

# 深度學習積體電路設計 Lab

# Deep Learning IC Design LAB

[LAB 1 : Booth Algorithm](#)

---

[LAB 2 : Approximate Average](#)

---

[LAB 3-1 : Vivado軟體開發環境教學](#)

---

[LAB 3-2 : Custom IP的建置與使用](#)

---

[LAB 4 : CDMA系統建立與資料傳輸](#)

---

[LAB 5 : Convolution運算系統](#)

---

[LAB 6 : IC contest 2019](#)

---

[LAB 7 : MNIST影像辨識系統](#)

# LAB 1 : Booth Algorithm

## 1. Introduction

Booth algorithm is a multiplication operation that multiplies two numbers in two's complement notation. The detail algorithm is described as below:

- (1) Assume that the multiplicand  $m$  is  $x$ -bit and multiplier  $r$  is  $y$ -bit.

Initialize a register  $P$  for the final result, and the length is  $x+y+1$  bits.

The initial value of  $P$  is  $0(x \text{ bits})\_r(y \text{ bits})\_0(1 \text{ bit})$ .

- (2) The rightmost 2 bits used for the selection of different executions.

| LSB | Execution                              |
|-----|--|
| 00  | No execution                           |
| 01  | Add $m$ to the left part of $P$        |
| 10  | Subtract $m$ from the left part of $P$ |
| 11  | No execution                           |

\* all of the overflow can be ignored during the execution

- (3) **Arithmetically** shift 1 bit on  $P$ .
- (4) Repeat the step (2) and (3) for  $y$  times.
- (5) The final answer is obtained by dropping the LSB from  $P$ .

## 2. Design Specifications

### 2.1 Block Overview



### 2.2 I/O Interface

| Signal Name | I/O | width | Description  |
|-------------|-----|-------|--------------|
| in1         | I   | 6     | Multiplicand |
| in2         | I   | 6     | Multiplier   |
| out         | O   | 12    | Product      |

### 2.3 File Description

| File Name  | Description  |
|------------|--|
| booth.v    | The top module of the design.  |
| booth_tb.v | The testbench file. You are <b>not allowed</b> to modify the content in this file. |

## 3. Scoring

### 3.1 Functional Simulation (pre-sim) [100%]

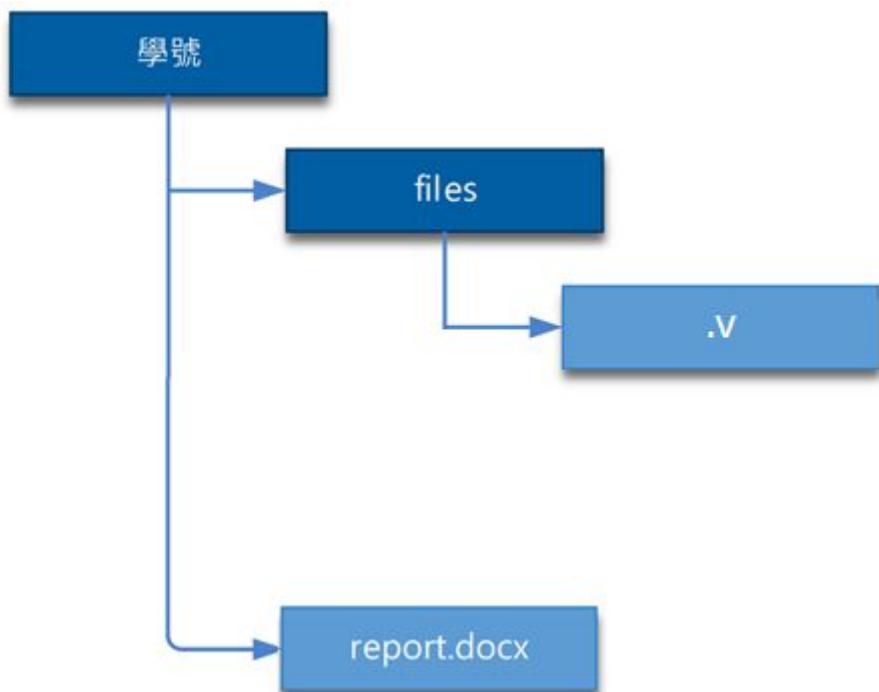
All of the result should be generated correctly, and you will get the following message in ModleSim simulation.

```
"-----  
# 4020 data is correct  
# 4021 data is correct  
# 4022 data is correct  
# 4023 data is correct  
# 4024 data is correct  
# 4025 data is correct  
# 4026 data is correct  
# 4027 data is correct  
# 4028 data is correct  
# 4029 data is correct  
# 4030 data is correct  
# 4031 data is correct  
# 4032 data is correct  
# -----PASS-----  
# All data have been generated successfully!  
# ** Note: $stop : E:/2016DICHW/HW2/booth_tb.v(43)  
# Time: 201700 ns Iteration: 0 Instance: /booth_tb  
# Break in Module booth_tb at E:/2016DICHW/HW2/booth_tb.v line 43
```

## 作業繳交格式

本作業同學須完成電路設計與報告撰寫，請依照下圖的資料夾格式來繳交完成的檔案

1. 繳交檔名為學號.zip，裡面會有一個自己學號的資料夾
2. 學號的資料夾中，會有一個report.docx和一個files的資料夾
3. files中會有所有的.v檔
4. 報告內容請參考report.docx



# LAB 2 : Approximate Average

## 1. Introduction

Please design a computational system whose transfer function is defined as follow. A series of 8-bit positive integer is generated as the input of the computational system by the test bench. The output value Y is a 10-bit positive integer, which is calculated according to equations (1), (2), (3) and (4).

$$Xavg_j = \left\lfloor \frac{\sum_{i=j}^{j+n-1} X_i}{n} \right\rfloor \dots \quad (1)$$

where  $X_i$  is the value of the  $i$ th input data and  $j \geq 1$ .

$$XS = \{X_j, X_{j+1}, X_{j+2}, \dots, X_{j+n-1}\} \dots \quad (2)$$

$$Xappr_j = \begin{cases} Xappr_j = Xavg_j & \text{if } Xavg_j \in XS \\ X_i & | (X_i \in XS) \text{ and } (X_i < Xavg_j) \text{ and } (Xavg_j - X_i \text{ is minimal}) \\ & \text{if } Xavg_j \notin XS \end{cases} \dots \quad (3)$$

where  $Xappr_j$  is the value of the  $j$ th approximate average.

$$Y_j = \left\lfloor \frac{\sum_{i=j}^{j+n-1} (X_i + Xappr_j)}{n-1} \right\rfloor \dots \quad (4)$$

where  $Y_j$  is the value of the  $j$ th output data.

The computational system produces the output sequence according to the given input sequence. Each input and output data in the respective sequence is indexed. This index, in terms of hardware, is the relative time when the input data is given or the output data is ready. Thinking as a hardware designer, the approximate average is chosen from the last  $n$  input data which should be stored in the system. The system should be able to calculate the integral part of the real average of the last  $n$  input data first. And then if the integral part of

the real average equals to any one of the last  $n$  input data, the approximate average is simply the integral part. Else the approximate average is the one which is one of the last  $n$  input data whose value is smaller than and closest to the integral part of the real average. The above descriptions stated the desired operations as those defined by equations (1), (2), and (3).

After the approximate average is obtained, the output value can be calculated according to equation (4). First, the last  $n$  input value is added by the corresponding approximate average. And then they are summed up and divided by  $n-1$ . The output value is the quotient after division.

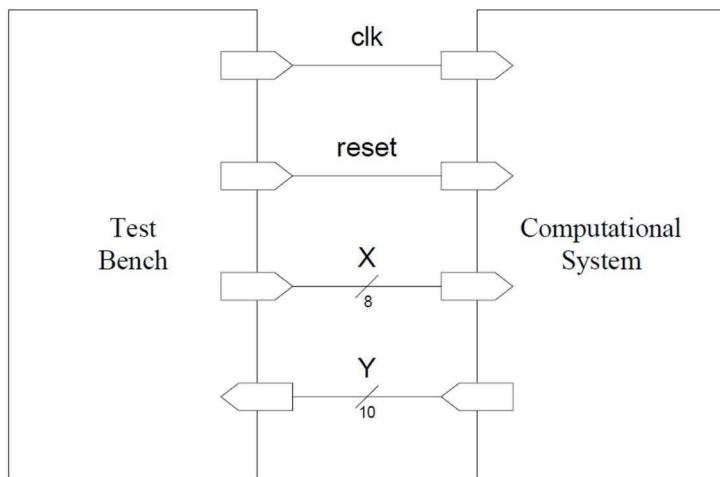
For example, assume that  $n=4$ ,  $X_1=3$ ,  $X_2=24$ ,  $X_3=16$ ,  $X_4=8$ , and  $X_5=3$ . After the first 5 input items are given, the system should store them and calculate the output value. The average of the first 4 input values is 12(only the integral part is left). Since it is not in the set of  $\{X_1, X_2, X_3, X_4\}$ , the system selects one from  $\{X_1, X_2, X_3, X_4\}$  as the approximate average whose value is smaller than 12 and close to 12. In this case, the approximate average is 8. So the first output value is calculated as

$$\lfloor [(3 + 8) + (24 + 8) + (16 + 8) + (8 + 8)] / [(4 - 1)] \rfloor = 27 .$$

Similar to those described above, when the 5th input data item is given, the system should store  $X_2$ ,  $X_3$ ,  $X_4$  and  $X_5$  and calculate the corresponding output value. The 2nd output value should be the same as the first one because the values stored in the system is the same.

## 2. Design Specifications

### 2.1 Block Overview

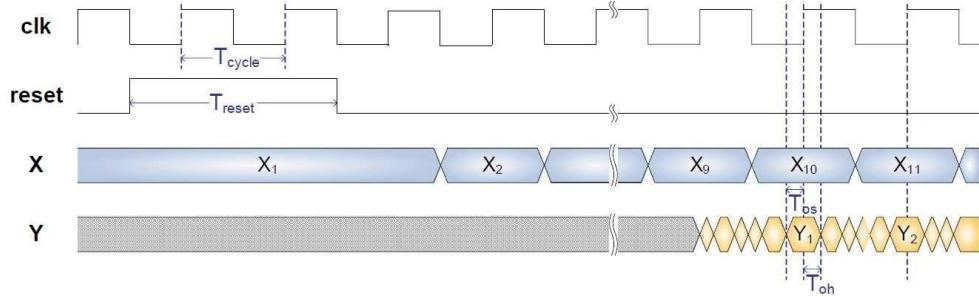


### 2.2 I/O Interface

| Signal Name | I/O | width | Description   |
|-------------|-----|-------|---|
| clk         | I   | 1     | clock for the computational system                          |
| reset       | I   | 1     | reset the state of the computational system when it asserts |

|   |   |    |  |
|---|---|----|--|
| X | I | 8  | input data of the computational system |
| Y | O | 10 | computed output                        |

### 2.3 Timing Diagrams



| Symbol      | Description  | Value         |
|-------------|--|---------------|
| $T_{cycle}$ | clock period   | user defined  |
| $T_{reset}$ | reset pulse width  | $2 T_{cycle}$ |
| $T_{os}$    | setup time from valid output to positive edge of <i>clk</i>  | 0.5ns         |
| $T_{oh}$    | hold time from positive edge of <i>clk</i> to invalid output | 0.5ns         |

The I/O timing diagram is as shown above. **For this homework,  $n$  in equations (1)(4) are fixed to 9.** The computational system is reset by asserting reset signal for 2 periods. The input X is changed to the next at the negative edges of the clock while output Y is checked by the test bench at positive edges of the clock. Note that the output should be stable around the positive edges of the clock. The setup and hold time requirements for the output are listed in Table. The first output data should be valid after the input data changes from 9th one to 10th and before the next positive clock edge. After that, the output should be changed to the next at the next positive clock edge and so on, that is to say, the test bench checks one output value per clock cycle.

### 2.4 File Description

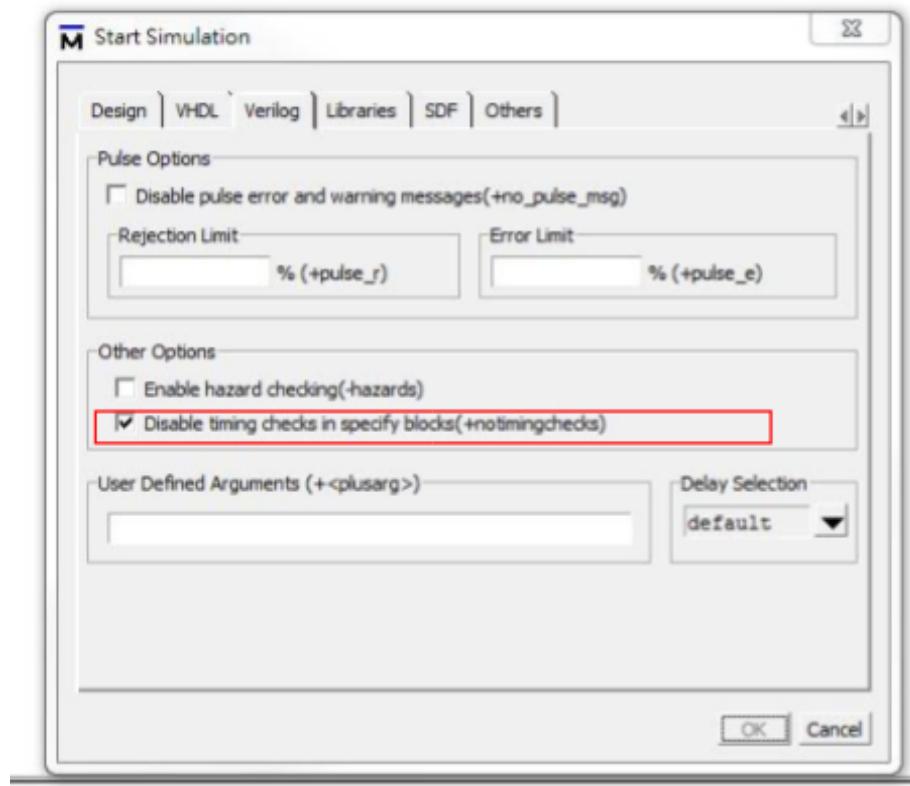
| File Name      | Description                     |
|----------------|---------------------------------|
| CS.v           | RTL code for using Verilog      |
| testfixture.v  | Test bench for verifying design |
| in.dat         | Input patterns                  |
| out_golden.dat | Golden output patterns          |

### 3. Scoring

#### 3.1 Functional Simulation (pre-sim) [100%]

All of the result should be generated correctly, and you will get the following message in ModelSim simulation. You can turn off the timing check in presim only.

```
VSIM 21> run -all
#
#
# -----
#
# All data have been generated successfully!
#
# -----PASS-----
#
#
#
# ** Note: $finish      : E:/2016DICHW/HW3/testfixture.v(122)
#           Time: 200400 ns Iteration: 2 Instance: /test
```



(Hexadecimal number)

| <i>index i</i> | $X_i$ | $Xavg_j$ | $Xappr_j$ | $\sum_{i=j}^{j+n-1} (X_i + Xappr_j)$ | $Y_j$ | <i>index j</i> |
|----------------|-------|----------|-----------|--------------------------------------|-------|----------------|
| 1              | 8f    |          |           |                                      |       |                |
| 2              | 0b    |          |           |                                      |       |                |
| 3              | 5d    |          |           |                                      |       |                |
| 4              | 20    |          |           |                                      |       |                |
| 5              | f3    |          |           |                                      |       |                |
| 6              | 3e    |          |           |                                      |       |                |
| 7              | e5    |          |           |                                      |       |                |
| 8              | 03    |          |           |                                      |       |                |
| 9              | 0c    | 5c       | 3e        | 056a                                 | 00ad  | 1              |
| 10             | 74    | 59       | 3e        | 054f                                 | 00a9  | 2              |
| 11             | 79    | 65       | 5d        | 06d4                                 | 00da  | 3              |
| 12             | 01    | 5b       | 3e        | 0561                                 | 00ac  | 4              |
| 13             | 30    | 5c       | 3e        | 0571                                 | 00ae  | 5              |
| 14             | 2e    | 46       | 3e        | 04ac                                 | 0095  | 6              |
| 15             | a4    | 52       | 30        | 0494                                 | 0092  | 7              |
| 16             | 76    | 45       | 30        | 0425                                 | 0084  | 8              |
| 17             | 84    | 54       | 30        | 04a6                                 | 0094  | 9              |
| 18             | 51    | 5b       | 51        | 0614                                 | 00c2  | 10             |
| 19             | d6    | 66       | 51        | 0676                                 | 00ce  | 11             |
| 20             | 70    | 65       | 51        | 066d                                 | 00cd  | 12             |
| 21             | 35    | 6b       | 51        | 06a1                                 | 00d4  | 13             |
| 22             | 10    | 68       | 51        | 0681                                 | 00d0  | 14             |
| 23             | 23    | 66       | 51        | 0676                                 | 00ce  | 15             |
| 24             | e7    | 6e       | 51        | 06b9                                 | 00d7  | 16             |
| 25             | 3b    | 67       | 51        | 067e                                 | 00cf  | 17             |
| 26             | 6d    | 65       | 51        | 0667                                 | 00cc  | 18             |
| 27             | 34    | 61       | 3b        | 0584                                 | 00b0  | 19             |
| 28             | 61    | 54       | 3b        | 050f                                 | 00a1  | 20             |
| 29             | 89    | 57       | 3b        | 0528                                 | 00a5  | 21             |
| 30             | bf    | 67       | 61        | 0708                                 | 00e1  | 22             |
| 31             | dc    | 7d       | 6d        | 0840                                 | 0108  | 23             |
| 32             | d3    | 91       | 89        | 09ec                                 | 013d  | 24             |
| 33             | 9c    | 88       | 6d        | 08a5                                 | 0114  | 25             |
| 34             | 8f    | 92       | 8f        | 0a2b                                 | 0145  | 26             |

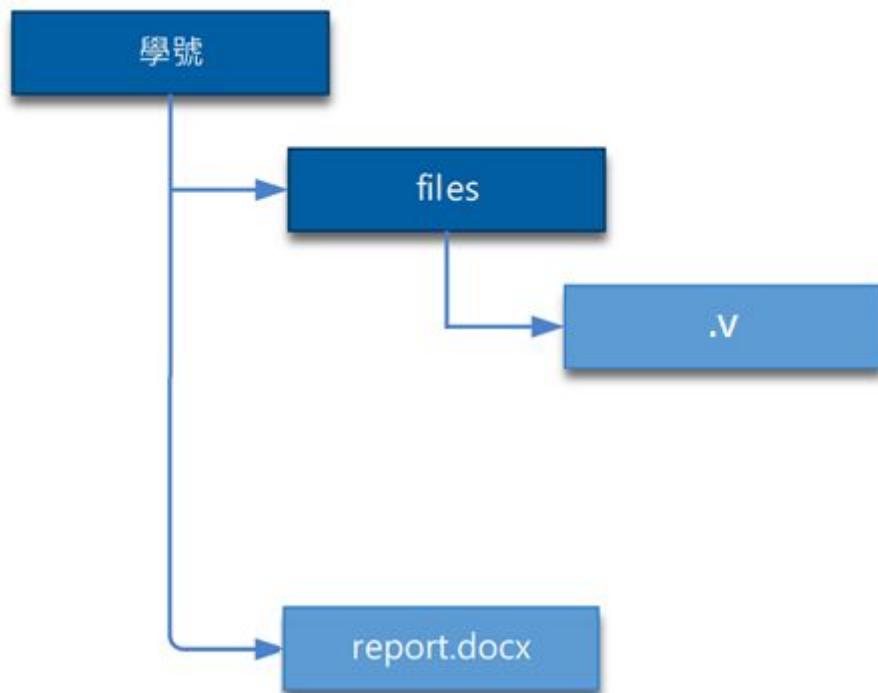
(Decimal number)

| <i>index i</i> | $X_i$ | $Xavg_j$ | $Xappr_j$ | $\sum_{i=j}^{j+n-1} (X_i + Xappr_j)$ | $Y_j$ | <i>index j</i> |
|----------------|-------|----------|-----------|--------------------------------------|-------|----------------|
| 1              | 143   |          |           |                                      |       |                |
| 2              | 11    |          |           |                                      |       |                |
| 3              | 93    |          |           |                                      |       |                |
| 4              | 32    |          |           |                                      |       |                |
| 5              | 243   |          |           |                                      |       |                |
| 6              | 62    |          |           |                                      |       |                |
| 7              | 229   |          |           |                                      |       |                |
| 8              | 3     |          |           |                                      |       |                |
| 9              | 12    | 92       | 62        | 1386                                 | 173   | 1              |
| 10             | 116   | 89       | 62        | 1359                                 | 169   | 2              |
| 11             | 121   | 101      | 93        | 1748                                 | 218   | 3              |
| 12             | 1     | 91       | 62        | 1377                                 | 172   | 4              |
| 13             | 48    | 92       | 62        | 1393                                 | 174   | 5              |
| 14             | 46    | 70       | 62        | 1196                                 | 149   | 6              |
| 15             | 164   | 82       | 48        | 1172                                 | 146   | 7              |
| 16             | 118   | 69       | 48        | 1061                                 | 132   | 8              |
| 17             | 132   | 84       | 48        | 1190                                 | 148   | 9              |
| 18             | 81    | 91       | 81        | 1556                                 | 194   | 10             |
| 19             | 214   | 102      | 81        | 1654                                 | 206   | 11             |
| 20             | 112   | 101      | 81        | 1645                                 | 205   | 12             |
| 21             | 53    | 107      | 81        | 1697                                 | 212   | 13             |
| 22             | 16    | 104      | 81        | 1665                                 | 208   | 14             |
| 23             | 35    | 102      | 81        | 1654                                 | 206   | 15             |
| 24             | 231   | 110      | 81        | 1721                                 | 215   | 16             |
| 25             | 59    | 103      | 81        | 1662                                 | 207   | 17             |
| 26             | 109   | 101      | 81        | 1639                                 | 204   | 18             |
| 27             | 52    | 97       | 59        | 1412                                 | 176   | 19             |
| 28             | 97    | 84       | 59        | 1295                                 | 161   | 20             |
| 29             | 137   | 87       | 59        | 1320                                 | 165   | 21             |
| 30             | 191   | 103      | 97        | 1800                                 | 225   | 22             |
| 31             | 220   | 125      | 109       | 2112                                 | 264   | 23             |
| 32             | 211   | 145      | 137       | 2540                                 | 317   | 24             |
| 33             | 156   | 136      | 109       | 2213                                 | 276   | 25             |
| 34             | 143   | 146      | 143       | 2603                                 | 325   | 26             |

# 作業繳交格式

本作業同學須完成電路設計與報告撰寫，請依照下圖的資料夾格式來繳交完成的檔案

1. 繳交檔名為**學號.zip**, 裡面會有一個自己學號的資料夾
2. 學號的資料夾中，會有一個**report.docx**和一個**files**的資料夾
3. **files**中會有所有的.v檔
4. 報告內容請參考**report.docx**



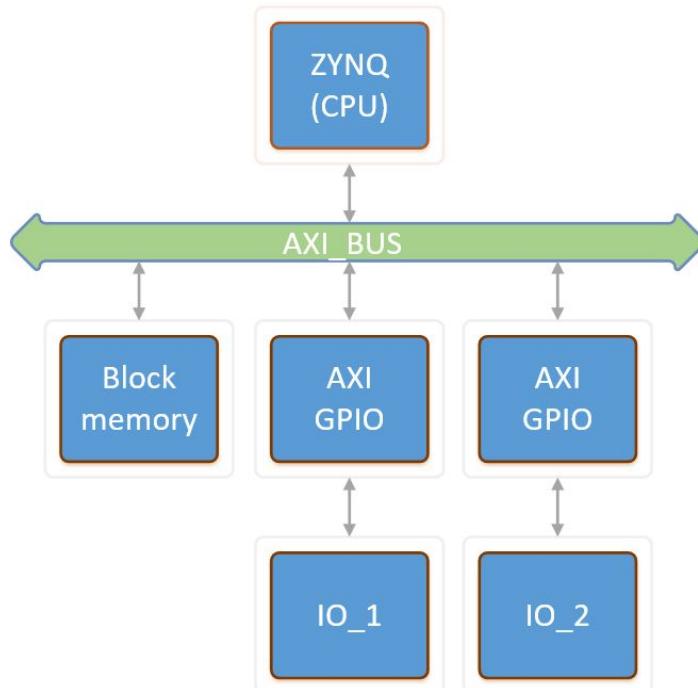
# LAB 3-1 : Vivado軟硬體開發環境教學

## 作業說明

本作業的目的為熟悉 Vivado 的開發環境，Vivado 為 Xilinx 公司的全新開發軟體，專用於 Xilinx FPGA 的軟硬體整合開發，同學需要根據以下步驟逐步完成作業，藉此來了解 Vivado 開發工具的使用，以及系統的 memory map IO 架構與寫法。

## Memory mapped IO 架構

pynq 內部的系統架構如下



ZYNQ processor 要對外使用 memory 或是 input/output (IO)，都要透過 AXI\_BUS 來傳輸資料，access IO 的方式就是直接操作 IO 對應的 memory 區段，這種操作方式及稱為 memory mapped IO (MMIO)，操作方式大致如下：

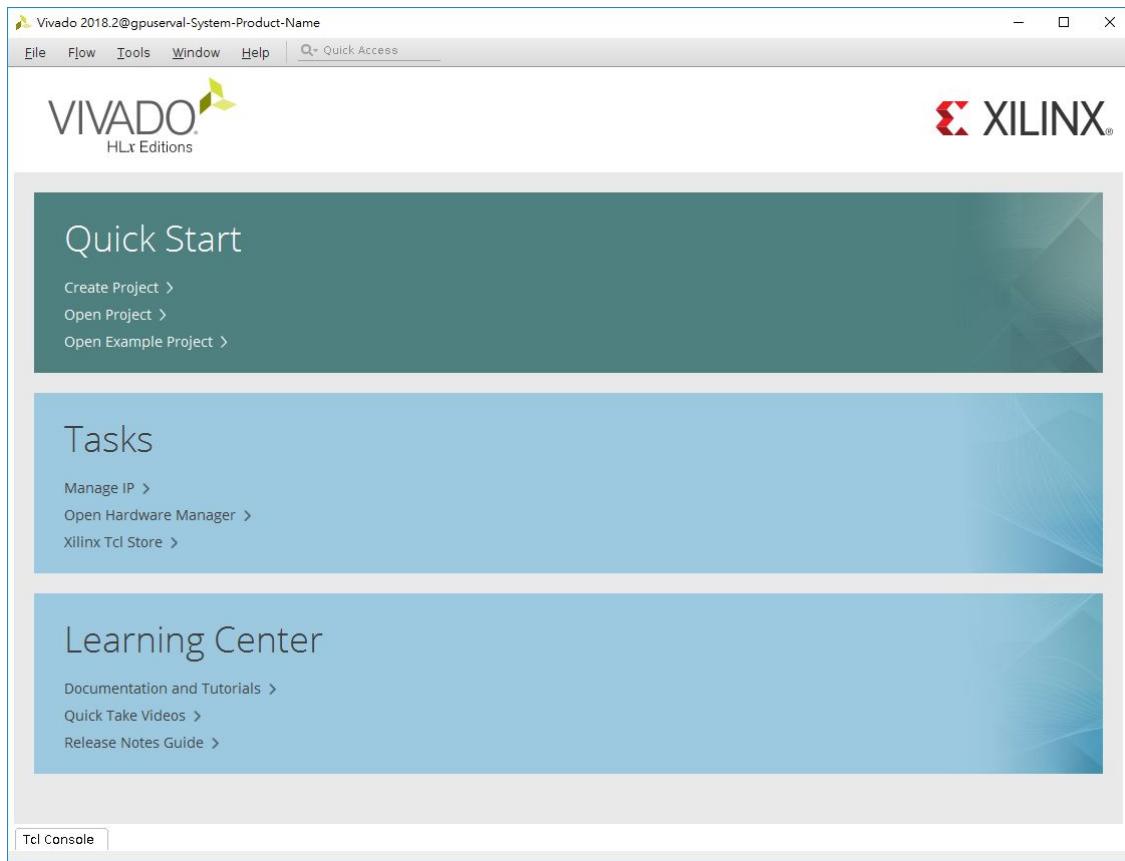
已知PYNQ各IO與memory對應的address如下

|              |                 |
|--------------|-----------------|
| ZYNQ         | 0x0000 ~ 0x0FFF |
| Block memory | 0x1000 ~ 0x1FFF |
| IO_1         | 0x2200 ~ 0x220F |
| IO_2         | 0x2300 ~ 0x23FF |

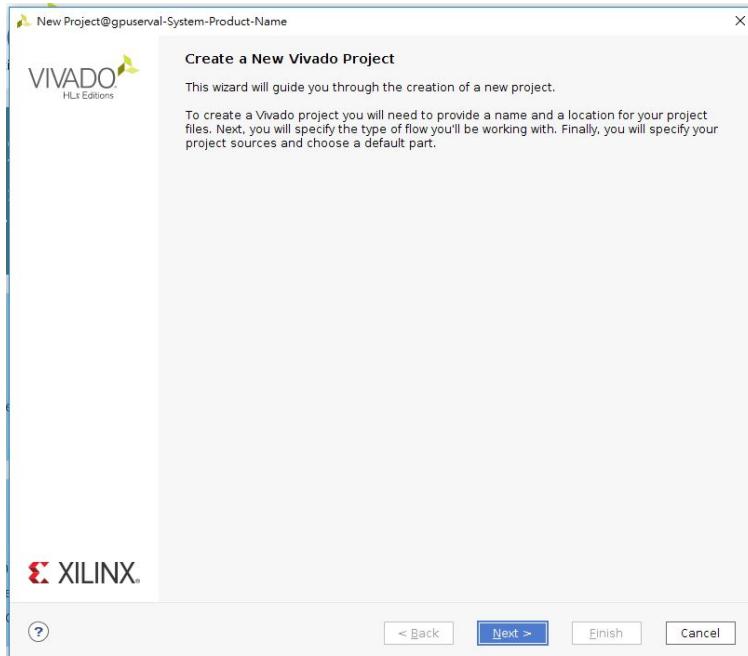
如果ZYNQ processor要去操作IO\_1，只要系統的AXI\_BUS建立好(硬體完成)，在軟體端(python jupyter)對記憶體位址0x2200~0x220F的資料作讀寫，就等於直接操作IO\_1，這就是memory mapped IO的操作方式。詳細MMIO的python code寫法請參考下面連結：  
[https://pynq.readthedocs.io/en/latest/pynq\\_libraries/mmio.html](https://pynq.readthedocs.io/en/latest/pynq_libraries/mmio.html)

## Vivado專案建置教學

1. 打開vivado後，即可看見下面的視窗

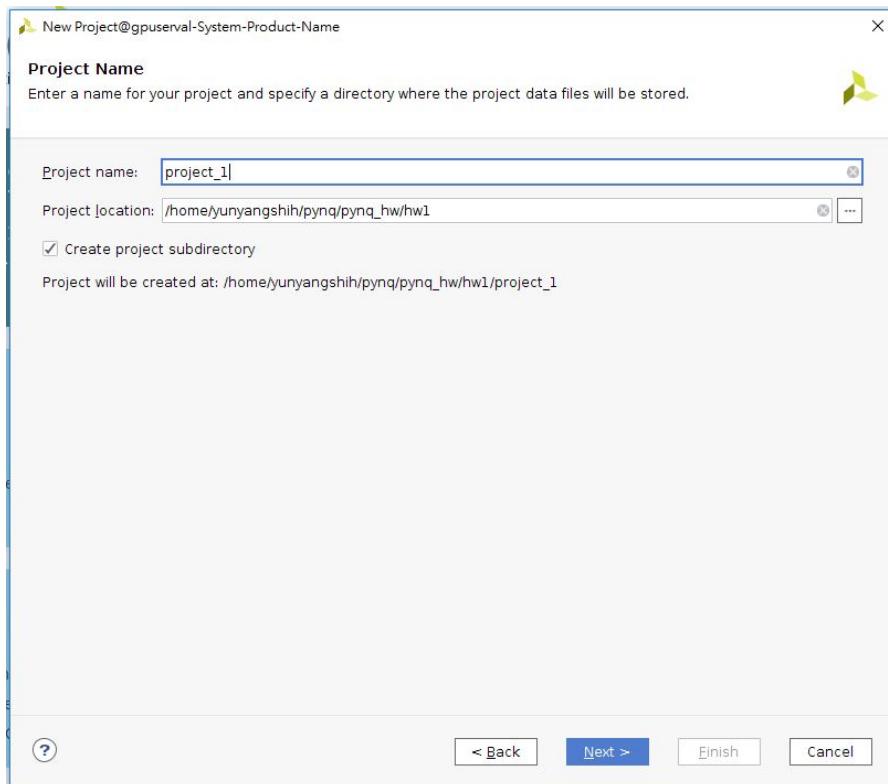


2. 選擇 "Create Project" 來創立新的專案，點選"Next >"



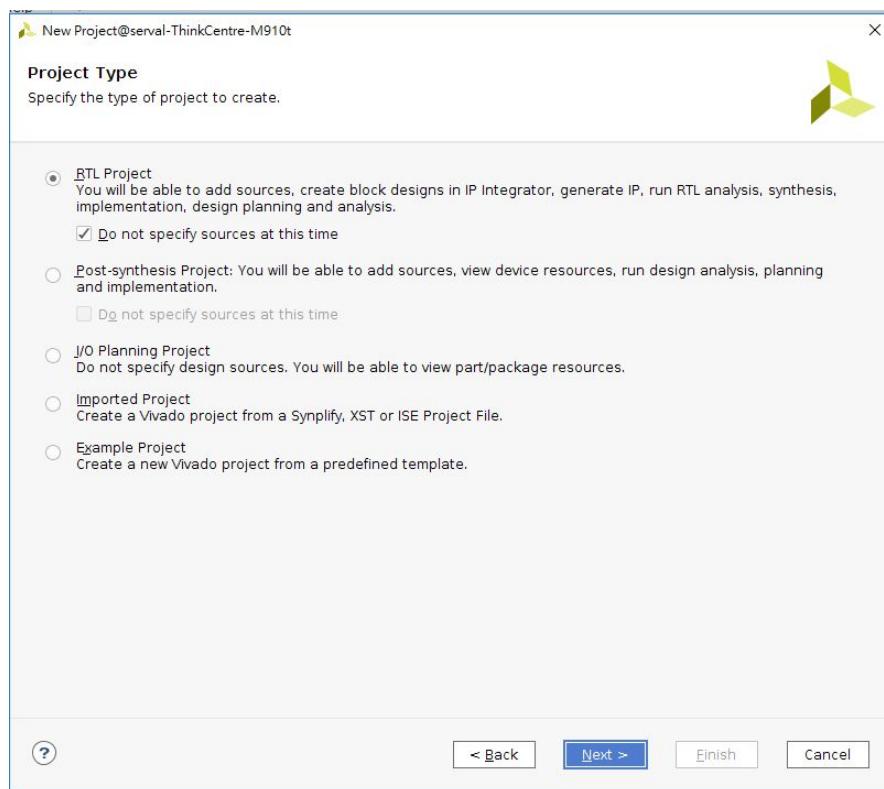
**專案名稱和路徑設定一下(可直接用默認的)**

如果"Create project subdirectory"打勾，會建立一個專案的專屬資料夾，請打勾

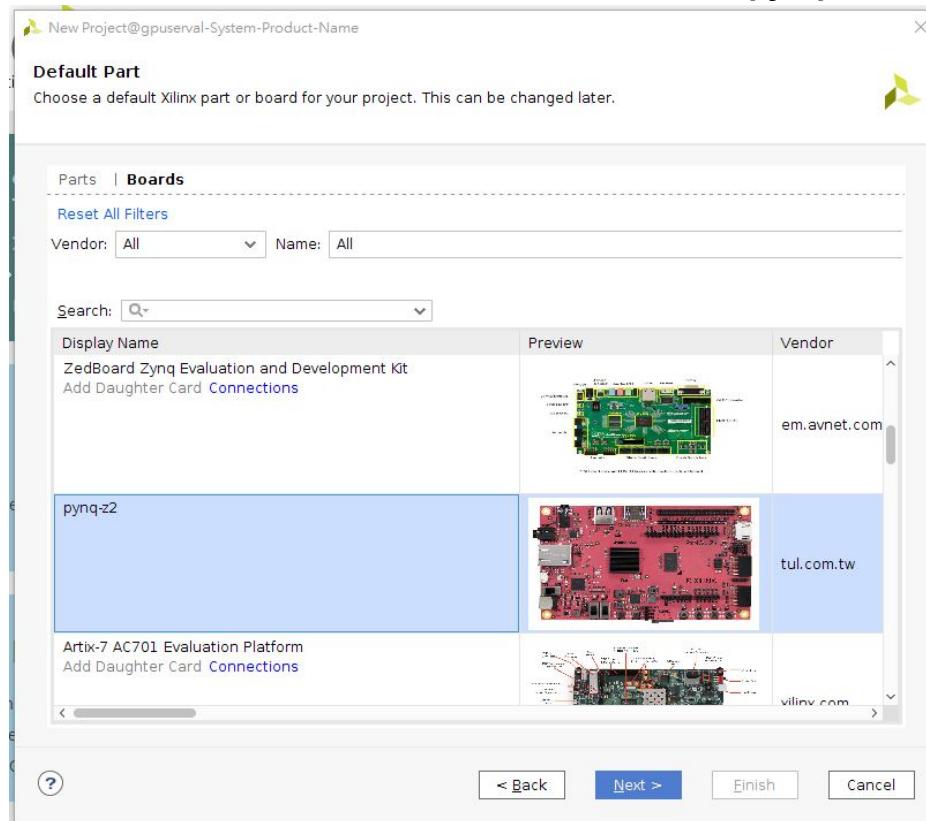


**選RTL Project**

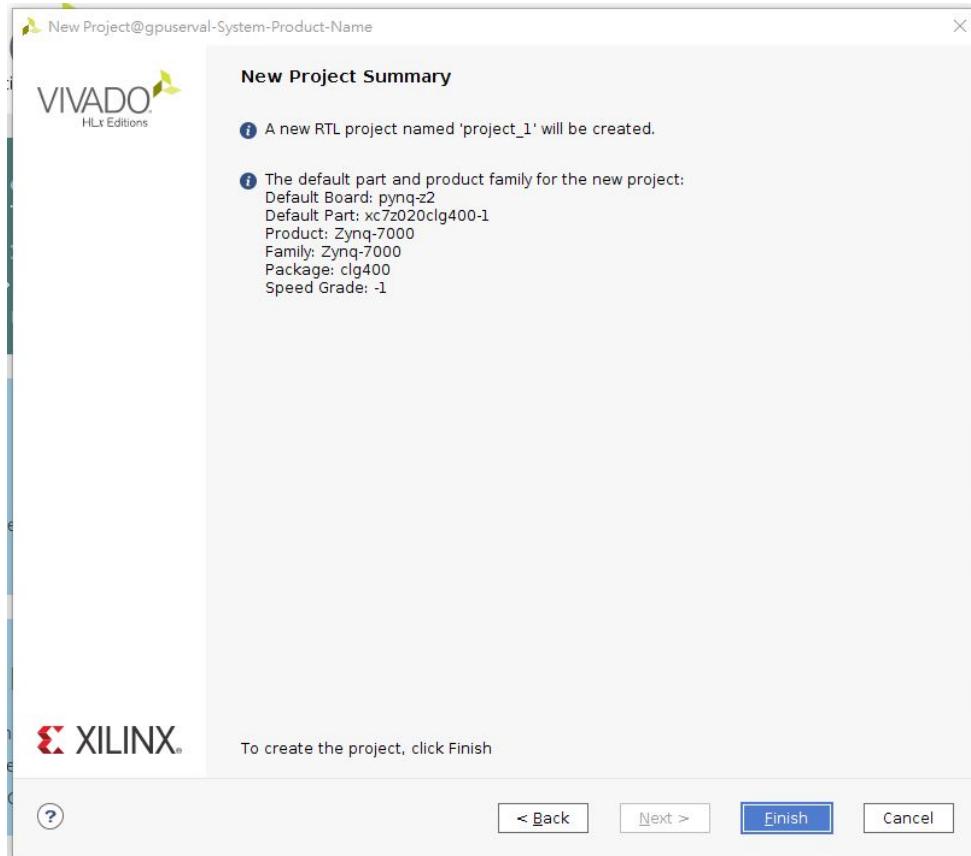
"Do not specify sources at this time"打勾 (如要載入code就不要勾)



選擇開發板，先點選畫面上方的Boards，然後點選 "pynq-z2"



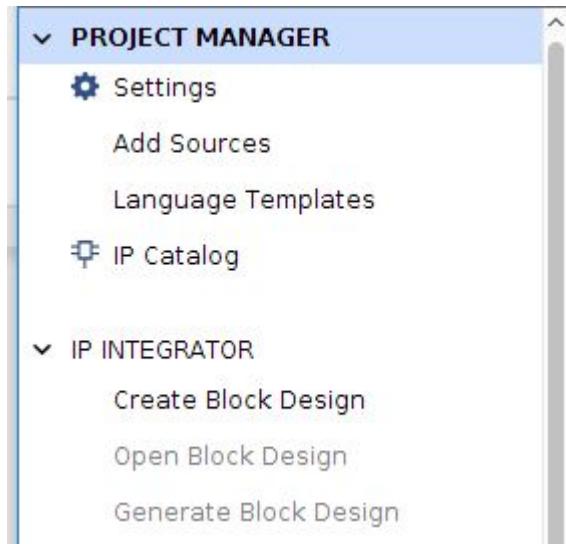
確認專案資訊無誤後，點 "Finish"



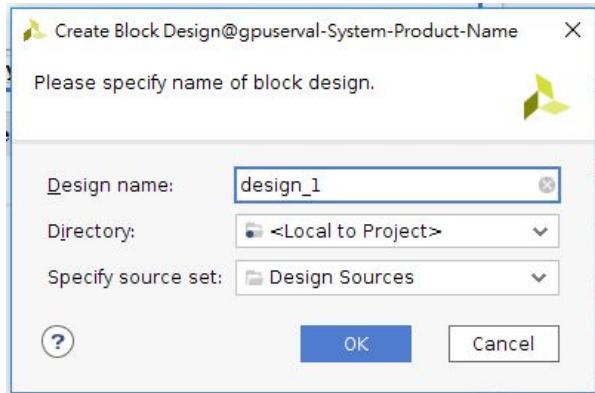
## Block design 介面使用教學

Block design 為Vivado的特色之一，使用者可以透過連接現成的或自製的IP，來生成對應的HDL code。以下為使用教學：

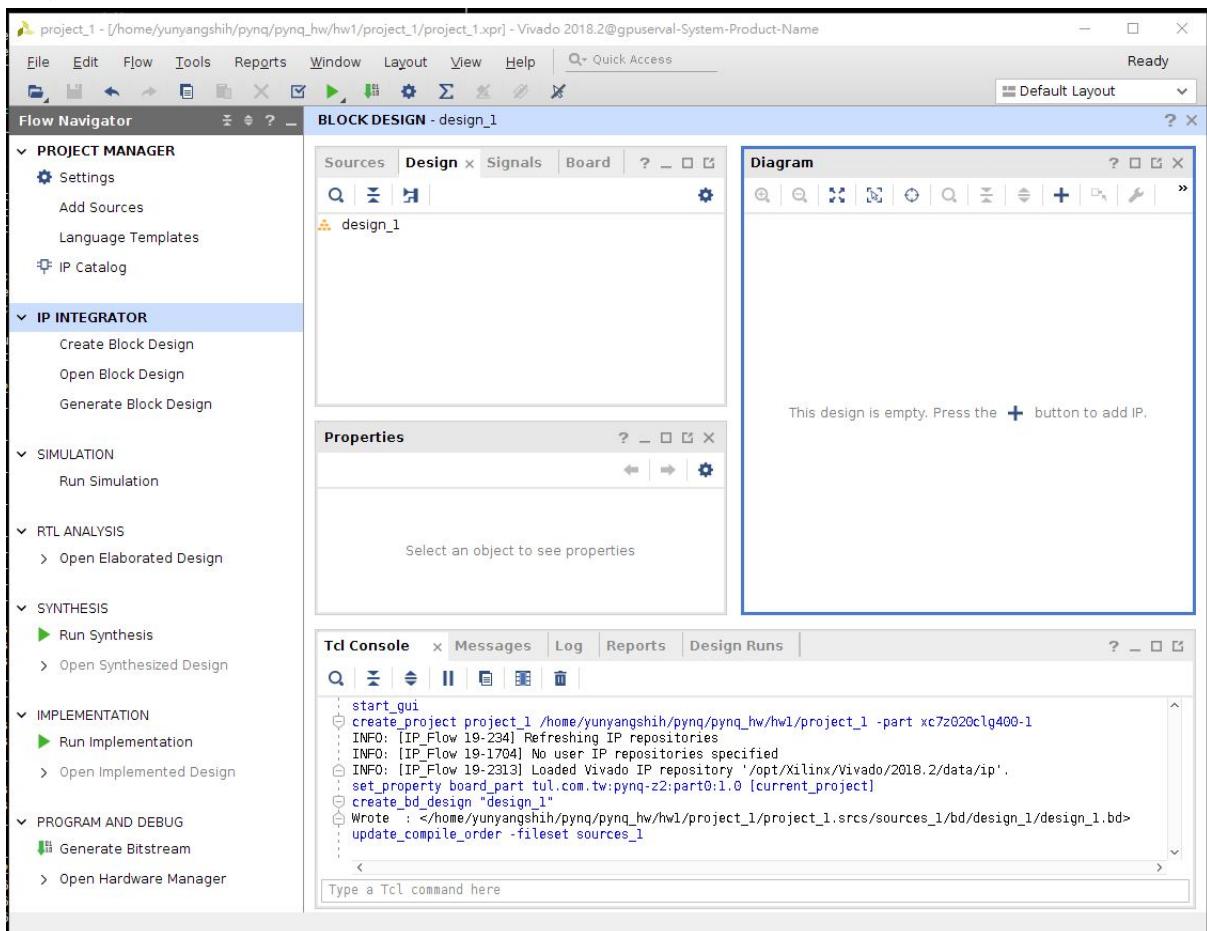
### 1. 點選專案頁面左側的" Create Block Design "



名稱和路徑都用默認的即可，點"OK"



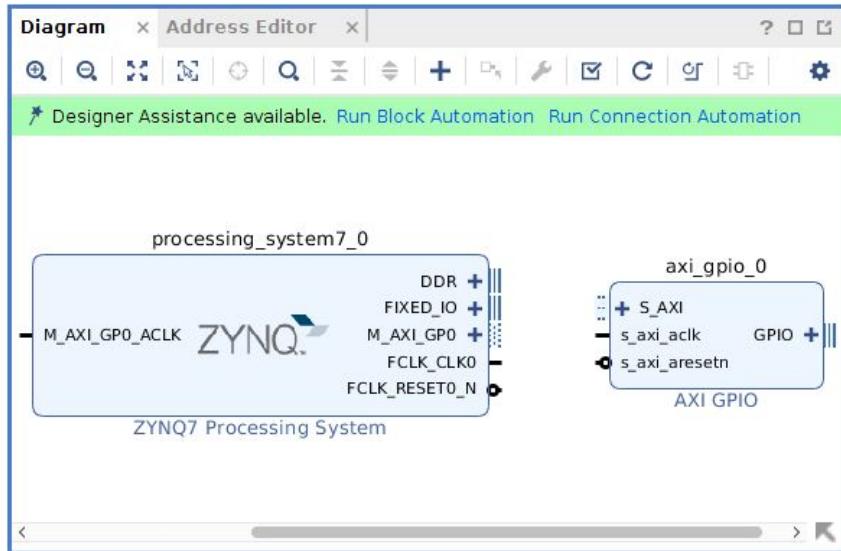
會顯示出以下畫面，可在畫面中點" + "任意加入IP



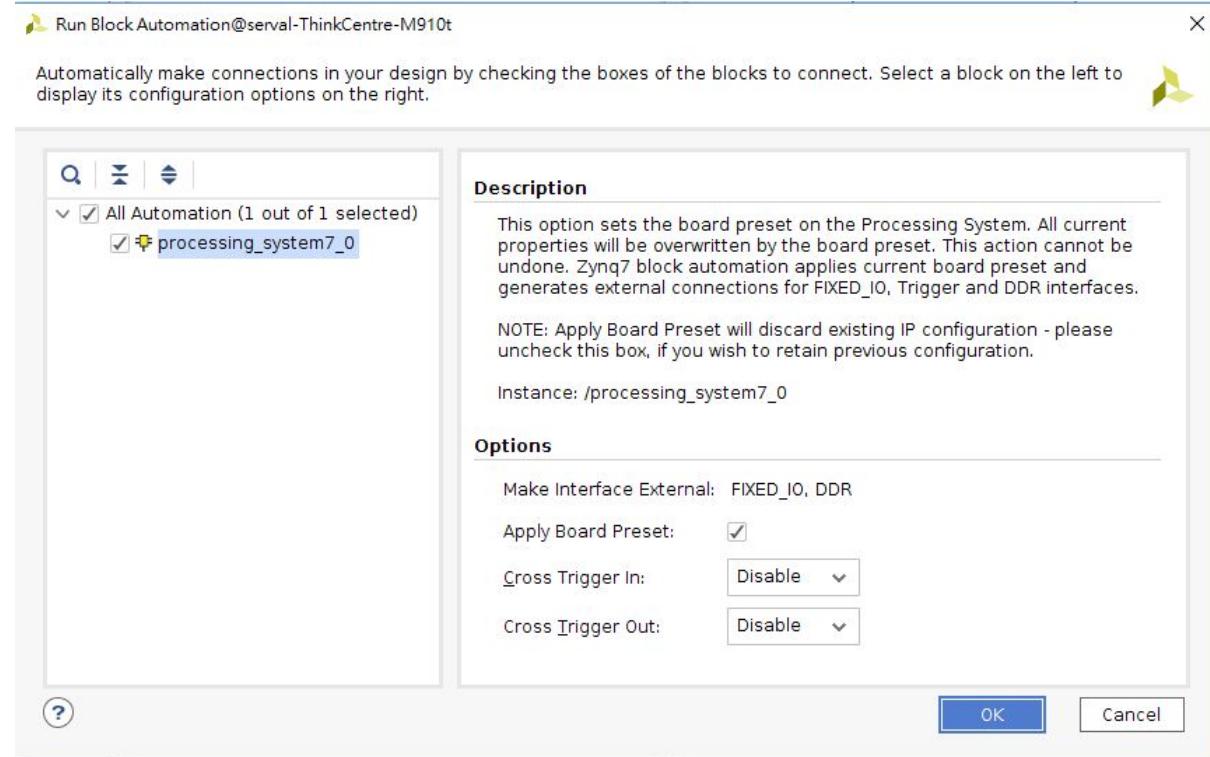
## 2. 用 + 加入" ZYNQ7 Processing System " 和 " AXI GPIO " , 共2個IP

ZYNQ為一個ARM的處理器，可設計其內部的軟體

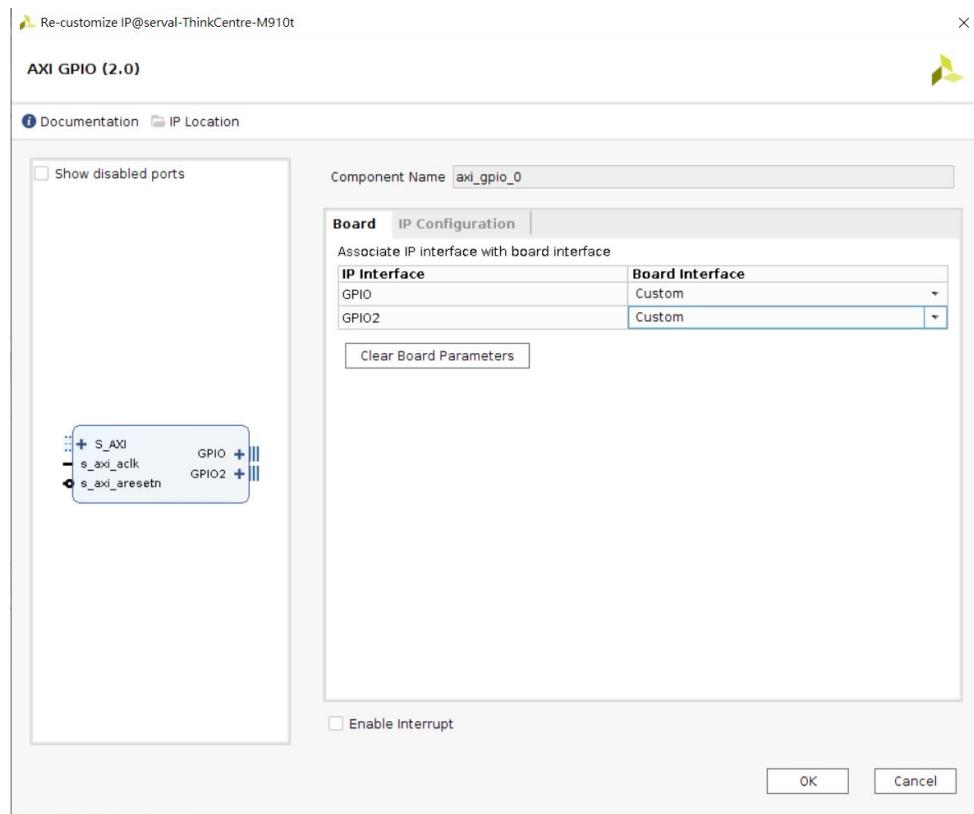
AXI GPIO會將GPIO的數位訊號轉換成AXI BUS的訊號



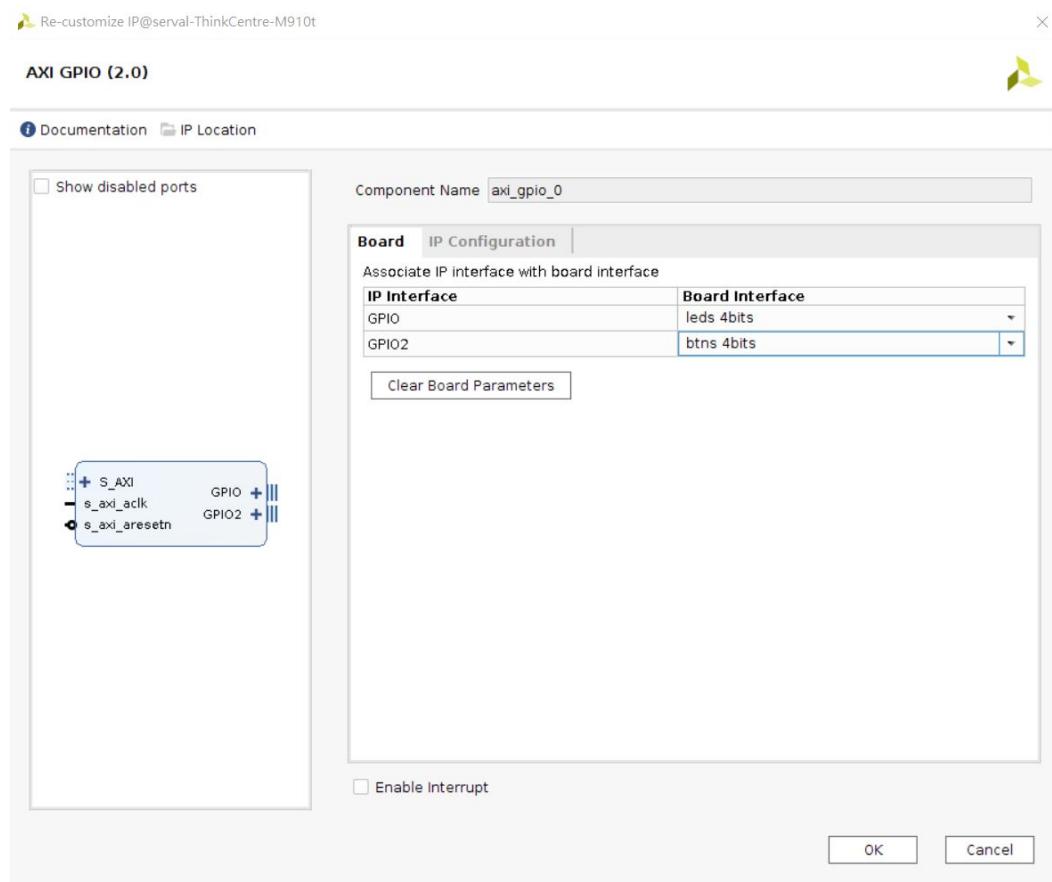
3. 點"Run Block Automation" 出現以下畫面，直接按OK，讓ZYNQ的DDR和FIXED\_IO接出去，軟體端才能正常對memory操作，



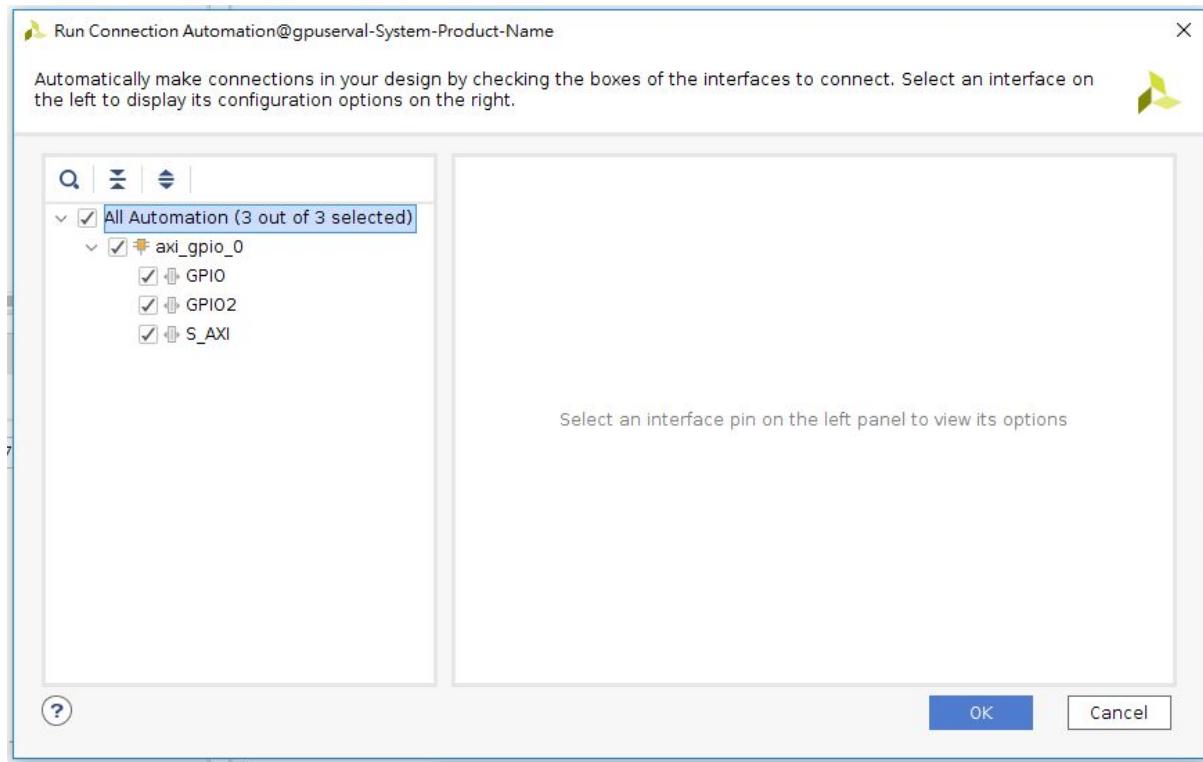
4. 在axi\_gpio\_0圖示上面點右鍵，點選"Customize Block"，即可設定IP參數



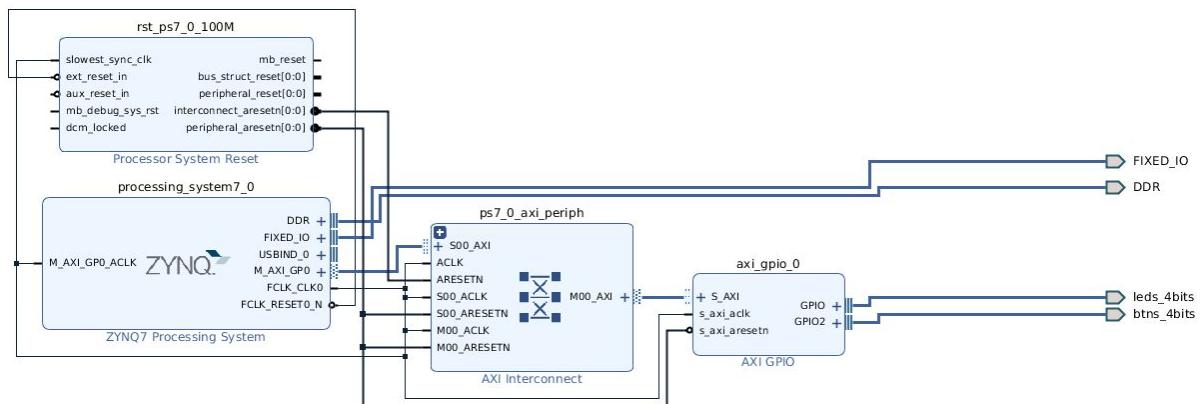
將GPIO設成leds 4bits, GPIO2設成btns 4bits, 然後按"OK"完成



5. 點" Run Connection Automation ", 此處全部打勾即可, Vivado會自動幫你接上AXI bus和reset, 如果以後的設計有多個AXI IP, 則需詳細設定。



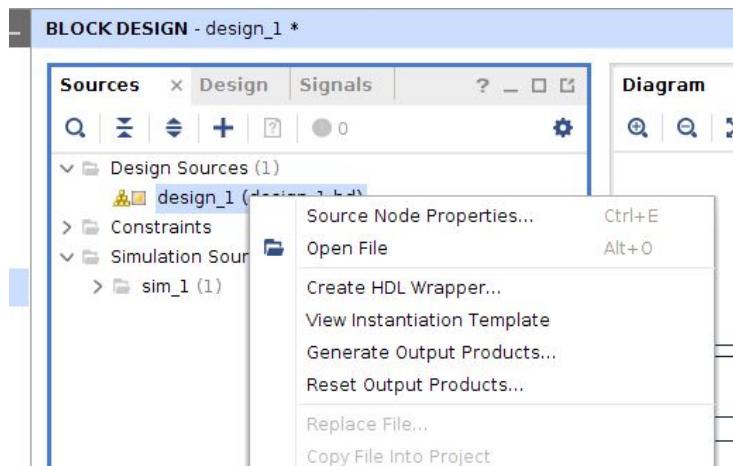
如此即完成一個GPIO訊號(硬體)透過AXI bus傳遞到ARM core(軟體)的電路  
IP的擺放位置可能與下圖不同, 但不影響電路功能



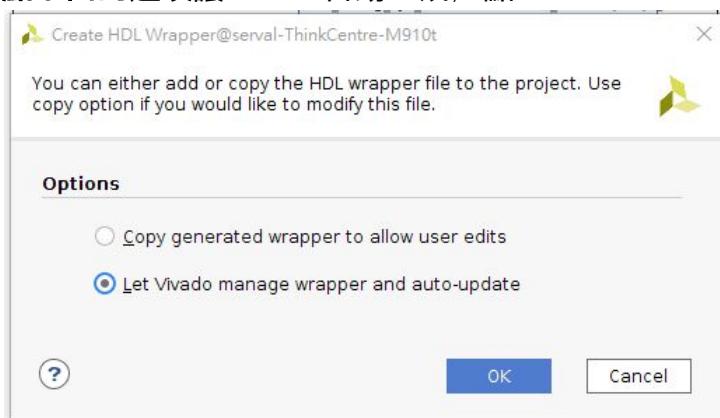
## 6. 按Ctrl + S儲存一下

根據前面設計的電路產生對應的HDL code

點左上角的Sources, 針對剛剛設計的block design(design\_1)點右鍵, 點 " Create HDL Wrapper "。



點下面的選項讓Vivado自動生成，點"OK"

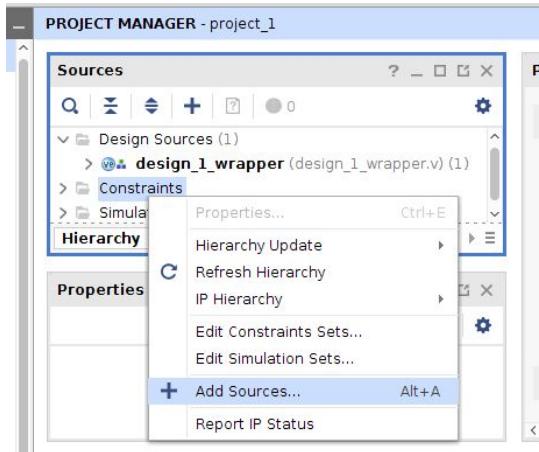


生成完畢，密密麻麻的verilog code

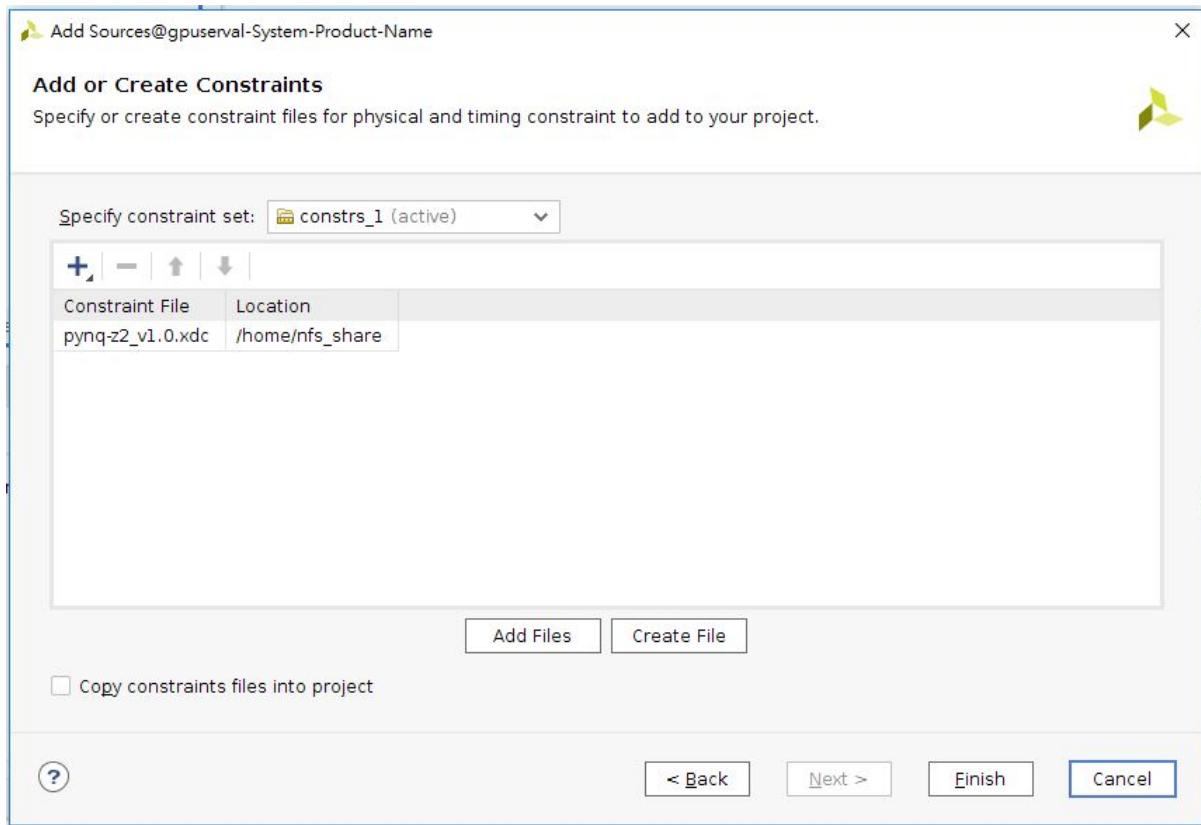
## 合成上板子

1. 完成硬體設計後，要先放入腳位檔(**pynq-z2\_v1.0.xdc**)

右鍵點選Vivado左上的Constraints資料夾，選擇ADD Sources

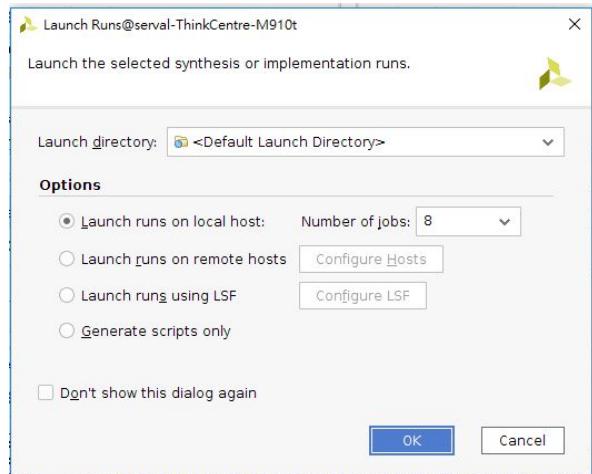


**選擇ADD or create constraints, 然後點"+"再選ADD Files加入pynq-z2\_v1.0.xdc  
(檔案在moodle上的HW3 files 壓縮檔中, 請同學自行下載)**

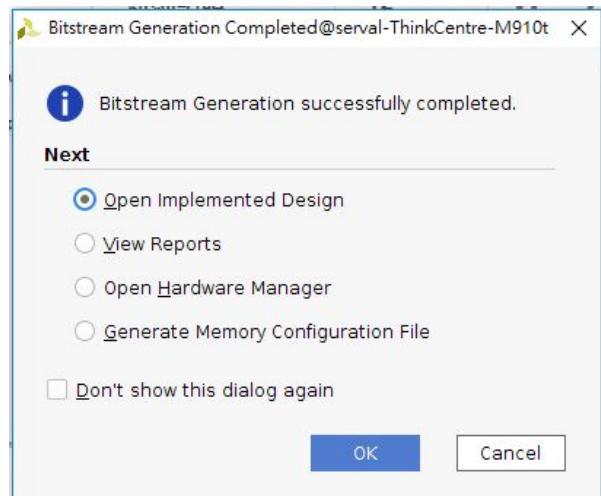


按"Finish"結束

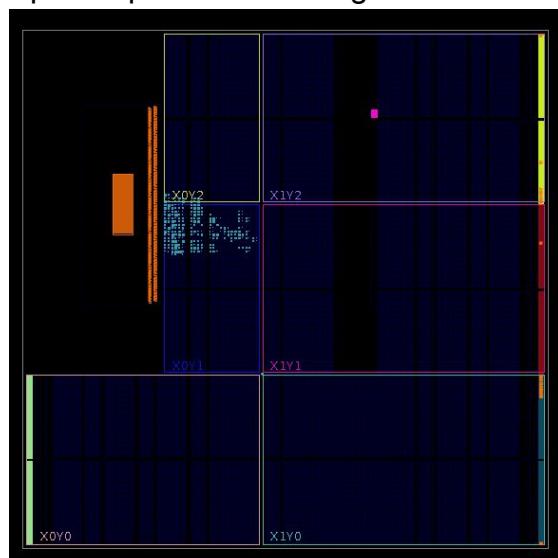
2. 點選左下方的"Generate Bitstream", 接下來Vivado會要你先跑合成, 按Yes, jobs會決定用幾個核心Run, 選完後OK



跑完後會出現以下畫面



Open Implemented Design 的資訊打開後的FPGA 使用情形

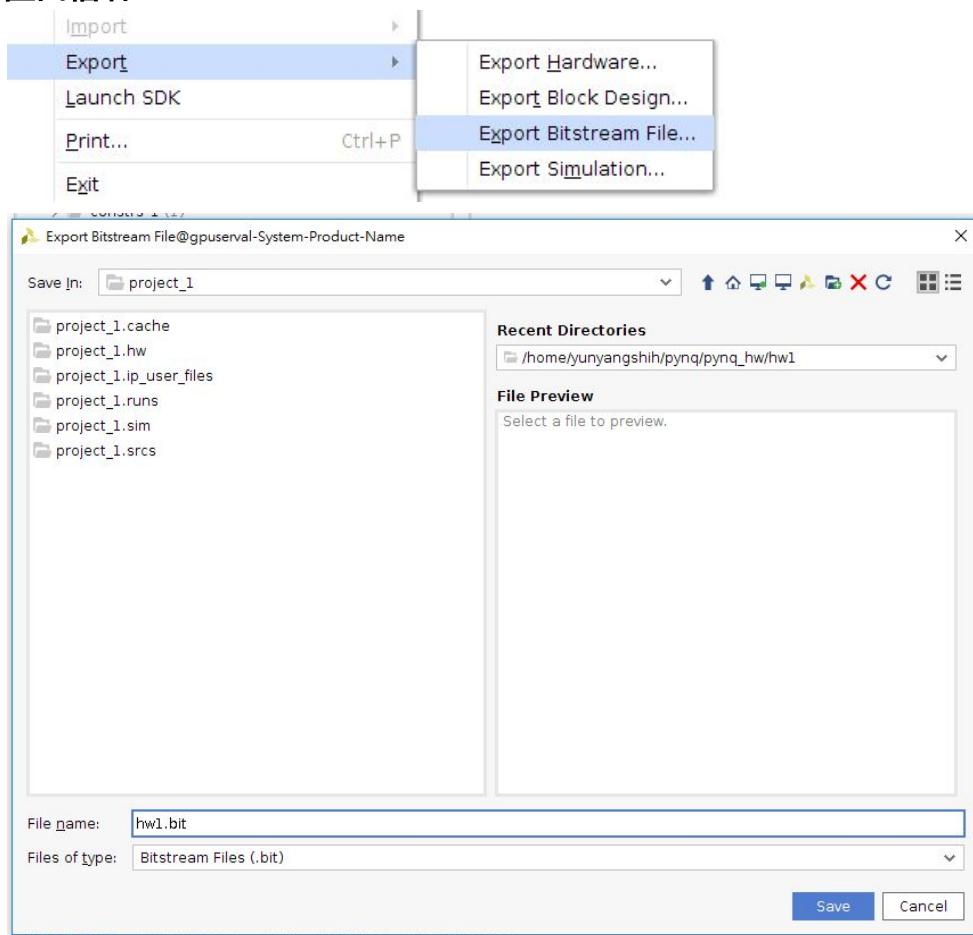


跑完後硬體燒錄檔(bitstream)已經產生，可以透過左邊的Open Synthesized Design看合成後的電路，Open Implemented Design可看FPGA資源使用等資訊

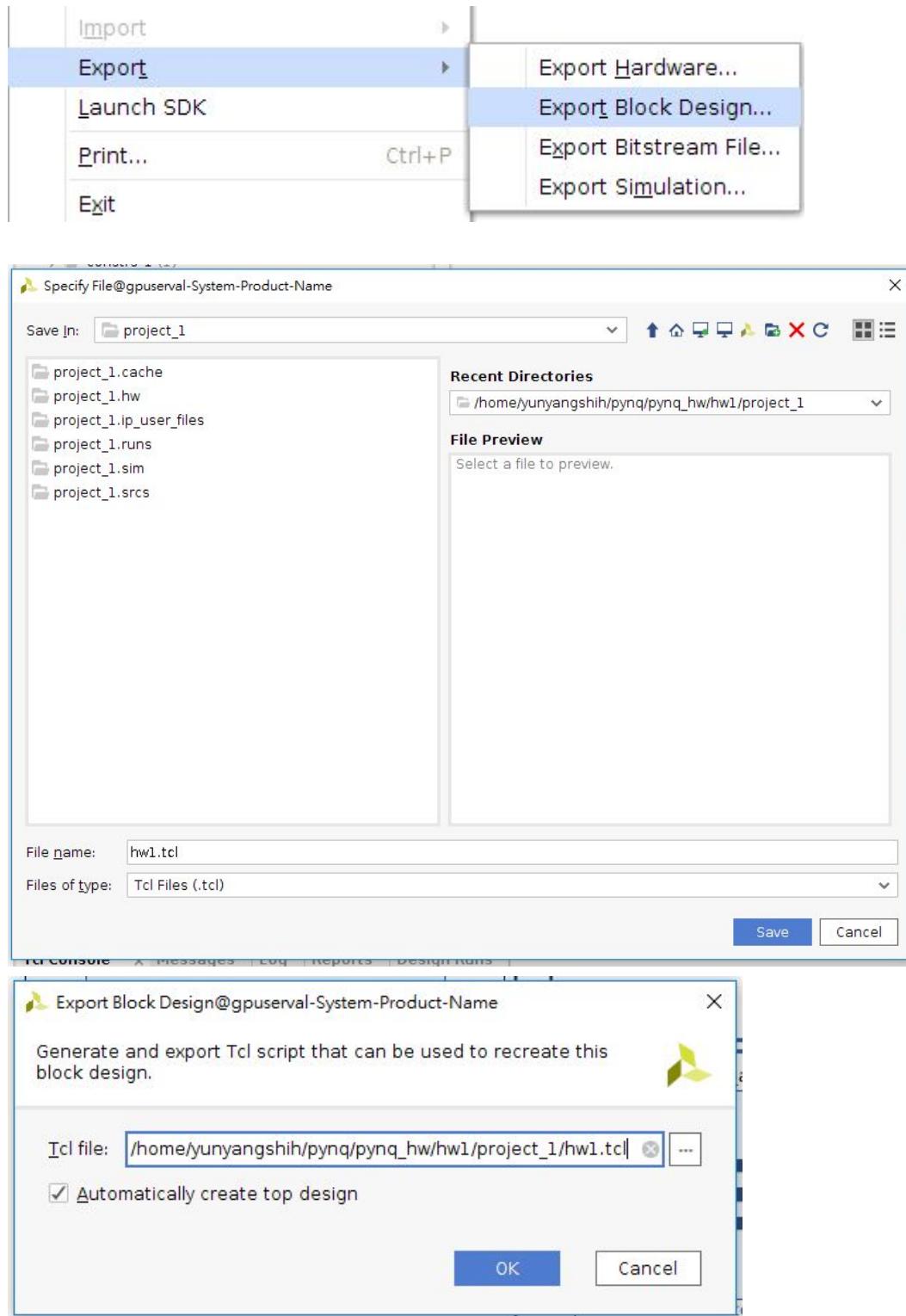
- ▼ SYNTHESIS
  - ▶ Run Synthesis
  - > Open Synthesized Design
  
- ▼ IMPLEMENTATION
  - ▶ Run Implementation
  - > Open Implemented Design
  
- ▼ PROGRAM AND DEBUG
  - ◀ Generate Bitstream
  - > Open Hardware Manager

## pynq jupyter使用準備

1. 點左上角的File -> Export -> Export bitstream匯出硬體bit檔(要先退出 Implemented Design)  
**匯出檔名 : hw3.bit**



**2. 點左上角的File -> Export -> Export block design匯出硬體tcl檔  
匯出檔名 : hw3.tcl**



**3. 將pynq的網路線接到筆電的網路  
或連接至同一個router  
用ssh連線到pynq的OS**

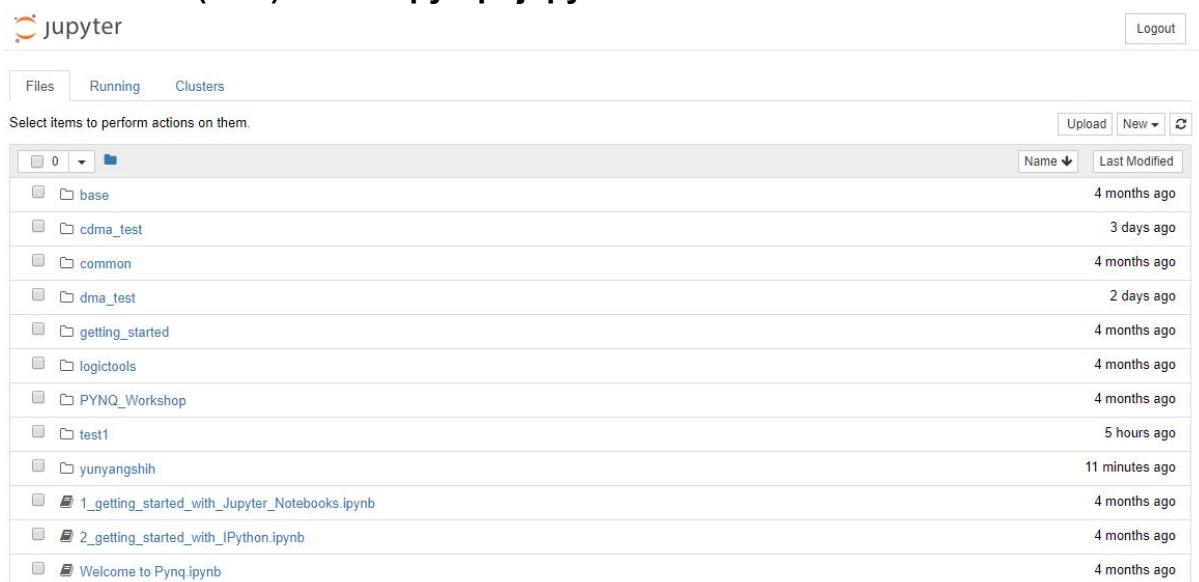
然後在jupyter\_notebooks資料夾中建一個hw3的資料夾

4. 用scp或其他方式，將你剛剛匯出的hw3.bit 和hw3.tcl檔搬到hw3中

## pynq jupyter使用教學

!!!!注意!!!!：pynq同時只能燒入一個硬體，所以不要兩個人同時使用，使用前請先通知其他使用者

1. 打開瀏覽器(任意)，連線至pynq的jupyter



The screenshot shows the Jupyter Notebook interface with the 'Files' tab selected. On the left is a file tree showing a directory structure with several notebooks and other files. On the right is a list of files with their names, last modified times, and download icons.

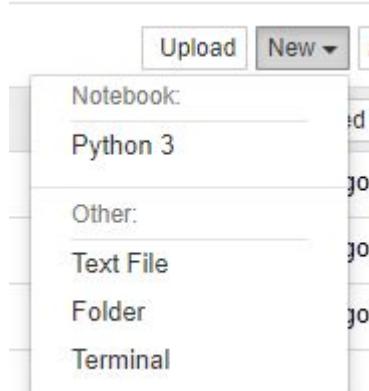
| Name   | Last Modified  |
|--|----------------|
| base   | 4 months ago   |
| cdma_test                                      | 3 days ago     |
| common   | 4 months ago   |
| dma_test                                       | 2 days ago     |
| getting_started                                | 4 months ago   |
| logictools                                     | 4 months ago   |
| PYNQ_Workshop                                  | 4 months ago   |
| test1  | 5 hours ago    |
| yunyangshih                                    | 11 minutes ago |
| 1_getting_started_with_Jupyter_Notebooks.ipynb | 4 months ago   |
| 2_getting_started_with_IPython.ipynb           | 4 months ago   |
| Welcome to Pynq.ipynb                          | 4 months ago   |

2. 點開 "自己的資料夾" -> hw3，會看到你剛剛放好的兩個檔案

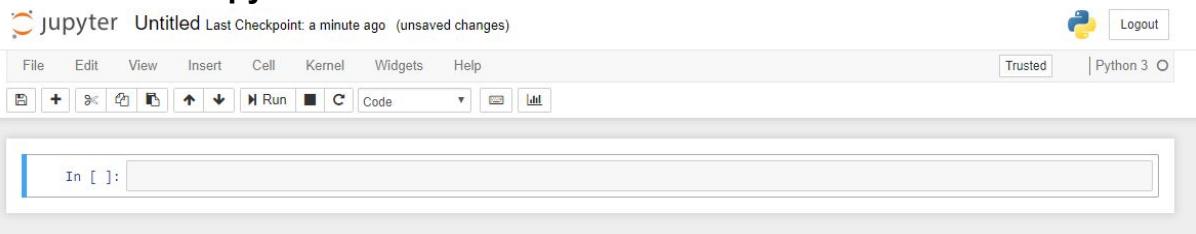


The screenshot shows the Jupyter Notebook interface with the 'Files' tab selected. A specific folder named 'hw3' is expanded, showing its contents: an empty directory, a file named 'hw1.bit', and a file named 'hw1.tcl'. The 'hw1' folder is also expanded.

3. 點右上角的New -> Python 3



#### 就會創立一個 .ipynb 檔



#### 4. 第一個cell放入以下code，此code的目的為重新燒入硬體檔

```
from pynq import Overlay  
axi_gpio_design = Overlay("./hw3.bit")
```

執行結果不會顯示任何東西

#### 5. 下一個cell放入以下code，來查看ZYNQ對應到的AXI slave，上方的 "+" 按鈕可增加cell

```
axi_gpio_design.ip_dict
```

執行結果：

```
In [2]: axi_gpio_design.ip_dict  
  
Out[2]: {'axi_gpio_0': {'addr_range': 65536,  
                      'driver': pynq.lib.axigpio.AxiGPIO,  
                      'fullpath': 'axi_gpio_0',  
                      'gpio': {},  
                      'interrupts': {},  
                      'phys_addr': 1092616192,  
                      'state': None,  
                      'type': 'xilinx.com:ip:axi_gpio:2.0'}}}
```

由此可以看到剛剛建立的axi\_gpio\_0有被ZYNQ看到，證明硬體有燒入成功

#### 6. 下一個cell放入以下code，來查看AXI slave的address

```
gp0_address = axi_gpio_design.ip_dict['axi_gpio_0']['phys_addr']
print("Physical address of gp0: 0x" + format(gp0_address, '02x'))
```

執行結果：

```
gp0_address = axi_gpio_design.ip_dict['axi_gpio_0']['phys_addr']
print("Physical address of gp0: 0x" + format(gp0_address, '02x'))
```

```
Physical address of gp0: 0x41200000
```

可以看到axi\_gpio\_0的操作address為 0x41200000

7.下一個cell放入以下code，來算出led和button的操作address，由於pynq的每個slave register的大小為2個word，所以第一個gpio和第二個gpio的byte address差了8

```
led_addr = gp0_address + 0
btn_addr = gp0_address + 8
```

8.下一個cell放入以下code，導入MMIO後，讓leds和buttons變成可以操作該address的io變數。

```
from pynq import MMIO
leds = MMIO(led_addr, 8)
buttons = MMIO(btn_addr, 8)
```

9.下一個cell放入以下code，即可讀取到pynq右下方按鈕的狀態

```
print(f"buttons: {buttons.read()}")
```

執行結果：

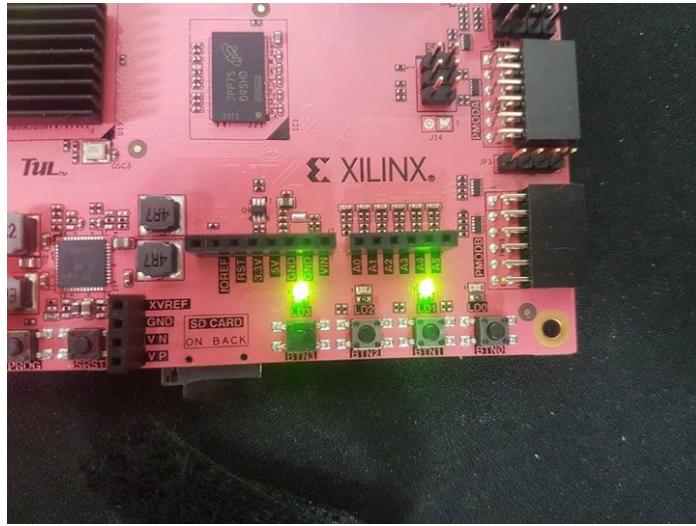
```
print(f"buttons: {buttons.read()}")
```

```
buttons: 0
```

10.下一個cell放入以下code，即可寫入pynq右下方的LED，要寫入時，必須先將該register的write tri-state buffer(也就是address = base adress + 4 的地方)寫成0，才能寫入資料

```
leds.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to output
leds.write(0x0, 0xa) #1010
```

執行結果：



## 整個ipynb如下：

```
from pynq import Overlay
axi_gpio_design = Overlay("./hw1.bit")

axi_gpio_design.ip_dict

{'axi_gpio_0': {'addr_range': 65536,
 'driver': pynq.lib.axigpio.AxiGPIO,
 'fullpath': 'axi_gpio_0',
 'gpio': {},
 'interrupts': {},
 'phys_addr': 1092616192,
 'state': None,
 'type': 'xilinx.com:ip:axi_gpio:2.0'}}
```

```
gp0_address = axi_gpio_design.ip_dict['axi_gpio_0']['phys_addr']
print("Physical address of gp0: 0x" + format(gp0_address, '02x'))
```

```
Physical address of gp0: 0x41200000
```

```
led_addr = gp0_address + 0
btn_addr = gp0_address + 8
```

```
from pynq import MMIO
leds = MMIO(led_addr, 8)
buttons = MMIO(btn_addr, 8)
```

```
print(f"buttons: {buttons.read()}")
```

```
buttons: 0
```

```
leds.write(0x4, 0x0)# Write 0x0 to location 0x4; Set tri-state to output
leds.write(0x0, 0xa)#1010
```

# LAB 3-2 : Custom IP的建置與使用

## 作業說明

本作業需將硬體電路包成vivado的ip，以便放入軟硬整合的系統中，並完成對應的軟體設計來驗證系統的功能。本作業已提供硬體的verilog code，請同學根據下面的教學完成包裝ip的工作，並根據hw3-1的步驟與概念，來完成系統建置和對應jupyter軟體的撰寫。

## intellectual property (IP) 簡介

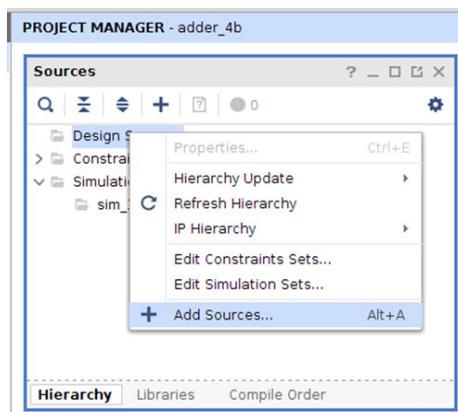
**矽智財**，全稱**智慧財產權核**（英語：**intellectual property core**），是在積體電路的可重用設計方法學中，指某一方提供的、形式為邏輯單元、晶片設計的可重用模組。矽智財通常已經通過了設計驗證，設計人員以矽智財為基礎進行設計，可以縮短設計所需的周期。

參考資料：

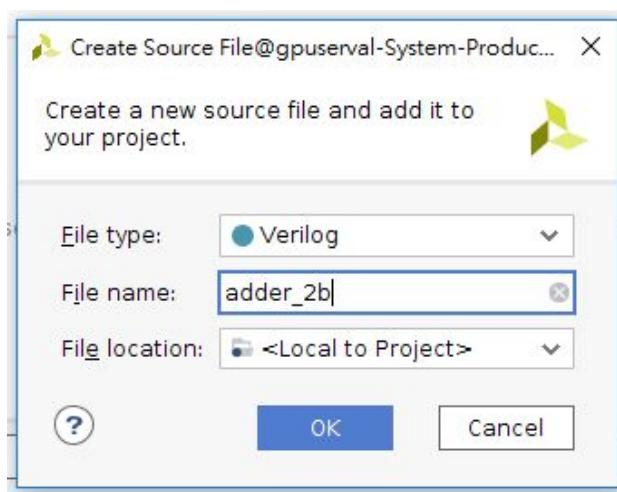
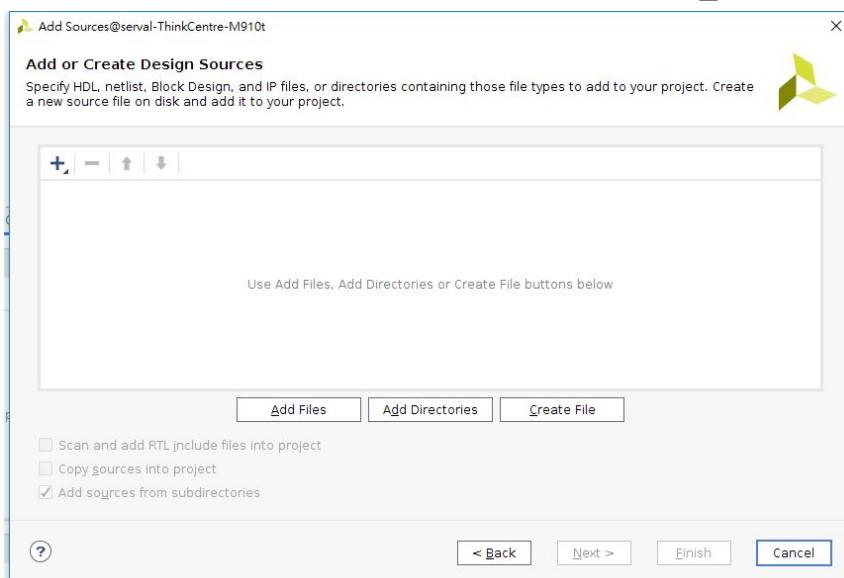
[https://en.wikipedia.org/wiki/Semiconductor\\_intellectual\\_property\\_core](https://en.wikipedia.org/wiki/Semiconductor_intellectual_property_core)  
<https://china.xilinx.com/products/intellectual-property.html>

## Custom IP建置

1. 創一個IP專用的資料夾(位置不限)，在裡面建置一個vivado專案，名稱取為  
**adder\_2b**(專案建置請參考hw3-1)
2. 對Design Sources按右鍵，選Add Sources，之後選Add or create design sources

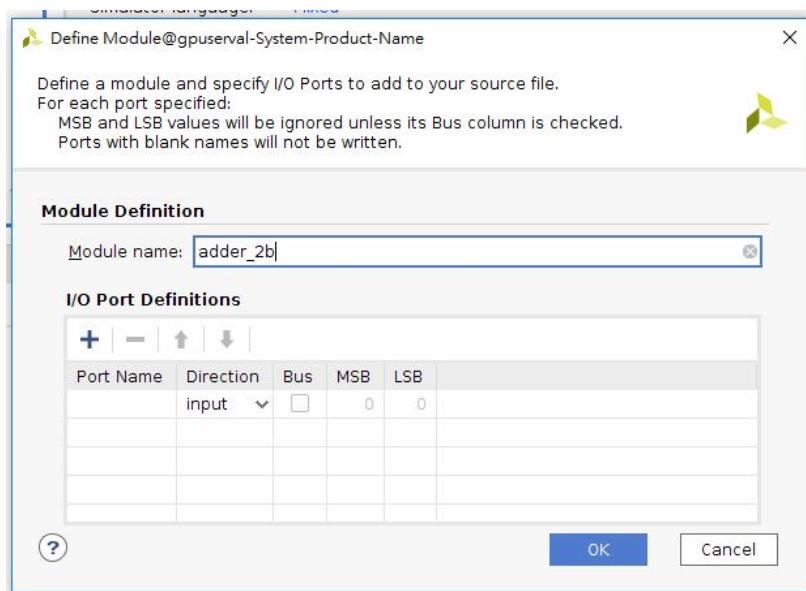


出現以下畫面，選Create File，名字一樣取adder\_2b



按OK，再按Finish 完成，會出現Define Module 畫面

Define Module直接點Cancel



在Design Sources中找到adder\_2b.v，左鍵點兩下打開，貼上以下程式碼  
此設計為用按鈕控制做加法或減法

```
module adder_2b(clk, rst, in_4b, sw0, out_4b);

input clk,rst;
input [3:0]in_4b;
input sw0;
output reg [3:0]out_4b;

always@(*)begin
if(sw0)
    out_4b = {2'b0, in_4b[3:2]} + {2'b0, in_4b[1:0]};
else
    out_4b = {2'b0, in_4b[3:2]} - {2'b0, in_4b[1:0]};

end

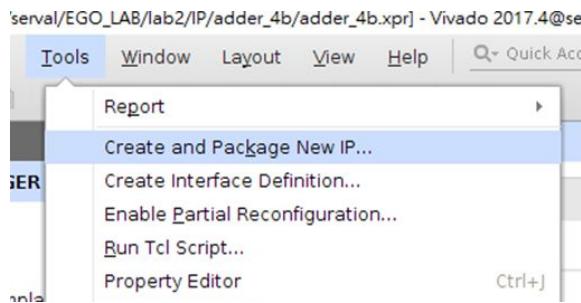
endmodule
```

此硬體用按鈕去控制加法器的功能，如果有自己想做的功能也可修改

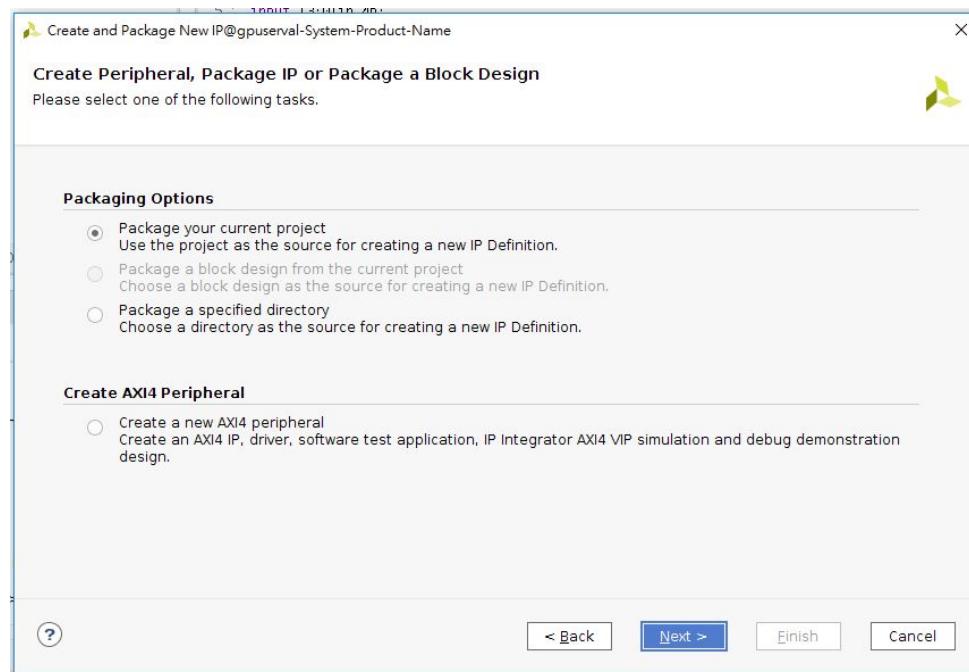
### 3. Ctrl + s儲存後，點左邊的”Run Synthesis”合成電路

- ▼ SYNTHESIS
  - ▶ Run Synthesis
  - > Open Synthesized Design

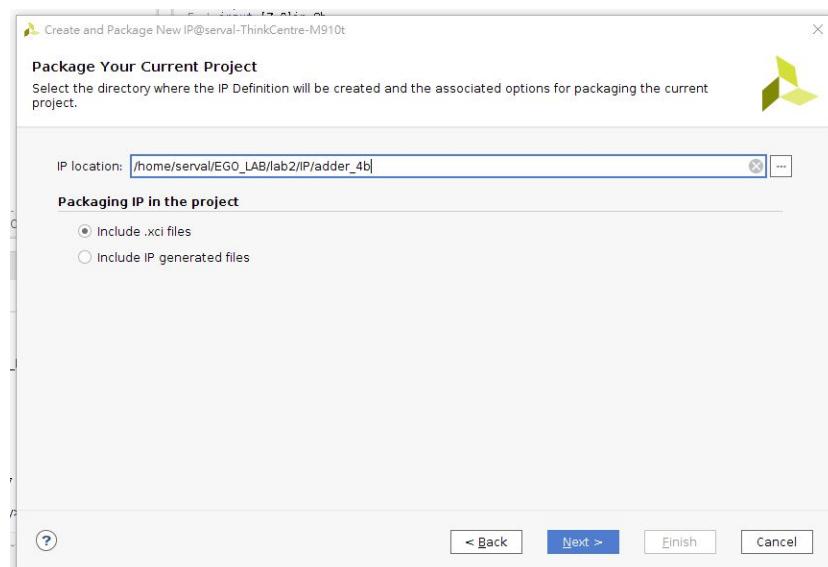
#### 4. 合成完後，選上方的Tools -> Create and Package New IP...



#### 點Next，選"Package your current project"



#### IP location選在專案的資料夾(默認也可以)，選"include .xci files"，點Next -> Finish



會出現以下畫面，Packaging Steps 可看到這顆IP的相關資訊，選最下面的"Review and Package"，按Package IP即可製成新的IP，完成後可關閉專案

Review and Package

1 warning 2 info messages

**Summary**

Display name: adder\_2b\_v1\_0  
Description: adder\_2b\_v1\_0  
Root directory: /home/yunyangshih/pynq/pynq\_hw/hw2/ip/adder\_2b

**After Packaging**

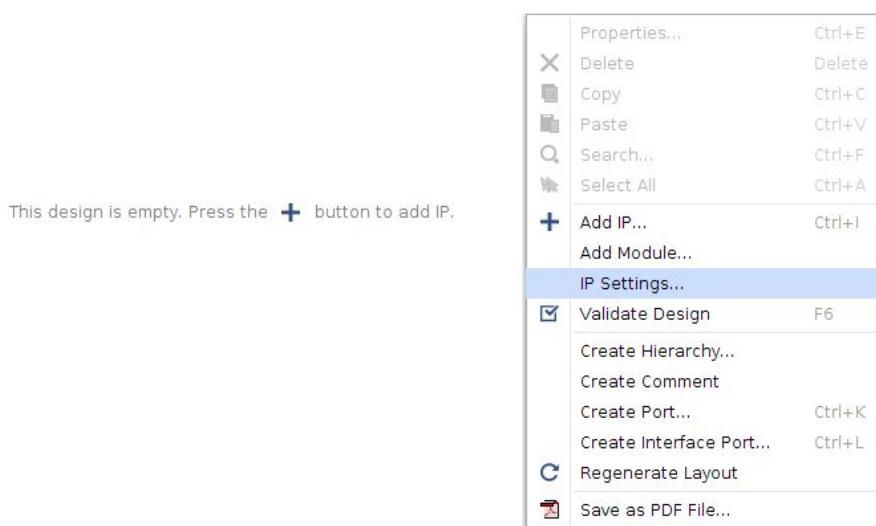
An archive will not be generated. Use the settings link below to change your preference  
IP will be made available in the catalog using the repository -  
/home/yunyangshih/pynq/pynq\_hw/hw2/ip/adder\_2b  
[Edit packaging settings](#)

[Package IP](#)

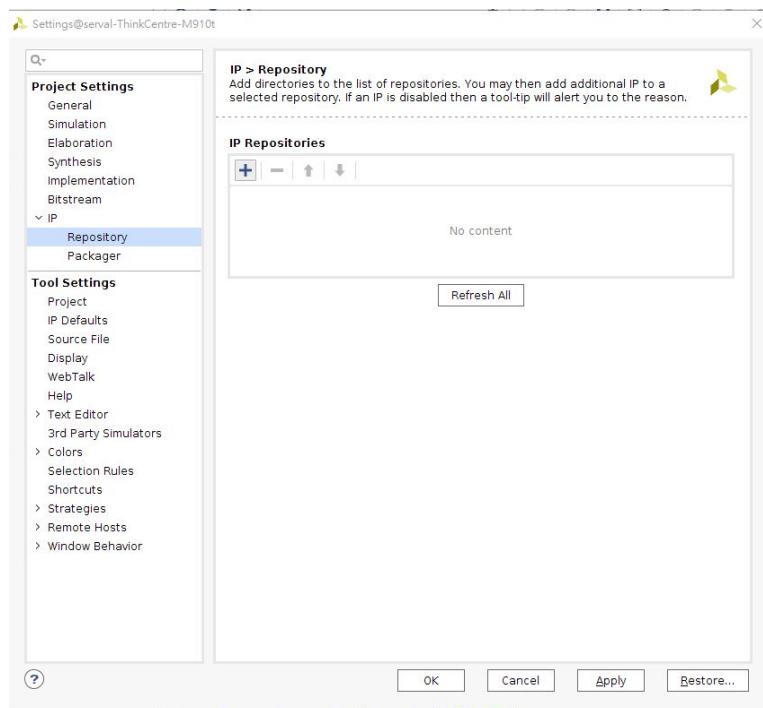
執行結果會在 專案目錄下(此為adder\_2b) 產生IP

## Custom IP使用

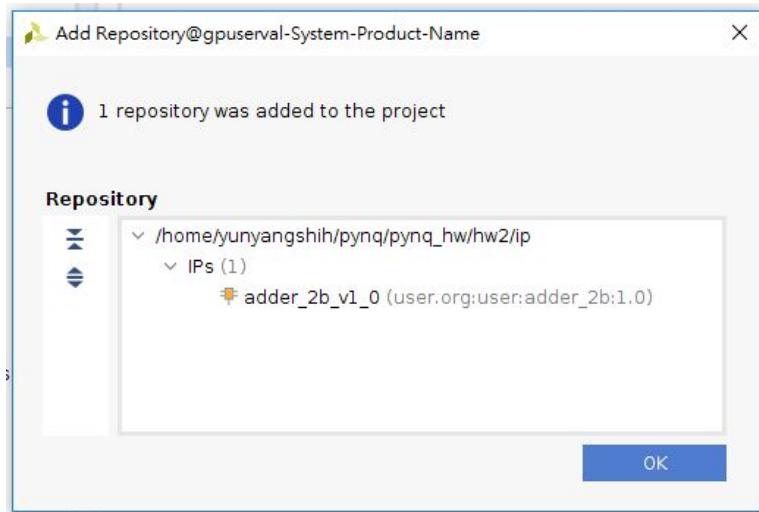
1. 另外產生一個專案，名稱 project\_2 (參考hw3-1 的步驟)
2. 點左邊的Create Block Design，點OK
3. 在Diagram中點右鍵，選IP Settings



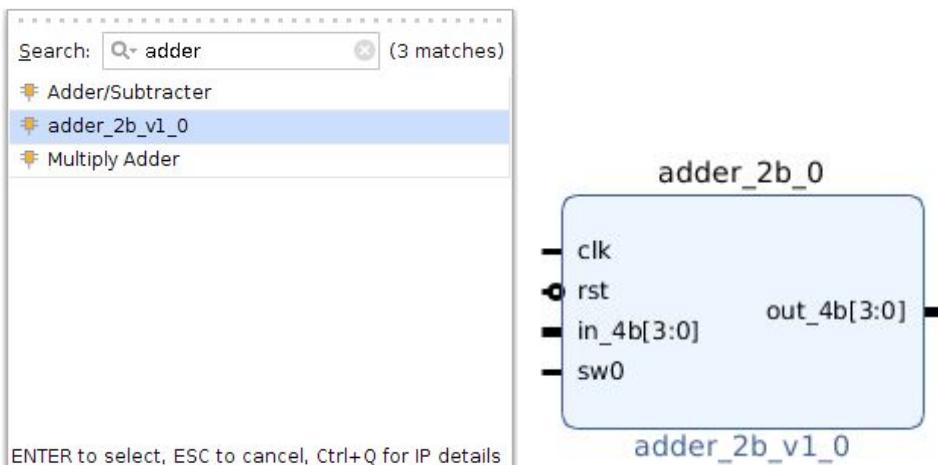
**點開左方的IP，點選Repository，點 + 來添加IP路徑，選剛剛IP專案的路徑(外層路徑也行，他會自己往內尋找)**



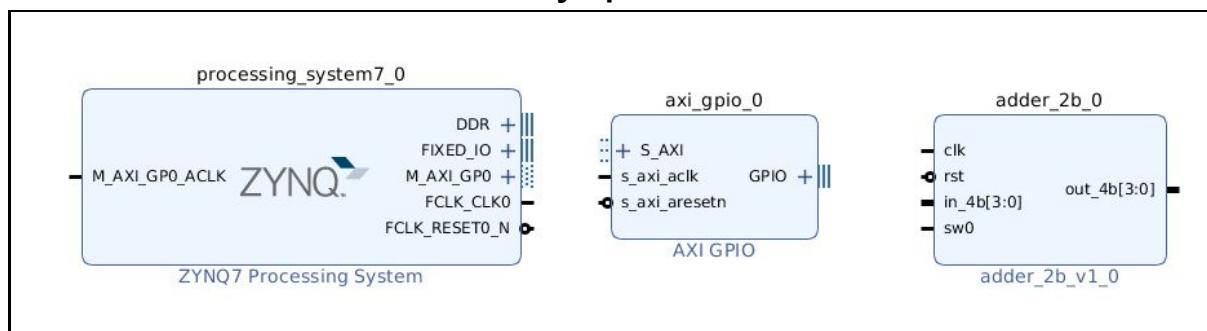
**有找到IP會出現以下訊息，點OK，點OK**



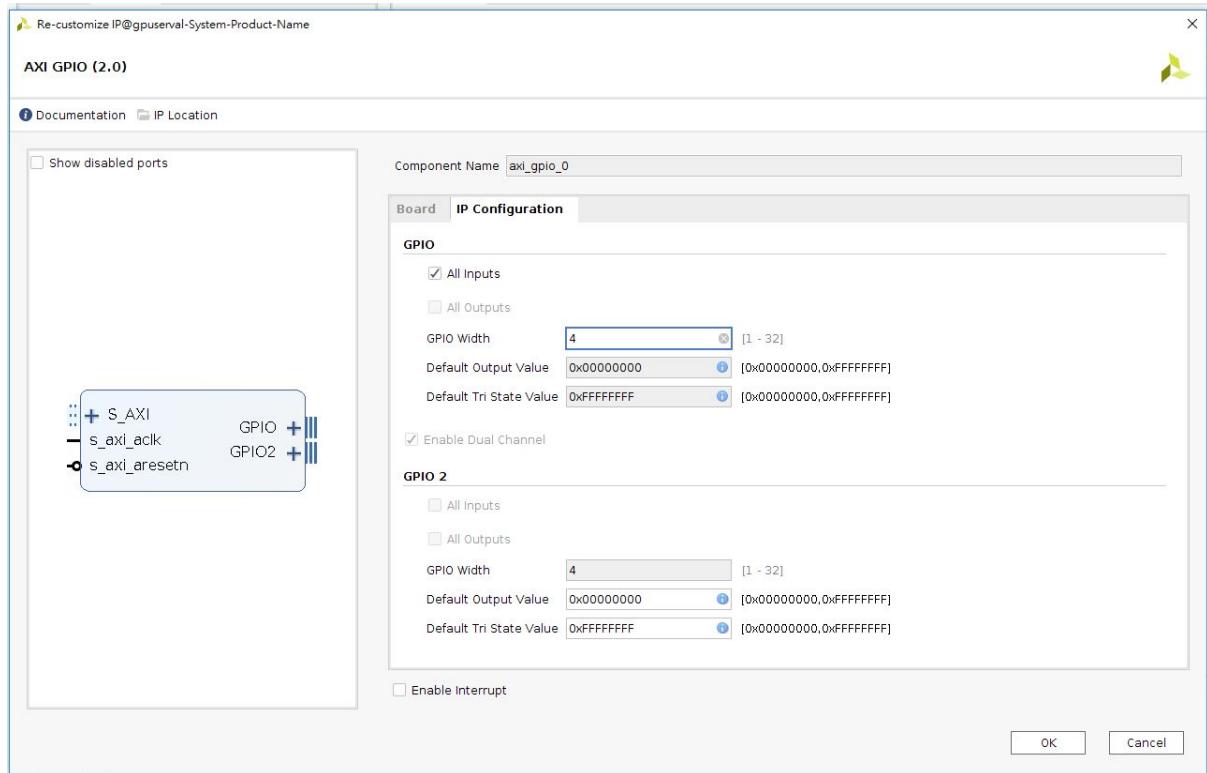
4. 回到Diagram中，按+即可找到自己建置的IP，加入Diagram



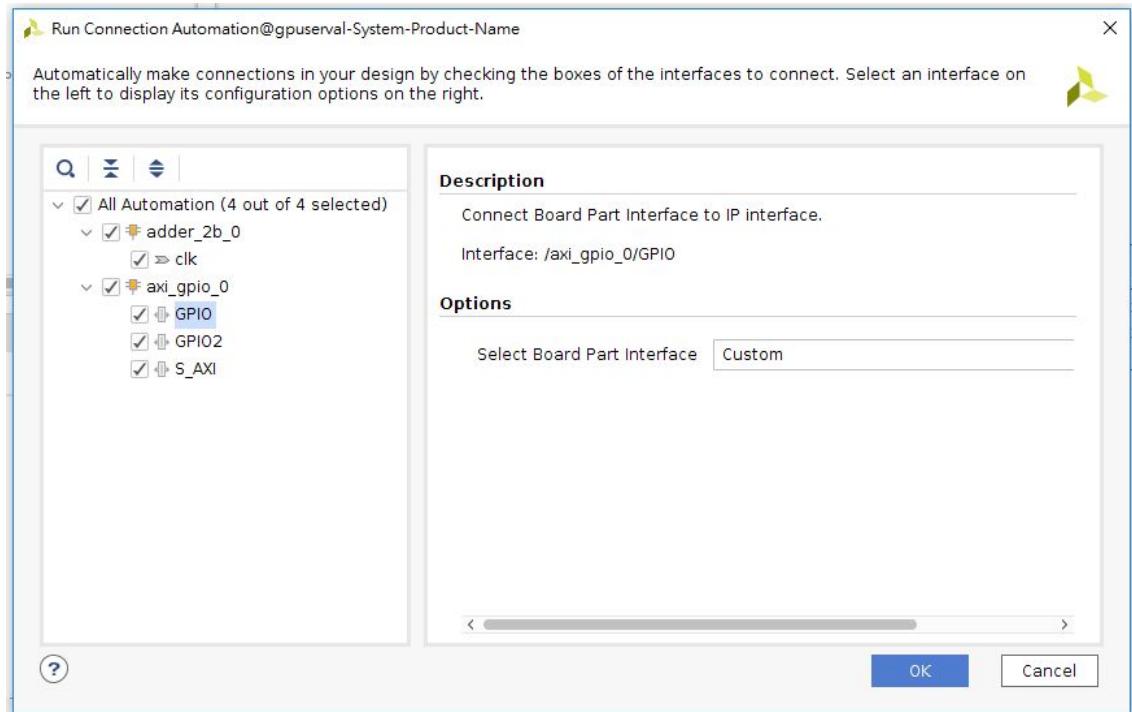
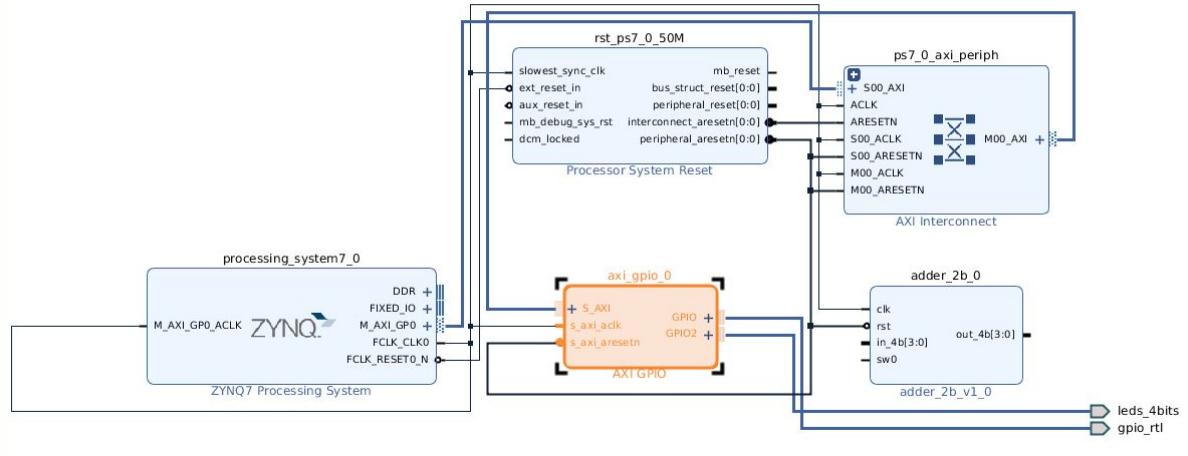
5. 之後的步驟如hw1，用+按紐加入Zynq、AXI GPIO



6. 對AXI\_GPIO點三下來開啟設定，在Board設定把GPIO2設成leds 4bits，在IP Configuration設定GPIO為All inputs，GPIO Width為4，點"OK"

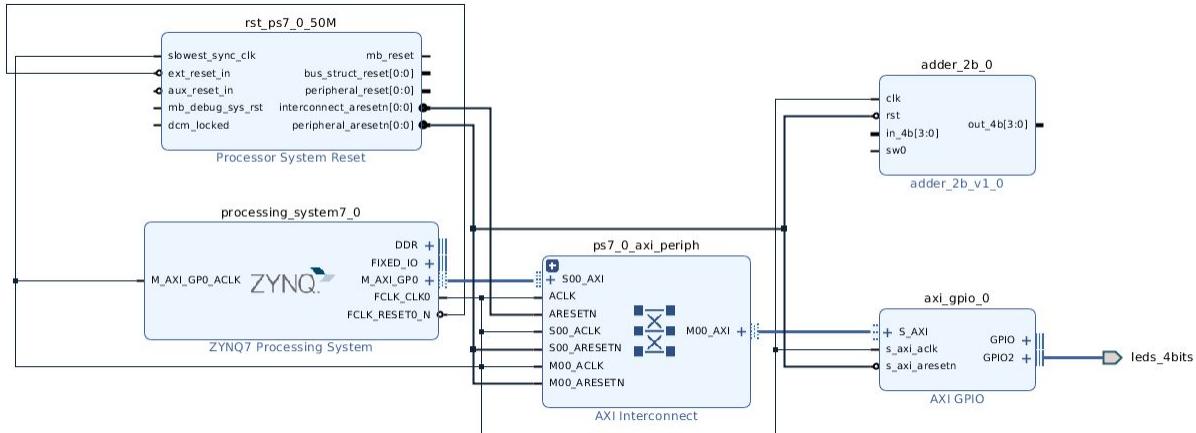


7. 點上面的"Run Connection Automation", GPIO要設定一下, 點兩下"GPIO"然後  
設成"Custom", 點最上面的"All Automation", 點"OK"

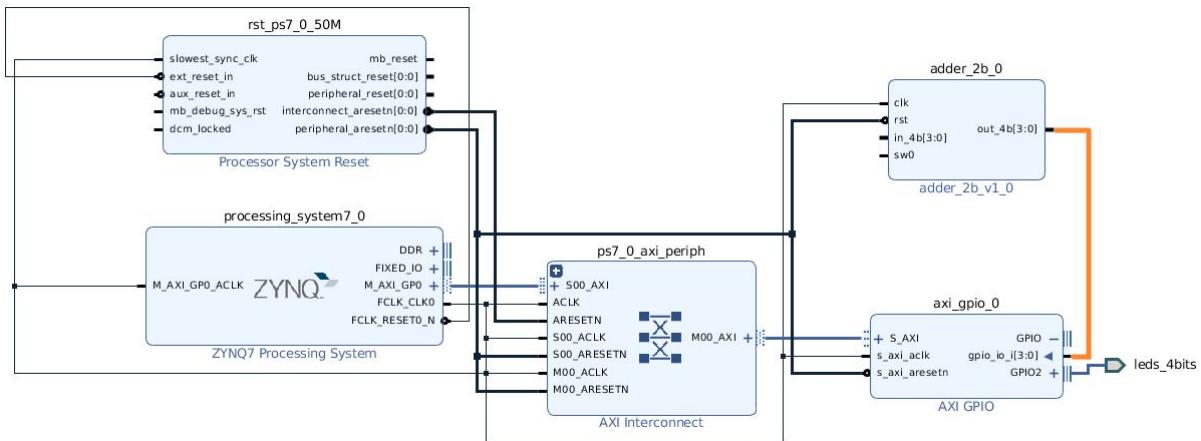


8. 完成後如下圖(IP配置位置可能不同，不影響功能)

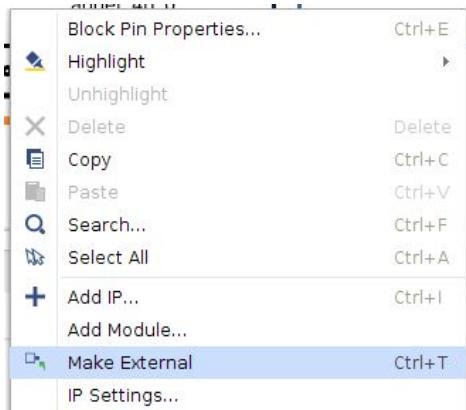
9. 由於有加自己的IP，所以要修改接線，**把gpio\_rtl拿掉(左鍵點，然後按delete)**，重新排列一下IP(為了更好理解功能)，如下圖

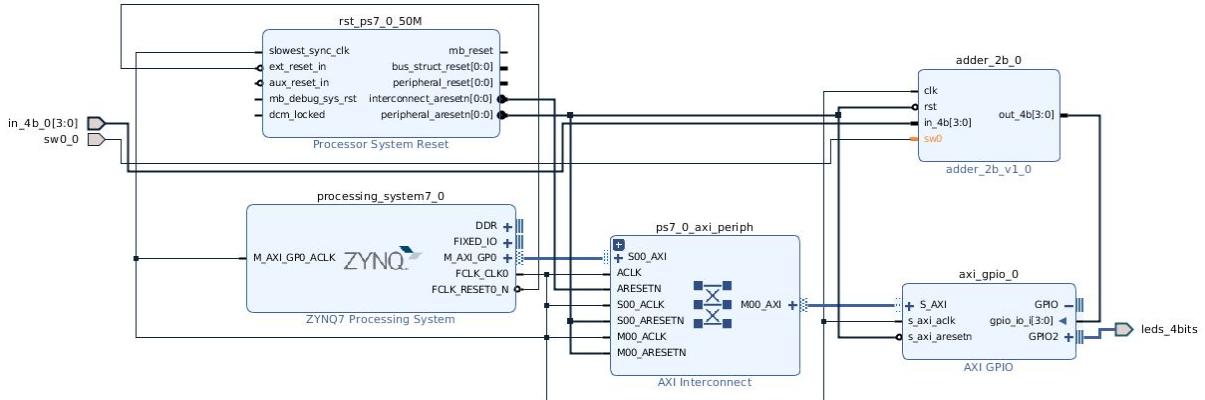


## 10. 把axi\_gpio\_0的GPIO的“+”點開，把 "gpio\_io\_i" 接到 "out\_4b" (點左鍵拖曳)

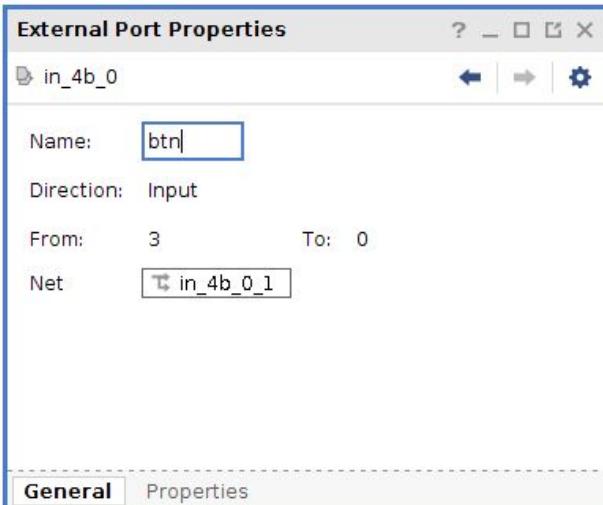


## 11. 對 "in\_4b" 和 "sw0" 分別點右鍵，選 "Make External"，調一下位置後如下

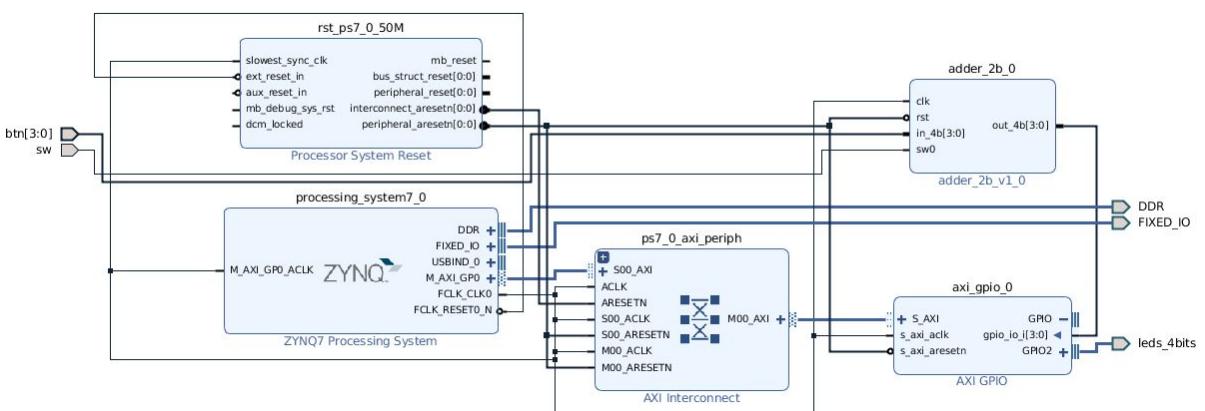




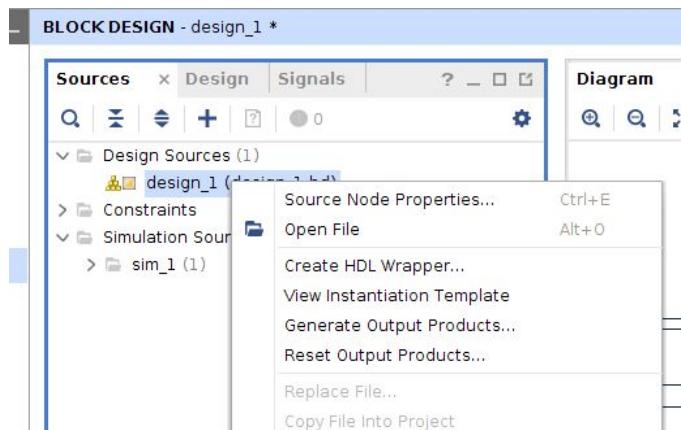
12. 對 "in\_4b\_0" 和 "sw0\_0" 點一下，修改左方 "External Port Properties" 的 "Name"根據角位名稱分別修改為 "btn" 和 "sw"



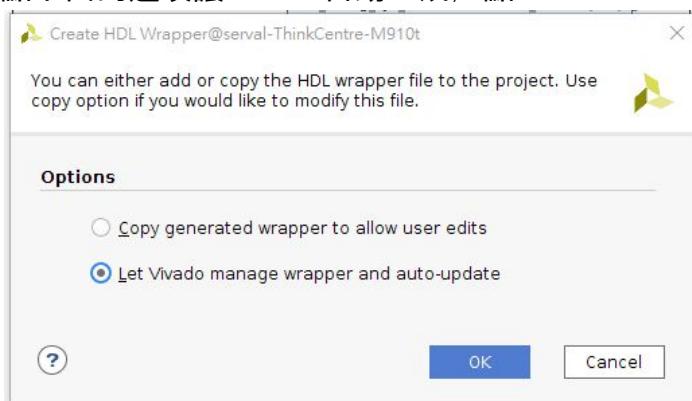
13. 點" Run Block Automation " 讓ZYNQ的 DDR 和 FIXED\_IO 接出去，軟體端才能正常對memory操作，直接點OK，完成圖如下，Ctrl + s 存一下



點左上角的source, design\_1點右鍵，點 "Create HDL Wrapper "。



點下面的選項讓Vivado自動生成，點OK



## 合成上板子

根據hw3-1的合成步驟，載入腳位檔後完成硬體檔，並將.bit檔和.tcl檔放入pynq中

pynq jupyter

**python code 請同學依照hw3-1的方式自行完成，其功能如下：**

**開關(sw0)為0：**

**LED(4bits) = 前兩個按鈕(2bits) - 後兩個按鈕(2bits)**

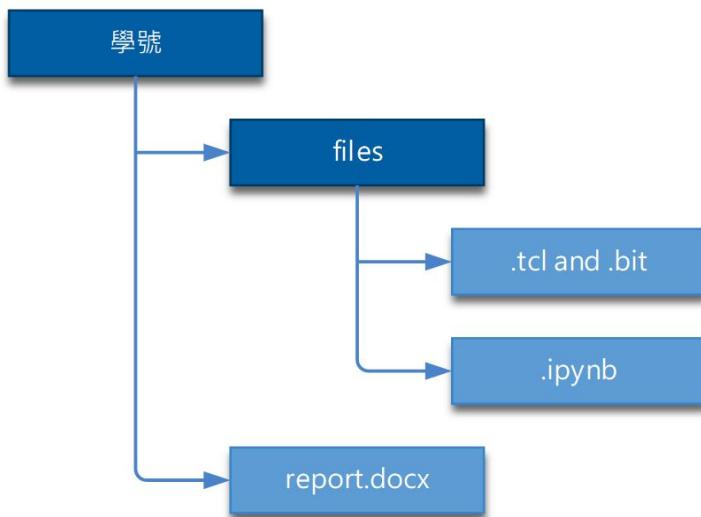
**開關(sw0)為1：**

**LED(4bits) = 前兩個按鈕(2bits) + 後兩個按鈕(2bits)**

# 作業繳交格式

本作業同學須完成系統設計與報告撰寫，請依照下圖的資料夾格式來繳交完成的檔案

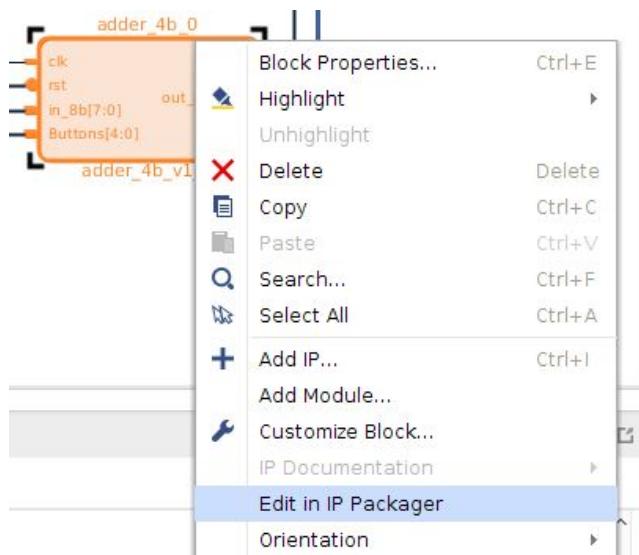
1. 繳交檔名為**學號.zip**, 裡面會有一個自己學號的資料夾
2. 學號的資料夾中，會有一個**report.docx**和一個**files**的資料夾
3. **files**中會有三個檔案，分別是**.tcl** 和 **.bit** 和 **hw3.ipynb**
4. 報告內容請參考**report.docx**



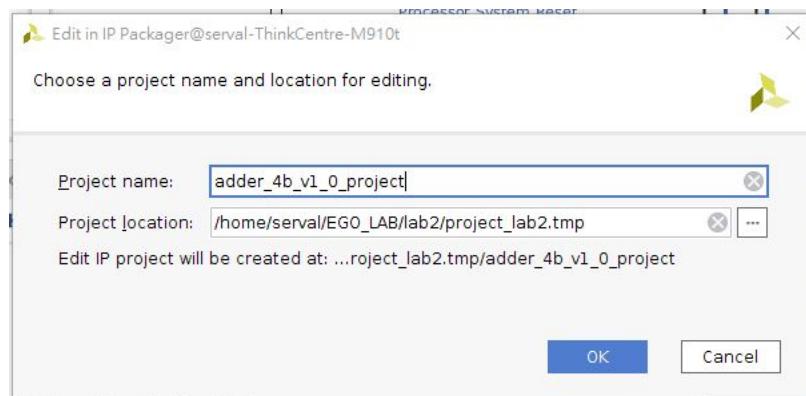
## 附錄：Custom IP在Block design中修改

如果需要修改自己建的IP，則參考以下步驟

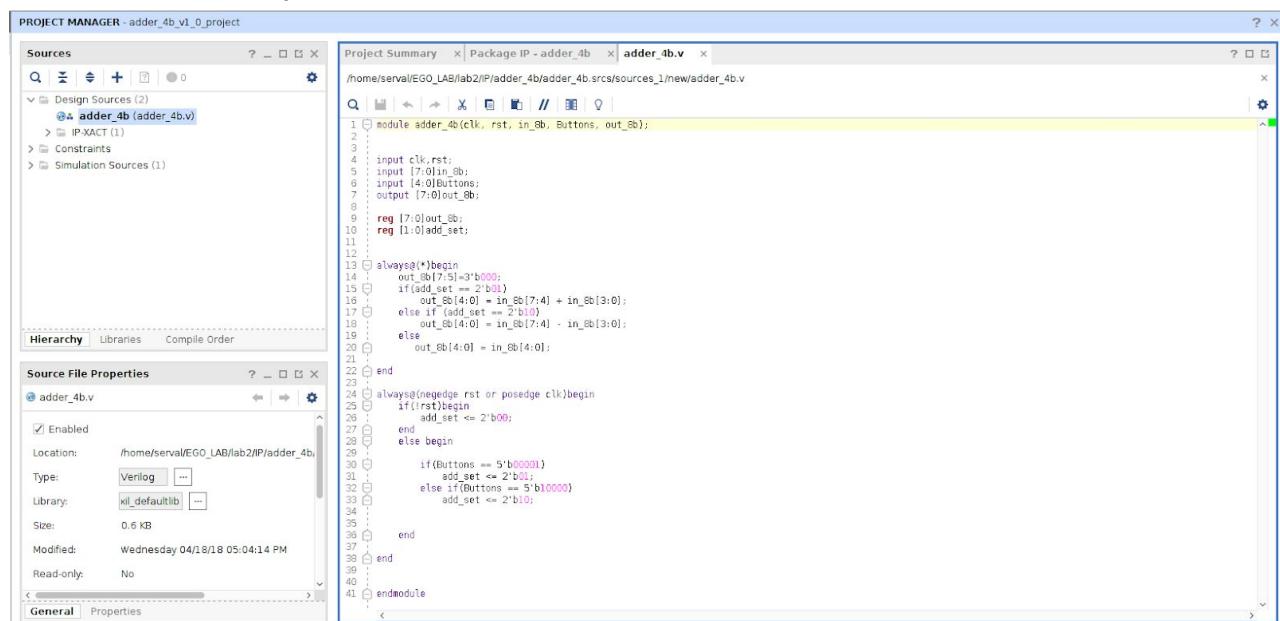
1. 在block design中，對IP點右鍵，選>Edit in IP Packager"



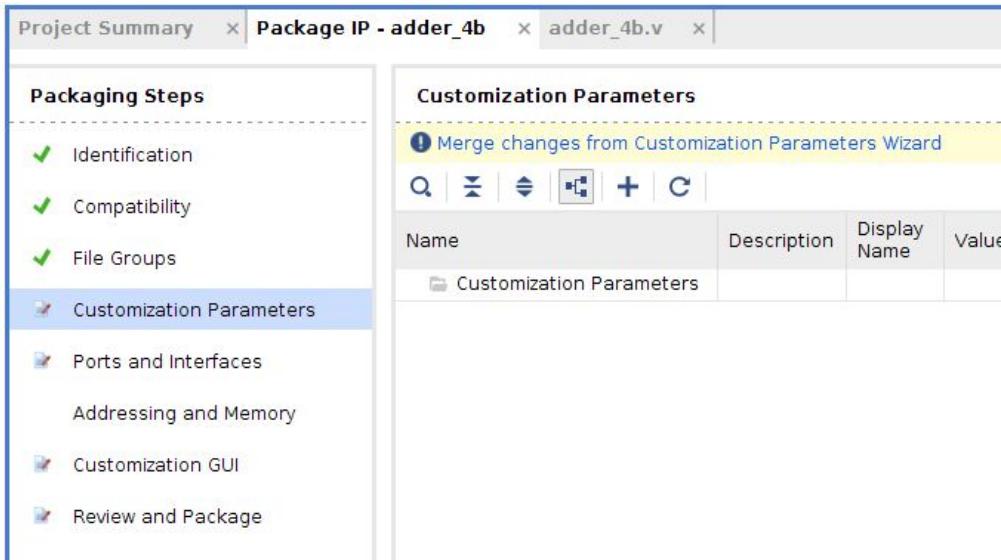
之後會開啟下面的視窗，直接點OK即可，Vivado會創立一個IP的暫時project



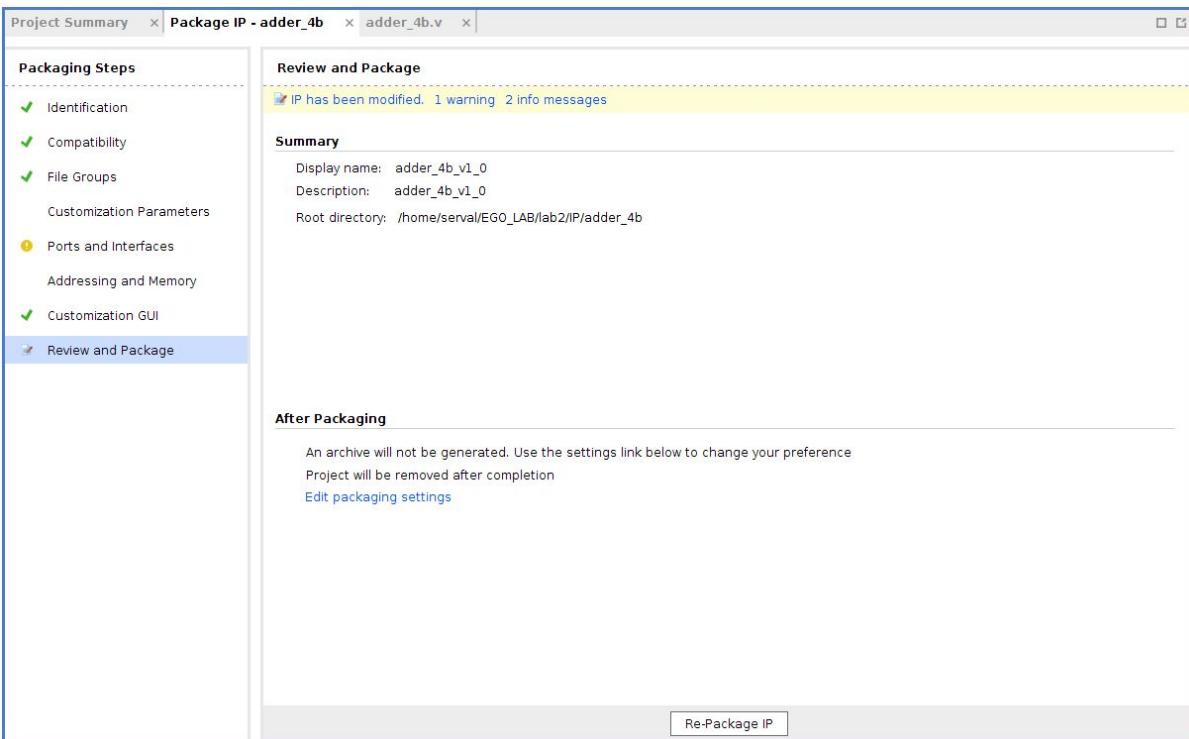
2. 直接在這個暫時project修改IP即可，修改完後要按Ctrl + s 存一下



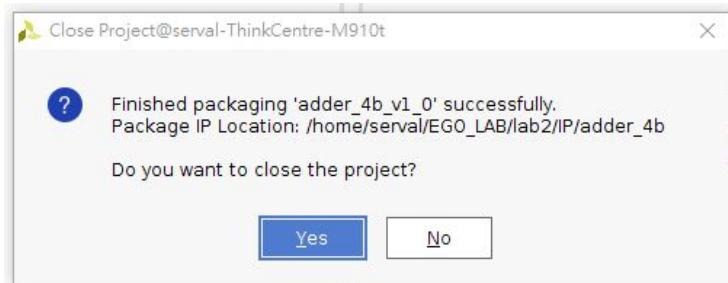
3. 修改完後點上面的Package IP - adder\_4b, vivado會發現IP已被更動，所以要重新設定和包裝，點選"Customization Parameters"，然後點選上面的"Merge changes from Customization Parameters Wizard"來自動重新設定



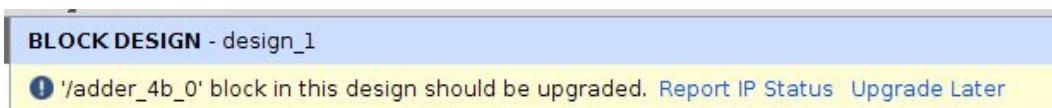
之後點選左邊"Review and Package"，點選下面的"Re-Package IP"來重新包IP，完成後即可關掉這個暫時的project



下面視窗點yes，關閉暫時的專案



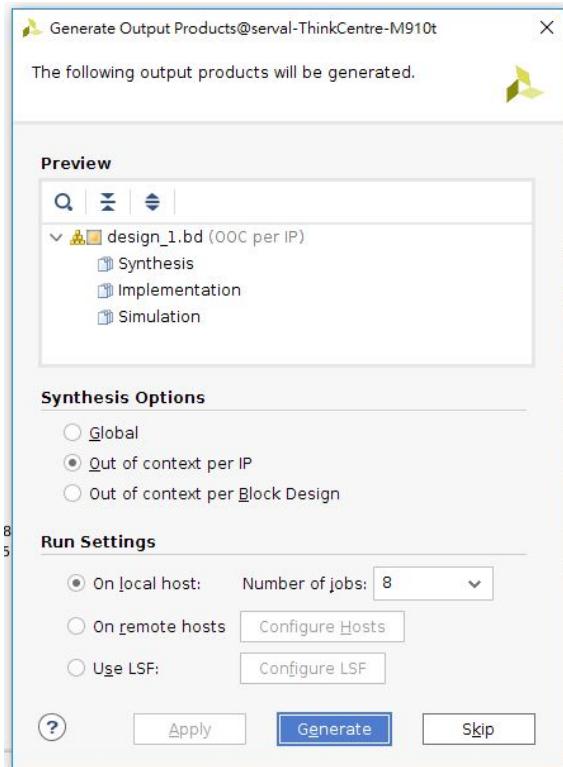
4. 回到Block design的部分，vivado會發現你的IP有所變更，所以block design要更新一下。  
 點上面的"Report IP Status"



之後block design下面會出現IP Status，直接點最下面的"Upgrade Selected"

| IP Status            |                                     |  |                            |                         |                 |                     |          |                 |
|----------------------|-------------------------------------|--|----------------------------|-------------------------|-----------------|---------------------|----------|-----------------|
| Source File          |                                     | Recommendation   | Change Log                 | IP Name                 | Current Version | Recommended Version | License  | Current Part    |
| design_1 (5)         |                                     |  |                            |                         |                 |                     |          |                 |
| adder_4b_0           | <input checked="" type="checkbox"/> | IP revision change. IP definition 'adder_4b_v1_0 (1.0)' changed on disk. | <a href="#">Upgrade IP</a> | adder_4b_v1_0           | 1.0 (Rev. 3)    | 1.0 (Rev. 5)        | Included | xc7z020clg484-1 |
| axi_gpio_0           | <input type="checkbox"/>            | Up-to-date   | No changes required        | AXI GPIO                | 2.0 (Rev. 17)   | 2.0 (Rev. 17)       | Included | xc7z020clg484-1 |
| processing_system7_0 | <input type="checkbox"/>            | Up-to-date   | No changes required        | ZYNQ7 Processing System | 5.5 (Rev. 6)    | 5.5 (Rev. 6)        | Included | xc7z020clg484-1 |
| ps7_0_axi_periph     | <input type="checkbox"/>            | Up-to-date   | No changes required        | AXI Interconnect        | 2.1 (Rev. 16)   | 2.1 (Rev. 16)       | Included | xc7z020clg484-1 |
| rst_ps7_0_S0M        | <input type="checkbox"/>            | Up-to-date   | No changes required        | Processor System Reset  | 5.0 (Rev. 12)   | 5.0 (Rev. 12)       | Included | xc7z020clg484-1 |

出現以下視窗，直接按X關掉



5. 下面的IP Status應該會出現下列訊息，按"Rerun"

 Report is out of date because the status of one or more IPs have changed. [Rerun](#)

6. IP更新完後要重新合成，以產生新的硬體檔(參考hw3-1的合成上板子)

# LAB 4 : CDMA系統建立與資料傳輸

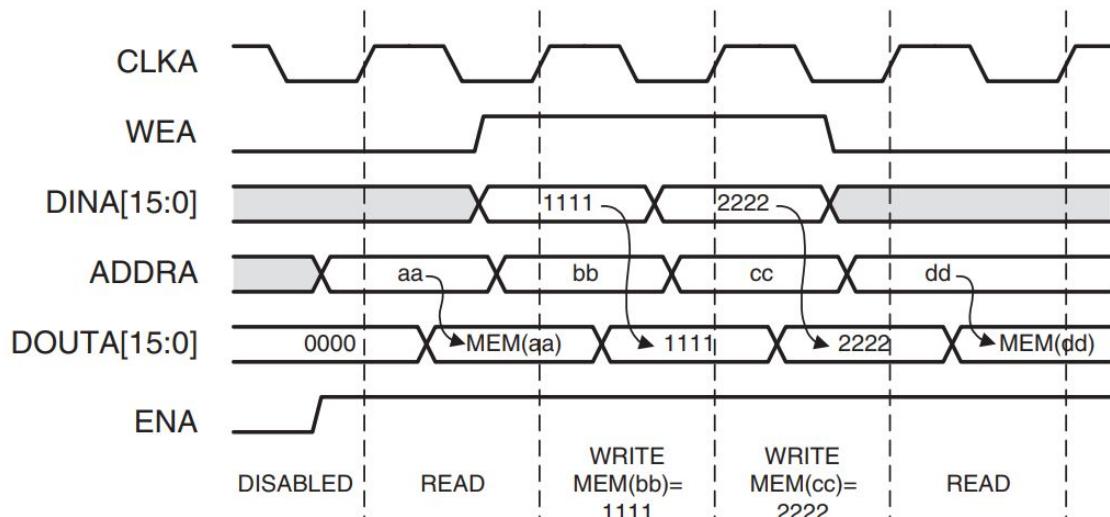
## 作業說明

本作業須使用Xilinx ip和自製的ip來完成一套乘法的系統，此系統由CDMA搬運資料至Block memory，由硬體計算完畢後再由CDMA將資料搬回ZYNQ，本作業提供mul16.v，請同學自行包成vivado可用的ip，並完成系統建置和jupyter軟體設計。

## IP 介紹

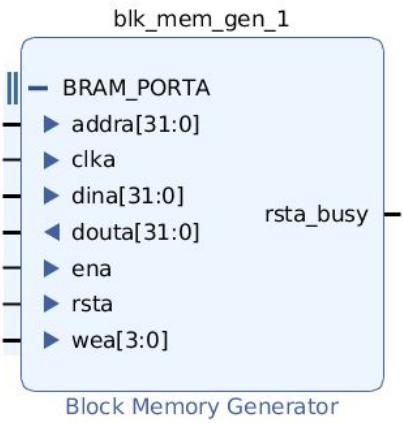
### Block memory

block memory generator是Xilinx vivado提供的一個IP，可生成Block memory，使用上和一般的sram差不多，其中可調整各種參數，像是memory size、single port or dual port等等，有點像memory compiler，**block memory 的address為byte address**，其讀寫的訊號如下圖：



vivado ip如下圖：(rst<sub>a</sub>\_busy是防止在rst時讀寫的腳位，可不使用)

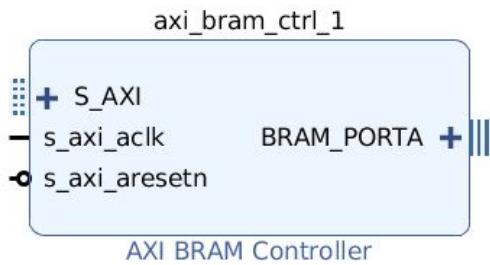
wea為對每個byte寫入的訊號，每次操作一個word(4bytes)的資料，所以wea有4個bit



## AXI bram controller

AXI bram controller是Xilinx vivado提供的一個IP，主要是將block memory的讀寫訊號轉乘AXI的格式，讓block memory可以和AXI BUS溝通

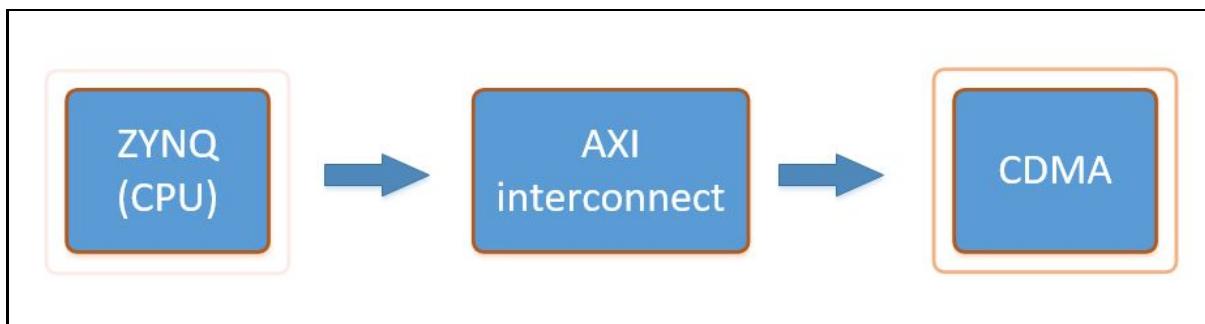
vivado ip如下圖：



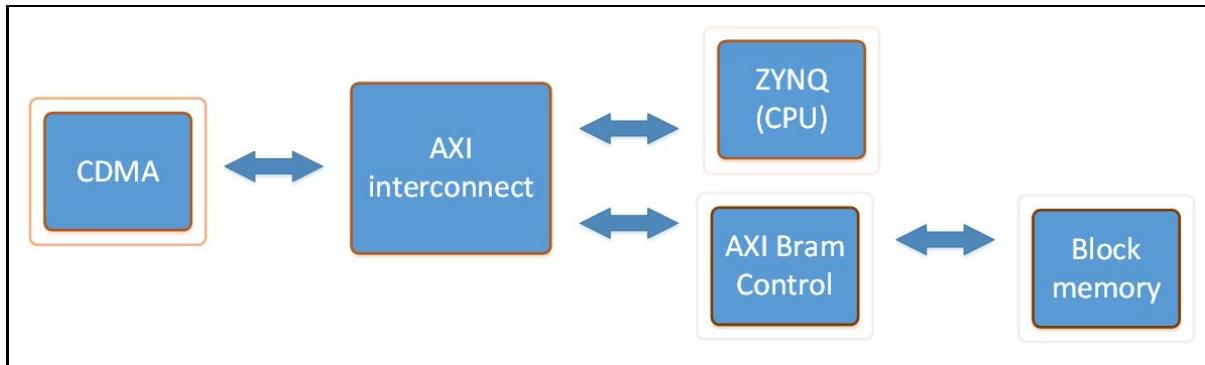
## CDMA

CDMA是Xilinx vivado提供的一個IP，可將資料在兩個memory address間搬移，和一般的DMA一樣，在系統上，CDMA同時具備Master和Slave的腳位，其操作方式大致如下：

1. 以ZYNQ為master、CDMA為Slave，透過AXI BUS來傳送控制資料，來告訴CDMA要從哪裡傳到哪裡、傳多少資料等等資訊

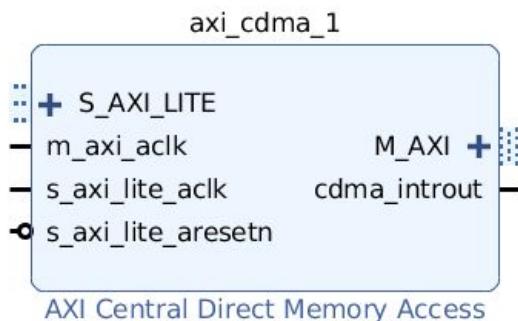


2.以CDMA為master、ZYNQ和block memory為slave，進行資料的傳送，block memory必須接上AXI Bram Controller才能將資料傳遞到AXI BUS上



vivado ip如下圖：

這裡不會用到SG模式，可以在CDMA的ip設定中把它關掉



## mul16

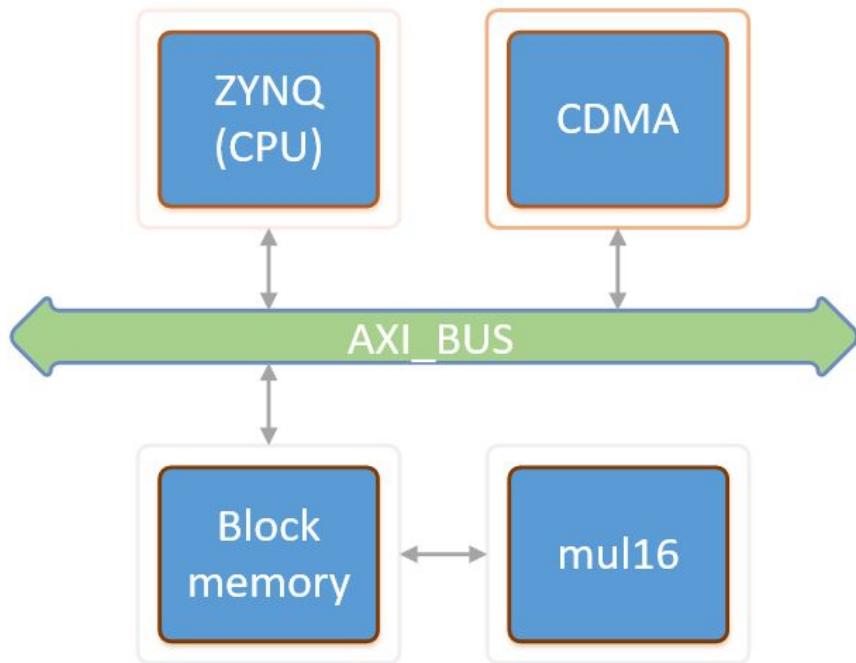
助教自己做的ip，做乘法運算，將block memory address為0x00和0x04的資料讀出來，相乘後放入0x0C中，所以讀取block memory的0x0C即可得到乘法結果。

作業中mul16只提供.v檔，請同學自行包裝成ip，請參考hw3-2

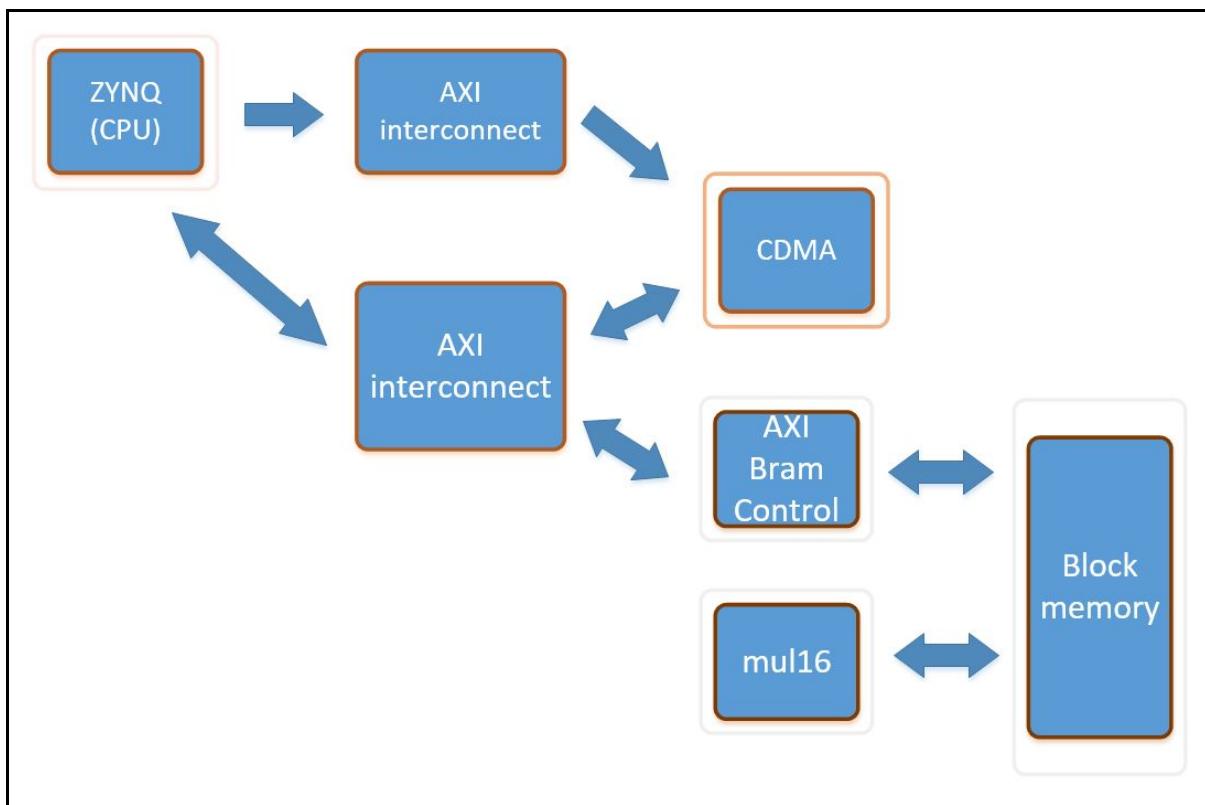
## 系統架構圖

ZYNQ、CDMA、Block memory進行資料傳遞時一定會通過AXI BUS，而mul16運算單元直接接在Block memory上，不須透過AXI\_BUS直接讀寫Block memory即可

系統的架構圖如下：

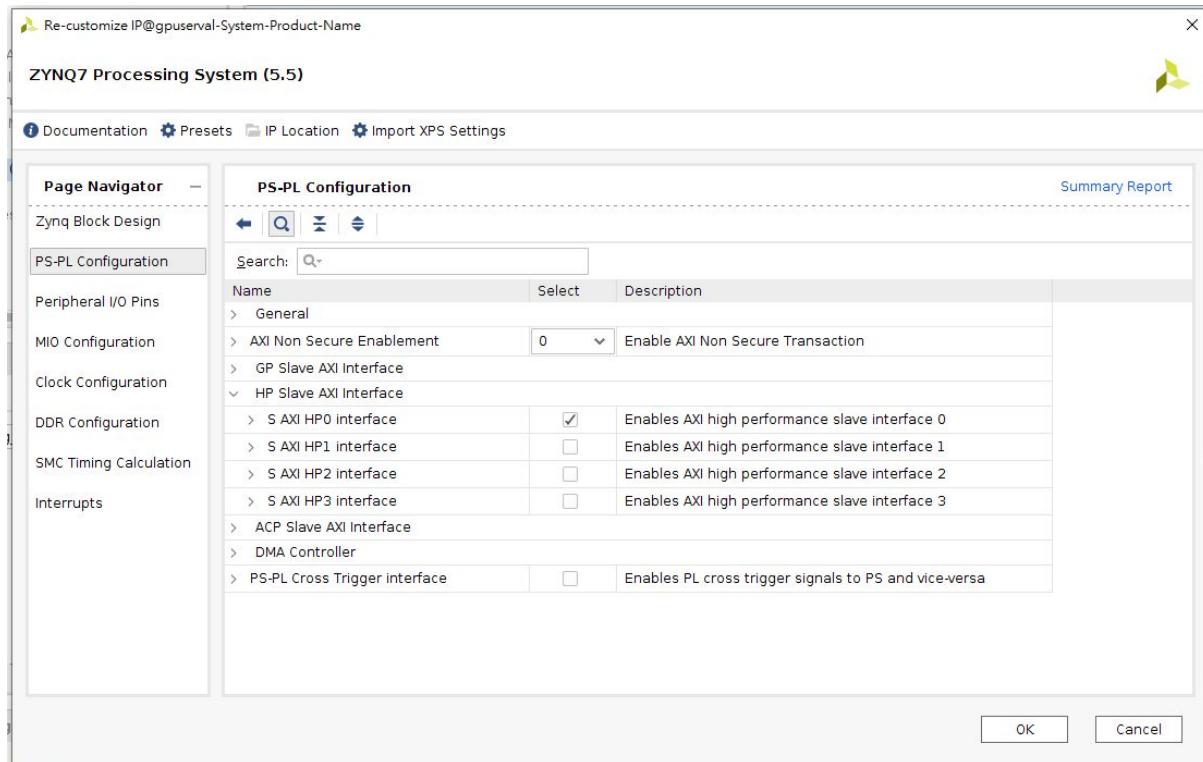


較詳細的vivado架構圖如下：

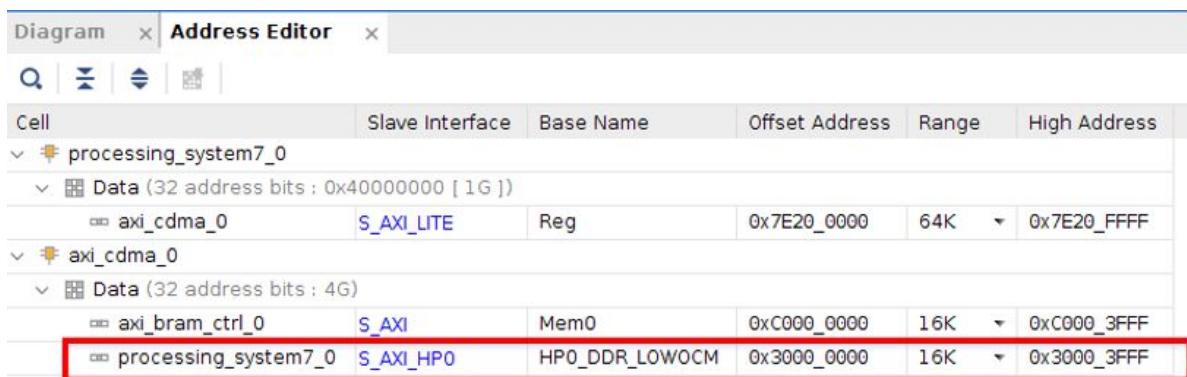


## 注意事項：

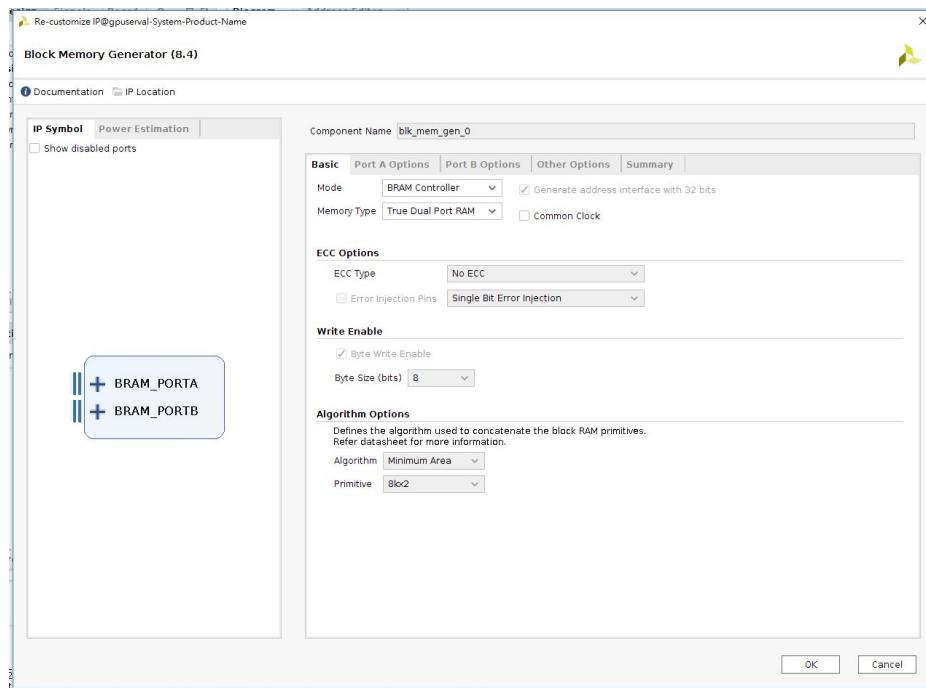
1. AXI interconnect可由系統自動生成，但自動生成時必須**注意master和slave的關係**，接錯則系統無法正確運行。ZYNQ memory要當成slave給cdma使用，則需要在ZYNQ加上一個slave port，可由ZYNQ的IP設定介面加入S AXI HP



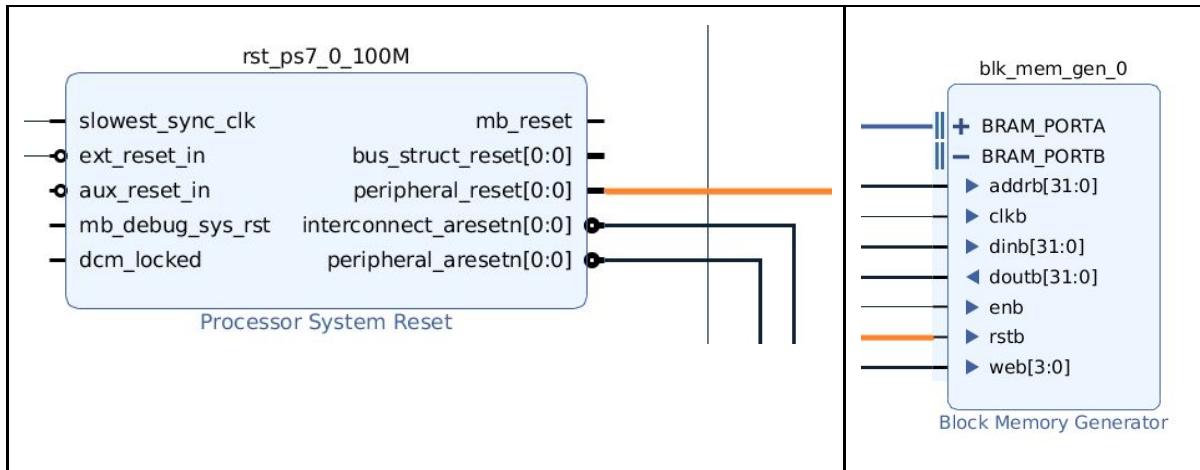
2. Block design的Address Editor可自行設定，而ZYNQ(processing\_system7\_0)的部分不要設定在0x00000000到0xFFFFFFFF之間，那裡是系統需要的空間，讀寫可能會造成系統錯誤



3. 由於CDMA和mul16都要操作block memory，此作業的block memory必須設定為True Dual Port RAM，可在block memory generator設定



4. True Dual Port RAM 的第二個port要接上mul16，必須手動一個一個接，其中 reset(rstb)為正緣觸發，必須接在Process System Reset的peripheral\_reset上



5. mul16的clk和其他ip的aclk等腳位接同一clk訊號即可，rst為負緣觸發，和其他ip同樣接在Process System Reset的peripheral\_aresetn上即可。

請同學根據此架構建置系統

pynq jupyter

系統執行步驟如下，請同學根據以下步驟完成jupyter：

1. 將兩個數放入系統的memory中，必須要放在address + 0 和 address + 4的位置，address必須要在ZYNQ可操作的記憶體內，可在block design介面中的Address Editor中看到系統可操作的記憶體。圖片中可見ZYNQ可操作0x30000000到0x30003FFF的範圍，大小可自行調整。

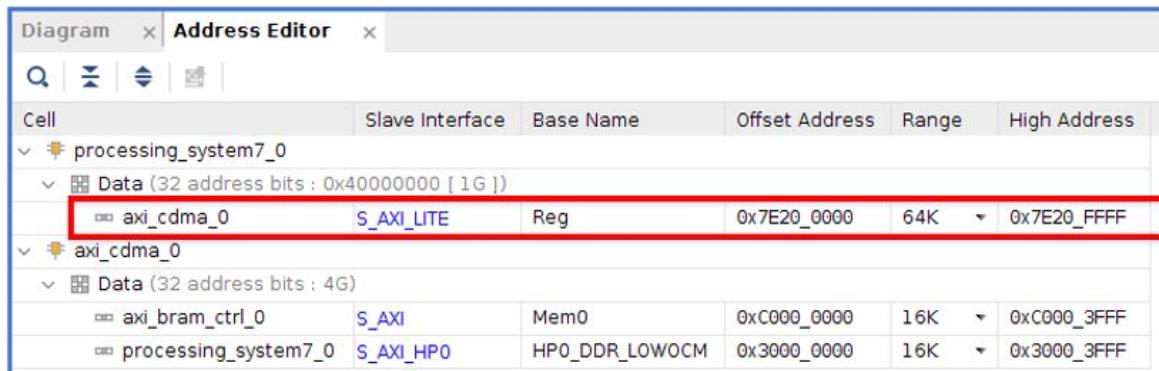
| Cell                                       | Slave Interface | Base Name      | Offset Address | Range | High Address |
|--|-----------------|----------------|----------------|-------|--------------|
| processing_system7_0                       |                 |                |                |       |              |
| Data (32 address bits : 0x40000000 [ 1G ]) |                 |                |                |       |              |
| axi_cdma_0                                 | S_AXI_LITE      | Reg            | 0x7E20_0000    | 64K   | 0x7E20_FFFF  |
| axi_cdma_0                                 |                 |                |                |       |              |
| Data (32 address bits : 4G)                |                 |                |                |       |              |
| axi_bram_ctrl_0                            | S_AXI           | Mem0           | 0xC000_0000    | 16K   | 0xC000_3FFF  |
| processing_system7_0                       | S_AXI_HPO       | HP0_DDR_LOWOCM | 0x3000_0000    | 16K   | 0x3000_3FFF  |

2. 設定CDMA將資料從ZYNQ搬到block memory, 設定步驟如下：
- 將CDMACR(CDMA\_control)寫入0x04來啟動cdma
  - 將SA(Source Address)寫入ZYNQ剛剛寫入資料的address
  - 將DA(Destination Address Address)寫入axi\_bram的address, 以上圖為例  
address = 0xC0000000
  - 將BTT寫入要傳遞的Byte數, 不能小於8, 因為至少要傳遞兩個word的資料,  
寫完cdma\_BTT後CDMA就會開始傳送

CDMA控制訊號對應address如下表：

| CDMA   | address |
|--------|---------|
| CDMACR | 0x00    |
| SA     | 0x18    |
| DA     | 0x20    |
| BTT    | 0x28    |

記得以上address操作時須加上cdma的控制address, 以下圖為例 cdma的控制address為0x7E200000



3. 設定CDMA將資料從block memory搬回ZYNQ, 設定步驟如下：
- 將CDMACR(CDMA\_control)寫入0x04來啟動cdma
  - 將SA(Source Address)寫入axi\_bram的address
  - 將DA(Destination Address Address)寫入ZYNQ的address
  - 將BTT寫入要傳遞的Byte數, 不能小於16, 因為至少要傳遞4個word的資料,  
寫完cdma\_BTT後CDMA就會開始傳送

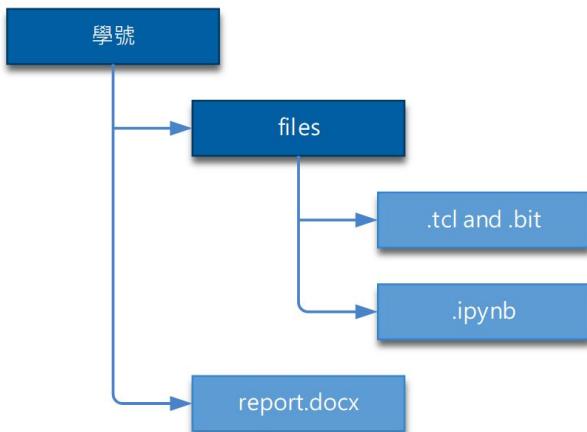
4. 讀取ZYNQ address + 0x0C的資料，就會得到乘法的答案

```
: print(f"ans of {a}*{b} = {sys_data.read(0x000C)}")  
ans of 60*20 = 1200
```

## 作業繳交格式

本作業同學須完成系統設計與報告撰寫，請依照下圖的資料夾格式來繳交完成的檔案

1. 繳交檔名為**學號.zip**, 裡面會有一個**自己學號的資料夾**
2. 學號的資料夾中，會有一個**report.docx**和一個**files**的資料夾
3. **files**中會有三個檔案，分別是**.tcl** 和 **.bit** 和 **.ipynb**
4. 報告內容請參考**report.docx**



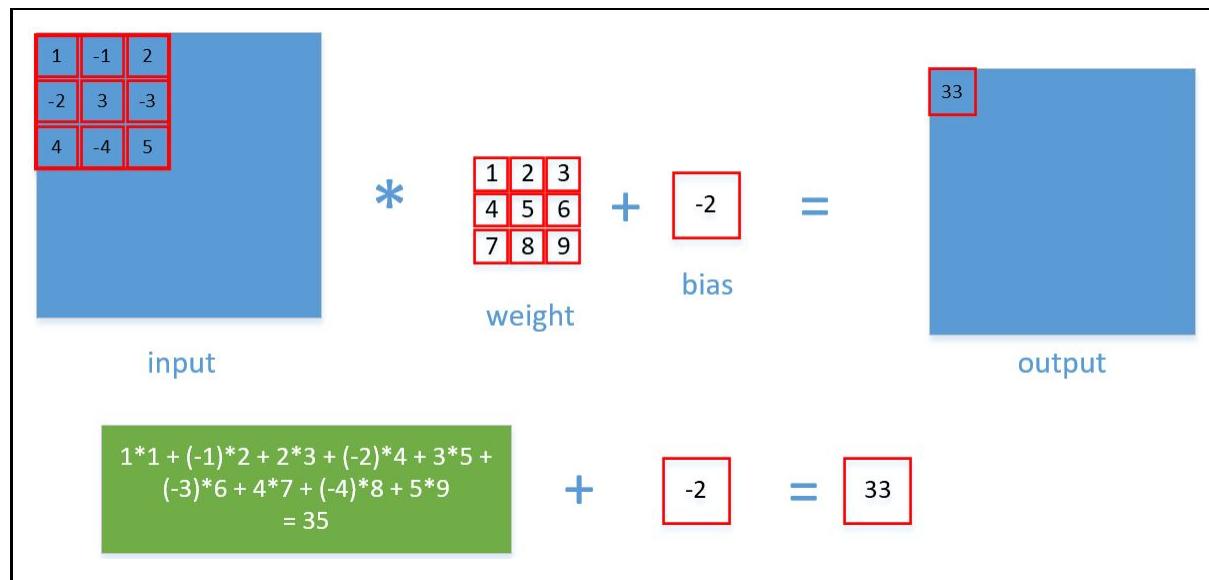
# LAB 5 : Convolution運算系統

## 作業說明

此作業同學必須自行完成convolution運算的ip，並用此ip來完成一個系統，作業會提供input.hex和golden.hex，其中input.hex有包含input、weight和bias等資訊，同學必須自行放入系統，而golden.hex則是正確答案給同學參考，此作業的軟體部分需自行生成output.hex檔案，且與golden.hex必須完全一樣，才算是功能正確，軟體端除了搬運資料外，不得有太多運算，全部的convolution運算必須在 convolution ip中完成。

## convolution運算

此作業須完成2D的convolution運算，convolution的運算概念如上課所教，在圖片中取一個和weight一樣大的window，其中每個元素與weight的每個元素相乘，結果再加總起來，最後加上bias即完成convolution運算的一個結果。



下表為資料的大小，同學必須完成對28\*28的input做convolution，得到26\*26的output

| data   | size  |
|--------|-------|
| input  | 28*28 |
| weight | 3*3   |

|        |       |
|--------|-------|
| bias   | 1     |
| output | 26*26 |

## 資料配置

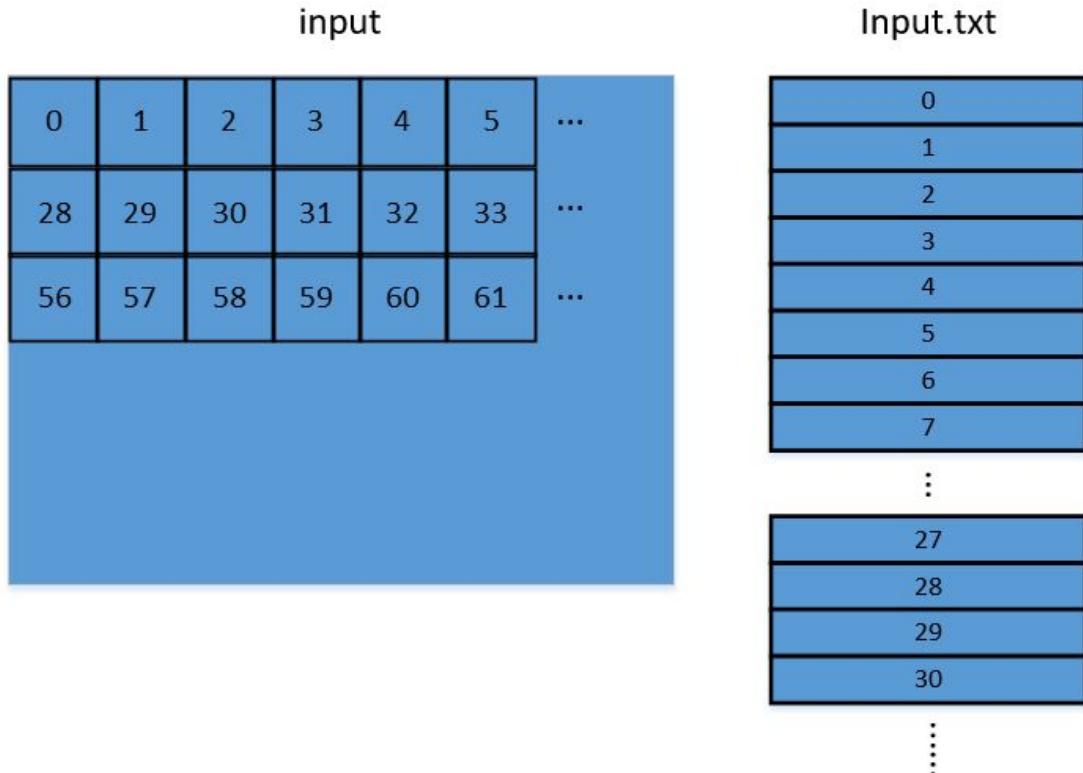
所有運算資料與答案皆為32bits的定點數，16bits整數和16bits小數



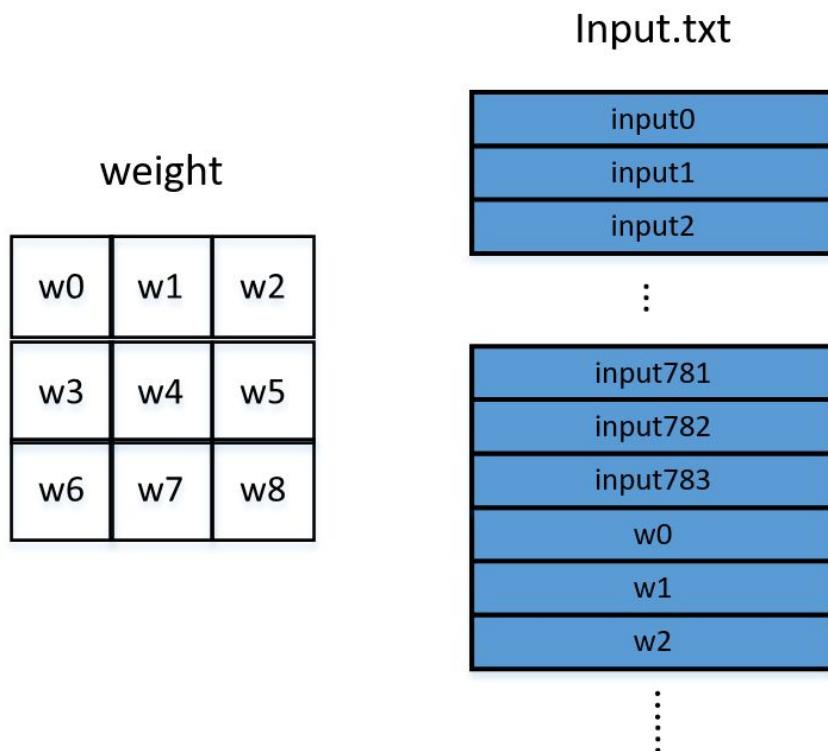
input.hex有包含input、weight和bias資訊，其大小與放置順序如下，單位是word(32bits)

| data   | size |
|--------|------|
| input  | 784  |
| weight | 9    |
| bias   | 1    |

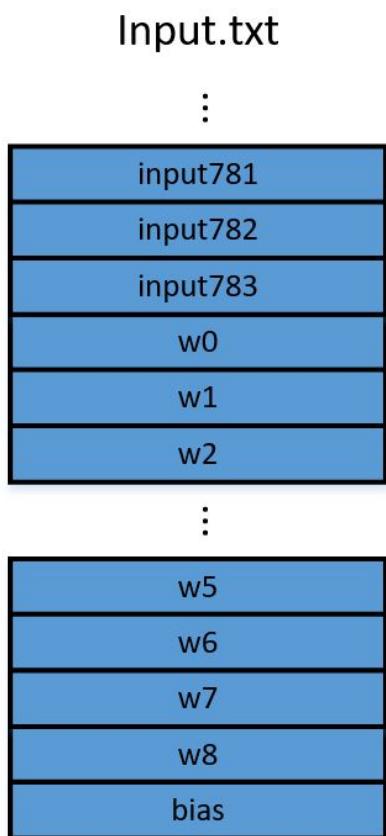
1. input放置方式為row major，照順序如下圖放置於input.hex



2. weight直接放置於最後一個input後面，放置方式也是row major

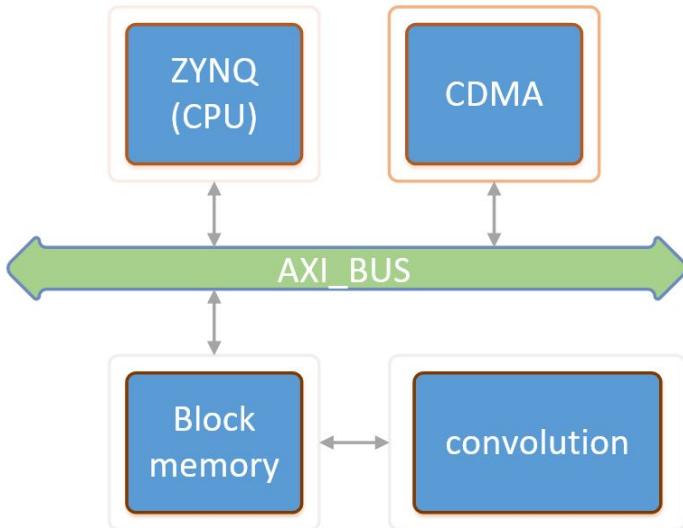


3. bias只有一個，直接放置於最後一個weight後面

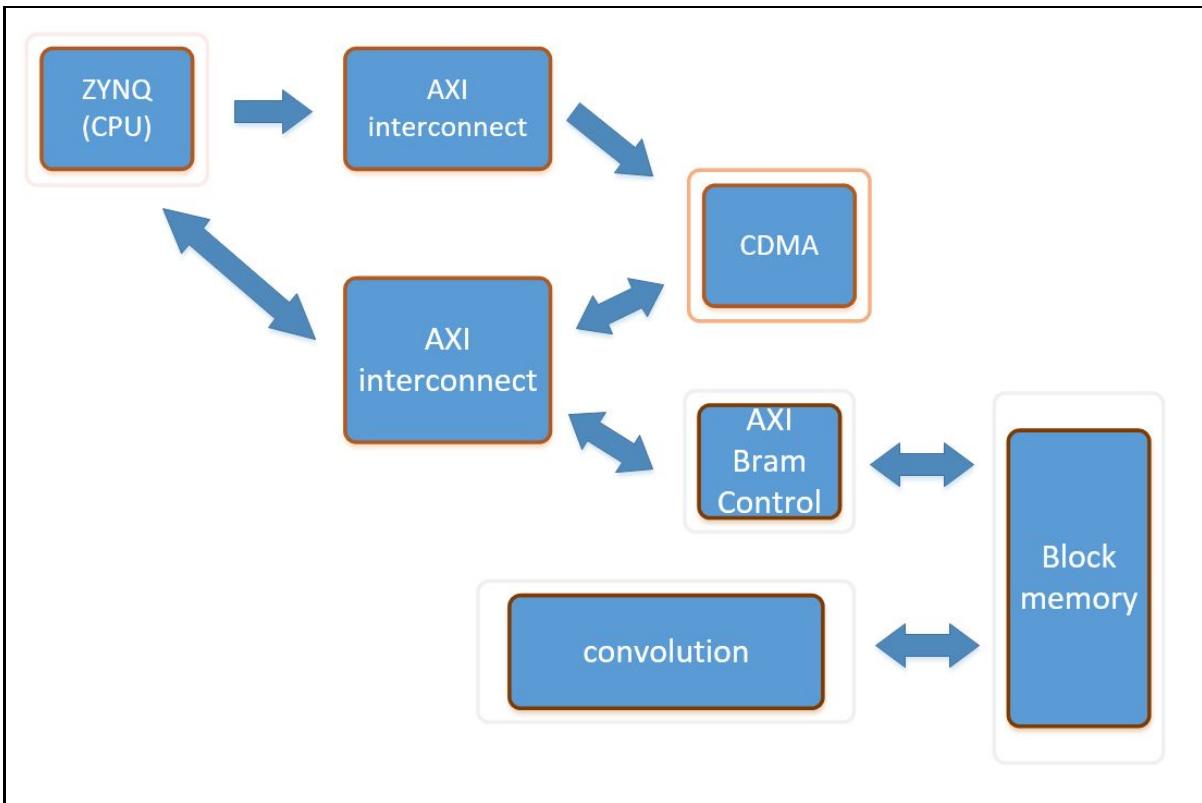


# 系統架構圖

下列系統架構圖僅供參考，大家可以自己設計系統架構。參考的系統的架構圖如下：



較詳細的vivado架構圖如下：



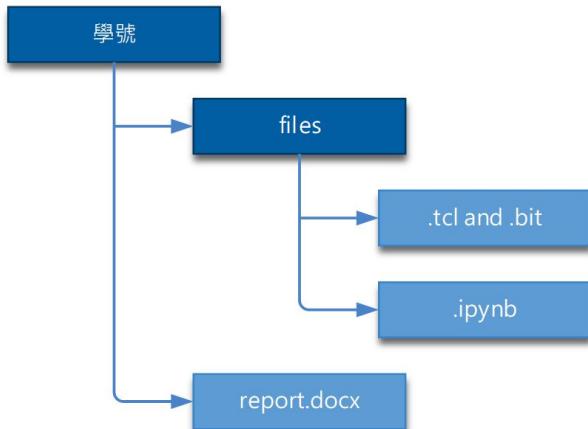
## pynq jupyter

同學的jupyter程式執行後，要生成一個output.hex，用於表示做完convolution運算後 $26 \times 26$ 的結果，跟input.hex一樣，以16進制、row major的表示方式，一筆資料一行的寫在output.hex中。output.hex的內容必須要和golden.hex比對，同學可以用linux的diff指令來比對兩個檔案是否一致，檔案完全一致才能算功能正確。

## 作業繳交格式

本作業同學須完成系統設計與報告撰寫，請依照下圖的資料夾格式來繳交完成的檔案

1. 繳交檔名為**學號.zip**，裡面會有一個自己學號的資料夾
2. 學號的資料夾中，會有一個**report.docx**和一個**files**的資料夾
3. **files**中會有三個檔案，分別是**.tcl** 和 **.bit** 和 **.ipynb**
4. 如未照上述方式擺放或檔名不符，扣5分
5. 報告內容請參考**report.docx**



# LAB 6 : IC contest 2019

## 1. 作業說明

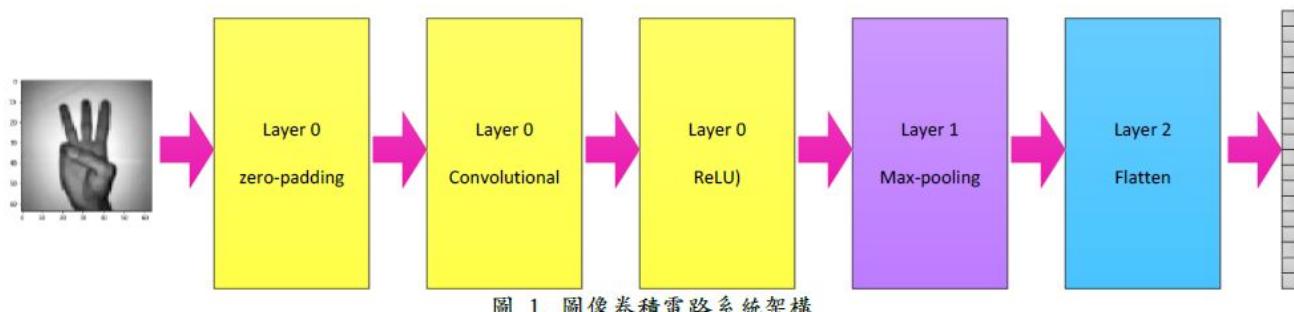
本題請完成一圖像卷積電路設計(Image Convolutional Circuit Design, 以下稱CONV 電路), CONV 電路輸入為一灰階圖像, 電路需完成3 層(Layer)的運算流程, 其順序為Convolutional(Layer 0)Max-pooling(Layer 1)Flatten(Layer 2)等共3 層的運算處理流程, 如下圖1. 所示。

首先在Layer 0 中, 輸入灰階圖像(尺寸為64(寬)x64(高) pixels)需先經zero-padding, 接著進行採用2 個濾鏡(或稱”核(Kernel)”)的圖像卷積(Convolutional)運算, 其Kernel 尺寸為3x3, 接著再經過ReLU 運算後方為Layer 0 的結果, 故其結果為2 張尺寸為 (寬)x64(高) pixels 的圖。

Layer 1 要進行最大池化(max-pooling)運算, 須採用2x2 的max-pooling 視窗及步幅(stride)為2 的規格進行。故結果將呈現2 張尺寸為32(寬) x 32(高) pixels 的圖。

最後Layer 2 則是進行平坦化(Flatten)處理, 將Layer 1 輸出的2 張32(寬) x 32(高) pixels 的圖依規格排序成一個2048 (32x32x2)個訊號值的序列並輸出即可完成。

上述所有Layer, 均可分層分次將處理結果寫回testfixture 內建的記憶體內 (如圖2. L0\_MEM0、L0\_MEM1、L1\_MEM0、L1\_MEM1 及L2\_MEM)中, 每一Layer 的輸出結果都有各自對應的記憶體存放空間用於放置輸出結果。



## 2. 設計規格

- 系統方塊圖

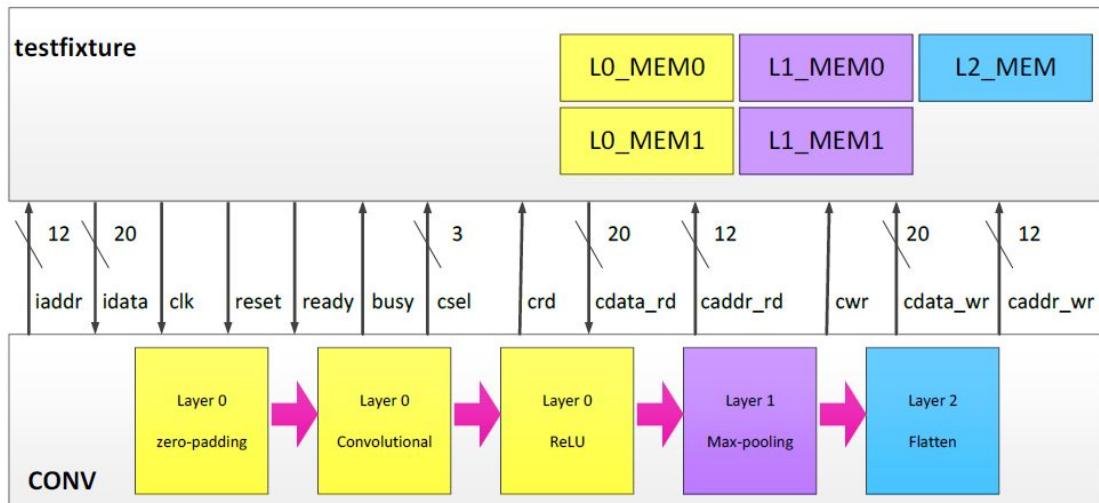


圖 2. 系統方塊圖

- 輸出入訊號和記憶體描述

表一、輸入/輸出信號

| Signal Name | I/O | Width | Simple Description   |
|-------------|-----|-------|--|
| clk         | I   | 1     | 系統時脈訊號。本系統為同步於時脈正緣之同步設計。   |
| reset       | I   | 1     | 高位準“非”同步(active low asynchronous)之系統重置信號。  |
| ready       | I   | 1     | 灰階圖像準備完成指示訊號。當訊號為 High 時，表示灰階圖像準備完成，此時 CONV 才可以開始向 testfixture 發送輸入灰階圖像資料索取位址。   |
| busy        | O   | 1     | 系統忙碌指示訊號。當 CONV 接收到 ready 訊號為 High，且 CONV 準備開始動作時，需將此訊號設為 High，表示準備開始進行輸入灰階圖像資料索取；待所有運算處理完成且輸出結果寫回 testfixture 後，需再將訊號設為 Low 表示動作結束。 |
| iaddr       | O   | 12    | 輸入灰階圖像位址訊號。指示欲索取哪個灰階圖像像素(pixel)資料的位址。  |
| idata       | I   | 20    | 輸入灰階圖像像素資料訊號，由 4 bits 整數(MSB)加上 16 bits 小數(LSB)組成，為有號數。testfixture 將 iaddr 所指示的位址之像素資料用此訊號送給 CONV。                                   |
| crd         | O   | 1     | CONV 運算輸出記憶體讀取效能訊號。當時脈正緣觸發時，若此訊號為 High，表示要進行讀取動作。testfixture 會將 caddr_rd 位址指示之資料讀取到 cdata_rd 上。                                      |
| cdata_rd    | I   | 20    | CONV 運算結果記憶體讀取訊號，由 4 bits 整數(MSB)加上 16 bits 小數(LSB)組成，為有號數。Testfixture 將記憶體資料傳送至 CONV 電路。  |
| caddr_rd    | O   | 12    | CONV 運算結果記憶體讀取位址。CONV 電路各層的運算結果利用此訊號指示將要讀取 testfixture 中所內建各層輸   |

|          |   |    |   |
|----------|---|----|---|
|          |   |    | 出結果之記憶體的哪個位址。   |
| cwr      | O | 1  | CONV 運算輸出記憶體寫入致能訊號。當時脈正緣觸發時，若此訊號為 High，表示要進行寫入動作。testfixture 會將 cdata_wr 內容寫到 caddr_wr 所指示之位址。  |
| cdata_wr | O | 20 | CONV 運算結果記憶體寫出訊號，由 4 bits 整數(MSB)加上 16 bits 小數(LSB)組成，為有號數。CONV 電路各層的運算結果利用此訊號輸出至 testfixture。  |
| caddr_wr | O | 12 | CONV 運算結果記憶體寫入位址。CONV 電路各層的運算結果利用此訊號指示將要寫入到 testfixture 中所內建各層輸出結果之記憶體的哪個位址。  |
| csel     | O | 3  | CONV 運算處理結果寫入/讀取記憶體選擇訊號。此訊號指示目前寫入/讀取資料為 CONV 電路中那一層的運算結果。說明如下：<br>3'b000: 表示沒有選擇記憶體。<br>3'b001: 寫入/讀取 Layer 0，Kernel 0 執行 Convolutional 的結果。<br>3'b010: 寫入/讀取 Layer 0，Kernel 1 執行 Convolutional 的結果。<br>3'b011: 寫入/讀取 Layer 1，將 Kernel 0 執行 Convolutional 後再進行 Max-pooling 運算的結果。<br>3'b100: 寫入/讀取 Layer 1，將 Kernel 1 執行 Convolutional 後再進行 Max-pooling 運算的結果。<br>3'b101: 表示寫入/讀取 Layer 2，Flatten 層的運算結果。 |

- 系統功能、時序及記憶體對應方式描述

系統方塊圖如圖 2. 所示。當 reset 啟動結束後，testfixture 會將 ready 訊號設為 High 表示灰階圖像資料及各層 Kernel 資料都已經準備完畢，接著 CONV 電路須將 busy 訊號設為 High 表示開始動作(如圖 3. t1 時間點)，而 testfixture 偵測到 busy 訊號為 High 後就會將 ready 訊號設為 Low 表示等待 CONV 電路處理題目所要求的動作(如圖 3. t2 時間點)。待各層所要求的動作都實現完成或已經將欲實現的 Layer 動作處理完成後，CONV 電路就可將 busy 訊號再次設為 Low 表示所有動作已經完成(如圖 3. t3 時間點)，此時 testfixture 就會準備下一張灰階圖像並將 ready 訊號設定為 High；並且 testfixture 一偵測到 busy 訊號再次被設定為 Low 就會立刻進行資料驗證比對，因此針對每一張輸入灰階圖像的運算處理，busy 訊號只允許在 CONV 電路開始動作時被設定為 High 一次，CONV 運算處理結束時設定為 Low 一次。在 busy 為 High 的過程中，CONV 電路可重複不限次數對 L0\_MEM0、L0\_MEM1、L1\_MEM0、L1\_MEM1 及 L2\_MEM 進行讀取及寫入動作。

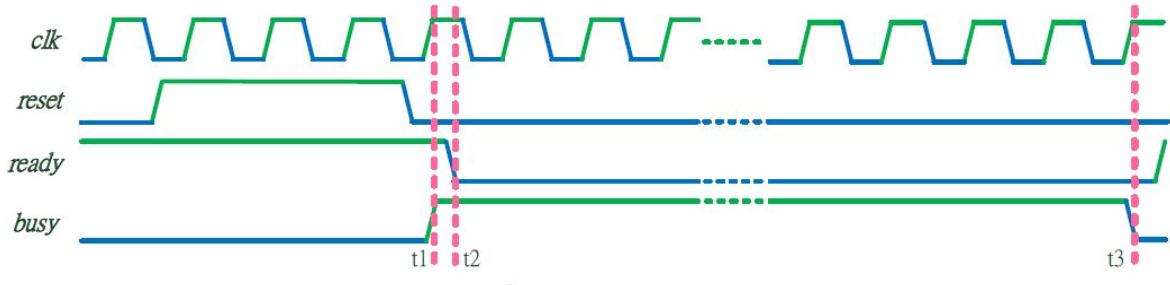


圖 3. 系統啟動及結束時序

關於CONV 電路各層之動作說明如下，整體流程如圖4.所示：

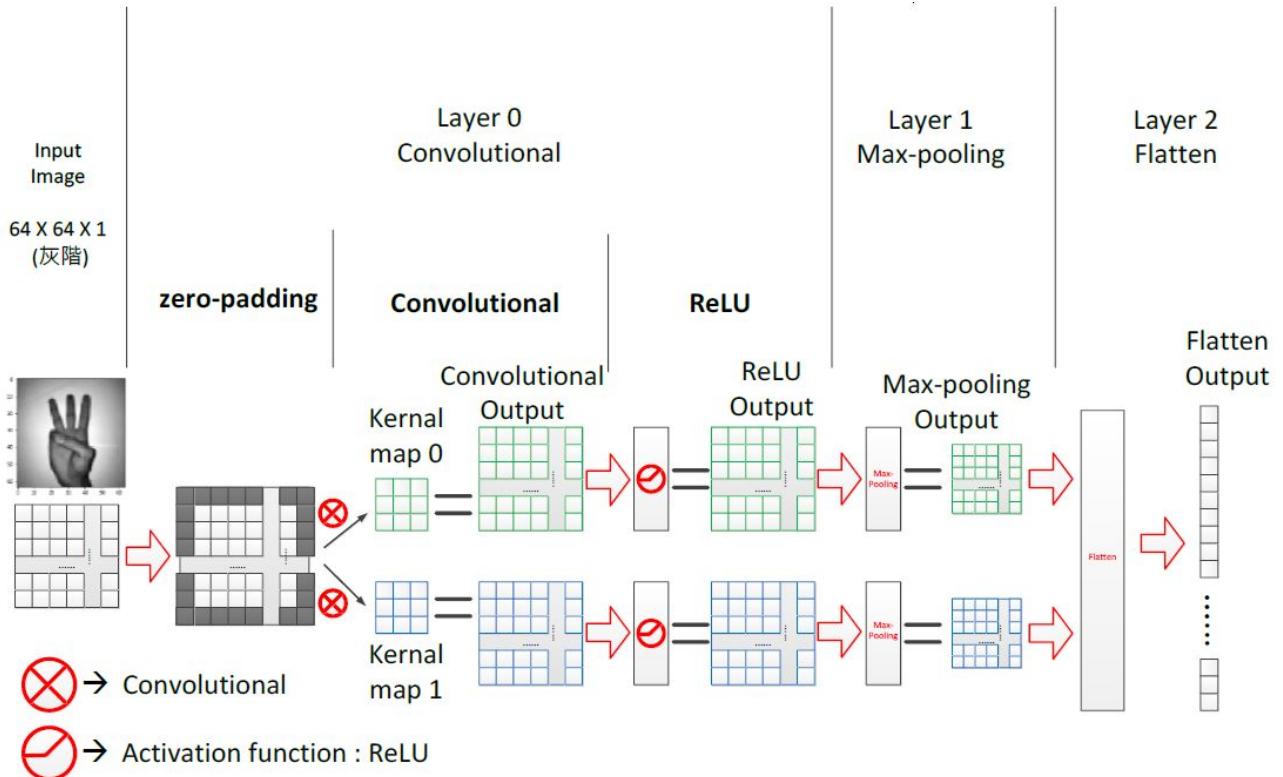


圖 4. CONV 電路系統流程圖

1. Input image 為本題輸入灰階圖像，其尺寸為64 pixels(寬)x 64 pixels(高)，存放於testfixture的記憶體中，灰階圖像各pixels 與其記憶體的對應方式如下圖6.說明。動作時序上，CONV電路需利用iaddr 發送欲索取圖像資料的位址到testfixture(如圖5. t1 時間點)，testfixture在每個時脈負緣後會將iaddr 所指示位址之pixel 資料利用idata 送入CONV電路(如圖5. t2時間點)。

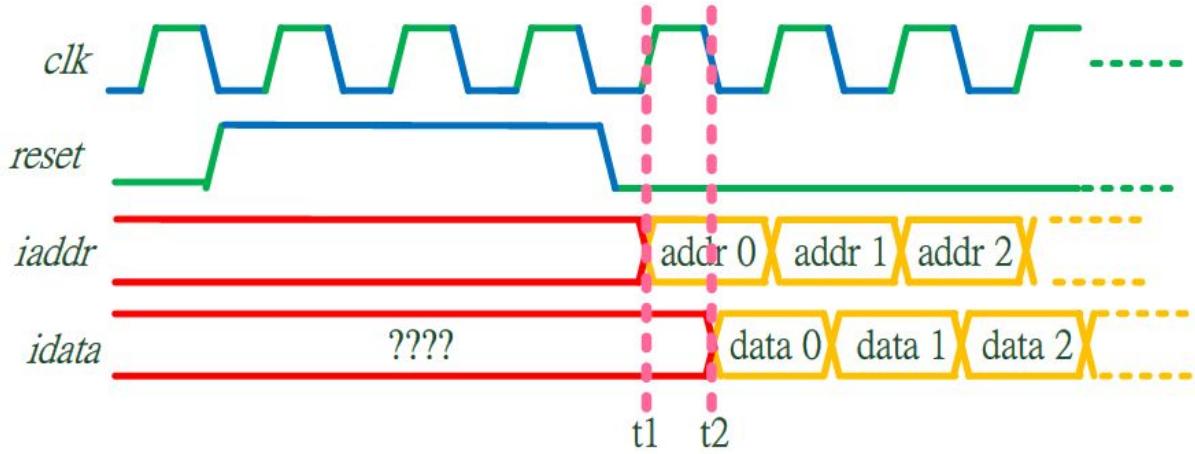


圖 5. 灰階圖像資料記憶體時序

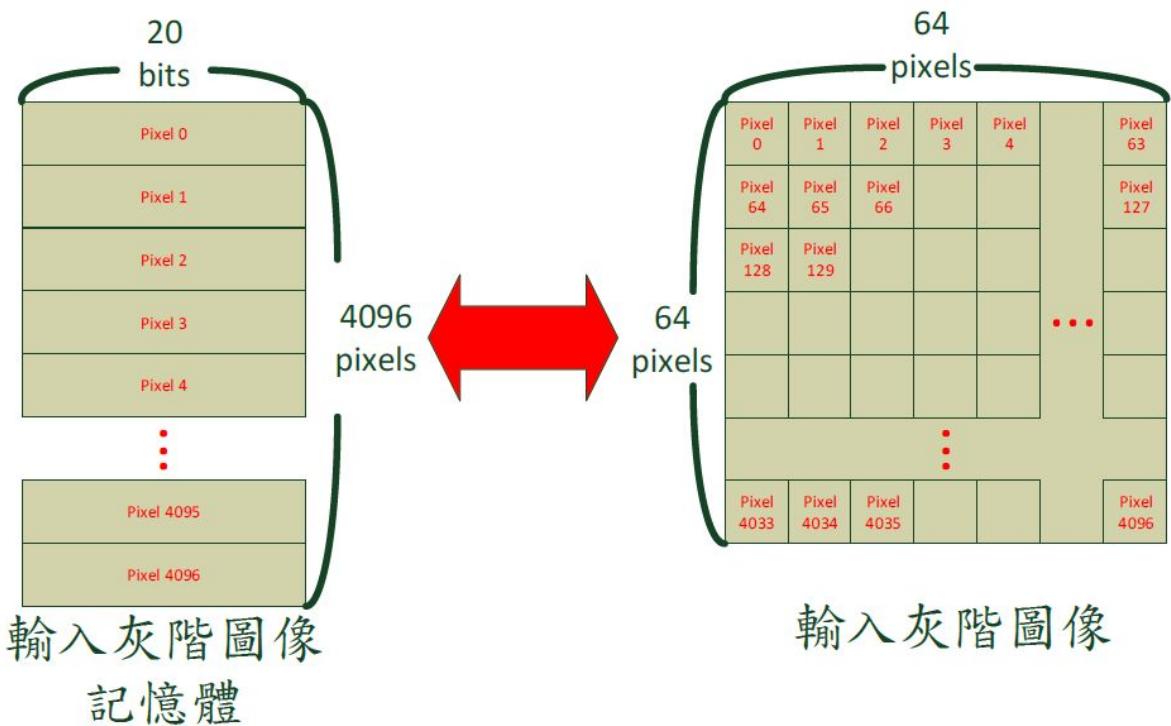


圖 6. 輸入灰階圖像及其記憶體對應方式

2. Layer 0 層所要進行的動作是將input image 進行Zero-padding 處理後，再分別與Kernel 0及Kernel 1 作Convolutional 運算處理後得到2張尺寸皆為64 pixels(寬)x 64 pixels(高)的圖，再將這兩張圖的每個pixels 分別進行ReLU 運算後得到Layer 0 的結果。

#### A. Convolutional

本題規定之Convolutional 處理為針對待處理圖像以固定移動1pixel 的間隔，將 $3 \times 3$  大小的 Kernel 疊到待處理圖上左上角，**如圖7.紅色框為例**，圖中每一個pixel 都跟對應到Kernel 上面的每個pixel 相乘，最後全部相加，得到  $1 \times 2 + 2 \times 0 + 3 \times 1 + 0 \times 0 + 1 \times 1 + 2 \times 2 +$

$3 \times 1 + 0 \times 0 + 1 \times 3 = 16$ , 接著Kernel往右移動一格, 如圖7.綠色框為例, 並做相同的運算得到結果為18, 依此類推依序由左到右, 由上到下移動(如圖7.淺藍色框)計算到所有pixels乘積後可得Convolutional結果(如圖7.中間所示)。最後每個pixel還要再各自加上bias值後才是Convolutional運算的最後輸出(如圖7.最右邊區塊)。

本題給定之Kernel 0 及Kernel 1 分別如圖8.及圖9.所示, Kernel 0 所對應的bias值為( 0.07446326, 10 進制)( 01310, 16 進制, 4bits 整數+16bits 小數) ; Kernel 1 所對應的bias 值為(-0.55241907, 10 進制)( F7295, 16 進制, 4bits 整數+16bits 小數)。

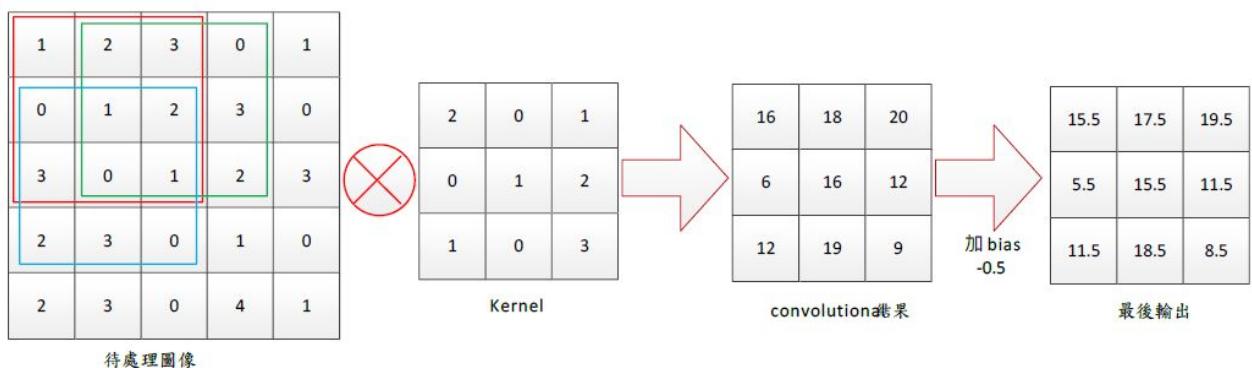


圖 7. Convolutional 運算範例

|           |           |           |
|-----------|-----------|-----------|
| 0.658674  | 0.573572  | 0.42681   |
| 0.0625617 | -0.439696 | -0.569037 |
| -0.34829  | -0.217964 | -0.327755 |

|           |          |           |
|-----------|----------|-----------|
| -0.143247 | 0.162391 | -0.212595 |
| 0.31637   | 0.184082 | 0.125697  |
| 0.23376   | -0.17419 | 0.368789  |

Kernel0 (10進制)

Kernel1 (10進制)

圖 8. 本題給定 Kernel 0 和 Kernel 1 (10 進制)

|       |       |       |
|-------|-------|-------|
| 0A89E | 092D5 | 06D43 |
| 01004 | F8F71 | F6E54 |
| FA6D7 | FC834 | FAC19 |

|       |       |       |
|-------|-------|-------|
| FDB55 | 02992 | FC994 |
| 050FD | 02F20 | 0202D |
| 03BD7 | FD369 | 05E68 |

Kernel0 (16進制)

Kernel1 (16進制)

圖 9. 本題給定 Kernel 0 和 Kernel 1 (16 進制) ( 4bits 整數+16bits 小數)

## B. Zero-padding

如圖7.範例所示，Convolutional 處理之結果會讓圖片尺寸縮小；但若在Convolutional 處理前先進行填補(padding)就可讓圖片尺寸維持相同，本題規定須將待處理圖像周圍填補1 pixel 的0。如圖10.所示為先做zero-padding 補1 pixel 後進行Convolutional 處理的例子。

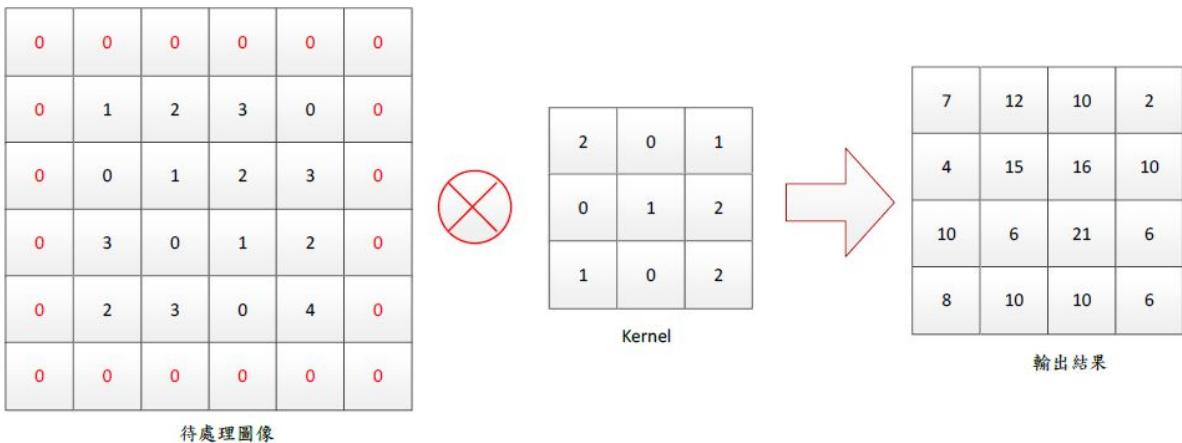


圖 10. Zero-padding 運算範例

## C. ReLU

ReLU(Rectified Linear Unit)可以以下算式表示，如果輸入pixel的值 $x$  小於或等於0則此pixel 的值須調整成 $y$  為0；但如果輸入pixel的值 $x$  大於0 則此pixel 的值仍為 $x$ 。Convolutional 處理後的每一個pixel 的值都要經過ReLU 運算後方為Layer 0 的最終結果。

$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

D. CONV 電路須將Layer 0 的結果共2 張圖，分別用caddr\_wr 指定寫回testfixture 記憶體的位址，運算結果資料用cdata\_wr 送到testfixture，每張圖有各自獨立記憶體因此須設定csel 為3'b001(經Kernel 0)及3'b010(經Kernel 1)來啟動各自的記憶體。其輸出圖與記憶體的對應方式如圖11.所示。其中csel=3'b001所讀寫的記憶體為L0\_MEM0；cesl=3'b010 所讀寫的記憶體為L0\_MEM1。

以上L0\_MEM0 及L0\_MEM1 之資料寬度20bits，且為4bits 整數+16bits 小數，Layer0 實際計算結果可能大於20 bits，參賽者須取整數4bits 及小數點後16 bits 之後做進位(第17 個bit 做四捨五入)作為輸出。

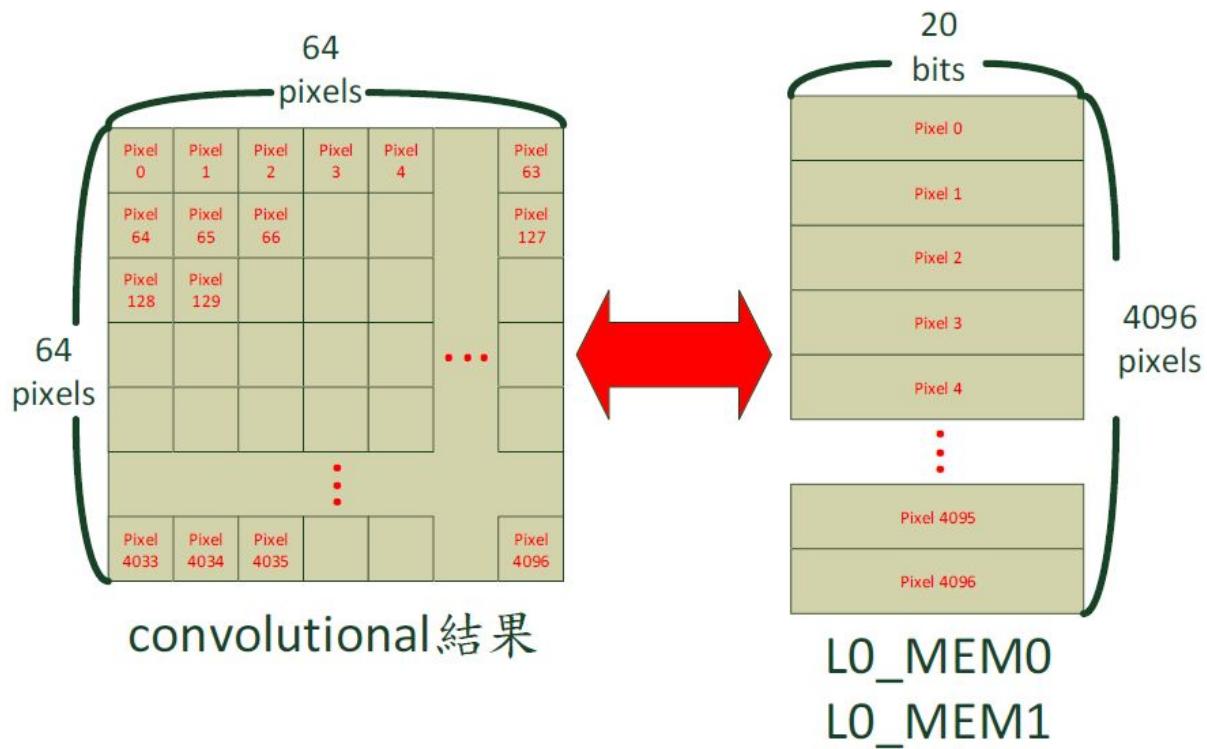
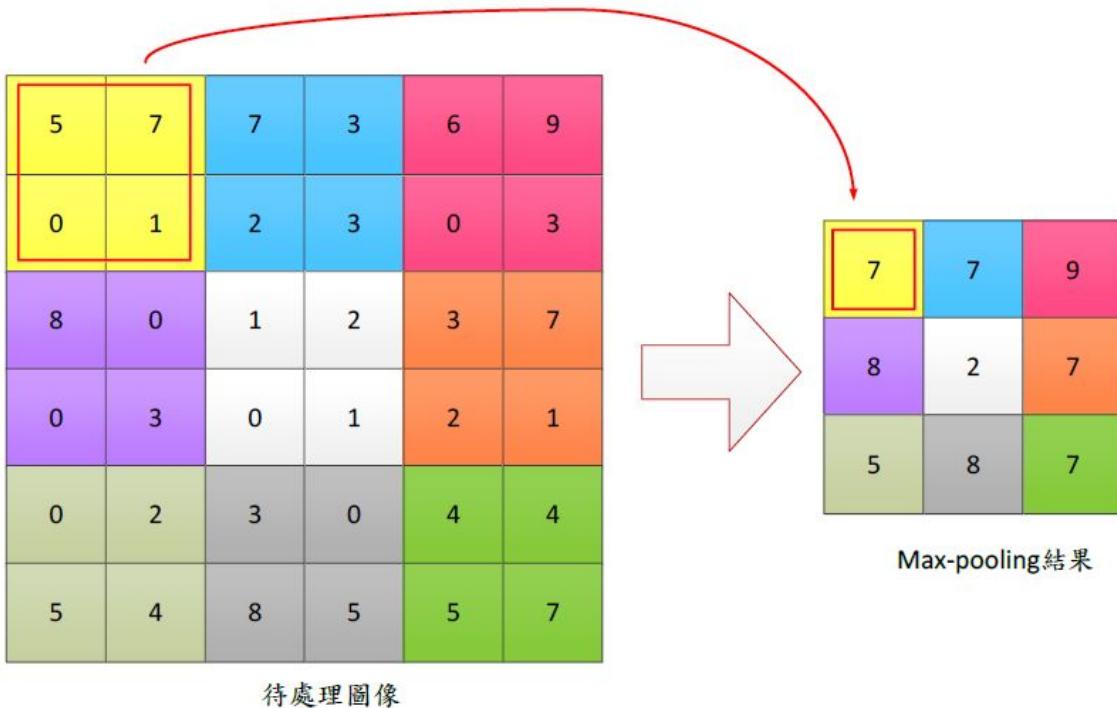


圖 11. L0\_MEM0/L0\_MEM1 與 convolutional 結果對應方式

3. Layer 1 層要進行max-pooling 運算，最大池化層(max-pooling)是縮小水平及垂直空間的運算，本題規定以如下圖所示將2 pixels(寬)X 2 pixels(高)的範圍整合成一元素，縮小空間大小。因此經max-pooling 後的結果圖之尺寸只會有輸入圖尺寸的1/2。

將Layer 0 輸出的2 張64 pixels(寬)x 64 pixels(高)的圖，各自經max-pooling 運算後輸出2 張32 pixels(寬)x 32 pixels(高)的結果圖，max-pooling 的過程如圖12.所示。本題規定max-pooling 視窗大小為 $2 \times 2$ ，將輸入Layer 1 的目標圖像依序由左到右，由上到下，步幅(stride)為2，依序取 $2 \times 2$  大小的pixels，取最大值為其輸出，因此圖12.左上角的 $2 \times 2$  pixels(黃色區域)為 $\text{max}(5, 7, 0, 1)$ ，因此結果為最大值7；接著往右移動2格視窗(淺藍色區域)計算 $\text{max}(7, 3, 2, 3)$ 得最大值為7，其餘依此類推。



待處理圖像

圖 12. max-pooling 運算說明範例

**max-pooling** 後的2張結果圖結果分別用caddr 指定寫回testfixture 記憶體的位址，運算結果資料用cdata 送到testfixture，每張圖有各自獨立記憶體因此須設定csel 為3'b011(經Kernel 0)及3'b100(經Kernel 1)來啟動各自的記憶體。其輸出圖與記憶體的對應方式如**圖13.**所示。其 csel=3'b011 所讀寫的記憶體為L1\_MEM0；cesl=3'b100 所讀寫的記憶體為L1\_MEM1。

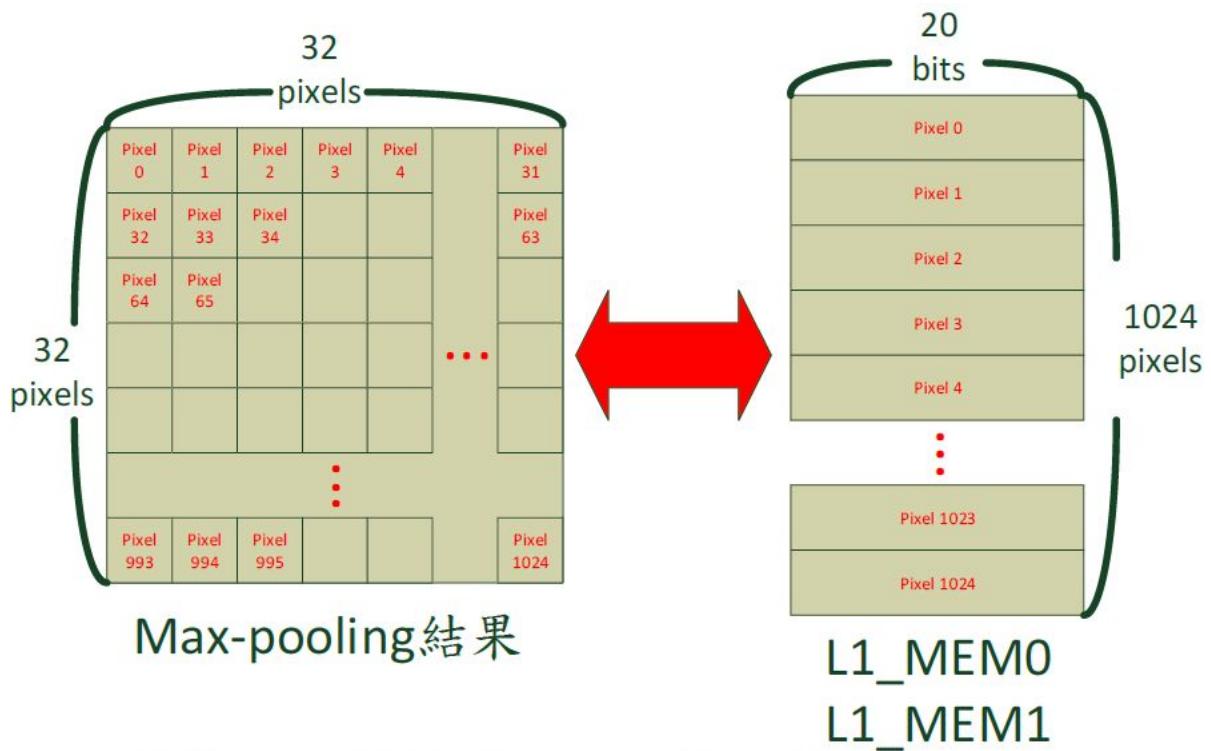


圖 13. L1\_MEMORY 與 max-pooling 結果對應方式

4. Layer 2 則是平坦化(flatten)運算。將Layer 1 的輸出結果共2 張 $32 \times 32$  pixels 結果圖依序”Kernel 0 Kernel 1”及”pixels 累加”排列成一長度為2048 筆的pixels 的資料向量(如下圖14.所示)，再利用csel 設定為3'b101，用caddr 指定L2\_MEMORY 的位址，由位址0 開始依序將2048 筆資料用cdata 送出寫到testfixture。

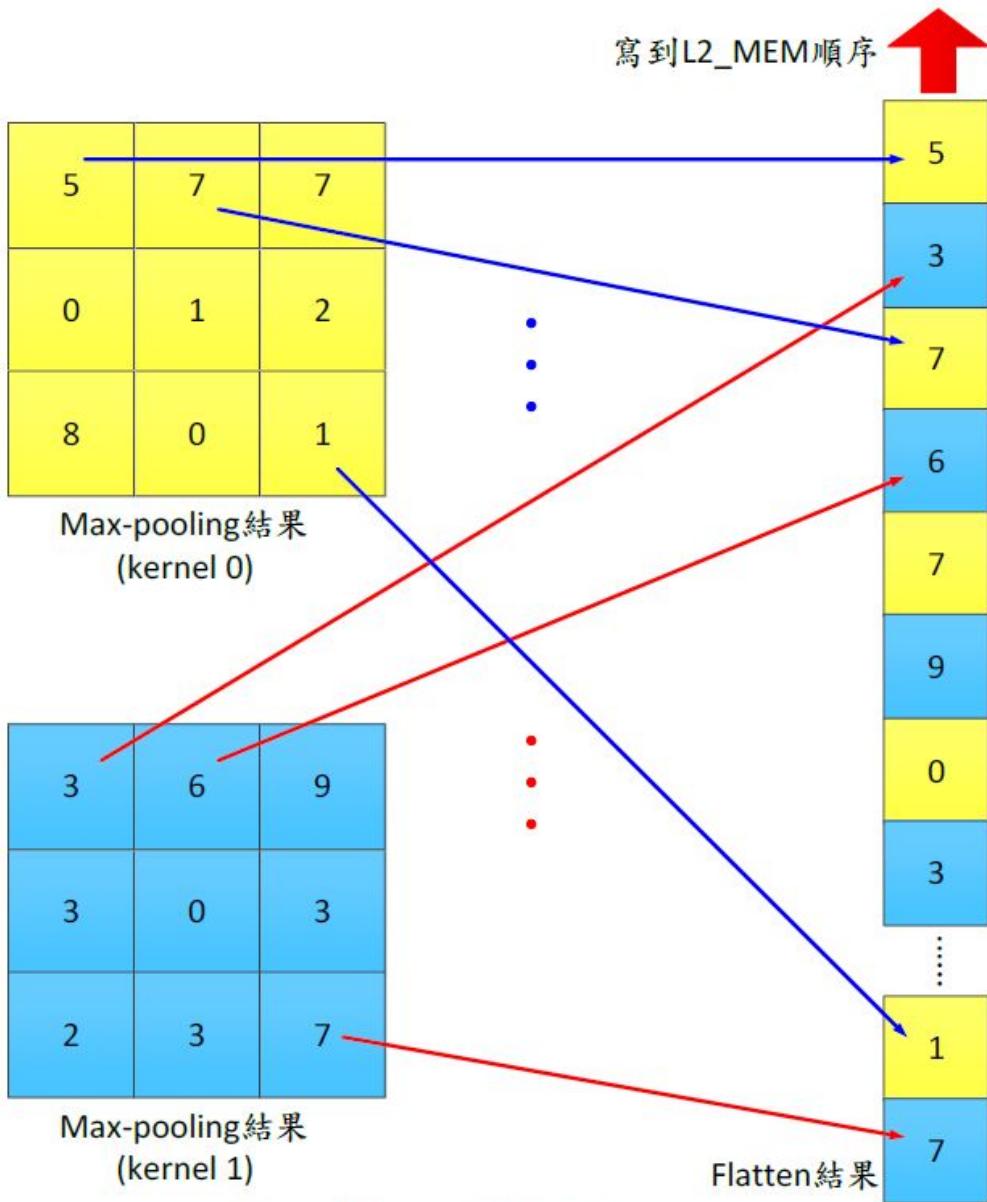


圖 14. Flatten 方式範例

特別注意：以上所述之L0\_MEMORY、L0\_MEMORY1、L1\_MEMORY、L1\_MEMORY1 及L2\_MEMORY 之記憶體寬度皆為20 bits(由4 bits 整數(MSB)加上16 bits 小數(LSB)組成)。

- L0\_MEMORY、L0\_MEMORY1、L1\_MEMORY、L1\_MEMORY1 及L2\_MEMORY 之動作時序

各層輸出資料記憶體L0\_MEMORY、L0\_MEMORY1、L1\_MEMORY、  
L1\_MEMORY1 及L2\_MEMORY 皆為RAM model 且控制方式及時序皆相同，  
都可進行寫入及讀取動作。採用不同的csel 設定值啟動各層輸出相  
對應的記憶體，使用cwr 作為寫入致能訊號，crd 作為讀取致能訊  
號。

讀取時，使用caddr\_rd 為記憶體位址， cdata\_rd 作為讀取資料訊號。動作時序如下圖15. 說明，當時脈負緣觸發時若crd 為High，則會在觸發後立刻將caddr\_rd 所指示位址的資料讀取到cdata\_rd 上(如圖15. t1 時間點)。

寫入時，使用caddr\_wr 為記憶體位址， cdata\_wr 作為寫入資料訊號。動作時序如下圖16. 說明，當時脈正緣觸發時若cwr 為High，則會將這時cdata\_wr 的資料寫入到caddr\_wr 所指示位址上(如圖16. t1 時間點)。

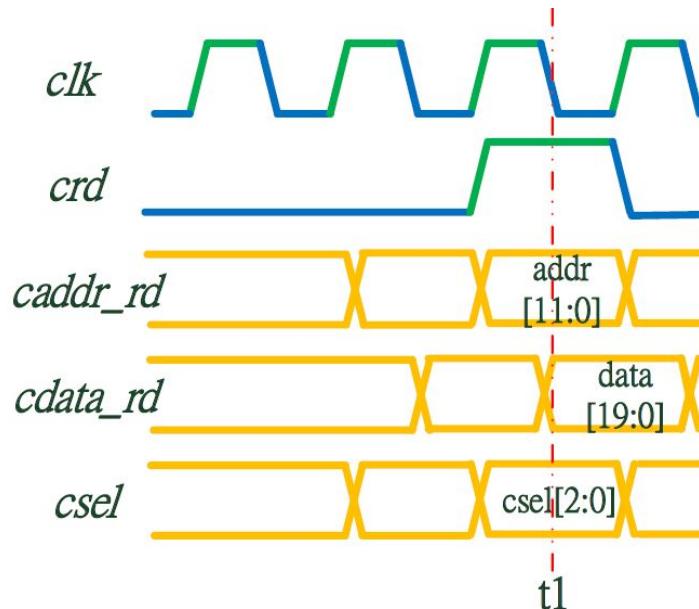


圖 15. 輸出資料記憶體 L0\_MEM0、L0\_MEM1、L1\_MEM0、L1\_MEM1 及 L2\_MEM 讀取動作時序圖

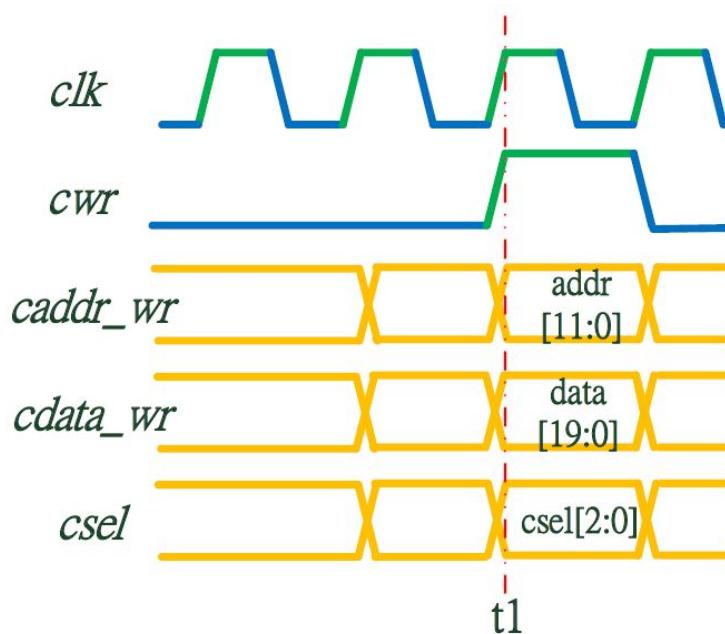


圖 16. 輸出資料記憶體 L0\_MEM0、L0\_MEM1、L1\_MEM0、L1\_MEM1 及 L2\_MEM 寫入動作時序圖

檔案說明如下表所示

| 檔名                             | 說明                                  |
|--------------------------------|-------------------------------------|
| testfixture.v                  | 測試樣本檔。此測試樣本檔定義了時脈週期與測試樣本之輸入及預期輸出信號。 |
| CONV.v                         | 參賽者所使用的設計檔，已包含系統輸/出入埠之宣告            |
| ./dat_grad/cnn_sti.dat         | 測試樣本檔案                              |
| ./dat_grad/cnn_layer0_exp0.dat | Layer 0 比對樣本檔案(Kernel 0)            |
| ./dat_grad/cnn_layer0_exp1.dat | Layer 0 比對樣本檔案(Kernel 1)            |
| ./dat_grad/cnn_layer1_exp0.dat | Layer 1 比對樣本檔案(Kernel 0)            |
| ./dat_grad/cnn_layer1_exp1.dat | Layer 1 比對樣本檔案(Kernel 1)            |
| ./dat_grad/cnn_layer2_exp.dat  | Layer 2 比對樣本檔案                      |

## 評分方式

Report [30%]

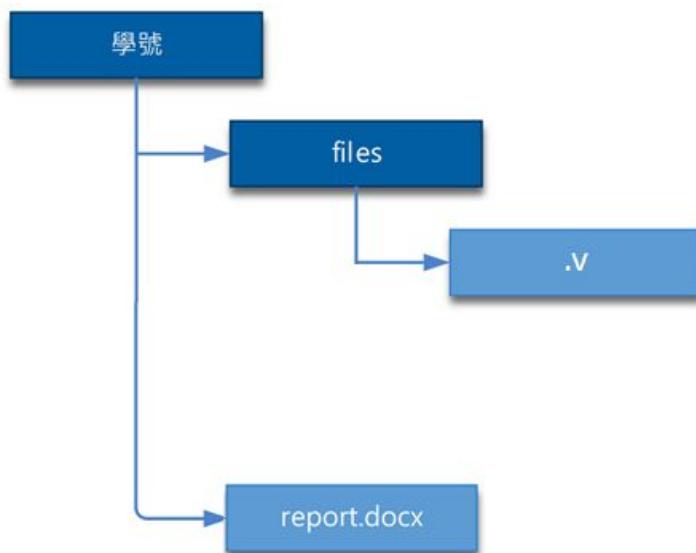
Functional Simulation (pre-sim) [70%]

完成CONV.v之所有功能，且simulation (使用modelsim 或 ncverilog 皆可) 結果無任何ERROR (需全部PASS，無部份給分)

## 作業繳交格式

本作業同學須完成電路設計與報告撰寫，請依照下圖的資料夾格式來繳交完成的檔案

1. 繳交檔名為學號.zip，裡面會有一個自己學號的資料夾
2. 學號的資料夾中，會有一個report.docx和一個files的資料夾
3. files中會有所有的.v檔
4. 如未照上述方式擺放或檔名不符，扣5分
5. 報告內容請參考report.docx



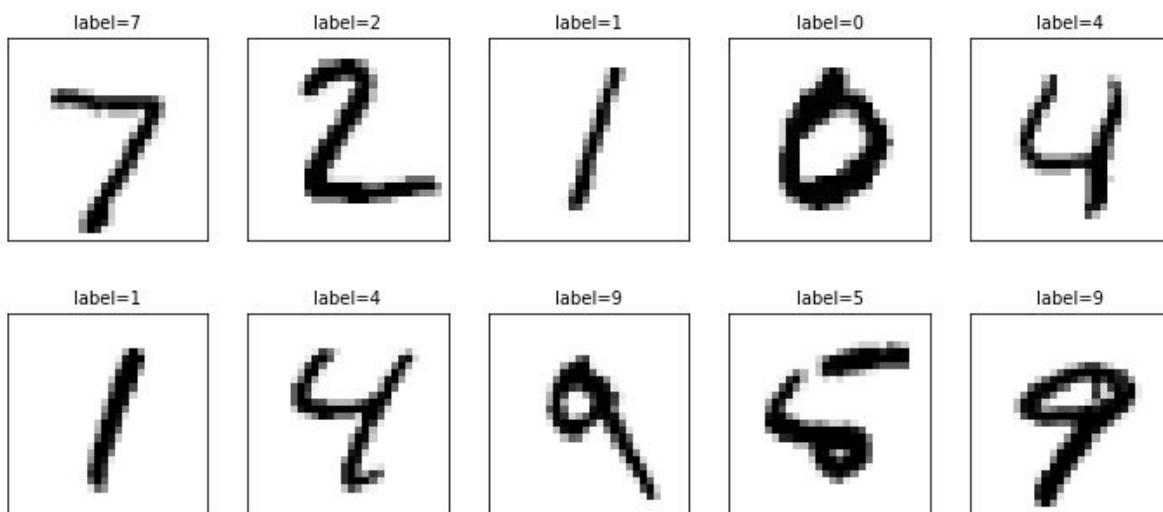
# LAB 7 : MNIST影像辨識系統

## 作業說明

此作業同學必須完成整個MNIST辨識系統，系統需完整辨識MNIST測試資料並將結果顯示出來，作業會提供一個已經訓練好的硬體神經網路，同學須根據其硬體需求來建置系統，作業會提供input、weight和bias等相關檔案，同學必須用pynq中的gcc編譯作業提供的C++ code，並用python燒入硬體，與軟體端的fully connected部分結合，完成整個系統。

## MNIST手寫數字辨識

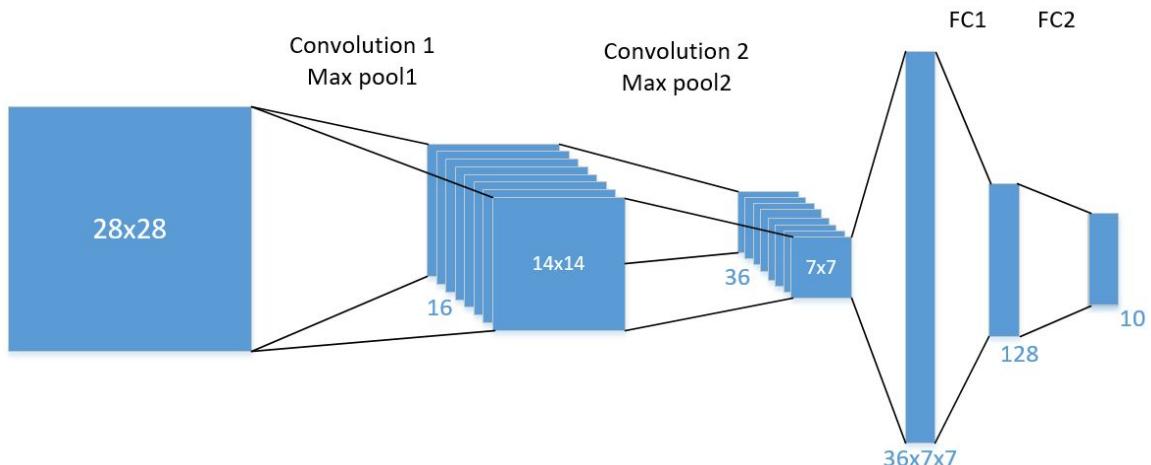
MNIST為一個手寫數字的圖片資料庫，每張圖片大小為 $28 \times 28$ 個pixel，每個pixel用8bits表示(0到255)，且都有對應的label，表示那張圖片真正表達的數字(0到9)，共有60000個training data、10000個testing data，在神經網路的模型下，只要少量的convolution和fully connected層神經網路(一般不大於3層)就有99%以上的準確度。



## 神經網路架構

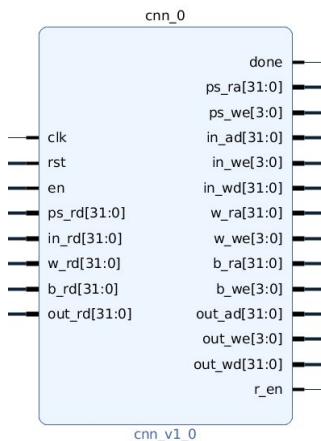
此作業的神經網路由軟體和硬體構成，硬體上，架構有兩層的convolution層，每一層convolution層都有經過zero padding，所以輸出的前兩維度大小與輸出一樣，每層convolution層後面都有一層 $2 \times 2$ 的max pooling層，最後硬體端輸出的結果為 $7 \times 7 \times 36$ 過完convolution的資料。軟體端主要是做flatten和fully connected的運算，fully connected共有兩層，使最後的資料輸出為10，代表數字0到9的機率，機率最高的即為辨識結果。此神經網路在MNIST資料上有超過99%的準確度，神經網路的架構圖如下：

| layer         | function               | output size |
|---------------|------------------------|-------------|
| input         | input資料(MNIST圖片)       | 28*28       |
| convolution 1 | 3*3 convolution 運算     | 28*28*16    |
| max pool 1    | 2*2 max pooling運算      | 14*14*16    |
| convolution 2 | 3*3 convolution 運算     | 14*14*36    |
| max pool 1    | 2*2 max pooling運算      | 7*7*36      |
| FC1           | fully connected運算(軟體端) | 128         |
| FC2           | fully connected運算(軟體端) | 10          |



## CNN運算ip

此作業所有硬體.v檔案都在hardware資料夾中，請同學以cnn.v為主模組，將所有.v檔包成一個vivado ip。包好的ip如下圖所示：



# 系統架構

由於此作業硬體直接提供，所以系統架構必須按照CNN的電路來建置。系統中會用到5個Block memory(簡稱Bram)和兩組AXI GPIO，5個Bram分別如下：

1. parameter Bram：放置神經網路運算相關參數
2. input Bram：放置輸入資料
3. weight Bram：放置已經訓練好的weight
4. bias Bram：放置已經訓練好的bias
5. output Bram：放置輸出資料

每個Bram大小最好都設置在64KB，同學可在Address Editor的range中做設置，vivado就會根據你設置的address range去調整Bram的大小。如同作業4，每個Bram都會有兩組port(true dual ports)，一組接到已經包裝好的CNN ip下，另一組接到cdma，使CNN ip可以透過Bram和ZYNQ(軟體端)做溝通。

另外兩組GPIO的資訊如下：

1. en : 1bit, 相當於start, ZYNQ告訴CNN ip資料已經放完，可以開始運算
2. done : 1bit, 相當於finish, CNN ip告訴ZYNQ已經運算完畢。可以去output Bram拿取運算結果。

所有CNN ip對應的訊號如下表：

| 訊號    | 簡介                     | 銜接處                  |
|-------|------------------------|----------------------|
| clk   | clock                  | system clock         |
| rst   | posedge reset          | system posedge reset |
| en    | start                  | axi gpio             |
| done  | finish                 | axi gpio             |
| r_en  | all Bram enable        | all Bram             |
| ps_ra | parameter address      | parameter Bram       |
| ps_rd | parameter read data    |                      |
| ps_we | parameter write enable |                      |

|        |                     |             |
|--------|---------------------|-------------|
| in_ad  | input address       | input Bram  |
| in_rd  | input read data     |             |
| in_we  | input write enable  |             |
| in_wd  | input write data    |             |
| w_ra   | weight address      | weight Bram |
| w_rd   | weight read data    |             |
| w_we   | weight write enable |             |
| b_ra   | bias address        | bias Bram   |
| b_rd   | bias read data      |             |
| b_we   | bias write enable   |             |
| out_ad | output address      | output Bram |
| out_rd | output read data    |             |
| out_we | output write enable |             |
| out_wd | output write data   |             |

## pynq jupyter

此作業的軟體端主要是C++ code, pynq jupyter只負責燒入硬體, 所以只要一行overlay燒入硬體即可, 請同學自行完成。

## C++ code

所有的C++ code和MNIST資料都放在software中, 請同學將整個資料夾放置到pynq開發板中。其中dma\_driver.h的以下部分同學必須進行修改, 確認作業提供的硬體driver上面的address都有對應到硬體上的address。

```
#define cdma_BASE 0x7E200000

#define P_BASE    0xC0000000
#define IN_BASE   0xC8000000
#define OUT_BASE  0xC2000000
#define W_BASE    0xC6000000
#define B_BASE    0xC4000000

#define zynq_BASE 0x30000000

#define done_BASE 0x41200000
#define en_BASE   0x41210000
```

define與address的對照表如下：

| define    | 對應address的ip    |
|-----------|-----------------|
| cdma_BASE | cdma的slave      |
| P_BASE    | parameter Bram  |
| IN_BASE   | input Bram      |
| OUT_BASE  | output Bram     |
| W_BASE    | weight Bram     |
| B_BASE    | bias Bram       |
| zynq_BASE | zynq slave(HP0) |
| done_BASE | done AXI GPIO   |
| en_BASE   | en AXI GPIO     |

完成軟體修正後，在pynq開發板上輸入以下指令

```
$sudo make
```

之後輸入一個0到9999的數字(輸入超過範圍，程式結束)

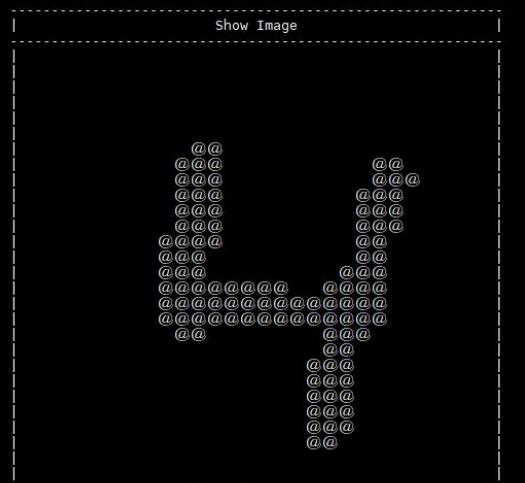
就會看到以下結果輸出在螢幕上。

Pred = 神經網路辨識出來的數字

True = 實際數字

```
Please input a number (0 ~ 9999): 56
=====
Hardware Accelerator Start =====
-- Conv Start --
-- Conv Finished --
-- Conv Start --
-- Conv2 Finished --
===== Hardware Accelerator Finish =====
=====
Software (Full-Connected) Finish =====
=====
Pred: 4 True: 4 =====

| Show Image |
|-----|-----|
|-----|-----|
```



# 作業繳交格式

本作業同學須完成系統設計與報告撰寫，請依照下圖的資料夾格式來繳交完成的檔案

1. 繳交檔名為**學號.zip**，裡面會有一個自己學號的資料夾
2. 學號的資料夾中，會有一個**report.docx**和一個**files**的資料夾
3. **files**中會有三個檔案，分別是**.tcl** 和 **.bit** 以及 **.ipynb**
4. 報告內容請參考**report.docx**

