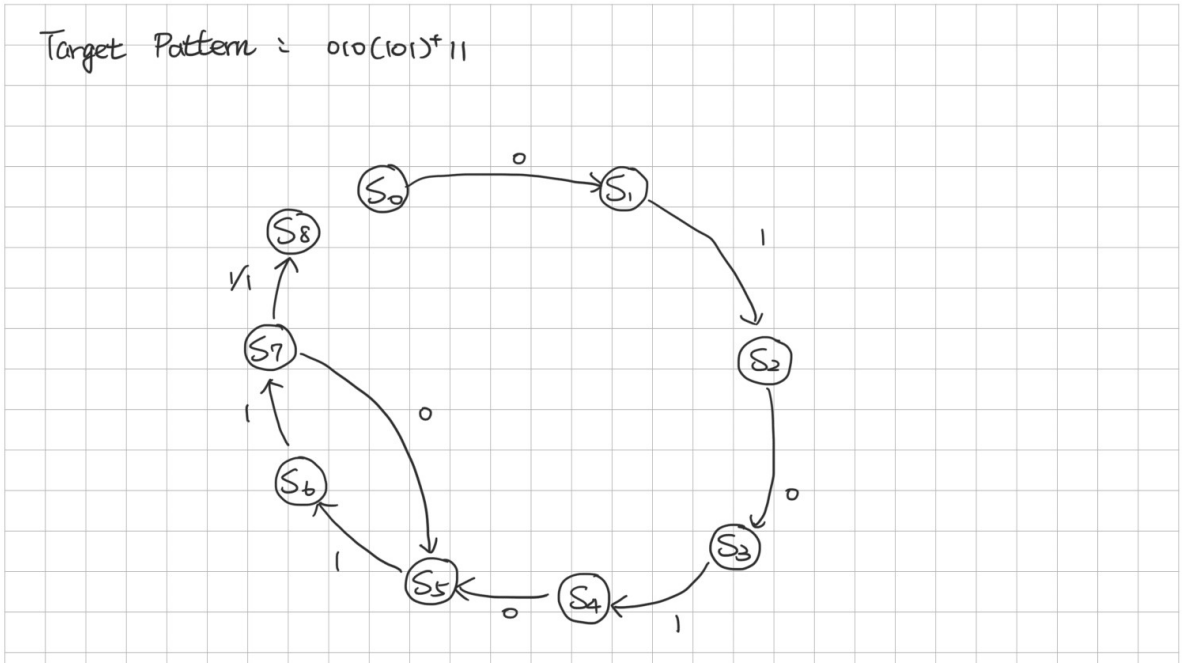
Lab 3 Pattern Matching

在這次的 Lab 當中我們要設計一個 FSM，去偵測當前的 Pattern 和 $010(101)^{+}11$ 是否相同，如果是就輸出 1，否則輸出 0。我在這次的 Lab 除了簡單的設計一個 FSM 以外，還有透過化簡做出另一個版本的解法，這裡會把兩種做法的設計過程都寫出來，分享一下在這次 Lab 當中學習到的知識。

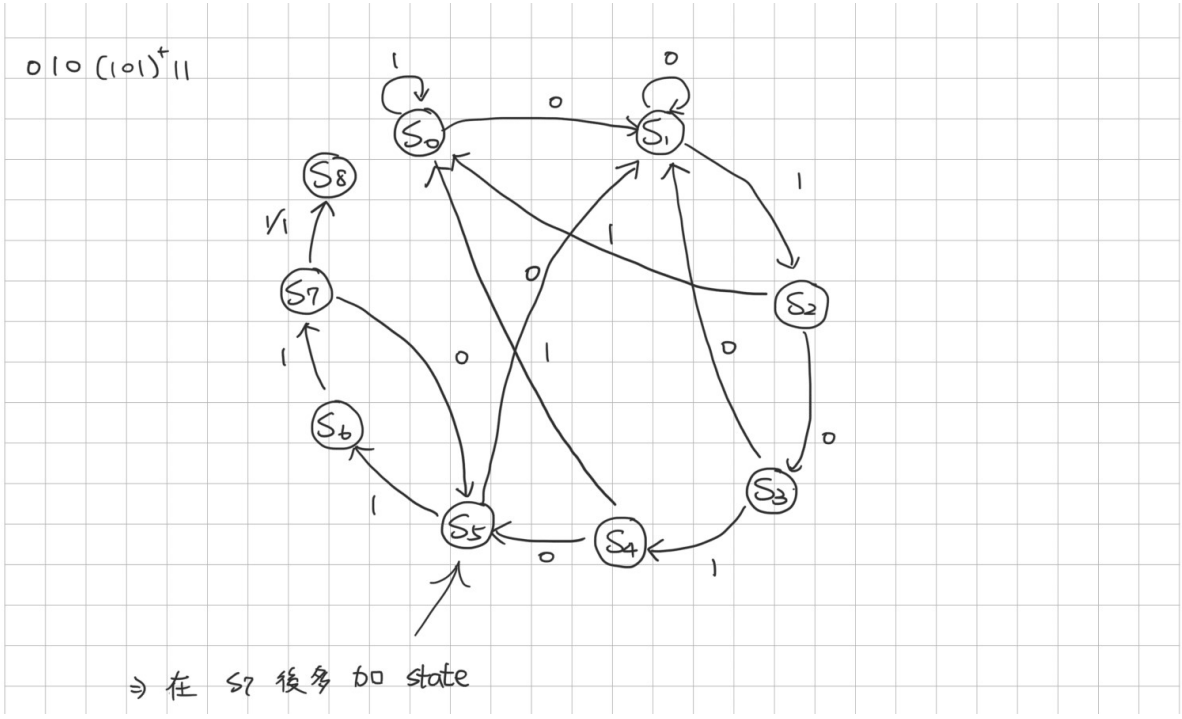
設計流程

這次的題目我覺得直觀上會想到要用 **Mealy Machine** 來實作，因此第一個步驟就是要畫出相對應的 State Diagram。畫 State Diagram 的時候不需要一開始就把每個 State 都畫出來，可以只先去看 Target Pattern 就好，這樣設計起來會比較順暢。我先依照 Target Pattern 畫出了一個可以符合的 State Diagram。

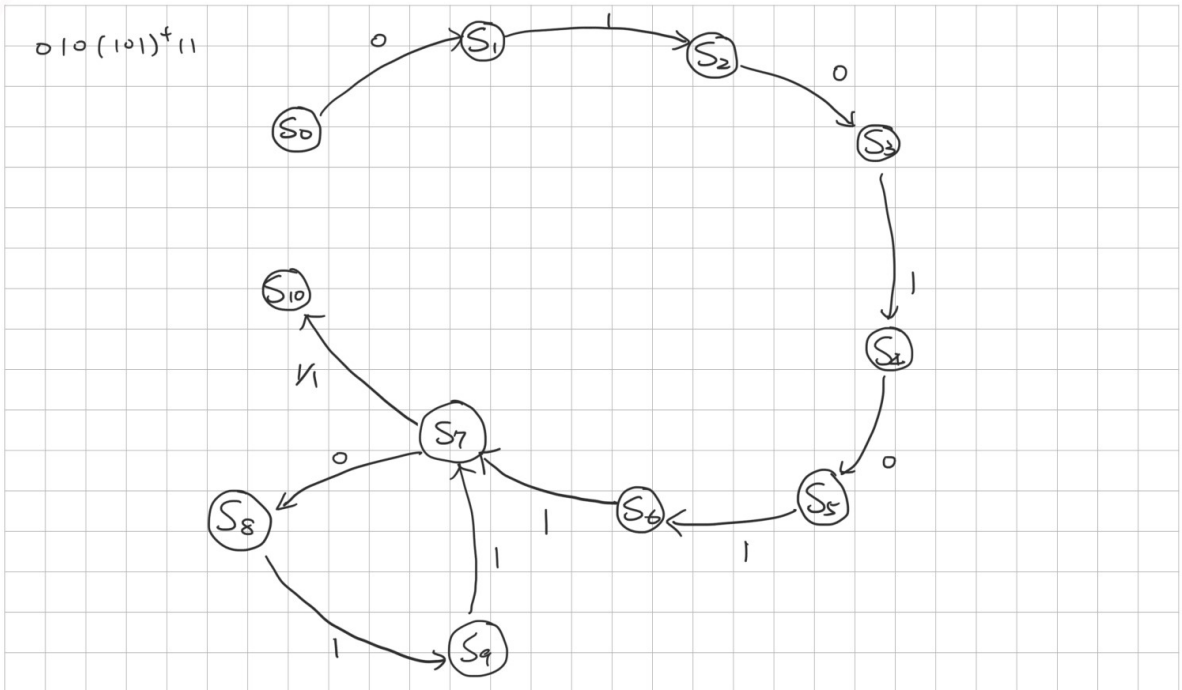


接下來嘗試把剩下的 State 加上去。但是在加上去的過程當中發現到這樣的 State Diagram 會有問題，如果在 S_6 接收到 0 的話，現在的 Pattern 可能是 0101010，這時候因為後綴有 01010，下一個狀態應該要是 S_5，然而這裡的 Pattern 也有可能會在 S_5~S_7 當中繞了幾圈才到 S_6，這時候的 Pattern 可能會是 0101011010，在這種情況下的下一個狀態應該要是 S_3，所以出現問題。

因此，考慮再多加 State 來把這兩種情況分開使其不衝突。

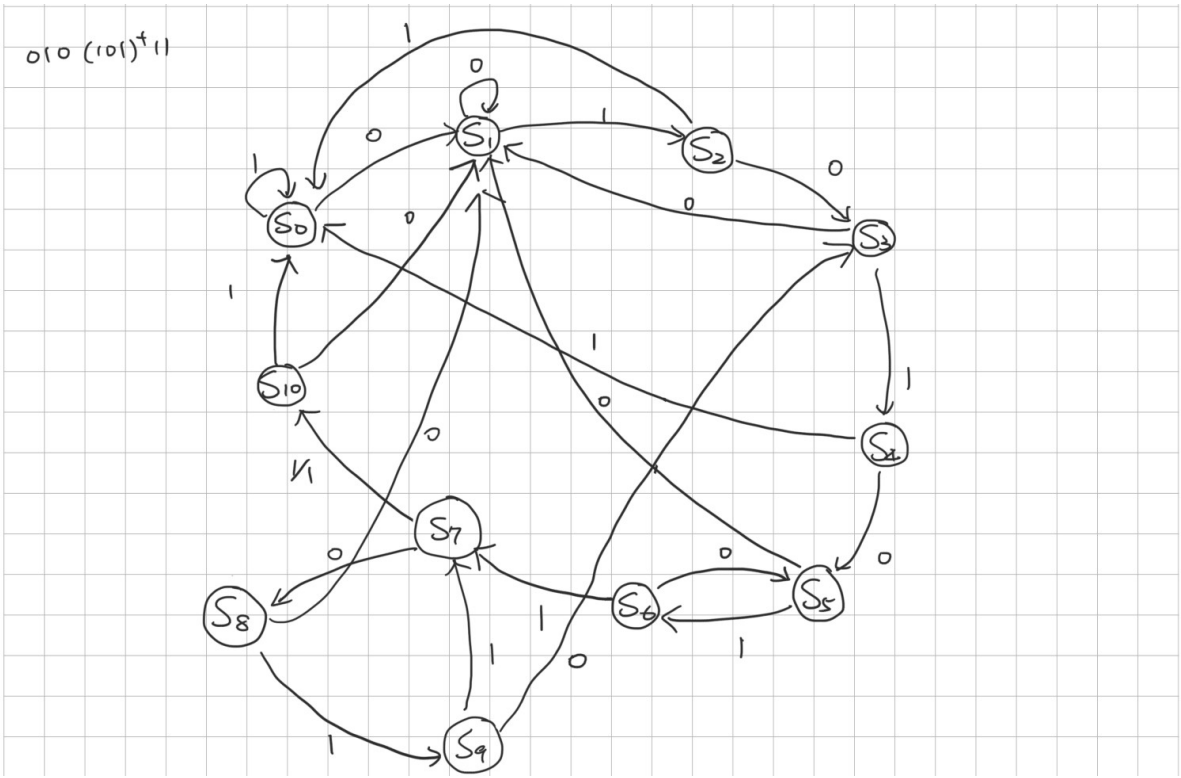


依照這樣的想法又畫了一張 State Diagram，這次如果要循環 101 的話會進到 S_7~S_9 的環當中，就可以跟 S_6 分開來考慮了。



接著一樣將剩餘的狀態補齊，這就完成了完整的 State Diagram 了！

> 附註：因為只有少數 output 為 1，因此這裡僅標記 1，沒有標記 output 的部份其 output 皆為 0。



有了 State Diagram 後，接下來就可以著手設計 verilog code，在 module 當中會記錄當前是在哪個 State，每個 State 我們已經知道在輸入為 0 和 1 時分別需要轉移到哪一個狀態，也知道應該要輸出什麼，那麼只需要用 `case` 就可以完成了。

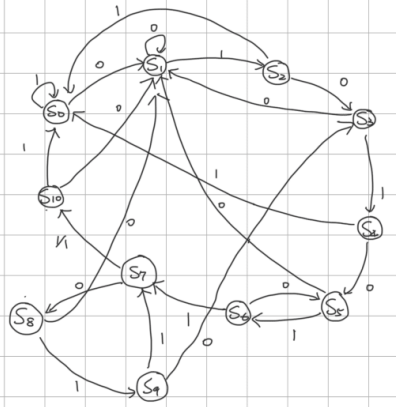
```
module PAT(clk, reset, data, flag);

    input clk, reset, data;
    output reg flag = 0;
    reg [3:0] state = 0;

    always @(posedge clk, posedge reset) begin
        if(reset == 1) begin
            flag <= 0;
            state <= 0;
        end
        else begin
            case(state)
                4'b0000: begin
                    if(data == 0) begin
                        state <= 4'b0001;
                        flag <= 0;
                    end
                    else begin
                        state <= 4'b0000;
                        flag <= 0;
                    end
                end
                4'b0001: begin
                    if(data == 0) begin
                        state <= 4'b0001;
                        flag <= 0;
                    end
                    else begin
                        state <= 4'b0010;
                        flag <= 0;
                    end
                 end
```

```
                // 以下雷同，省略不列出
            endcase
        end
    end
endmodule
```

簡化

雖然這樣的邏輯十分簡單易懂，但是 code 寫起來就是有很多重複的部分，過去在邏設的課程當中有教過用 Kmap 來直接得到輸入輸出等式的方法，接下來我就會依照這個 Design Flow 設計出另一個版本，也是我上傳的版本。流程大致上是先依照上面的 State Diagram 畫出 State Table，接下來透過 Kmap 得到 I/O 的 Equations，最後畫出 circuit 後完成 code。

雖然據說不同的 State Assignment 方式會對最後的 circuit 造成影響，不過目前我對於應該怎麼 assign 沒有太多想法，因此就直接先透過 10 進位轉 2 進位來做 assign。因為我們有 S_0~S_{10} 總共 11 個 State，因此我們至少需要 4 個 bits 才能夠表示。在 Filp-Flop 的選擇上我選擇 D F/F 來實作，照著 State Diagram 以及 State Assignment 完成下面兩張表格。



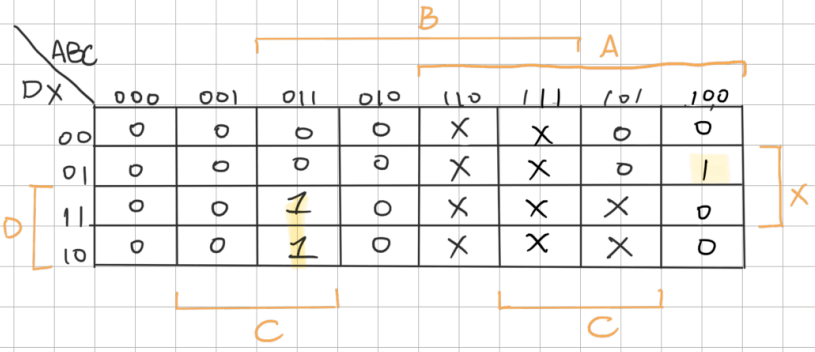
State Assignment

S_0	0000
S_1	0001
S_2	0010
S_3	0011
S_4	0100
S_5	0101
S_6	0110
S_7	0111
S_8	1000
S_9	1001
S_{10}	1010

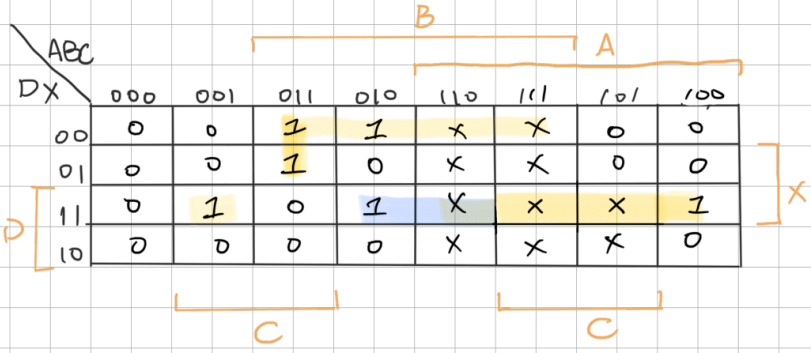
Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
S_0 0000	0001	0000	0	0
S_1 0001	0001	0010	0	0
S_2 0010	0011	0000	0	0
S_3 0011	0001	0100	0	0
S_4 0100	0101	0000	0	0
S_5 0101	0001	0110	0	0
S_6 0110	0101	0111	0	0
S_7 0111	1000	1010	0	1
S_8 1000	0001	1001	0	0
S_9 1001	0001	0111	0	0
S_{10} 1010	0001	0000	0	0

P.S. A B C D	$A^+B^+C^+D^+$ X=0	$A^+B^+C^+D^+$ X=1	D_A X=0	D_A X=1	D_B X=0	D_B X=1	D_C X=0	D_C X=1	D_D X=0	D_D X=1
0 0 0 0	0001	0000	0	0	0	0	0	0	1	0
0 0 0 1	0001	0010	0	0	0	0	0	1	1	0
0 0 1 0	0011	0000	0	0	0	0	1	0	1	0
0 0 1 1	0001	0100	0	0	0	1	0	0	1	0
0 1 0 0	0101	0000	0	0	1	0	0	0	1	0
0 1 0 1	0001	0110	0	0	0	1	0	1	1	0
0 1 1 0	0101	0111	0	0	1	1	0	1	1	1
0 1 1 1	1000	1010	1	1	0	0	0	1	0	0
1 0 0 0	0001	1001	0	1	0	0	0	0	1	1
1 0 0 1	0011	0111	0	0	0	1	1	1	1	1
1 0 1 0	0001	0000	0	0	0	0	0	0	1	0

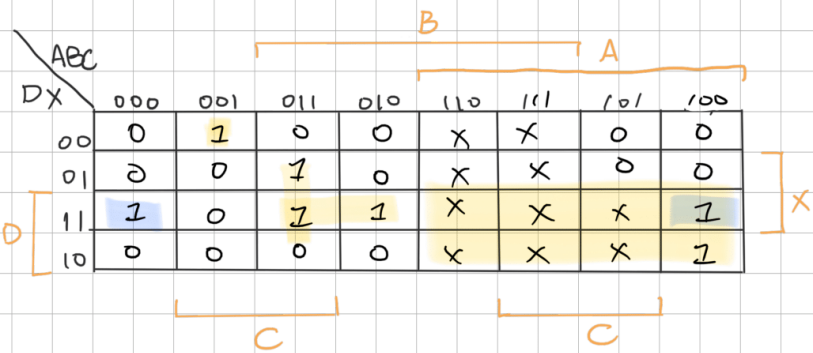
有了這張表格，接下來就可以畫出 Kmap 了！不過這次的輸入輸出總共有 5 個變數，這個 Kmap 比較複雜一點。



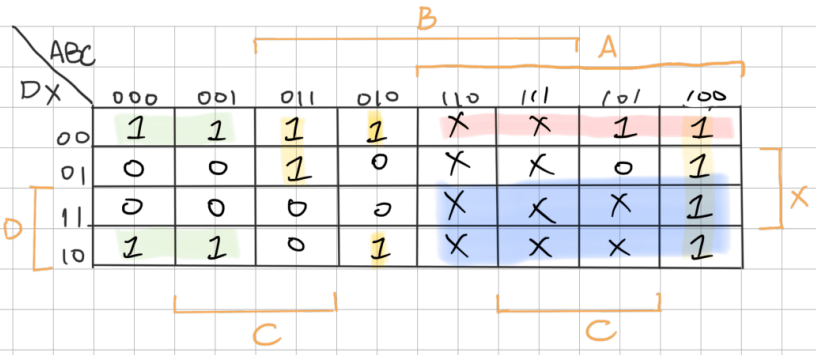
$$D_A = BCD + AB'C'D'X$$



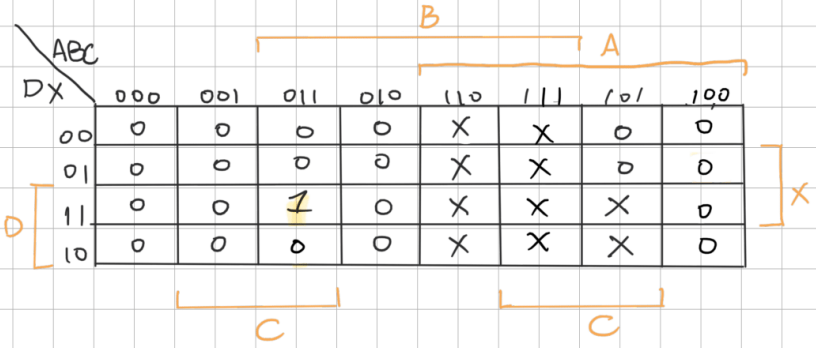
$$D_B = A'B'CDX + A'BCD' + BD'X' + BC'DX + ADX$$



$$D_C = B'C'DX + A'B'CD'X' + A'BCX + A'BDX + AD$$



$$D_B = A'B'X' + A'BCD' + A'BC'X' + AD'X' + AD + AB'$$



$$Z = A'BCDX$$

Result

$$\begin{cases} D_A = BCD + AB'C'D'X \\ D_B = A'B'CDX + A'BCD' + BD'X' + BC'DX + ADX \\ D_C = B'C'DX + A'B'CD'X' + A'BCX + A'BDX + AD \\ D_B = A'B'X' + A'BCD' + A'BC'X' + AD'X' + AD + AB' \end{cases}$$

$$Z = A'BCDX$$

根據上面的 Kmap 我們得到了每個輸入輸出的結果，那麼接下來只需要實作 D F/F 以及每個輸入輸出即可完成。

$$
\begin{cases}
D_A = BCD + AB'C'D'X \\
D_B = A'B'CDX + A'BCD' + BD'X' + BC'DX + ADX \\
D_C = B'C'DX + A'B'CD'X' + A'BCX + A'BDX + AD \\
D_D = A'B'X' + A'BCD' + A'BC'X' + AD'X' + AD + AB' \\
Z = A'BCDX
\end{cases}
$$

首先是 D Flip Flop 的部份，在接收到 reset 訊號的時候將 $Q, Qbar$ 分別設為 $1, 0$。

```
module D_FF(D, clk, rst, Q, Qbar);
    input D, clk, rst;
    output reg Q = 0, Qbar = 1;

    always @(posedge clk, posedge rst) begin
        if(rst == 1) begin
            Q <= 0;
            Qbar <= 1;
        end
        else begin
            Q <= D;
            Qbar <= ~D;
        end
    end
endmodule
```

接下來輸入輸出的部分就依照上方的等式構造就完成了。

```
module PAT(clk, reset, data, flag);

    input clk, reset, data;
    output reg flag;
    wire AQ, AQbar, BQ, BQbar, CQ, CQbar, DQ, DQbar;

    always @(posedge clk, posedge reset) begin
        if(reset == 1) begin
                flag <= 0;
        end
        else begin
               flag <= AQbar & BQ & CQ & DQ & data;
        end
    end

    D_FF A(
        (BQ & CQ & DQ) | (AQ & BQbar & CQbar & DQbar & data),
        clk, reset, AQ, AQbar
    );
    D_FF B(
        (AQbar & BQbar & CQ & DQ & data) | (AQbar & BQ & CQ & DQbar) | (BQ & DQbar & (~data)) |
(BQ & CQbar & DQ & data) | (AQ & DQ & data),
        clk, reset, BQ, BQbar
    );
    D_FF C(
        (BQbar & CQbar & DQ & data) | (AQbar & BQbar & CQ & DQbar & (~data)) | (AQbar & BQ & CQ
& data) | (AQbar & BQ & DQ & data) | (AQ & DQ),
        clk, reset, CQ, CQbar
    );
    D_FF D(
        (AQbar & BQbar & (~data)) | (AQbar & BQ & CQ & DQbar) | (AQbar & BQ & CQbar & (~data)) |
(AQ & DQbar & (~data)) | (AQ & DQ) | (AQ & BQbar),
        clk, reset, DQ, DQbar
    );
```

```
endmodule
```

遇到的問題

1. State Diagram 有點難第一次就知道哪裡會有問題

一開始覺得只需要先畫基本的 State Diagram，再來補完剩下的狀態就可以很快完成了，但是沒有想到中間會遇到還需要多加一些 State 才能畫出正確的 State Diagram。除此之外，補完剩下的狀態其實也很吃力，要去考慮要轉移到哪個狀態才是正確的。

2. 實作簡化版本

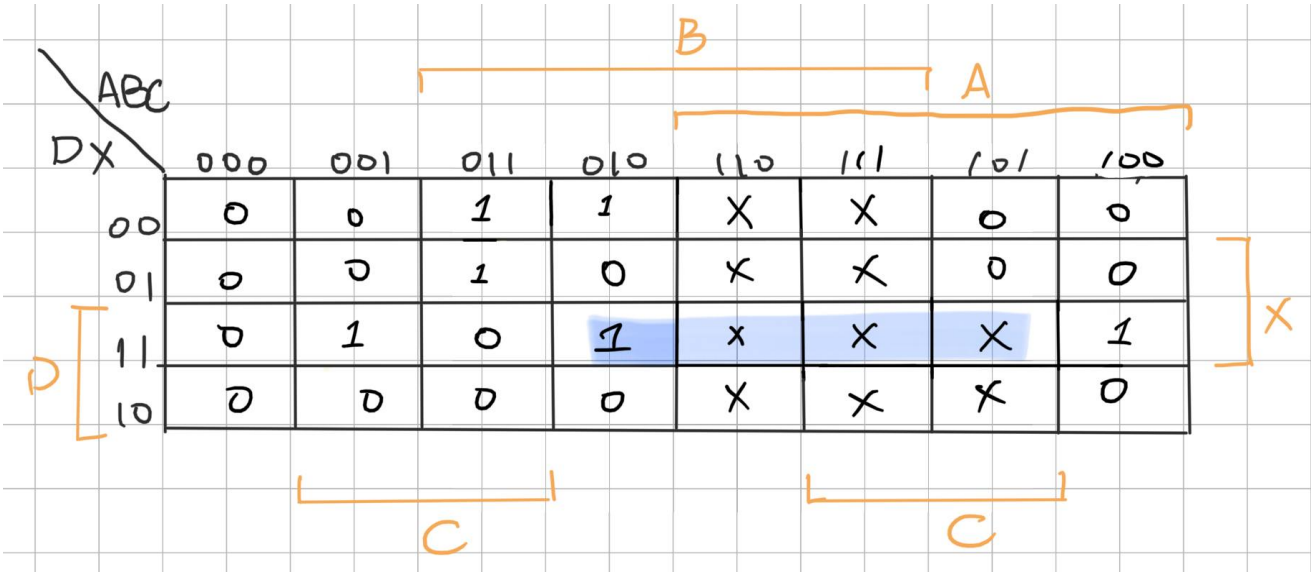
實作簡化版本的時候我曾寫了下面的 Code

```
module PAT(clk, reset, data, flag);

    input clk, reset, data;
    output reg flag = 0;

    wire AQ = 0, AQbar = 1;
    wire BQ = 0, BQbar = 1;
    wire CQ = 0, CQbar = 1;
    wire DQ = 0, DQbar = 1;

    // 略
endmodule
```

一開始我想說也許這樣可以確保第一次執行的時候初始值為 0 和 1，但是這樣的做法實際上是把這個接線的值變成一個 constant，因此跟我原先預想的設計是有出入的，在當時一直遇到輸出內容包含 `x` 或 `z` 的錯誤訊息，花了不少時間 debug，後來和學長討論過後才發現到原來是這個地方出了問題。

3. Kmap

這次的 Kmap 是有 5 個變數的，在求出 D_B 的等式的時候我原本預計這樣做，但是卻意外發現沒有辦法這樣化簡，最後只好只看 `1X` 而已。



想問助教的問題

1. 為什麼 always block 當中會需要一個 if statement

在設計的過程當中我寫了下列的 code，在 `sim` 都會是正常的，我也覺得運算起來是合理的，但是要透過 `dc_shell` 產生 `syn` 檔案的時候就會出現下面的錯誤訊息，我想知道為什麼。

```
module PAT(clk, reset, data, flag);

    input clk, reset, data;
    output reg flag;
    wire AQ, AQbar, BQ, BQbar, CQ, CQbar, DQ, DQbar;

    always @(posedge clk, posedge reset) begin
        flag <= AQbar & BQ & CQ & DQ & data;
    end
    // 略
endmodule
```

```
Error:  /users/course/2022S/LD17100000/u110062126/Assignment/3_Pattern_Maching/PAT.v:24: The statements in this 'always' block are outside the scope of the sy
nthesis policy. Only an 'if' statement is allowed at the top level in this always block. (ELAB-302)
```