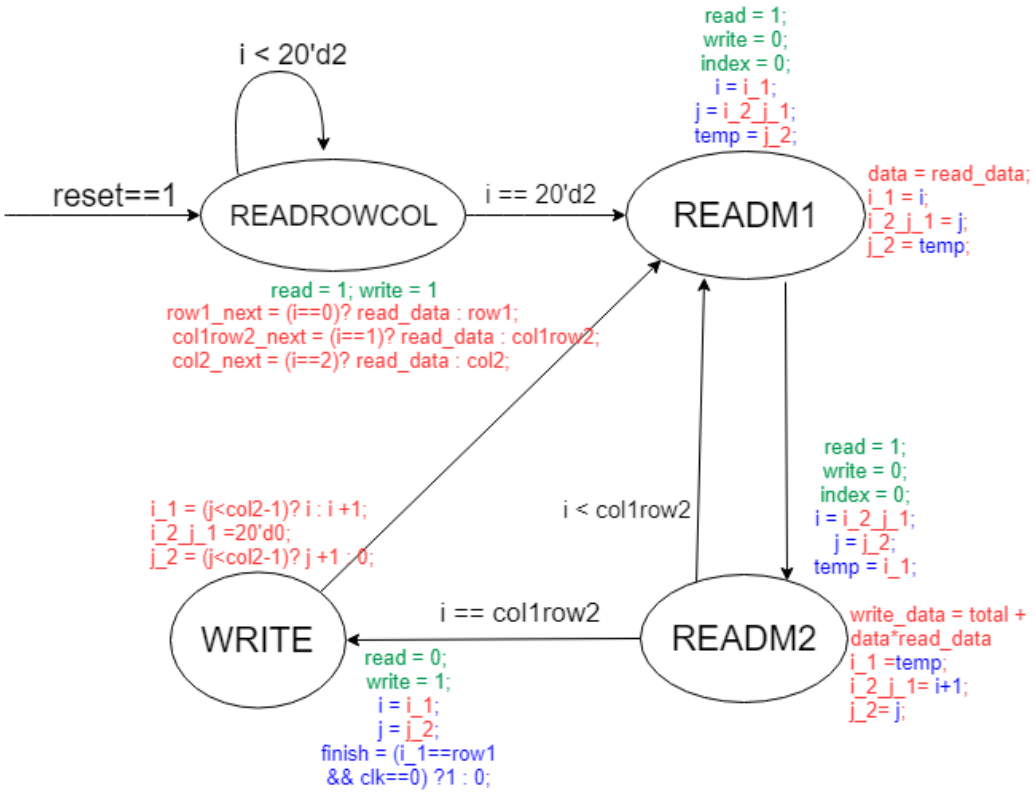
Lab4: Matrix Multiplication

1) 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

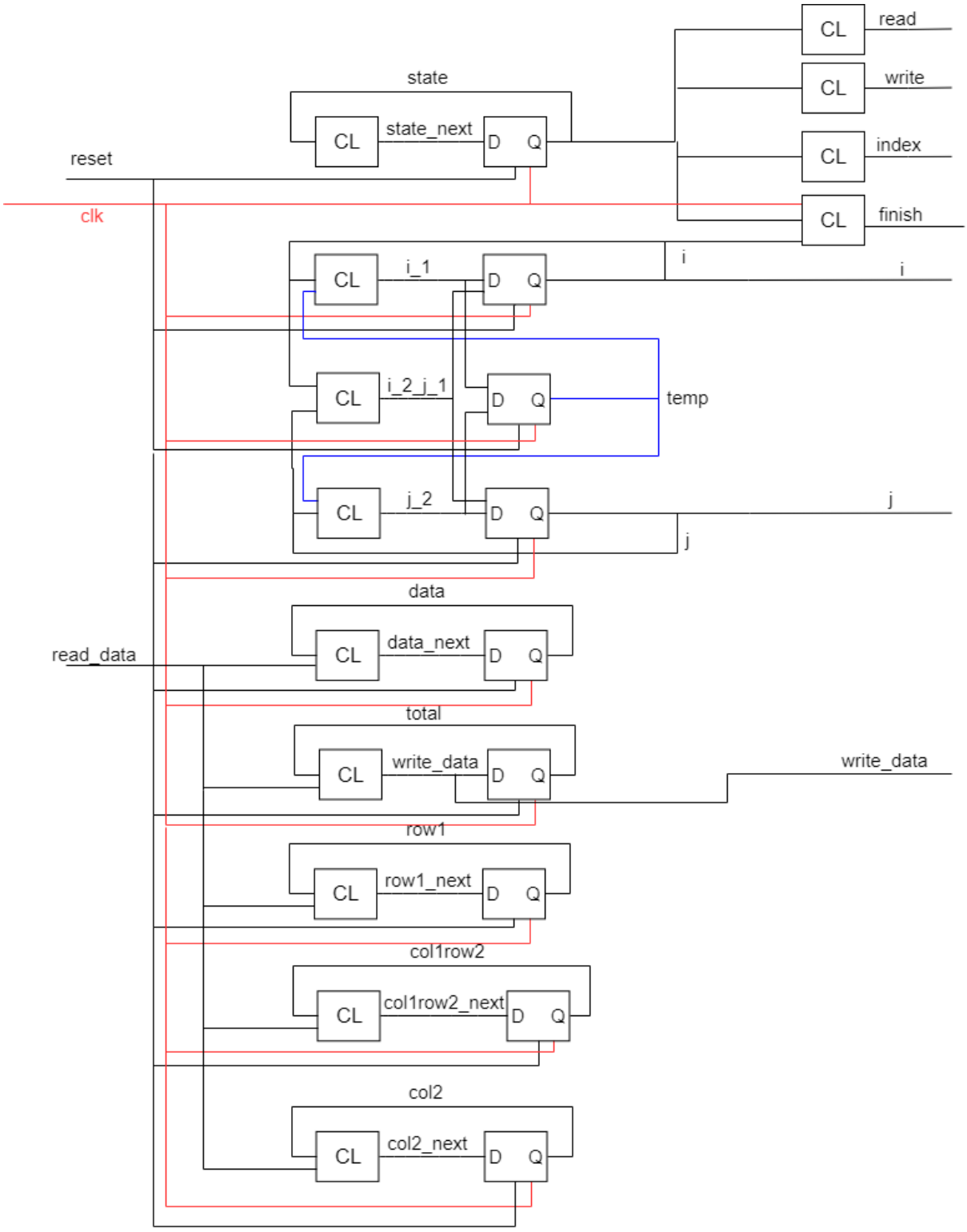
$$\begin{matrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \\ A_{30} & A_{31} & A_{32} \end{matrix} \quad X \quad \begin{matrix} B_{00} & B_{01} \\ B_{10} & B_{11} \\ B_{20} & B_{21} \end{matrix} \quad = \quad \begin{matrix} C_{00} & C_{01} \\ C_{10} & C_{11} \\ C_{20} & C_{21} \\ C_{30} & C_{31} \end{matrix}$$

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

2) Block Diagram



3) ncverilog simulation (sim)

```
[dld0119@ic26 ~/lab4]$ make sim
ncverilog header.v MM.v MM_tb.v +access+r
ncverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
Loading snapshot worklib.test:v ..................... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
FSDB Dumper for IUS, Release Verdi3_J-2014.12-SP3, Linux, 07/05/2015
(C) 1996 - 2015 by Synopsys, Inc.
*Verdi3* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file ma
y crash the programs that are using this file.
*Verdi3* : Create FSDB file 'MM.fsdb'
*Verdi3* : Begin traversing the scopes, layer (0).
*Verdi3* : End of traversing.
      2       3       3       4
      2       3       3       4
############################
Row:          0 Column:          0 is correct.
############################
############################
Row:          0 Column:          1 is correct.
############################
############################
Row:          0 Column:          2 is correct.
############################
############################
Row:          0 Column:          3 is correct.
############################
############################
Row:          1 Column:          0 is correct.
############################
############################
Row:          1 Column:          1 is correct.
############################
############################
Row:          1 Column:          2 is correct.
############################
############################
Row:          1 Column:          3 is correct.
############################
####################
#Congratulation!!!#
####################
Simulation complete via $finish(1) at time 6080 NS + 2
./MM_tb.v:118           $finish;
ncsim> exit
```

4) ncverilog simulation (syn)

```
[dld0119@ic26 ~/lab4]$ make syn
ncverilog header.v MM_syn.v MM_tb.v -v /theda21_2/CBDK_IC_Contest/cur/Verilog/tsm
c13.v +define+SDF +access+r
ncverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
Loading snapshot worklib.test:v ..................... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
FSDB Dumper for IUS, Release Verdi3_J-2014.12-SP3, Linux, 07/05/2015
(C) 1996 - 2015 by Synopsys, Inc.
*Verdi3* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file ma
y crash the programs that are using this file.
*Verdi3* : Create FSDB file 'MM_syn.fsdb'
*Verdi3* : Begin traversing the scopes, layer (0).
*Verdi3* : End of traversing.
      2       3       3       4
      2       3       3       4
############################
Row:          0 Column:          0 is correct.
############################
############################
Row:          0 Column:          1 is correct.
############################
############################
Row:          0 Column:          2 is correct.
############################
############################
Row:          0 Column:          3 is correct.
############################
############################
Row:          1 Column:          0 is correct.
############################
############################
Row:          1 Column:          1 is correct.
############################
############################
Row:          1 Column:          2 is correct.
############################
############################
Row:          1 Column:          3 is correct.
############################
###################
#Congratulation!!!#
###################
Simulation complete via $finish(1) at time 6080369 PS + 0
./MM_tb.v:118           $finish;
ncsim> exit
```

5) 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。