Lab 4 Finite State Machine

實驗報告

組長:劉奇泓 109033135

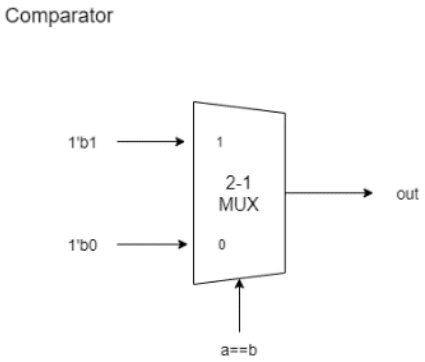
組員:洪聖祥 109062315

1. Content-addressable memory (CAM)

CAM 有 5 個 input 2 個 output，它的功能是當 ren = 1’b1 時會讀取 memory array 的值並跟 din 比較是否相同。如果有多個 address 符合條件則輸出 dout = 最大的 address hit = 1’b1。當 wen = 1’b1, ren = 1’b0 則把 din 寫進 memory array 中的 addr 位址。CAM 可以分成 2 個 sequential circuit 和 3 個 combinational circuit。2 個 sequential circuit 分別儲存並在 positive edge 更新兩個 output dout 和 hit 的值。3 個 combinational circuit 分別為 1 個 8-bit memory array、1 個 comparator array 和 1 個 priority encoder。

Comparator array

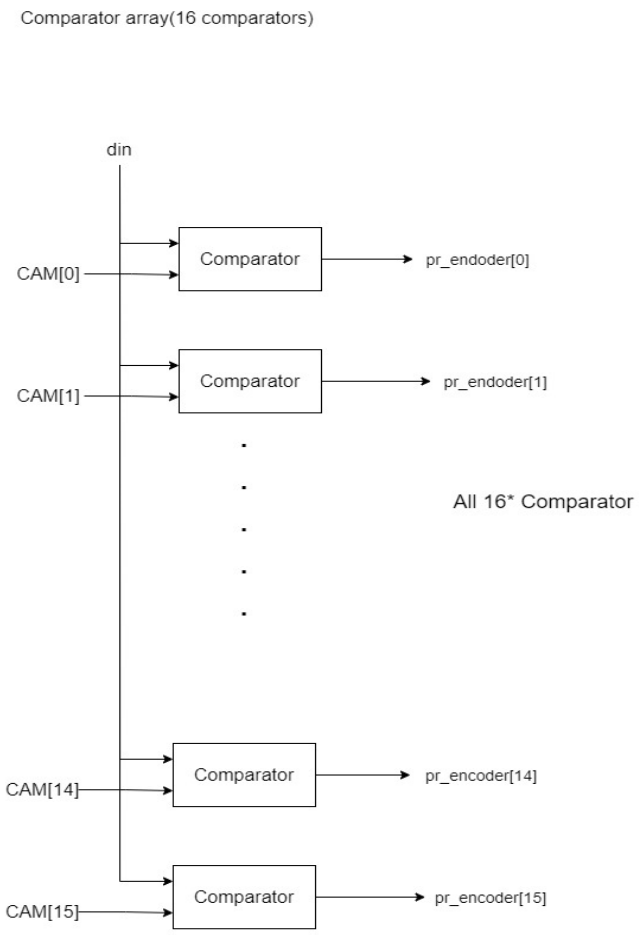
Block diagram:



Code

```
module Comparator(a, b, out);
input [7:0] a, b;
output out;
assign out = (a == b) ? 1'b1 : 1'b0;
endmodule
```

Comparator 用簡單的條件 a == b 來決定 out 是 1’b1 還是 1’b0。



Comparator array 總共有 16 個 comparators。第一個 comparator CAM[0]跟 din 比較。如果兩個相等，則輸出 1’b1 否則輸出 0 並把輸出值指定給 pr_encoder[0]。第二個 comparator 則是 CAM[1]跟 din 比較並把輸出值指定給 pr_encoder[1]。第 i 個 comparator CAM[i-1]跟 din 比較並把輸出值指定給 pr_encoder[i-1]。

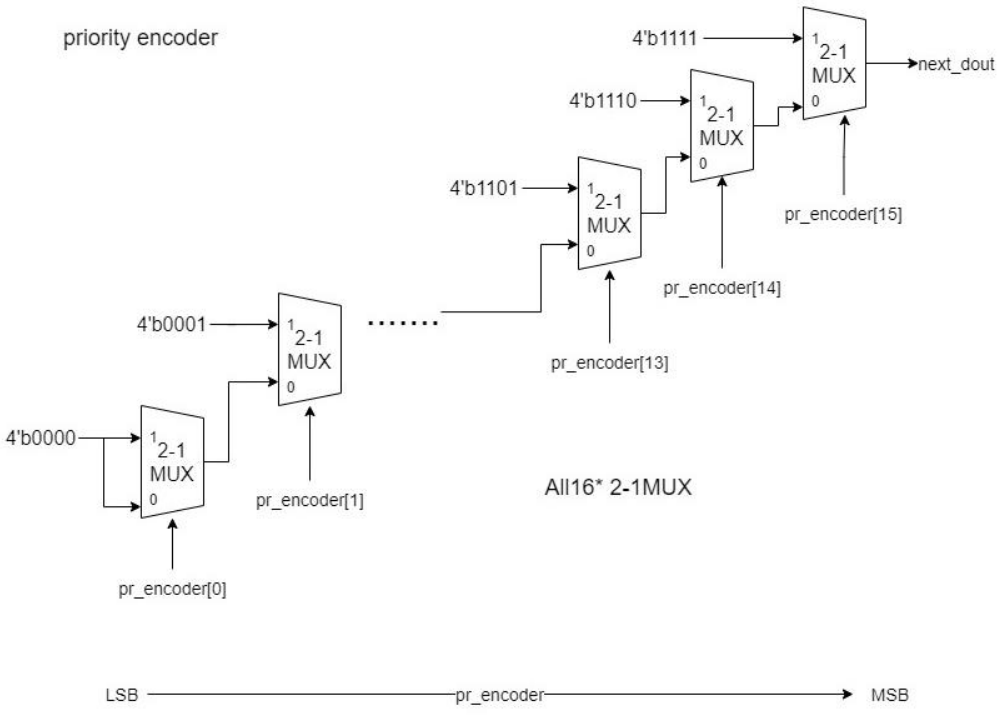
Code

```
always@(posedge clk)begin
    dout <= next_dout;
    hit <= next_hit;
    if({ren,wen}==2'b01)CAM[addr] = din;
end
Comparator c0(din, CAM[0], pr_encoder[0]);
Comparator c1(din, CAM[1], pr_encoder[1]);
Comparator c2(din, CAM[2], pr_encoder[2]);
Comparator c3(din, CAM[3], pr_encoder[3]);
Comparator c4(din, CAM[4], pr_encoder[4]);
Comparator c5(din, CAM[5], pr_encoder[5]);
Comparator c6(din, CAM[6], pr_encoder[6]);
Comparator c7(din, CAM[7], pr_encoder[7]);
Comparator c8(din, CAM[8], pr_encoder[8]);
Comparator c9(din, CAM[9], pr_encoder[9]);
Comparator c10(din, CAM[10], pr_encoder[10]);
Comparator c11(din, CAM[11], pr_encoder[11]);
Comparator c12(din, CAM[12], pr_encoder[12]);
Comparator c13(din, CAM[13], pr_encoder[13]);
Comparator c14(din, CAM[14], pr_encoder[14]);
Comparator c15(din, CAM[15], pr_encoder[15]);
Priority_Encoder pe(pr_encoder, tmp_dout, tmp_hit);
assign next_dout = (ren==1'b1) ? tmp_dout : 4'b0000;
assign next_hit = (ren==1'b1) ? tmp_hit : 1'b0;
```

instance 16 個 comparator 以 din CAM 為 input pr_encoder 為 output 來實作 Comparator Array

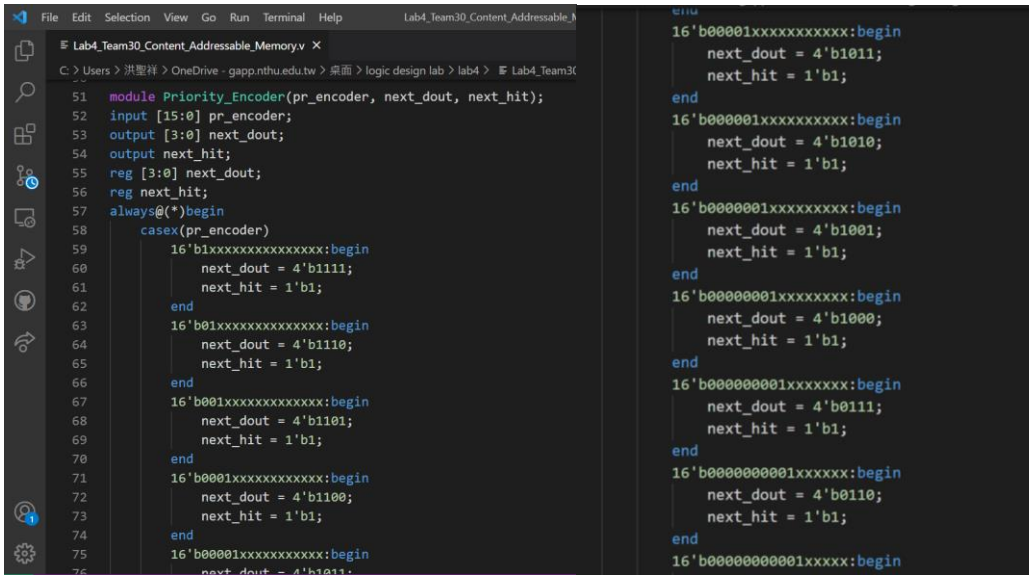
Priority encoder

Block diagram:



2^n to n Priority Encoder 的功能是當 2^n bit input 值的每一個 bit 中有 2 個以上的 bit = 1’b1，可以決定一個優先順序來決定 output。這時用普通的 encoder 會出現問題，因為普通的 encoder 限制只能有一個 bit = 1’b0 其他的 bit 則是 1’b1。我 16 to 4 Priority encoder 的實作是用 16 個 2-1 MUX 串起來。當 16 bit input 中有不只一個 bit 是 1’b1，我們需要優先順序(這裡是 significant bit)最高的輸入將會被優先輸出。

Code 片段:

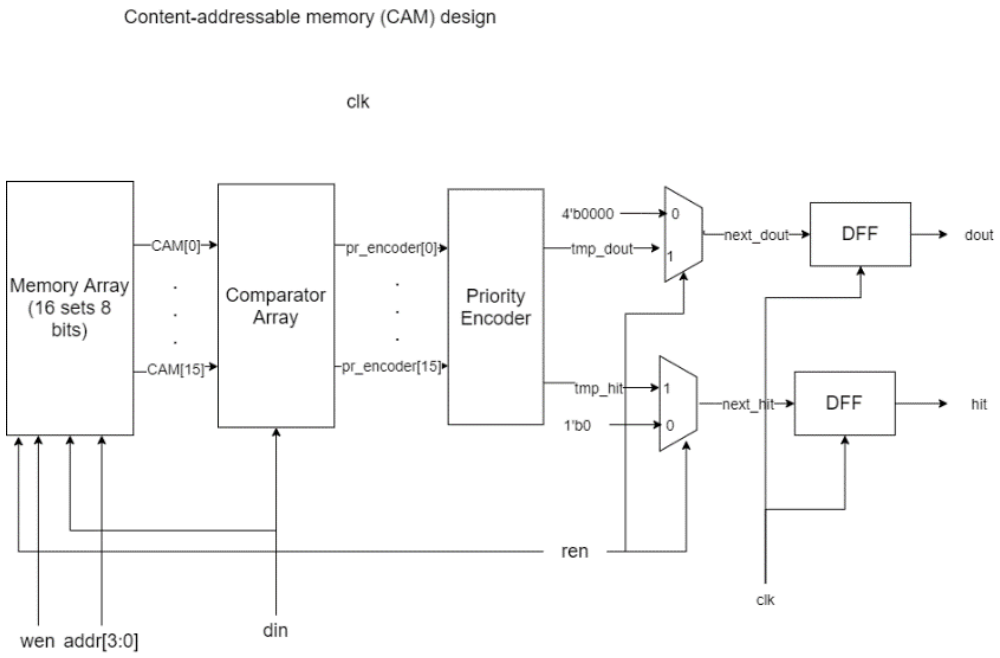


```
module Priority_Encoder(pr_encoder, next_dout, next_hit);
input [15:0] pr_encoder;
output [3:0] next_dout;
output next_hit;
reg [3:0] next_dout;
reg next_hit;
always@(*)begin
    casex(pr_encoder)
        16'b1xxxxxxxxxxxxxxx:begin
            next_dout = 4'b1111;
            next_hit = 1'b1;
        end
        16'b01xxxxxxxxxxxxxx:begin
            next_dout = 4'b1110;
            next_hit = 1'b1;
        end
        16'b001xxxxxxxxxxxxx:begin
            next_dout = 4'b1101;
            next_hit = 1'b1;
        end
        16'b0001xxxxxxxxxxxx:begin
            next_dout = 4'b1100;
            next_hit = 1'b1;
        end
        16'b00001xxxxxxxxxxx:begin
            next_dout = 4'b1011;
            next_hit = 1'b1;
        end
        16'b000001xxxxxxxxxx:begin
            next_dout = 4'b1010;
            next_hit = 1'b1;
        end
        16'b0000001xxxxxxxxx:begin
            next_dout = 4'b1001;
            next_hit = 1'b1;
        end
        16'b00000001xxxxxxxx:begin
            next_dout = 4'b1000;
            next_hit = 1'b1;
        end
        16'b000000001xxxxxxx:begin
            next_dout = 4'b0111;
            next_hit = 1'b1;
        end
        16'b0000000001xxxxxx:begin
            next_dout = 4'b0110;
            next_hit = 1'b1;
        end
        16'b00000000001xxxxx:begin
```

用 casex 舉出 16 種可能，就可以知道 next_dout 是什麼(ex:pr_encoder = 16’b0100101010101011，第 15 個 bit 的優先度最高，因此 next_dout = 4’b1110) next_hit 代表 pr_encoder 是否為 16’d0，若是則為 1’b0 反之為 1’b1。

CAM:

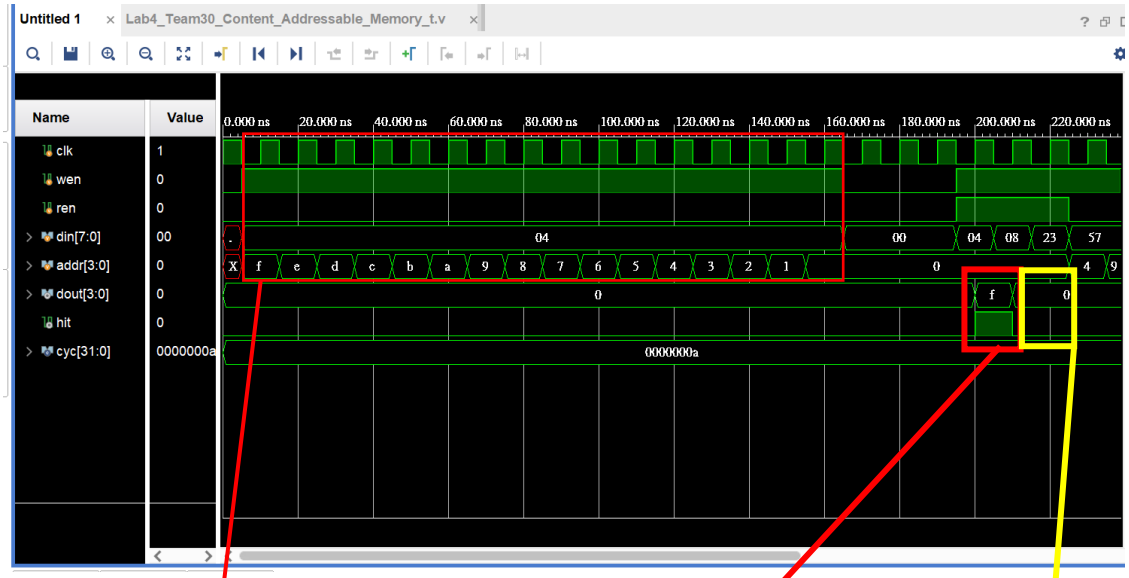
Block Diagram:



CAM 中的 memory array 用 reg [7:0]CAM[15:0]來模擬。當 ren == 1’b1 時，next_dout = tmp_dout，next_hit = tmp_hit。tmp_dout、tmp_hit 則是從 memory array 取值再經過 comparator array 和 priority encoder。{ren,wen} = 2’b01 時，把 din 存進 CAM[addr]中。每一個 posedge output hit、dout 都會更新他的值 dout <= next_dout hit <= next_hit。

波形圖:

下圖為確認當 memory 中的每一個 address 都是同一個值，當 din 是 address 中的值是否會輸出最大 address 而且 hit = 1’b1，並且 din 是 memory 中部存在的值是否輸出 1’b0 而且 hit = 1’b0。順帶檢查 ren = 1’b1 wen = 1’b1 時，會執行 read 的動作。

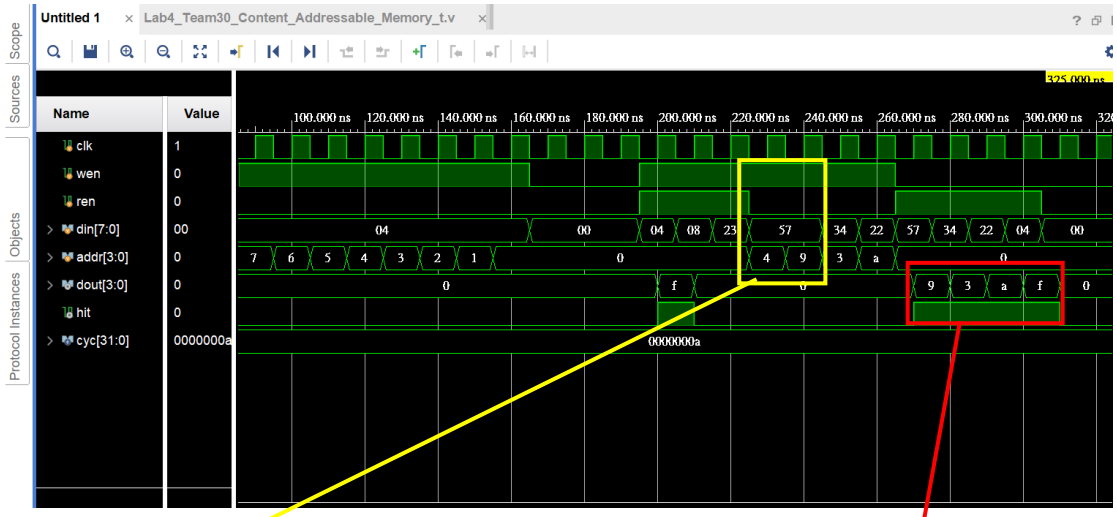


Memory 中所有位址都是 4’d4

確認當 ren,wen = 1’b1，din 在 memory 中是那些 address 並輸出最大的 address

確認當 ren,wen = 1’b1，din 在 memory 中是那些 address 如果沒有符合的 address，dout 輸出 4’b0000，hit = 1’b0

下圖為多試同一個 din 不同的 addr，在 ren = 1’b1 是否會輸出最大的 address



wen = 1’b1 ren = 1’b0，將 din = 4’d57 存進 memory 中的 4 和 9 的位址

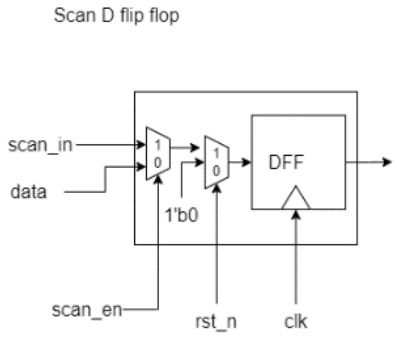
Ren = 1’b1 wen = 1’b0，din = 4’d57 在 memory 中有 2 個位址(4、9)，確認輸出最大的位址 9。確認其他 din 在 memory 中的位址確實輸出到 dout

2. Scan Chain Design

Scan Chain Design 由 8 個 Scan D-Flip Flop 和一個 4bit multiplier 所構成。

Scan D-Flip Flop

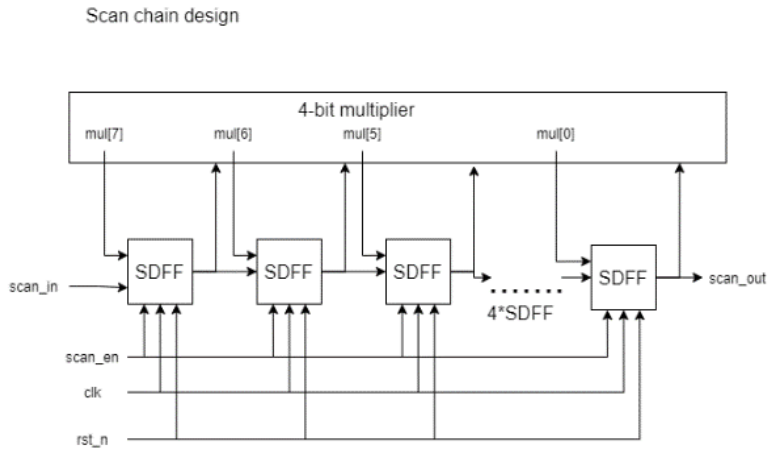
Block diagram:



Scan DFF 由兩個 input signal rst_n 和 scan_en 來決定甚麼值為 DFF 的 input。rst_n = 1’b0，DFF 的 input 為 1’b0，rst_n = 1’b1，scan_en = 1’b1，DFF 的 input 為 scan_in，scan_en = 1’b0，DFF 的 input 為 data。

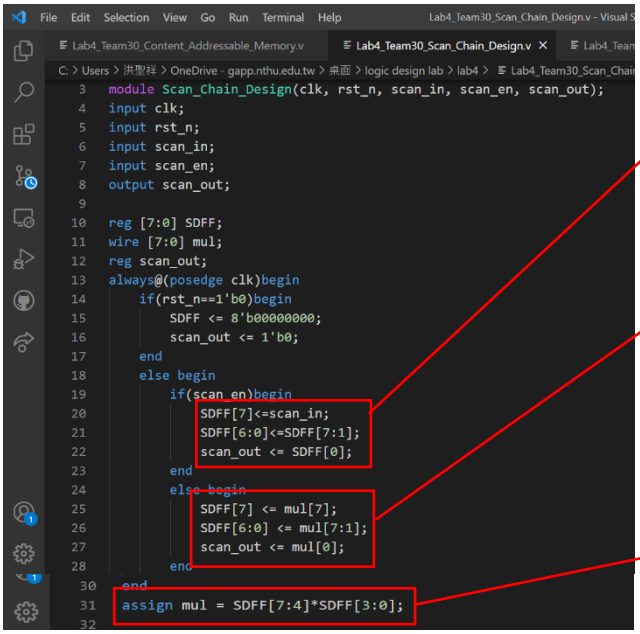
Scan Chain Design

Block diagram



Scan Chain Design 是當 rst_n = 1’b1 positive edge、scan_en = 1’b1 時，scan_in 會 1 個 bit 1 個 bit 從左傳到右最後到 output scan_out。scan_en = 1’b0 時，4bit multiplier 的每一個 bit 會傳入 SDFF 並 1 個 bit 1 個 bit 從左傳到右最後到 output scan_out。

Code:



```
3  module Scan_Chain_Design(clk, rst_n, scan_in, scan_en, scan_out);
4  input clk;
5  input rst_n;
6  input scan_in;
7  input scan_en;
8  output scan_out;
9
10 reg [7:0] SDFF;
11 wire [7:0] mul;
12 reg scan_out;
13 always@(posedge clk)begin
14     if(rst_n==1'b0)begin
15         SDFF <= 8'b00000000;
16         scan_out <= 1'b0;
17     end
18     else begin
19         if(scan_en)begin
20             SDFF[7]<=scan_in;
21             SDFF[6:0]<=SDFF[7:1];
22             scan_out <= SDFF[0];
23         end
24         else begin
25             SDFF[7] <= mul[7];
26             SDFF[6:0] <= mul[7:1];
27             scan_out <= mul[0];
28         end
30 end
31 assign mul = SDFF[7:4]*SDFF[3:0];
32
```

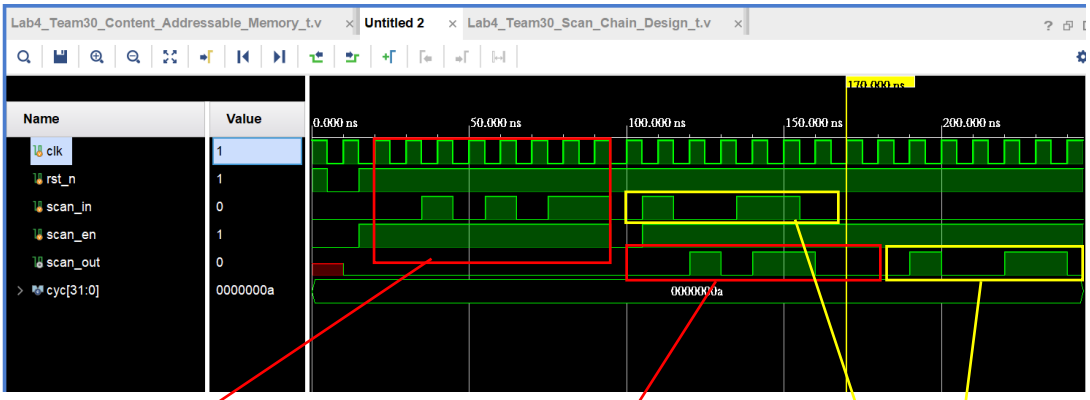
scan_in 1 個 bit 1 個 bit 從左傳到右

4 bit multiplier 的 output 接到對應的 SDFF

4 bit multiplier

波形圖:

下圖測試在 rst_n = 1’b0 後 rst_n = 1’b1、scan_en = 1’b1 經過 8 個 positive edge 後 scan_en = 1’b0 再經過 7 個 positive edge，scan_out 的值是否正確，並確認是否能 overlap scan_in



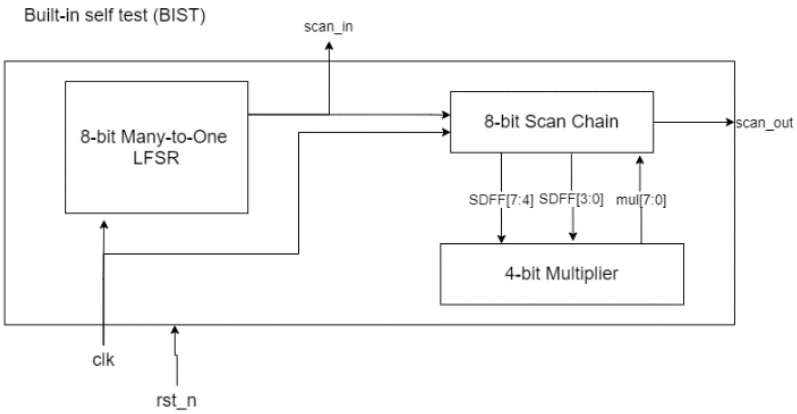
scan_in 依序為 0,0,1,0,1,0,1,1 因為最早輸入的 scan_in 最早傳到最右邊的 SDFF，因此我們的 2 個 4bit multiplier 的 input 為上述的值反過來，也就是 11010100，因此 2 個 input 為 1101 以及 0100。

4 bit multiplier 的 2 個 input 為 1101 以及 0100，因此 output 為 00110100。因為 least significant bit 輸入最右邊的 SDFF 所以最早輸出的 output scan_out 為 least significant Bit，因此 scan_out 依序為 multiplier 每個 bit 反過來 00101100。

確認兩波形圖一樣(scan_in scan_out overlap)

2. Built-in self test (BIST)

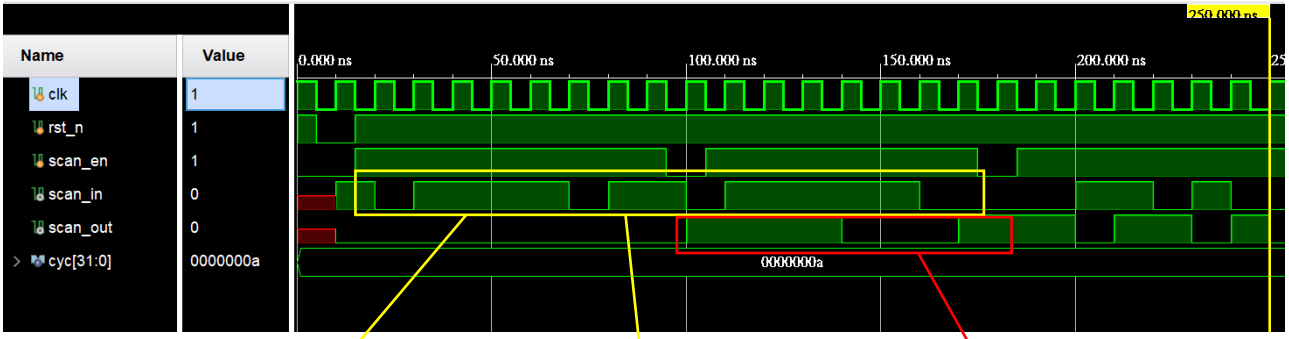
Block diagram:



BIST 由 Many-to-One LSFR 、8 bit Scan Chain 和 4 bit multiplier 所組成。只是將 advanced question2 的 scan chain design 以 Many-to-One LSFR output 的 most significant bit 為 scan chain 的 input。

波形圖:

下圖為確認 Many-to-One LSFR 是否正確，以及 scan_en = 1’b0 後 scan_en 再次 = 1’b1 後 scan_out 的值是否為 4 bit multiplier 輸出的值。也確認 scan_in scan_out 可以 overlap。

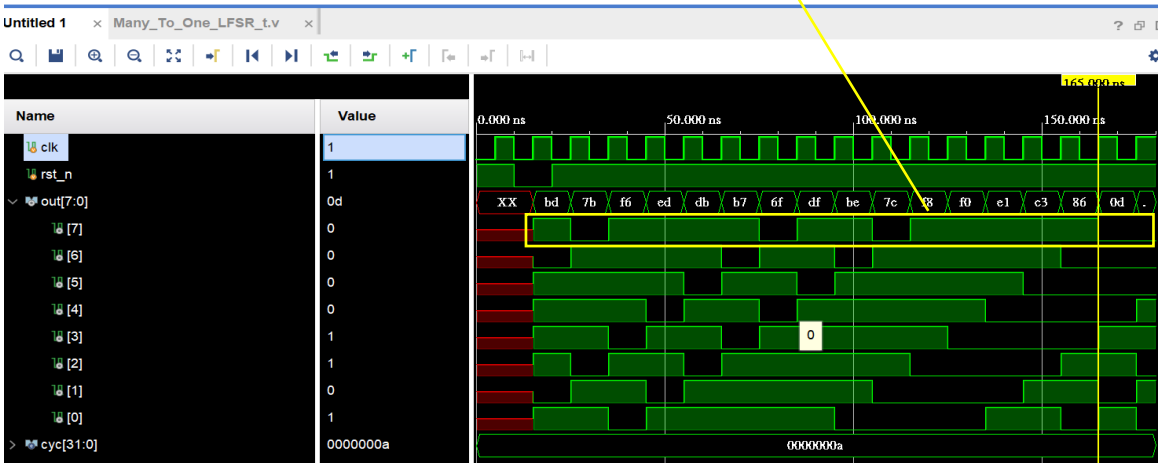


前 8 個 cycle Multiplier 的 input 為 1011 和 1101

波形圖與 M_T_O_LFSR output 的 most significant 相符

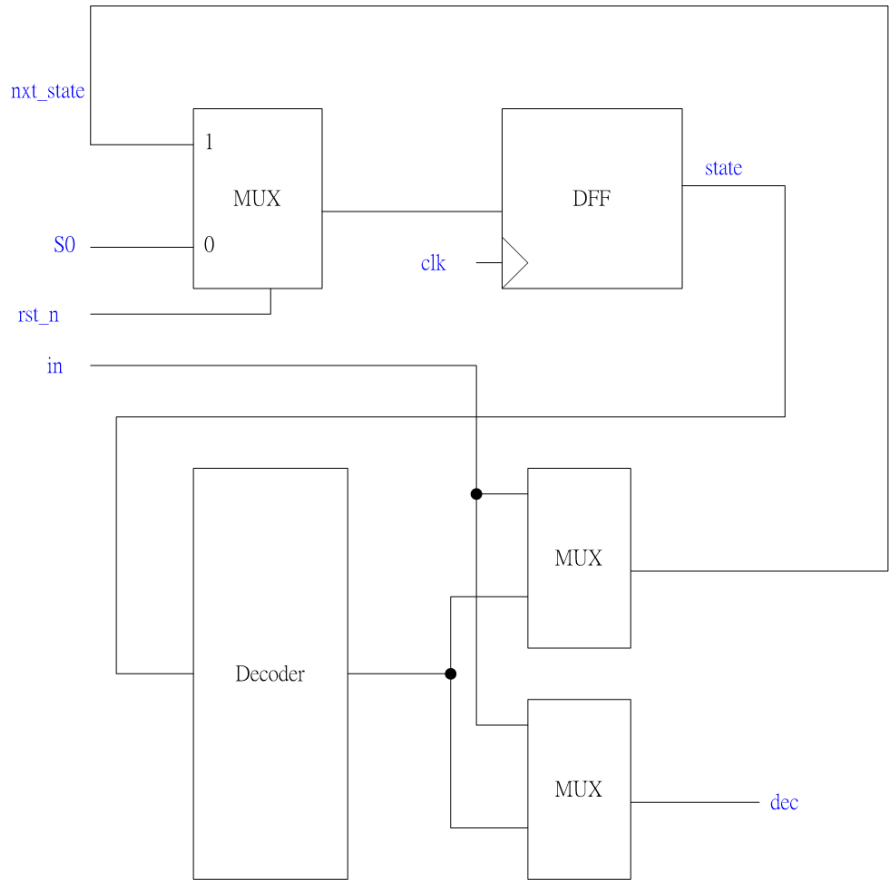
11*13 = 143 = 8'b10001111 從 lsb 輸出，scan_out 依序為 11110001

Many_To_One_LFSR 波形圖



4. Mealy machine sequence detector

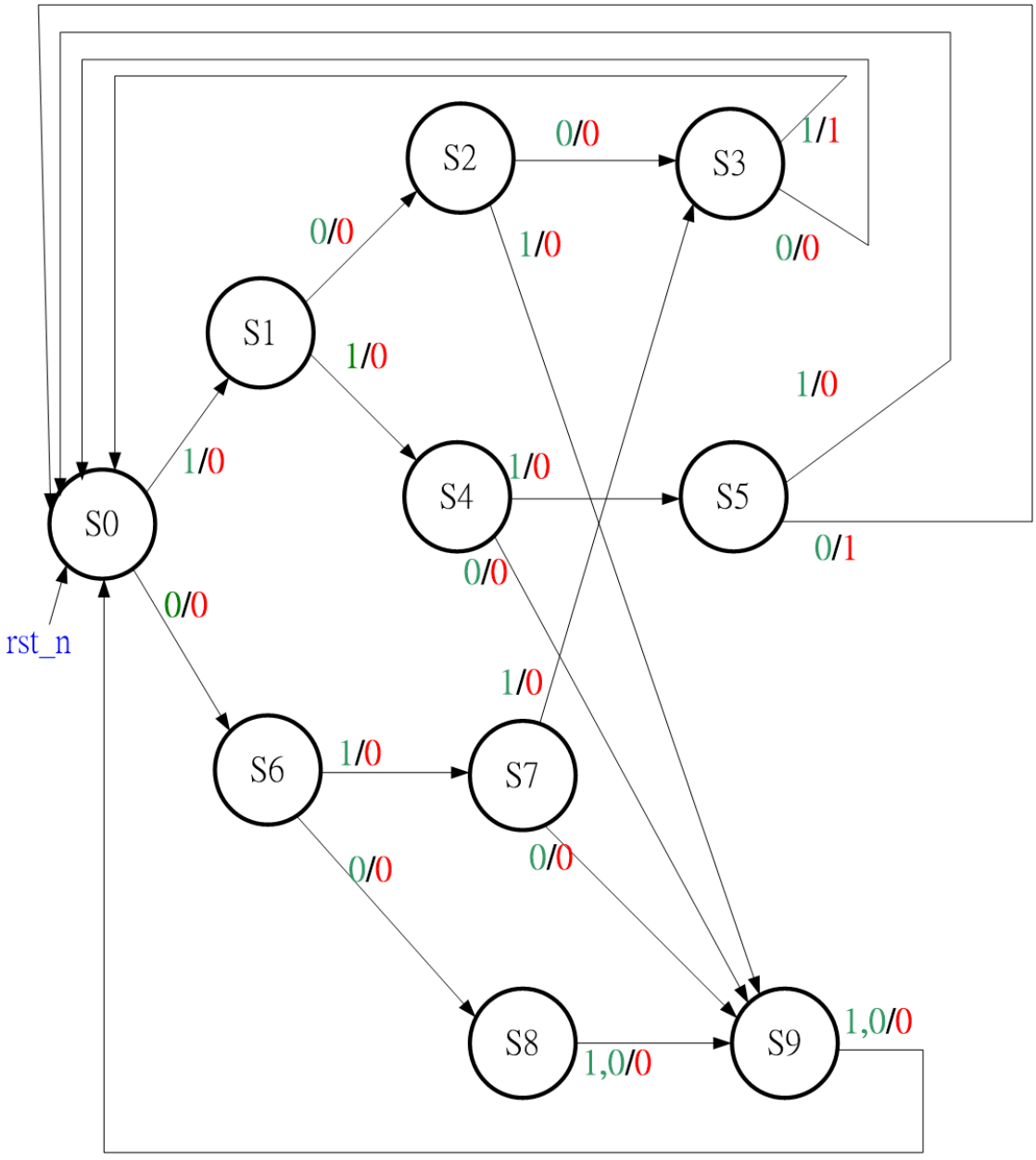
Block diagram:



這題的 mealy machine sequence detector 要偵測 0111、1001、1110 三個 sequence，並在輸入四個 input 後就返回到 initial state 再次進行偵測，input 有 clk、rst_n、in，output 是 dec，module 內部有 state、nxt_state 來運作 mealy machine。一開始用一個 MUX 選擇如果 rst_n = 0 的話輸入 S0 到 DFF，反之輸入 nxt_state，接著 DFF 將 state 輸入到一個 decoder 判斷目前在哪個 state，最後再輸入到 MUX 由 in 判斷輸出 dec 的值與下個 cycle 的 nxt_state。

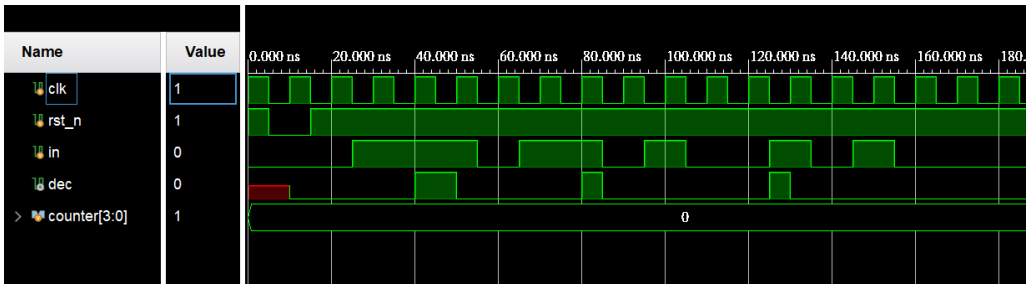
```
module Mealy_Sequence_Detector (clk, rst_n, in, dec);
input clk, rst_n;
input in;
output reg dec;
reg [3:0] state;
reg [3:0] nxt_state;

parameter S0 = 4'b0000;
parameter S1 = 4'b0001;
parameter S2 = 4'b0010;
parameter S3 = 4'b0011;
parameter S4 = 4'b0100;
parameter S5 = 4'b0101;
parameter S6 = 4'b0110;
parameter S7 = 4'b0111;
parameter S8 = 4'b1000;
parameter S9 = 4'b1001;
```

State diagram:

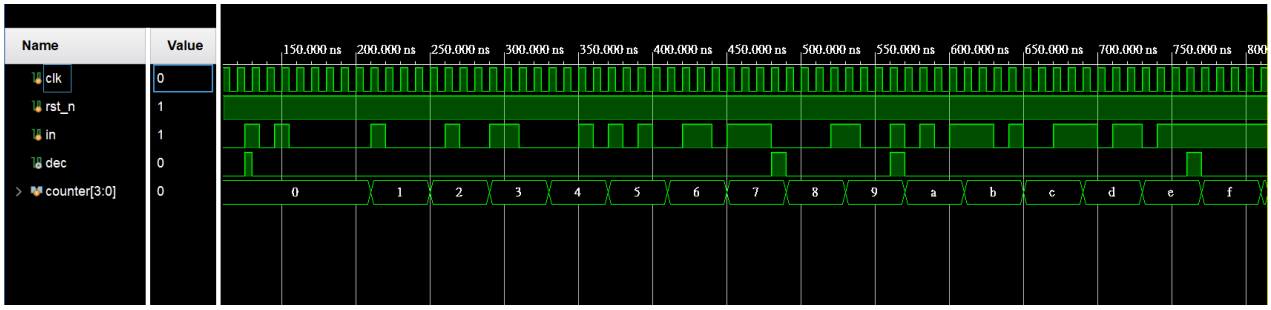


rst_n == 0 時 state 會到 S0，然後開始接收 input sequence 並隨著 clk 變換 state，如果連續在四個 state 接收到 0111、1110 或 1001 的話會 output 1，不是以上的 sequence 則 output 0，而因為是 mealy machine，output 會由 state 和 input 決定，所以輸出是 asynchronous 的。所有的路徑都會在 4 個 cycle 之後再次回到 S0，重新偵測新的 4-bit sequence。

波形圖：



先按照講解 ppt 上的波形範例測試，確認波形無誤



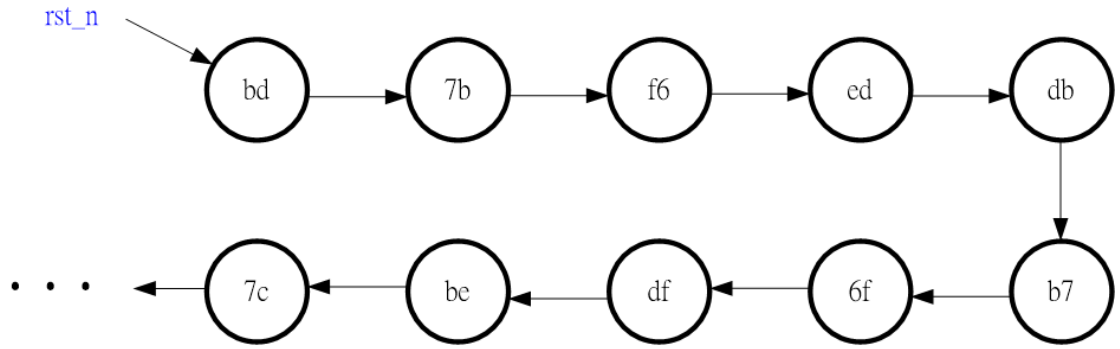

```
    @(negedge clk)
    repeat(16) begin
    in = counter[0];
    @(posedge clk)
    in = counter[1];
    @(posedge clk)
    in = counter[2];
    @(posedge clk)
    in = counter[3];
    @(posedge clk)
    counter = counter + 1'b1;
    end
    @(negedge clk)
    $finish;
    end

    endmodule
```

再來利用一個 4-bit counter 來輸入 input sequence，以四個 cycle 為單位，依序輸入 counter[0] ~ counter[3]，然後再將 counter + 1，如此一來便能測試所有的 4-bit input sequence 組合，可以看到只有在 sequence 為 0111、1001、1110 時 dec 才會 output 1。

5. Basic question 3、4

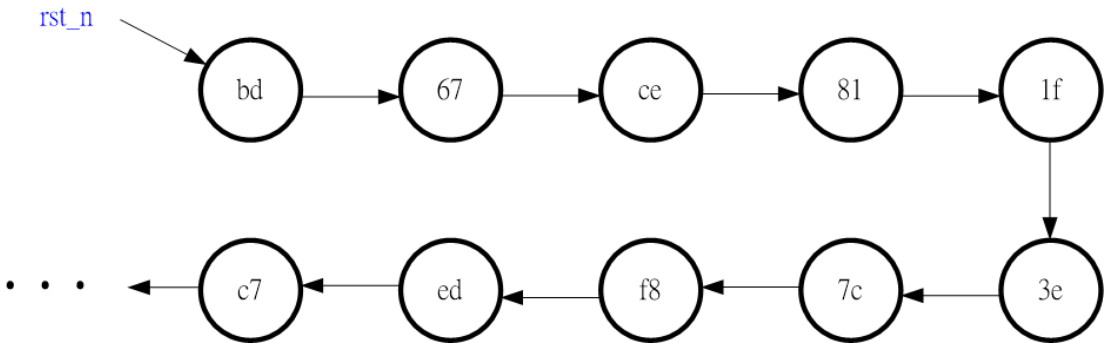
- Many-to-one LFSR

State diagram:



- One-to-many LFSR

State diagram:



這兩種 LFSR 如果把 DFFs reset 成 8’d0，會使 DFF xor 之後的結果永遠都是 0，造成每個 state 的 output 都是 0，就失去本來要產生亂數的目的了。

6. 心得

洪聖祥: 這次 lab 終於比較簡單，spec 上 block diagram 也直接畫出來，code 的部分就直接照著圖上打就好了。 這次比較花時間的地方是寫 report。雖然說 spec 已經把 block diagram 的大概方向畫出來了，但要詳細解釋設計細節就比較困難了。從這次 lab 我發現如果有 block diagram 再寫 code 是一件很簡單的事情，因為只要描述這個 block 的行為然後把 block 跟 block 連起來就行了。

劉奇泓:這次的 lab 第一次開始寫 FSM，我熟悉了 mealy machine 和 moore machine 的差別，也了解把每個 state 定義清楚，並畫好完整的 state diagram 是很重要的!尤其是在第四題 mealy sequence

detector 把 state diagram 畫好後照著打出來，就能很快地做好了，除此之外這次的 lab 也學到了 LFSR 的運作，並在 fpga 題運用，我學會了如何用硬體的角度來製作亂數產生器。希望這些經驗都能累積起來，在下次的 lab 提升效率，使我有能力處理更多難關。

7. 分工

洪聖祥:advanced question 1、2、3，report 1、2、3

劉奇泓: basic question 3、4 state diagram ，advanced question 4、fpga 題，report 4、fpga 題。