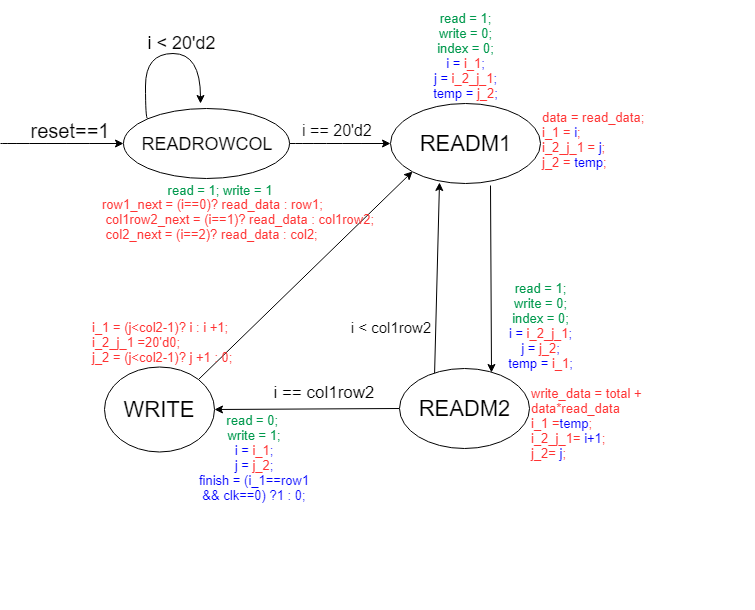
**Lab4: Matrix Multiplication**

# State Transition Graph



綠色的字為直接用assign設定，沒有出現在always block裡。

## Reason

**READROWCOL：**將read、write都設為1，根據i值的不同，可以分別得到矩陣的行數、列數。讀取完畢（i==2），到下一個state（READM1）

**READM1：**read為1，write為0，index為0，i = i\_1是目前要計算的答案在新矩陣中的行數，j = i\_2\_j\_1是目前相乘的第幾個數字（=目前相乘的Matrix 1的I = Matrix 2的j）。暫時把（j\_2）目前要計算的答案在新矩陣中的列數存到temp裡面。

得到read\_data後存在product內，product經過DFF變成data。

**READM2：**read為1，write為0，index為1，i = i\_2\_j\_1是目前相乘的第幾個數字（=目前相乘的Matrix 1的I = Matrix 2的j）。j= j\_2是目前要計算的答案在新矩陣中的列數。暫時把（i\_1）目前要計算的答案在新矩陣中的行數存到temp裡面。

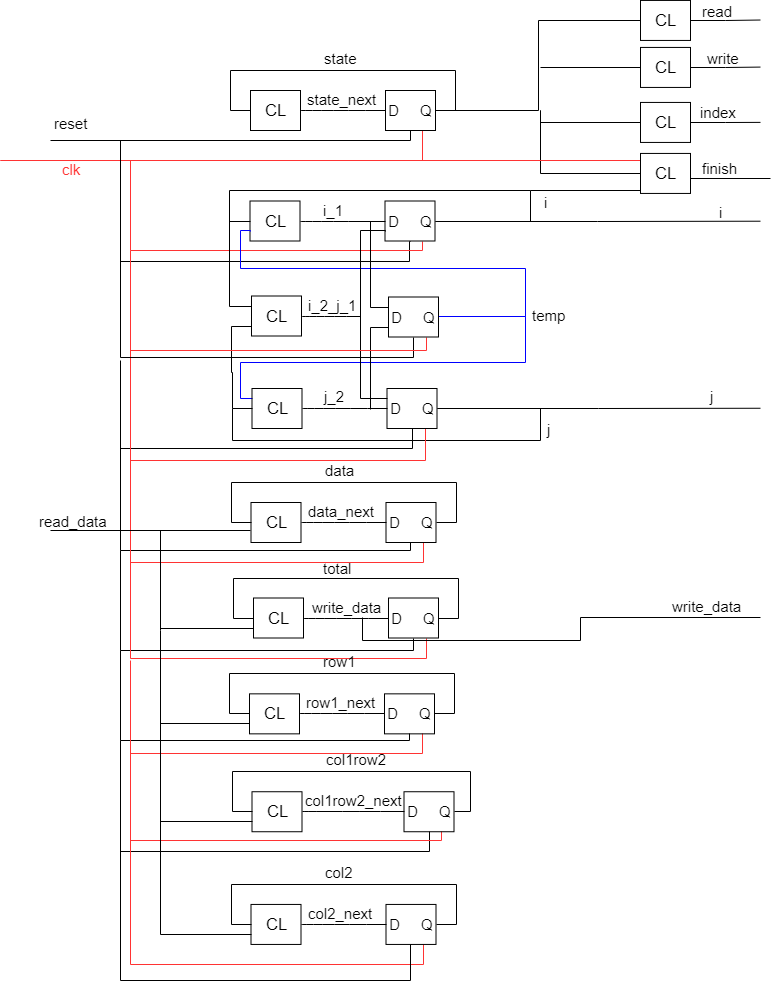
write\_data = total + data\*read\_data 得到read\_data後與M1得到的data相乘並加到write\_data中。然後i（之後會存到i\_2\_j\_1）要+1，如果i\_2\_j\_1到達邊界（i==col1row2）代表該位置的答案計算完成，到下一個state（WRITE）

**WRITE**：read = 0，write = 1（寫入資料），i = i\_1是目前要計算的答案在新矩陣中的行數，j = j\_2是目前要計算的答案在新矩陣中的列數。j\_2要+1（計算下個位置的答案），如果到達邊界（j==col2-1），就計算下一行（i\_1 = i+1），j\_2歸零。當計算到最後一個答案就立起finish。

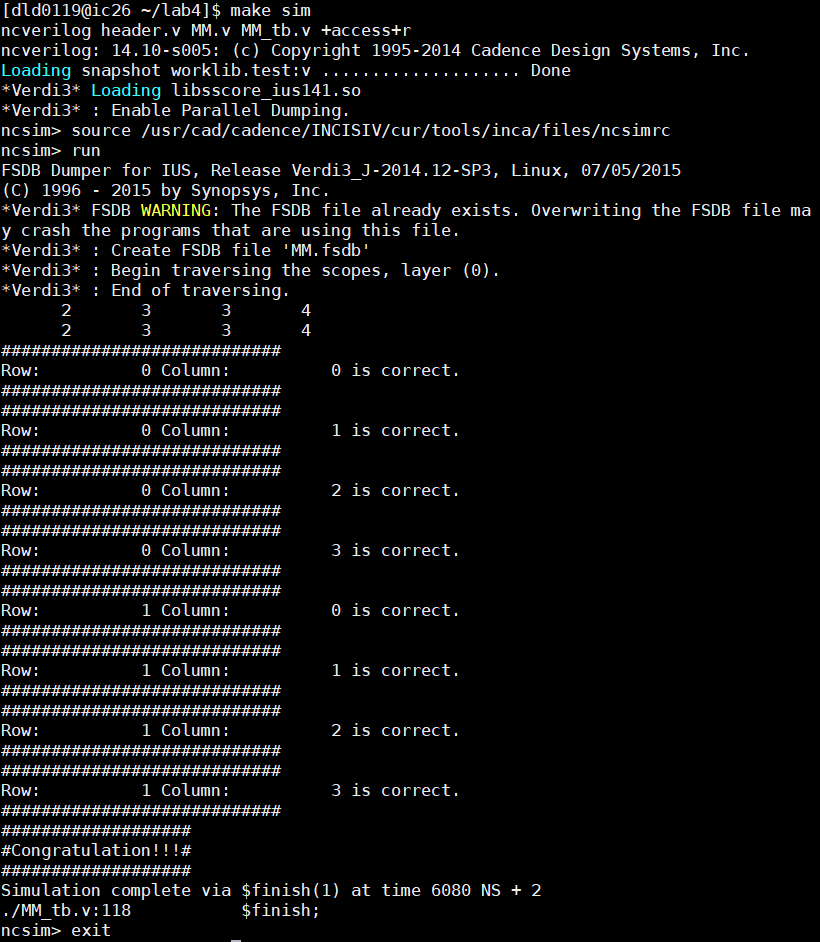
*Ex 1: i\_1 = 1, j\_2 = 0, i\_1\_j\_2 =2*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A00 | A01 | A02 |  | B00 | B01 |  | C00 | C01 |  | |  |
| A10 | A11 | A12 | X | B10 | B11 | = | C10 | C11 |  | |  |
| A20 | A21 | A22 | B20 | B21 | C20 | C21 |  | |  |
| A30 | A31 | A32 |  |  |  |  | C30 | C31 |  | |  |
| *read = 1 write = 0*  *index = 0 i = i\_1 j = i\_2\_j\_1 temp = i\_2* | | | *read = 1 write = 0*  *index = 1 i = i\_2\_j\_1 j = j\_2 temp = i\_1* | | | | *read = 0 write = 1 i = i\_1 j = j\_2* | | |
|
|
|

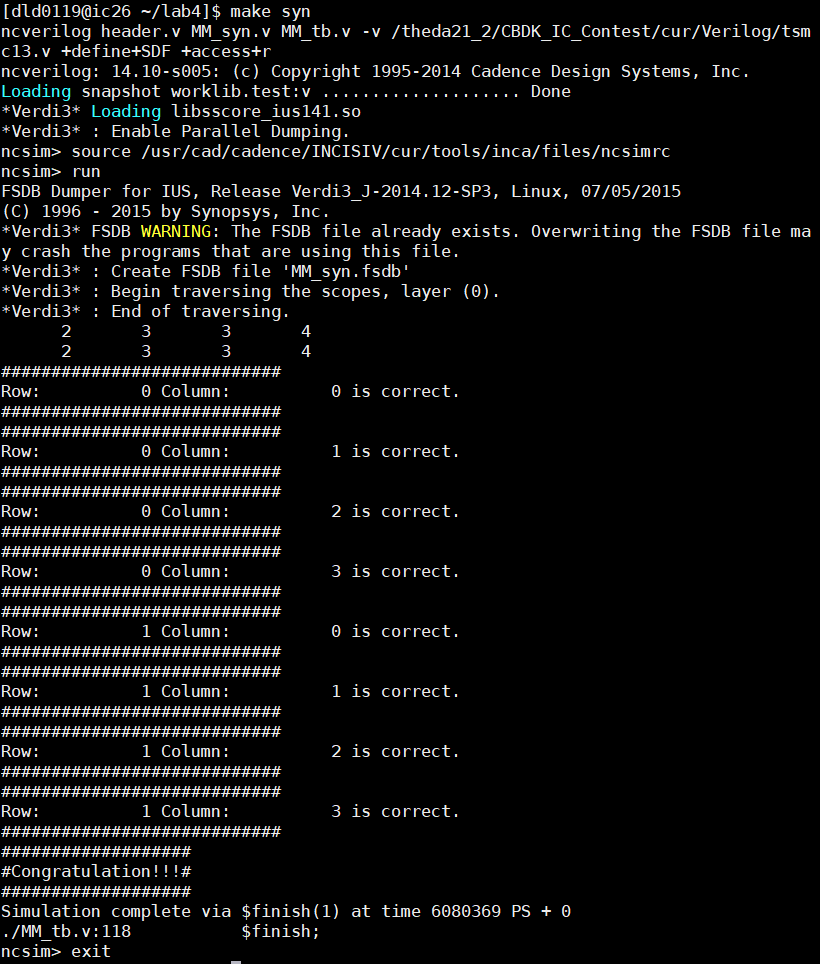
# Block Diagram



# ncverilog simulation (sim)



# ncverilog simulation (syn)



# Discussion

## Variables

起初對於variables的關係要怎麼寫覺得很混亂，後來整理出以下幾個原則：

* 變數分為三種：要flip-flop的N/next組（一定要是register）、P/prev組（不一定），和只跟combinational logic有關的C組（型別為wire）
* 在always@(posedge clk)begin 裡面給P組賦值（用<=），通常右邊是對應的N組（如果是reset就initialize）
* 在always@(\*)begin case(state)裡給N組賦值（用=），右邊是P組的變數們。
* 最後assign C組變數們的function，右邊是P組的變數

另外data、total、read\_data、write\_data等等沒有declare成signed會出錯。

## States

還有一個一開始沒搞懂的重要概念，是關於處理state的流程。現在的理解是這樣的：

1. 設定好next\_state
2. 經過posedge state <= next\_state
3. Top module (tb) 根據state給值
4. 根據之前的state和環境對得到的input做處理，設定下一個環境

最後一項是我們在case(state)裡面要寫的東西，例如說在READM1裡我們要做的事情就是對READM1的read\_data做處理，並設定要給tb的READM2的環境（N組）。

而在READM2後，我們要考慮的下一個state有兩種可能（READM1和WRITE），因此會有很多條件判斷，可以選擇在case(state)裡面判斷（根據P決定N），或是在進入DFF的時候判斷（根據N決定P）。

在WRITE state中，當j已經到達col的邊界的時候，要歸零並且讓i+1（類似進位的概念）。這個判斷其實也可以拆成另一個state來做（這樣finish也比較好設計），可是我不喜歡。

## Finish

覺得時間順序很難掌控，至今還沒有很懂要怎麼確保資料寫入之後才立finish，都是最後一筆出錯，然後看nWave慢慢修的。據說比較保險的做法是延遲一個clock cycle才立finish。

## tb bug

之前回報給助教的bug，對比過新舊tb之後，也獲得新的啟示。就是假設 A = 2, B = X（未知），則A==B不成立，A!=B也不成立。換句話說，!(A==B) 和 A!=B不一樣。

## Time Violation

一開始寫的版本有latch的問題（等號左邊和右邊有相同的變數），sim可以過，但是dv syn過不了，把latch改成乾淨的flip-flop之後就完全沒有time violation問題了，也不用去改clk。