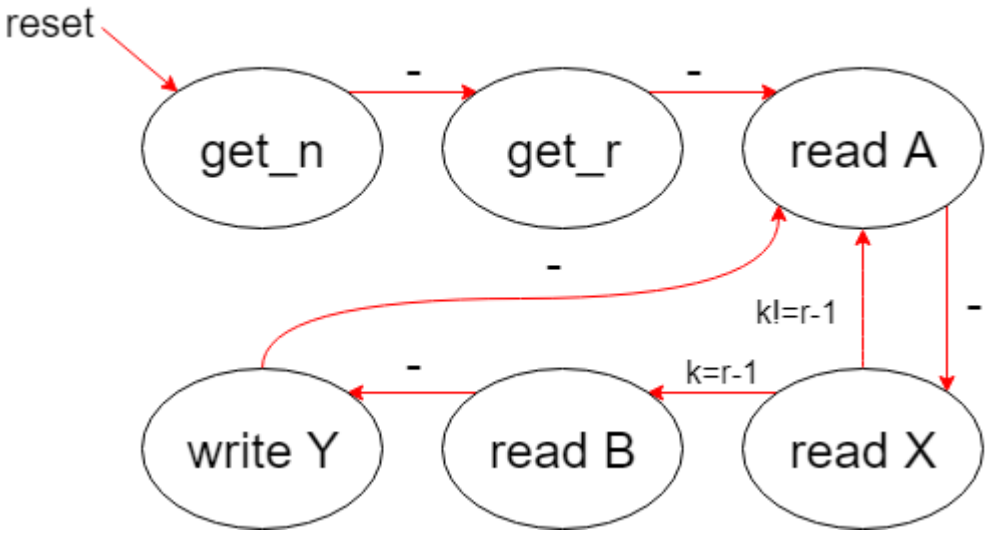
Logic Design

210510210 詹其佐

Lab4

1.state 的設計&解釋



- 這次 lab 要做的是 Y=A*X+B，並且根據 opcode 來決定現在要取得 n, r 或者讀取 A, X, B 還是把值寫到 Y。我的做法是用 I, J 來記錄 Y 的 row 跟 col，因為 A 的 i 就等於 Y 的 I，而 X 的 j 就等於 Y 的 J (這邊的 i, j 有大小寫之分，小寫代表 output，大寫是我自己設的變數)，而 B 的 I, j 都跟 Y 的 I, J 一樣。我又設了一個變數 k，因為 A 的 j 和 X 的 i 一樣，k 會累加，直到 k=r-1 就代表 A 跟 X 的運算都做完了，之後就會跳到 B 去，這時的 i 跟 j 就直接用 I, J 就可以了。

- State get_n: 目的是取得 n 的值，用 opcode 000 取得。
- State get_r : 目的是取得 r 的值，用 opcode 001 取得，並且無條件進入 state read_A。
- State read_A: 讀取 A 後，將 A 存在 DFF_A 裡面。無條件進入 state read_X。此時 i=I, j=k 這樣讀到的 A 才是對的。Ans 此時存取的是 0 或者是累加過程中的 ans。
- State read_X: 讀到 X 後，就可以將 ans 存成 ans+A*X 此時的 X 是 in_data。接著將下一個 k+1，存入 DFF。此時 i=k, j=J。在這個 state 會判斷，k 是否等於 r-1，如果 k!=r-1 就回到 read_A 繼續做運算，否則就跳到 read_B 去做最後一步的加法。
- State read_B: 這裡的 ans=ans+B，B 是 in_data，這裡的 i=I, j=J 因為 B 的 I, j 跟 Y 的一樣。最後跳到 state write_Y。
- State write_Y: 這邊有三個工作，(1)去更改下一個 I, J，(2)輸出 Y，(3)判斷 fin 要不要拉起來。第一個工作判斷 J 是不是走到那一列的最後一個了，如果是那 I++且 J=0，不然就 I 不動 J++。第二個動作就是將 opcode 設成 101，同時設定 out_data = ans。第三個工作，就是他 I, J

走完最後一個矩陣的最後一個元素，他會跑到矩陣左下角元素的下面。如下圖:

1	2	3
4	5	6
7	8	9

會跑到這裡

所以，我們判斷 I=n, J=0 時，fin 就會拉起來。

2. Ncverilog Simulation Result:

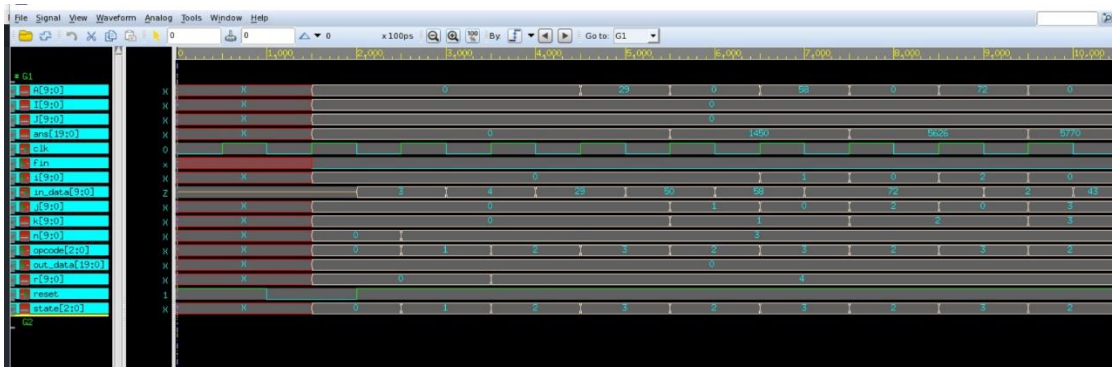
- Before Design Vision synthesis:

```
matrix size ||| n =    3, r =    4
---------------------------------
Row:          0 Column:          0 is correct.
Row:          0 Column:          1 is correct.
Row:          0 Column:          2 is correct.
Row:          1 Column:          0 is correct.
Row:          1 Column:          1 is correct.
Row:          1 Column:          2 is correct.
Row:          2 Column:          0 is correct.
Row:          2 Column:          1 is correct.
Row:          2 Column:          2 is correct.
---------------------------------
Congratulation!!!
```

- After Design Vision synthesis:

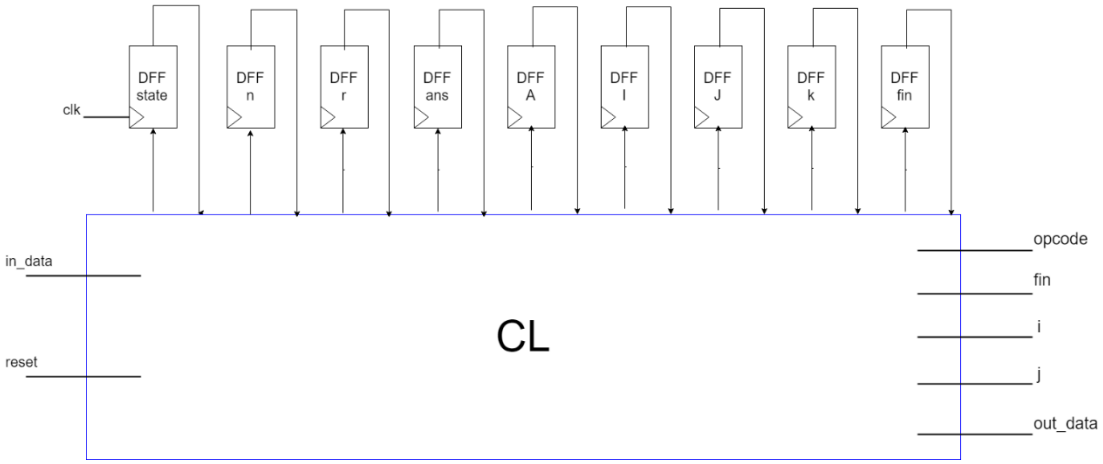
```
Row:          0 Column:          0 is correct.
Row:          0 Column:          1 is correct.
Row:          0 Column:          2 is correct.
Row:          1 Column:          0 is correct.
Row:          1 Column:          1 is correct.
Row:          1 Column:          2 is correct.
Row:          2 Column:          0 is correct.
Row:          2 Column:          1 is correct.
Row:          2 Column:          2 is correct.
---------------------------------
Congratulation!!!
```

3. nWave Result:

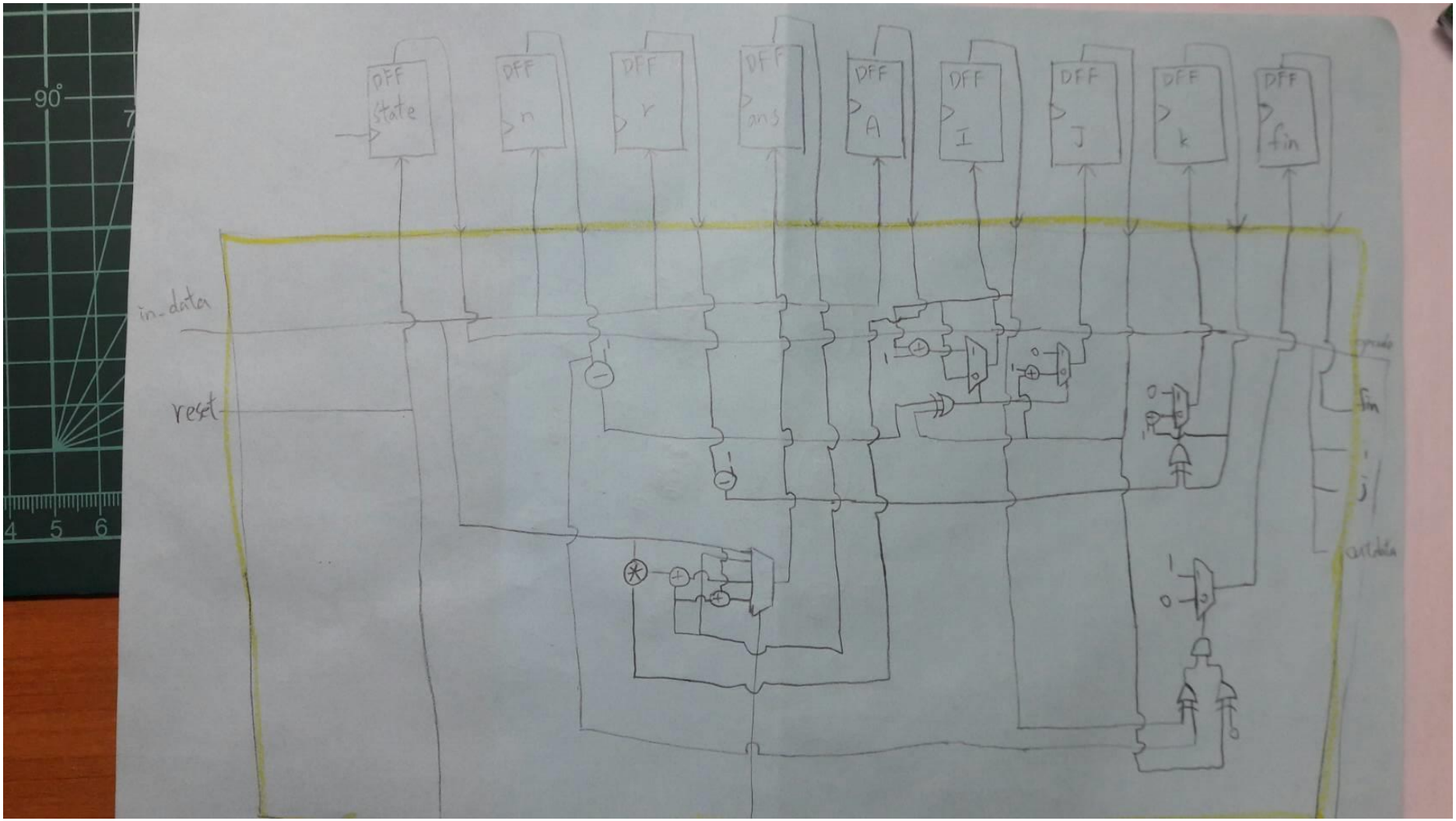


原本我以為沒有 reset 而且連 clk 都是整片綠的，後來才發現原來要放大，不然他預設好像就蠻緊密的。

4. Block Diagram:



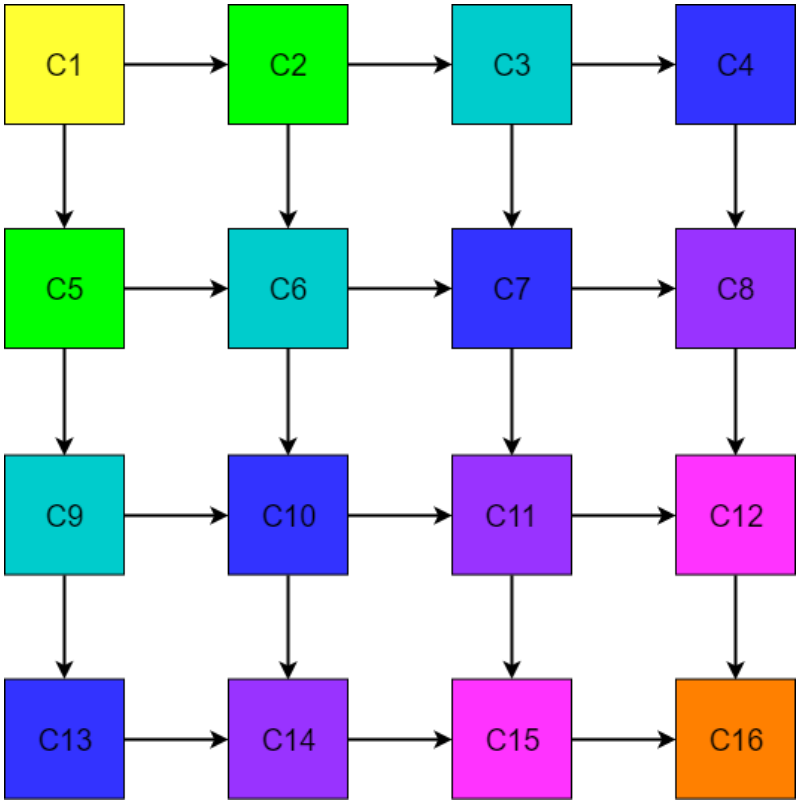
CL 裡面如下圖，其中裡面有一些小的判斷，來決定 I 跟 J 的值，fin 則是用一個 mux 來看他要為 1 還是 0，A 會拿來跟 in_data 相減跟相加。


DFF
state
DFF
n
DFF
r
DFF
ans
DFF
A
DFF
I
DFF
J
DFF
k
DFF
fin
in_data
reset
fin
j

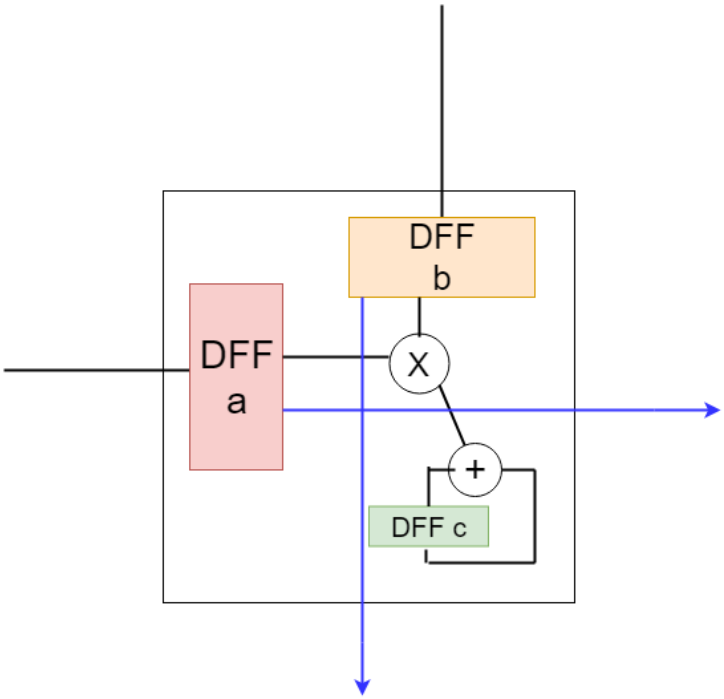
5. Problem encountered and discussion:

- 由於有做過 lab3 所以這次的 lab 比較好上手一點，但中間也還是有遇到很多問題。
- 在處理 index 的時候一開始沒想到用大寫的 I, J 去固定住，所以想得很痛苦，不知道該怎麼才能按照要求算出來。後來想到就又處理不了 k 的問題，所以就用了很多的 DFF。
- 一開始我不知道 opcode 是跟著他自己的 state，我以為也是傳入這個 opcode 他會在下個 state 作用，後來搞清楚了，他不是在 DFF 裡面。
- 本來我的 fin 會慢很多個 cycle 才拉起來，後來詢問助教，照著助教的建議用 DFF 來存，就改善很多。

6.Bonus



- SA: 先假設 PE 做完了，把接口按照圖形接上，在做的過程中我發現，如果沒有 delay 那 16 個 PE 會一起做，所以我在 reset 的時候讓他延遲一下，確保他在執行前的一個 cycle 會成功 reset。下面實作 PE。



- PE: 我用到 3 個 DFF，一個拿來存進來的 a，一個存 b，還有一個會存答案 c，新的 c 會等於舊的 c 再加上 a, b 的乘積。當做到 a, b 都剩下 0 的時候，c 就會保持住。