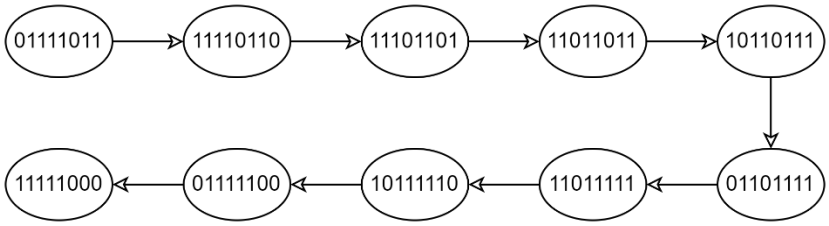
Lab 4 report

組員：110062221 李品萱
110062213 唐翊雯

I. Many-to-one linear-feedback shift register (LFSR)

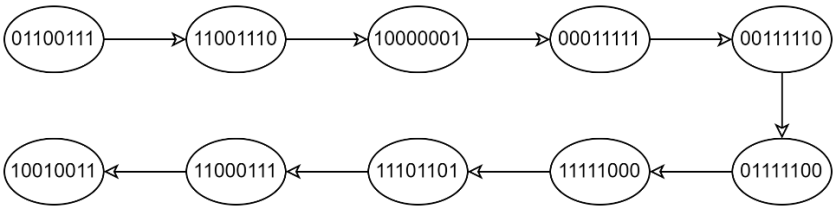
在 basic question 3 中，我們需要用 DFF 及 xor gate 做出一個 LFSR，這一題的 state transition diagram 如下圖。



在 spec 中，我們 reset 時將 DFF reset 為 8’b1011101，此時若在 reset 時改將 DFF reset 為 8’b0，則之後的 output 都會是 0，因為 DFF 單純 trigger clock 接收及給出對應的 output，而 xor gate 在收到的兩個 input 都是 0 的情況下也只會給出 0，因此這個情況下電路出來的值不會有任何改變。

II. One-to-many linear-feedback shift register (LFSR)

basic question 4 與前一題類似，其 state diagram 如下圖。



同樣的，若我們 reset 時改將 DFF reset 為 8'b0，之後的 output 也都會是 0，因為 DFF 會接收 0 再送出 0，而 xor gate 收到兩個 0 時 output 0，因此最後的結果都會是 0。

III. Content-addressable memory (CAM) design

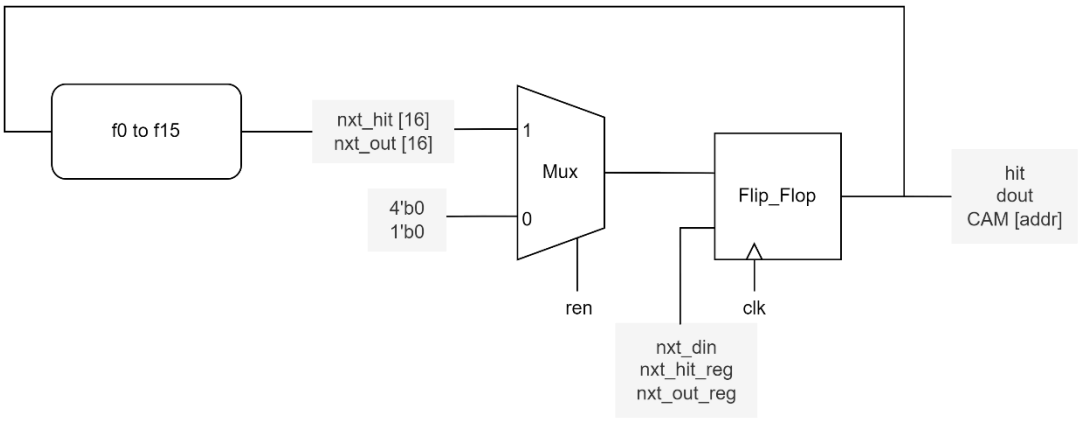
這題我們需要做出一個 CAM，能夠將 din 存在對應的 addr，及 output din 對應的 addr。

首先，在讀取的部分，我們先看 nxt_din 會是什麼，如下圖。

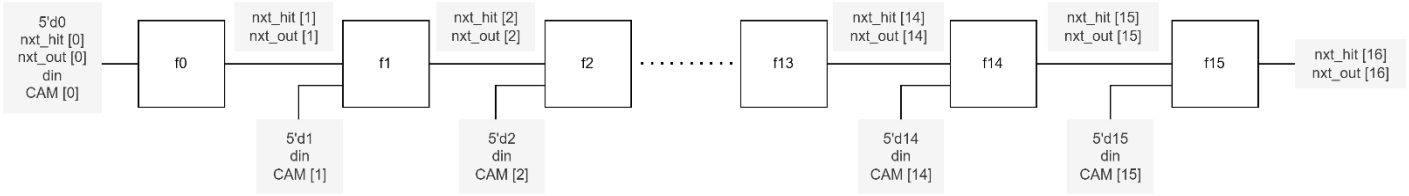
```
assign nxt_din = (wen & !ren) ? din : CAM[addr];
```

這麼做的原因是，在 sequential block 中，我們只在 wen 為 1 且 ren 為 0 時做讀取，此時為了不造成 inferred latch 的問題出現，我們先看現在是否符

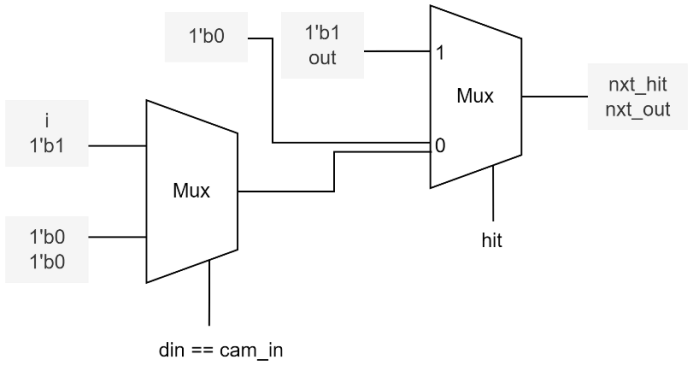
合讀取的條件，若符合，nxt_din 即會接 din，不符合時 nxt_din 則會接該 addr 位置當前的值。這樣我們在 sequential 的部分就只要處理 ren 為 1 或 0 對應到的 hit 與 dout。這題電路圖如下。



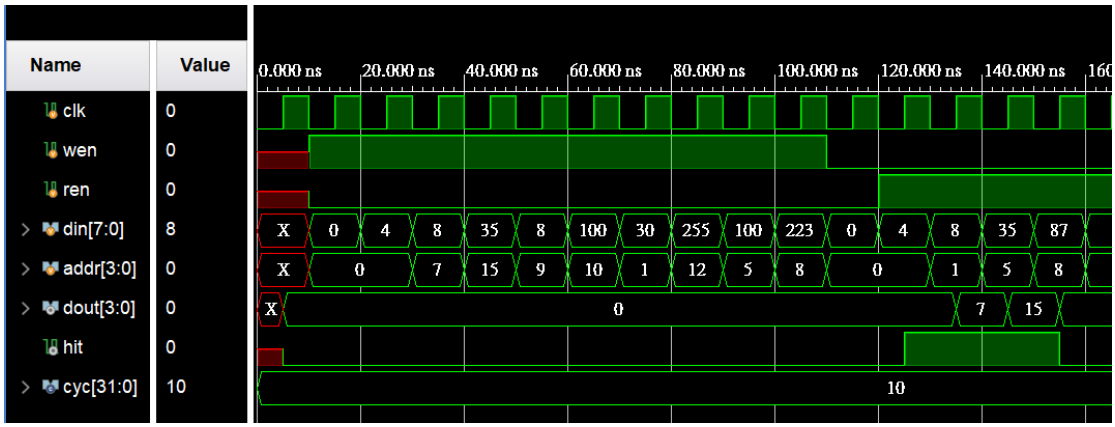
上圖的 f0 to f15 如下圖。在 comparator array 及 priority encoder 的部分原先我們想到的是用 for loop 讀過整個 CAM，但上課時有提過 for loop 是無法 synthesized 的，因此我們將 for loop 的概念實做出來，我們寫了一個 for_loop 的 module，將 16 個 for_loop 的 module 接起來，每次將上一個的 output 作為下一個的 input，並送進要檢查的 addr 及 CAM 的值，如下圖，我們最後得到 nxt_hit[16]及 nxt_out[16]即為所求。



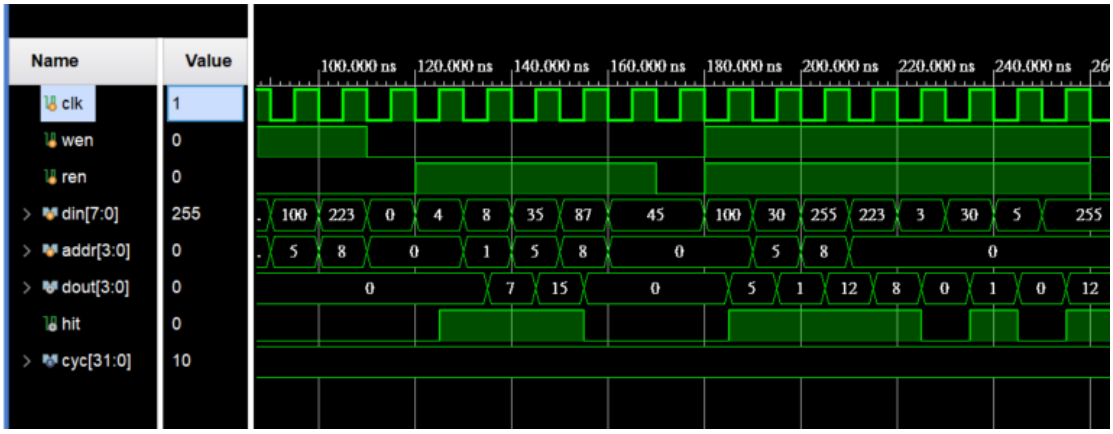
每個 for_loop 的設計如下。我們先看現在是否已經找到要讀的值，如果沒有再看現在的值是否是我們要找的。這邊的設計上我們也顧及若有多個 match 的 addr 要 output 最小的，因此一旦 hit 值為 1，後面便不會再抓對應的 addr，整個過程運用了 comparator array 及 priority encoder 的概念。



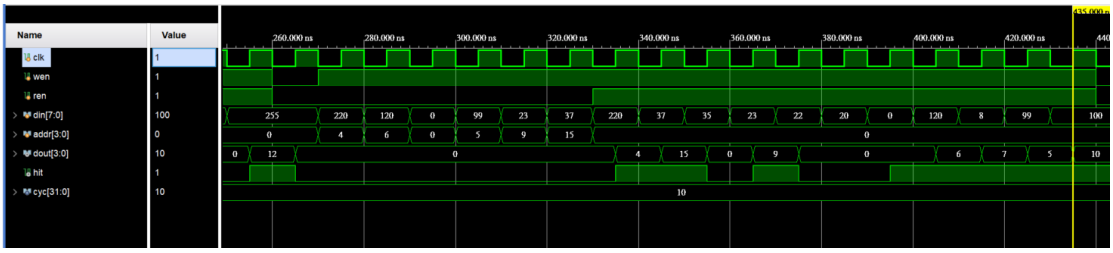
在 testbench 的部分，我們先做出跟 spec 相同的圖，初步確認正確性。



而後我們確認 ren 及 wen 同時為 1 時 CAM 只會做寫的動作，並且 output 的是最小的 addr，如下圖的 din 為 100 會 output 5 而不是 10。

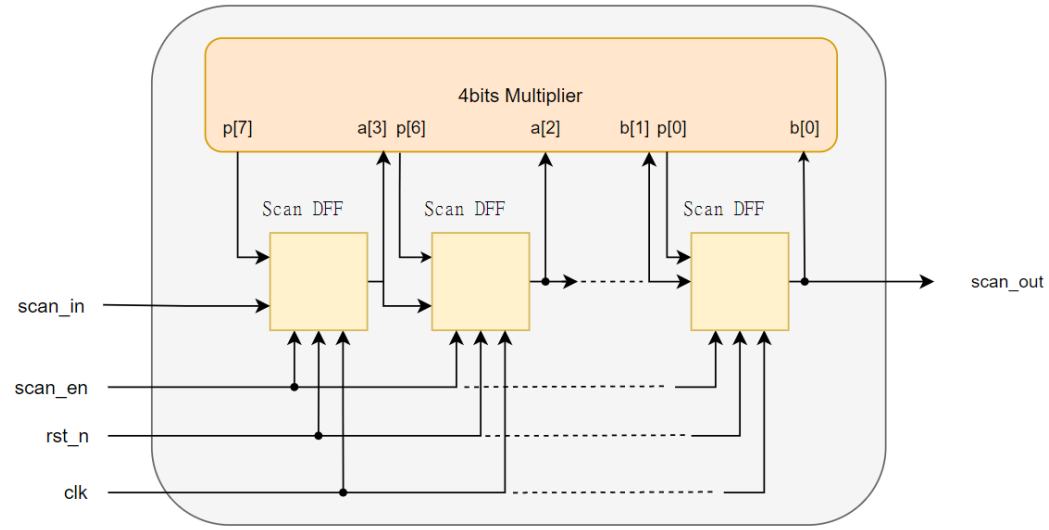


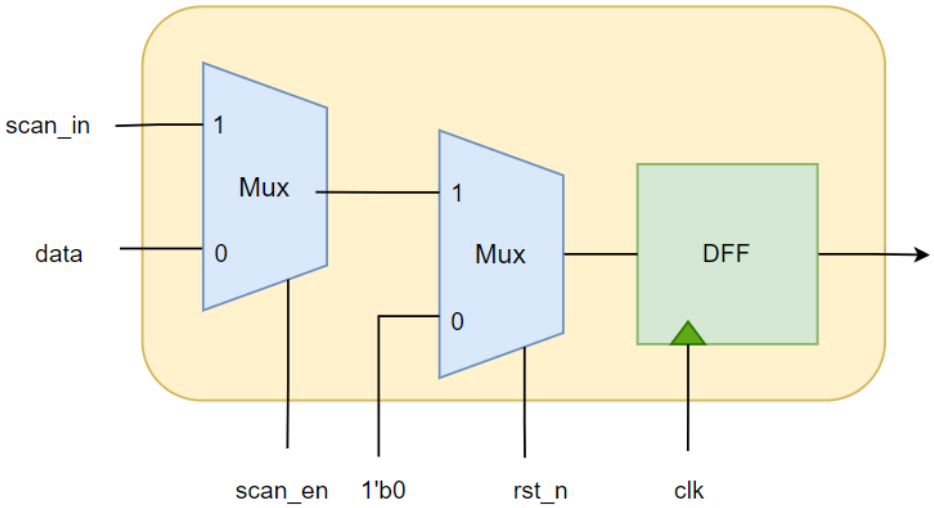
後面我們又再測了一些讀寫的動作，確認 CAM 及 addr 的大小有符合題目的要求，且若有新的值存在有值的 addr 位置，新的值會直接覆蓋掉舊的值，如下圖原本 35 寫在 CAM[15]的位置，但因為這個位置後來寫進了 37，因此會 dout 及 hit 都是 0，而黃線位置的 100 原本在 addr 為 5 及 10 都能找到，但因為 5 寫進了 99 因此 din 為 100 的狀況會 output 10。



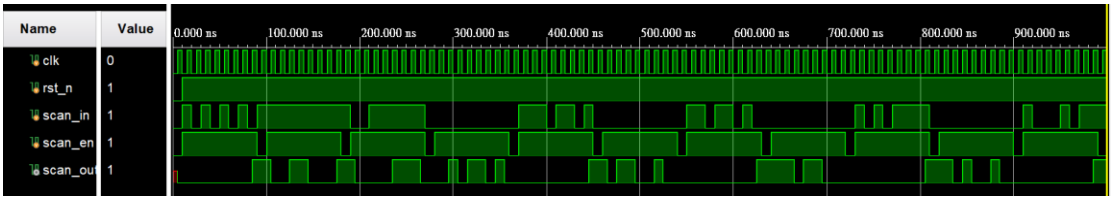
IV. Scan chain design

這題的 diagram 在 spec 上已經有給了，如下圖：



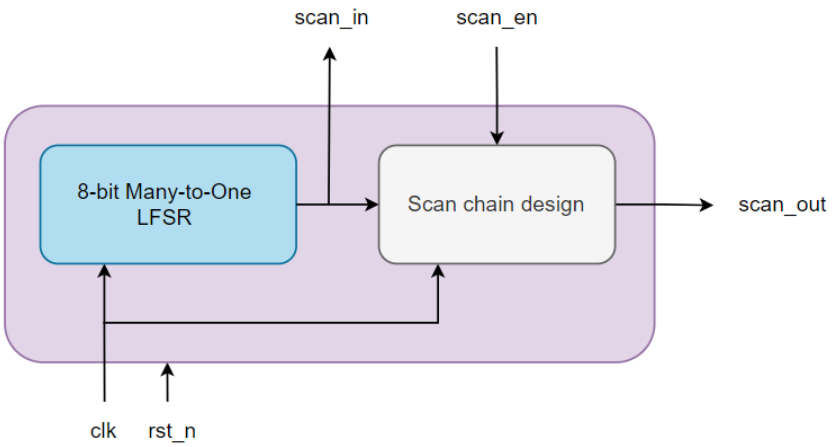


寫的時候就完全按照這張圖去接就好。Testbench 的部分，我們用 random 來 generate scan_in 的值，檢查的時候將這 8 個 bit 依 diagram 分成 a、b 兩數並檢查 scan_out 的值是否確實為兩數相乘的結果，waveform 如下：



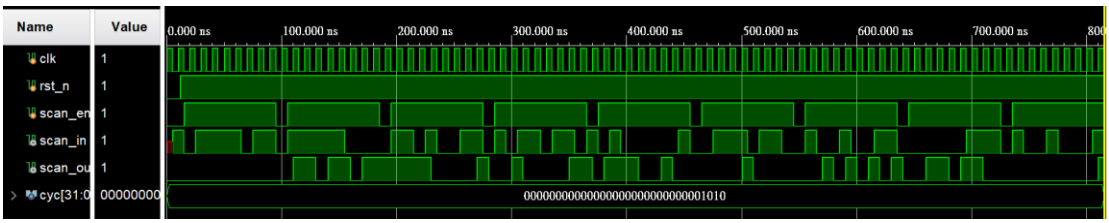
V. Built-in self test

這題的 diagram 在 spec 上也已經有給了，如下圖：

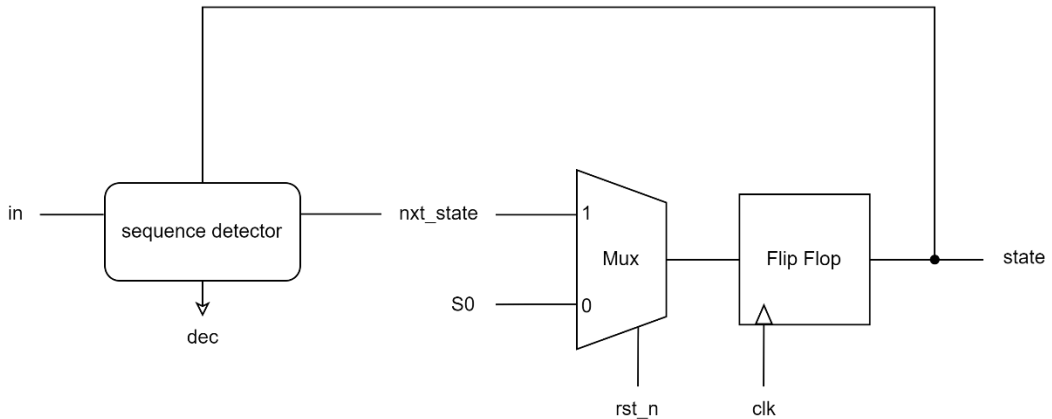


我們只需要將 basic question 3 與 advance question 2 的 module 接起來就好了。advance question 2 的 module 可以原封不動的搬進來，basic question 3 的 module 則有稍微需要修改的地方：除了 output 要根據 spec 的要求改成 MSB 以外，這個 module 的 output 作為 scan chain design 的 input，不能和 scan chain design 一樣使用 positive edge trigger。因此我們將這個 module 改成 negative trigger，避免 input 在 posedge 改值。

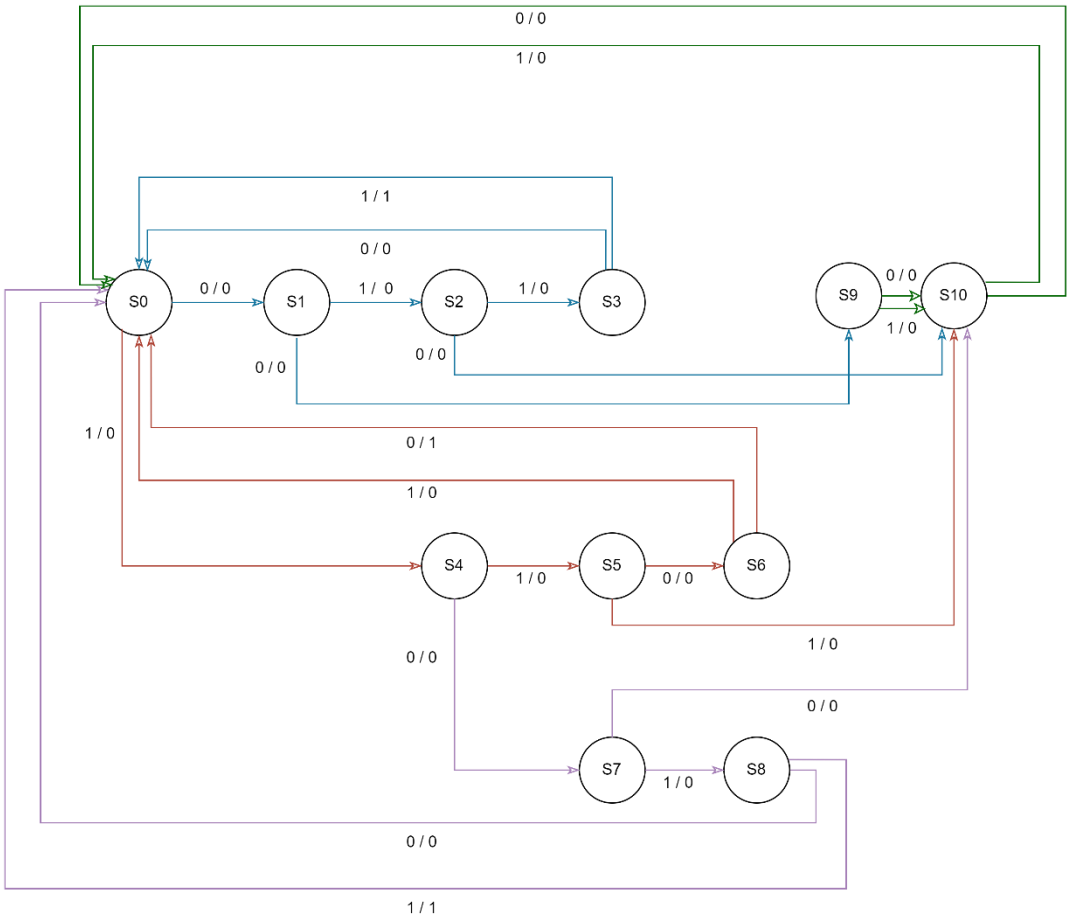
Testbench 的部分，我們一開始 reset 之後，便照 advance question 2 的說明一樣，讓 scan_en=1'b1 持續 8 個 clock cycle、讓 scan_en=1'b0 持續 1 個 clock cycle，再讓 scan_en=1'b1 持續 8 個 clock cycle，並一直重複下去。而檢查的方法和 advance question 2 檢查的方式相同。



VI. Mealy machine sequence detector



如上圖（sequence detector 代表下圖的 state transition），這題我們需要做一個每 4 個 bits detect 一次的 mealy machine，首先我們先畫出它的 state diagram，如下圖。



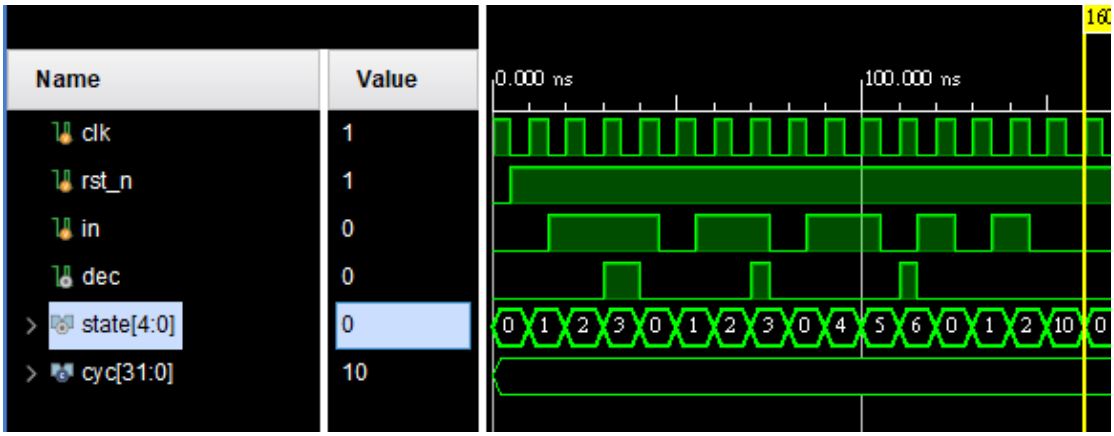
dec 為 1 的條件為，吃到 0111，1011 或 1100 三種 input sequence，對應的 state 大致可分成三組：

- 0111 -> (S0, S1, S2, S3)
- 1011 -> (S0, S4, S7, S8)
- 1100 -> (S0, S4, S5, S6)

由於它是 4 bits detect 一次，因此即使吃到不合法的輸入也要吃完 4 個 bits，所以我們用 S9 及 S10 來處理。觀察發現，S0 吃到 1 或 0 都有可能產生合法輸入，此時如果下一個輸入是不合法的，我們會需要再經過兩個 state 再回到 S0，而 S9 就是在做這件事，S10 同理，當我們吃到第三個輸入發現不合法時我們會需要一個 state 去吃最後的 input，再回到 S0。

在設計 state transition 的部份我們先寫出三組合法 sequence 各自的 state 變化，發現他們都可以共用 S0，而 1100 及 1011 可以共用 S4，因此處理合法 sequence 的部份我們只需要 S0 到 S8 即可，接著依照 input 對應的 state 接起來即為所要的 state diagram。

Testbench 的部分我們先確認能得到與 spec 上相同的結果，為了方便 debug，我們將 state output 出來，如下圖。

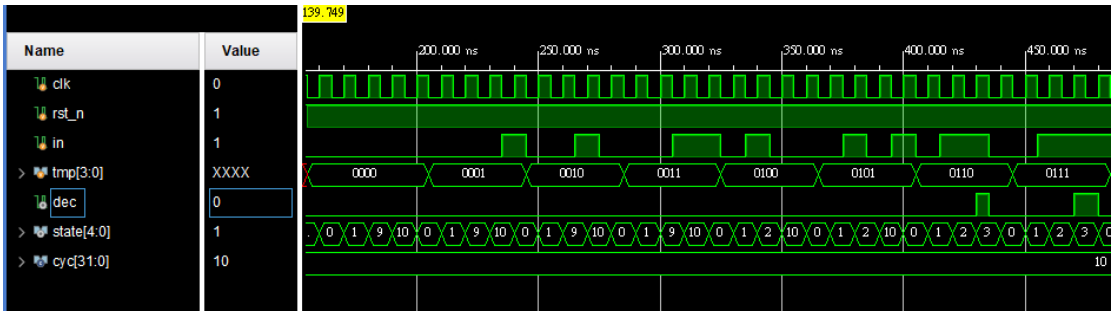


在第一次跑到 S3 時，可以看到 dec 為 1 的狀態持續了不只一個 clock，這是因為它是一個 mealy machine，前面起來的 1 是 state change 的 1，但此時 clock 還沒起來，又收到 1，而 input 變 output 就會變，所以又會 output 1。

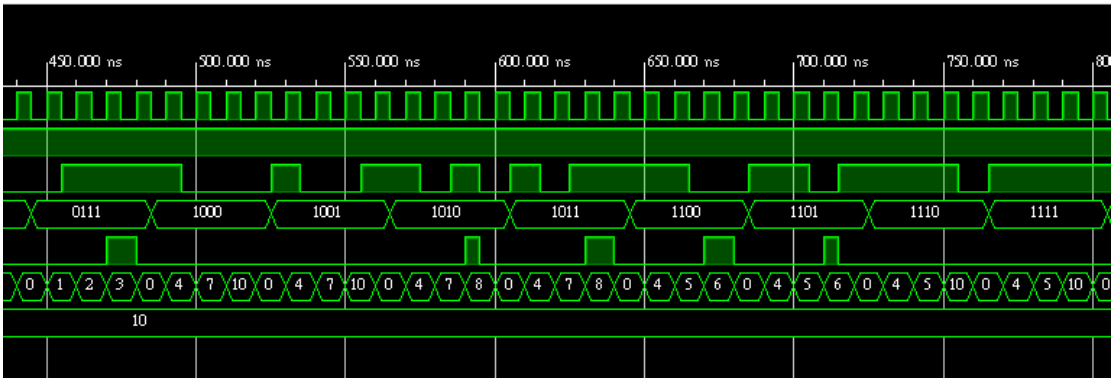
而後我們枚舉所有可能的 input sequence，code 如下圖。

```
tmp = 4'b0;
@ (negedge clk)
repeat(2**4)begin
    in = tmp[3];
    @ (negedge clk) in = tmp[2];
    @ (negedge clk) in = tmp[1];
    @ (negedge clk) in = tmp[0];
    @ (negedge clk) tmp = tmp+1'b1;
end
```

waveform 如下面兩張圖，其中，在 tmp 為 0110 出現 dec 為 1 是因為它在 s3 變成下一個 state 之前收完 0111。



下圖中 1010 及 1101 的區間看到的 dec 為 1 原因同上，使他們為 1 的 sequence 分別為 1011 及 1100。



VII. FPGA Demonstration – 1A2B game

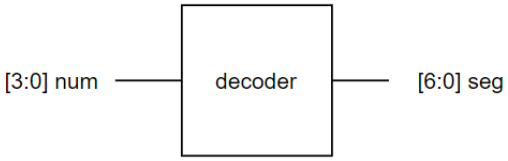
這次的 FPGA demonstration 要做一個 1A2B 的遊戲，整個實作可以分成數個 part，以下將一一說明：

- fpga input 的處理

  在這次的遊戲中，input 包含 button 與 switch。Button 的處理就和上次的 lab 一樣，要使用 debounce 與 one pluse 兩個 module，以得到我們想要的 signal。

- 使用 finite state machine 處理整體遊戲運作

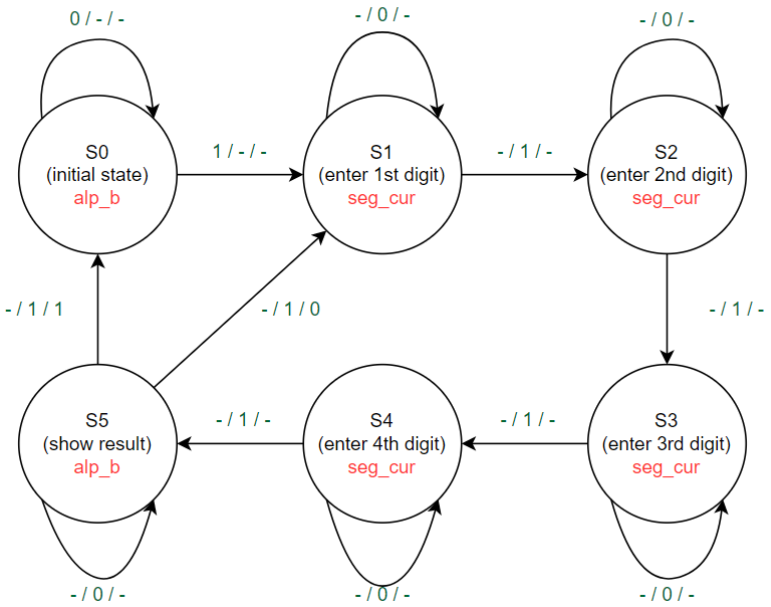
  這次遊戲的整體運作可以用一個 Moore machine 來表示。其中 input 是 start_op / enter_op / res_a == 4'd4 這三個值，分別代表 start 是否被按下、enter 是否被按下，以及 input 的答案是否正確 (4A0B)。而 output 是 seg0 的值，代表 7-segment display 最右邊的 digit 應該要有的 output。其中 alp_b 是 display 出 b 所需的值 (7' b1100000)，而 seg_cur 是根據當前 switch 給的值，所對應的 7-segment。為此我們寫了一個 input_to_seg 的 module，將數字跟 7-segment 的值做對應：



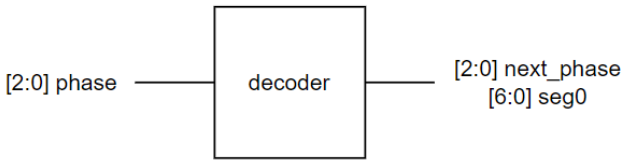
[3:0] num	[6:0] seg
4'd0	7'b0000001
4'd1	7'b1001111
4'd2	7'b0010010
4'd3	7'b0000110
4'd4	7'b1001100
4'd5	7'b0100100
4'd6	7'b0100000

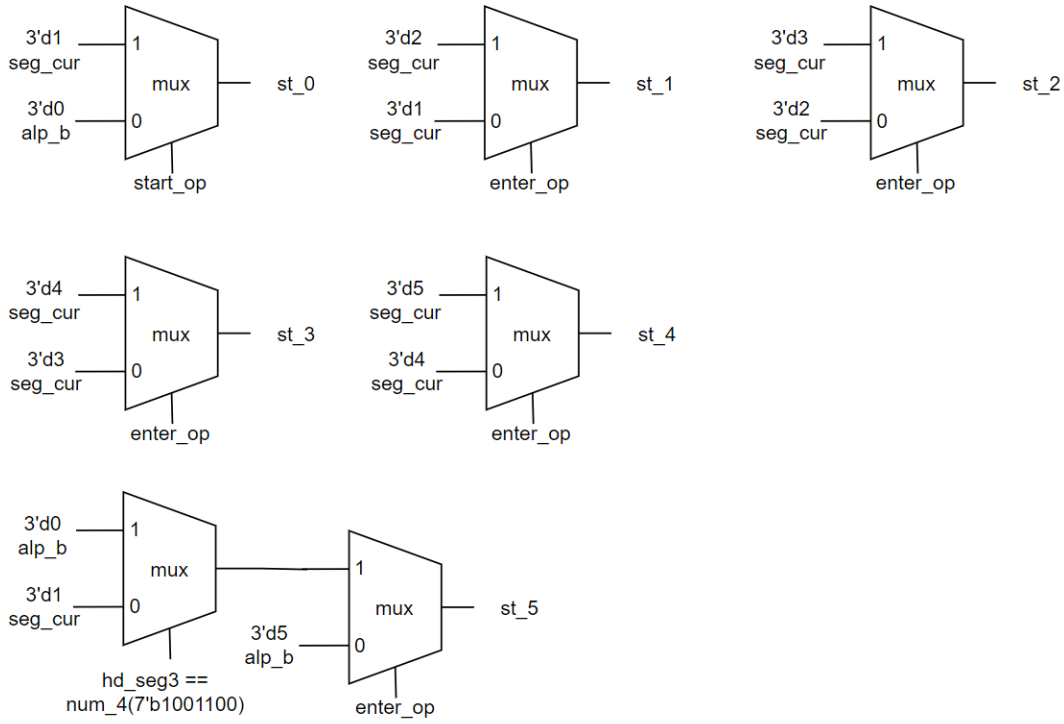
4’d7	7'b0001111
4’d8	7'b0000000
4’d9	7'b0000100

而整體的 state transition diagram 如下，其中 input 以 start_op / enter_op / res_a == 4’d4 這三個值來表示：



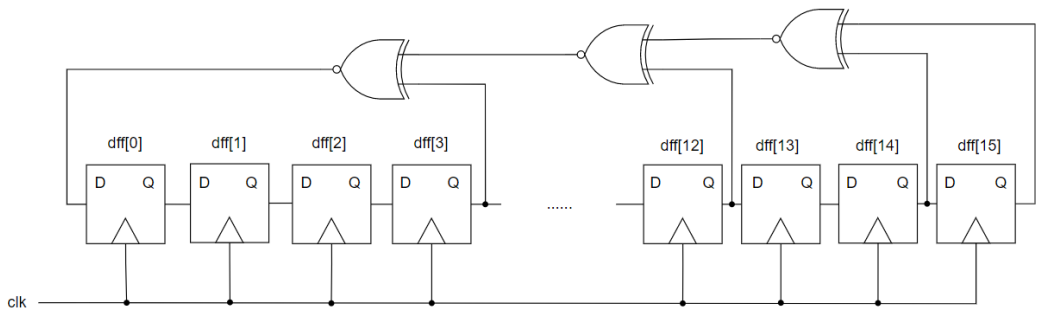
Block diagram 如下，next_phase 是指下一個應該要到的 state，而 hd_seg3 是 7-segment 最左邊的數字，在上一個 state 應該要設成的值。其他名字的定義同 state transition diagram：





[2:0] phase	[2:0] next_phase, [6:0] seg0
3’d0	st_0
3’d1	st_1
3’d2	st_2
3’d3	st_3
3’d4	st_4
3’d5	st_5

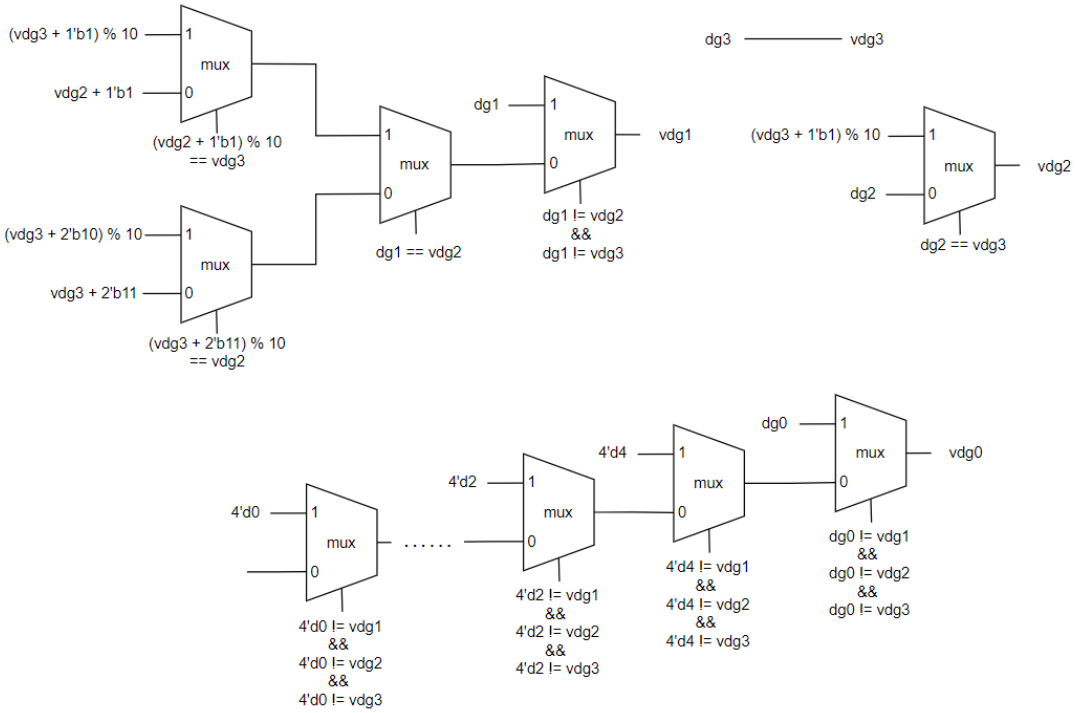
- 隨機生成一個合法的答案
  生成答案的部分，我們使用了一個 16bits 的 LFSR，每 4 個 bits 為一單位生出一個四位數的答案，其 block diagram 如下：



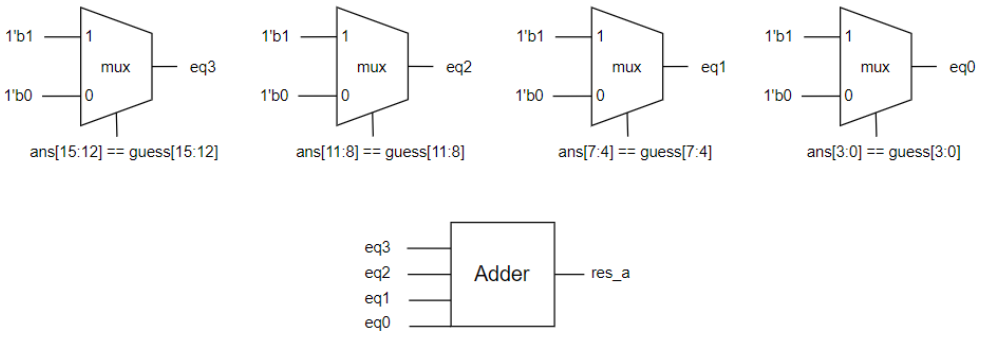
由於數字必須要在 0~9 之間，當生好數字後我們先每一位都 mod 10：

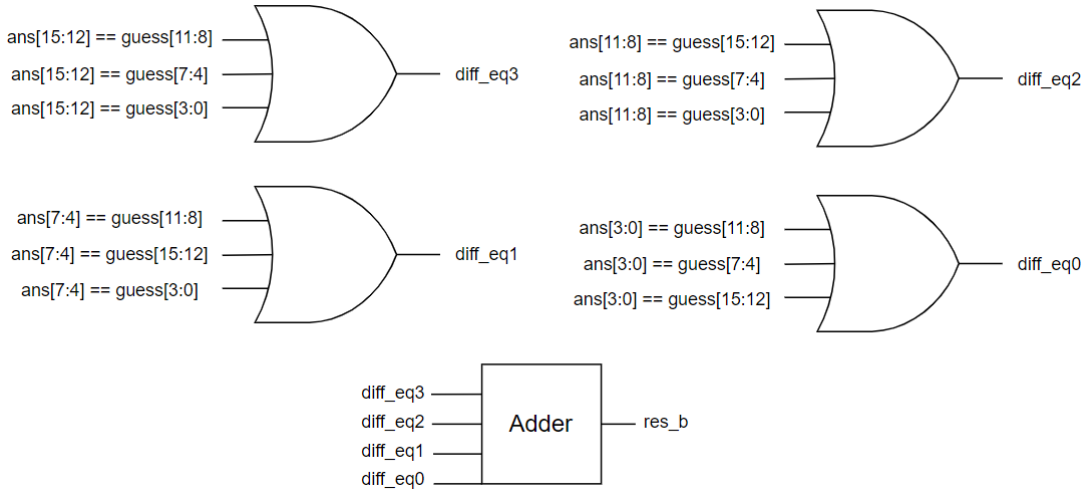
```
assign dg3 = out[15:12] % 10;
assign dg2 = out[11:8] % 10;
assign dg1 = out[7:4] % 10;
assign dg0 = out[3:0] % 10;
```

之後再從最高位開始，對重複的數字做處理，替換成不重複的其他數字，block diagram 如下，其中 vdgn 指的是第 n 個 digit 經過處理後的結果。vdg0 中間省略的部分接了很多相似的 mux，枚舉了 4’d0~4’d9 之間的所有值：



- 檢查答案與使用者的輸入相差多少
  在使用者輸入完 4 個 digit 後，我們要比較正確答案與使用者的輸入，計算出對應的 A 有多少個(res_a) 和 B 有多少個(res_b)，其 block diagram 如下：





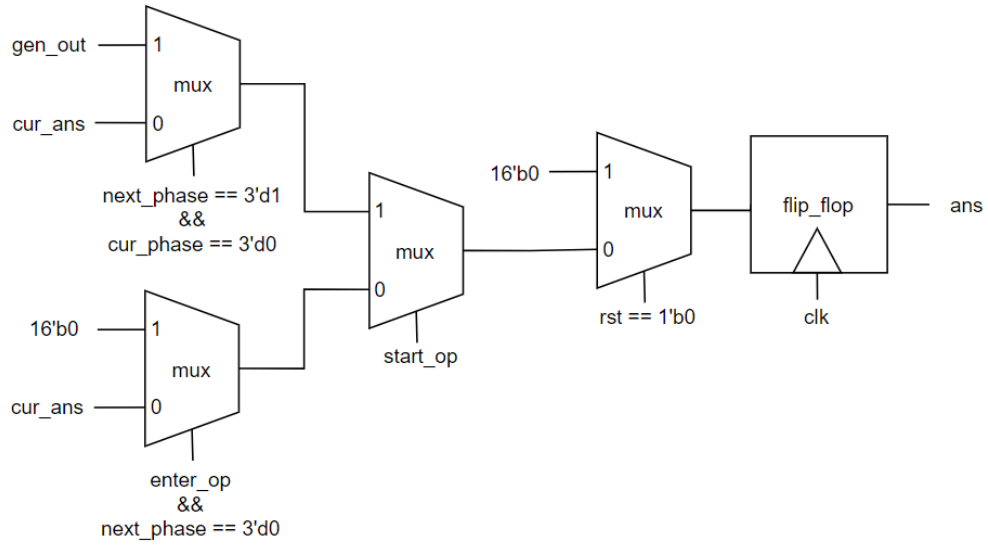
- fpga output 的處理

  output 總共有兩個：顯示答案的 16 個 LED 以及 7-segment display。

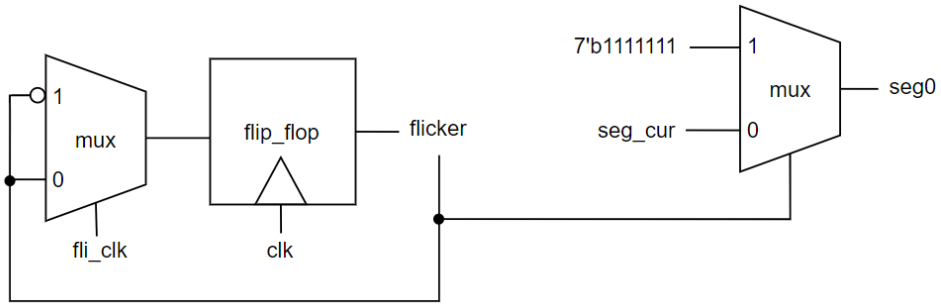
  16 個 LED，也就是答案的部分總共有三種狀況：

  - state 為 S0 時值為 16’b0，所有 LED 皆不亮
  - 在 state 從 S0 變成 S1 時要生成一個新的答案
  - 其他時候維持原本的值

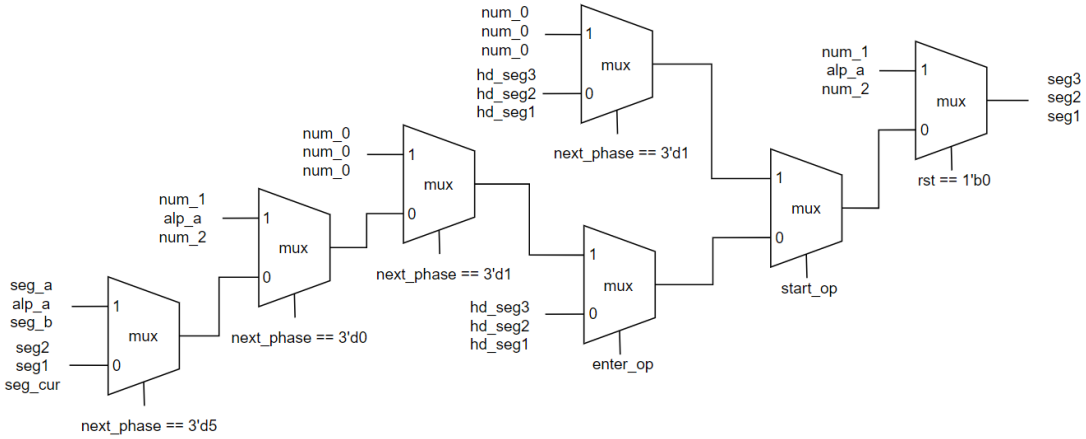
  我們將這個部分根據上述三種狀況，使用 sequential circuit 來處理，block diagram 如下：



7-segment display 的部分，我們將最右邊的 digit(以下稱 seg0) 與其他三個 digit(以下由高到低位稱 seg3, seg2, seg1) 分開處理。seg0 的值在 input 的部分已經處理好了，剩下閃爍的部分要處理。我們使用前一次 Lab 實作的 clock divider module，generate 一個處理閃爍頻率的 clk (fli_clk)，這部分的 block diagram 如下：



而 seg3, seg2, seg1，最主要的部分是在猜的那幾個 state 必須在按下 enter 後將數字左移，其 block diagram 如下：



其中 num_i 代表 7-segment 要顯示數字 i 所對應的值、alp_i 代表 7-segment 要顯示字母 i 所對應的值、seg_a 和 seg_b 為 res_a 與 res_b 對應至 7-segment 的值。

VIII. Summary

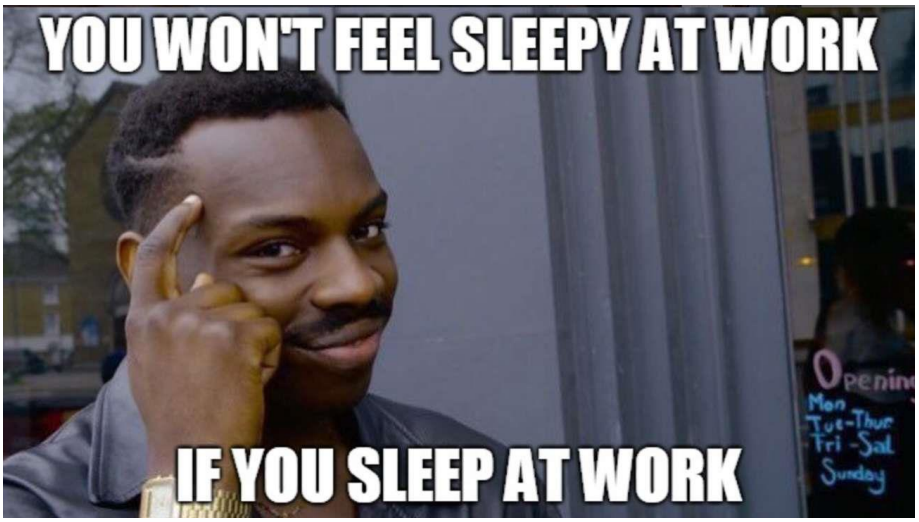
這次的 lab 我們學到了 finite state machine 的實作及應用，也持續熟悉 memory 的操作。

這幾次 lab 的操作下來，我們在寫硬體的時候更能夠以硬體的方式思考及設計，例如這次的 CAM 我們一開始想到要跑過整個 memory 找答案，但它不像軟體可以直接跑迴圈，因此我們去思考我們的電路圖應該長怎樣進而做出對應的設計，也會去想麼避免 inferred latch 等等會使電路沒辦法正確接好的問題。

在 Mealy machine sequence detector 的部分，一開始寫的時候以為它是每次多往後看一個 bit 所以多花了一點時間，後來發現與 spec 的圖對不上才又改正，這題讓我們學到了從設計 state transition 到設計出對應的 mealy machine。

FPGA 的部分，讓我們學習到如何應用 state transition diagram 來使遊戲的實作變得有條理、容易實作。有很多東西要好好思考該用 sequential 還是 combinatial 的方式實作、輸出合法答案的部分也需要仔細思考，做完這一題學到很多東西。

這次的 Lab 除了讓我學到許多知識外，也讓我領悟了人生大道理，如下圖：



IX. Contributions

- **Code:**

  Content-addressable memory (CAM) design by 唐翊雯

  Scan chain design by 李品萱

  Built-in self test by 李品萱

  Mealy machine sequence detector by 唐翊雯

  FPGA - 1A2B game by 李品萱

- **Report:** 各自描述負責的題目