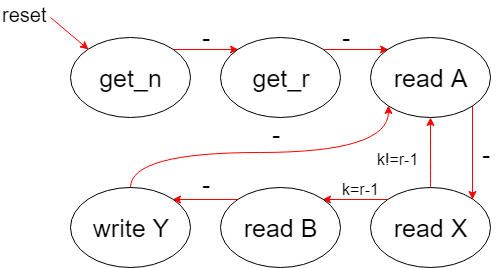
Logic Design

210510210 詹其侁

Lab4

1.state的設計&解釋



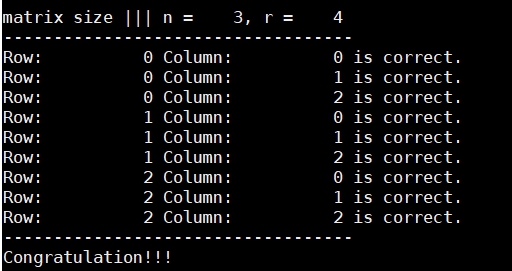
* 這次lab要做的是Y=A\*X+B，並且根據opcode來決定現在要取得n, r或者讀取A, X, B還是把值寫到Y。我的做法是用I, J來記錄Y的row跟col，因為A的i就等於Y的I，而X的j就等於Y的J (這邊的i, j有大小寫之分，小寫代表output，大寫是我自己設的變數)，而B的I, j都跟Y的I, J一樣。我又設了一個變數k，因為A的j和X的i一樣，k會累加，直到k=r-1就代表A跟X的運算都做完了，之後就會跳到B去，這時的i跟j就直接用I, J就可以了。
* State get\_n: 目的是取得n的值，用opcode 000取得。
* State get\_r : 目的是取得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:



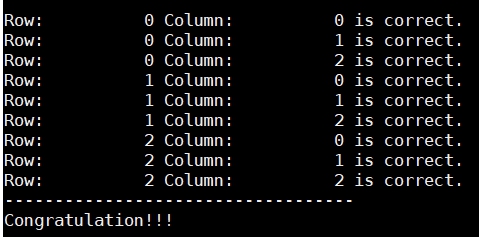
所以，我們判斷I=n, J=0時，fin就會拉起來。

2. Ncverilog Simulation Result:

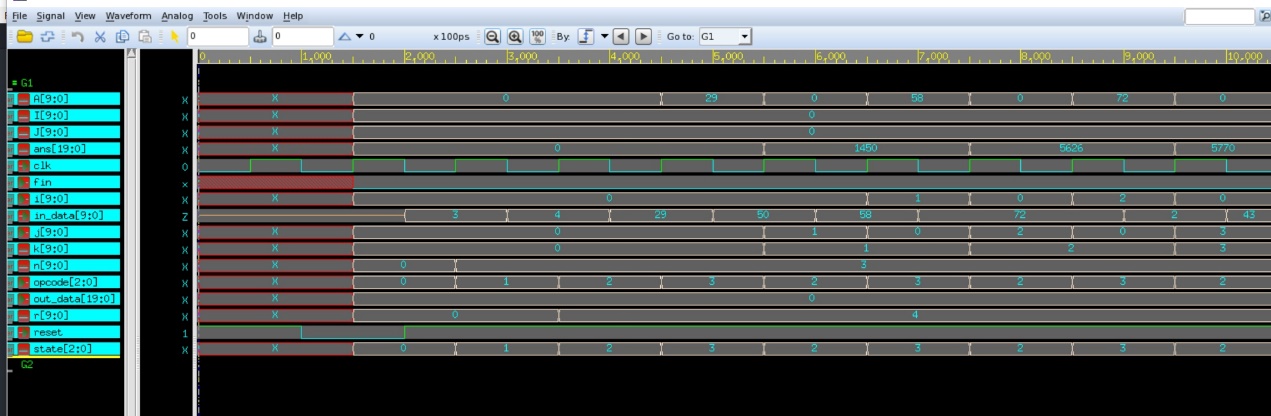
• Before Design Vision synthesis:



• After Design Vision synthesis:

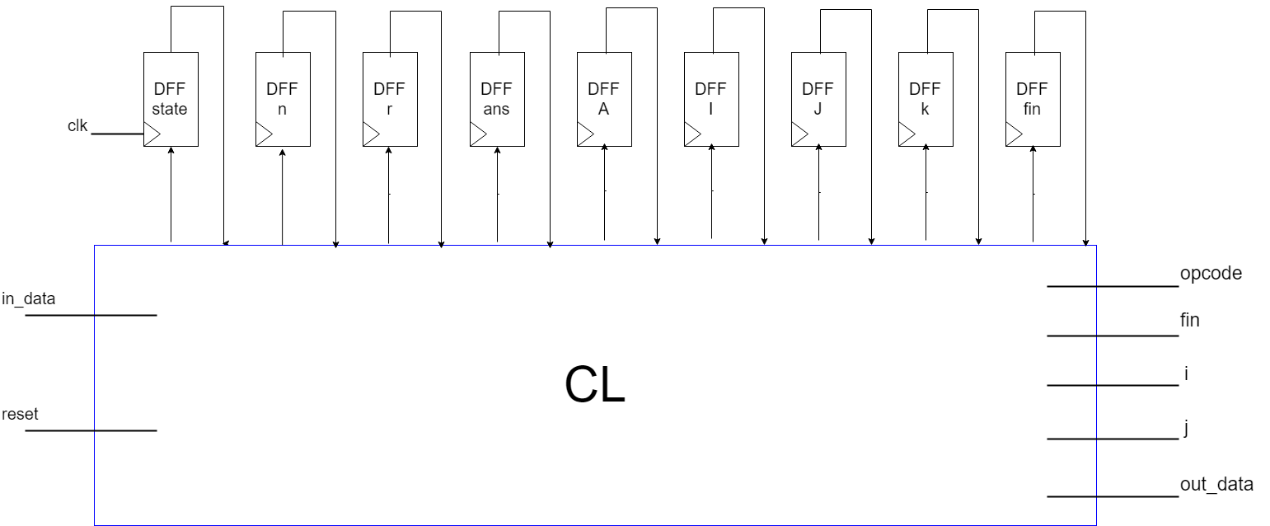


3. nWave Result:

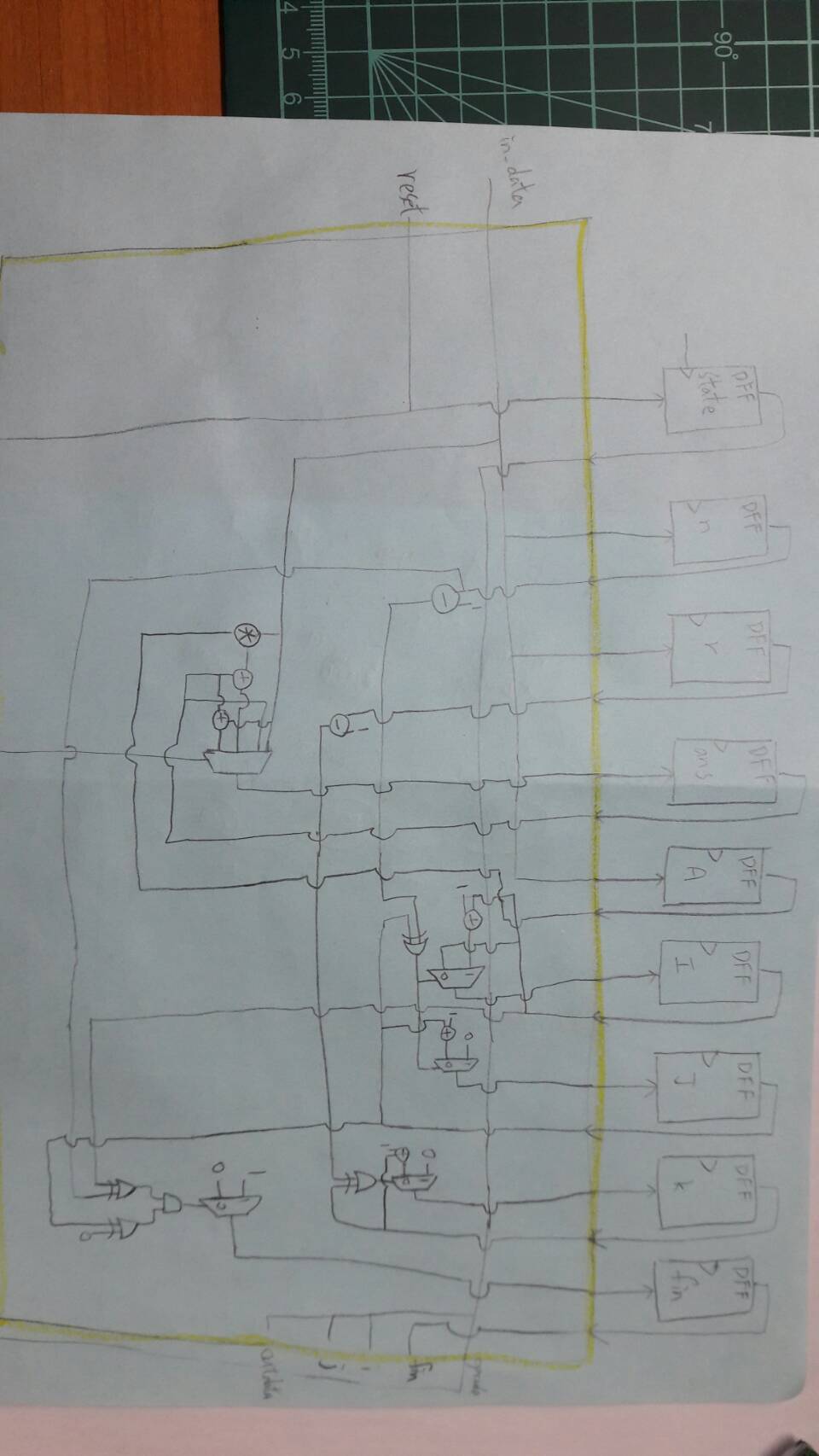


原本我以為沒有reset而且連clk都是整片綠的，後來才發現原來要放大，不然他預設好像就蠻緊密的。

4. Block Diagram:



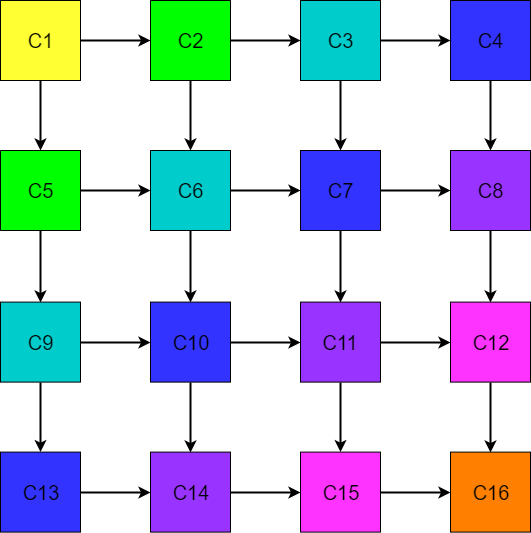
CL裡面如下圖，其中裡面有一些小的判斷，來決定I跟J的值，fin則是用一個mux來看他要為1還是0，A會拿來跟in\_data相減跟相加。



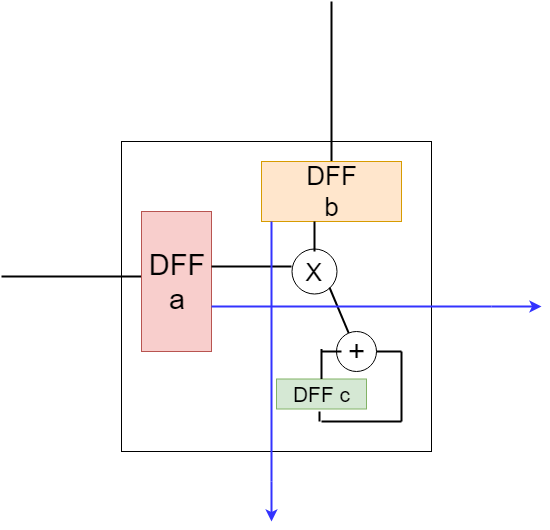
5. Problem encountered and discussion:

* 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。