LAB4

第七組

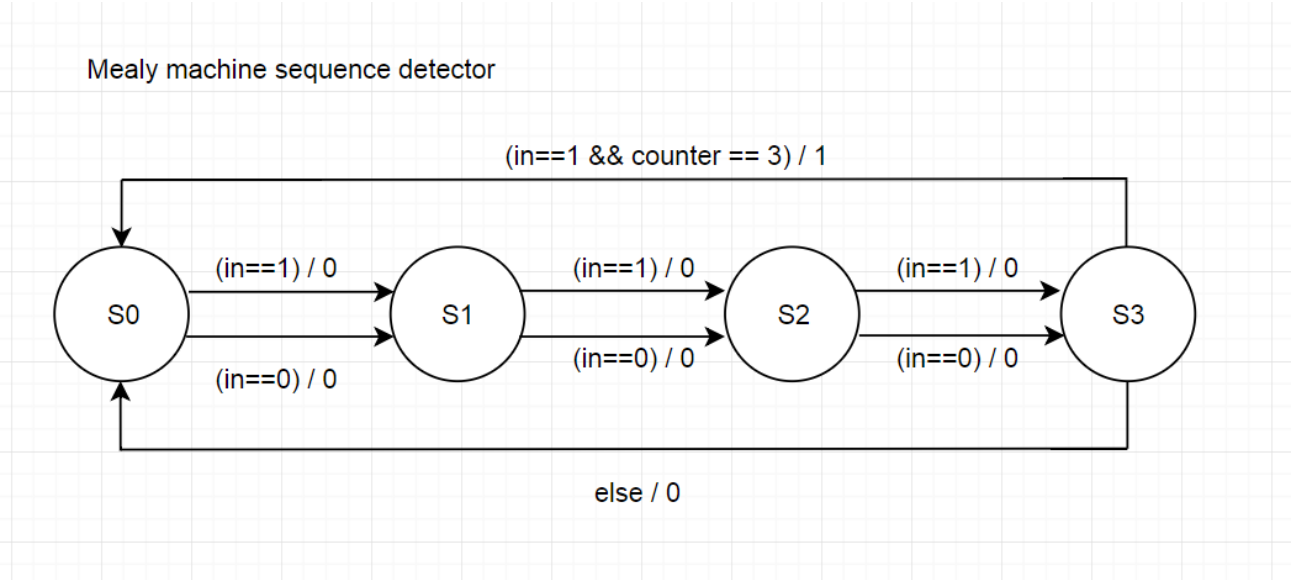
黎佑廷 105030009

郭家瑋 105030015

VERILOG QUESTION 1:

MEALY MACHINE SEQUENCE DETECTOR

✓ State-transition diagram:



當 state == S0 時，不管怎樣都是輸出 0，而且下一個 state 一定是 S1，只是當 in == 1 時，我將 counter + 1；

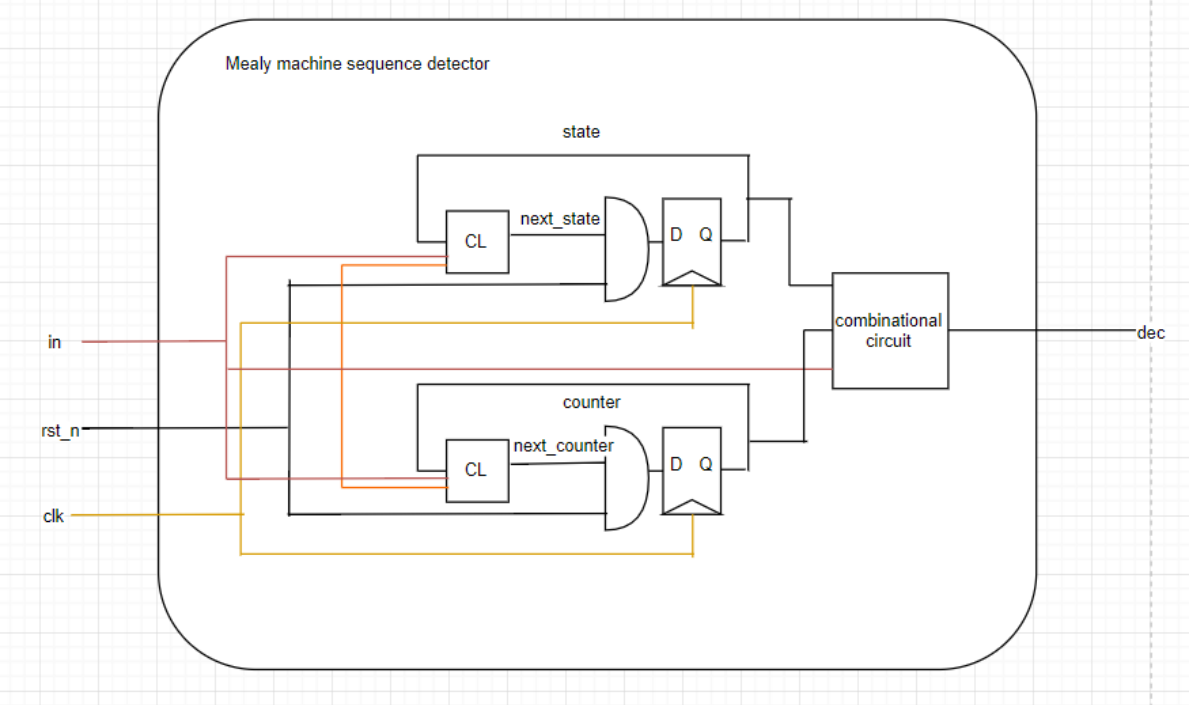
當 state == S1 時，也是不管怎樣都輸出 0，而且下一個 state 一定是 S2，只是當 in == 0 時，我將 counter + 1；

當 state == S2 時，也是不管怎樣都輸出 0，而且下一個 state 一定是 S3，只是當 in == 0 時，我將 counter + 1；

當 state == S1 時，如果 in == 1 而且 counter == 3，代表已經偵測到 1001 這個 pattern，所以輸出 1，其他的狀況則是輸出 0，下一個 state 回到 S0 以繼續偵測接下來的 pattern。

因為無論 input 為何，state 的變換都是 S0->S1->S2->S3 所以每四個 bit 都會重新 detect 一次。

✓ Block Diagram



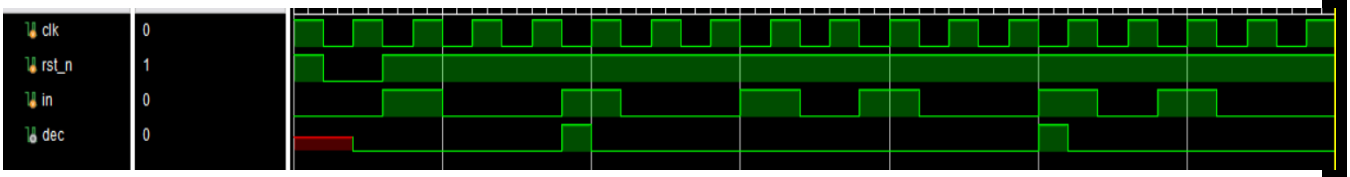
這題我所使用的驗證方法是給予跟老師投影片一樣的 input，看看出來的 waveform 有沒有跟投影片上的一樣。

Code 的部分如下(給予 1001_0010_1001_0100 的 input):

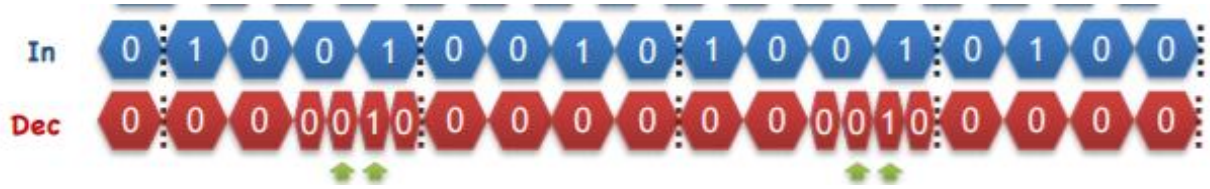
```
initial begin
  @ (negedge clk) rst_n = 1'b0;
  @ (posedge clk)
  @ (negedge clk)
    rst_n = 1'b1;
    in = 1'b1;

  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) $finish;
end
```

✓ Waveform



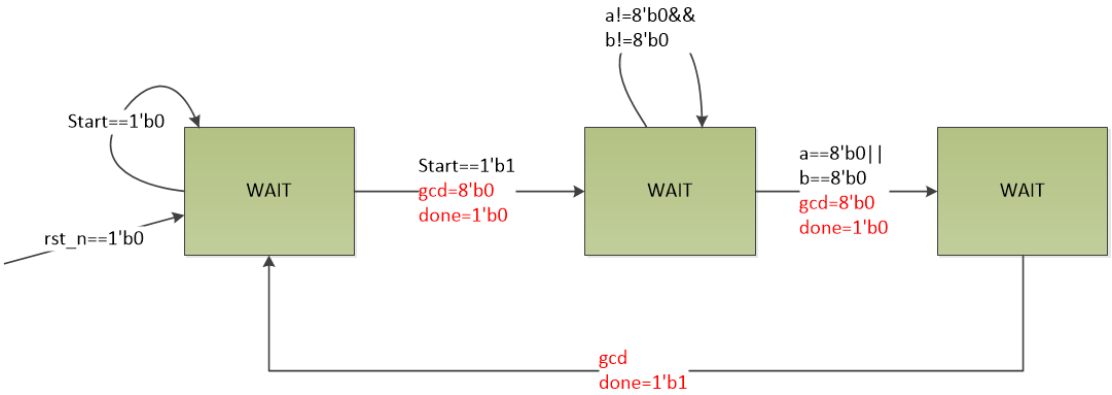
可以發現這個 waveform 跟老師投影片的一模一樣



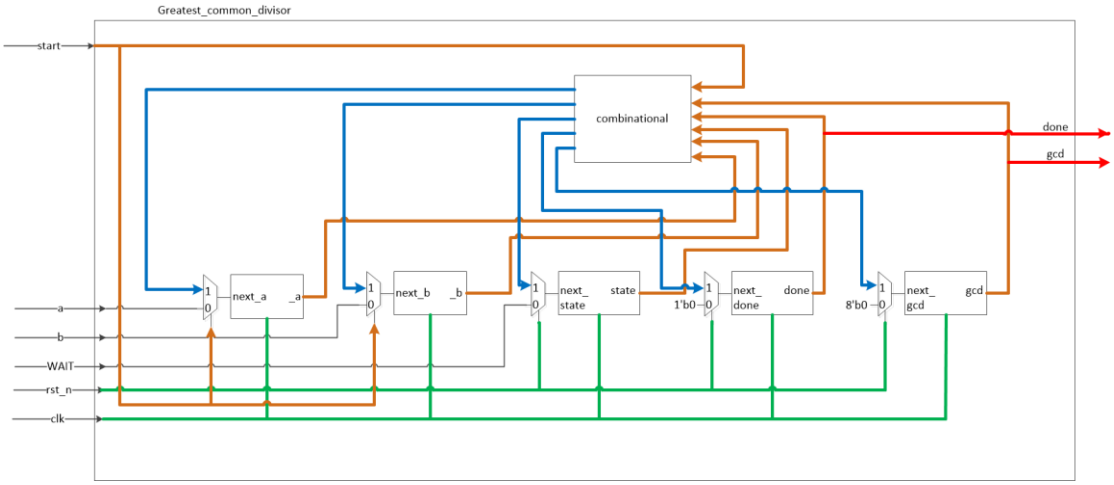
VERILOG QUESTION 2

64 X 8 MEMORY ARRAY MEM

✓ State transition



✓ Block diagram



在 state transition diagram 中。首先由 rst=0 使系統進入 WAIT state，因此 rst_n 也是初始所有的 DFF 的信號。在 rst=0 時，會初始 state 為 WAIT、done=0、gcd=0。若 start 等於 0 就會一直在 WAIT state 中，當 start=1，會由 combinational 電路馬上讀進 a、b，存入 next_a、next_b。指定 next_state=CAL state，此時的 gcd 和 done 都是 output=0。在 CAL state 中，若 a!=0&&b!=0，就會一直在 CAL state 中互相相減。直到 a 或 b 等於 0，就會將不等於零的一方設為 next_gcd，並且 next_done=1，next_state=FINISH。會需要 next_gcd 和 next_done 的原因在於，由於 spec 的 CAL state 到 FINISH state 的輸出皆為 0，而在 FINISH state 才 output gcd 與 done，因此要放在 DFF 中。最後將 next state 設為 WAIT state。

有想過要不要打 default。照理說，宣告兩個 bit 的 state，卻只有三個 state 應該要打 default。不過，我在 default 中，將 next_state=WAIT 會使的就算還沒 reset_n，一開始 state 就會是 WAIT，這樣會跟 spec 不符，所以還是刪去了。

測試的方法為在 negedge clk 給出 rst 的初始信號，將 state 固定在 WAIT，給出 a 與 b，此時的 a 與 b 不是真正的 a 與 b，因為 start!=1，只是為了測試在 start=1 時能不能拿到對的 a 與 b。

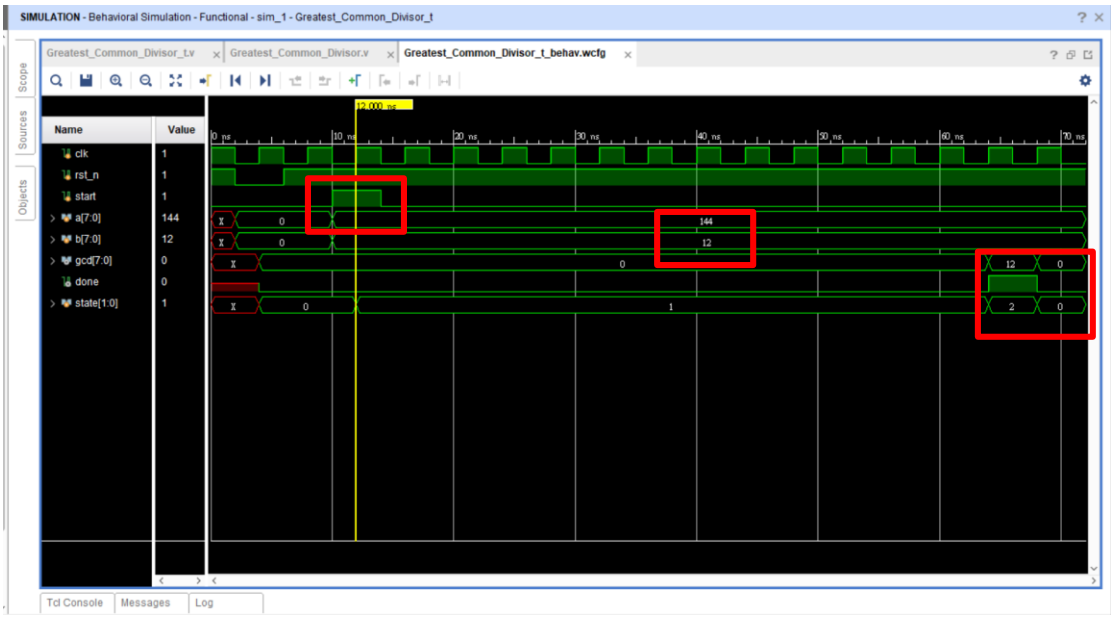
過一個 cycle 的 negedge clk，將 rst=1。再過一個 cycle 的 negedge clk，讓 start=1，使 state 能進入 CAL，此時給出將要計算的 a、b。

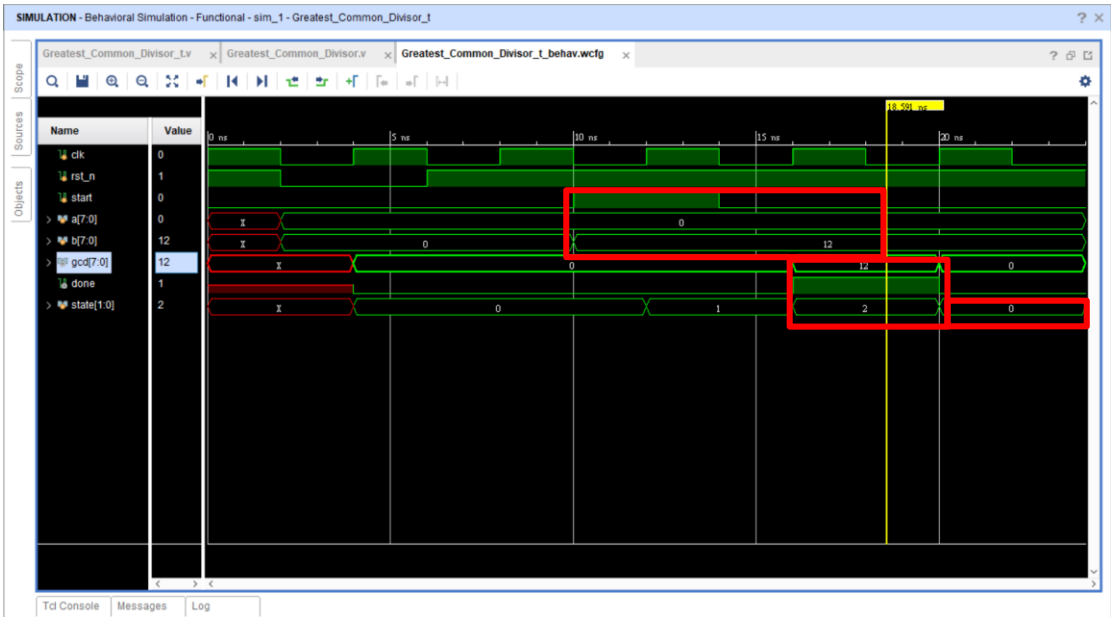
再過一個 cycle 讓 start=0。最後，因為結果易於檢查，僅用人工檢查結果。

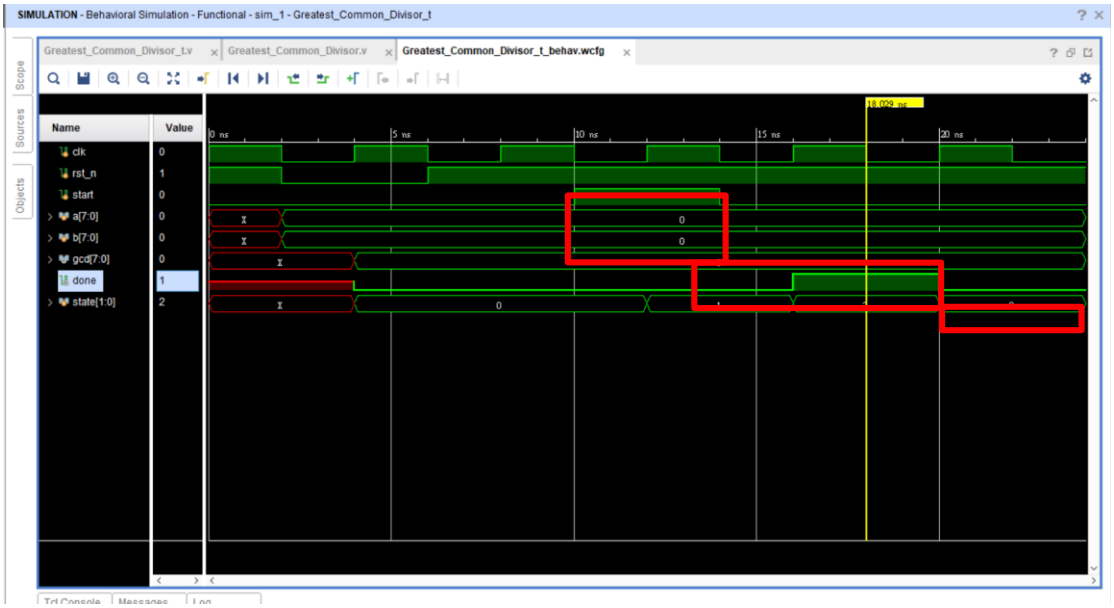
如圖：

```
25  initial begin
26
27      @ (negedge clk) begin
28          rst_n = 1'b0;
29          a = 8'd0;
30          b = 8'd0;
31      end
32
33
34      @ (negedge clk) rst_n = 1'b1;
35
36      @ (negedge clk) begin
37          start = 1'b1;
38          a = 8'd144;
39          b = 8'd12;
40      end
41
42      @ (negedge clk) start = 1'b0;
43
44  end
45
```

✓ Waveform


SIMULATION - Behavioral Simulation - Functional - sim_1 - Greatest_Common_Divisor_t
Greatest_Common_Divisor_t_behav.wcfg
Name
Value
clk
rst_n
start
a[7:0]
b[7:0]
gcd[7:0]
done
state[1:0]
Tcl Console
Messages
Log


SIMULATION - Behavioral Simulation - Functional - sim_1 - Greatest_Common_Divisor_t
Greatest_Common_Divisor_t_behav.wcfg
Name
Value
clk
rst_n
start
a[7:0]
b[7:0]
gcd[7:0]
done
state[1:0]
Tcl Console
Messages
Log

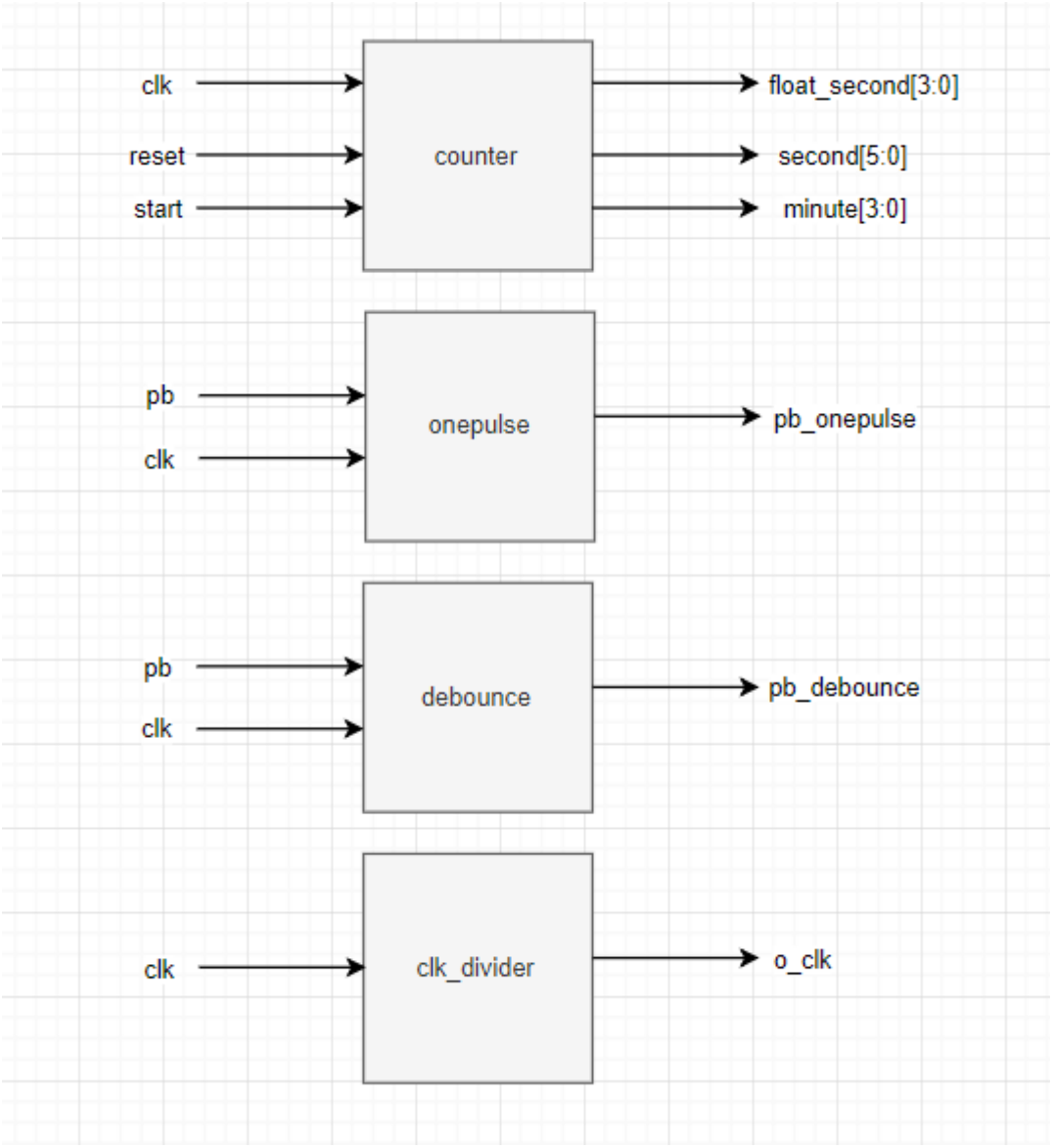

SIMULATION - Behavioral Simulation - Functional - sim_1 - Greatest_Common_Divisor_t
Greatest_Common_Divisor_t_behav.wcfg
Name
Value
clk
rst_n
start
a[7:0]
b[7:0]
gcd[7:0]
done
state[1:0]

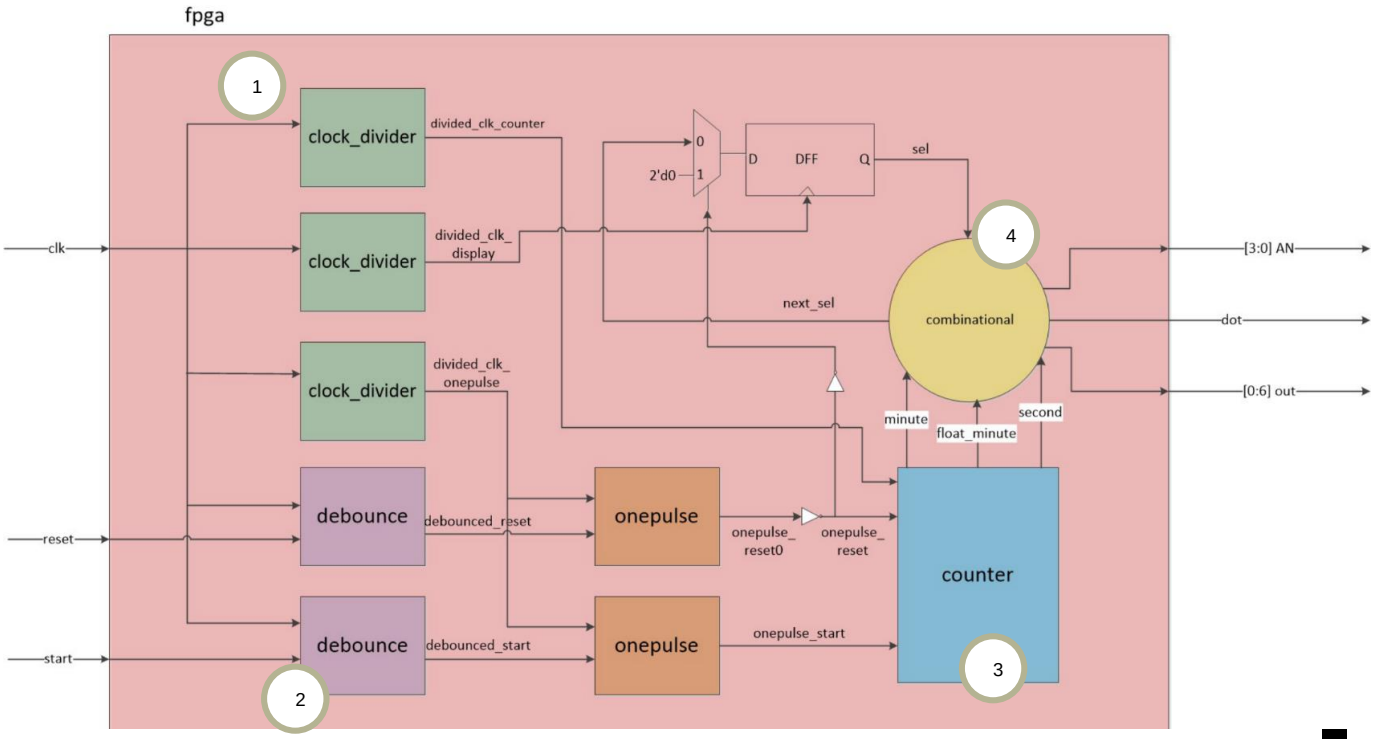
VERILOG QUESTION 3

FPGA

✓ Block diagram

在 top module：fpga 中包含了以下小 module，為了簡化整體的 block diagram，先放上小 module 本身的 port，在整體的 block diagram 就省略小 module 的 port。





✓ 說明

①

關於 clock，因為這次的精準度是 0.1sec 所以要先將板子我提供的頻率(100MHZ)除以 10^7 這樣頻率就會是 10HZ，週期就會是 0.1 秒，由於不同情況下會需要不一樣的 clock 來進行運算，所以我用 clock_divider 做出了三個不一樣的 clock:

divided_clk_counter : 用於顯示數字的快慢，因為精準度是 0.1sec 所以就除以 10^7，用於 counter 這個 module。

divided_clk_onepulse : 用於 onepulse 這個 module 上，它的頻率必須跟 divided_clk_counter 一樣，這樣才不會發生在 counter 在遇到 posedge clk 前 onepulse 的訊號就變回 0 的尷尬情形。

divided_clk_display : 用於讓四個七段顯示器分別通電的時間週期。我將內建 clock 乘以 2^{15}。若四個七段顯示器通電時間太短則來不及通電，無法顯示正確的數字，若太長則無法造成明顯的視覺暫留。

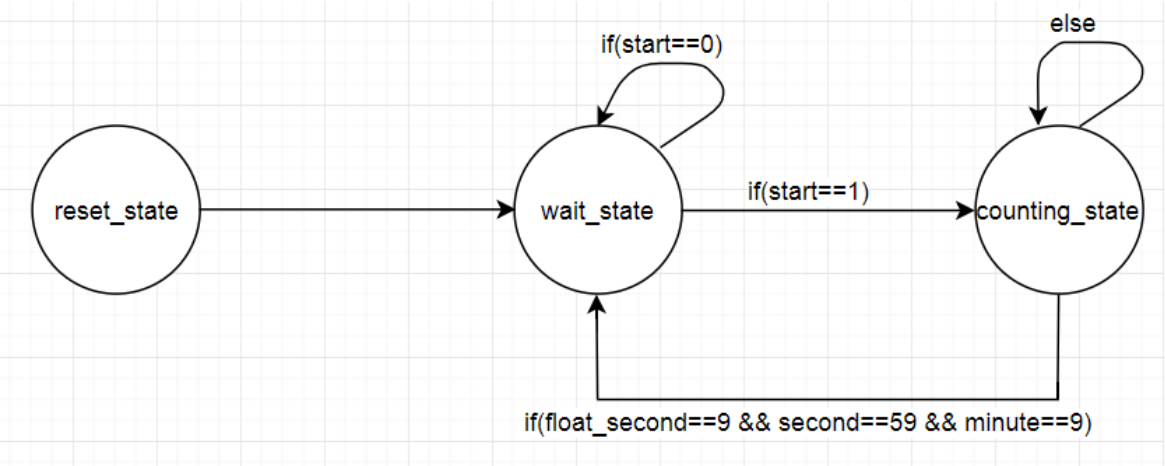
②

clk 設定好後，處裡按鈕的去除雜訊，按鈕包括 rst 與 flip。debounce module 包含四個 DFF 相連，DFF 的 clk 都使用 BASYS 內建 clk，即 100MHz。當四個 DFF output 都是 1 代表準確的按下，輸出 debounced_reset、debounced_start。

③

關於 counter 這個 module，它的主要任務就是計算時間，並將 output(float_second、second、minute)傳給 top module，詳細解釋如下:

✓ State transition diagram



在偵測到 reset 訊號時，我將 output 都 reset 成 0、state 設給 reset_state，如果沒有偵測到 reset 訊號，那 state、float_second、second、minute 就會分別等於 conbinational block 計算出來的 next_state、next_float_second、next_second、next_minute，code 如下圖所示:

```
always@(posedge clk)begin
    if(!reset)begin
        float_second <= 4'd0;
        second <= 6'd0;
        minute <= 4'd0;
        state <= reset_state;
    end
    else begin
        state <= next_state;
        minute <= next_minute;
        second <= next_second;
        float_second <= next_float_second;
    end
end
```

在 reset_state 時，下一個 state 設給 wait_state 以等待 start 訊號，有關時間的變數則是保持不動，code 如下圖所示:

```
case (state)
    reset_state:begin
        next_minute = minute;
        next_float_second = float_second;
        next_second = second;
        next_state = wait_state;
    end
```

在 wait_state 時，主要的任務就是等待 start 訊號，如果 start 訊號來了，那下一個 state 就是 counting_state，如果沒來就是繼續在 wait_state 等待，有關時間的變數則是保持不動，code 如下圖所示:

```
wait_state:begin
    if(start)begin
        next_state = counting_state;
    end
    else begin
        next_state = wait_state;
    end
    next_minute = minute;
    next_float_second = float_second;
    next_second = second;
end
```

counting_state 的主要任務就是進行時間的運算，如果小數第一位等於 0.9 時(0.9 秒)我們必須將它歸零，並把 second 加 1，如果 second 等於 59 秒，那也一樣必須將它歸零，並把 minute 加 1，如果 minute 跟 float_second 都等於 9，而且 second 也等於 9 就代表已經算到底了，接下來要歸零，並進入 wait_state 等待下一個 start 訊號再開始計算，code 如下圖所示:

```
counting_state:begin
    if(float_second == 4'd9)begin
        next_float_second = 0;
        if(second == 6'd59)begin
            next_second = 0;
            if(minute == 4'd9)begin
                next_minute = 0;
                next_state = wait_state;
            end
            else begin
                next_minute = minute + 1;
                next_state = counting_state;
            end
        end
        else begin
            next_second = second + 1;
            next_minute = minute;
            next_state = counting_state;
        end
    end
    else begin
        next_float_second = float_second + 1;
        next_second = second;
        next_minute = minute;
        next_state = counting_state;
    end
end
```

④

這個部分主要是處理如何將 float_second、second 以及 minute 正確的顯示在七段顯示器上，因為七段顯示器有四個，需要分別通電而不能同時，因此我利用 sel 依序去跑，來決定現在哪個七段顯示器要通電，用來初始的訊號是 onepulse_reset，而這裡的 DFF 則是用 clk_display，code 如下圖所示:

```
always@(posedge divided_clk_display)begin
    if(!onepulse_reset)begin
        sel <= 2'd0;
    end
    else begin
        sel <= next_sel;
    end
end
```

但後來在 demo 的時候助教跟我說控制四個七段顯示器的變數並不用吃 reset 訊號，這樣 reset 時才不會發生問題，所以我後來把它改掉了，code 如下:

```
always@(posedge divided_clk_display)begin
    /*if(!onepulse_reset)begin
        sel <= 2'd0;
    end
    else begin*/
        sel <= next_sel;
    //end
end
```

至於 conbinational 的架構大概長這樣(如下圖所示)，其中 sel = 0 是用來處理最右邊的七段顯示器，sel = 1 是右邊數來第二個，sel = 2 是右邊數來第三個，sel = 3 是最左邊的。

```
always@(*)begin
    if(sel == 2'd0)begin···
    end
    else if(sel == 2'd1)begin···
    end
    else if(sel == 2'd2)begin···
    end
    else begin···
    end
end
```

✓ 心得(BY 郭家瑋)

這次的 lab 感覺比較輕鬆。比較有釐清上問題的是，我本來搞不太清楚 reset 跟 start 的差異，後來想清楚應該是，reset 是初始的信號，而 start 是把 state 固定在 WAIT 或是 CAL 開始計算的信號。這樣也比較符合 reset 與 start 的字詞意義。Reset 是在系統發生奇怪錯誤時，可以按下，讓系統不管是現在在何 state，都能回到最初的狀態，也是 WAIT，而 start 是一種從 IDLE(WAIT)開始的信號。

✓ 心得(BY 黎佑廷)

我覺得這次 lab 的第一題不難，因為之前數位邏輯設計好像也寫過差不多的題目，所以寫起來比較沒那麼陌生，至於 fpga 的部分，由於上次的 lab 已經讓我熟悉了如何讓七段顯示器四個都顯示，以及 clk 等相關問題，所以這次寫起來並沒有上次費力，但也很充實，讓我對按鈕、七段顯示器的運作更加熟悉。

成員:

黎佑廷: 負責第 1 題以及 fpga

郭家瑋: 負責第 2 題以及 fpga