

國立中山大學電機工程學系

實用數位系統設計

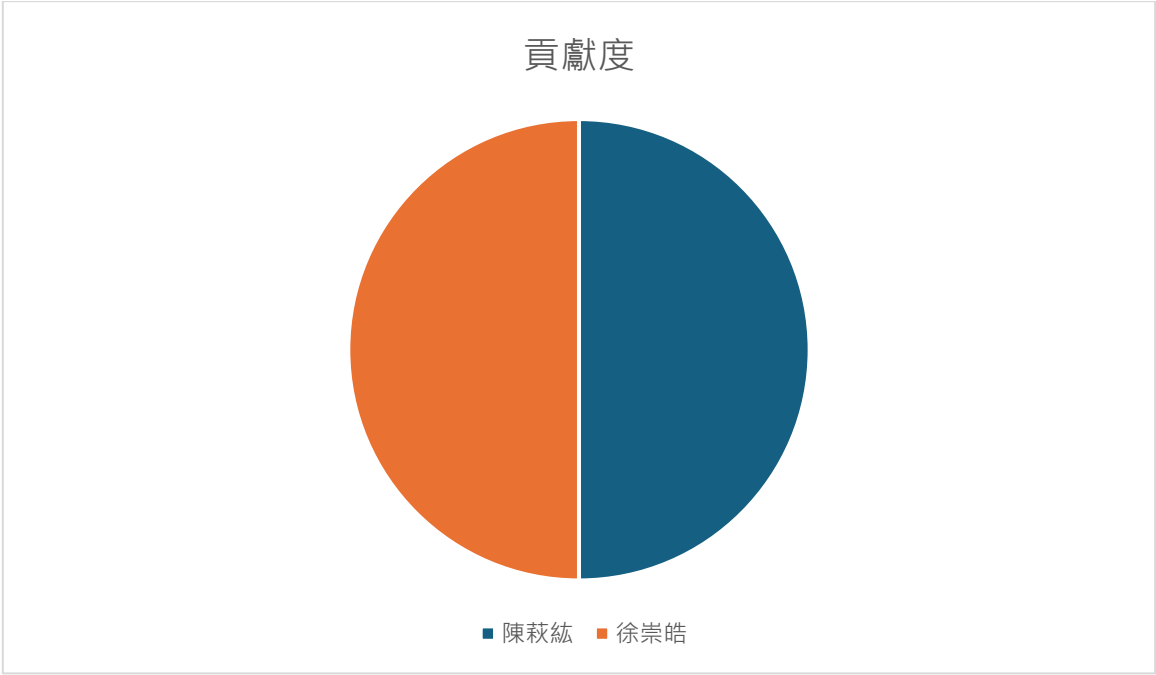
期末專題書面報告

E00063	姓名	學號	分工與貢獻確認
組長	陳菽紘	C110152353	陳菽紘
組員	徐崇皓	C110152334	徐崇皓

分工情況與貢獻百分比

陳菂紘 @ 徐崇皓 #

卷積 電路設計	最大池化 電路設計	模組驗證	模組整合	電路 合成與優化	報告撰寫
#	# @	@	# @	#	@

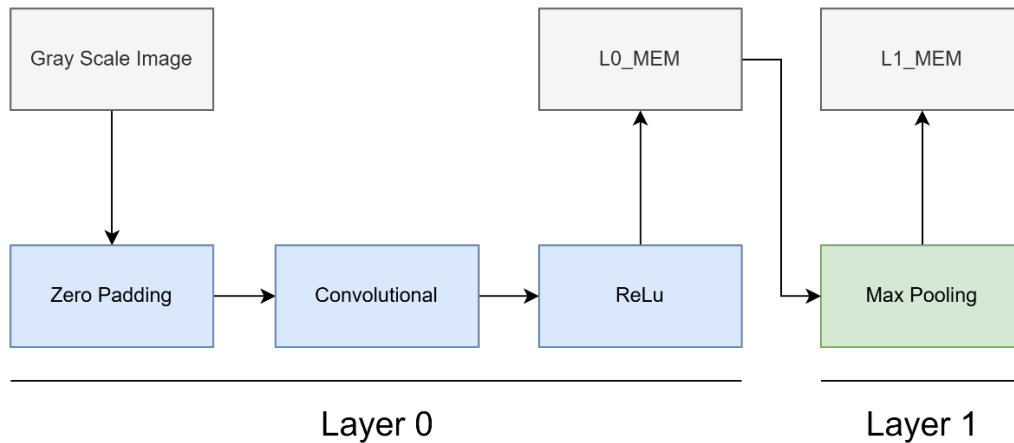


設計實作摘要

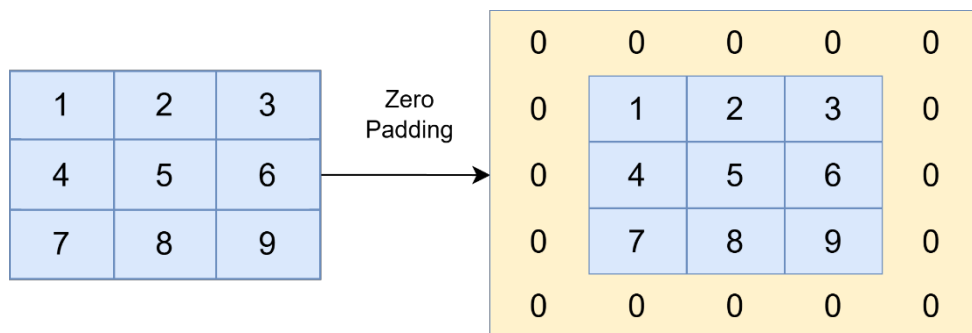
是否通過 RTL（合成前,pre-synthesis）驗證？	是
是否通過 gate-level(post-synthesis)驗證？	是
Clock period(ns)	9
Area(um ²)	27,598.026713
Power(mW)	1.4313
Simulation time(ns) (合成後)	331,904
Coding style check?	Yes
Code coverage evaluation?	Yes
其他重要特色	流水線乘法器選用、平行運算

壹、演算法

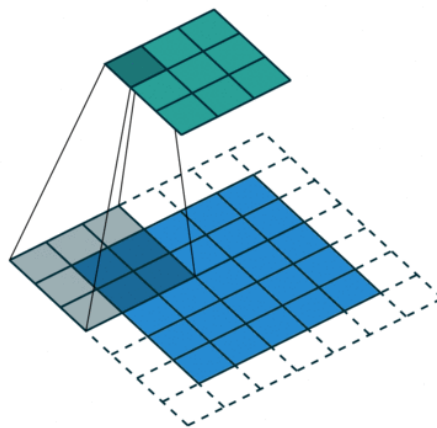
本次期末專題要實作的是一圖像卷積電路，包含兩層的運算，第一層(Layer 0) Convolutional 需進行 Zero Padding、Convolution、ReLU 運算，第二層(Layer 1)進行 Max Pooling 運算，並將各層運算後之結果分別寫回至記憶體之存儲空間內即完成。



首先是第一層(Layer 0)的 Zero Padding，由於圖像在經過 Convolution 過後的尺寸會縮小，所以我們在圖像外圍填入像素 0 來保持圖像原有的尺寸。



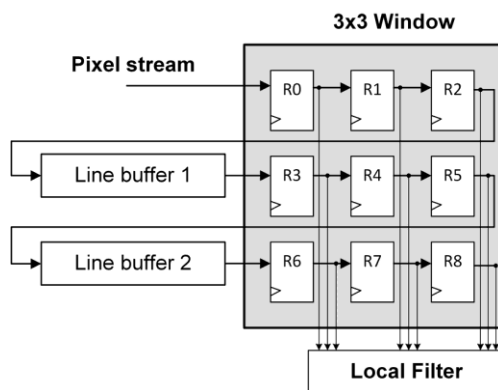
再來則是 Convolution 運算，我們將做完 Zero Padding 的圖像與題目提供的 3x3 Kernel 從左到右由上而下，針對映射到的座標與 Kernel 相乘，然後將相乘的結果加總後再與 bias 值相加。



而我們在本次的期末專題針對 Convolution 運算評估了兩種運算模式演算法，並在以下做介紹：

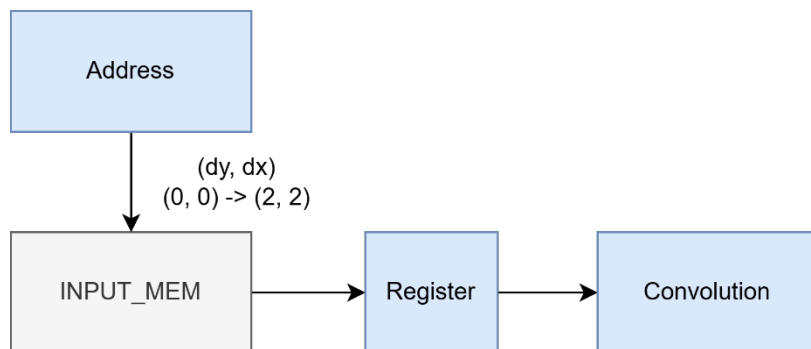
I. FIFO（參考哭哭彼得潘組別並改良）

利用 FIFO 特性實做兩條與圖像同寬之 LineBuffer 以此儲存兩列像素，不斷推入每次從記憶體中讀取的像素，在電路執行完 $\text{width} * 2 + 3$ 個 cycle 後取得完整 3x3 Window，而本次的運算結果即為有效之 Convolution 值，若遇到圖像邊界後則須等待三個 cycle 的推入來取得有效 3x3 Window。



II. 直接存取

在每次的 Convolution 運算中，記憶體中抓取當次 3x3 Window 所需要的像素並在抓取像素的同時進行運算，一次的 Convolution 運算需要九個 cycle 來完成。



最終我們依照兩種演算法的特性與優缺點列出下表：

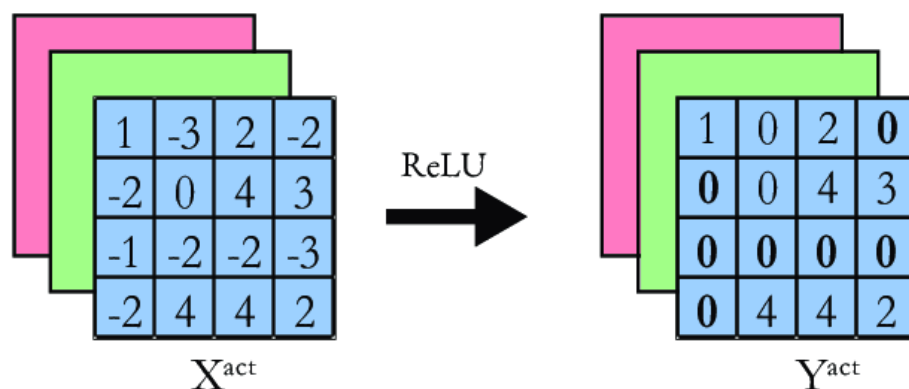
	FIFO	直接存取
完整運算存取時間	Height*Width	$(\text{Height}-2) * (\text{Width}-2) * 9$
暫存器數量	$\text{Width} * 2 + 3$	1
乘法器數量	9	1

可以看到採用 FIFO 的演算法在完整運算存取時間裡面是遠勝直接存取演算法，這是因為 FIFO 演算法在讀入圖像會依序推入 Line Buffer，並透過 FIFO 的特性重複利用已存入的像素，而不像直接存取法花時間去重複讀取曾取得之像素。

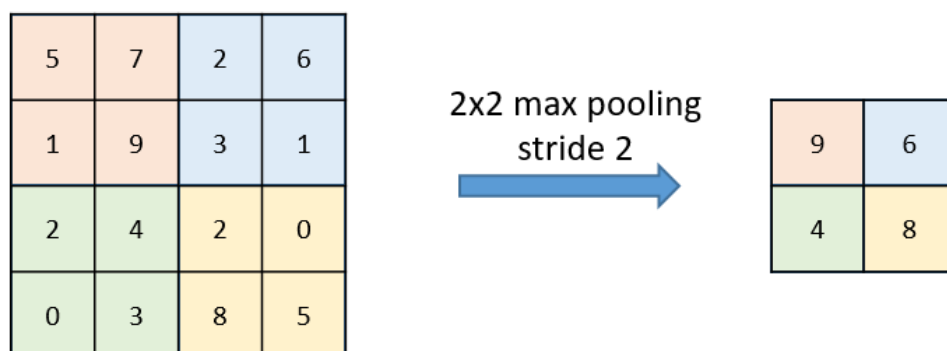
但面積則是 FIFO 演算法劣勢之處，它需要 $Width*2+3$ 個暫存器來確保能夠儲存到 3×3 Window 的左上角之值，且 Window 內的每個元素都需要搭配一個乘法器來發揮 FIFO 演算法的優點，直接存取演算法只需要各一個暫存器與乘法器即可實作。

而我們在考量到直接存取演算法所使用之元件較少，且後續可透過流水線乘法器優化，所以最終決定採用直接存取演算法實作本次的期末專題。

第一層(Layer 0)的最後是 ReLU 運算，題目要求將 Convolution 後所得到之結果經過啟動函數後存入 L0_MEM，而題目給定的啟動函數是只保留大於 0 的數值，而其他輸入數值都將等於 0。



而第二層(Layer 1)我們須進行 Max Pooling，將第一層(Layer 0)的結果進行 2×2 Window、Stride 2 的最大值選擇運算，最後將結果存入 L1_MEM。

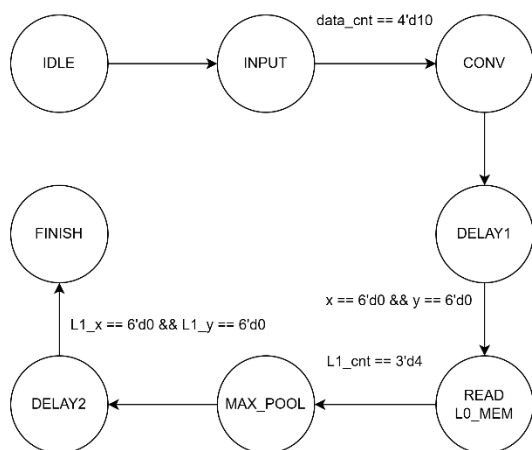


貳、電路架構比較

我們針對這次的圖像卷積電路設計了兩種電路架構並評估優劣，一種是原先架構，另一種是加入流水線乘法器與平行計算架構，以下將比較兩種架構間的差異。

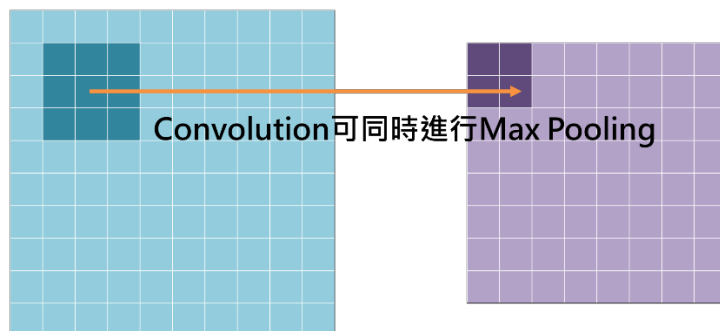
1. 無流水線乘法器與平行計算架構

在這個架構中，我們直接用乘法符號由合成工具選用單周期的乘法器，也沒有採用平行運算的設計讓 Max Pooling 與 Convolution 同時進行，由於這個架構不需要多餘的安排去處理同時存取 Max Pooling 與 Convolution 記憶體時會發生的衝突，也因為不需使用平行運算所以可以共用運算單元，所以它的優點就是設計簡單、面積較小；但缺點就是運作週期取決於合成工具採用之單周期乘法器的速度，且 Max Pooling 與 Convolution 不會同時進行導致將增加許多 Cycle 去分開運算。



2. 加入流水線乘法器與平行計算架構

而在這個架構中，我們加入了流水線乘法器與平行計算，選用 Synopsys DesignWare IP 庫中的 Four-Stage Pipelined Multiplier，將乘法切為四個步驟，能夠幫助我們壓低運作週期，而同時間我們也可以將 Max Pooling 與 Convolution 透過平行運算的方式一起進行，而優點就如同上面描述，可以降低運作週期、大幅減少 Cycle；缺點則是因為採用流水線乘法器，每個像素進行乘法運算時會需要四個週期才會完成，需另外注意第一筆資料送入乘法器運算的前三個週期不是有效的結果；而平行運算雖然能大幅減少 Cycle，但是我們需要在時序上將 Max Pooling 與 Convolution 存取記憶體的時間隔開，避免發生衝突，而這些都會增加我們在設計電路上面的複雜度；另外平行運算也會需要多餘的運算單元，所以將會導致面積會比另一個架構來的更大。



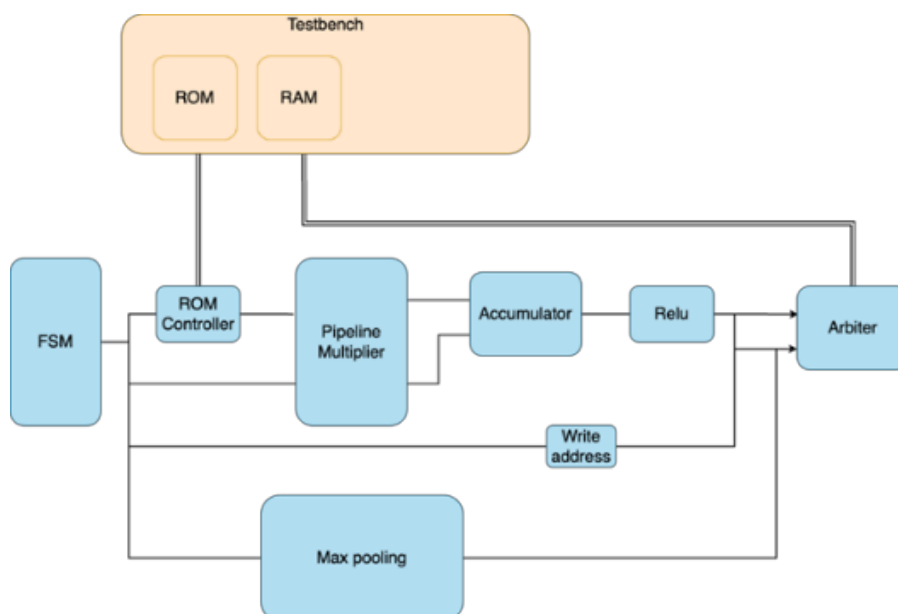
最終我們依照兩種電路架構的特性與優缺點列出下表：

	架構 1	架構 2
設計複雜度	易	難
面積	低	高
週期時長	慢	快
總週期數	多	少

而在考量到整體電路表現後我們採用有加入流水線乘法器與平行運算的架構 2。

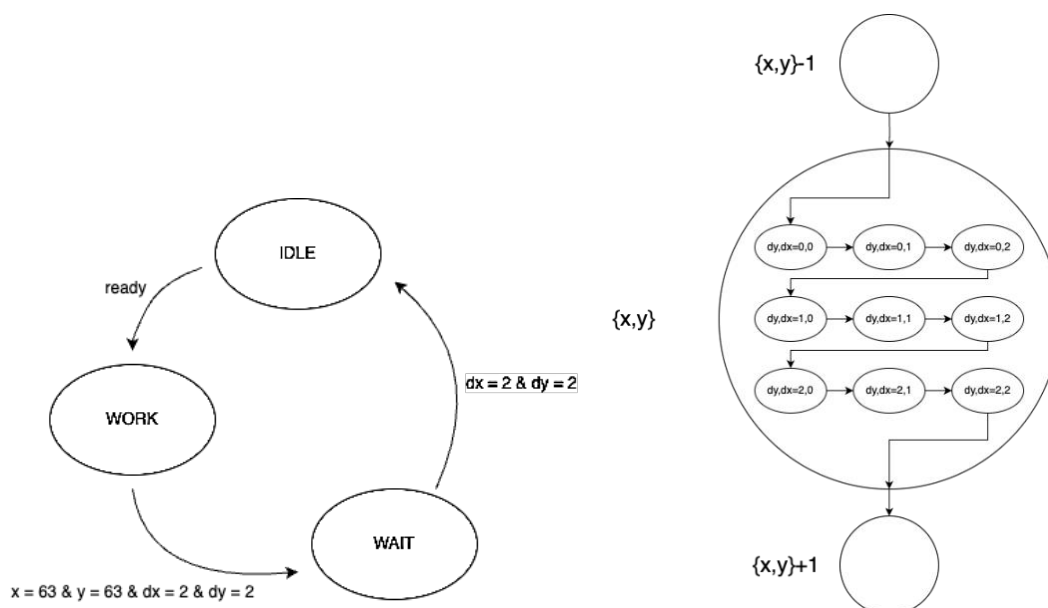
參、電路架構介紹

以下是我們這次電路的整體架構圖：



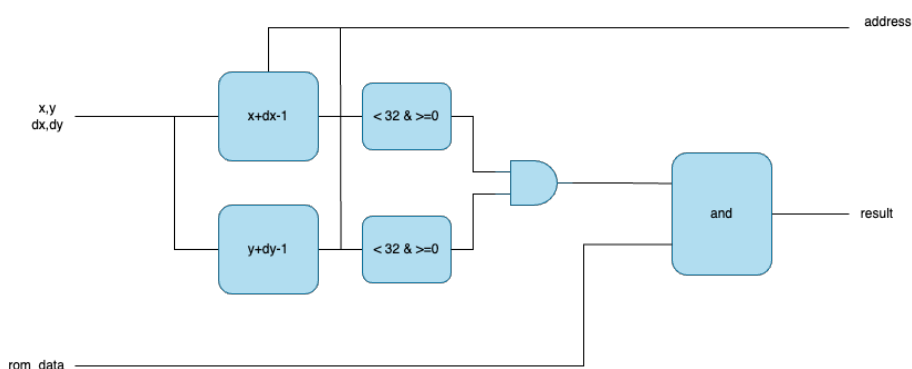
本次設計的圖像卷積電路包含了：

I. FSM



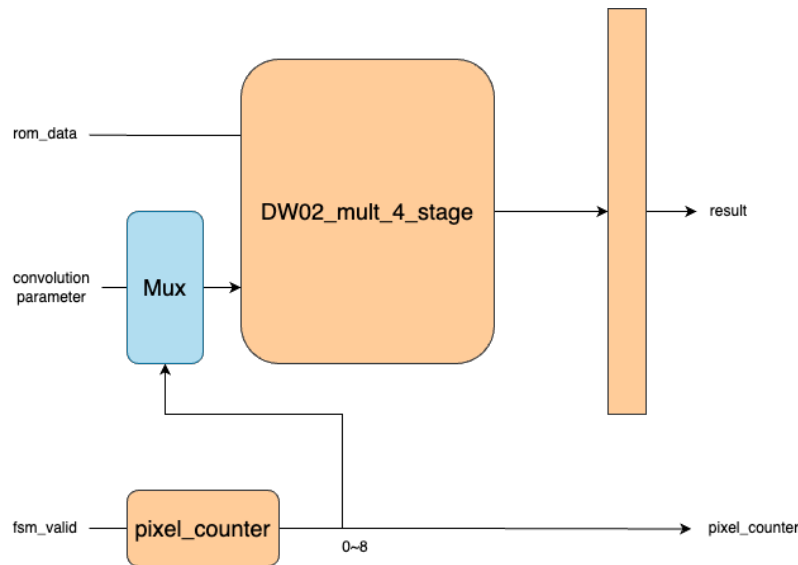
FSM 模組負責輸出每次要讀取的像素座標， x, y 代表的是目前 3x3 Window 中心點在圖像中的座標， dx, dy 則是代表在 Window 中的座標， dx 為 Window 中的水平位置， dy 則為 Window 中的垂直位置，依序輸出座標至結束為止。

II. ROM Controller



ROM Controller 模組負責將 x, y, dx, dy 這四個數值結合在一起，計算出對應到圖檔的位址，並判斷該次的存取位址是否為合法，若非法則輸出 0，這樣就可達到 Zero Padding 的功能。

III. Pipeline Multiplier



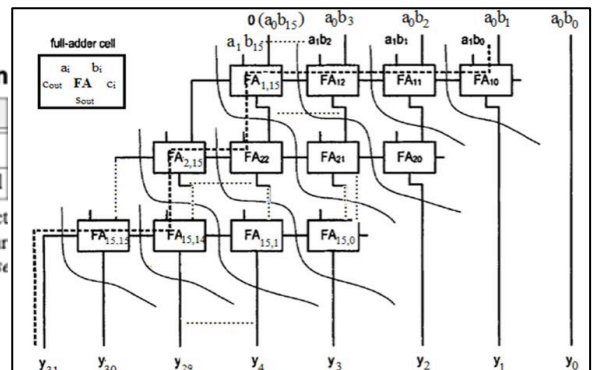
Pipeline Multiplier 模組中我們選用了 Synopsys DesignWare IP 庫裡面的四級流水線設計乘法器，而根據該 IP 的 Datasheet 可以發現它會使用 Carry-save Array ($\text{pixel_width} + \text{kernel_width} \leq 48$) 作為該乘法器的架構去合成，而我們也可以根據 Datasheet 的備註了解到，與 Booth-recoded Wallace-tree 架構相比，在兩數的寬度和小於 48 的條件下，Carry-save Array 乘法器將有面積小的優勢；除此之外，也因為它是規則形狀的架構，在後端佈局上會更加輕鬆。

Table 3: Synthesis Implementation

Implementation Name	Function
csa ^b	Carry-save array synthesis model
str	Booth-recoded Wallace-tree synthesis model

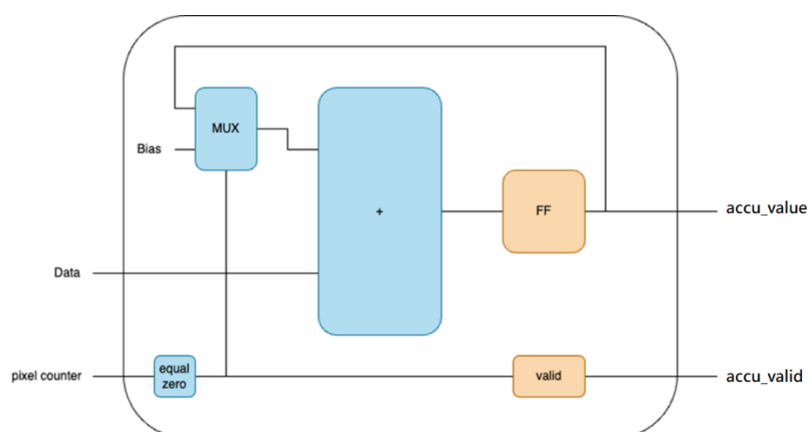
a. During synthesis, Design Compiler will select the appropriate architect. However, you may force Design Compiler to use one of the architectur. For more details, please refer to the *DesignWare Building Block IP Use*

b. The csa implementation is only valid when the sum of A_width and $B_width \leq 48$ bits, as it has no area benefit beyond 48 bits.



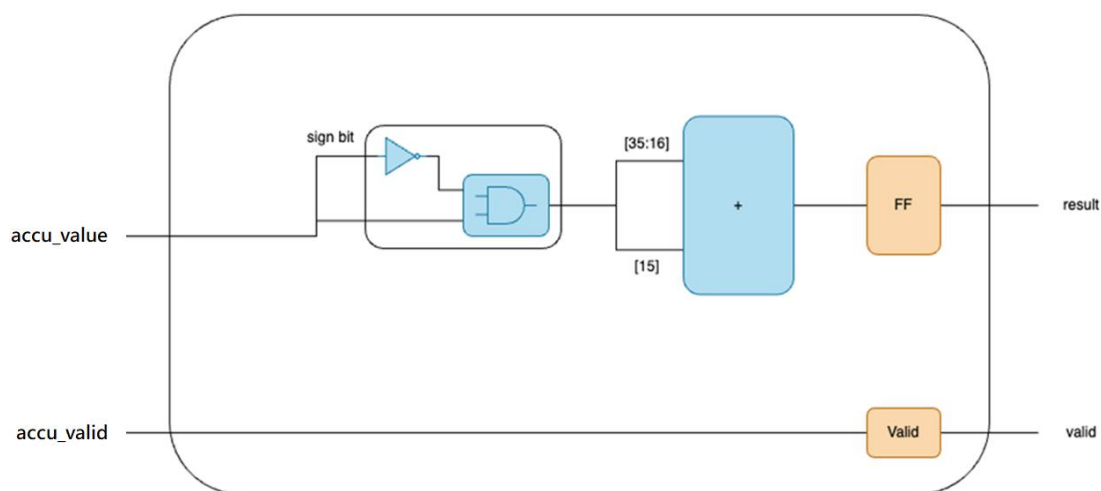
而運作上，像素會依照九宮格的順序一個一個送進乘法器，我們需要根據座標去選擇對應到的 Kernel，這邊我們使用了計數器來計算目前運算到 3x3 Window 的哪一個座標，而因為採用了流水線設計的關係，當前計數到的座標會延遲四個週期才會輸出結果，因此需要再另外計算對應的輸出，而我們實現的方式是更改 counter 初始值，並將多工器的資料線重新排列後便可對應到原本的座標。

IV. Accumulator



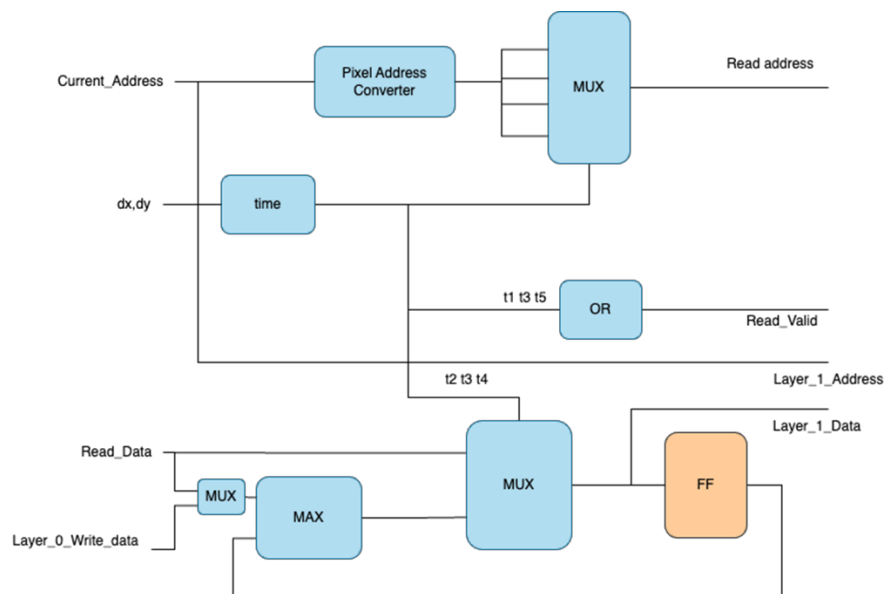
Accumulator 模組會將 Pipeline Multiplier 模組計算完畢的數值進行累加，而第一筆數值因為不須與暫存器累加，所以我們運用多工器去選擇 bias 與第一筆相加儲存至暫存器。

V. ReLU

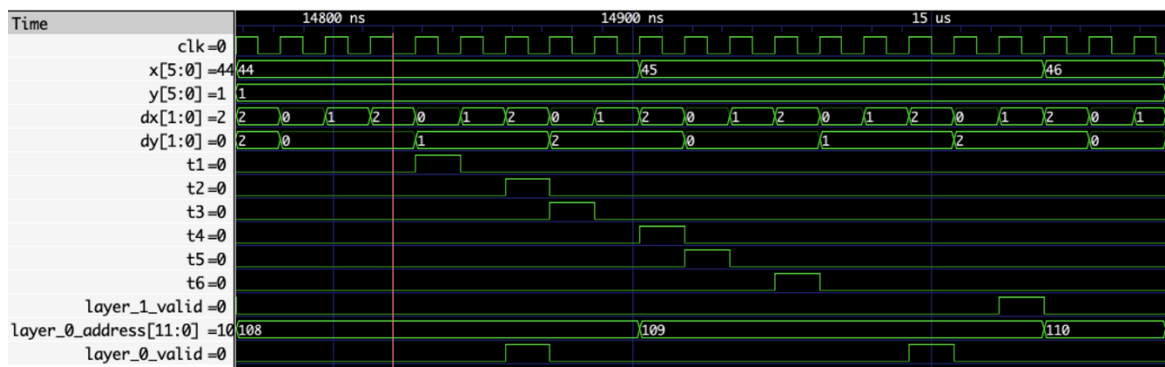


ReLU 模組我們這邊是參考了呵呵組別的設計，透過判斷 sign bit 來實現正數保留，而四捨五入的部分我們則透過加上需要數值範圍的前一位來實現。

VI. Max Pooling



Max Pooling 模組我們使用先前的 FSM 來控制，設置 t1~t6 來控制狀態，當 t1 發起的時候我們從記憶體讀取此次 max pooling 要比較的左上角數值,在 t2 時儲存較大值至暫存器，(t3 t4) (t5 t6)則分別做右上與左下，最後剩右下也就是 convolution 正在計算的值也剛好算完準備寫入記憶體，這時只需從記憶體讀取該值做最後比較即可完成，而我們也將該次 Max Pooling 的輸出在 Address 改變前（也就是 dx==1 dy==2 時）輸出。



肆、合成流程

由於 GUI 的使用效率相對較低，因此我們決定參考 IC Design Contest 提供的合成 TCL 腳本和 Constraints 檔來進行本次的電路合成。

ii. 合成指令

Command : dc_shell -f dc_syn.tcl

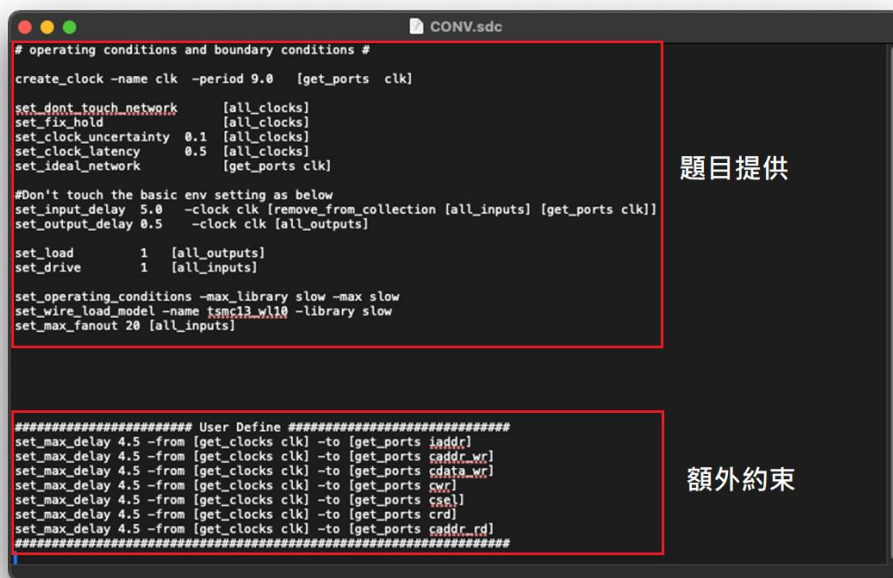
iii. Constraints

1. 由題目所提供

- 基本的時序約束
- I/O 約束
- 設定操作環境

2. 增加新的約束

- 輸出至記憶體時會透過一個組合電路後才輸出，記憶體要求在負緣前送達，因此要控制在一半的週期時間前送達。



```
# operating conditions and boundary conditions #
create_clock -name clk -period 9.0 [get_ports clk]

set_dont_touch_network [all_clocks]
set_fix_hold [all_clocks]
set_clock_uncertainty 0.1 [all_clocks]
set_clock_latency 0.5 [all_clocks]
set_ideal_network [get_ports clk]

#Don't touch the basic env setting as below
set_input_delay 5.0 -clock clk [remove_from_collection [all_inputs] [get_ports clk]]
set_output_delay 0.5 -clock clk [all_outputs]

set_load 1 [all_outputs]
set_drive 1 [all_inputs]

set_operating_conditions -max_library slow -max_slow
set_wire_load_model -name tsmc13_wl10 -library slow
set_max_fanout 20 [all_inputs]

##### User Define #####
set_max_delay 4.5 -from [get_clocks clk] -to [get_ports iaddr]
set_max_delay 4.5 -from [get_clocks clk] -to [get_ports caddr_wr]
set_max_delay 4.5 -from [get_clocks clk] -to [get_ports caddr_wr]
set_max_delay 4.5 -from [get_clocks clk] -to [get_ports cwr]
set_max_delay 4.5 -from [get_clocks clk] -to [get_ports csel]
set_max_delay 4.5 -from [get_clocks clk] -to [get_ports crd]
set_max_delay 4.5 -from [get_clocks clk] -to [get_ports caddr_rd]
```

iv. Design Compiler 腳本

1. 統一將路徑與名稱設定成變數供之後腳本使用（方便之後修改路徑）。

- set Design_Source ../src
- set Project_Name CONV

2. 讀取電路檔案。

- `analyze -format verilog { $Design_Source/$Project_Name.v }`
- `elaborate CONV`
- `current_design $Project_Name`
- `link`

3. 讀取 Constraint 檔。

- `source -echo -verbose $Design_Source/$Project_Name.sdc`
- `check_design`

4. 將直接輸出插入 Buffer Gate。

- `set_fix_multiple_port_nets -all -buffer_constants [get_designs *]`

5. 使用 Ultra High Effort 設定進行合成。

- `Compile_ultra`

6. 設定輸出資料夾。

- `set Project_Name_syn [append Project_Name "_syn"]`
- `file mkdir Results`
- `file mkdir Reports`

7. 儲存 SDF 延遲檔。

- `write_sdf -version 1.0 "./Results/$Project_Name_syn.sdf"`

8. 輸出合成後 netlist 檔。

- `write -format verilog -hierarchy -output "./Results/$Project_Name_syn.v"`

9. 儲存面積報告。

- `report_area > ./Reports/area.log`

10. 儲存時序報告。

- `report_timing -delay_type min > ./Reports/timing_min.log`
- `report_timing -delay_type max > ./Reports/timing_max.log`

11. 儲存功耗報告。
 - `report_power > ./Reports/power.log`
12. 打開 GUI (若有需要可以開啟)。
 - `gui_start`



```
#####
# File Name      : dc_syn.tcl
# Author       : ChongHa_Xu
# Description  :
#
# Design Source-
#   Circuit    :Circuit_Name.v(.sv)
#   Constraints :Constraint.sdc
#
#####
set Design_Source ./src
set Project_Name CONV
#####

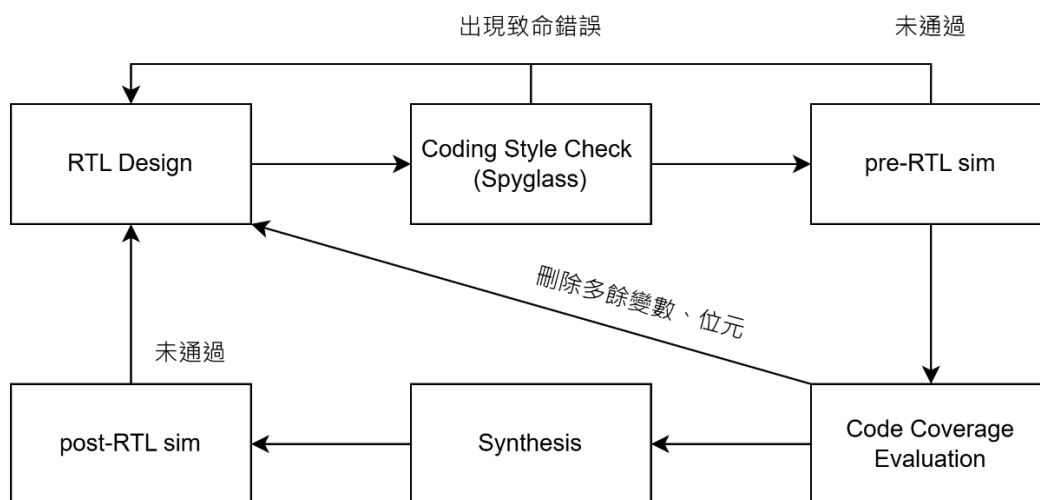
##### 讀取Verilog檔案 #####
##### Read Files #####
#####
analyze -format verilog { $Design_Source/$Project_Name.v }
elaborate CONV
current_design $Project_Name
link
#####

##### Setting Constraints #####
#####
source -echo -verbose $Design_Source/$Project_Name.sdc
check_design
set_fix_multiple_port_nets -all -buffer_constants [get_designs *]
##### 讀取電路Constraints #####

##### Synthesis all design #####
#####
compile_ultra
##### 合成電路 #####

##### Write Out #####
##### 儲存合成結果 #####
set Project_Name_syn [append Project_Name "_syn"]
file mkdir Results
file mkdir Reports
write_sdf -version 1.0 "./Results/$Project_Name_syn.sdf"
write -format verilog -hierarchy -output "./Results/$Project_Name_syn.v"
report_area > ./Reports/area.log
report_timing -delay_type min > ./Reports/timing_min.log
report_timing -delay_type max > ./Reports/timing_max.log
report_power > ./Reports/power.log
#####
```

伍、驗證流程



陸、邏輯合成與驗證

一、架構 1（無流水線乘法器與平行計算）

I. 電路面積

```

Number of ports:          105
Number of nets:           2048
Number of cells:          1795
Number of combinational cells: 1637
Number of sequential cells: 157
Number of macros/black boxes: 0
Number of buf/inv:        258
Number of references:      96

Combinational area:       16556.439451
Buf/Inv area:             1770.388182
Noncombinational area:    5114.266037
Macro/Black Box area:     0.000000
Net Interconnect area:    228201.137512
Total cell area:          21670.705488
Total area:               249871.843000
  
```

	Area(um ²)	Gate Count
NAND2	5	1
架構 1	21670.705488	≈4334

II. 功率消耗

```
Operating Conditions: slow  Library: slow
Wire Load Model Mode: top

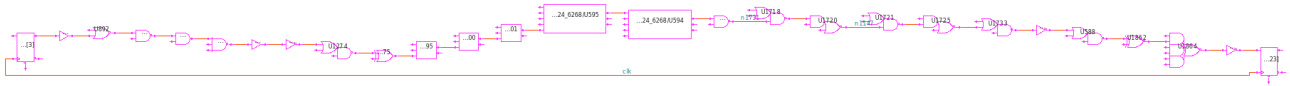
Design      Wire Load Model      Library
-----
CONV        tsmc13_wl10          slow

Global Operating Voltage = 1.08
Power-specific unit information :
  Voltage Units = 1V
  Capacitance Units = 1.000000pf
  Time Units = 1ns
  Dynamic Power Units = 1mW      (derived from V,C,T units)
  Leakage Power Units = 1pW

  Cell Internal Power  = 444.4033 uW  (49%)
  Net Switching Power  = 469.8568 uW  (51%)
  Total Dynamic Power   = 914.2600 uW  (100%)
  Cell Leakage Power    = 17.1988 uW
```

	架構 1
Total Dynamic Power(mW)	0.91426
Cell Leakage Power(uW)	17.1988

III. 最長路徑標示與延遲



Operating Conditions: slow Library: slow
Wire Load Model Mode: top

Startpoint: counter_data_reg[3]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: data_conv_sum_reg[23]
(rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max

Des/Clust/Port	Wire Load Model	Library
CONV	tsmc13_wl10	slow

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.50	0.50
counter_data_reg[3]/CK (DFFRX1)	0.00	0.50 r
counter_data_reg[3]/Q (DFFRX1)	0.65	1.15 f
U569/Y (BUFX4)	0.27	1.42 f
U892/Y (NOR2X1)	0.42	1.84 r
U1256/Y (NAND2X2)	0.32	2.17 f
U1257/Y (AND2X2)	0.35	2.52 f
U717/Y (NAND3X2)	0.15	2.67 r
U773/Y (BUFX12)	0.27	2.94 r
U565/Y (CLKINX1)	0.51	3.45 f
U1274/Y (OAI21X1)	0.42	3.87 r
U1275/Y (XOR2X1)	0.29	4.16 f
U1295/CO (ADDFXL)	0.54	4.70 f
U1300/CO (ADDFXL)	0.53	5.23 f
U1301/CO (ADDFXL)	0.52	5.75 f
DP_OP_104J1_124_6268/U595/S (CMPR42X1)	0.45	6.21 r
DP_OP_104J1_124_6268/U594/S (CMPR42X1)	0.91	7.11 r
U591/Y (NAND2XL)	0.35	7.46 f
U1718/Y (OAI21X1)	0.37	7.83 r
U1720/Y (AOI21X1)	0.29	8.12 f
U1721/Y (AOI21X4)	0.17	8.30 r
U1725/Y (AOI21X4)	0.15	8.44 f
U1733/Y (AOI21X4)	0.24	8.68 r
U1734/Y (INVX6)	0.17	8.85 f
U588/Y (OAI21XL)	0.46	9.31 r
U1862/Y (XNOR2X1)	0.38	9.69 f
U1864/Y (AOI222X1)	0.41	10.10 r
U669/Y (INVX1)	0.16	10.27 f
data_conv_sum_reg[23]/D (DFFRX2)	0.00	10.27 f
data arrival time		10.27
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	0.50	10.50
clock uncertainty	-0.10	10.40
data_conv_sum_reg[23]/CK (DFFRX2)	0.00	10.40 r
library setup time	-0.13	10.27
data required time		10.27
data required time		10.27
data arrival time		-10.27
slack (MET)		0.00

```

Operating Conditions: slow  Library: slow
Wire Load Model Mode: top

Startpoint: counter_layer1_reg[0]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: counter_layer1_reg[0]
(rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min

Des/Clust/Port      Wire Load Model      Library
-----
CONV                tsmc13_wl10           slow

Point              Incr      Path
-----
clock clk (rise edge)          0.00      0.00
clock network delay (ideal)    0.50      0.50
counter_layer1_reg[0]/CK (DFFRX1) 0.00      0.50 r
counter_layer1_reg[0]/Q (DFFRX1) 0.49      0.99 r
U704/Y (NOR2X1)                0.17      1.16 f
counter_layer1_reg[0]/D (DFFRX1) 0.00      1.16 f
data arrival time              1.16

clock clk (rise edge)          0.00      0.00
clock network delay (ideal)    0.50      0.50
clock uncertainty              0.10      0.60
counter_layer1_reg[0]/CK (DFFRX1) 0.00      0.60 r
library hold time             -0.05      0.55
data required time             0.55

data required time             0.55
data arrival time             -1.16

slack (MET)                    0.61

```

可以看到 timing_max 與 timing_min 都符合 slack

IV. 模擬時間

```

-----
START!!! Simulation Start .....

-----

Layer 0 (Convolutional Output) with Kernel 0 is correct !
Layer 1 (Max-pooling Output) with Kernel 0 is correct!

-----

----- S U M M A R Y -----

Congratulations! Layer 0 data have been generated successfully! The result is PASS!!
Congratulations! Layer 1 data have been generated successfully! The result is PASS!!

-----

Simulation complete via $finish(1) at time 653378 NS + 0

```

合成前模擬耗時 653,378(ns)

```

-----
START!!! Simulation Start .....
-----

Layer 0 (Convolutional Output) with Kernel 0 is correct !
Layer 1 (Max-pooling Output) with Kernel 0 is correct!
-----

----- S U M M A R Y -----

Congratulations! Layer 0 data have been generated successfully! The result is PASS!!
Congratulations! Layer 1 data have been generated successfully! The result is PASS!!
-----

Simulation complete via $finish(1) at time 653379138 PS + 0

```

合成後模擬耗時 653,379(ns)

二、架構 2（有流水線乘法器與平行計算）

I. 電路面積

```

Number of ports:                183
Number of nets:                 2270
Number of cells:                1912
Number of combinational cells:  1510
Number of sequential cells:     397
Number of macros/black boxes:   0
Number of buf/inv:             274
Number of references:           89

Combinational area:             16388.397162
Buf/Inv area:                   1884.114015
Noncombinational area:          11209.629551
Macro/Black Box area:           0.000000
Net Interconnect area:          241267.898071

Total cell area:                 27598.026713
Total area:                     268865.924784

```

	Area(um ²)	Gate Count
NAND2	5	1
架構 2	27598.026713	≅5520

II. 功率消耗

```
Operating Conditions: slow  Library: slow
Wire Load Model Mode: top

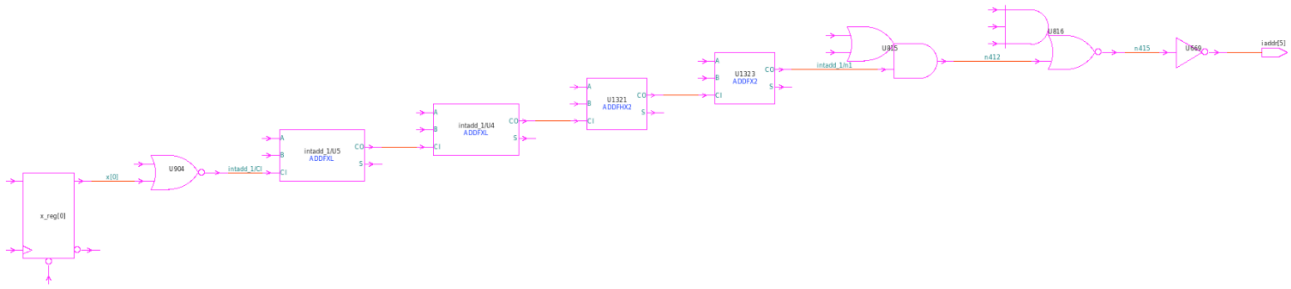
Design      Wire Load Model      Library
-----
CONV        tsmc13_wl10          slow

Global Operating Voltage = 1.08
Power-specific unit information :
  Voltage Units = 1V
  Capacitance Units = 1.000000pf
  Time Units = 1ns
  Dynamic Power Units = 1mW      (derived from V,C,T units)
  Leakage Power Units = 1pW

  Cell Internal Power  = 952.0519 uW   (67%)
  Net Switching Power  = 479.2629 uW   (33%)
  -----
  Total Dynamic Power   = 1.4313 mW   (100%)
  Cell Leakage Power    = 21.8150 uW
```

	架構 2
Total Dynamic Power(mW)	1.4313
Cell Leakage Power(uW)	21.81450

III. 最長路徑標示與延遲



Operating Conditions: slow Library: slow
Wire Load Model Mode: top

Startpoint: x_reg[0] (rising edge-triggered flip-flop clocked by clk)
Endpoint: iaddr[5] (output port clocked by clk)
Path Group: clk
Path Type: max

Des/Clust/Port	Wire Load Model	Library
CONV	tsmc13_wl10	slow

Point	Incr	Path
clock network delay (ideal)	0.50	0.50
x_reg[0]/CK (DFFRX1)	0.00	0.50 r
x_reg[0]/Q (DFFRX1)	0.73	1.23 f
U904/Y (NOR2X1)	0.40	1.63 r
intadd_1/U5/CO (ADDFXL)	0.51	2.14 r
intadd_1/U4/CO (ADDFXL)	0.51	2.65 r
U1321/CO (ADDFHX2)	0.28	2.93 r
U1323/CO (ADDFHX2)	0.37	3.30 r
U815/Y (OA21XL)	0.33	3.63 r
U816/Y (AOI31X4)	0.30	3.93 f
U669/Y (INVX16)	0.46	4.40 r
iaddr[5] (out)	0.00	4.40 r
data arrival time		4.40
max delay	4.50	4.50
clock network delay (ideal)	0.50	5.00
clock uncertainty	-0.10	4.90
output external delay	-0.50	4.40
data required time		4.40
data required time		4.40
data arrival time		-4.40
slack (MET)		0.00

Operating Conditions: slow Library: slow
Wire Load Model Mode: top

Startpoint: pipeline_multiplier_u2/multi/clk_r_REG83_S2
(rising edge-triggered flip-flop clocked by clk)
Endpoint: pipeline_multiplier_u2/multi/clk_r_REG84_S3
(rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: min

Des/Clust/Port	Wire Load Model	Library
CONV	tsmc13_wl10	slow

Point	Incr	Path
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.50	0.50
pipeline_multiplier_u2/multi/clk_r_REG83_S2/CK (DFFQX1)	0.00	0.50 r
pipeline_multiplier_u2/multi/clk_r_REG83_S2/Q (DFFQX1)	0.30	0.80 f
pipeline_multiplier_u2/multi/clk_r_REG84_S3/D (DFFQX1)	0.00	0.80 f
data arrival time		0.80
clock clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.50	0.50
clock uncertainty	0.10	0.60
pipeline_multiplier_u2/multi/clk_r_REG84_S3/CK (DFFQX1)	0.00	0.60 r
library hold time	-0.09	0.51
data required time		0.51
data required time		0.51
data arrival time		-0.80
slack (MET)		0.30

可以看到 timing_max 與 timing_min 都有符合 Slack

IV. 模擬時間

```
-----  
START!!! Simulation Start .....  
-----  
Layer 0 (Convolutional Output) with Kernel 0 is correct !  
Layer 1 (Max-pooling Output) with Kernel 0 is correct!  
-----  
----- S U M M A R Y -----  
-----  
Congratulations! Layer 0 data have been generated successfully! The result is PASS!!  
Congratulations! Layer 1 data have been generated successfully! The result is PASS!!  
-----  
Simulation complete via $finish(1) at time 331902 NS + 0
```

合成前模擬耗時 331,902(ns)

```
-----  
START!!! Simulation Start .....  
-----  
Layer 0 (Convolutional Output) with Kernel 0 is correct !  
Layer 1 (Max-pooling Output) with Kernel 0 is correct!  
-----  
----- S U M M A R Y -----  
-----  
Congratulations! Layer 0 data have been generated successfully! The result is PASS!!  
Congratulations! Layer 1 data have been generated successfully! The result is PASS!!  
-----  
Simulation complete via $finish(1) at time 331904295 PS + 0
```

合成後模擬耗時 331,904(ns)

三、效能比較

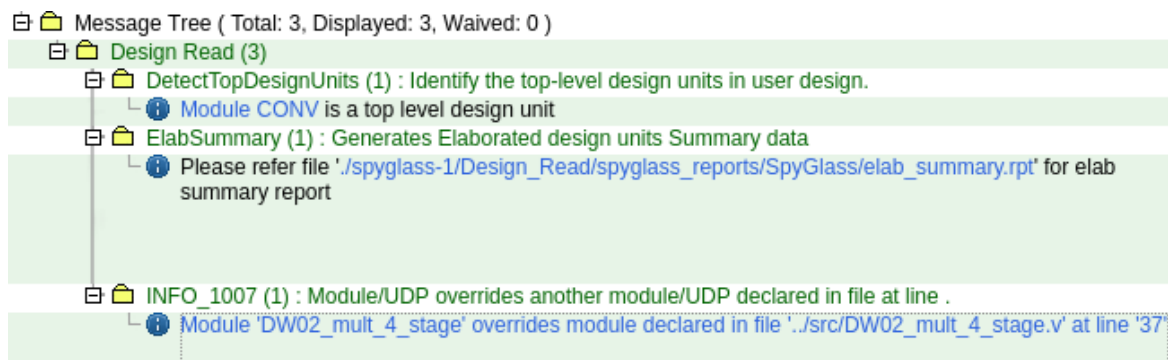
根據以上數據列出兩個架構版本互相比較表：

	架構 1	架構 2
Total Dynamic Power(mW)	0.91426	1.4313
Total Cell Area(um ²)	21,670.705488	27,598.026713
Total Simulation Time(ns)	653,379	331,904
Performance	14,159,183,881	9,159,895,458

我們可以發現雖然架構 2 在面積與功耗上輸給了架構 1，但在模擬時間與表現上遠遠的贏過了架構 1，而這也是我們會改為採用架構 2 的原因：能夠大幅減少計算時間使綜合表現達到更好。

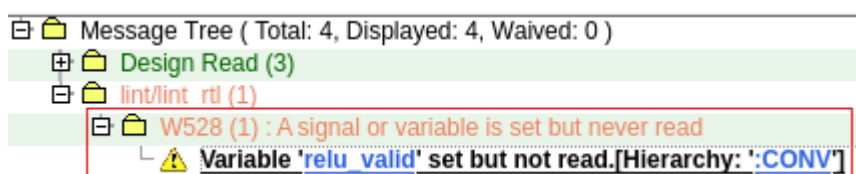
柒、Coding Style 驗證

一、Design Read

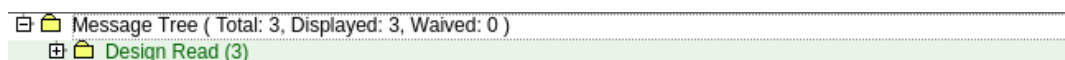


在進行 Design Read 的過程中沒有檢查出任何錯誤。

二、Lint 檢查



在進行 Lint RTL 檢查時出現 W528 警告，深入了解後發現是我們在設計 RTL 時有多出一個變數叫”relu_valid”但沒有使用上，在移除該變數後重新檢測即無警告提醒。



捌、測試覆蓋率

我們使用 Synopsys 所提供的 vcs 以及 dve 來觀察測試覆蓋率，並觀察了共五種的測試覆蓋率 Line、Toggle、FSM，Condition 與 Branch，下圖是將最終架構（架構 2）進行 Code Coverage Evaluation 的結果：

I. 整體覆蓋率

Name	Score	Line	Toggle	FSM	Condition	Branch
testfixture	87.91%	90.48%	91.62%	75.00%	94.94%	87.50%
u_CONV	92.44%	100.00%	92.10%	75.00%	97.37%	97.73%
rom_controller_u1	94.37%	100.00%	88.73%			
relu_u4	97.15%	100.00%	91.45%			100.00%
pipeline_multiplier_u2	92.31%	100.00%	94.14%		84.62%	90.48%
multi	86.91%	100.00%	94.41%		81.82%	71.43%
max_pooling_u5	96.60%	100.00%	86.42%		100.00%	100.00%
accumulator_u3	100.00%	100.00%	100.00%		100.00%	100.00%

II. Toggle

我們實際去了解為何會有未被覆蓋的情況，發現是因為 TB 在測資設計上沒辦法完整使用到所有的位元或狀態，而 pipeline_multiplier 則是因為 DesignWare IP 原本的設計所以沒有完整覆蓋。

III. FSM

	✓ 0	✓ 1	✓ 2
✓ 0 IDLE	-	-	✓
✓ 1 WAIT	✓	-	-
✓ 2 WORK	✗	✓	-

WORK 狀態沒有跳至 IDLE 狀態的情況是因為 TB 不會有在 WORK 狀態時拉高 Reset 的情況（跳轉條件）發生。

玖、心得與討論

一、陳菽紘：

這次的期末專題讓我學會如何與他人一同完成一份小型專案，透過團隊分工與構想，先劃分出各自的工作與時程，想辦法在有限的時間內討論出一個可行的方法並實踐，涉及到雙方共同工作時也需要仰賴雙方的溝通與協調能力，對於我來說算是少有的體驗。

而本次作業中我認為最困難的地方是想辦法構思出一份能夠在眾多組別脫穎而出的演算法與硬體架構，我們評估了許多種版本的設計，最終選定了一份我們認為符合面積與效能平衡的版本，在期中進度報告前就完成合成前後模擬，原先我們打算就不再變動設計，而在聆聽完各組別的報告後我們打算再進一步優化整個電路不管是時序或是面積等設計，最終將 clock period 從 10ns 壓至 9ns 甚至更低，但我們打算保險一點就以 9ns 收尾，我認為這份專題的表現應該能夠進入班級前 10% 內。

我們在電路設計的中途也有遇到相當多的困難，像是 Convolution 與 Max Pooling 平行運算之間會有的記憶體存取衝突，我們在這邊卡了好幾天的時程，最終我們畫出波形圖，釐清兩者間的時序關係，最終成功設計出能夠正確運作的版本，而這次的成功也讓我們模擬時間大幅減少，也將可能會是我們能夠脫穎而出的關鍵。

最後，我也透過報告的撰寫中了解到對於一個電路設計該如何詳細並精確的展現出電路運作原理、優勢還有比較優劣等...，這對於未來在進行不管是任何的報告都有相當大的幫助。

非常感謝這學期教授的授業與助教的測驗作業批改、解惑。

二、徐崇皓：

這次的期末作業對我來說是一個前所未有的體驗，從中我收穫了許多。由於這個電路我之前已經設計過一次，因此在實現其基本功能上並不困難。真正的挑戰在

於如何選擇一個優秀的架構來撰寫，從而使電路效能達到最佳狀態。此外，我還考慮到了許多其他因素，使得這次設計更加全面和完善。

首先，我們必須明白在電路設計中，架構的選擇對於效能的影響是至關重要的。一個好的架構不僅能提高電路的運行速度，還能降低功耗，提升整體效率。在這次作業中，我選擇了使用 CSA Pipeline 的結構來設計乘法器。這種結構最顯著的優勢是能同時運算多筆資料，平均一個週期就可以輸出一筆結果。

在設計過程中，我深入研究了 CSA Pipeline 結構的特點及其優缺點。這讓我對如何優化電路設計有了更深入的理解。使用 CSA Pipeline 結構不僅能提高運算速度，還能在進行 Layout 時更好地壓縮電路的面積。雖然這次作業並未考慮 Layout 後的效能，但我認為這種思考方式對於未來的設計工作是非常有益的。

除了選擇合適的架構外，Constraints 的設定也是一個重要的考量因素。在這次作業中，我學習了如何根據實際需求來設計和調整 Constraints，從而確保電路在各種運行環境下都能保持穩定高效的性能。通過這樣的實踐，我深刻體會到了時序設計的重要性。時序設計不僅僅是為了滿足設計規範，更是為了在實際應用中保證電路的可靠性和穩定性。

這次作業讓我認識到，數位 IC Design 是一個需要全面考慮多方面因素的過程。從選擇架構、優化設計到設定約束，每一個環節都需要細心推敲和反復驗證。只有在不斷的嘗試和改進中，才能設計出高效可靠的電路。這種全方位的考量和設計思維，讓我在學習中受益匪淺。

非常感謝教授與助教在這一整學期中的傳道授業解惑！！！！