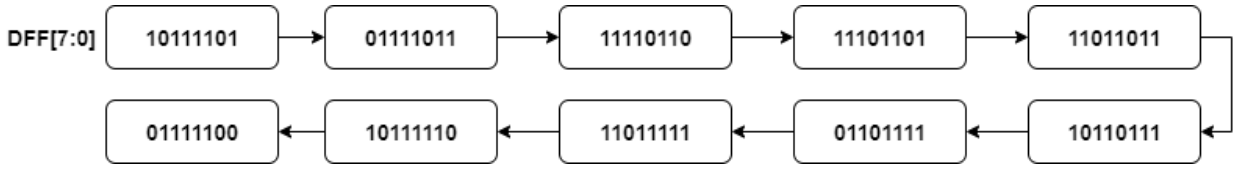
Lab4_Team32_Report

組別：Team32

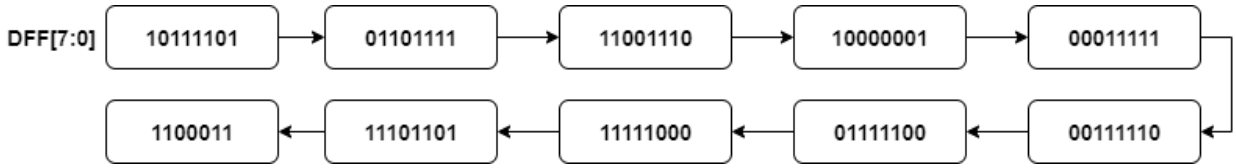
組長：蘇勇誠（108062373）

組員：張晏瑄（108062273）

State Transition Diagram of DFFs in many-to-one LFSR



State Transition Diagram of DFFs in one-to-many LFSR



Content-addressable memory (CAM)

Design

依題目敘述，當要write時，din送進 CAM 後，會先儲存在 data line 上。當要read時，會先經由一個比較器比對 data line 上的資料是否和 din 相同，再經由 Priority Encoder 將 din 所對應的最高位址傳出來，而若有找到對應的值，則 hit 會變為 1 ，反之為 0。

Detail

Sequential Circuit:

判斷當 `wen`, `ren` 為何時 `dout` 和 `hit` 的情況。

1. `wen == 1'b0 && ren == 1'b0` ：CAM 不做事， `dout` 和 `hit` 皆為 0
2. `wen == 1'b0 && ren == 1'b0` ：將 `din` 存入 `cam_data` 內，其對應的 index 為 `addr`
3. `ren == 1'b1` ：當 `ren` 為 1 時，不論 `wen` 的值為何，都優先讀取，接著判斷 `din` 是否有和寫入 CAM 的 data 配對，若有配對到，則 `dout` 設為該對應的 `addr` 以及 `hit` 設為 1，反之則 `dout` 為 0。

Combinational Circuit:

- Comparator Array

  `reg [15:0] pri` 為比對的 array，當 `ren` 為 1 時，逐一比對在 CAM 的 data 是否有和 `din` 相同，若有相同，則對應的 `pri[index]` 會被設為 1，而其 index 即為 `din` 所寫入的 data address。比對完後，將 `pri` 送入 Priority Encoder 去判斷最高位數為 1 的數為何。

```
if(ren == 1'b1) begin
    pri[0] = (cam_data[0] === din)? 1'b1: 1'b0;
        .
        .
        .
    pri[15] = (cam_data[15] === din)? 1'b1: 1'b0;
end
```

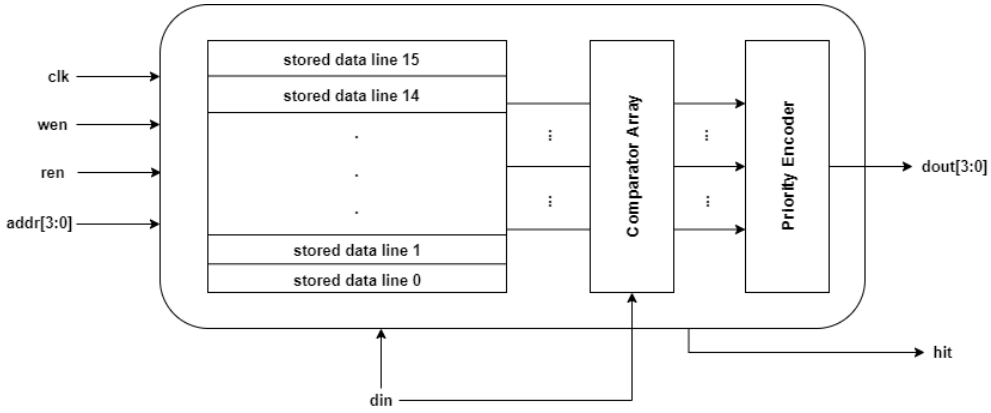
- Priority Encoder:

  設計 1 個 8 to 3 的 priority encoder，只有在 `pri[8]` 後才會用到最高位數，故判斷 `dout` 的最高位數是否為 1 時，只需將 `pri[8]`~`pri[15]` OR起來 。而判斷是否 CAM 的 data 內有存放要求 din，只需將所有 `pri` 的 bit OR 起來。

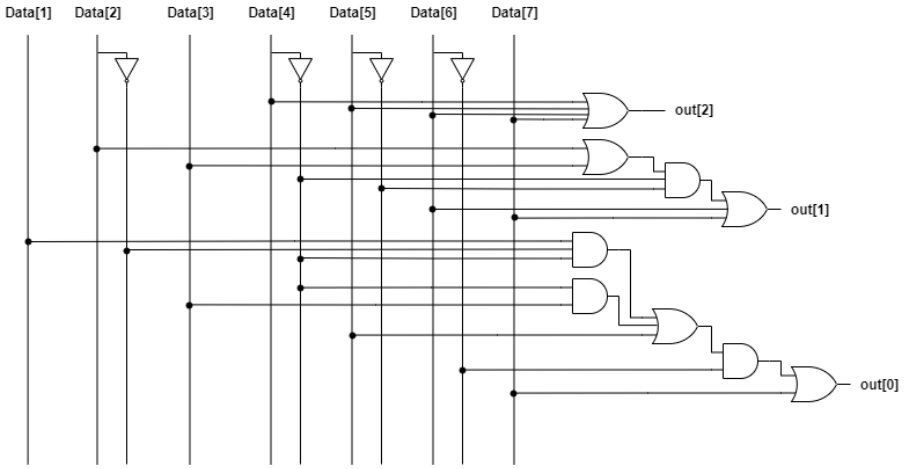
```
priority_encoder pe0(pri[7:0], en0);
priority_encoder pe1(pri[15:8], en1);

assign sel = pri[8] | pri[9] | pri[10] | pri[11] | pri[12] | pri[13] | pri[14] | pri[15];
assign match = |pri;

//在combinational always block內
if(sel == 1'b1) begin
    pri_addr = {1'b1, en1};
end
else begin
    pri_addr = {1'b0, en0};
end

module priority_encoder(data, out);
input [7:0] data;
output wire [2:0] out;

assign out[0] = (((~data[6]) & ((~data[4]) & (~data[2]) & data[1]) | ((~data[4]) & data[3]) | data[5])
assign out[1] = ((~data[5]) & (~data[4]) & (data[2] | data[3]) | data[6] | data[7]);
assign out[2] = (data[4] | data[5] | data[6] | data[7]);
endmodule
```

假設 `din` 為 8 時，在 CAM 內對應的 `addr` 有 7 和 9，則 `pri` 則會在第7、9個 bit 為 1，其餘為 0，而透過 Priority Encoder，我們即能得出最高位數為 1 的數是 9，並將 `dout` 設為 9。

Logic Diagram



Content Addressable Memory



8-to-3 Priority Encoder

Testbench

設計 testbench 時，讀寫個分為 3 種情況：

Write:

1. 每個 `addr` 皆存放不同的 data

2. 每個 `addr` 皆存放相同的 data
3. 部分 `addr` 存放相同的 data，部分存放不同的

Read:

1. 讀取時所要求的 `din` 全部都沒有在 data line 裡
2. 讀取時所要求的 `din` 全部都有在 data line 裡
3. 讀取時所要求的 `din` 部分有在 data line 裡

同時並測試 `wen` 和 `ren` 的各種組合是否有做正確的動作

```
// each of the addr store the different din
for(i = 0; i < 16; i = i + 1) begin
    @(negedge clk) begin
    addr = i;
    din = i * 2;
    ren = 1'b0;
    wen = 1'b1;
    end
end
// cannot find the match din in cam_data[addr]
for(i = 0; i < 16; i = i + 1) begin
    @(negedge clk) begin
    addr = 4'd0;
    din = i + 100;
    ren = 1'b1;
    wen = 1'b0;
    end
end
```

Scan chain design

Design

如題所述，scan chain 主要有三種行為：

1. Scan in

   `scan_en` 為 `1'b1`，並會傳入 `scan_in` 至 SDFF 中。

2. Capture

   `scan_en` 為 `1'b0`，並將存在 SDFF 的值送入 multiplier 內進行運算完後，再將算出的值存回 SDFF 裡 → 需設計一 multiplier 進行運算。

3. Scan out

   `scan_en` 為 `1'b1`，並將 `SDFF[7]` 的值作為 `scan_out` output 出來。

Detail

Sequential Circuit

負責處理 SDFF 值的傳送，當 `rst_n` 為 `1'b0` 時，reset 所有的 SDFF 值。而當 `rst_n == 1'b1`，依照 scan_en 的值去做處理，當 `scan_en == 1'b1` 時，做 Scan in 和 Scan out 的動作，若 `scan_en == 1'b0` 時，做 Capture 的行為，並將值送回 SDFF 裡。

```
always @(posedge clk) begin
    if(rst_n == 1'b0) begin
        DFF <= 8'b0;
    end
    else begin
        if (scan_en == 1'b1) begin
            DFF[7:1] <= DFF[6:0];
            DFF[0] <= scan_in;
        end
        else begin
            DFF <= {p[0], p[1], p[2], p[3], p[4], p[5], p[6], p[7]};//data
        end
    end
end
```

Combinational Circuit

利用 4-bit multiplier 去計算 p 的值，`p = a[3:0] * b[3:0]`，p 即為做 Capture 動作後在 multiplier 裡所 output 的值。

先將要做運算的 a, b 設好，然後做完 capture 動作後的 p，會在 `scan_en == 1'b0` 時被送回各個 SDFF：

```
Multiplier_4bit Mul(a, b, p);

always @(*) begin
    b = {DFF[4], DFF[5], DFF[6], DFF[7]};
    a = {DFF[0], DFF[1], DFF[2], DFF[3]};
end
```

Scan_out

```
//combinational for output
assign scan_out = DFF[7];
```

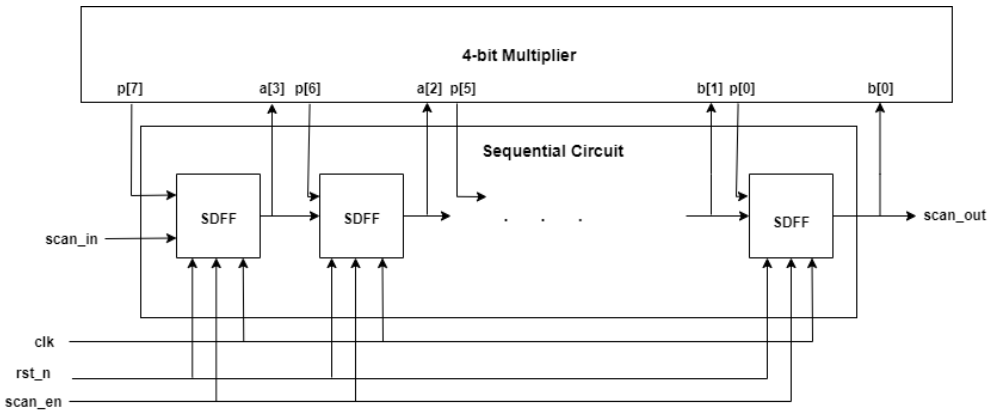
Multiplier

在 Lab2 即有做過 Gate Level 的 multiplier，將類似概念稍加修改即可

```
module Multiplier_4bit(a, b, p);
input [3:0] a, b;
output [7:0] p;
```

```
wire [3:0] w0, w1, w2, w3, w4, w5;

assign w0 = {1'b0, a[3] & b[0], a[2] & b[0], a[1] & b[0]};
assign w1 = {a[3] & b[1], a[2] & b[1], a[1] & b[1], a[0] & b[1]};
assign w2 = {a[3] & b[2], a[2] & b[2], a[1] & b[2], a[0] & b[2]};
assign w3 = {a[3] & b[3], a[2] & b[3], a[1] & b[3], a[0] & b[3]};

assign p[0] = a[0] & b[0];

adder adder0(w0, w1, 1'b0, {w4[2:0], p[1]}, w4[3]);
adder adder1(w2, w4, 1'b0, {w5[2:0], p[2]}, w5[3]);
adder adder2(w3, w5, 1'b0, p[6:3], p[7]);

endmodule
```

Logic Diagram



Testbench

由於a[3:0]與b[3:0]總共有8個bits，故{a[3:0], b[3:0]}一共有256總可能，分別將256總可能送入SDFF，並測試該256種可能之p[7:0]是否會等於a[3:0] * b[3:0] 。

```
repeat(2 ** 8) begin

        @(negedge clk) begin
        data = {a, b};
        rst_n = 1'b1;
        scan_en = 1'b1;
        {a, b} = {a, b} + 8'd1;
        scan_in = b[0];
        data = {a, b};
        end
        for(i = 1; i < 8; i = i + 1) begin
            @(negedge clk)
              scan_in = data[i];
        end

        @(negedge clk) begin
          rst_n = 1'b1;
```

```
          scan_en = 1'b0;
        end

        for(i = 0; i < 8; i = i + 1) begin
            @(negedge clk) begin
              rst_n = 1'b1;
              scan_en = 1'b1;
              p[i] = scan_out;
            end
        end
        if((a * b) !== p)
            $display("error occurs at a=%d b=%d a*b=%d p=%d", a, b, a*b, p);
    end
```

Built-in self test

Design

如題目所述，將 Basic Q3 的 many-to-one LFSR 作為亂數產生器產生亂數後，將產生的值作為 `scan_in` 送進 Advanced Q2 的 8-bit Scan Chain，Scan_chain 在 `scan_en == 1'b1` 時會將 SDFF[7] 的值作為 scan_out 送出。

Detail

8-bit Many-to-One LFSR

Sequential Circuit

依 Basic Q3 要求，當 `rst_n == 1'b0` 時，reset `DFF[7:0]` to `8'b10111101`，當 `rst_n == 1'b1` 時，`out` 會接受下個 `next_out`

```
always @(posedge clk) begin
    if(rst_n == 1'b0)
        out <= 8'b10111101;
    else begin
        out <= next_out;
    end
end
```

Combinational Circuit

依照 Q3 的圖刻。

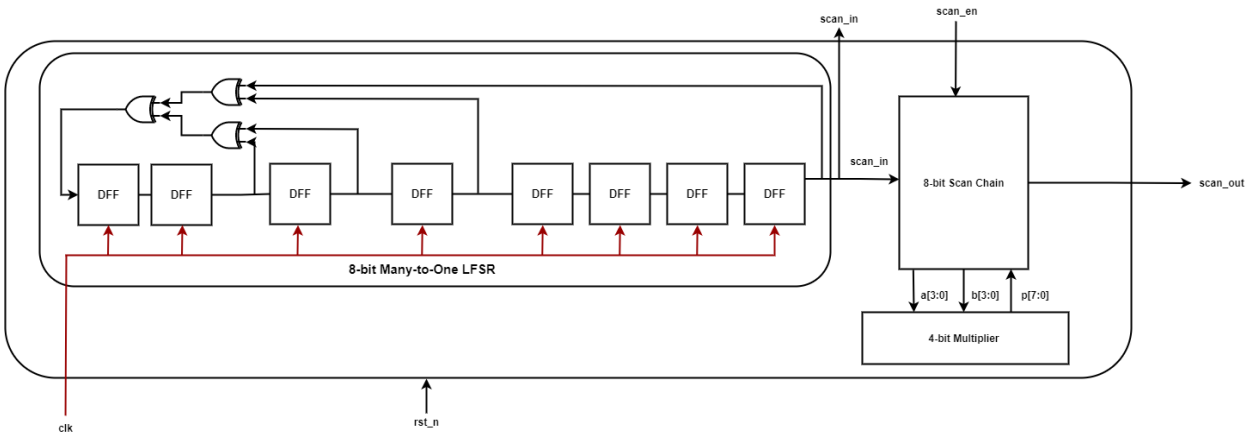
```
assign in_DFF0 = out[7] ^ out[3] ^ out[2] ^ out[1];
assign data = out[7];

always @(*)begin
```

```
    next_out = {out[6:0], in_DFF0};
end
```

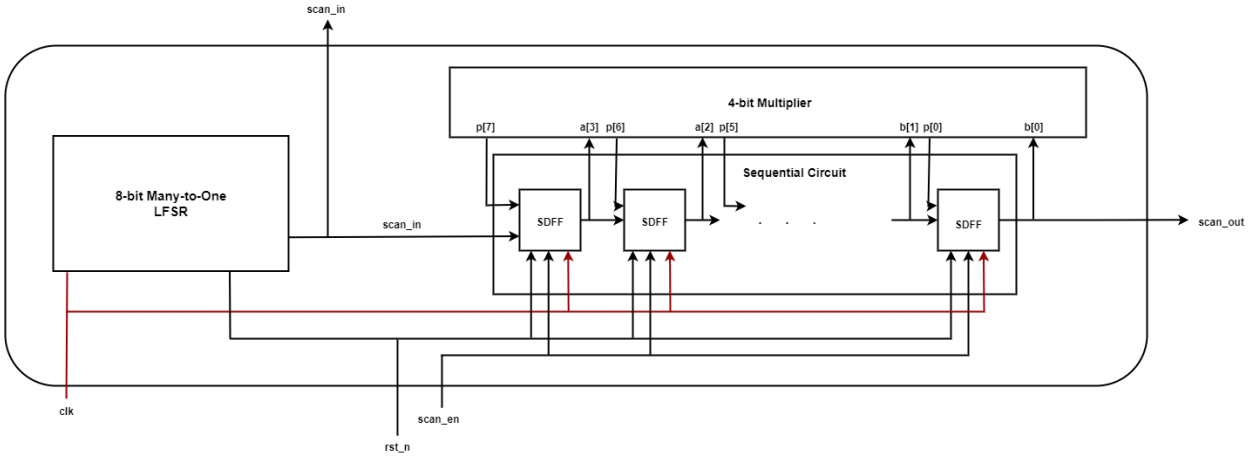
8-bit Scan Chain

Detail 基本上如前一題所述一樣，不同的是 `scan_in` 是由 8-bit Many-to-One LFSR 傳進。

Logic Diagram



Part of Built-in self test_8-bit Many-to-One



Part of Built-in self test_8-bit Scan Chain

Testbench

先用以下testbench找到Many to One LFSR為多少個數字循環一次。執行完以下tb，可以發先經過256次後LFSR的output會等於10111101（rst_n == 0時，output亦等於10111101）。

```
initial begin
    @(negedge clk)
    rst_n = 1'b0;
    repeat(2 ** 10) begin
        @(negedge clk) begin
        rst_n = 1'b1;
        i = i + 1;
        if(out == 8'b10111101)
            $display("at %d",i);
        end
    end
    $finish;
end
```

由於上述觀察，可得知經過256個scan chain循環後，能窮舉所有情況。因此將built-in scan chain重複執行256次，即可驗證此電路。

```
repeat(2 ** 8) begin

        for(i = 0; i < 8; i = i + 1) begin
            @(negedge clk) begin
            rst_n = 1'b1;
            scan_en = 1'b1;
            data[i] = scan_in;
            end
        end

        @(negedge clk) begin
        rst_n = 1'b1;
        scan_en = 1'b0;
        p = data[3:0] * data[7:4];
        {b, a} = data;
        end

        for(i = 0; i < 8; i = i + 1) begin
            @(negedge clk) begin
                rst_n = 1'b1;
                scan_en = 1'b1;
                m[i] = scan_out;
            end
        end
        if(m != p)
            $display("error m=%d p=%d", m, p);
    end
```

Mealy machine sequence detector

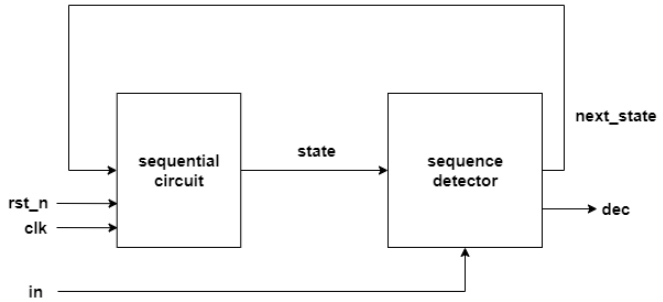
Design

每 4 bit 偵測一次數字，若為特定連續數字，0111、1001、1110，則 output `dec` 為 1。

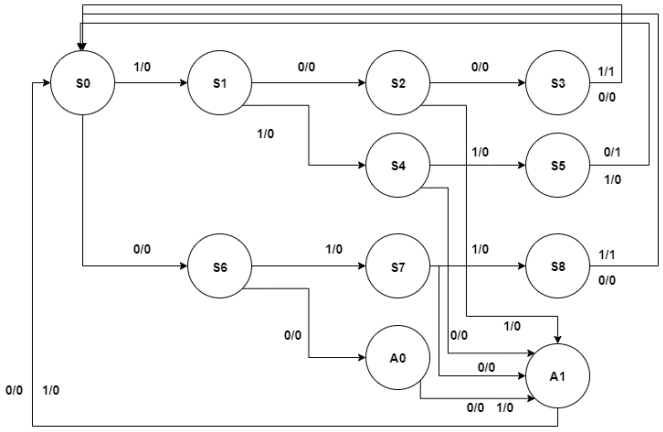
Detail

由於每 4 bit 偵測一次，故不須考慮 01110 這種情況要算兩次，而設計時加上了兩個 abort state，若第 2 個 bit 出現時已不可能有特定連續數字的情況時，將 state 移轉到 abort state (A0, A1)，讓它順利跑完剩下的 bit 並 output `0`。

Block Diagram



State Transition Diagram



Testbench

設計 testbench 時，先設置一個 `seq`，裡面有 4 bit，會從 `4'b0000` 跑到 `4'b1111`，接著，設置一 for loop，在 negative edge 時，會分別將 `seq` 的各個 bit 送進 `in` 裡面，經由 sequence detector 偵測後，可以看到在 `0111`、`1001`、`1110` 時 `dec` 有正確的顯示 1。

```
reg [4-1:0] seq = 4'b0000;
for(i = 0;i < 16;i = i + 1) begin
    @(negedge clk)
    seq = seq + 4'd1;
    rst_n = 1'b1;
    in = seq[0];

    @(negedge clk)
```

```
    rst_n = 1'b1;
    in = seq[1];

    @(negedge clk)
    rst_n = 1'b1;
    in = seq[2];

    @(negedge clk)
    rst_n = 1'b1;
    in = seq[3];
end
```

Learning

1. 在做 mealy, moore machine 類型的題目時，先把 state diagram 畫出來然後照著刻比較輕鬆
2. 要分清楚 combinational 和 sequential circuit，要切記不能 multiple assignment，在寫兩個 verilog code 時，同時腦袋也要把概念的電路圖描繪出來，比較不容易接錯搞混
3. 要注意不要用 high level 的語法去寫，要考量寫出來的 code 是否能正確轉化為電路圖實作
4. 瞭解 many-to-one LFSR 可做為亂數產生器去測試設計的 module

分工

- 蘇勇誠（108062373）

  Scan chain design

  Built-in self test

- 張晏瑄（108062273）

  State Transition Diagram of DFFs in many-to-one LFSR

  State Transition Diagram of DFFs in one-to-many LFSR

  Content-addressable memory (CAM)

  Mealy machine sequence detector