11010EECS207001Logic Design Lab

LAB5：Keyboard and Audio Modules

TEAM9

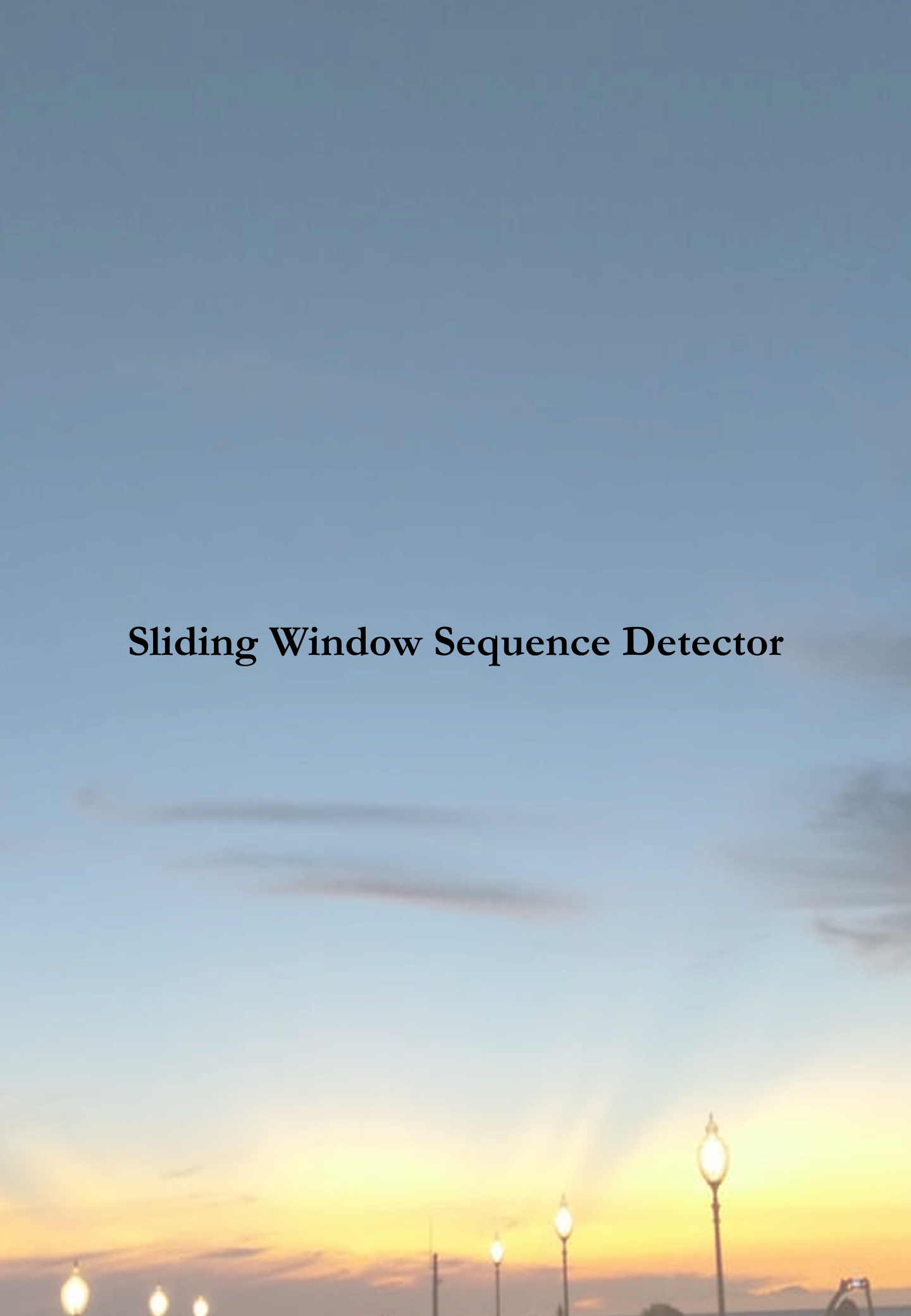
組長：108062213 顏浩昀

組員：106062304 黃鈺舒

Prof. Chun-Yi Lee

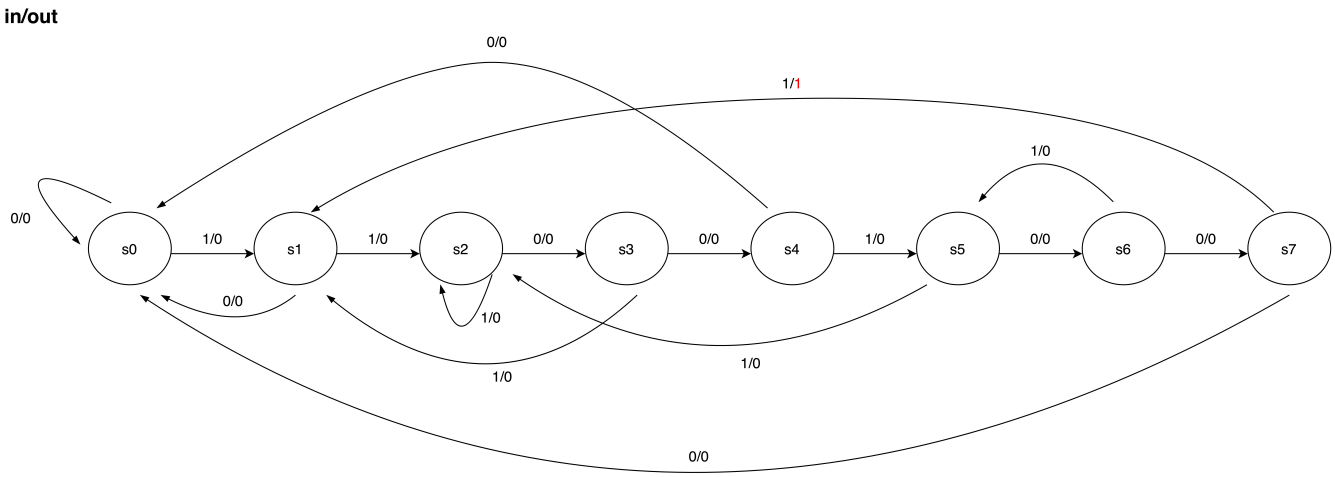
2021.11.25

Sliding Window Sequence Detector

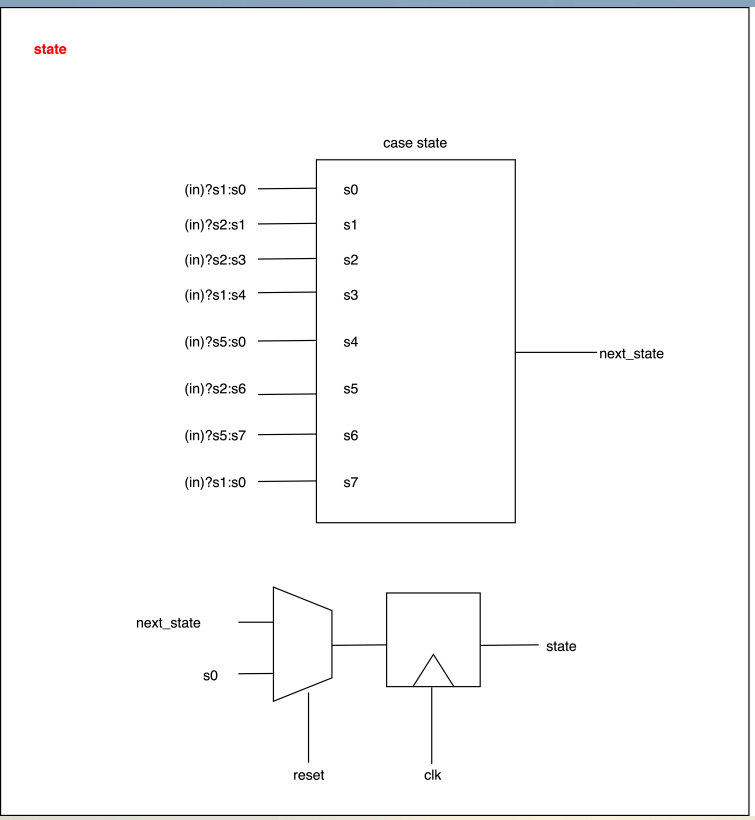


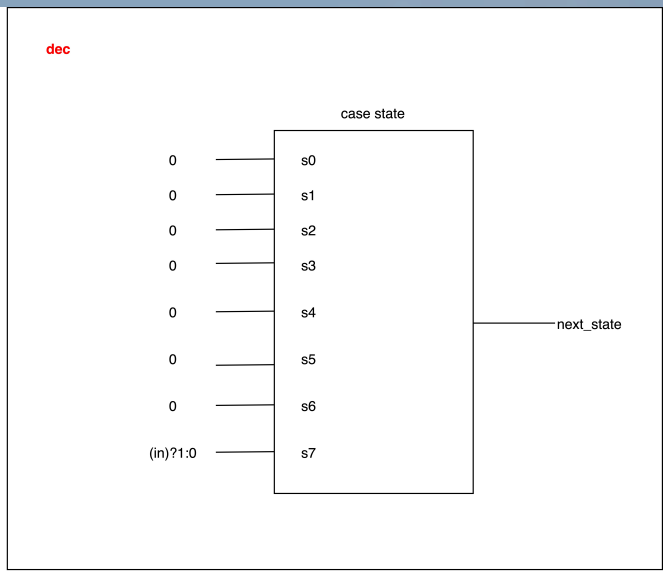
I. Diagram

(i) State diagram



(ii)電路設計圖





II. Explanation

這題是mealy machine實作，重點在於清楚判斷每一個next_state的狀況。在sequential circuit只需讓state <= next_state及reset時state為s0即可。在combinational circuit部分透過case(state)與input(in)去判斷next_state與dec值。

整體實作並不困難，state transition的流程也已經於前面的圖詳盡說明。

（下方為一小部分實作）

```
always@(posedge clk) begin
    if(!rst_n) begin
        state <= s0;
    end
    else begin
        state <= next_state;
    end
end
```

```
case(state)
    s0: begin
        if(in) begin
            next_state = s1;
            dec = 1'b0;
        end
        else begin
            next_state = s0;
            dec = 1'b0;
        end
    end
```

III. Testbench

設計幾筆測資測試不同情形：

1. 基本狀況：seq = 11001001

2. 測試即使上一組有dec=1的情形， 仍然可以接續state transition：seq = (1)_1001001

3. 測試多個10的情形：seq = (1)_100_10101010_01

4. 測試在中途reset：seq = (1)_1001, reset, _001

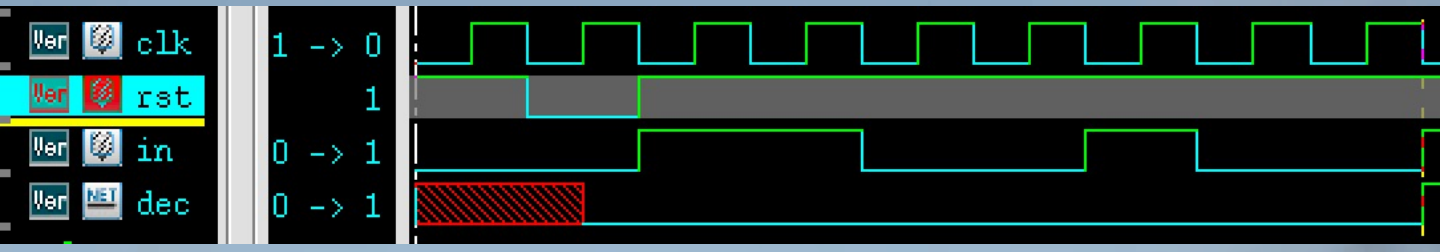
5. 測試沒有10存在：seq = (1)_1000001

6. 測試沒有1100在開頭：seq = (1)_1101001

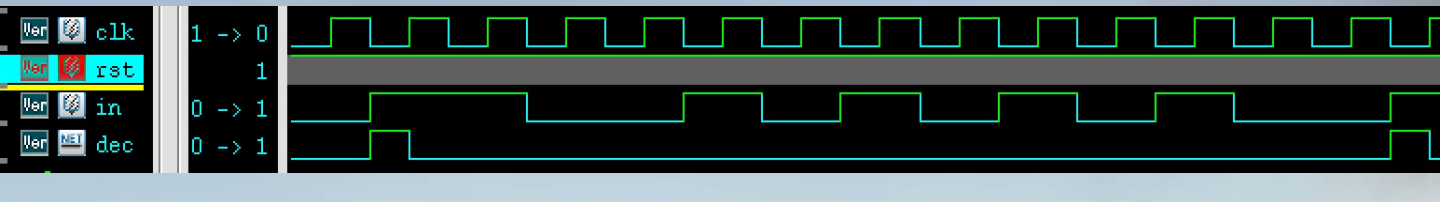
7. 測試不是01結尾：seq = (1)_1001000

附上部分測試結果

（simple test）



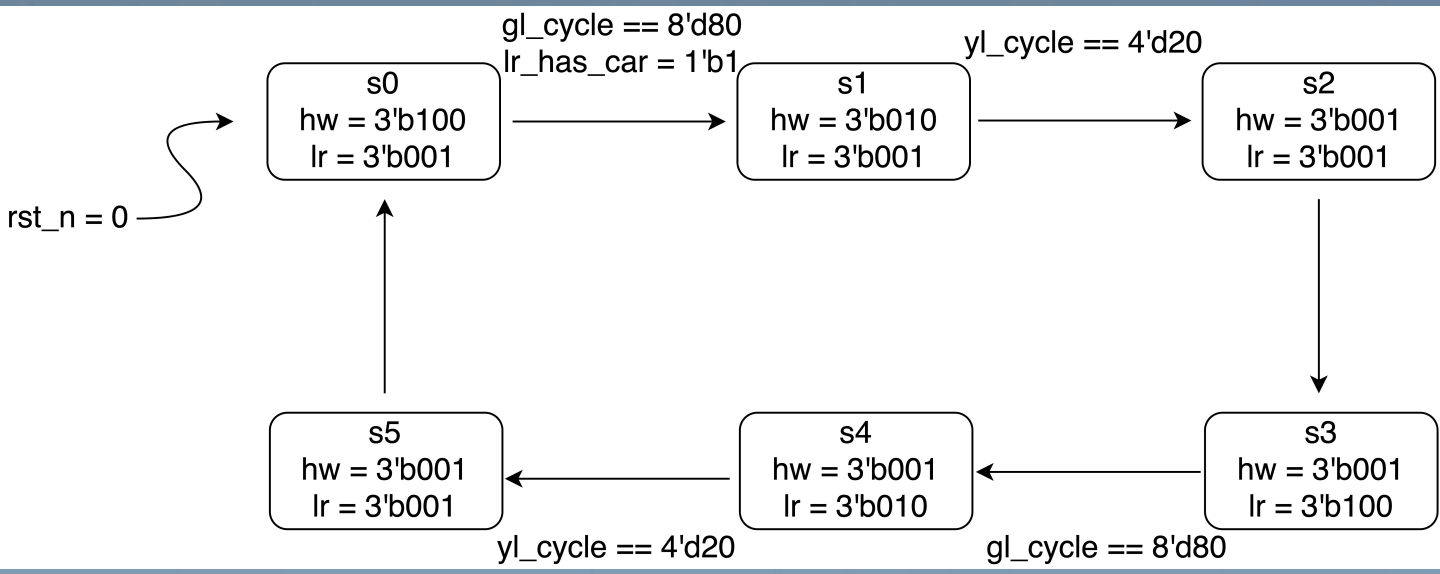
（more than one 10 exist）



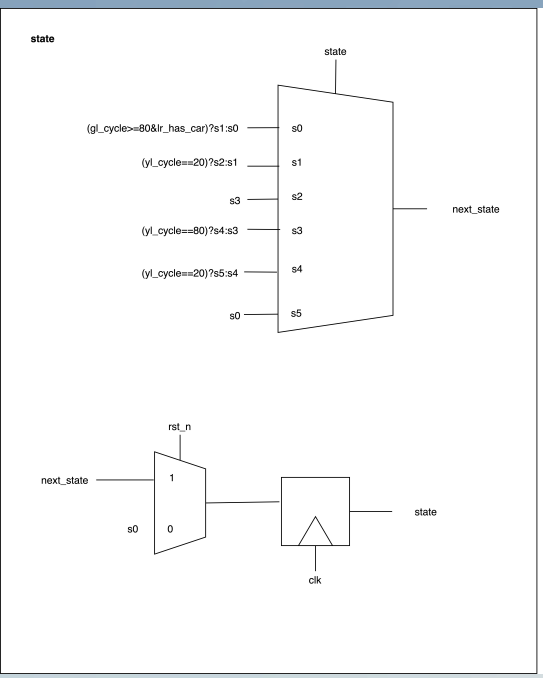
Traffic light controller

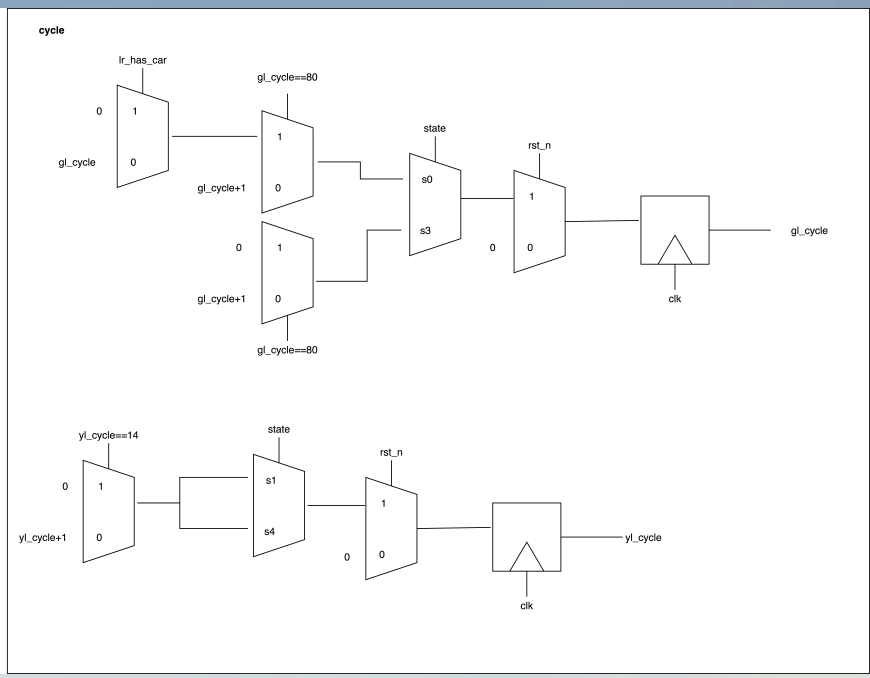
I. Diagram

(i) state transition diagram



(ii) circuit design

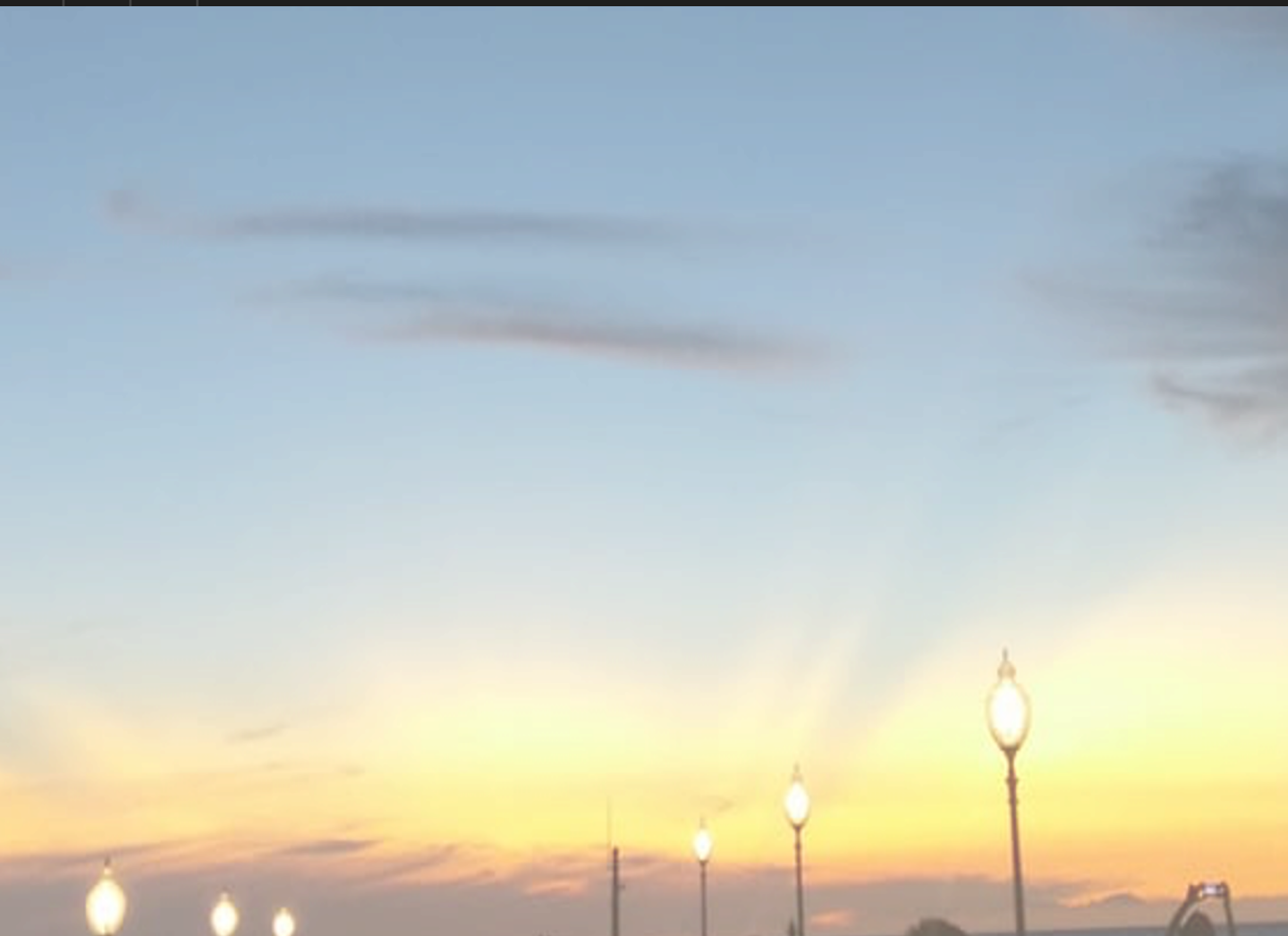




II. Explanation

本題最主要的架構是要判斷根據lr是否有車來做hw跟lr的state變化。當lr有車且hw>=80個clk就改變state，但如果hw>=80,lr沒車的話就不用改變，讓他直接hold在80以避免clk一直加上去lr卻沒車而使計算clk的reg overflow，如圖所示。接著照著實作的state diagram跑完一次循環。

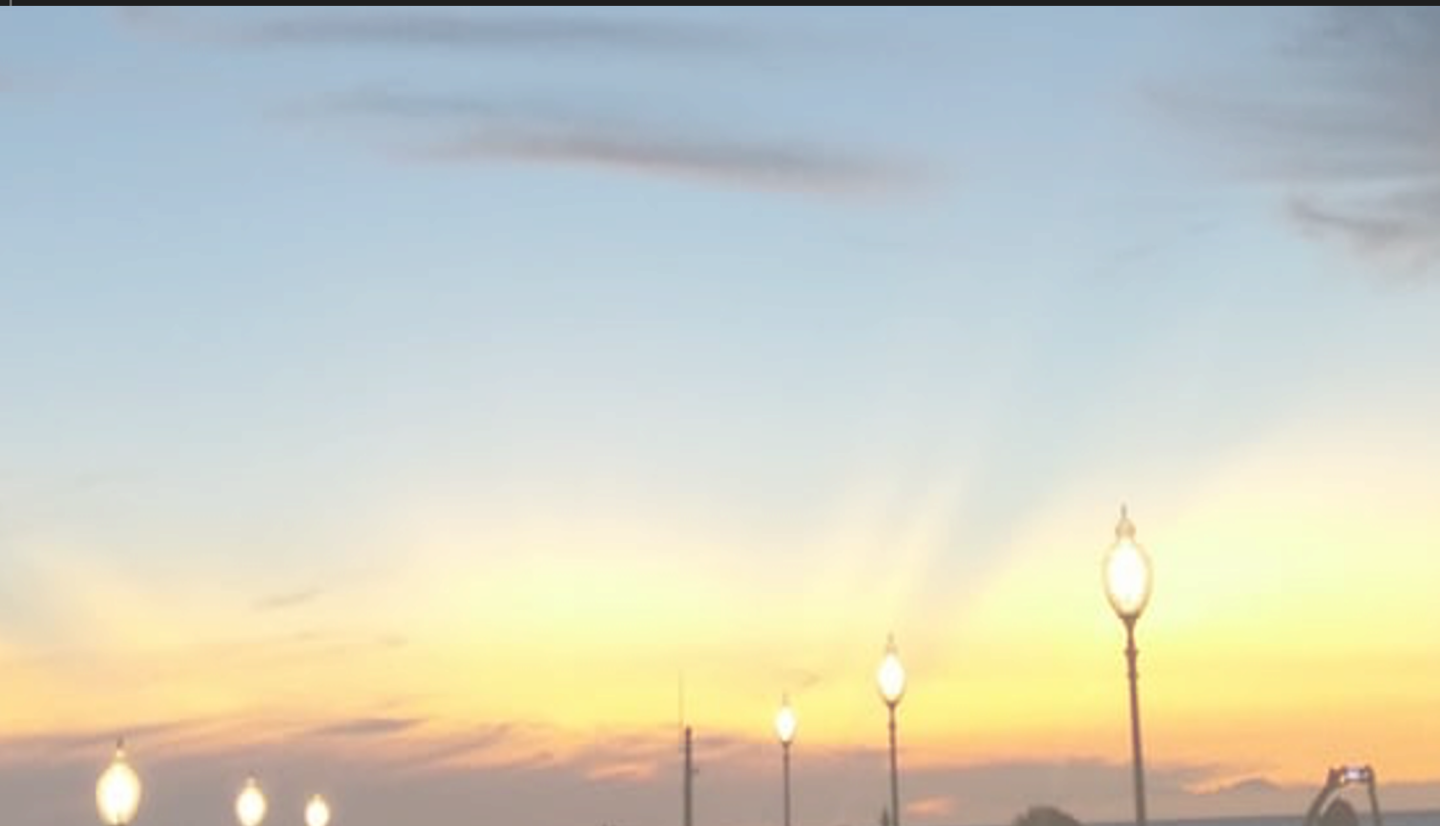
```
always @(posedge clk) begin
    case (state)
        s0:begin
            if(gl_cycle >= 8'd80 && lr_has_car==1'b1)begin
                gl_cycle <= 0;
            end
            else if(gl_cycle >= 8'd80 && lr_has_car==1'b0)begin
                gl_cycle <= 8'd80;
            end
            else begin
                gl_cycle <= gl_cycle +1;
            end
        end
```



III.testbench

本題測試分為三階段，首先測試小於80個clk但lr有車時hw是否會改變state(變黃燈)，第二階段測試大於80個state，lr有車時是否可以偵測到，最後測試當clk大於256時，計算greenlight 持續clk數的reg是否會因overflow而變成0，導致新的循環小於80個clk所以沒有偵測到lr有車而換state。

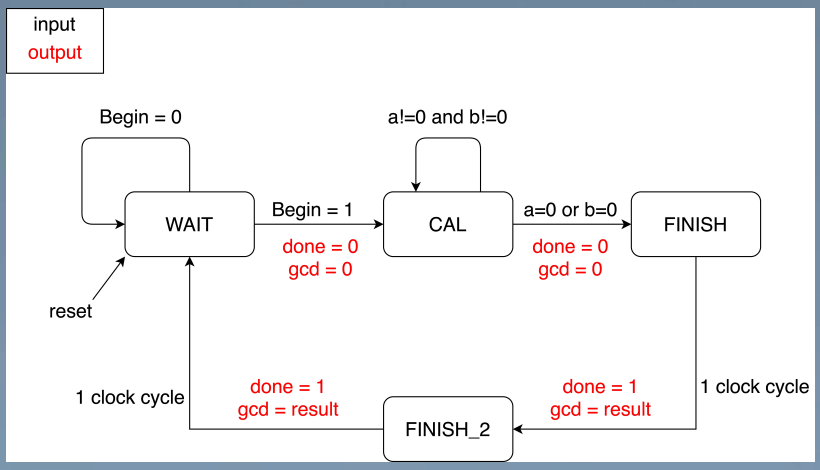
```
    # (70*`CYC);
    car = 1'b1;  // has car but < 80;
    # (31*`CYC); //lr = green but don't has car
    car = 1'b0;
    # (101*`CYC); //一個循環

    # (100*`CYC); // <80 但沒車
    car = 1'b1; // has car;//應該變黃燈
    # (21*`CYC); // lr_gl;
    car = 1'b0; // no car;
    # (101*`CYC);

    # (100*`CYC); // <256，因為greenlight的reg開八個，檢查是否會overflow而沒有換state
    car = 1'b1; // has car;//應該變黃燈
    # (21*`CYC); // lr_gl;
    car = 1'b0; // no car;
    # (101*`CYC);
```



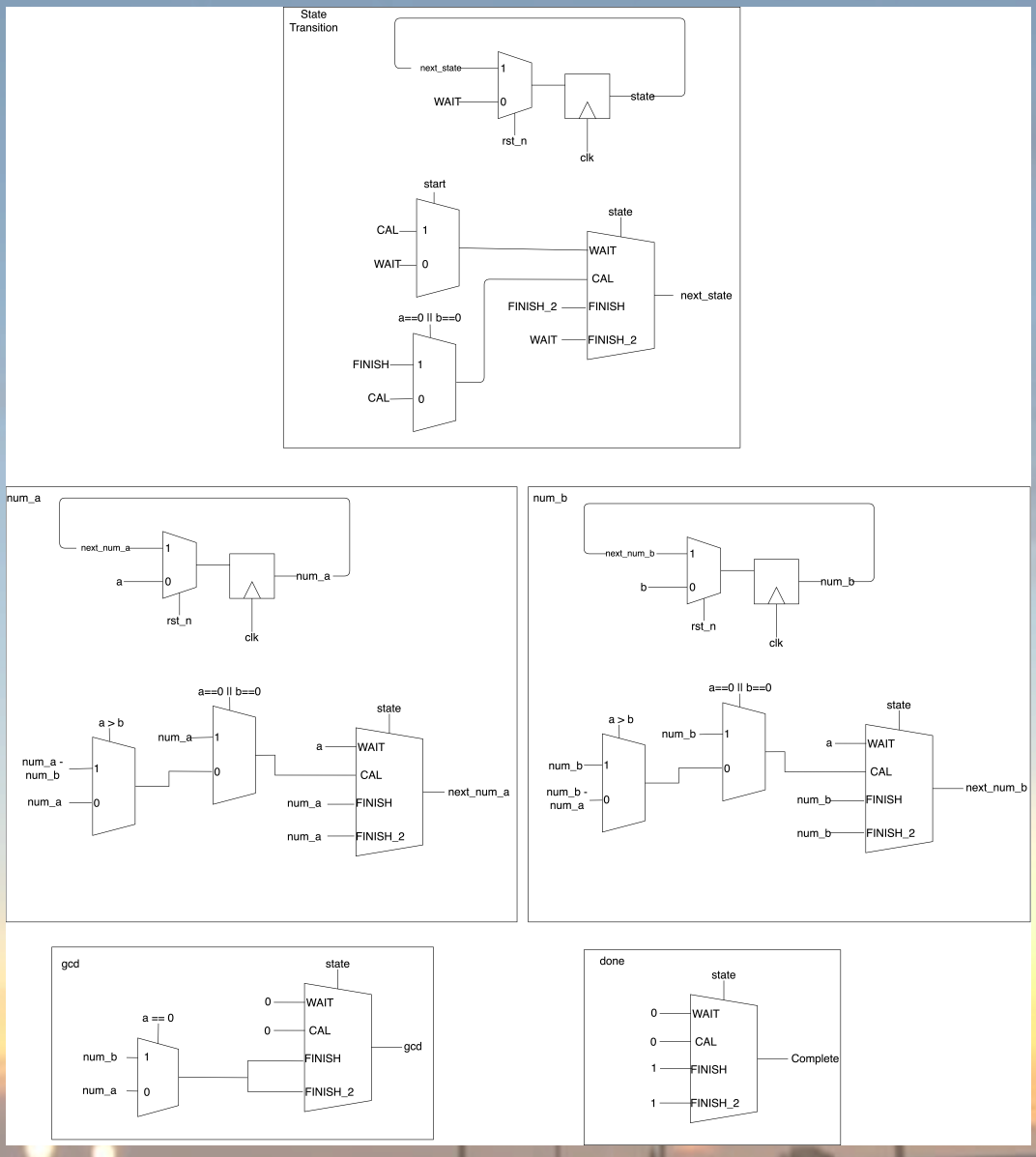
Greatest Common Divisor

I. Diagram

(i) State diagram



(ii)電路設計圖



II. Explanation

本題設計分為sequential circuit與combinational circuit兩部分。

Sequential circuit將state及num_a, num_b設為next值，並在 reset時分別設為WAIT及a, b

```
always @(posedge clk) begin
    if(!rst_n) begin
        state <= WAIT;
        num_a <= a;
        num_b <= b;
    end
    else begin
        state <= next_state;
        num_a <= next_a;
        num_b <= next_b;
    end
end
```

Combinational part用case分別討論state為WAIT, CAL, FINISH (FINISH_2)時的狀況。首先是WAIT，在這個state，兩個output (done 與 gcd) 皆為 0，next_num_a, next_num_b分別為a, b，而next_state要根據start去做判斷。若start = 1, next_state = CAL，若start = 0, next_state = WAIT。

接著討論state CAL。在這個state，兩個output (done 與 gcd) 皆為 0。然後我們要運算gcd，因此值會依據現在的num_a, num_b有不同的情形。如果num_a, num_b任一者為0，則next_state = FINISH，並且next_num_a, next_num_a維持 num_a, num_b；反之，則next_state = state，並比較num_a, num_b的大小，將大的數設為“大數-小數”。

```
if(num_a == 16'b0 || num_b == 16'b0) begin
    next_state = FINISH;
    next_a = num_a;
    next_b = num_b;
end
else begin
    next_state = CAL;
    if(num_a > num_b) begin
        next_a = num_a - num_b;
        next_b = num_b;
    end
    else begin
        next_a = num_a;
        next_b = num_b - num_a;
    end
end
```

最後是state = FINISH的狀況，此時會將 output (done) 改為1，並且output (gcd) 會等於num_a, num_b中不為0的數。然後由於題目要求在求出結果後要維持兩個clock，因此在這裡多設計一個state為FINISH_2，讓state為FINISH時next_state = FINISH_2，用來hold output value，且當state = FINISH_2時next_state = WAIT。

III. Testbench

在reset後開始測試測資，每筆測資皆在negedge clk時給值，並透過done==1確保給值時上一組測資已經完成。

下列為測資列表：

1. 測試gcd為a的情形：a=5, b=200

2. 測試gcd為b的情形：a=63, b=7

3. 測試gcd為1且兩數皆為質數：a=97, b=11

4. 測試gcd為1且兩數皆非質數：a=25, b=192

5. 測試start=0不會進入state CAL，在之後delay 20ns（10個clock cycle可以保證a=2,b=1應該要計算完成）

6. 測試一些隨機較大的數，透過迴圈從a=35747,b=21483開始，每次a+=212,b+=307，共經過50次。將結果用$display方式印出，並透過C++17的std::gcd進行相同的測資比較。

（下圖為display的順序、verilog測資與c++測資）

```
task test;
    $display("a=%d b=%d gcd=%d done=%b", a, b, gcd, done);
endtask
```

```
    @(negedge clk)
        start = 1;
        a = 16'd35747;
        b = 16'd21483;
repeat(50) begin
    @(done == 1) begin
    test;
    #4;
    @(negedge clk)
        a = a + 16'd212;
        b = b + 16'd307;
end
end
```

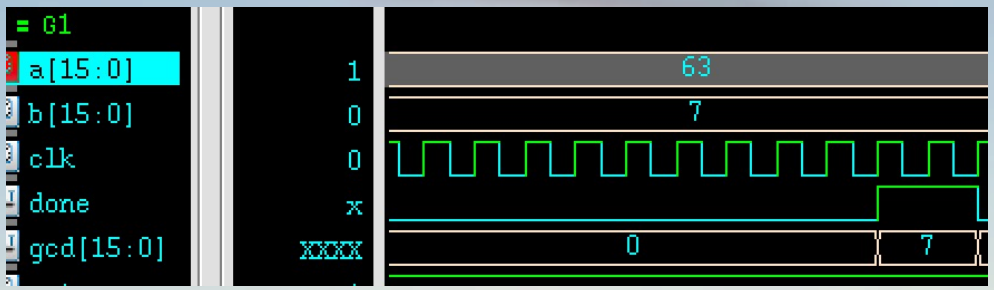
```
#include<iostream>
#include <numeric>

using namespace std;

int main(){

    int a = 35747, b = 21483, c;
    c = 50;
    while(c--) {
        std::cout<<"a="<<a<<" b="<<b<<" gcd="<<gcd(a,b)<<std::endl;
    a+=212;
    b+=307;
    }

    return 0;
}
```

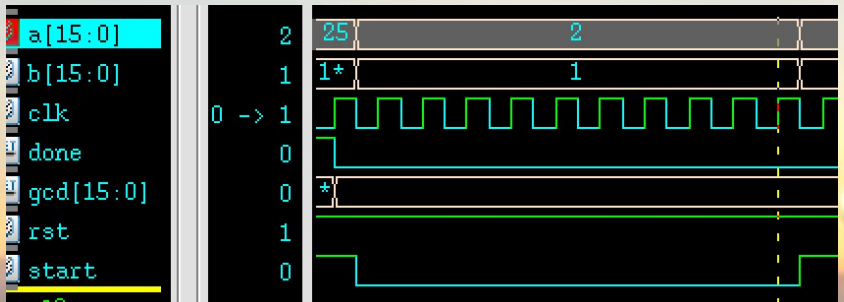
III. Testbench

（左半部為c++結果，右半部為verilog結果）

```
a=35747 b=21483 gcd=1     a=35747 b=21483 gcd=    1 done=1
a=35959 b=21790 gcd=1     a=35959 b=21790 gcd=    1 done=1
a=36171 b=22097 gcd=1     a=36171 b=22097 gcd=    1 done=1
a=36383 b=22404 gcd=1     a=36383 b=22404 gcd=    1 done=1
a=36595 b=22711 gcd=13    a=36595 b=22711 gcd=   13 done=1
a=36807 b=23018 gcd=1     a=36807 b=23018 gcd=    1 done=1
a=37019 b=23325 gcd=1     a=37019 b=23325 gcd=    1 done=1
a=37231 b=23632 gcd=1     a=37231 b=23632 gcd=    1 done=1
a=37443 b=23939 gcd=1     a=37443 b=23939 gcd=    1 done=1
a=37655 b=24246 gcd=1     a=37655 b=24246 gcd=    1 done=1
a=37867 b=24553 gcd=1     a=37867 b=24553 gcd=    1 done=1
a=38079 b=24860 gcd=1     a=38079 b=24860 gcd=    1 done=1
a=38291 b=25167 gcd=1     a=38291 b=25167 gcd=    1 done=1
a=38503 b=25474 gcd=1     a=38503 b=25474 gcd=    1 done=1
a=38715 b=25781 gcd=29    a=38715 b=25781 gcd=   29 done=1
a=38927 b=26088 gcd=1     a=38927 b=26088 gcd=    1 done=1
a=39139 b=26395 gcd=1     a=39139 b=26395 gcd=    1 done=1
a=39351 b=26702 gcd=13    a=39351 b=26702 gcd=   13 done=1
a=39563 b=27009 gcd=1     a=39563 b=27009 gcd=    1 done=1
a=39775 b=27316 gcd=1     a=39775 b=27316 gcd=    1 done=1
a=39987 b=27623 gcd=1     a=39987 b=27623 gcd=    1 done=1
a=40199 b=27930 gcd=1     a=40199 b=27930 gcd=    1 done=1
a=40411 b=28237 gcd=1     a=40411 b=28237 gcd=    1 done=1
a=40623 b=28544 gcd=1     a=40623 b=28544 gcd=    1 done=1
a=40835 b=28851 gcd=1     a=40835 b=28851 gcd=    1 done=1
a=41047 b=29158 gcd=1     a=41047 b=29158 gcd=    1 done=1
a=41259 b=29465 gcd=1     a=41259 b=29465 gcd=    1 done=1
a=41471 b=29772 gcd=1     a=41471 b=29772 gcd=    1 done=1
a=41683 b=30079 gcd=1     a=41683 b=30079 gcd=    1 done=1
a=41895 b=30386 gcd=1     a=41895 b=30386 gcd=    1 done=1
a=42107 b=30693 gcd=13    a=42107 b=30693 gcd=   13 done=1
a=42319 b=31000 gcd=1     a=42319 b=31000 gcd=    1 done=1
a=42531 b=31307 gcd=1     a=42531 b=31307 gcd=    1 done=1
a=42743 b=31614 gcd=1     a=42743 b=31614 gcd=    1 done=1
a=42955 b=31921 gcd=1     a=42955 b=31921 gcd=    1 done=1
a=43167 b=32228 gcd=1     a=43167 b=32228 gcd=    1 done=1
a=43379 b=32535 gcd=1     a=43379 b=32535 gcd=    1 done=1
a=43591 b=32842 gcd=1     a=43591 b=32842 gcd=    1 done=1
a=43803 b=33149 gcd=1     a=43803 b=33149 gcd=    1 done=1
a=44015 b=33456 gcd=1     a=44015 b=33456 gcd=    1 done=1
a=44227 b=33763 gcd=1     a=44227 b=33763 gcd=    1 done=1
a=44439 b=34070 gcd=1     a=44439 b=34070 gcd=    1 done=1
a=44651 b=34377 gcd=1     a=44651 b=34377 gcd=    1 done=1
a=44863 b=34684 gcd=377   a=44863 b=34684 gcd=  377 done=1
a=45075 b=34991 gcd=1     a=45075 b=34991 gcd=    1 done=1
a=45287 b=35298 gcd=1     a=45287 b=35298 gcd=    1 done=1
a=45499 b=35605 gcd=1     a=45499 b=35605 gcd=    1 done=1
a=45711 b=35912 gcd=1     a=45711 b=35912 gcd=    1 done=1
a=45923 b=36219 gcd=1     a=45923 b=36219 gcd=    1 done=1
a=46135 b=36526 gcd=1     a=46135 b=36526 gcd=    1 done=1
logout                    Simulation complete via $finish(1) at time 14102 NS + 0
Saving session...         ./Lab5_Team9_Greatest_Common_Divisor_t.v:74        #2 $finish;
```

（下方波形確認finish hold 2 clock cycles及start=0不作運算）

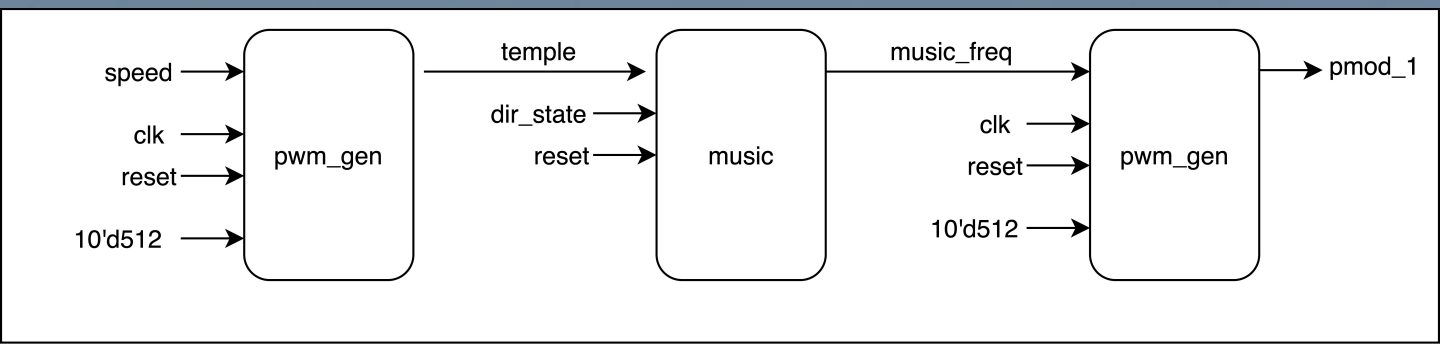


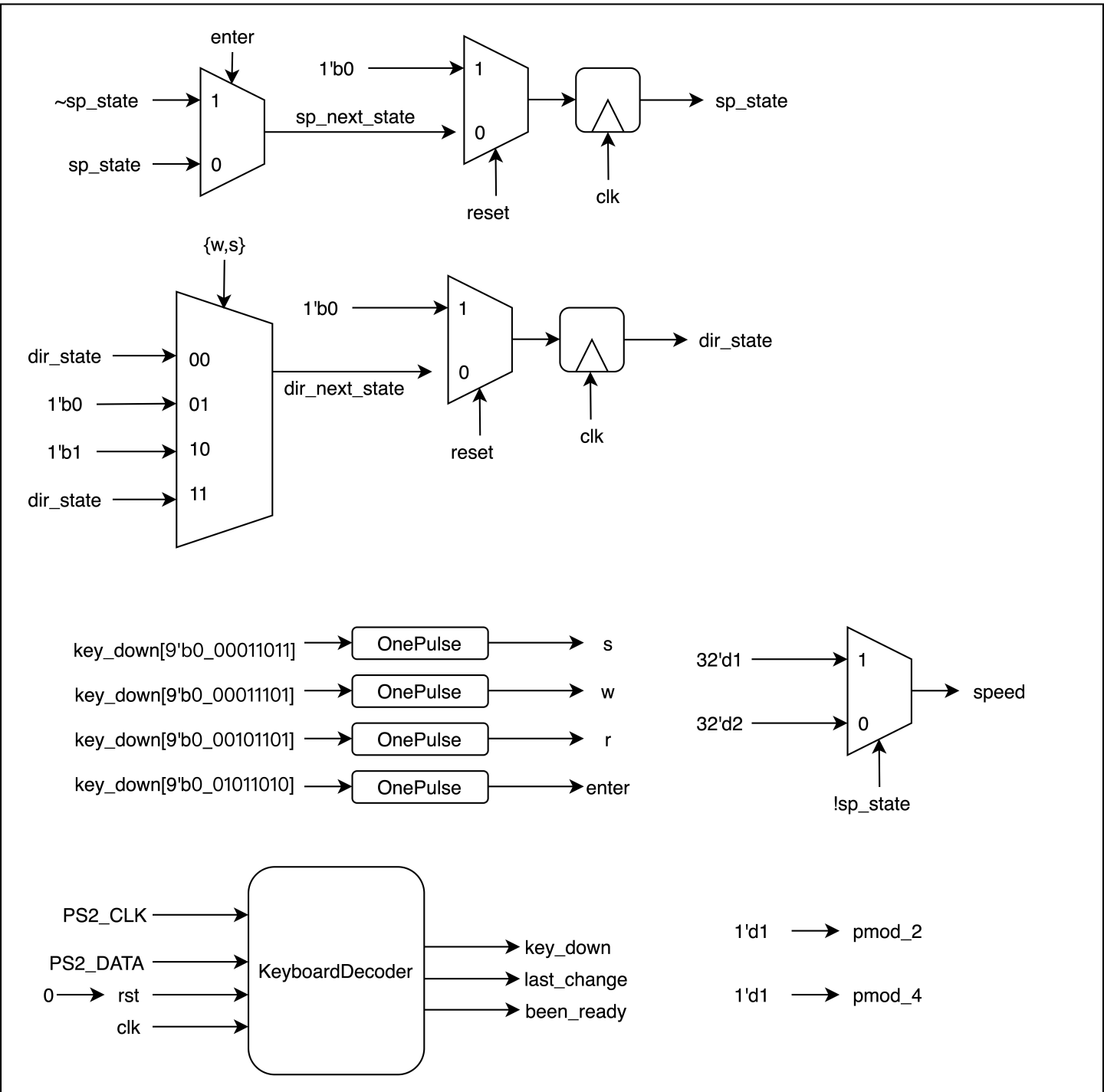


Music FPGA

I. Diagram

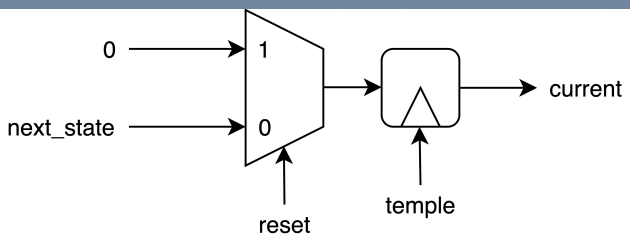
(i) Block diagram

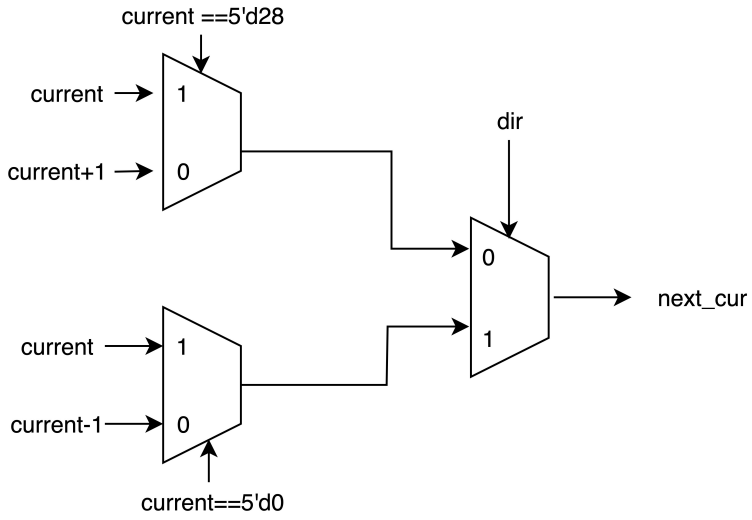


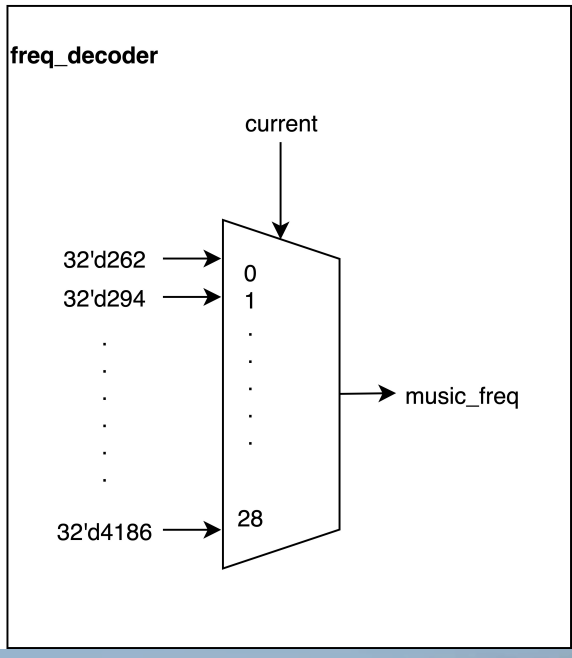


I. Diagram

(ii)music module circuit design







II. Explanation

在Top module中，負責各個按鍵（keyboard）的Onepulse處理，以及連接keyboard、2xPWM與music module。Top module還有負責速度控制的部分，透過state轉換改變速度要求，並將我們想要的速度頻率傳入第一個PWM製作合適的頻率。此外還有選擇方向，同樣透過state轉換判斷音階向上或向下，然後傳入music module。

```
always @(*) begin
    if(r) begin
        next_speed_state = ~speed_state;
    end
    else begin
        next_speed_state = speed_state;
    end

    if(w==1'b1 && s==1'b0) begin
        next_dir_state = 1'b0;
    end
    else if (w==1'b0 && s==1'b1) begin
        next_dir_state = 1'b1;
    end
    else begin
        next_dir_state = direction_state;
    end
end
```

```
assign button_s = (key_down[9'b0_00011011]) ? 1'b1:1'b0;
assign button_w = (key_down[9'b0_00011101]) ? 1'b1:1'b0;
assign button_r = (key_down[9'b0_00101101]) ? 1'b1:1'b0;
assign button_enter = (key_down[9'b0_01011010]) ? 1'b1:1'b0;
```

```
PWM_gen speed_pwm( .clk(clk), .reset(reset), .freq(speed), .duty(10'd512), .PWM(temple) );
OnePulse s_op( .signal_single_pulse(s), .signal(button_s), .clock(clk) );
OnePulse w_op( .signal_single_pulse(w), .signal(button_w), .clock(clk) );
OnePulse r_op( .signal_single_pulse(r), .signal(button_r), .clock(clk) );
OnePulse reset_op( .signal_single_pulse(reset), .signal(button_enter), .clock(clk) );
OnePulse reset_14_op( .signal_single_pulse(reset_14), .signal(button_enter), .clock(clk_14) );
```

除了Top module外，這次實作的部分還有music module。這個module是要判斷現在這時刻邀輸出哪個頻率的聲音。透過第一個PWM做出的頻率當作改變音階的頻率，然後用Top module中的direction判斷向上或向下。然後將current的值放入decoder中得出music frequency。最後這個music frequency會透過Top module再接入第二個PWM中，並接上音源裝置輸出。

```
always @(*) begin
    if(!dir) begin
        if(current == 5'd28) begin
            next_cur = current;
        end
        else begin
            next_cur = current + 5'd1;
        end
    end
    else begin
        if(current == 5'd0) begin
            next_cur = current;
        end
        else begin
            next_cur = current - 5'd1;
        end
    end
end
```

```
module freq_decoder (…
);

    always @(*) begin
        case(current)
            0  : freq = 32'd262; //C4
            1  : freq = 32'd294;
            2  : freq = 32'd330;
            3  : freq = 32'd349;
            4  : freq = 32'd392;
            5  : freq = 32'd440;
            6  : freq = 32'd494;

            7  : freq = 32'd523; //C5
            8  : freq = 32'd587;
```

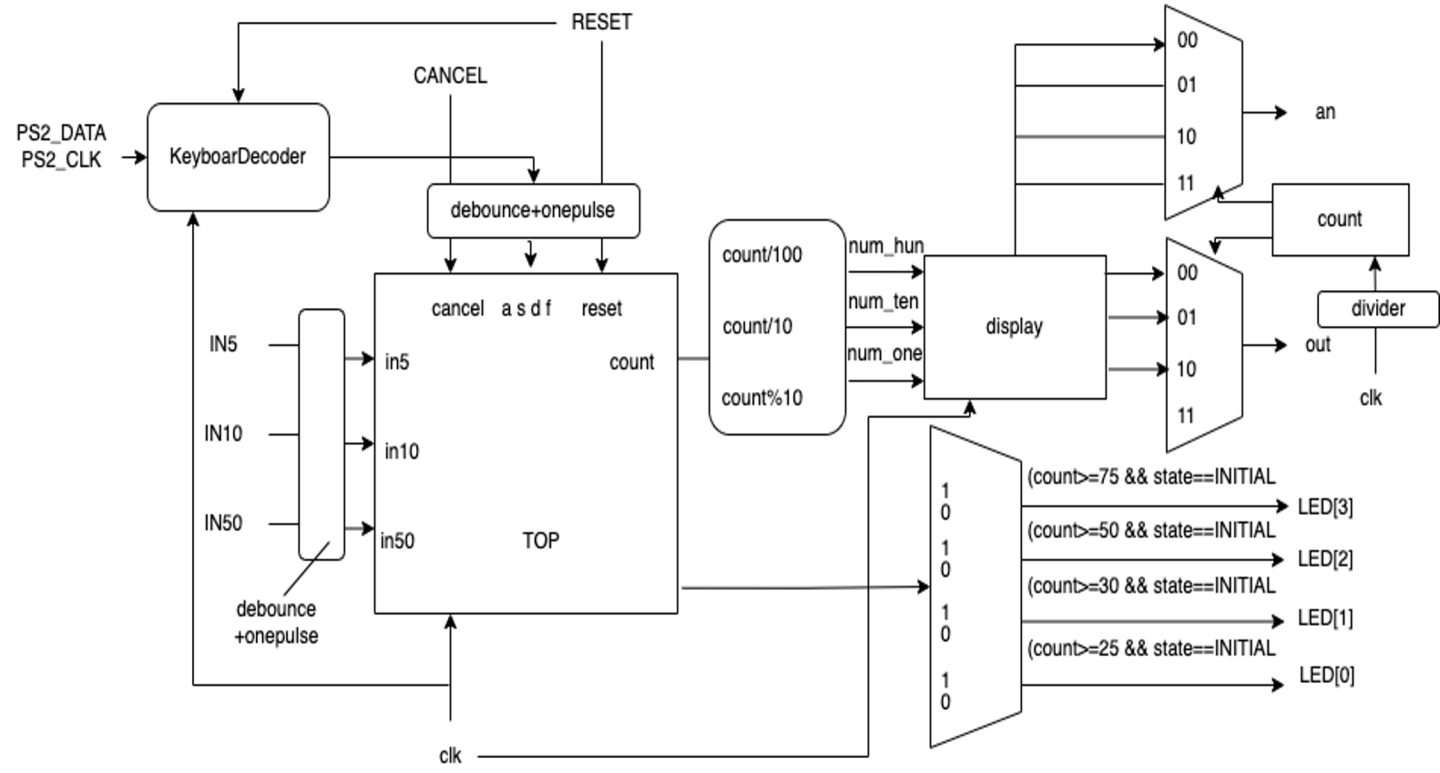
```
music gen_tone( .clk(temple), .reset(reset_14), .dir(direction_state), .key_freq(key_freq));
PWM_gen tone_pwm( .clk(clk), .reset(reset), .freq(key_freq), .duty(10'd512), .PWM(pmod_1) );
```

Vending machine FPGA

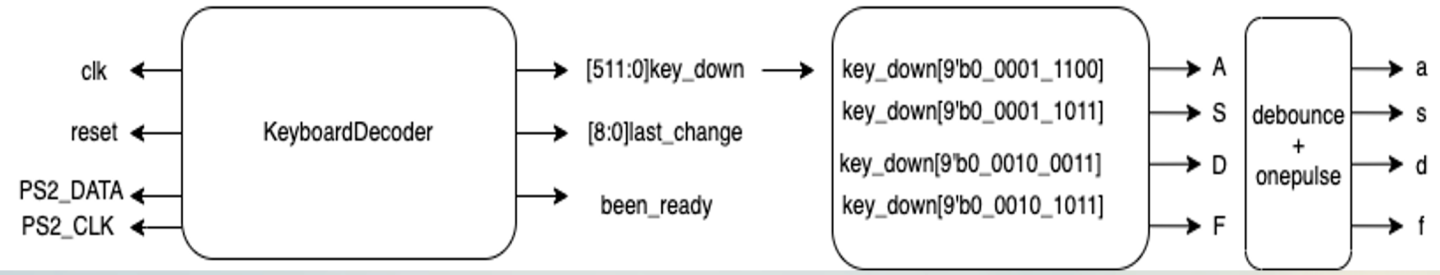
I. Diagram

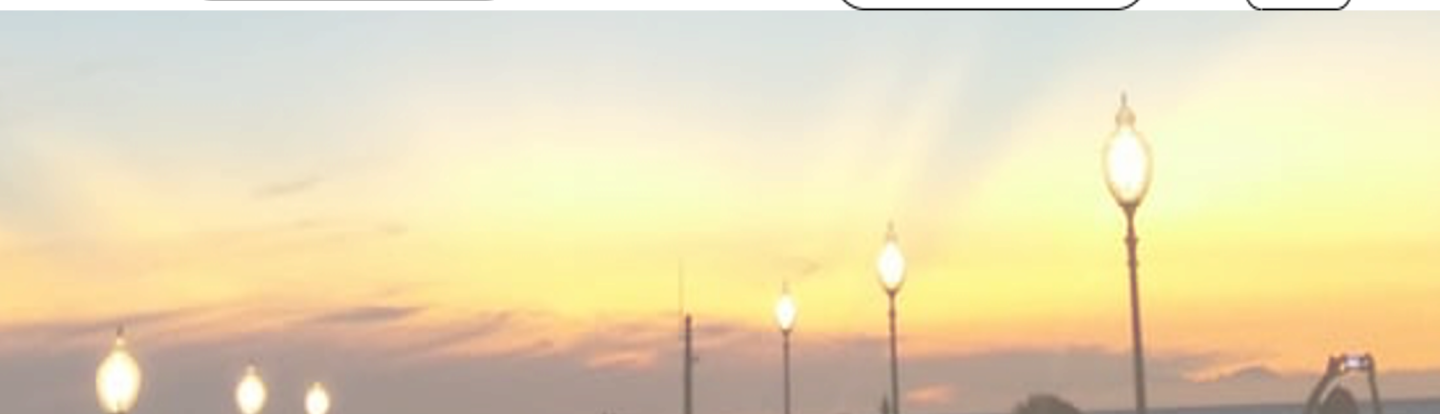
(i) 電路設計圖

TOP



KeyboardDecoder

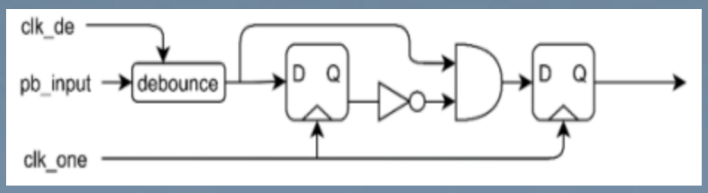




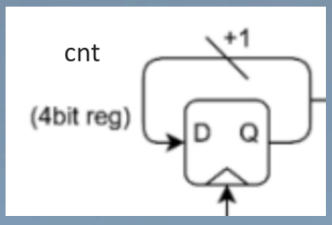
I. Diagram

(i) 電路設計圖

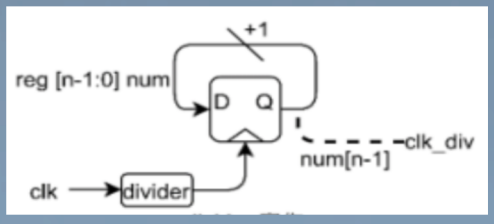
debounce



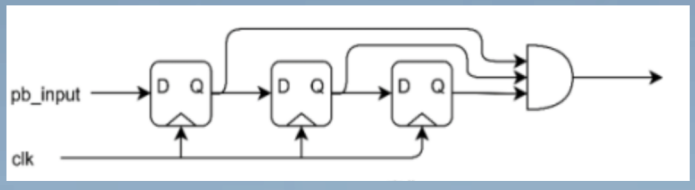
count



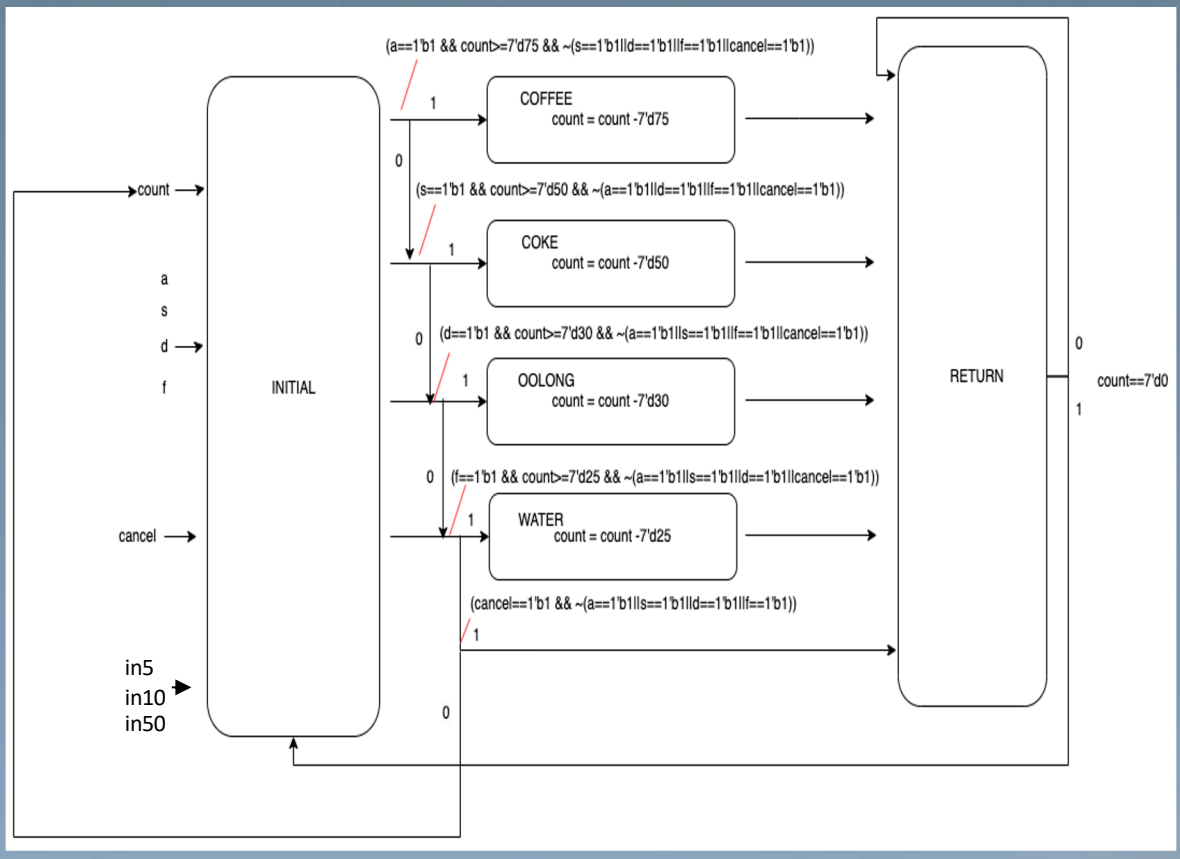
divider

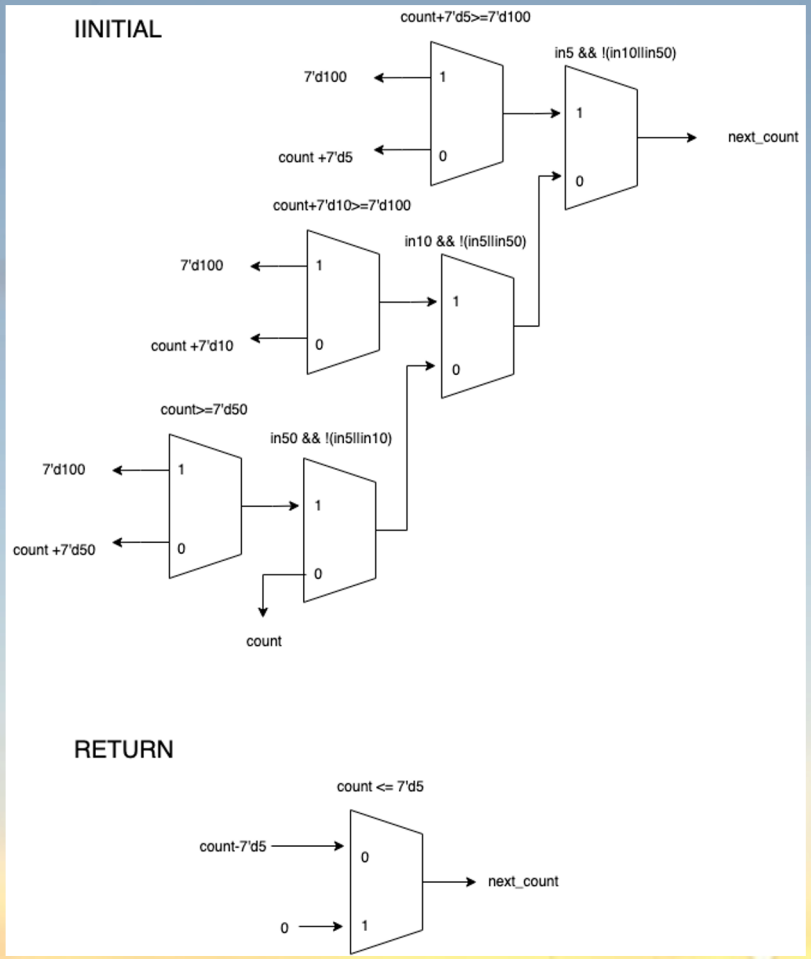


onepulse

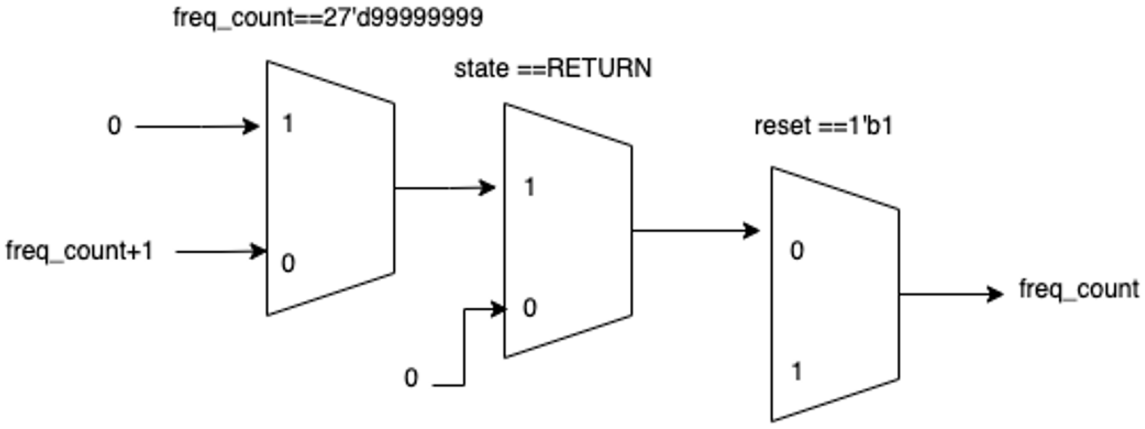


State

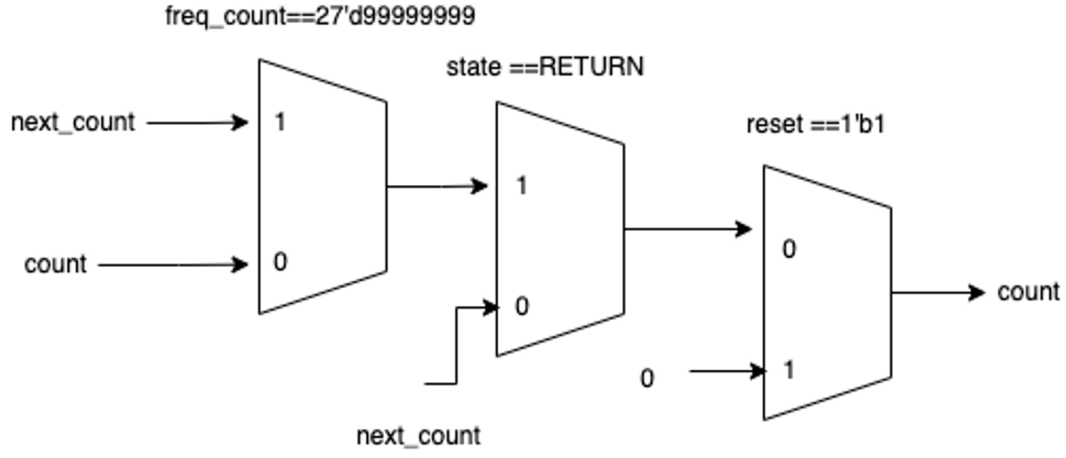

(a==1'b1 && count>=7'd75 && ~(s==1'b1||d==1'b1||f==1'b1||cancel==1'b1))
COFFEE
count = count -7'd75
(s==1'b1 && count>=7'd50 && ~(a==1'b1||d==1'b1||f==1'b1||cancel==1'b1))
COKE
count = count -7'd50
(d==1'b1 && count>=7'd30 && ~(a==1'b1||s==1'b1||f==1'b1||cancel==1'b1))
OOLONG
count = count -7'd30
(f==1'b1 && count>=7'd25 && ~(a==1'b1||s==1'b1||d==1'b1||cancel==1'b1))
WATER
count = count -7'd25
(cancel==1'b1 && ~(a==1'b1||s==1'b1||d==1'b1||f==1'b1))
count
a
s
d
f
cancel
in5
in10
in50
INITIAL
RETURN
count==7'd0


IINITIAL
count+7'd5>=7'd100
7'd100
count +7'd5
in5 && !(in10||in50)
next_count
count+7'd10>=7'd100
7'd100
count +7'd10
in10 && !(in5||in50)
count>=7'd50
7'd100
count +7'd50
in50 && !(in5||in10)
count
RETURN
count <= 7'd5
count-7'd5
0
next_count

freq_count



count



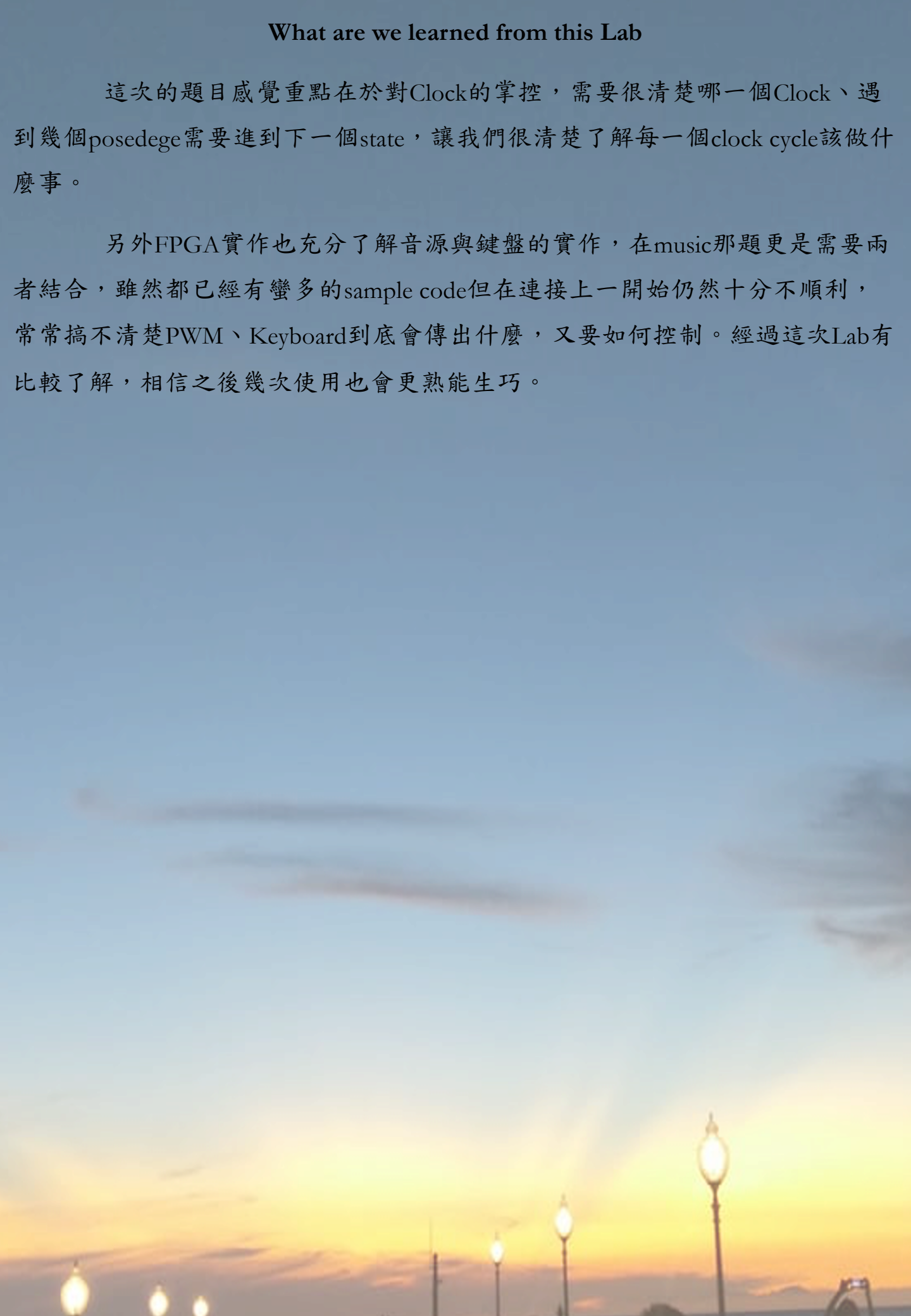
II. Explanation

本次fpga較為複雜，除了電路圖包含許多module之外，State Diagram 中 count 的給值與計算步驟也相當多。最一開始input會藉由in5,in10,in50輸入，若錢不足夠或沒有選擇飲料，就會回到initial持續購買得動作，在這中間都可以按下button輸入硬幣(錢)，上限為100。當累加起來的錢足夠買飲料時，就可以用鍵盤輸入a,s,d,f變換state，在條件允許買飲料且選擇飲料之後，也就是第二頁INITIAL所畫，會變成coffe,code,oolong,water四種飲料的state，同時count會減掉對應飲料品項的價錢，output到RETURN state，因為一次只能買一種飲料。在RETURN state中，count會每次退五塊錢(如第二頁)，我們用frequent count來計算，當count數到我們設定的週期後，才會將nextcount餵進count中，外面有包一層if判斷是不是return state，確保只有return會做frequent的運算，其他count會直接給nextcount(如第三頁所示)。return在扣完錢之前會一直停在return，當錢全退光後回到initial，繼續下一輪買飲料的循環。接板子的部分有使用17倍clk接7segment使價錢可以顯示個位數到百位數，數字是用當前count粉別除以100的商數、除以10的商數以及餘數傳入display中，再用輸入的數字判斷現在7segment該給的值，再用一個四倍器分成四段給an值跟out值，將現在價錢顯示在板子上，同時判斷現在count購買哪些飲料來輸出led四個bit的1或0，如第三頁圖，在過程中只要按下rst鈕，count會直接歸零，若按下cancel鈕且沒有選飲料會回到initial state。

What are we learned from this Lab

這次的題目感覺重點在於對Clock的掌控，需要很清楚哪一個Clock、遇到幾個posedege需要進到下一個state，讓我們很清楚了解每一個clock cycle該做什麼事。

另外FPGA實作也充分了解音源與鍵盤的實作，在music那題更是需要兩者結合，雖然都已經有蠻多的sample code但在連接上一開始仍然十分不順利，常常搞不清楚PWM、Keyboard到底會傳出什麼，又要如何控制。經過這次Lab有比較了解，相信之後幾次使用也會更熟能生巧。



Cooperation

108062213 顏浩昀：

sequence detector實作、電路圖繪製、testbench製作、traffic light 電路圖繪製、gcd實作、電路圖繪製、testbench製作、music FPGA實作、電路圖繪製、Report製作

106062304 黃鈺舒 ：

sequence detector 實作、Traffic Light實作、testbench製作、vending machine FPGA實作、電路圖繪製、Report製作