，並且在 start==1 時轉換成下個 state CAL，也就是 next_state=CAL。當 state==CAL，便會執行我們的 Booth multiply，總共執行 4 個 clk cycle，並在 4 個 cycle 後將 state 變成 FINISH，也就是 next_state=FINISH。而 booth multiply 的方法為 ： 我們利用一個 reg[8:0] ans 以及 reg[3:0]cur_a, cur_b 做運算。在 WAIT 變到 CAL 的時候，將 input a 跟 b 的值存到 cur_a,cur_b 中，並讓 ans = {A,cur_b,cur_b_{-1}}，且 A 的初始值為 4'b0000，cur_b_{-1} 的初始值也是 0。接著便是連續判斷 4 次，總共有四種情況 : (1) cur_b_0 cur_b_{-1} == 10 : A = A - cur_a，並且將 ans 右移 1 bit (2) cur_b,cur_b_{-1} == 11 : 沒有操作，並且將 ans 右移 1 bit (3) cur_b,cur_b_{-1} == 01 : A = A + cur_a，並且將 ans 右移 1 bit (4) cur_b,cur_b_{-1} == 00 : 沒有操作，並且將 ans 右移 1 bit。每一個 clk cycle 分別判斷為哪種情況，並做出相對應的操作。當在 state==CAL 過了 4 個 cycle，next_state 就會變為 FINISH，並且如果 cur_a==-4'd8，output p= -ans[8:1]，其他情況則是 p= ans[8:1]。並在一個 cycle 後 state 變回 WAIT，完成我們的設計。

我們測試的測資有 a=-8、b=-8,a=-8、b=-5,a=-8、b=-2,a=-8、b=4 以下是固定 a。



a=-6、b=4,a=-4、b=4,a=-2、b=4 以下為固定 b，waveform 如下:



FPGA_1

這題主要的作法是 Trace code 看出 basic lab 中的 module 分別能做哪些功能，再針對我們需要的部分改一下 code，於是分成每個 module 講解:

PlayerCtrl:

PlayerCtrl 則是用來判斷現在撥放到譜的哪個位置以及決定要往上撥放還是往下撥放。

這個 module 有三個 input clk,reset,dir，當 dir==1 時往上撥放，dir==0 時往下撥放，且在每次 posedge clk 時判斷。最後當 reset==1 時，將撥放位置重置到 Music 的最前面，並把 dir 設為 1。

Parameter BEATLEAGTH 代表我們譜中音符的!

ibeatNum。

```
6  module PlayerCtrl (
7      input clk,
8      input reset,
9      input dir,
10     output reg [7:0] ibeat
11 );
12 parameter BEATLEAGTH = 30;
13
14 always @(posedge clk, posedge reset) begin
15     if (reset)
16         ibeat <= 0;
17     else if (ibeat > 0 && ibeat < BEATLEAGTH) begin
18         if(dir) ibeat <= ibeat + 1;
19         else ibeat<=ibeat-1;
20     end else begin
21         if(ibeat==0 && dir==1) ibeat<=ibeat+1;
22         else if(ibeat==BEATLEAGTH && dir==0) ibeat<=ibeat-1;
23         else ibeat<=ibeat;
24     end
25 end
26
27 endmodule
```

PWM_gen:

PWM_gen 的用途跟 clock divider 很像，能根據我們的需求產生對應的 frequency。

根據 input [31:0] freq 來決定一秒要撥放幾個音符，例如我們題目需要兩種頻率 :1 秒一個音與 0.5 秒一個音(1 秒兩個音)，則這兩個情形的 freq 分別等於 1 跟 2。Count_duty 的作用則是讓產生的 frequency 圖形可以整理比較漂亮。

```
10 module PWM_gen (
11     input wire clk,
12     input wire reset,
13     input [31:0] freq,
14     input [9:0] duty,
15     output reg PWM
16 );
17
18 wire [31:0] count_max = 100_000_000 / freq;
19 wire [31:0] count_duty = count_max * duty / 1024;
20 reg [31:0] count;
22 always @(posedge clk, posedge reset) begin
23     if (reset) begin
24         count <= 0;
25         PWM <= 0;
26     end else if (count < count_max) begin
27         count <= count + 1;
28         if(count < count_duty)
29             PWM <= 1;
30         else
31             PWM <= 0;
32     end else begin
33         count <= 0;
34         PWM <= 0;
35     end
36 end
37
38 endmodule
```

在我們的 Top module 中，總共使用了 3 個 PWM_gen : (1)Tone_gen : 負責聲音輸出 (2) Fast_gen : 產生 1 秒兩個音符的 frequency (3) Slow_gen : 產生 1 秒一個音符的 frequency

SampleDisplay :

SampleDisplay 則是用來偵測按下鍵盤的哪個鍵以及做出相對應的功能，在這題需要用的四個鍵為 w、s、enter、r。首先，要先找出這四個鍵各自的 MakeCode : W=>1D,S=>1B,R=>2D,ENTER=>5A 接著再分別給四個鍵各自的 key_num，從 3'b000 ~ 3'b011。最後再決定按下每個鍵時會發生甚麼事，例如按下 W 會讓 dir 變成 1。我將 code 放在下面

Top :

Top 便是將上述的 module 全部整合以後，做出題目的要求。

```
13      parameter [8:0] KEY_CODES [0:3] = {
14          9'b0_0001_1101, // W => 1D
15          9'b0_0001_1011, // S => 1B
16          9'b0_0010_1101,  // R => 2D
17          9'b0_0101_1010   // Enter => 5A
18      };
70      always @ (*) begin
71          case (last_change)
72              KEY_CODES[00] : key_num = 3'b000;
73              KEY_CODES[01] : key_num = 3'b001;
74              KEY_CODES[02] : key_num = 3'b010;
75              KEY_CODES[03] : key_num = 3'b011;
76              default       : key_num = 3'b111;
77          endcase
78      end
```

```
39      always @ (posedge clk) begin
40          if(enter) enter<=0;
41          else enter<=enter;
42          if (been_ready && key_down[last_change] == 1'b1) begin
43                  if(key_num==3'b000)begin
44                      dir<=1;
45                      speed<=speed;
46                      enter<=0;
47                  end
48                  else if(key_num==3'b001)begin
49                      dir<=0;
                        speed<=speed;
                        enter<=0;
                    end
                    else if(key_num==3'b010)begin
                        dir<=dir;
                        speed<=~speed;
                        enter<=0;
57                  end
58                  else if(key_num==3'b011)begin
59                      dir<=1;
60                      speed<=0;
61                      enter<=1;
62                  end
63                  else begin
64                      dir<=dir;
65                      speed<=speed;
66                      enter<=0;
67                  end
68          end
69      end
```

FPGA_2:

這題是要寫一個販賣機，因為連續兩次都不是我負責 FPGA，所以這次花了很多時間在這題上面，原本我是想了三個 state，分別為投幣、買、退錢，後來想想發現買可以跟投幣合併，因為最多只會買一罐，這樣比較方便，所以最後我只有買和退錢兩個 state。

```
clock_div_display cd0(clk,clk_display);
debounce d0(deb0, Top, clk);
onepulse o0(deb0, clk, rst_n);//top

debounce d1(deb1, Down, clk);
onepulse o1(deb1, clk, cancel);//down

debounce d2(deb2, Left, clk);
onepulse o2(deb2, clk, five);//Left

debounce d3(deb3, Right, clk);
onepulse o3(deb3, clk, fifty);//right

debounce d4(deb4, Center, clk);
onepulse o4(deb4, clk, ten);//center
```

這邊就是負責 clock divider 和五個按鍵的 onepulse 跟 debounce。

```
always@(posedge clk) begin
    if(rst_n) begin
        money <= 7'd0;
        state <= insert_state;
        cancnt <= 32'd0;
    end
    else begin
        state <= next_state;
        money <= next_money;
        cancnt <= cancnt_next;
    end
end
```

上面這個 sequential circuits 則是負責處理歸零跟 state 的轉換，money 代表現在總共投了多少錢，cancnt 則是在處理退錢時算秒數的 counter。

```
assign LED[3] = (state == 2'b00 && money >= 8'd75) ? 1 : 0;
assign LED[2] = (state == 2'b00 && money >= 8'd50) ? 1 : 0;
assign LED[1] = (state == 2'b00 && money >= 8'd30) ? 1 : 0;
assign LED[0] = (state == 2'b00 && money >= 8'd25) ? 1 : 0;
```

這是在處理 LED 顯示，必須在 state 在投幣的時候且錢需大於要買的飲料。

```
module insertmoney(seg0, seg1, seg2, seg3, money);
input [6:0] money;
output reg [6:0] seg0, seg1, seg2, seg3;

always @(*) begin
    if(money >= 7'd100) begin
        seg0 = 7'b0000001;
        seg1 = 7'b0000001;
        seg2 = 7'b1001111;
        seg3 = 7'b1111111;
    end
    else if(money == 7'd95) begin
        seg0 = 7'b0100100;
        seg1 = 7'b0000100;
        seg2 = 7'b1111111;
        seg3 = 7'b1111111;
    end
    else if(money == 7'd90) begin
        seg0 = 7'b0000001;
        seg1 = 7'b0000100;
        seg2 = 7'b1111111;
        seg3 = 7'b1111111;
    end
```

這個 module 則是在處理 display 數字的部分，因為沒想到更好的辦法所以選擇用窮舉的方式，以下就先省略。

```
   always @ (posedge clk, posedge rst) begin
       if (rst) begin
           drink <= 3'b000;
       end else begin
           drink <= 3'b000;
           if (been_ready && key_down[last_change] == 1'b1) begin
               if (key_num != 3'b000)begin
                   drink <= key_num;
               end
           end
       end
   end

   always @ (*) begin
       case (last_change)
           KEY_CODES[00] : key_num = 3'b001;
           KEY_CODES[01] : key_num = 3'b010;
           KEY_CODES[02] : key_num = 3'b011;
           KEY_CODES[03] : key_num = 3'b100;
           default       : key_num = 3'b000;
       endcase
   end
```

在處理鍵盤的部分我也是研究了一陣子，最後我用了一個 drink 當作我選擇的飲料，如果沒選就是 3’b000，剩下就以此類推。

```
if(drink == 3'b000) begin
    if(ten) begin
        if(money + 7'd10 >= 7'd100) next_money = 7'd100;
        else next_money = money + 7'd10;
    end
    else if(five) begin
        if(money + 7'd5 >= 7'd100) next_money = 7'd100;
        else next_money = money + 7'd5;
    end
    else if(fifty) begin
        if(money + 7'd50 >= 7'd100) next_money = 7'd100;
        else next_money = money + 7'd50;
    end
    else if(cancel) next_state = cancel_state;
    else begin
        next_money = money;
        next_state = state;
    end
end
```

這邊就是在 insert_state 的時候，因為還沒選飲料所以 drink 要等於 3’000，在投每個錢的時候也要判斷有沒有超過 100，另外就是 cancel 這個鍵要換 state。

```
else if(drink == 3'b001) begin
    if(money >= 7'd75) begin
        next_money = money - 7'd75;
        next_state = cancel_state;
    end
    else begin
        next_money = money;
        next_state = state;
    end
end
else if(drink == 3'b010) begin
    if(money >= 7'd50) begin
        next_money = money - 7'd50;
        next_state = cancel_state;
    end
    else begin
        next_money = money;
        next_state = state;
    end
end
```

這邊就是處理買每個飲料的時候，要先判斷錢夠不夠，不夠就 state 維持，其他飲料也是如此。

```
2'b01: begin

    if(cancnt == 32'd100000000) begin

        if(money == 7'd0) begin
            //drink = 3'b000;
            next_state = insert_state;
            next_money = money;
        end
        else begin
            next_state = state;
            next_money = money - 7'd5;
        end
         cancnt_next = 32'd0;
    end
    else begin
        cancnt_next = cancnt + 32'd1;
        next_money = money;
        next_state = state;
    end
end
default: begin
    next_money = 7'd0;
    next_state = insert_state;
end
```

這裡就是處理退錢，用一個 counter 倒數計時，這樣每一秒就會減 5，當我的 money 等於 0 就轉回原本 insert_state。

分工: 徐嘉徽寫 advanced3、4，FPGA 1

李佳栩寫 advanced1、2，FPGA 2

心得:

徐嘉徽 ：

這次的 advanced question 跟之前沒有甚麼太大的差別，所以寫起來沒有遇到太多問題，主要是第四題的 booth multiplier 要上網查很多資料才慢慢了解。比較麻煩的是 FPGA 1 要用到新教的鍵盤跟揚聲器，都要慢慢觀察之前 basic lab 的 code 去了解每個地方代表的功能是甚麼，才能改造成自己需要的功能，這次的 FPGA 是一個比較新奇的體驗 ！

李佳栩:

這次是久久以來第一次寫 FPGA，我其實遇到了很多問題，其中最大的問題還是 coding style 的問題，我常常 case 都忘記 default，導致我的 FPGA 跑不出東西，然後寫太多 if-else 導致我的變數跑來跑去，這樣也沒辦法合成。我才了解到 coding style 的重要，另外我想提出我找到的 bug，我發現在當我 95 塊錢然後要投入 50 塊的時候，我明明有寫 money + 50 要小於 100，但還是會加上去，後來我才想到因為 95+50 等於 145，已經超過我的 money 的 bit(最多到 128)，其實已經超過 100 了但是因為 bit 不夠所以錯了，最後我就把 money 多開一個 bit 就解決了，其實還有另一 bug 就是我的鍵盤只能夠買一次東西，後來發現是忘了 default，才解決問題的，verilog 的難度真的超乎我的想像，但我會加油的。