109021115 吳嘉濬

The design of your FSM

根據老師 L5 講義第 49 頁所展示設計 FSM 的主要步驟，開始著手設計這次 lab 的 FSM。首先是依照題目要求畫出它的 state diagram，因為 input data 只有兩種可能，及”0”或”1”，所以對每一個 state 而言，他們的 outdegree 皆為 2，此訊息可作為檢查是否有遺漏的情況發生。我以 state A 表示初始狀態，state B 表示有 match 到前 1 個 bit，state C 表示有 match 到前 2 個 bits，以此類推，一直到 state G 表示有 match 到前 6 個 bits，至於 state H, I, J 則比較特別，因為這次的 pattern 中間允許在特定位置重複”101”，所以 state H, I, J 分別對應有 match 到前 7 + 3t, 7 + (3t+1), 7 + (3t+2)個 bits，其中 t 為非負整數。定義好每個 state 代表的意義後，從初始狀態 state A 開始畫，每個 state 都要畫出 input data 是”0”, “1”時下一個 state 會是如何，如果符合題目 pattern 該位置的 bit 則往下一個 state 邁進；如果沒有的話，則判斷目前為止所偵測到的 pattern，從最尾端往前一個 bit 一個 bit 的對照，找出符合 pattern 且是 match 到最多 bit 的情況下所對應的 state，做為要前往的 state，而如果一個都 match 不了，以這次 lab 為例，即最後一個 bit 是 1，則回到初始狀態 state A。以這種方法我們能夠完整地把 state diagram 畫出來。

畫完 state diagram 之後，開始畫 state table，在那之前，為了之後能夠把我設計的 FSM 轉化成 verilog 的 code 或是方便做 K map，我賦予每個 state 一個 binary code，賦予 state A 為 0000，state B 為 0001，以此類推，一直到 state J 為 1001，隨後依照前面所繪製的 state diagram 畫出完整的 state table。

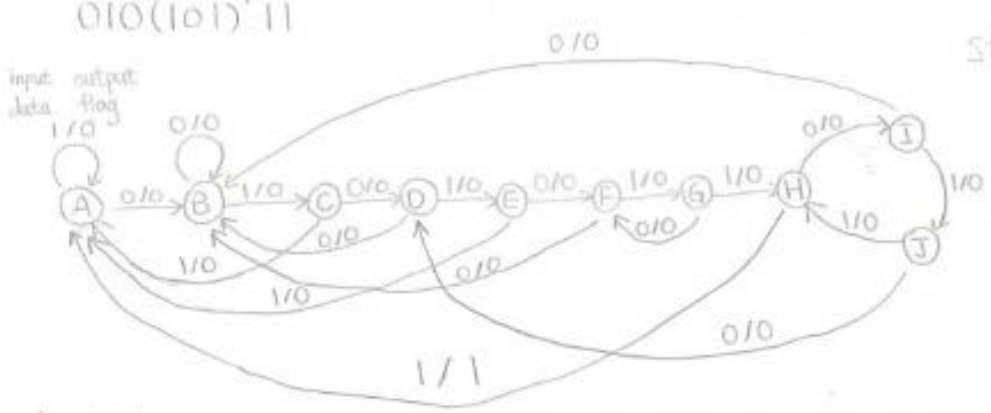
最後，做 state reduction，檢查是否存在兩個或以上的 equivalent state，雖然早在畫 state diagram 時就已經考慮過了，即 state H, I, J 這三大類，所以做到這一步時已經沒有需要 reduce 的 state 了。

經由以上這些步驟，前置作業便準備完成，得到的這些資訊足以讓我完成這次的 lab(不用做到最後一步，也就是用 K map 找出 state equation 和 output equation)。

下圖為 state diagram 和 state table 的完整詳細內容。

$010(101)^+11$

state diagram



A: initial state
B: matched the first bit
C: matched the first 2 bits
⋮
G: matched the first 6 bits
H: matched the first $7+3t$ bits, $t\geq 0, t\in\mathbb{Z}$
I: matched the first $7+3t+1$ bits, $t\geq 0, t\in\mathbb{Z}$
J: matched the first $7+3t+2$ bits, $t\geq 0, t\in\mathbb{Z}$

state table

present state	next state		output flag	
	data=0	data=1	data=0	data=1
A 0000	0001	0000	0	0
B 0001	0001	0010	0	0
C 0010	0011	0000	0	0
D 0011	0001	0100	0	0
E 0100	0101	0000	0	0
F 0101	0001	0110	0	0
G 0110	0101	0111	0	0
H 0111	1000	0000	0	1
I 1000	0001	1001	0	0
J 1001	0011	0111	0	0

How you implement your FSM in verilog

這次 lab 的 FSM 寫在 PAT.v 檔底下，檔案內只有唯一的 module 名為 PAT，此 module 裡主要分為三大區塊，各自有各自的功能，以下分別詳細說明。

1. state register

以 always block 作為主體，並把 posedge clk 擺在 activity list 內，意味著這整個 FSM 是 positive clock triggered。

在 reset=1 的情況下，我讓 state 設為 0000，即初始狀態 state A。

在 reset=0 的情況下，state<=next_state，可以得知我是以 D flip-flop 作為設計 FSM 時所使用的 memory element。

下圖為 state register 的完整 code。

```
always@(posedge clk)begin
        if(reset)
                state<=4'b0000;
        else
                state<=next_state;
end
```

2. next_state combinational circuit

一樣以 always block 為主體，activity list 擺” * “，意味著只要 always block 內的訊號有所改變就要跑一次整個 always block，利用此方法可以避免少考慮訊號的事情發生。

在 reset=1 的情況下，我讓 next_state 設為 0000，即把下一狀態設為初始狀態 state A。

在 reset=0 的情況下，考慮現在的 state，並依照 input data 的值，對應並找出 next_state 為何。

值得一提的是，我們是找到 next_state 的值，而非直接改 state 的值，一直到下一次 positive clock 才會賦予 next_state 的值到 state，成為”現在”的 state 值。

下圖為 next_state combinational circuit 的部分 code。

```
always@(*)begin
        if(reset)
                next_state=4'b0000;
        else begin
                if(state==4'b0000)begin
                        if(data==1'b0)
                                next_state=4'b0001;
                        else
                                next_state=state;
                end
                else if(state==4'b0001)begin
                        if(data==1'b0)
                                next_state=state;
                        else
                                next_state=4'b0010;
                end
                else if(state==4'b0010)begin
                        if(data==1'b0)
                                next_state=4'b0011;
                        else
                                next_state=4'b0000;
                end
```

3. output combinational circuit

依舊是以 always block 為主體，activity list 擺” * “，意味著只要 always block 內的訊號有所改變就要跑一次整個 always block，利用此方法可以避免少考慮訊號的事情發生。

在 reset=1 的情況下，state 回歸初始狀態 state A，很值觀地可以知道 output flag=0。

在 reset=0 的情況下，考慮現在的 state 以及 input data，唯有在 state 0111 時且 input data 為 1 時 output flag 才是 1，其他情況下 output flag 都是 0。

值得補充的是，因為我讓 flag 在 always block 底下操作並得出 flag 的值而非單純用 assign 的方式去賦值，所以我在定義 output flag 時有多加 reg，讓 flag 不是預設的 wire signal。

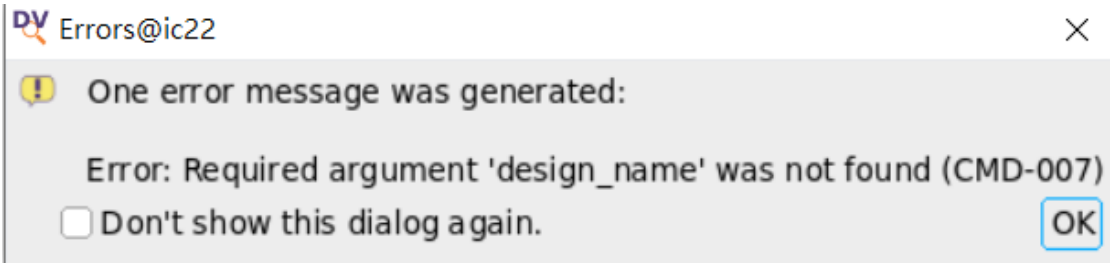
```
output reg flag;
```

下圖為 output combinational circuit 的完整 code。

```
always@(*)begin
        if(reset)
                flag=1'b0;
        else begin
                if(state==4'b0111)begin
                        if(data==1'b1)
                                flag=1'b1;
                        else
                                flag=1'b0;
                end
                else
                        flag=1'b0;
        end
end
```

The problems you faced and how you deal with it

做這次 lab3 的過程中遇到的瓶頸相對於 lab2 少上許多，或許可以說是幾乎沒有，在實作 verilog 或是使用 design vision 時，大致上沒有遇到比較明顯的困難，唯有之前在做 lab2 使用 design vision 在 elaborate 時，出現的以下 error 訊息依舊存在。



而根據 lab2 的經驗，如果我無視它繼續往下做的話，仍然可以正常執行 synthesis，這次也不例外。至於會跳出這個視窗的原因，我把它歸咎於版本過舊，其實我第一次使用 MobaXterm 這個軟體不是在邏設課程中，而且我也不會特別定期去手動更新它，所以版本應該還停留在 1, 2 年前，導致這個 error 發生。

不同於前幾次 lab 遇到的困難都是使用電腦 simulation 和 synthesis 等技術問題，這次 lab 出現的問題反而是發生在設計 FSM 這件事上。我原先在定義 state 時其實沒有 state I 和 state J，因為我直觀認為 state I 和 state F 等價，state J 和 state G 等價(以 match pattern 的式子來看是位在同一個 bit 的位置)，所以當位在 state H，input data=0 時，我一開始是讓他跑到 state F 而非改良後新增的

state I。然而，當我考慮 0101011010111 這一字串時，照著 state diagram 走是有 match 到整個 pattern 並 output flag=1 的，但實際上是沒有的 match 到，這才發現原來 state F 不等價於 state I，因為兩者在面對 input data=1 時對應的 next state 其實不一樣，即 state G 和 state J，至於 state G 和 state J 不等價的地方在於，當 input data=0 時，state G 的 next state 是 state F，而 state J 的 next state 是 state D。

The questions you want to ask TAs

聽學長說下學期的邏輯設計實驗 loading 還蠻重的，有超多的 lab 等著我，想知道是不是到時候每一次 lab 都要自己設計 testbench，看了一下這次的 testbench 我感覺比設計 module 本身還要複雜 QQ。