AOC Lab3 MAC

Advisor: CCTsai

TA：M16111048 湯詠涵

Email：course.aislab@gmail.com

Eyeriss

- Reference paper :

Y. -H. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, Jan. 2017, doi: 10.1109/JSSC.2016.2616357.

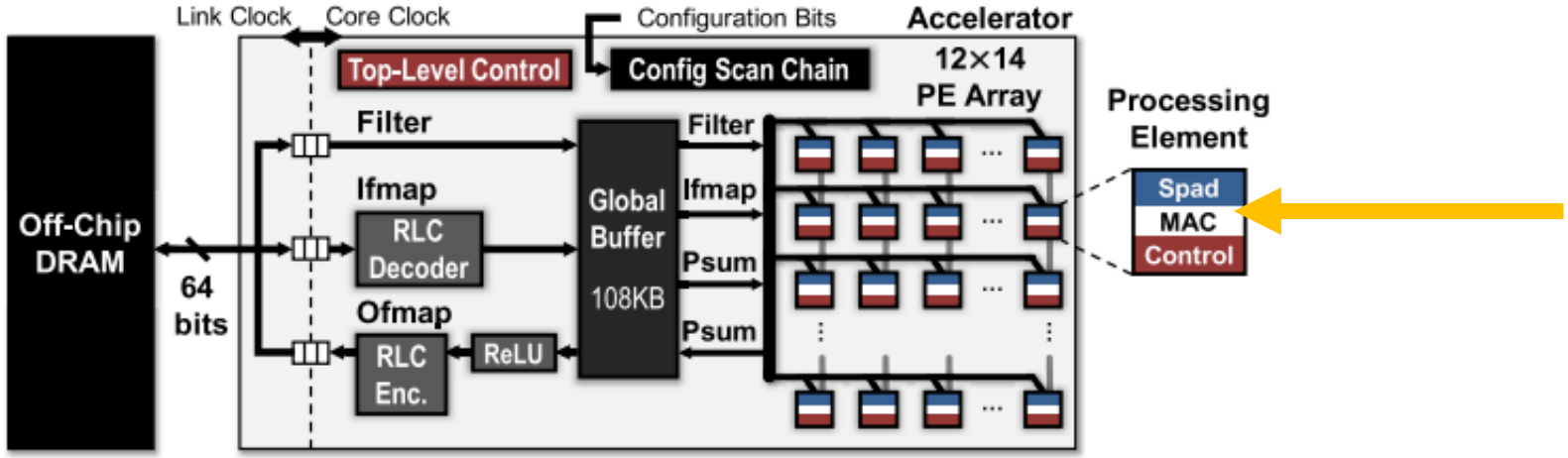
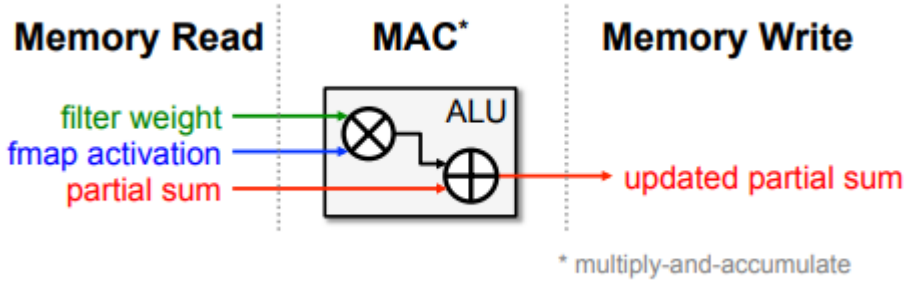


Fig. 2. Eyeriss system architecture.

MAC Introduction

- Full Name : Multiply and Accumulator
- Compute the product of two numbers and add that product to an accumulator.

$$a \leftarrow a + (b \times c)$$

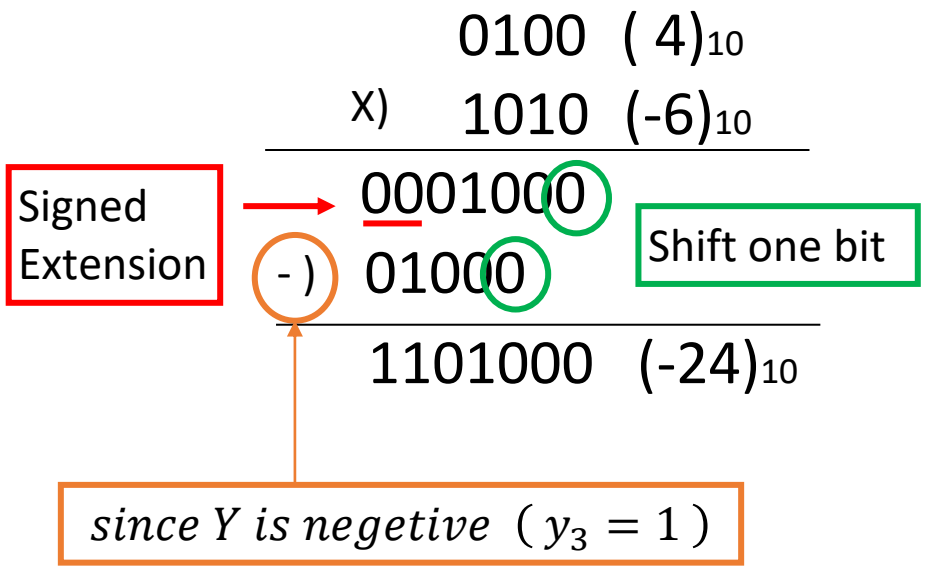


Metrics	LeNet 5	AlexNet
Top-5 error†	n/a	16.4
Top-5 error (single crop)†	n/a	19.8
Input Size	28×28	227×227
# of CONV Layers	2	5
Depth in # of CONV Layers	2	5
Filter Sizes	5	3,5,11
# of Channels	1, 20	3-256
# of Filters	20, 50	96-384
Stride	1	1,4
Weights	2.6k	2.3M
MACs	283k	666M
# of FC Layers	2	3
Filter Sizes	1,4	1,6
# of Channels	50, 500	256-4096
# of Filters	10, 500	1000-4096
Weights	58k	58.6M
MACs	58k	58.6M
Total Weights	60k	61M
Total MACs	341k	724M
Pretrained Model Website	[56]‡	[57, 58]

2896M memory accesses required

Signed Binary Multiplication -- Rebertson Algorithm

Example : multiplicand X = 0100 ,
multiplier Y = 1010 , $Y = y3\ y2\ y1\ y0$, $Y = -2^3 + 2^1$



Value of 4-bit signed number

$$Y = \begin{cases} \sum_{i=0}^{3} 2^i y_i & , if\ Y\ is\ positive\ or\ zero\ (\ y_3 = 0\ ) \\ -2^3 + \sum_{i=0}^{2} 2^i y_i\ , & if\ Y\ is\ negetive\ (\ y_3 = 1\ ) \end{cases}$$

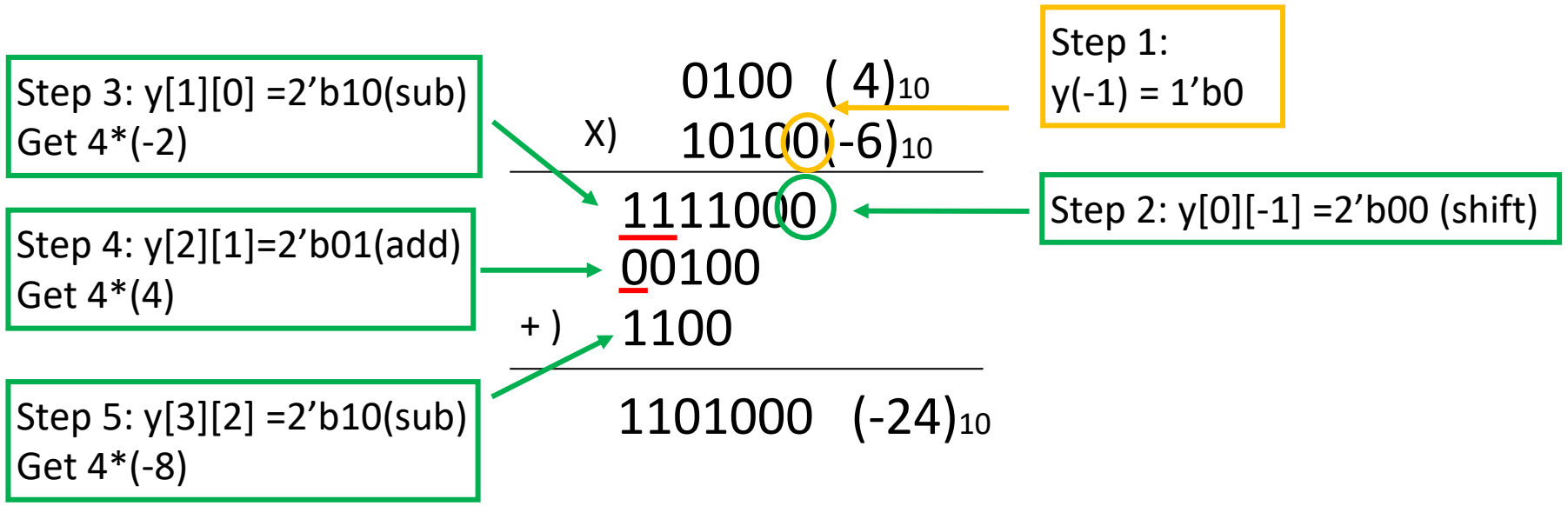
Example : $Y_1 = (1010)_2 = (-6)_{10} = (-2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0)$
$Y_2 = (0110)_2 = (+6)_{10} = (\ 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0)$

Product of 2 signed numbers

$$\text{product} = \begin{cases} (\sum_{i=0}^{3} 2^i y_i) \times X & , if\ Y\ is\ positive\ or\ zero\ (\ y_3 = 0\ ) \\ (-2^3 + \sum_{i=0}^{2} 2^i y_i) \times X\ , & if\ Y\ is\ negetive\ (\ y_3 = 1\ ) \end{cases}$$

Booth's Algorithm

- multiplicand X : 0100 / two's complement (-X) : 1100
- Analysis multiplier Y : 1010
- Seen 1010 as 1000 + 0010 , $(-6)_{10} = (-8)_{10} + (+2)_{10}$
- Seen 0010 as 1110 + 0100 , $(+2)_{10} = (-2)_{10} + (+4)_{10}$
- $(-6)_{10} = (-2)_{10} + (4)_{10} + (-8)_{10}$



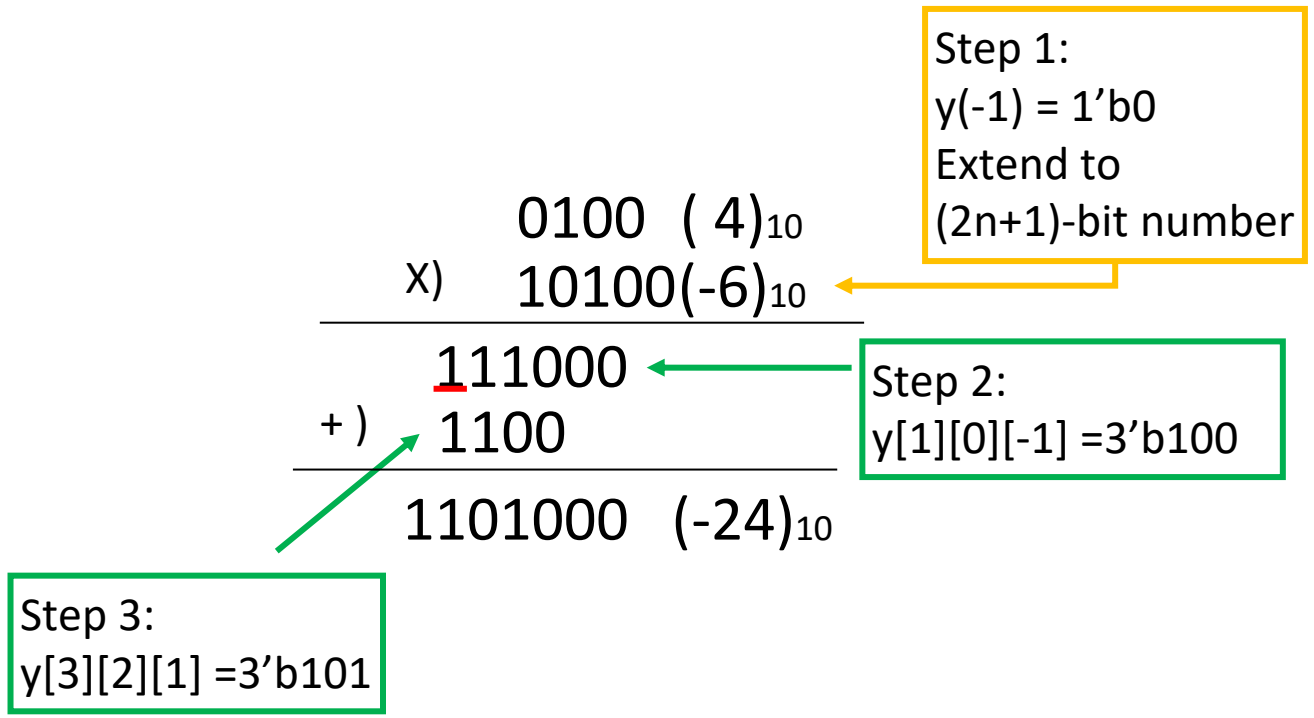
Multiplier	Operation
00,11	shift
01	Add X
10	Sub X / Add (-X)

Modified Booth's Algorithm

- Derivated from Booth's Algorithm

Recalled booth's algorithm

Multiplier	Operation
00,11	Shift
01	Add
10	Sub



Multiplier	X(multiplicand)	operation
000	Shift	Shift + shift
001	X	Shift + Add
010	2X-X = X	Add + Sub
011	2X	Add + Shift
100	-2X	Sub + Shift
101	-2X+X	Sub + Add
110	-X	Shift + Sub
111	shift	Shift + Shift

Hint

- **Shift operator**: << , <<< , >>, >>>

  << , >> : logical

  <<< , >>> : arithmetic (can do signed extension)

- **Value rearrangement** :Y' = {{3{Y[3]}}, 1'b1} ;

  Y = 4'1010 -> Y' = 4'1111

- **Conditional Expression** : case, if

case treats all inputs the same

```
case(sel)
  2'b00: out = in1;
  2'b01: out = in2;
  2'b10: out = in3;
  2'b11: out = in4;
endcase
```

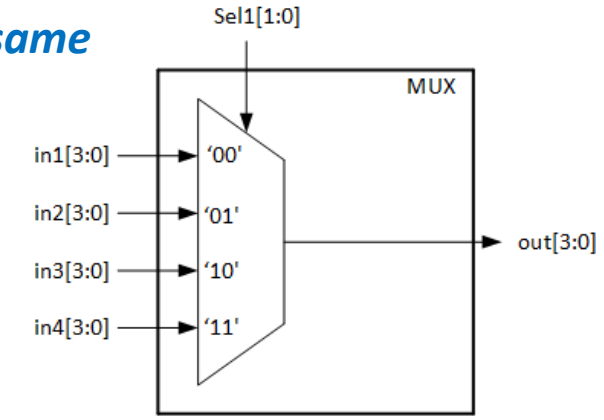


Fig 2

If-else leads to different delays for different inputs

```
if(sel == 2'b00) // sel1 = 1
   out = in1;
else if (sel == 2'b01) // sel2 = 1
   out = in2;
else if (sel == 2'b10) //sel3 =1
  out = in3;
else
  out = in4;
```

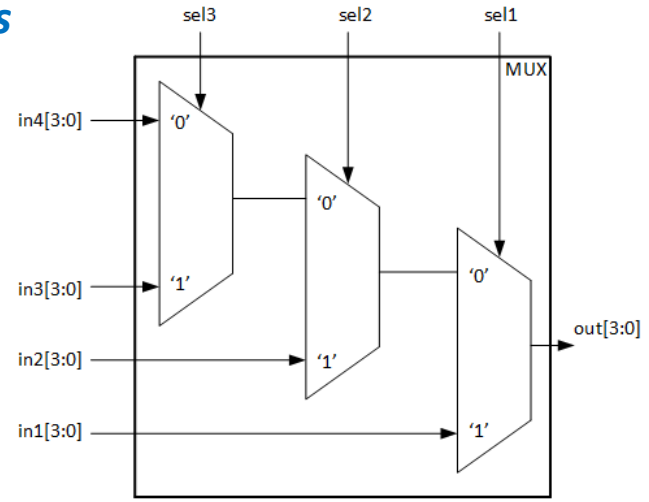
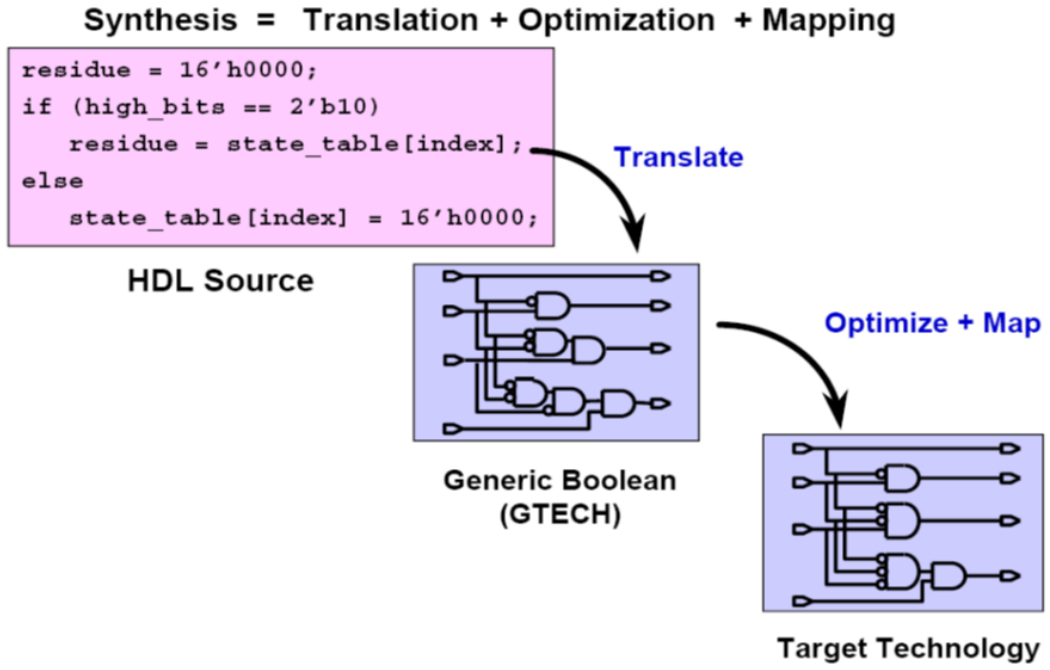


Fig 1

Synthesizable Verilog

- **Synthesizable** : be able to **be represented in hardware logic gates**
- non-synthesizable : cannot be translated to hardware



- 圖片來源
- http://www.icfgblog.com/index.php/EDA/50.html
- https://link.springer.com/content/pdf/bbm:978-81-322-2791-5/1.pdf

The list of synthesizable and non-synthesizable Verilog constructs is tabu-lated in the following Table

Verilog Constructs	Used for	Synthesizable construct	Non-Synthesizable Construct
module	The code inside the module and the endmodule consists of the declarations and functionality of the design	Yes	No
Instantiation	If the module is synthesizable then the instantiation is also synthesizable	Yes	No
initial	Used in the test benches	No	Yes
always	Procedural block with the reg type assignment on LHS side. The block is sensitive to the events	Yes	No
assign	Continuous assignment with wire data type for modeling the combinational logic	Yes	No
primitives	UDP's are non-synthesizable whereas other Verilog primitives are synthesizable	Yes	No
force and release	These are used in test benches and non-synthesizable	No	Yes
delays	Used in the test benches and synthesis tool ignores the delays	No	Yes
fork and join	Used during simulation	No	Yes
ports	Used to indicate the direction, input, output and inout. The input is used at the top module	Yes	No
parameter	Used to make the design more generic	Yes	No
time	Not supported for the synthesis	No	Yes

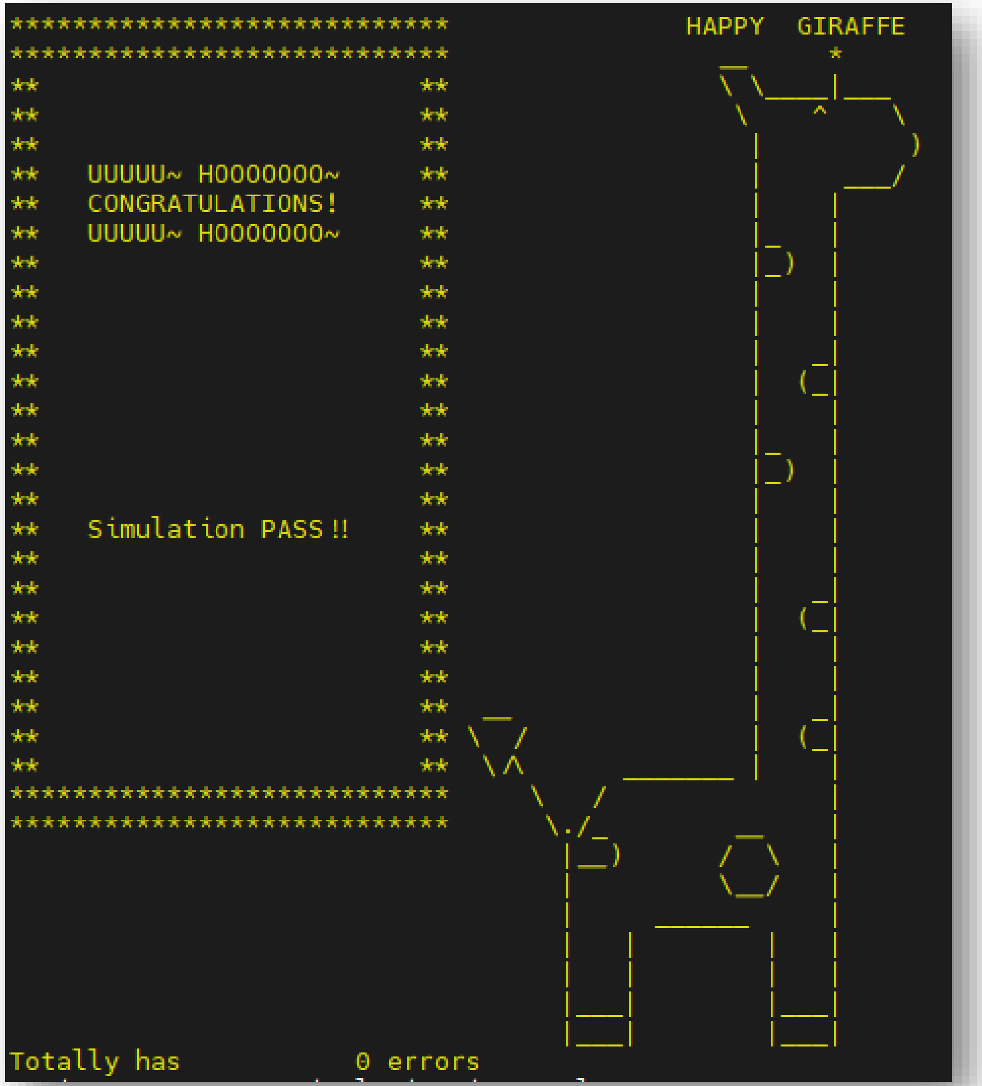
(continued)

(continued)

real	Not supported for synthesis	No	Yes
functions and task	Both are synthesizable. Provided that the task does not have the timing constructs	Yes	No
loop	The for loop is synthesizable and used for the multiple iterations.	Yes	No
Verilog Operators	Used for arithmetic, bitwise, unary, logical, relational etc are synthesizable	Yes	No
Blocking and non-blocking assignments	Used to describe the combinational and sequential design functionality respectively	Yes	No
if-else, case, casex, casez	These are used to describe the design functionality depending on the priority and parallel hardware requirements	Yes	No
Compiler directives (‘ifdef,‘undef, ‘define)	Used during synthesis	Yes	No
Bits and part select	It is synthesizable and used for the bit or part select	Yes	No

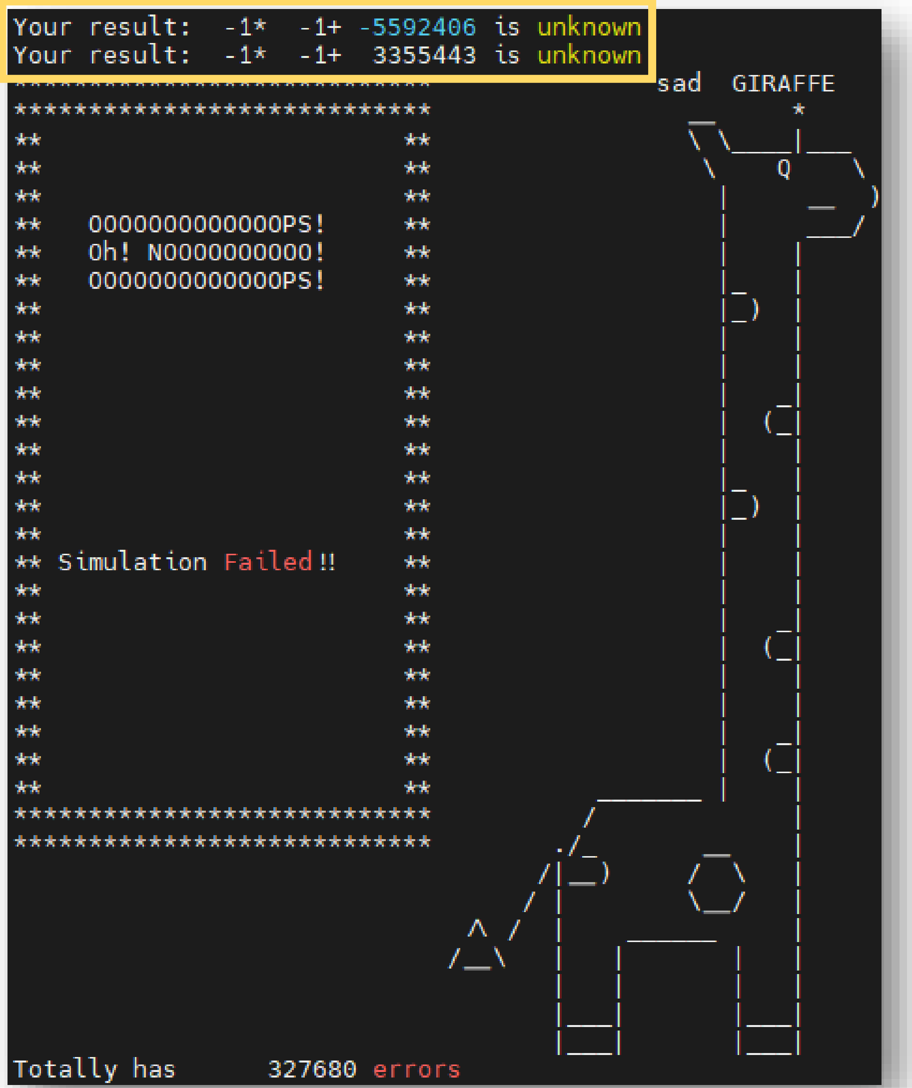
Simulation Result

WithOut Errors

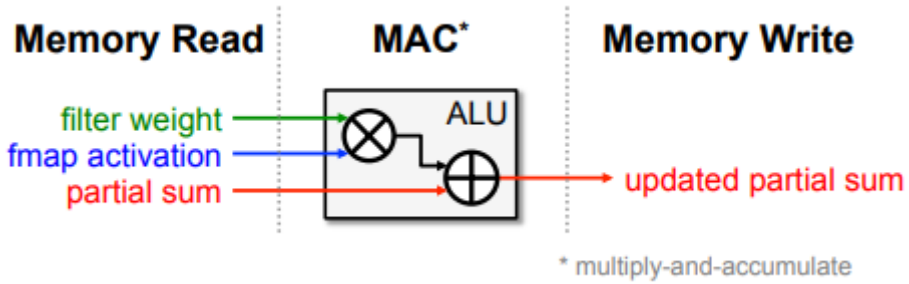


Incorrect computation

With Errors



Specification



- **Implement rules:**
  - By SystemVerilog or Verilog
  - Synthesizable Combinational Circuit
  - **(** $signed(ifmap) * $signed(filter)**)** is not allow
  - Choose one algorithm
  - No Plagiarism
- **Input**
- 8-bit ifmap
- 8-bit filter

Reduce Data movement, Storage cost ,
Energy and time per MAC operation,
Energy per memory bandwidth

- 24-bit partial sum
- **Output**
- 24-bit updated partial sum

Enough bandwidth for accumulation

SystemVerilog : `include "MAC**.sv**"
Verilog : `include "MAC**.v**"

MAC.sv MAC_tb.sv

```
1   `include "MAC.sv"
2
3   module MAC_tb;
4       reg signed [7:0]   a;
5       reg signed [7:0]   b;
6       reg signed [23:0]  c;
7       wire [23:0] result;
8
```

Report

- Screenshot of simulation result
- (10%)Introduce your MAC operation (algorithm)
- (10%)Waveform explanation with two examples
- (10%)Introduce the function of each module

StudentID_lab3.zip

StudentID_lab3

StudentID_Name_report. Pdf (30%)

MAC_tb. sv

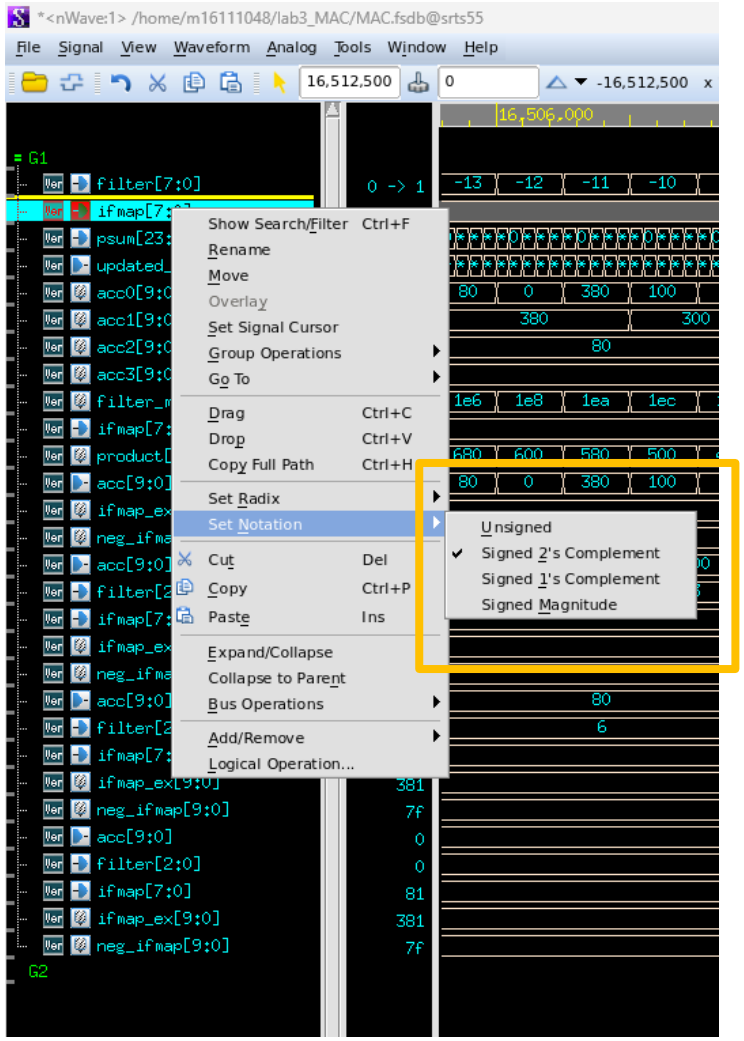
MAC. sv or MAC.v
(All pass 70% , each error minus 3%)

其他自動生成的檔案

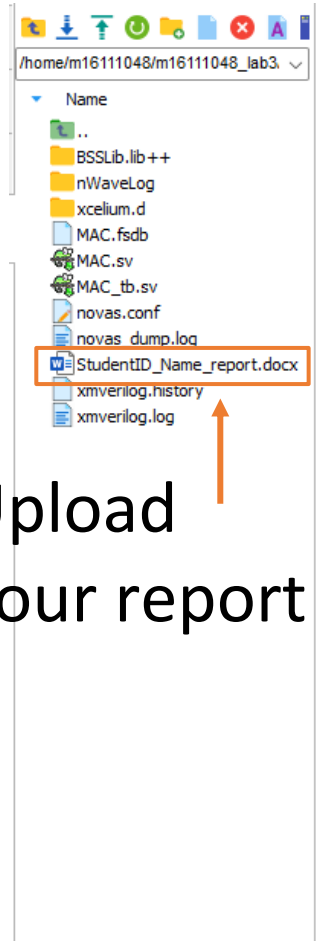
Office Hour : 3 / 23 14.~16.
3 / 30 14.~16.
room 電機館 92508

Deadline : 4/6 (Thu.) 23.59

Value Representation Format : signed 2's complement

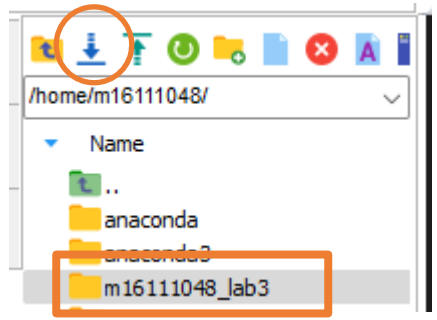


Finished Page and Files

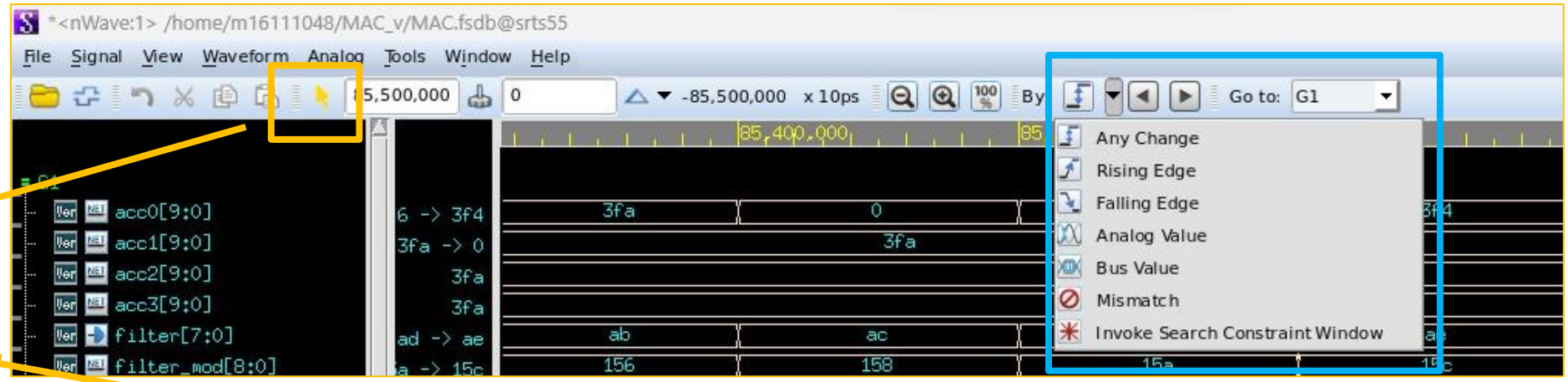


Upload your report

Download your file

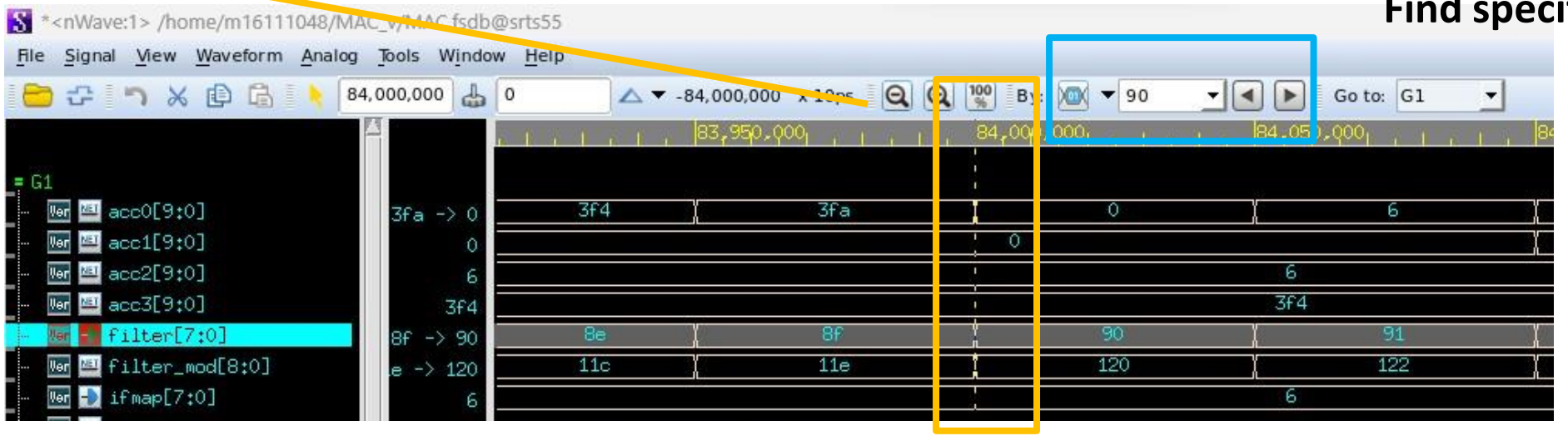


Debugging Tips



Put cursor in the middle of your page

Find specific value



- *Thank you for listening!*