

Lab4_Team1_Report

組員

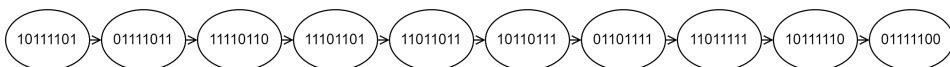
107071016 施泳瑜

109062320 朱季葳

Basic Question3

Drawing the state transition diagram of the DFFs in LFSR for the first ten states after `rst_n` is raised to 1'b1

`DFF[7:0] = 8'b10111101` 之往後 10 個 state transition 圖：

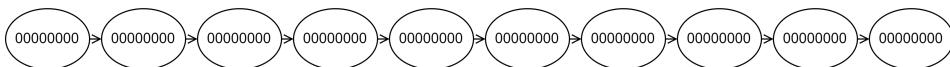


(the result from the testbench)

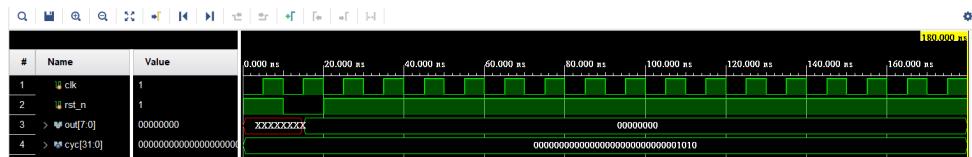


Describe what happens if we reset the DFFs to 8'd0

若 `DFF[7:0] = 8'b00000000`，則每一個 state 都會是 8'd0：



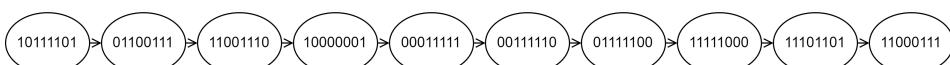
(the result from the testbench)



Basic Question4

Drawing the state transition diagram of the DFFs in LFSR for the first ten states after `rst_n` is raised to 1'b1

`DFF[7:0] = 8'b10111101` 之往後 10 個 state transition 圖：

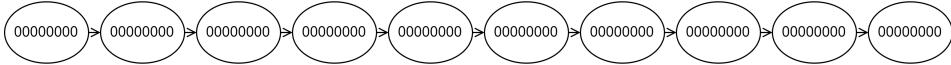


(the result from the testbench)



Describe what happens if we reset the DFFs to 8'd0

若 DFF[7:0] = 8'b00000000，則每一個 state 都會是 8'd0：



(the result from the testbench)



Advanced Question1

Drawing of the design of Verilog Advanced Question 1

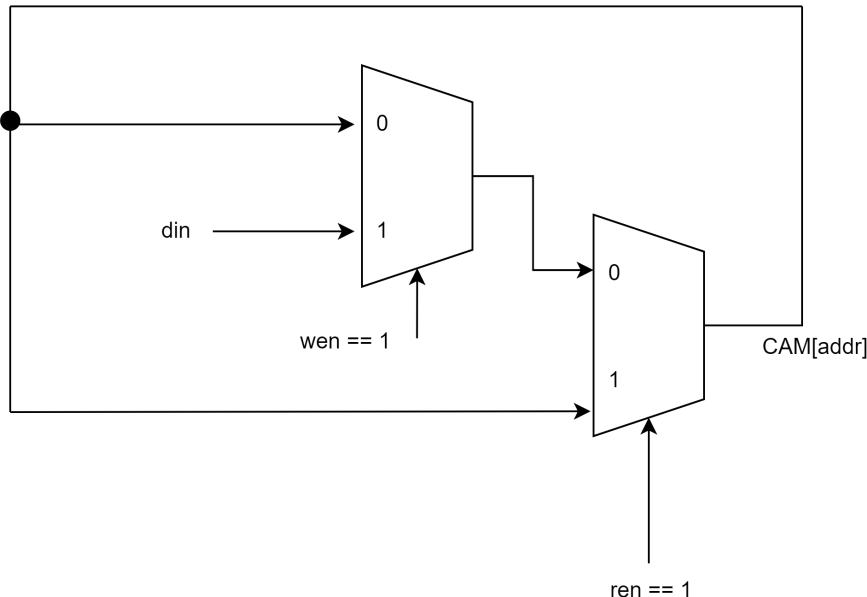
SEQUENTIAL PARTS

dout & hit

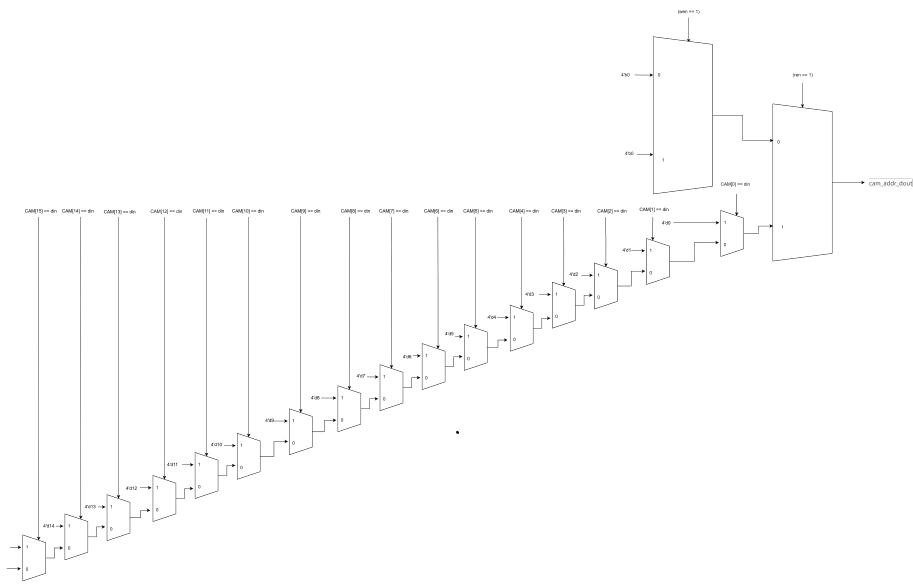


COMBINATIONAL PARTS

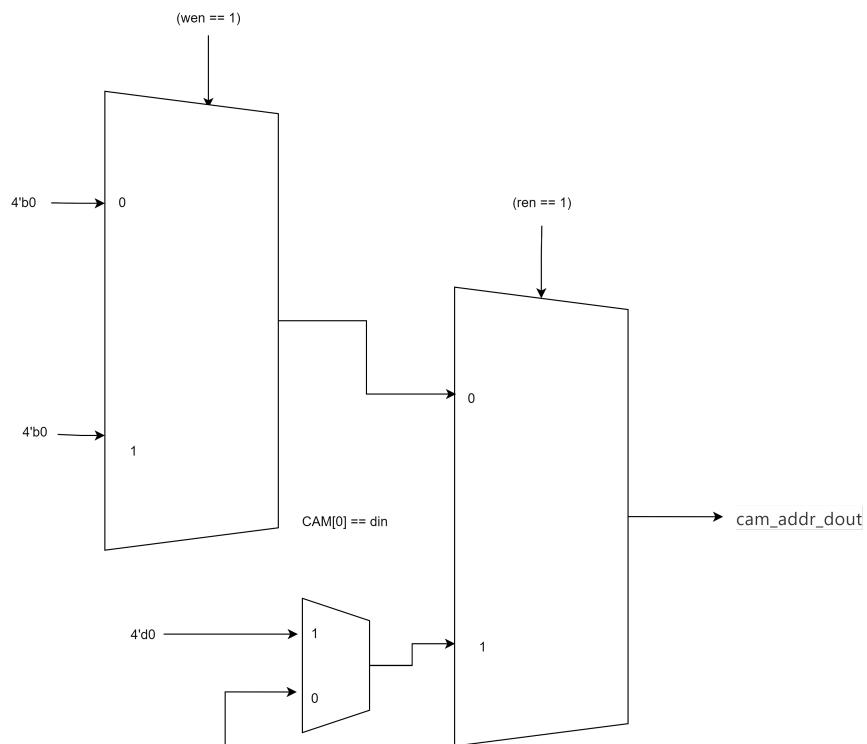
1. CAM



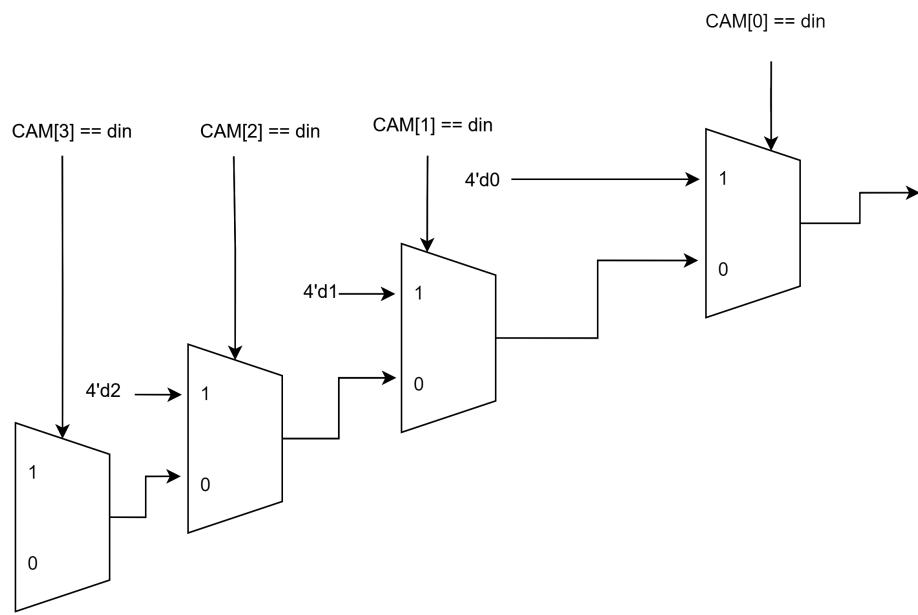
2. cam_addr_dout



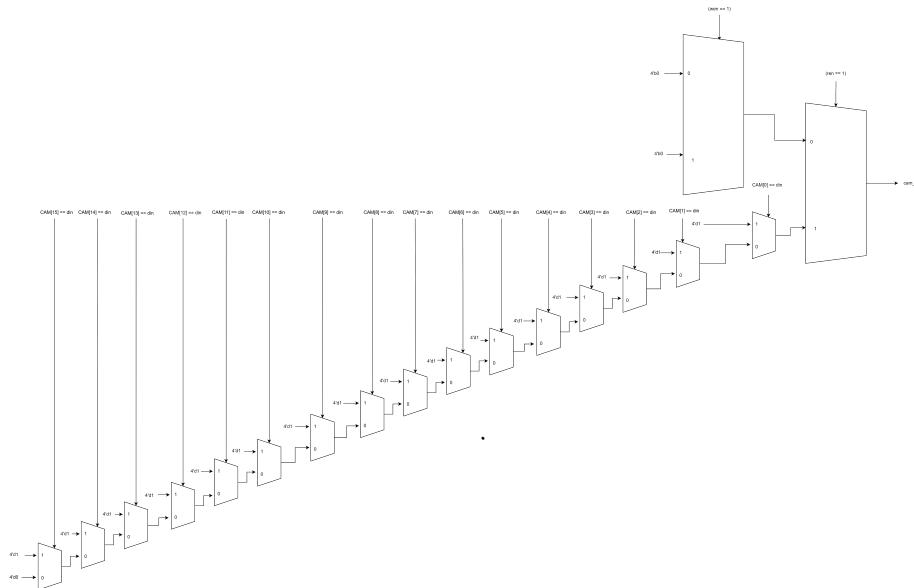
a. 右邊局部放大圖



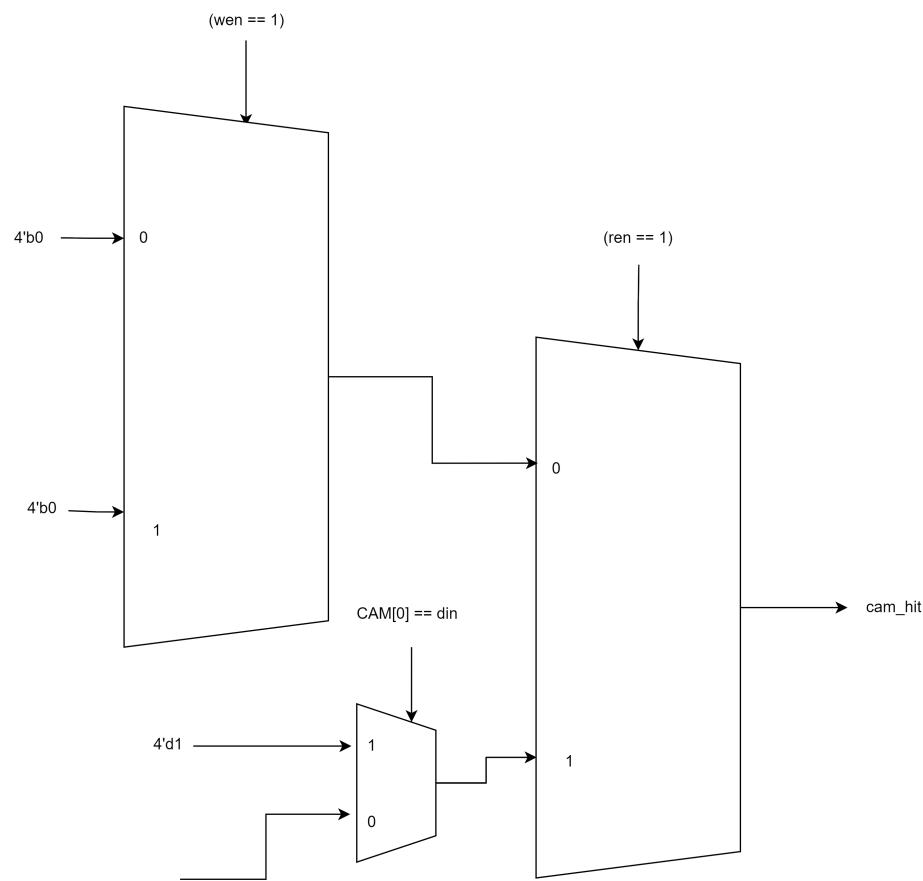
b. 左邊局部放大圖 (銜接上者)



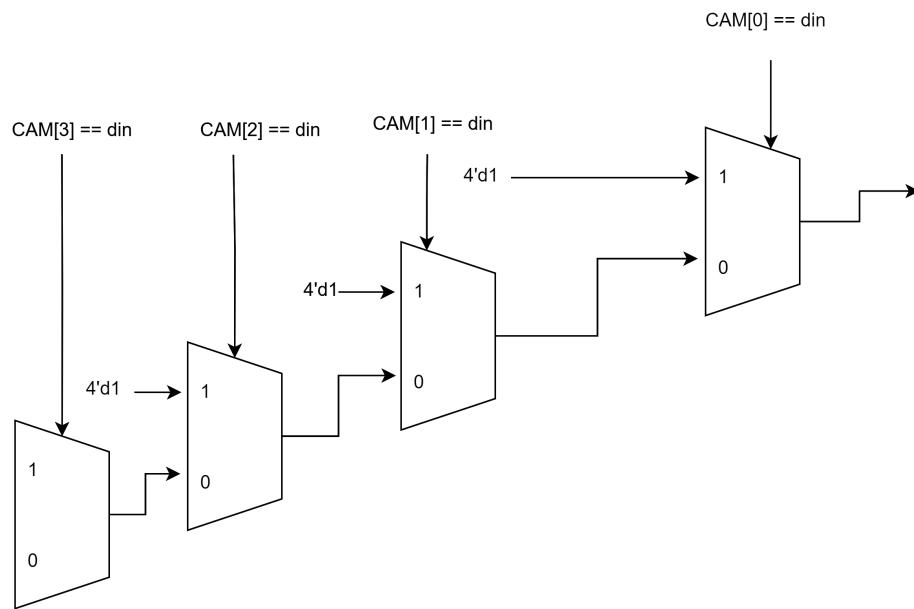
2. cam_hit



a. 右邊局部放大圖



b. 左邊局部放大圖 (銜接上者)



Requirements

1. 當 $(ren, wen) = (0, 1)$ 時
 - a. 把 din 寫入 $CAM[addr]$
 - b. $dout = 4'd0$ & $hit = 1'b0$

2. 當 $(ren, wen) = (1, 0)$ 時

- a. 找得到對應得值則設定 dout 為儲存 din 之 matching data's address 值 & hit = 1'b1

- i. one matching 情況 : dout = 唯一值

- ii. multiple matches 情況 : 眾多 address 之中最「小」



- b. 找不到則設 dout = 4'd0 & hit = 1'b1

3. 當 $(ren, wen) = (1, 1)$ 時 · 做與2相同之行為

4. 當 $(ren, wen) = (0, 0)$ 時 · dout = 4'd0 & hit = 1'b0

Design Explanation

我們的設計分為兩個部分

SEQUENTIAL PARTS

1. 當正緣觸發時 · 更新 dout 和 hit 的值 :

- 將 cam_addr_dout 的值用來更新 dout
- 將 cam_hit 的值用來更新 hit

COMBINATIONAL PARTS

2. 依據不同的狀況來更新 cam_addr_dout / cam_hit 的值 :

- a. $ren = 1$ 時 : 才有機會改變

- 會依序 address(0~15) · 從小(0)到大(15)去檢測 CAM[i] == din 條件是否成立 · 依此抓取最小或唯一的 matching data's address 可能值 i · 作為 cam_addr_dout 輸出 ; 若有找到 · cam_hit = 1

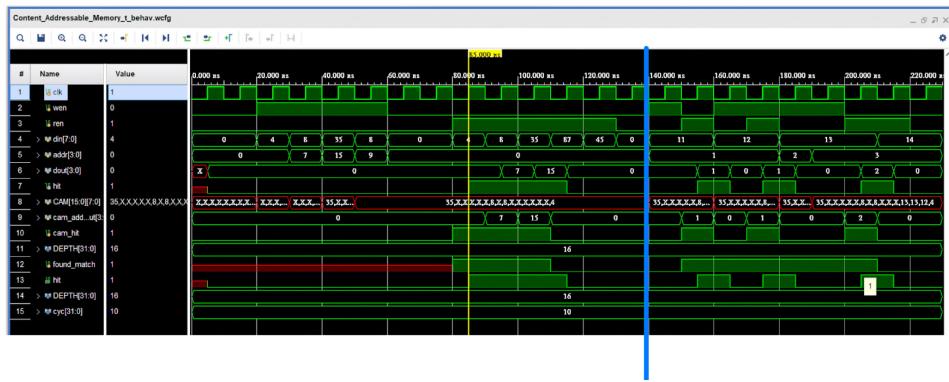
- b. $ren = 0$ 時 : 無論 $wen = 1$ or 0 · 皆設定為 4'b0

3. 更新 CAM

- $ren = 0 \& wen = 1$ 時才有機會改變 · 亦即 (ren, wen) 必為 $(0, 1)$ 之組合
- 此時做寫入行為 → CAM[addr] = din

Testbench Design & Result Explanation

波形圖截圖



TESTBENCH設計解釋

我們分成兩部分來設計 testbench:

1. 藍線前為與 ppt 範例相同之輸入 (wen、ren、addr、din)
→ Write into CAM、Read from CAM、No read and write 三種情況均符合預期結果

2. 藍線後為我們做的各種檢測

----- PART1 : 做 write / 兩種 read 的基本檢測 -----

- a. (ren, wen) = (0, 1) : write to addr1
- b. (ren, wen) = (1, 0) : read one matching data in CAM, and output the matcing data's address , i.e. **addr1**

- c. (ren, wen) = (0, 1) : write to addr2

- d. (ren, wen) = (1, 1) : read + output **addr2**

→ 透過 read 檢測 CAM 有無透過 write 機制成功寫入 / read 機制成功讀出，結果符合預期

----- PART2 : 做 read multiple matching 檢測 -----

- a. (ren, wen) = (0, 1) : write to addr1

- b. (ren, wen) = (0, 1) : write to addr2

- c. (ren, wen) = (1, 0) : read + output **addr1** (為較小的 **addr** 值)

→ 的確有選擇 smallest address 輸出，結果符合預期

----- PART3 : 做 read empty 檢測 -----

- a. (ren, wen) = (1, 0) : read + output "0"

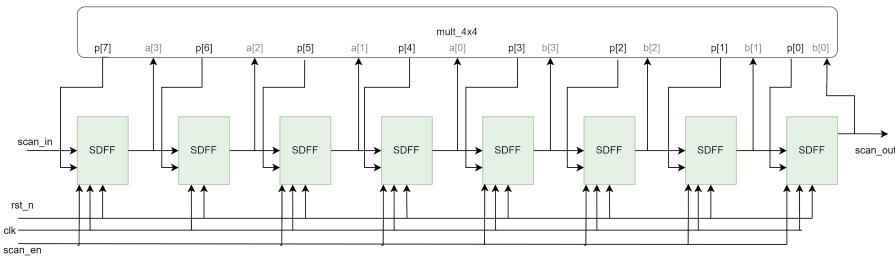
→ 找不到相匹配的 address 時輸出0，符合預期

如此一來，透過基本+額外特殊情況之檢驗，可更加確保我們設計出的 CAM 正確性。

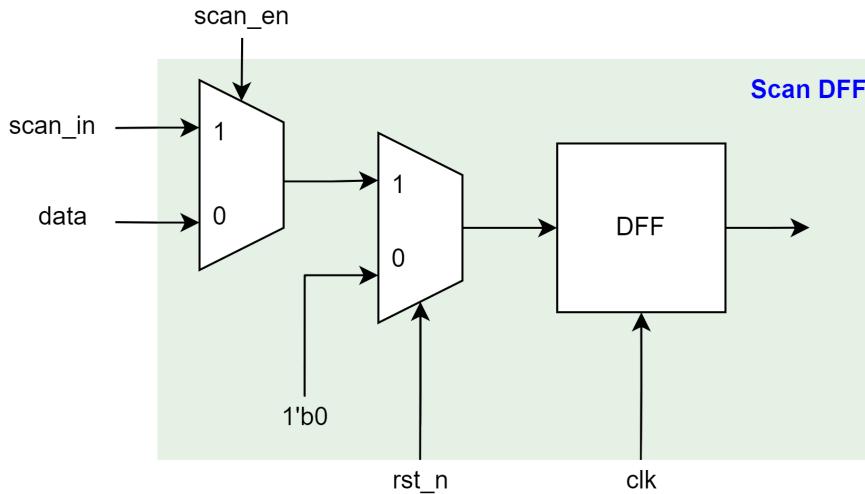
Advanced Question2

Drawing of the design of Verilog Advanced Question 2

- Scan_Chain_Design ⚡ This is top module



- Scan DFF (SDFF)



Requirements

1. $\text{rst_n} = 0$
 - all SDFFs to 1'b0
2. Scan DFF 如同設計圖所示：
 - $\text{scan_en} = 1$: 電路工作在所謂 scan mode , 所以鎖存 scan_in 的值
 - $\text{scan_en} = 0$: 電路工作在正常功能狀態並能把 data 的值鎖存
3. 設計具備 scan chain 的 4-bit multiplier
 - step1. 做 scan replacement , 把電路中的 normal 時序單元 **DFF** 替換為一個 scan 時序單元 **SDFF**
 - step2. 做 scan stitching , 把上一步中得到的 scan DFF 的 Q 和 scan_in 連接在一起形成 scan chain 。而通過 scan chain 的連續動作 , 就可以把問題從對複雜時序電路的測試轉化成測試組合電路 。

Design Explanation

1. 使用模組化方式設計以下模組：

- Scan_Chain_Design : 設計用來測試的一個技巧，是本次作業的主軸，會以 4-bit 乘法器作為目標物進行檢測範例。
- SDFF : 用於組成 scan chain 的 **scan DFF**
- mult_4x4 : 用於 Scan_Chain_Design 的測試目標模組
4-bit multiplier
- Full_adder : 用於 mult_4x4 中的加法運算需求

[簡介各個模組]

2. Scan_Chain_Design :

使用 8 個 SDFF 組成 scan chain，scan_in 輸入名稱 s7 的 SDFF 之後，一路傳過 7 個 SDFF (名稱 s6、s5、s4、s3、s2、s1、s0)，最終輸出 b[0] 作為 scan_out。其中各個 SDFF 的 **data** 輸入值分別是 4-bit Multiplier 輸出的 p[7]、p[6]、p[5]、p[4]、p[3]、p[2]、p[1]、p[0] (由圖左到右)；**scan_in** 則是上一個 (左邊) SDFF 的輸出值；**scan_en** 則是此模組的 input，用於決定此 clk 要輸出的是 data 還是 scan_in。

3. SDFF :

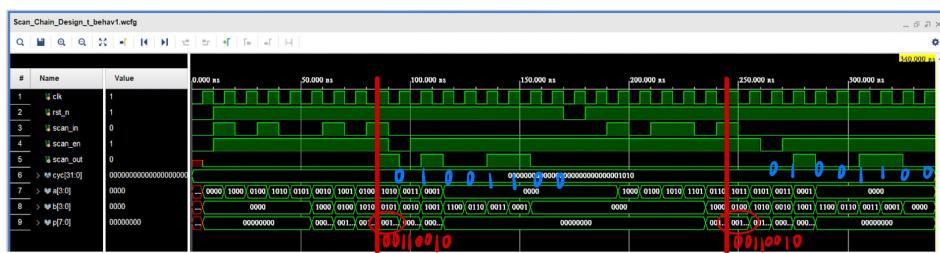
使用兩個 if 實現選取 dout 的目的，第一層以 rst_n 作為 Mux 的塞選，=0 則輸出 0，否則進入第二層，再來以 scan_en 作為塞選，=1 則選取 scan_in，=0 則選取 data，此外所有 defalt 皆為輸出 0。

4. Full_adder、mult_4x4

兩者皆延續使用先前作業之成果。

Testbench Design & Result Explanation

波形圖截圖



TESTBENCH設計解釋

我們依照 Three phases : Scan In、Capture、Scan Out 來改動 **scan_en**，並輸入一輪 8 clock cycle 的兩組 **scan_in** 來看結果是否符合往後生成相對應的 **scan_out** 結果。

[情況一]

Phase1 : Scan In

一開始先將 **scan_en** 設定為 1，**scan_in** 依序輸入 1、0、1、0、0、1、0、1，讀入 scan chain bit-by-bit 之後， $a = 4'b1010$ 、 $b = 4'b0101$ 。

Phase2 : Capture

將 **scan_en** 設定為 0 維持一個 cycle，此時的 circuit 會去做邏輯運算，依照現在存於 SDFFs 的 **a** & **b** 值算得 **p**，並存於 SDFFs 中。

(心中已有理想結果 $p = 00110010 (=1010 \times 0101)$ ，可於下階段對答案)

Phase3 : Scan Out

再將 **scan_en** 設定為 1，存於 SDFFs 中結果 **p** 便會藉由 **scan_out** bit-by-bit 的呈現出來。在此情況，**scan_out** 依序印出 0、1、0、0、1、1、0、0。

→ 經過三個階段之後，發現 **scan_out** 輸出結果符合心中預期，檢驗正確！

[情況二] (依照情況一的步驟再檢查另一個狀況)

Phase1 : Scan In

scan_in 依序輸入 0、1、0、1、1、0、1、0， $\Rightarrow a = 4'0101$ 、 $b = 4'b1010$ 。

Phase2 : Capture

將 **scan_en** 設定為 0 維持一個 cycle，現存於 SDFFs 的 **a** & **b** 值算得 **p** 並存於 SDFFs。

(心中已有理想結果 $p = 00110010 (=0101 \times 1010)$ ，可於下階段對答案)

Phase3 : Scan Out

再將 **scan_en** 設定為 1，結果 **p** 會 bit-by-bit 的由 **scan_out** 呈現出來。

在情況二，**scan_out** 依序印出 0、1、0、0、1、1、0、0。

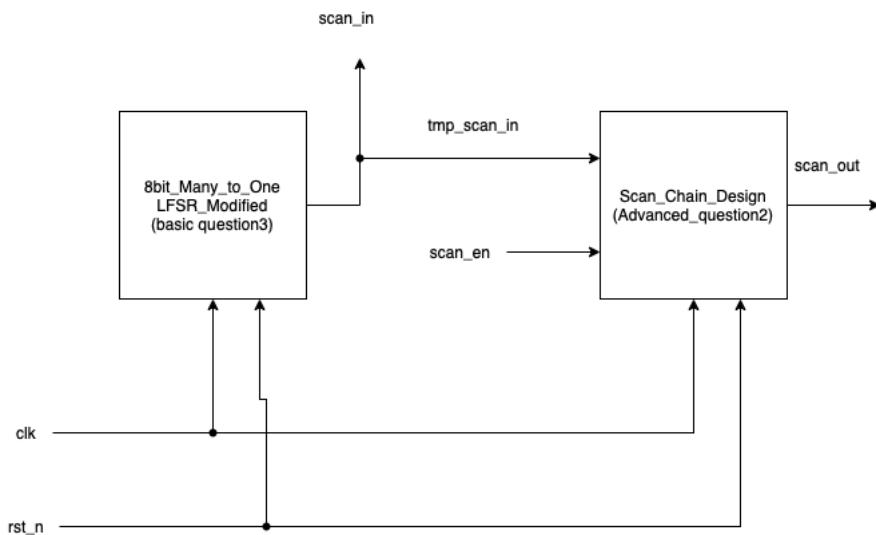
→ 經過三個階段之後，發現 **scan_out** 輸出結果符合心中預期，檢驗正確！

2 組乘法之檢驗皆為正確的結果。

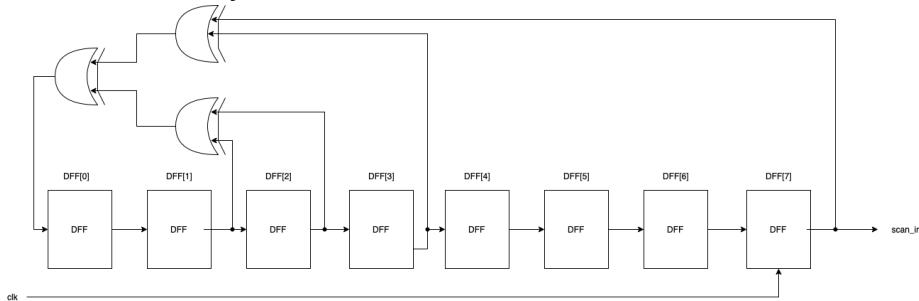
Advanced Question3

Drawing of the design of Verilog Advanced Question 3

- Top module



- Modified Many_to_One LFSR



Requirements

- reuse the scan chain module implemented in Advanced question 2
- modify the Many_to_One LFSR from the basic question 3 so that only the MSB of the LFSR is shifted into the scan chain
- test pattern is generated by LFSR
- send the test pattern to Scan Chain

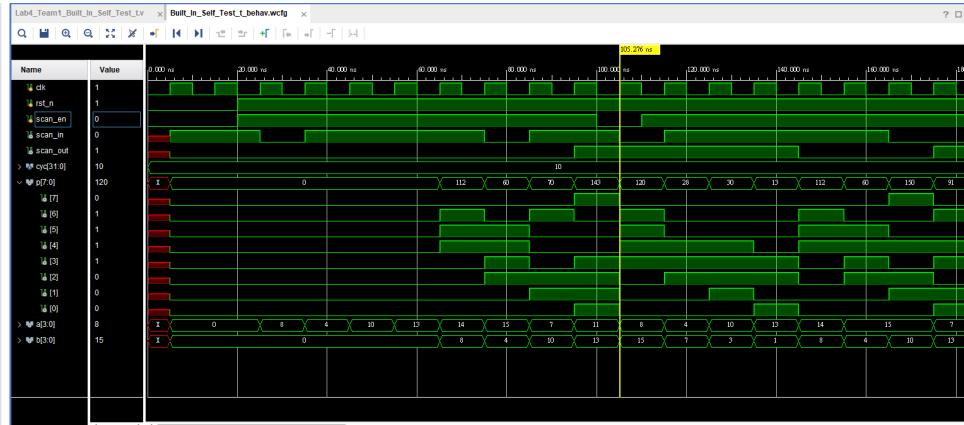
Design Explanation

1. 修改在basic question3所實作的LFSR來產生test_pattern
 - 原本的output是8bit，但是在這裡我們只需要Most Significant Bit，也就是DFF[7]

2. 把test pattern接到advanced question2所實作的module
並產生相對應的結果

Testbench Design & Result Explanation

波形圖截圖



我們可以從上圖中看到剛reset過後的scan_in(last bit of dff)
會是以1->0->1->1->1->0->1的順序產生，而其代表的是
是 $b_0 \rightarrow b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3$ ，所以
 $b = 4'b1101(13)$, $a = 4'b1011(11)$, $p = a \times b = 143 =$
 $8'b10001111$ 而後續的scan_out是依照 $p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4 \rightarrow p_5 \rightarrow p_6 \rightarrow p_7$ 來顯示，而依據波形圖，我們的設計顯示
1->1->1->1->0->0->0->1，正好符合我們的期望。

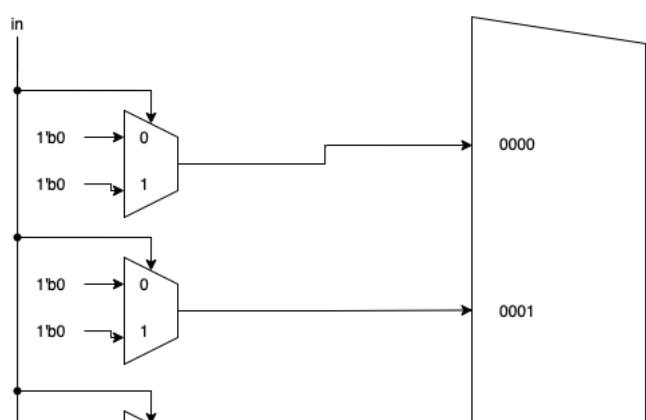
TESTBENCH設計解釋

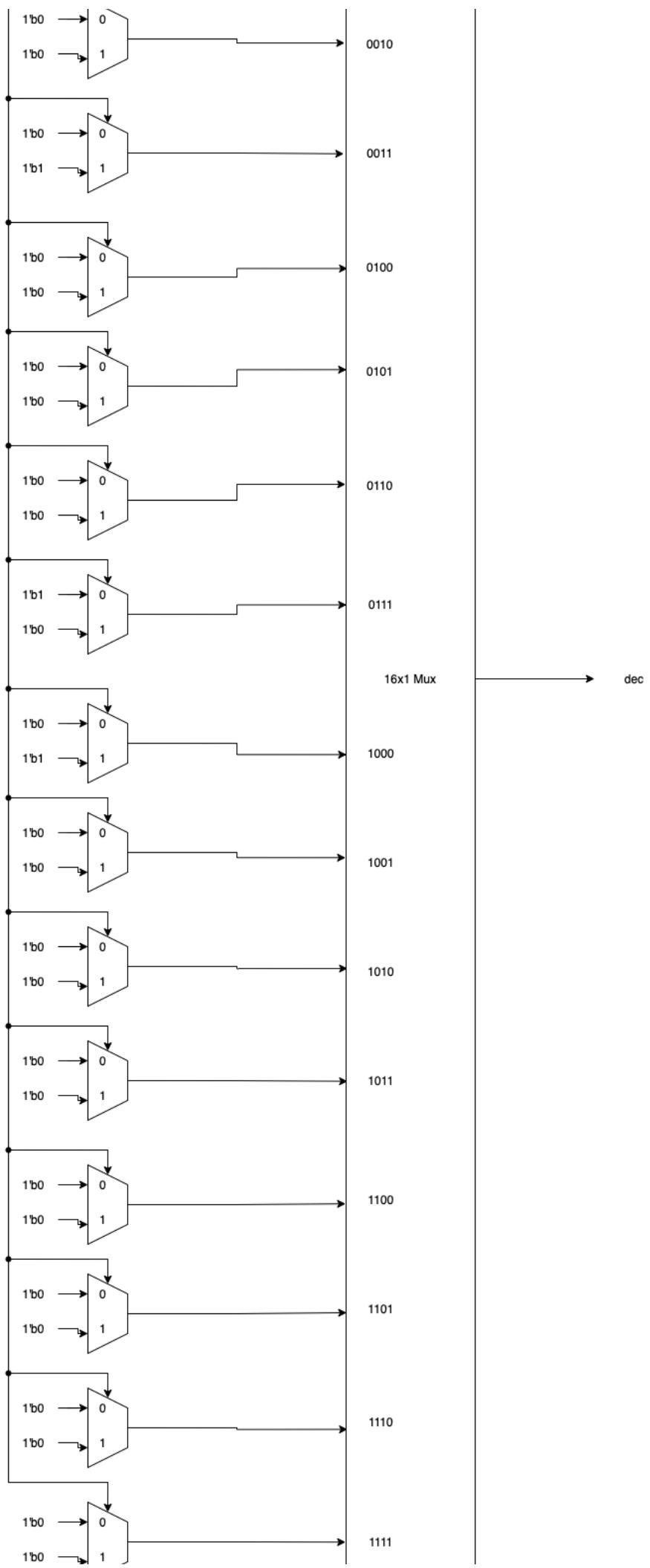
由於scan_in是由8bit Many to one LFSR來隨機產生的，所以我們依照第二題所提供的three-phase behavior pattern來改動scan_en，並重複這樣的設計來確保不同的scan_in有產生相對應的結果。

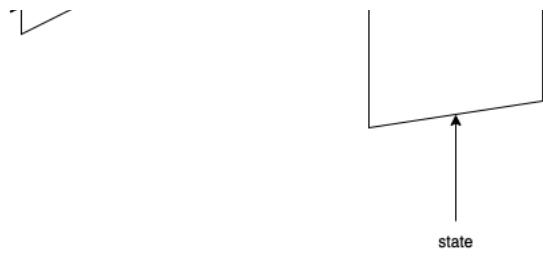
Advanced Question4

Drawing of the design of Verilog Advanced Question 4

- dec

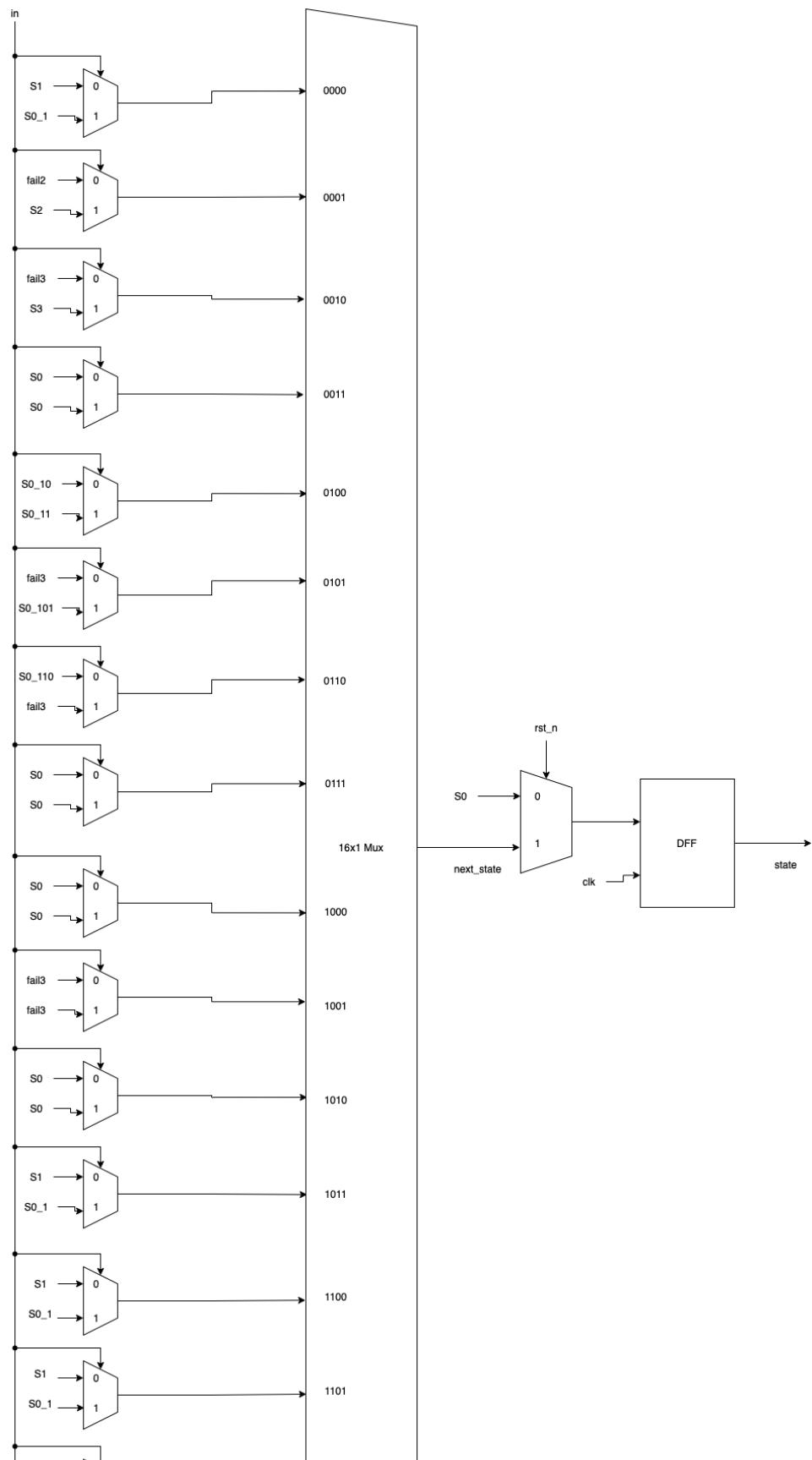


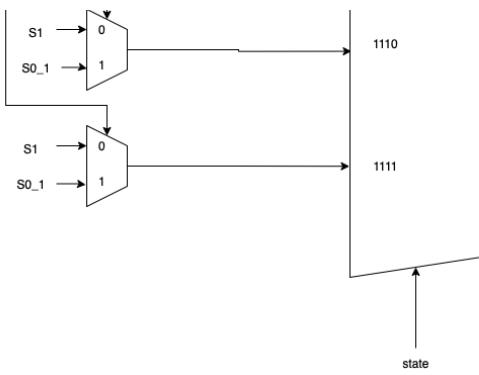




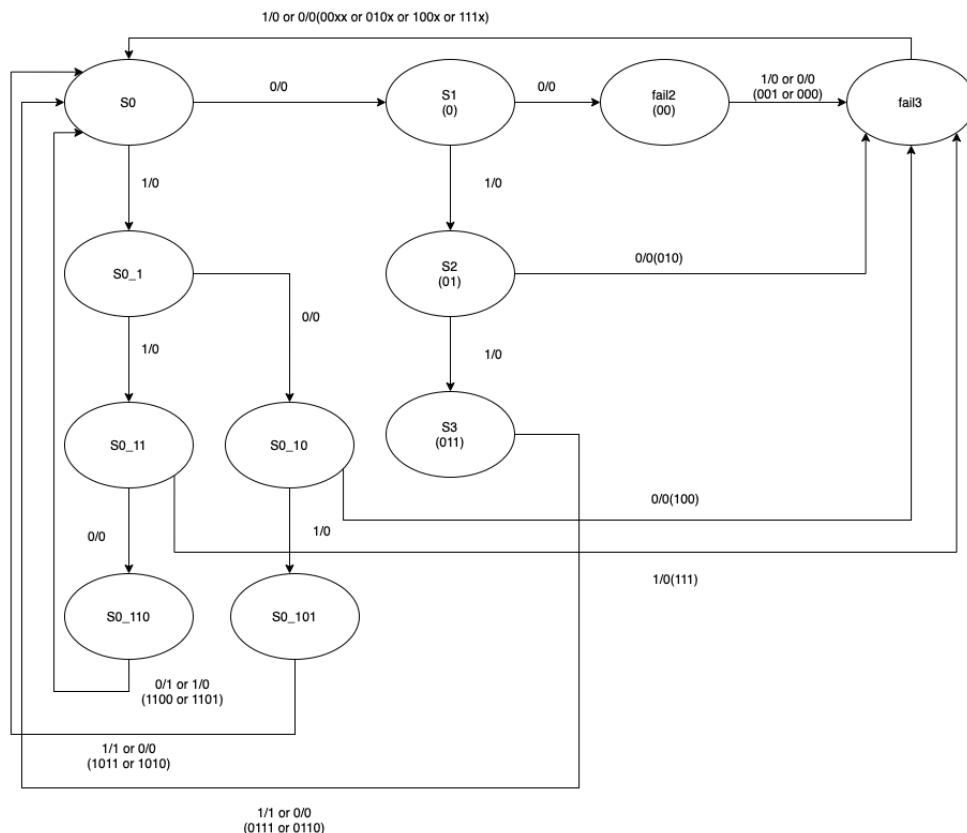
- state

-





State Diagram of Verilog Advanced Question 4



Requirements

1. 設計一個Mealy machine
2. 該mealy machine會依據前面所收到的四個bit來決定 output
3. 當偵測到0111, 1011, 1100時，dec設為1

Design Explanation

在我們的設計中，除了原本的S0~S3，我們依據情況多加入了數個state如下

- S0_1 : state S0收到1後的state
- S0_10 : state S0_1收到0後的state

- S0_11 : state S0_1收到1後的state
- S0_110 : state S0_11收到0後的state
- S0_101 : state S0_10收到1後的state
- fail2 : 在收到的前兩個bit為00時代表這一組sequence不會有機會讓 $dec = 1$ ，所以設置一個state名為fail2
- fail3 : 在收到前三個bit非011, 101, 110時，代表這一組sequence不會有機會讓 $dec = 1$ ，所以設置一個state名為fail3

由上面所設置的states可以得知，我們只有在該sequence有機會使得 $dec = 1$ 時才會多幫他們設置state，否則state的數量會過多且沒有這個必要性。

而在我們的設計中，我們在sequential circuit的部分更新state的值，並在combinational circuit的部分決定下一個要去的state還有 dec 的值。

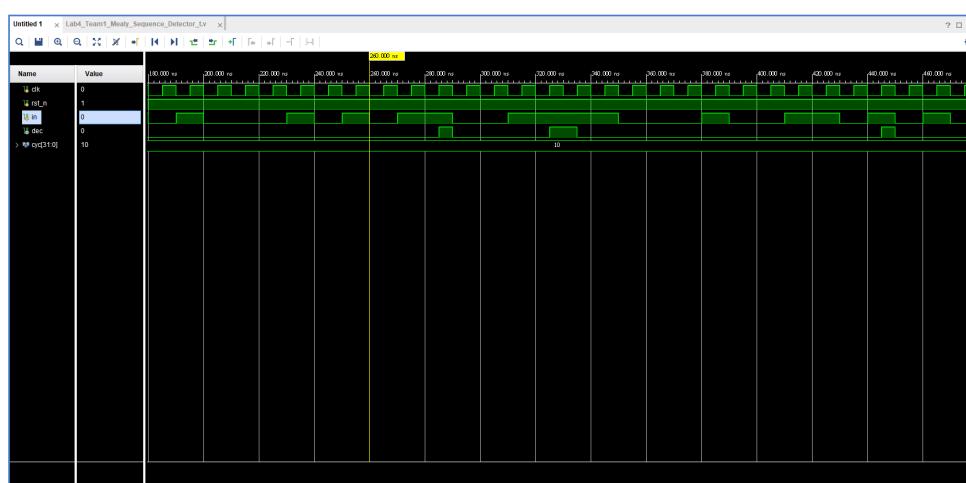
而只有在以下情況會使得 $dec=1$:

- state = S3 , in = 1 (0111)
- state = S0_110, in = 0 (1100)
- state = S0_101, in = 1 (1011)

最後在決定next_state的部分，則依據上述情況去接到相對應的state，然後比較特別的一點是S0_110, S0_101, S3, fail3無論input為多少，都會接回到S0，因為每經過四個bit要re-detect一次。

Testbench Design & Result Explanation

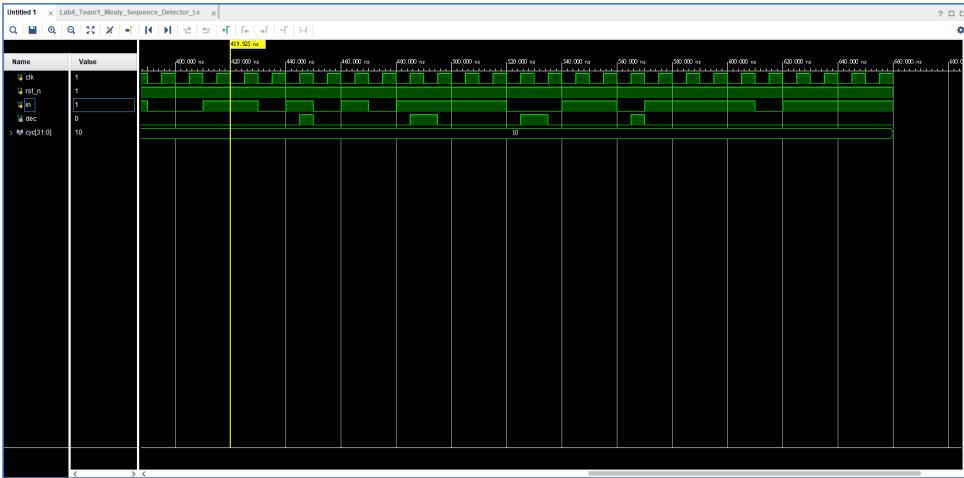
波形圖截圖



在上圖的部分，我們可以看到以下結果：

- 前四個bit為0110時會升起半個cycle

- 前四個bit為0111時會升起一個cycle



在上圖的部分，我們可以看到以下結果：

- 前四個bit為1010時會升起半個cycle
- 前四個bit為1011時會升起一個cycle
- 前四個bit為1101時會升起半個cycle
- 前四個bit為1100時會升起一個cycle

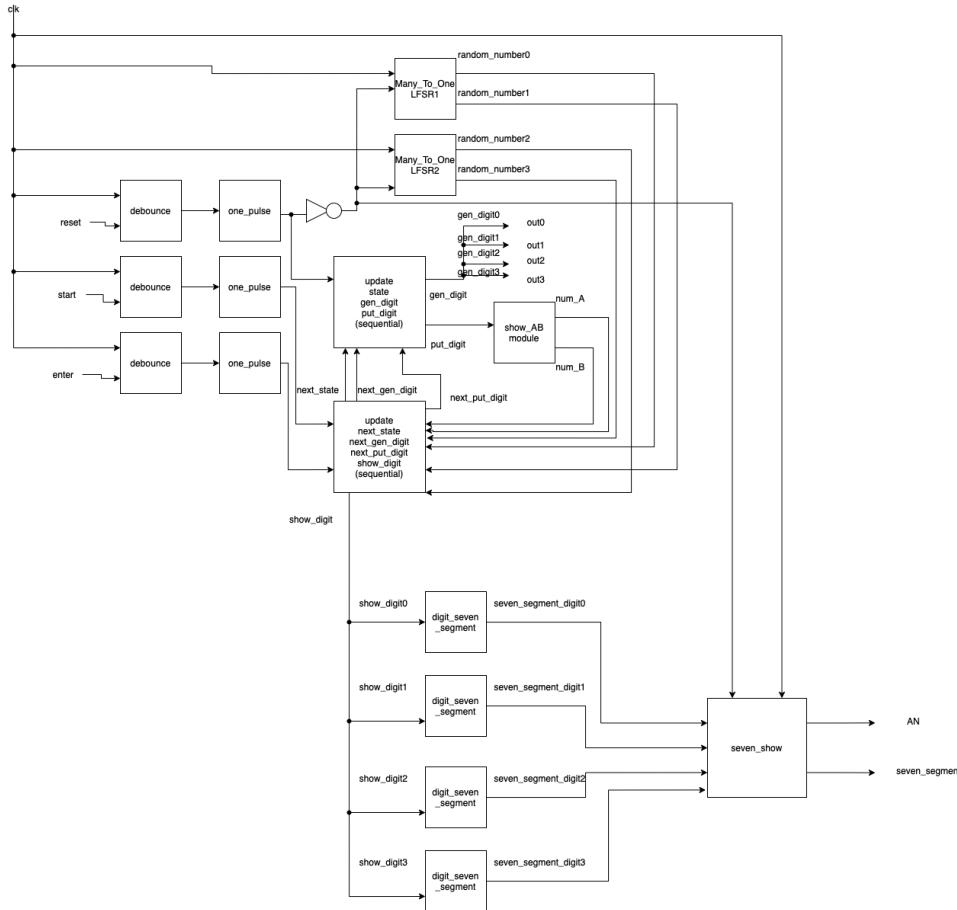
這是因為dec並不是positive edge trigger，所以在我們的tb設計中，訊號在negative edge trigger時才會改變，所以dec會升起半個cycle。如：在0110的case中，state收到前面011的sequence之後會跑到S3，而如果state是S3且in = 1時，dec則會升起，而在negative edge trigger的時候in已經改變，且不是我們所想要的值，此時dec就降回0了。反之，如果最後一個in是1的話，dec則會維持升起，直到下一個positive edge trigger為止（此時state = S0，而不管input為多少，dec都是0）

TESTBENCH設計解釋

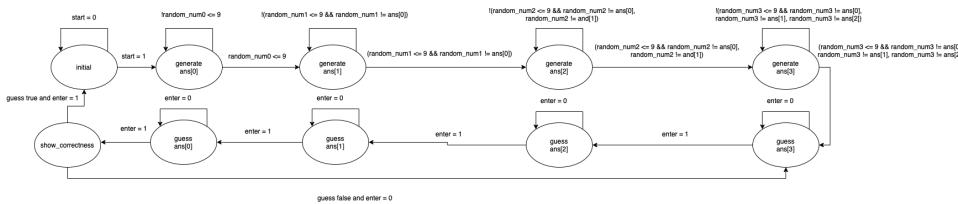
我們的設計方式有點類似窮舉，我們將四個cycle為一組，每一組輸入的in值暫且稱為in1, in2, in3, in4，而在我們的設計中，我們跑過(in_1, in_2, in_3, in_4) = (0, 0, 0, 0) ~ (1, 1, 1, 1) 來確認所有的可能。

FPGA Demonstration 1

Drawing of the design of FPGA Demonstration



State diagram of the design of FPGA Demonstration1



Design Explanation

1. FPGA_1A2B

- 設定 9 種 state
 - init = 4'b0000
 - 按下 reset 時，將所有 digit 皆設定為 0
 - display 顯示 “1A2b”，等待 start button 被按下
 - 按下 start 時，LED 依序顯示相對應的值
 - gen_ans0 = 4'b0001
 - 將符合標準的 random_number0 輸入成為第 0bit 的 ans

- gen_ans1 = 4'b0010
 - 將符合標準的 random_number1 輸入成為第 1bit 的 ans
- gen_ans2 = 4'b0011
 - 將符合標準的 random_number2 輸入成為第 2bit 的 ans
- gen_ans3 = 4'b0100
 - 將符合標準的 random_number3 輸入成為第 3bit 的 ans
- guess_ans3 = 4'b0101
 - 猜第三個 bit 的答案的狀態
- guess_ans2 = 4'b0110
 - 猜第二個 bit 的答案的狀態
- guess_ans1 = 4'b0111
 - 猜第一個 bit 的答案的狀態
- guess_ans0 = 4'b1000
 - 猜第零個 bit 的答案的狀態
- show_correctness = 4'b1001
 - show xAxb 的狀態

○ 參數設定

- gen_digit0~4 : 正確答案的 4-bit 數字 (以 next_gen_digit0~4 更新)
- put_digit0~4 : 猜測中的 4-bit 數字 (以 next_put_digit0~4 更新)
- show_digit0~4 : seven-segment 要顯示的數字
- out1、out2、out3、out4 : LED 要顯示的 4*(4-bit) 數字

○ 如何產生不重複的四位數?

使用兩個 LFSR 生成兩組 8-bit 數字，以 out[7:4]%10 & out[3:0]%10 作為兩個輸出，並用暴力法檢測是否有重複的，如: 生成 0、1 digit 時，檢測兩者是否相等；而生成 2、3 時，除了考慮兩者是否相等之外，還要與已經生成的 0、1 digit 做比對，不相等時才能取其值。

- 對於 button : reset、start、enter 都做 debounce 及 one_pulse 的動作，並依序存值於 debounced_reset、debounced_start、debounced_enter => one_pulsed_reset、one_pulsed_start、one_pulsed_enter (我們先做 debounce，再做one_pulse)
- 設定一種 clock用於 display，名為clk_display
- 使用四個 digit_seven_segment，將數字 (digit) 轉換成要顯示的格式 (seven_segment)
- 呼叫 show_AB module 顯示出幾 A 幾 B

2. show_AB

計算出幾 A 幾 B，寫下 num_A, num_B 判斷條件，若位置相符，則 A+1；若數字相符&位置不符，則 B+1。

3. seven_show

顯示相對應的答案數字在 LED 上，需要注意的一點是在猜數字的時候，最右邊的數字要呈現一閃一閃的，所以我們有設置一個 8bit counter 來完成此任務，

- counter < 8'b10000000 的時候：seven segment亮出數字
- counter >= 8'b10000000 的時候：不亮數字

如此一來，使得一個 cycle 裏面有一半時間亮數字，另一半時間數字是暗的，而在這裡有個特別的地方是我們把 counter 設置為 8-bit，而這樣是因為 display control 使 用輸出頻率為 $1/2^{17}$ 的 clk，所以如果使用 17bit 的 counter 的話會太慢，所以我們最後選擇取用 8-bit counter 作為使用。

4. digit_seven_segment

每個數字對應到 seven_segment 值，如下：

digit	seven_segment
0(4'b0000)	7'b0000001
1(4'b0001)	7'b1001111
2(4'b0010)	7'b0010010
3(4'b0011)	7'b0000110
4(4'b0100)	7'b1001100
5(4'b0101)	7'b0100100
6(4'b0110)	7'b0100000
7(4'b0111)	7'b0001111
8(4'b1000)	7'b0000000
9(4'b1001)	7'b0001100
10(4'b1010)	7'b0001000
11(4'b1011)	7'b1100000
default	7'b0000001

5. Many_To_One_LFSR1、Many_To_One_LFSR2 Ifsr2

使用到 basic question 3所撰寫的 LFSR 以製造 random number。

- 選擇reset值為 8'b10111101 的 Many_to_one LFSR
 - out[3:0] 用於random_number0的生成
 - out[7:4] 用於random_number1的生成
- 選擇reset值為 8'b01100111 的 Many_to_one LFSR
 - out[3:0] 用於random_number2的生成
 - out[7:4] 用於random_number3的生成

6. one_pulse

因此次 FPGA 也使用到 push buttons 的方式，而當 button 被按一次必須要延續一個 clock-cycle，因此將 lab3 FPGA 題寫的 one_pulse 應用於此，用以確保按下 button 的情況要維持一個 clock。

7. debounce

因此次 FPGA 也使用到 push buttons 的方式，因此將 lab3 FPGA 題寫的 Debounce Circuit 機制應用於此，透過

宣告一連串的 DFF，將 register 層層傳遞來處理訊號的 meta-stability 問題，只有在全是 1 才是 1、全是 0 才是 0，如此一來可以處理掉不乾淨的訊號。

8. clk_for_display

設計一個輸出頻率為 $1/2^{17}$ clk 的 clk_count，用於 display_control 的近似同步顯示，是給 Display 的 clock。

I/O pin assignment

A、Clock (clk)

Input	package pin
clk	W5

B、Switches

SW[15:12]

Input	package pin
input_num[0]	W2
input_num[1]	U1
input_num[2]	T1
input_num[3]	R2

C、Buttons

Input	package pin
enter	T17
reset	T18
start	U18

D、LED

Input	package pin
out0[0]	U16
out0[1]	E19
out0[2]	U19
out0[3]	V19

Input	package pin
out1[0]	W18
out1[1]	U15
out1[2]	U14
out1[3]	V14

Input	package pin
out2[0]	V13
out2[1]	V3
out2[2]	W3
out2[3]	U3

Input	package pin
out3[0]	P3
out3[1]	N3
out3[2]	P1
out3[3]	L1

E ` 7-segment display

Output	package pin
AN[0]	U2
AN[1]	U4
AN[2]	V4
AN[3]	W4
segment[0]	U7
segment[1]	V5
segment[2]	U5
segment[3]	V8
segment[4]	U8
segment[5]	W6
segment[6]	W7

Contribution

朱季葳：advanced question 3、4; FPGA

施泳瑜：advanced question 1、2; report of FPGA

What we have learned from Lab4

1. 第一題本來用 `for loop` 實踐，後來在寫報告時和隊友互相討論之下覺得這樣的 `coding style` 不好，圖也難畫，因而改了 `if-else` 的方式，圖就好畫很多，深刻感受到軟體、硬體語言的不同，在往後的設計上也會特別注意。
2. 在 advanced question 2 實作時，常因 `clock cycle` 不夠熟悉的關係，而在看 `simulation` 結果的時候對不起來讓腦袋一次又一次卡住，但在慢慢對、慢慢看、慢慢改之後很有感地對時序電路的觀念又再度補強，做完之後挺有收穫的，因為以上而認為 `scan chain` 在這個單元是一項很有趣的實踐。
3. FPGA 題的部分，很多小細節沒注意到就會卡超久！在這次 1A2B game 中也常因為時序問題 `digit` 常顯示在不對的位置，在不斷修正之後，對於能成功建立出一個小遊戲挺有成就感的。