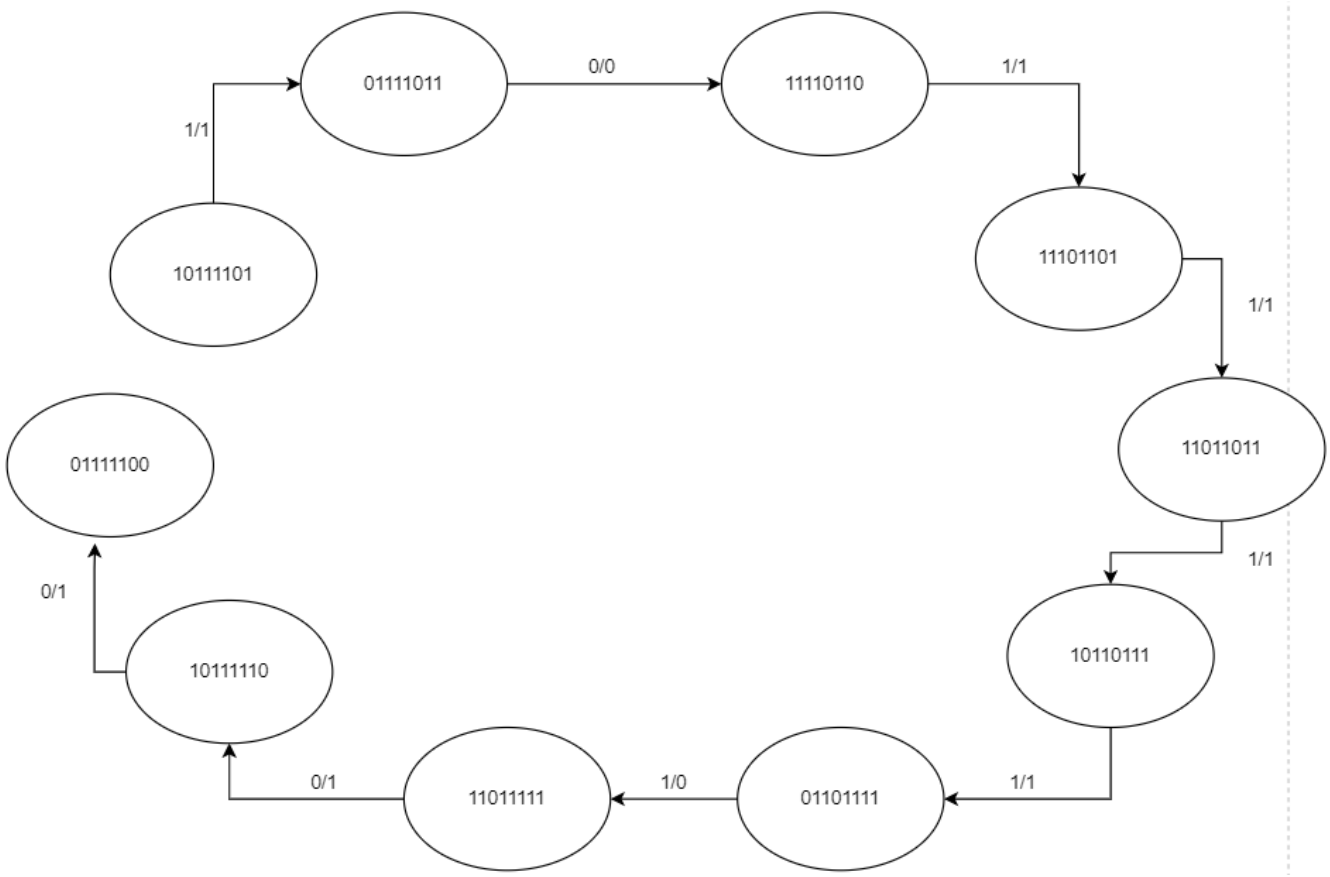
Lab 4 Team 20 report

109062222 徐嘉徽

109062119 李佳栩

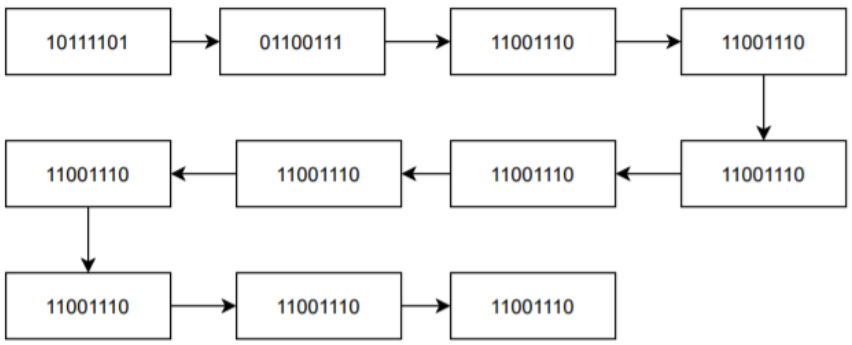
Basic 3:



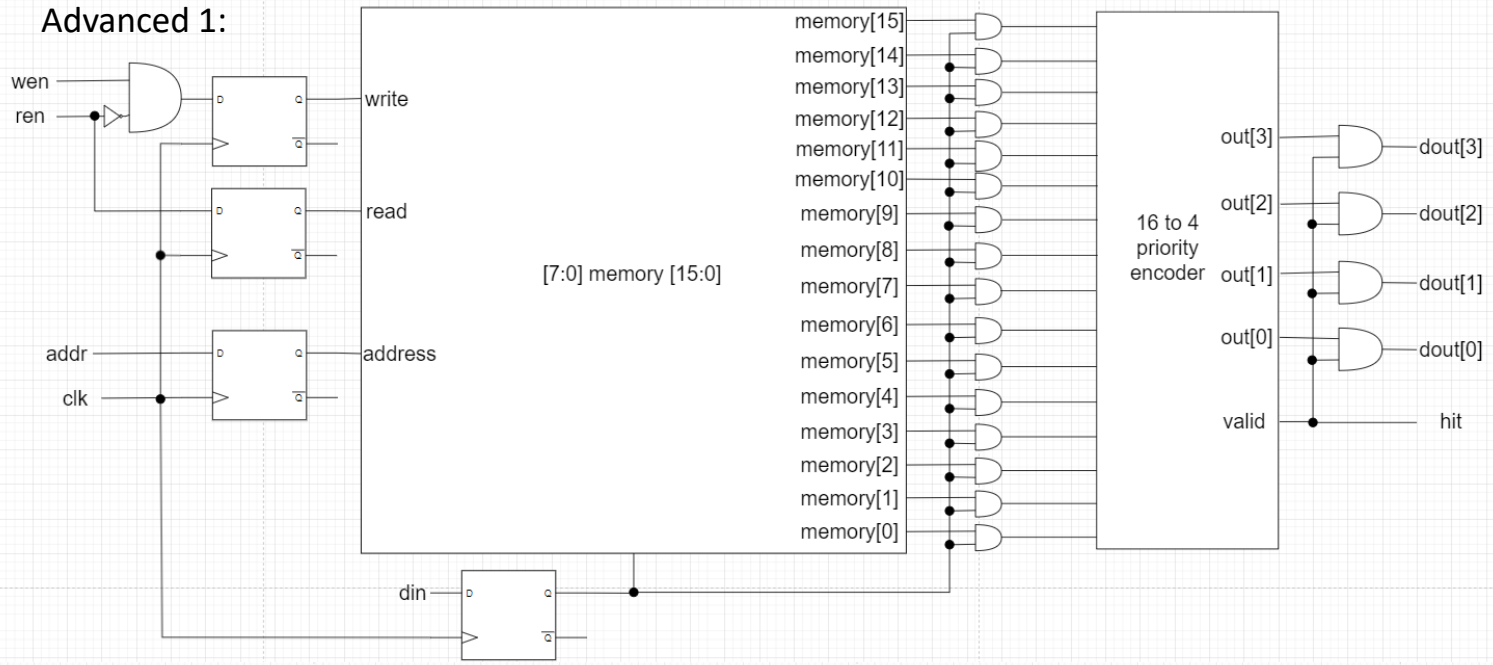
如果將 DFF reset 到 8’b0，會導致之後的每個 DFF[7:0]值都是 8’b0，out 也永遠是 1’b0。

Basic_4:

如果將 DFF reset 到 8’b0，會導致之後的每個 DFF[7:0]值都是 8’b0，out 也永遠是 1’b0。



Advanced 1:

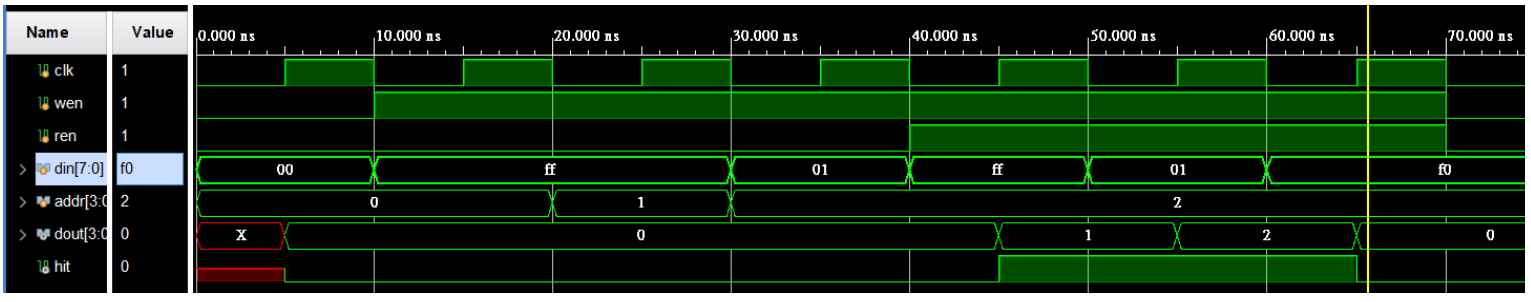


Truth table for 16 to 4 priority encoder

Input																Output				
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	Y3	Y2	Y1	Y0	Valid
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
X	X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	0	1	1	0	1
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	1	1	1	1
X	X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0	0	0	1
X	X	X	X	X	X	X	X	X	1	0	0	0	0	0	0	1	0	0	1	1
X	X	X	X	X	X	X	X	X	X	1	0	0	0	0	0	1	0	1	0	1
X	X	X	X	X	X	X	X	X	X	X	1	0	0	0	0	1	0	1	1	1
X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	0	1	1	0	0	1
X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	0	1	1	0	1	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	1	1	1	0	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	1	1	1

首先，我們利用 lab 3 basic question 2 中的 Memory Array 作一點改變，每個 word 設定為 8 個 bit([7:0])，且總共有 16 個 word([15:0])。在這次 lab 中，我們對 Memory Array 作了以下改變 ： 當 ren==1 時，memory 會同時讀出 16 個位置中的每個 word，而當 ren==0 && wen==1 時，會將 din 的值存入到 addr 在 memory 的相應位置且不會有任何 word 被 memory 輸出。所以當 ren==0 && wen==1 時，output 跟 hit 不重要，而當 ren==1 時，我們再將 memory 所輸出的 16 個 word[7:0]分別與 din[7:0] and 起來，也就是比較每個 word 是否與 din 的值相同。比較完後會有三種情況 ：(1)所有 word 都跟 din 不同，hit=0 (2)有一個 word 與 din 相同，輸出位置且 hit=1 (3)有超過一個 word 與 din 相同，只輸出 memory 位置最高的，hit=1。為了完成(1)(2)(3)，我們利用 16 to 4 priority encoder，這樣便能只輸出最高的 memory 位置，而 valid 的值表示有沒有 word 跟 din 一樣，也就是 hit 代表的意思。但 16 to 4 priority encoder 較難利用 truth table 實做出來，附上他的 truth table 顯示它的功用。

Testbench 的部分，我們將 10ns 設為一個 clk cycle，並且測試上述的(1)(2)(3)種情況。首先讓 ren=0,wen=1,din=8’b11111111 (ff) 且 addr=4’b0000，並過一個 cycle 後讓 addr=4’b0001，這樣便讓 8’b11111111 存到 memory[0]和 memory[1]中。再一個 cycle 後讓 din=8’b00000001 (01)且 addr=4’b0010，讓 8’b00000001 存入 memory[2]。下一個 cycle 將 ren=1,wen=1，並每隔一個 cycle 讓 din=8’b11111111(ff)、8’b00000001(01)、8’b11110000(f0)，以分別測試第(3)，第(2)，第(1)種情況，跑出來的 waveform 如下 ：



每種情況所對應的 dout 跟 hit 皆符合，代表此設計符合 CAM 的功能。

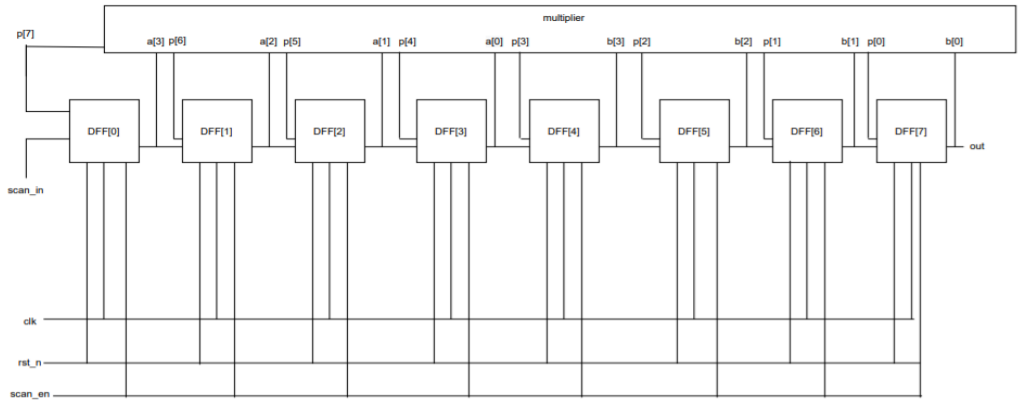
Advance_2:

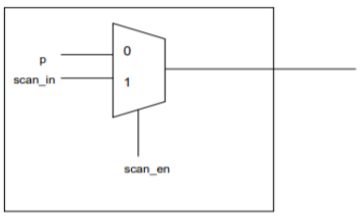
這題是要我們做出一個 scan-chain，總共有三個階段，分別是 scan_in, capture, scan_out，一開始我寫的時候有分成三個 if-else，後來有同學在討論區上面問說 in 跟 out 重疊的情況，後來仔細想想才發現是可以重疊的，所以寫成，當 scan_en 等於 0 的時候，我的 dff 再去抓 p 的值，也就是 a*b。

```
if(scan_en) begin
    dff[7] <= scan_in;
    dff[0] <= dff[1];
    dff[1] <= dff[2];
    dff[2] <= dff[3];
    dff[3] <= dff[4];
    dff[4] <= dff[5];
    dff[5] <= dff[6];
    dff[6] <= dff[7];
    scan_out <= dff[1];
end
else begin
    scan_out <= p[0];
    dff[7:0] <= p[7:0];
end
```

另外我的 combinational circuits 則是負責寫乘法器的部分，我原本寫在 sequential circuits 裡面，但沒想到的是 sequential circuits 裡面是用舊的值，導致我的答案會錯，後來拉出來寫就可以準確抓到 a 跟 b 的值。

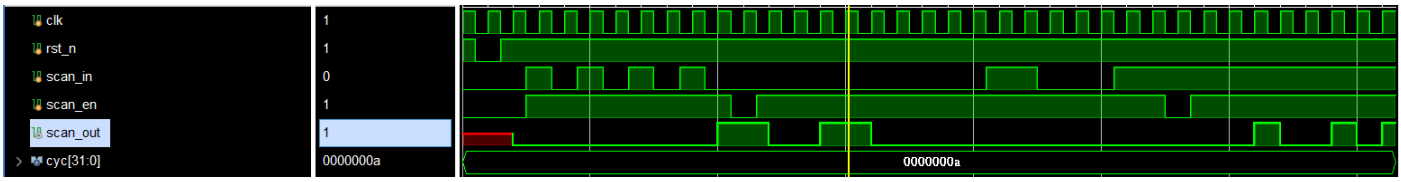
下圖是整個 scan_chain 的示意圖。





測試方法：

餵給 scan_chain 不同的值觀察 output，不過因為每次都是檢查一個 bit 太慢了，而且 scan_in 的順序是反過來的，在檢查的時候也很麻煩，所以我在 testbench 有把 a,b,p 給設成 output，以方便觀察。我用的數字是 5*5 和 6*12，結果都有滿足我的答案。

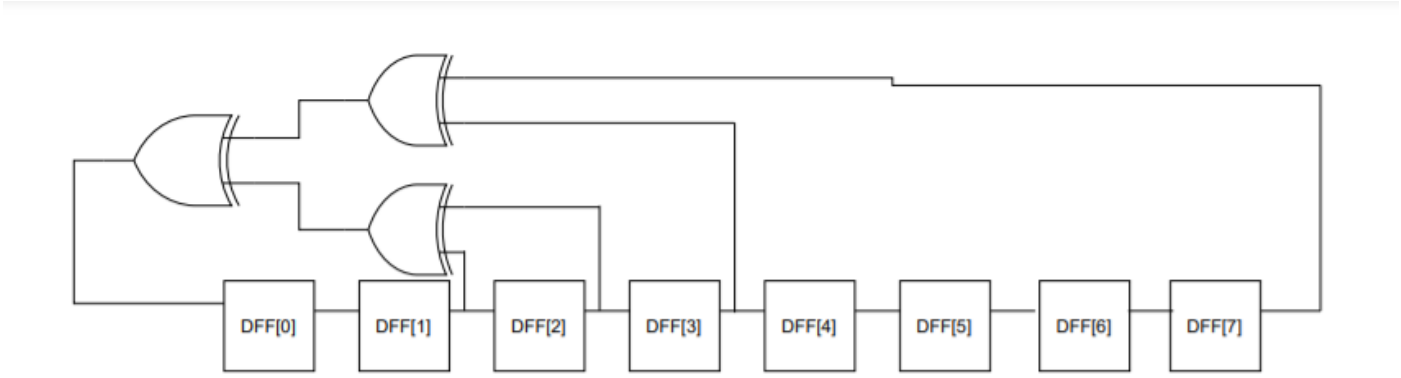


Advance_3:

這題是利用 basic 和 advance 的題目做成的題目，認識了 BIST，主要作法是把 LFSR 產生的值當成 scan_in 傳給 Scan_Chain，唯一要做的就是把 LFSR 改成每次只傳 MSB 就好了。

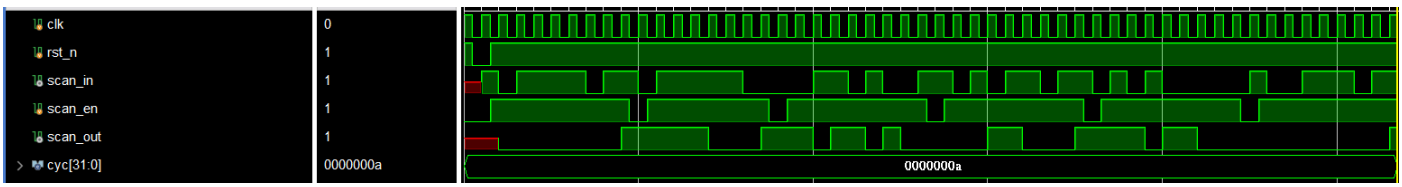
下圖是 LFSR 的示意圖。

DFF[7]是我的 output。

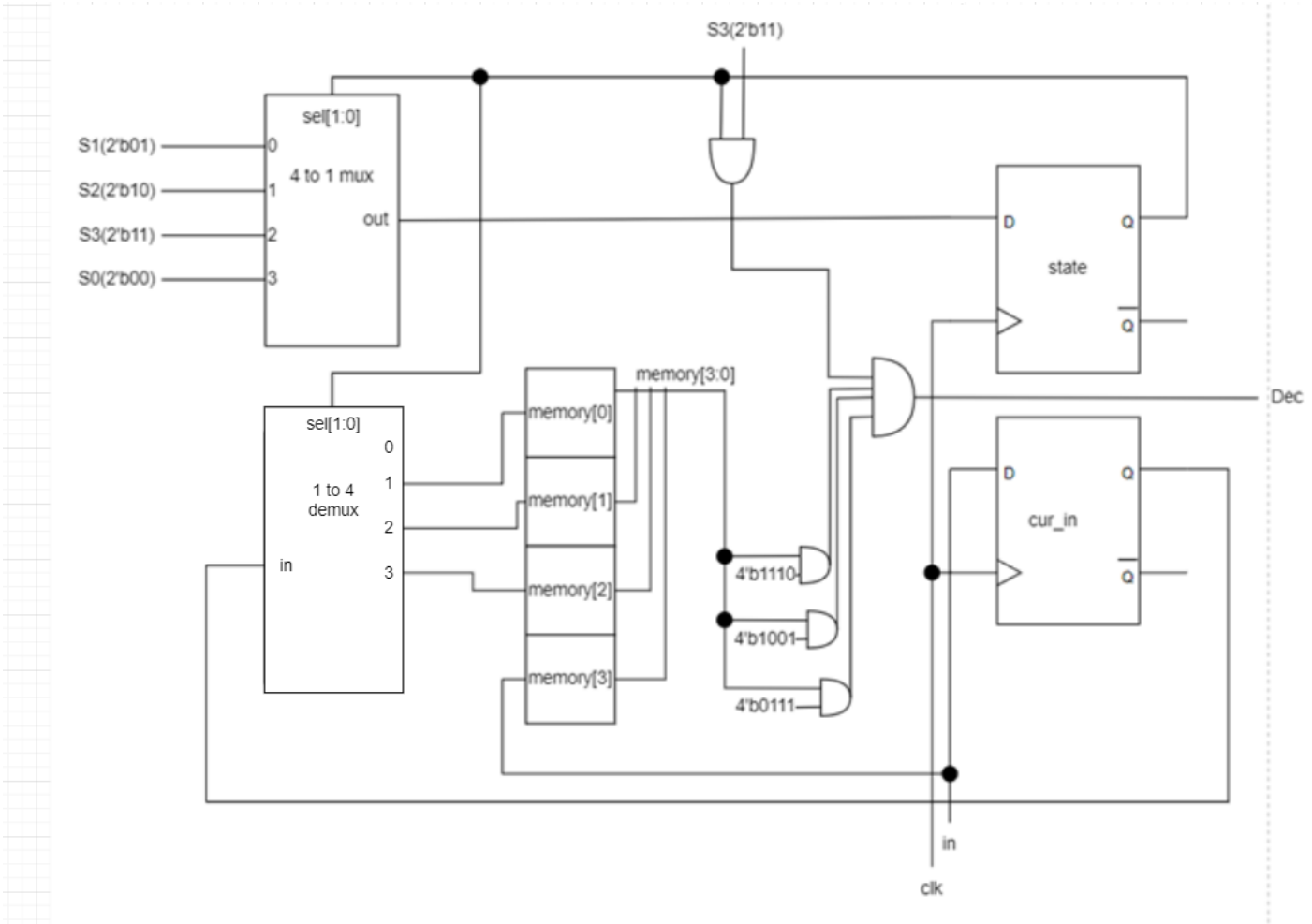


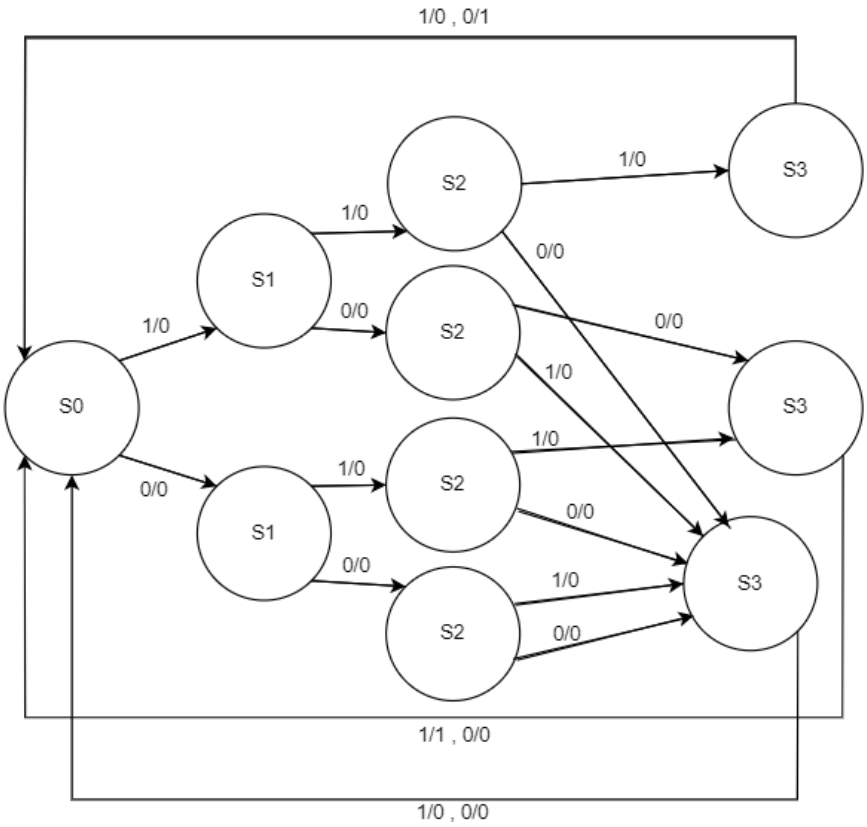
測試方法：

控制 Scan_in, capture, Scan_out 的階段，觀察 output 有沒有等於 a*b。一開始先 reset，然後開始丟數字進去，那我是每 scan 到 8 個數字就 scan_out，這樣也可以順便測試 overlap 的狀況。



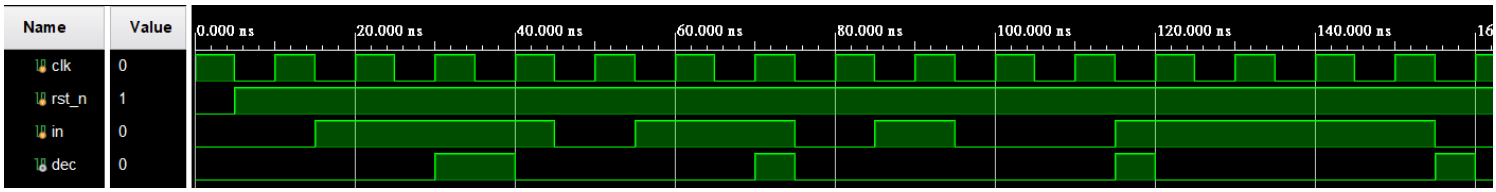
Advance_4:


S3(2'b11)
sel[1:0]
S1(2'b01)
S2(2'b10)
S3(2'b11)
S0(2'b00)
0
1
2
3
4 to 1 mux
out
D
Q
state
Q
memory[3:0]
Dec
sel[1:0]
0
1
2
3
1 to 4
demux
in
memory[0]
memory[1]
memory[2]
memory[3]
4'b1110
4'b1001
4'b0111
D
Q
cur_in
Q
in
clk


1/0 , 0/1
S0
S1
S1
S2
S2
S2
S3
S3
S3
1/0
0/0
1/0
0/0
1/0
0/0
1/0
0/0
1/0
0/0
1/0
0/0
1/1 , 0/0
1/0 , 0/0

這題利用四種 state : S0、S1、S2、S3 的 mealy machine 來完成 sequence detector。首先利用兩個 DFF 代表 state 與 cur_in。在 state 的 input(也就是 next_state)利用 4 to 1 mux 來完成 state 轉換，並在 sel 接上 state 的 output，當 sel==S0，next_state=S1、sel==S1，next_state=S2、sel==S2，next_state=S3、sel==S3，next_state=S0。在每一個 cycle，都會將 in 記錄成 cur_in，並利用一個 memory[3:0]來記錄 sequence 的每一個 bit，我們利用 1 to 4 demux 來完成這件事，選擇方法如圖，memory[3]則是直接將接上 in，不受到 clk 的影響。由於我們只有在 S3 時會記錄完 4 個 bit 並決定 sequence 是否符合 4'b1110、4'b1001、4'b0111，因此我們將 memory[3:0]分別跟 4'b1110、4'b1001、4'b0111 判斷是否一樣，並再將結果 or 起來，最後再判斷 state 是否為 S3，便能決定 Dec。

Testbench 的部分我們，我們將 10ns 設為一個 clk cycle。並測試過每個會讓 dec=1 的 sequence，也就是 4'b1110、4'b1001、4'b0111，同時也測試了幾個隨機的 input sequence，以觀察 dec 是否只會在 S3 時變成 1。測試結果如下:



分工: 徐嘉徽寫 1、4，李佳栩寫 2、3 ，report 的分工也一樣

心得:

徐嘉徽：這次要利用 state transition 來設計出題目要求的功能及電路，雖然 code 變的簡短許多，但要思考的事情也變得更多，尤其是要注意每個 clk cycle 時的 state 是否正確，以及 next_state 是否符合要求。最重要的是這次的 advanced 有使用 CAD 來測試，上次沒有使用 CAD 測試，導致最後改出來的結果跟預期相差很大，我們之後也會多加利用 CAD 測試，才不會再出現意料之外的錯誤。

李佳栩：這次我負責的是 2.3 題，雖然不是 FSM 的題目，不過也讓我在寫 code 的時候發現自己容易粗心的毛病，這次的內容也比較簡單，根據圖片接起來就可以了，不過也是有些小問題要注意，像是我一直都很不熟悉 sequential circuits 跟 combinaitonal circuits 分開的寫法。還有經過前一次 lab 的教訓，這次都有使用 CAD 測試，希望這次的作業能夠達到我們給自己的目標。