# project C 更新

```
`define INPUT_START 'h0000
`define WEIGHT_START 'h100000
`define OUTPUT_START 'h180000
`define SIM_END 'hFFFFF
`define SIM_END_CODE -32'd1
```

```
while (1)
begin
    #(`CYCLE)
    if (`mem_word(`SIM_END) == `SIM_END_CODE)
    begin
        break;
    end
end
$display("\nDone\n");
```

- 如果完成整層convolution，請對DRAM的‘hFFFFF位置寫-1，testbench才能判斷已經完成。
- 所有DRAM裡資料的起始值可以自己更改，但相應的HEX檔內的位置必須改成你自己設定的位置。
- 此次總共提供六種測資，測資的資訊在下兩張投影片，在testbench中判斷是否正確的方式如下圖，可自行更改想要測試多少筆資料，若對於測資有問題，煩請通知課程助教。

```
for (i = 0; i < num; i++)
begin
    if (`mem_word(`OUTPUT_START + i) !== GOLDEN[i])
    begin
        $display("DRAM[%8d] = %h, expect = %h", `OUTPUT_START + i, `mem_word(`OUTPUT_START + i), GOLDEN[i]);
        err = err + 1;
    end
    else
    begin
        $display("DRAM[%8d] = %h, pass", `OUTPUT_START + i, `mem_word(`OUTPUT_START + i));
    end
end
```

# Model 1 – 3x3 filter size

| layer | Input map | kernel size | filters | Output map |
|---|---|---|---|---|
| conv1 | 416 * 416 * 3 | 3 | 16 | 414 * 414 * 16 |
| conv2 | 104 * 104 * 32 | 3 | 64 | 102 * 102 * 64 |
| conv3 | 52 * 52 * 64 | 3 | 128 | 50 * 50 * 128 |
| conv4 | 13 * 13 * 512 | 3 | 1024 | 11 * 11 * 1024 |

* Because these layers are discontinuous, you need to read each layer individually.

# Model 1 – 3x3 filter size

| layer | Input map | kernel size | filters | Output map |
|---|---|---|---|---|
| conv1 | 416 * 416 * 3 | 5 | 16 | 412 * 412 * 16 |
| conv2 | 104 * 104 * 32 | 5 | 64 | 100 * 100 * 64 |
| conv3 | 52 * 52 * 64 | 5 | 128 | 48 * 48 * 128 |
| conv4 | 13 * 13 * 512 | 5 | 1024 | 9 * 9 * 1024 |

* Because these layers are discontinuous, you need to read each layer individually.

# DRAM module for RTL code

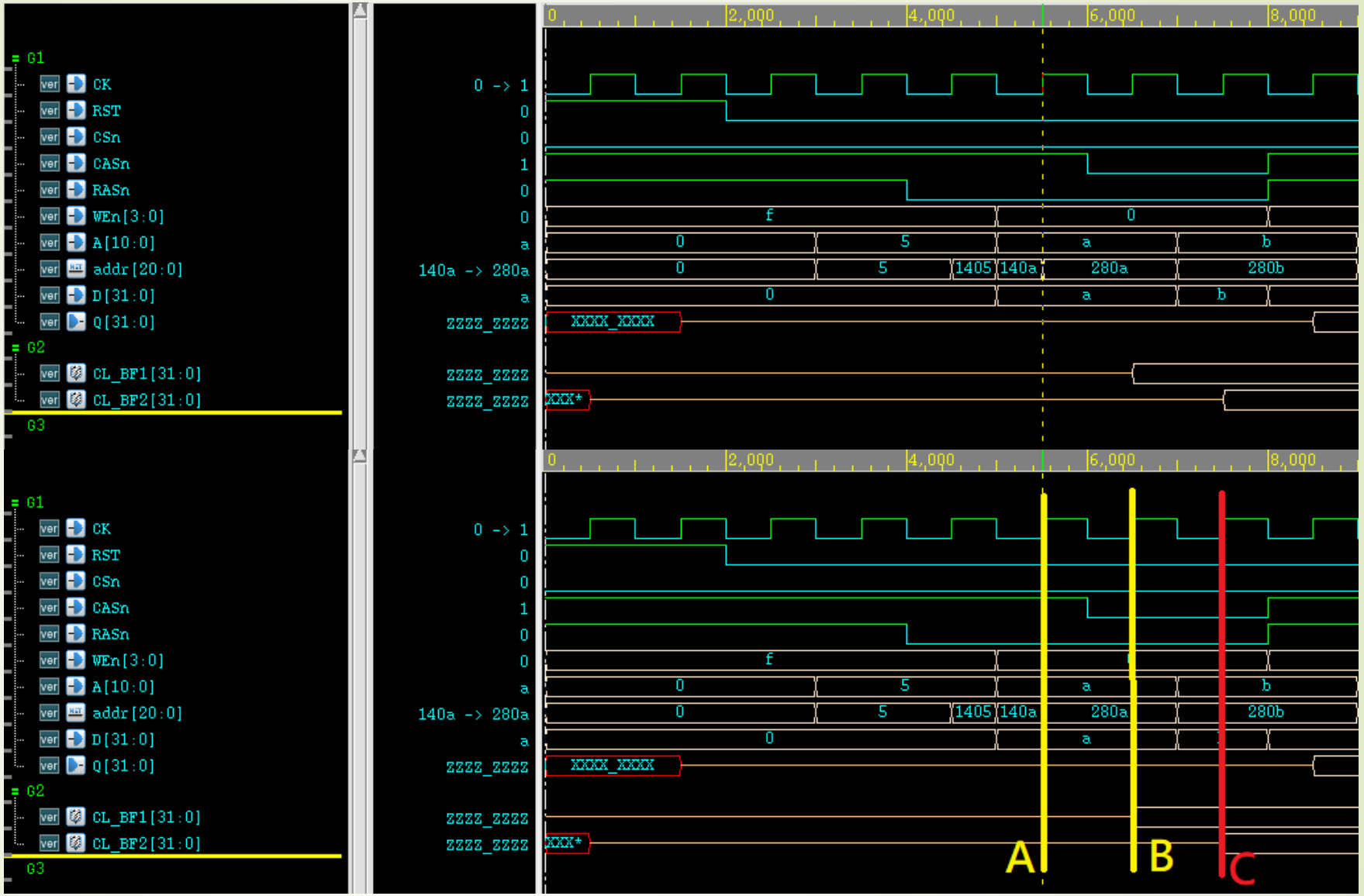
- RTL code 可以使用Verilog或SystemVerilog撰寫。
- RTL code所採用的環境可以是任何環境，包含NC-Verilog、Modelsim、Xilinx等Toolchain，DEMO時必須攜帶筆電使用自己環境呈現。
- 提供的DRAM_tb.sv和Makefile是使用NC-Verilog及Verdi環境，如需使用其他環境則必須修改DRAM_tb.sv程式中fsdbDump的部分。
- 下頁開始是介紹DRAM 的行為。

# DRAM (1)

| | | | | |
|---|---|---|---|---|
| DRAM | System signals | | | |
| | CK | input | 1 | System clock |
| | RST | input | 1 | System reset(active high) |
| | Memory ports | | | |
| | CSn | input | 1 | DRAM Chip Select (active low) |
| | Wen | input | 4 | DRAM Write Enable (active low) |
| | RASn | input | 1 | DRAM ROW Access strobe (active low) |
| | CASn | input | 1 | DRAM Column Access strobe (active low) |
| | A | input | 11 | DRAM Address input |
| | D | input | 32 | DRAM data input |
| | Q | output | 32 | DRAM data output |
| | Memory space | | | |
| | Memory_byte0 | reg | 8 | Size : [0:2097151] |
| | Memory_byte1 | reg | 8 | Size : [0:2097151] |
| | Memory_byte2 | reg | 8 | Size : [0:2097151] |
| | Memory_byte3 | reg | 8 | Size : [0:2097151] |

※ Row address is 11-bit and column address is 10-bit

# DRAM (2)

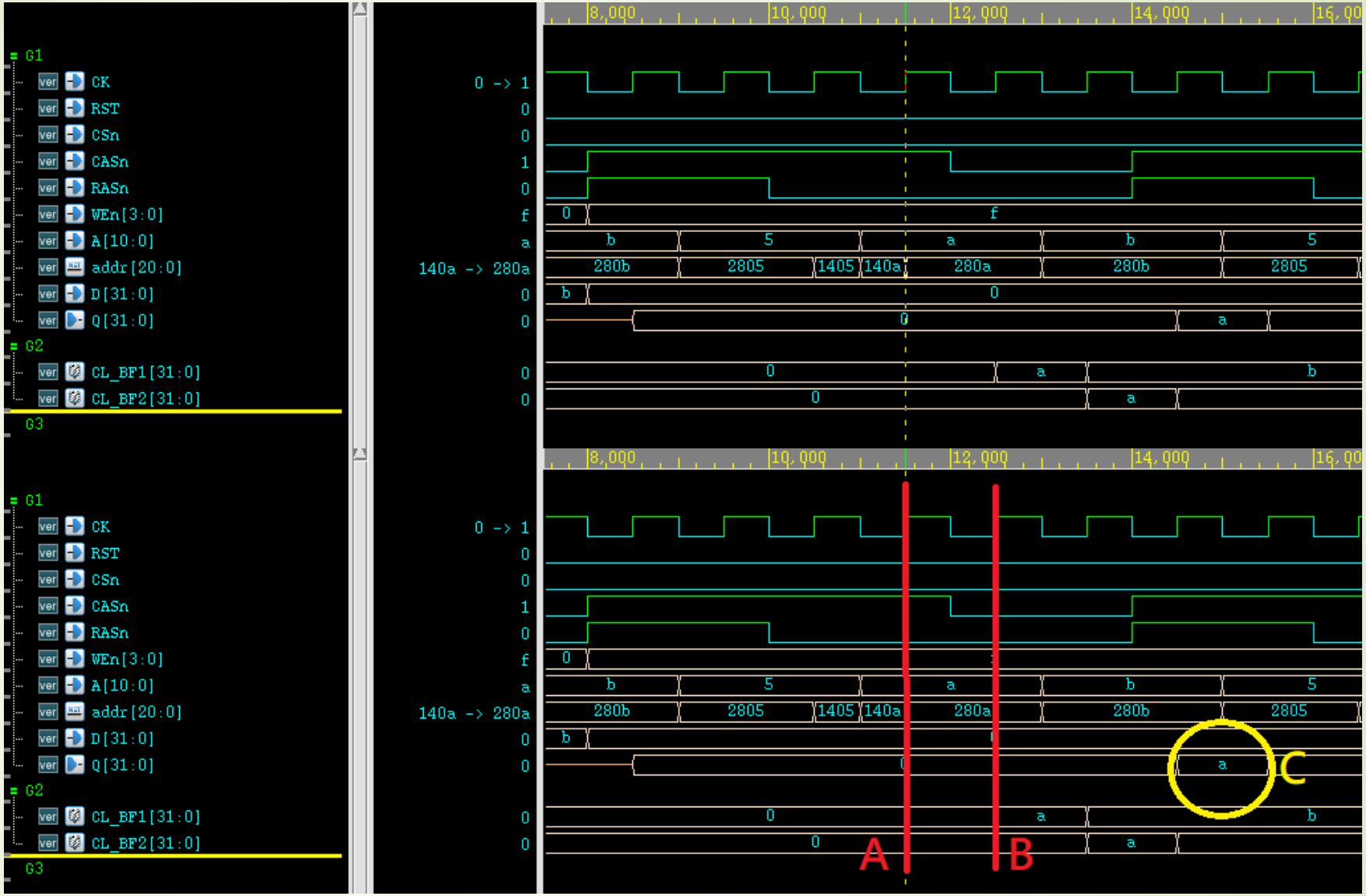



※ Write

A. RAS=0 -> Set Row (A should be ready)

B. CAS=0 -> Set Column(A, WEn and D should be ready)

# DRAM (3)




※ Read

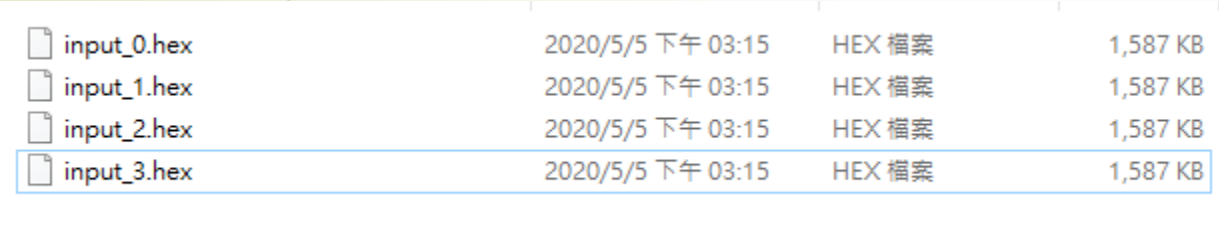
A. RAS=0 -> Set Row (A should be ready)

B. CAS=0 -> Set Column(A, WEn should be ready)

C. Latency : 2 cycle

# DRAM module for RTL code

- DRAM_INPUT資料夾內有4個檔案，會一起放入DRAM。




- 在top_tb.sv中會利用readmemh將資料讀入，下圖紅框中，因為此範例是在top_tb.sv宣告DRAM module，所以後方讀入的位置是這樣表示。如果你是在top.sv宣告DRAM module，則根據你的instance name會有不同寫法，綠色的註解為範例。

```
$readmemh({prog_path, "/input_0.hex"}, M1.Memory_byte0);
$readmemh({prog_path, "/input_1.hex"}, M1.Memory_byte1);
$readmemh({prog_path, "/input_2.hex"}, M1.Memory_byte2);
$readmemh({prog_path, "/input_3.hex"}, M1.Memory_byte3);

// $readmemh({prog_path, "/input_feature0.hex"}, top_1.DRAM_1.Memory_byte0);
// $readmemh({prog_path, "/input_feature1.hex"}, top_1.DRAM_1.Memory_byte1);
// $readmemh({prog_path, "/input_feature2.hex"}, top_1.DRAM_1.Memory_byte2);
// $readmemh({prog_path, "/input_feature3.hex"}, top_1.DRAM_1.Memory_byte3);
```

# DRAM module for RTL code

- Input feature的起始點為'h000000，Weight的起始點為'h100000。
- 下圖紅框讓我們可以直接在top_tb內存取DRAM的值，可以更容易Debug。

```
`define INPUT_START 'h0000
`define WEIGHT_START 'h100000

`define mem_word(addr){M1.Memory_byte3[addr],M1.Memory_byte2[addr],M1.Memory_byte1[addr],M1.Memory_byte0[addr]}
```

- 下圖會印出前60個的Input feature值，而Input feature總共有416*416*3，Weight則會有3*3*3*16，每個值皆為32bit。

```
initial begin
    #(`CYCLE*50000)
    for(i=0;i<60;i++)   //print first 60 value
    begin
        $display("%h",`mem_word(`INPUT_START + i));
    end
    $finish;
end
```

# DRAM module for RTL code

- DRAM裡使用4個reg array，因為實際上DRAM是可以針對特定的Byte去做存取，Wen訊號也因此為4bit。
- 但在作業中我們所使用的資料為32bit，存放在4個reg array中的同一個位置裡，所以只要同時操作4個Byte存或寫，並不需要將32bit分成4個Byte操作，具體來說就是存值時Wen=0，讀值時Wen=F。

| | | | | |
|---|---|---|---|---|
| DRAM | System signals | | | |
| | CK | input | 1 | System clock |
| | RST | input | 1 | System reset(active high) |
| | Memory ports | | | |
| | CSn | input | 1 | DRAM Chip Select (active low) |
| | Wen | input | 4 | DRAM Write Enable (active low) |
| | RASn | input | 1 | DRAM ROW Access strobe (active low) |
| | CASn | input | 1 | DRAM Column Access strobe (active low) |
| | A | input | 11 | DRAM Address input |
| | D | input | 32 | DRAM data input |
| | Q | output | 32 | DRAM data output |
| | Memory space | | | |
| | Memory_byte0 | reg | 8 | Size : [0:2097151] |
| | Memory_byte1 | reg | 8 | Size : [0:2097151] |
| | Memory_byte2 | reg | 8 | Size : [0:2097151] |
| | Memory_byte3 | reg | 8 | Size : [0:2097151] |

※ Row address is 11-bit and column address is 10-bit