

Communication Training System

Workbook

Table of Contents

- **Introduction**
- **Adding Configuration Parameter Stream**
 - **Vitis HLS Testbench**
 - **Vitis HLS Kernel Block**
 - **Jupyter Notebook Host Code**
- **New IP Integration**
 - **RTL Kernel IP**
 - **Composable Pipeline Connection in Vivado**
 - **Jupyter Notebook Host Code**
- **Lab**
 - **Error Correction Code**
 - **QAM RTL code**

Introduction

此系統為一 end-to-end Single-Input-Single-Output Orthogonal Frequency-Division Multiplexing (SISO-OFDM) 通訊系統，目的在於觀察不同演算法對整體系統造成的效果，並由於 FPGA real time的特性，可以即時觀察到不同參數對系統帶來的變化。

在接下來的章節中會敘述parameter stream的使用方法以及如何新增新的IP至原有系統，最後會有兩個lab提供實作練習。

Adding Configuration Parameter

由於此系統輸入和輸出都各只有一條stream，因此參數和輸入訊號會使用同一條stream做傳輸，在傳送input data之前會先傳送各個moduel需要的參數。

單一參數由64 bits所構成，其中包含16 bits的module編號，16 bits的module內參數編號，以及32 bits的參數數值，示意如下圖所示：

module_id 16 bits	para_id 16 bits	para_val 32 bits
----------------------	--------------------	---------------------

以下分別為在kernel內部、vitis_hls的testbench以及host端parameter stream的建構方法：

。 Vitis HLS Testbench

以下為在vitis_hls進行CSIM以及COSIM驗證時testbench中parameter stream輸入設定的範例：

```
36 ap_axiu_64 para_in;
37 ap_uint<16> module_id;
38 ap_uint<16> para_id;
39 ap_uint<32> para_val;
40
41 module_id = QAM_module_num; //給定參數所屬模組的id
42 para_id = 0; //給定參數在所屬模組中的id
43 para_val = DATA_LEN; //給定參數數值
44 para_in.data = (module_id, para_id, para_val); //合併所有資訊
45 data_in.write(para_in); //寫入input data stream
```

需要分別設定module_id, para_id以及para_val的數值，將資料合併成64-bit的形式之後即可寫入input stream。

。 Vitis HLS Kernel Block

以下為在kernel內部加入parameter stream讀取的使用方法：
kernel分成兩部分，第一部分為參數的讀取，第二部分為kernel主要功能的運作，在讀取到終止條件module_id == 16'hFFFF後，便會離開參數讀取的迴圈開始kernel的運行，當module id等於目前module的id時，代表該參數是目前module所屬的參數，因此會將他讀入module中，接著再根據para_id以及para_val對對應參數進行設定，若module id不等於目前module的id，則代表該參數不屬於目前的module，會寫入output stream傳入下一個module，最後判斷是否為終止條件(module_id = 16'hFFFF, para_id = 16'b0, para_val = 32'b0)，要注意終止條件也需要被寫入下一個module。

。 Jupyter Notebook Host Code

以下為host code的說明：

- (a) module_id : 填入會使用到的module對應的module_id, 擺放順序隨意。
 (b) len_parameter_id : 每個module中所會用到的參數個數, 順序需和module_id的位置對應。

```

21 module_id = np.array([0,2,3,4,5,6,7,8,9,10,12])
22 len_module_id = len(module_id)
23 len_parameter_id = [4,4,4,3,4,6,4,3,5,4,4] # # of parameters in each module
24 parameter_sum = sum(len_parameter_id)

```

- (c) 給定會使用到的參數數值。

```

26 data_len = 480
27 qam_num = 16
28 sym_num = 2
29 pilot_width = 4
30 CP_length = 16
31 TAPS_NUM = 1
32 SNR = 30
33 FFT_len = 64

```

- (d) parameter_id : module順序需與module_id對應

```

35 parameter_id = np.array([[data_len,qam_num,sym_num,pilot_width], # module_id = 0 pixl2sym
36 [data_len,qam_num,sym_num,pilot_width], # module_id = 2 QAM
37 [data_len,sym_num,pilot_width,CP_length], # module_id = 3 pilot insert
38 [data_len,sym_num,pilot_width], # module_id = 4 IFFT
39 [data_len,sym_num,pilot_width,CP_length], # module_id = 5 CP insert
40 [data_len,sym_num,pilot_width,CP_length,TAPS_NUM,SNR], # module_id = 6 channel
41 [data_len,sym_num,pilot_width,CP_length], # module_id = 7 CP remove
42 [data_len,sym_num,pilot_width], # module_id = 8 FFT
43 [data_len,sym_num,pilot_width,CP_length, TAPS_NUM], # module_id = 9 pilot remove
44 [data_len,qam_num,sym_num,pilot_width], # module_id = 10 deQAM
45 [data_len,qam_num,sym_num,pilot_width]],dtype=object)# module_id = 12 sym2pixl

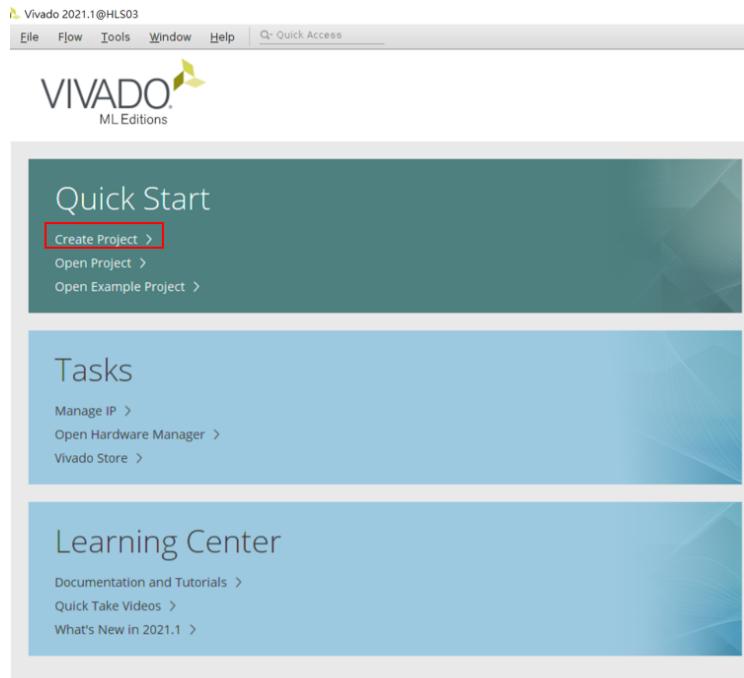
```

New IP Intergration

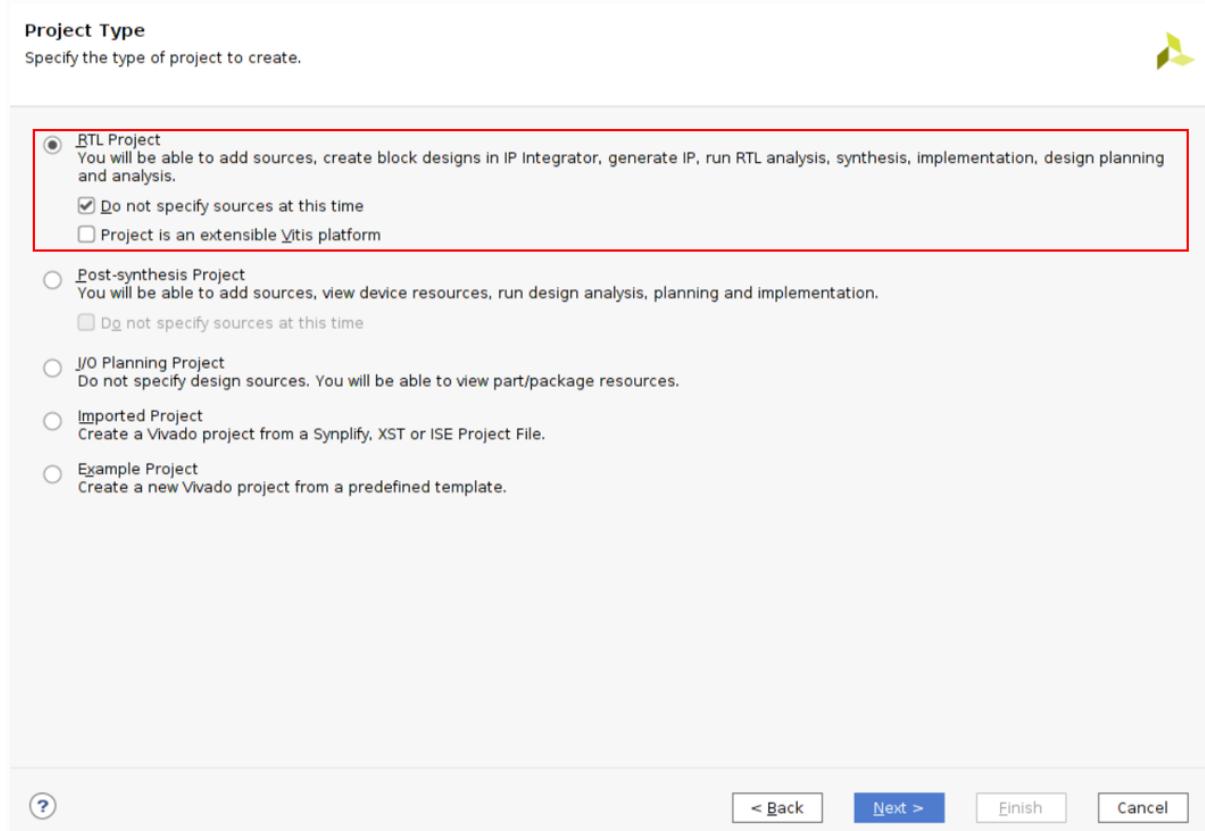
RTL source code到能夠在FPGA上面實現需要通過以下幾個步驟：

◦ Create vivado project

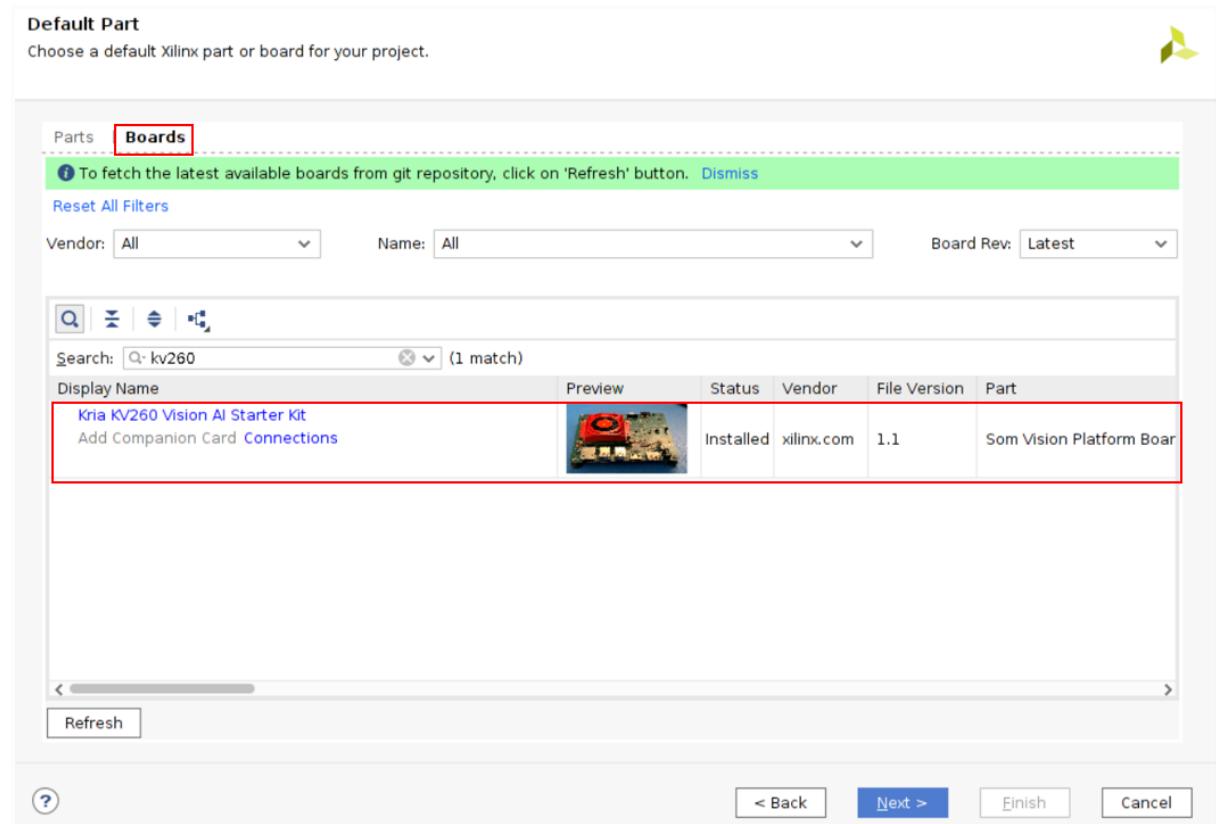
啟動vivado開發套件，並開啟新的專案，設定好專案存放的路徑：



選擇RTL Project, 勾選Do not specify sources at this time, 接著按Next>:



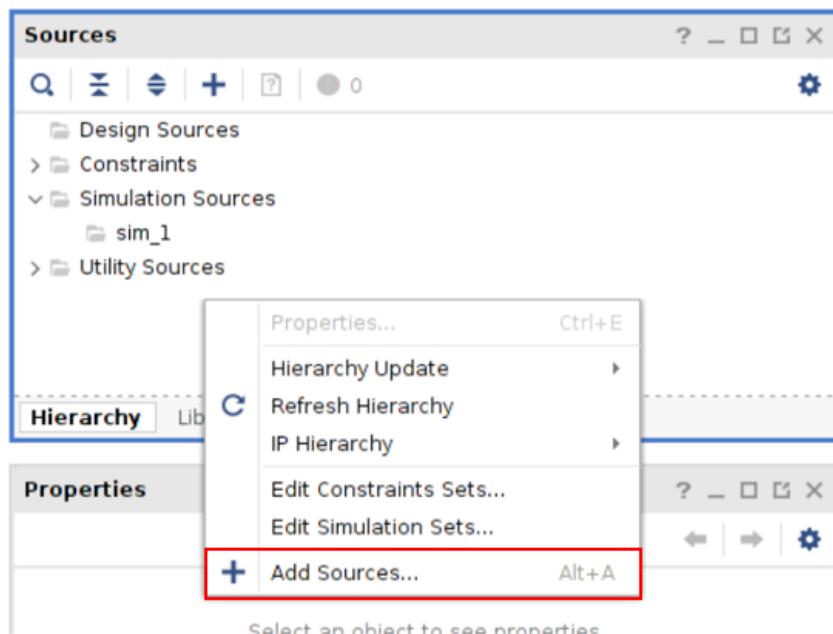
選擇Boards, 並在下方選 Kria KV260 Vision AI Started Kit, 點選Next>:



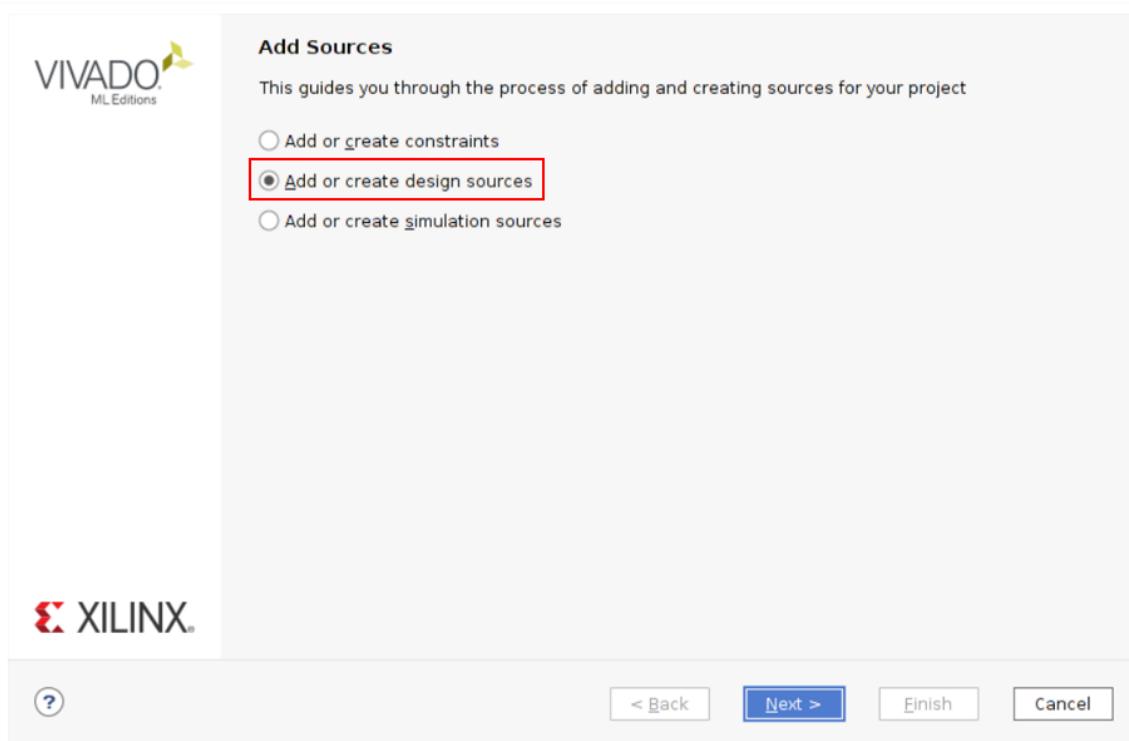
最後點選Finish, 完成Create Project。

>Create design source code

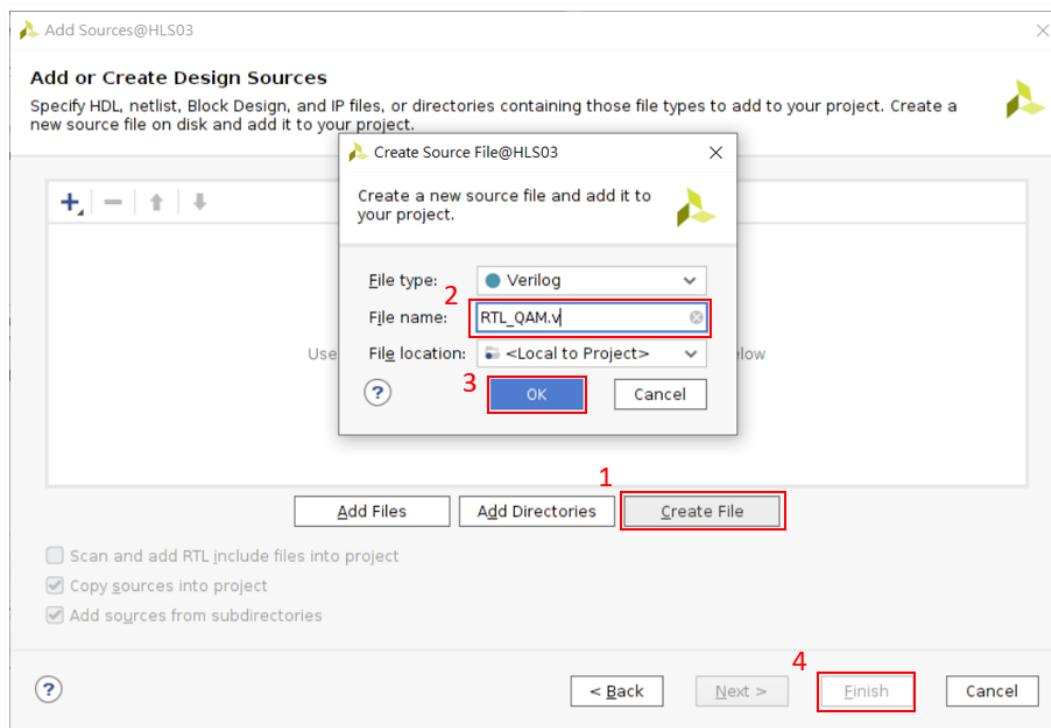
在Sources的窗口空白處按右鍵, 點選Add Sources...

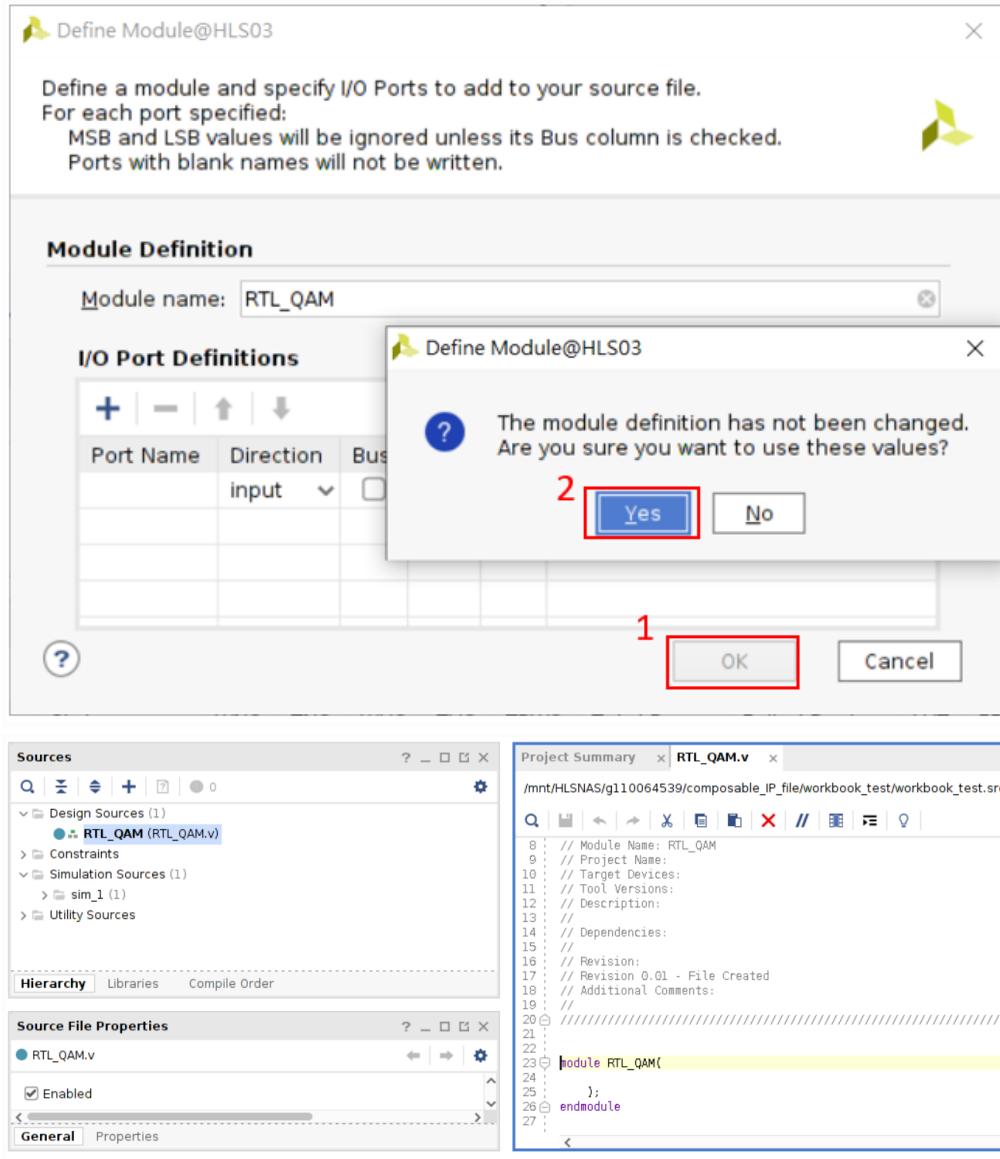


選擇Add or create design sources, 點選Next >



點選Create File, 並設定verilog檔名, 注意檔名後面需要加.v。

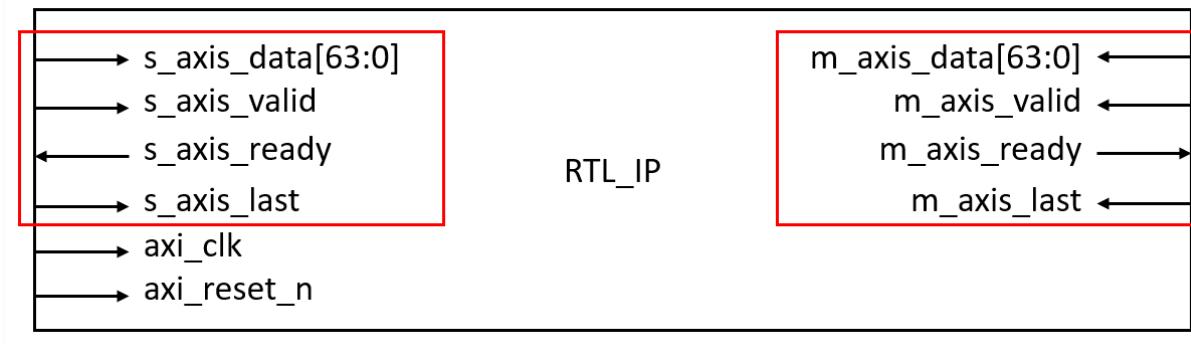




出現剛創建的verilog source code, 雙擊.v檔, 即可開始編寫verilog。

Design verilog code

我們開始設計RTL, 首先先確定好input及output的訊號, 下圖是AXI4-stream 的ports:



因此verilog程式碼 input及output ports:

```
23 module RTL_QAM #(parameter DATA_WIDTH=64,
24   parameter IN_WL=22,
25   parameter IN_IL=7)
26   (
27     input  axi_clk,
28     input  axi_reset_n,
29     //AXI4-S slave i/f
30     input  s_axis_valid,
31     input [DATA_WIDTH-1:0] s_axis_data,
32     input s_axis_last,
33     output s_axis_ready,
34     //AXI4-S master i/f
35     output reg m_axis_valid,
36     output reg [DATA_WIDTH-1:0] m_axis_data,
37     output reg m_axis_last,
38     input  m_axis_ready
39   );
```

接下來是AXI-Stream 的接口設計:

```
40
41   reg [DATA_WIDTH-1:0] n_m_axis_data;
42   reg cs,ns;
43   /////////////////////////////////
44   reg [1:0] state;
45   reg [DATA_WIDTH-1:0] s_axis_reg;
46   reg s_axis_valid_reg;
47
48   always @{posedge axi_clk or negedge axi_reset_n} begin    //
49     if(~axi_reset_n) begin
50       m_axis_data <= 0;
51       m_axis_valid <= 0;
52       cs <= 0;
53       state <= 0;
54       s_axis_reg <= 0;
55       s_axis_valid_reg <= 0;
56       m_axis_last <= 0;
57     end
58   end begin
59     case(state)
60       // ?0?0J0H?
61       2'd0: begin
62         if (s_axis_valid) begin
63           s_axis_reg <= s_axis_data;
64           s_axis_valid_reg <= 1;
65           state <= 1;
66         end
67       end
68       // ?0?0G
69       2'd1: begin
70         m_axis_data <= n_m_axis_data;
71         m_axis_valid <= 1;
72         m_axis_last <= s_axis_last;
73         state <= 2;
74         cs <= ns;
75       end
76       // ?000?0X0H?0000?
77       2'd2: begin
78         if (m_axis_ready) begin
79           s_axis_valid_reg <= 0;
80           m_axis_valid <= 0;
81           state <= 0;
82         end
83       end
84     endcase
85   end
86 endmodule
```

```

82     end
83     end
84     endcase
85   :
86   end
87 end
88 //-----combination-----//
89 assign s_axis_ready = (state == 0);

```

使用三個狀態的FSM來控制AXI stream的傳輸,

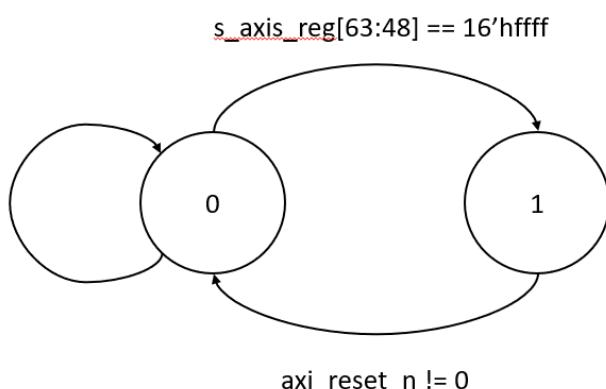
state = 0 : 在等input的stream傳s_axis_data及s_axis_valid過來, 並且s_axis_ready訊號為1, 若收到及s_axis_valid = 1, 會進入下一個狀態。

state = 1 : 做運算, 最後將要邏輯運算的結果給m_axis_data, m_axis_valid設為1, 並且將s_axis_last傳給m_axis_last,

state = 2 : 等待下一級的block的ready訊號, m_axis_ready = 1, 才將m_axis_valid設為0, 並回到state = 0。

下面是此電路要做的功能, 這邊是以16-QAM當作例子,

由於訊號一開始是parameter stream, 傳完後才開始做IP的功能, 因此需要用finite state machine(FSM):



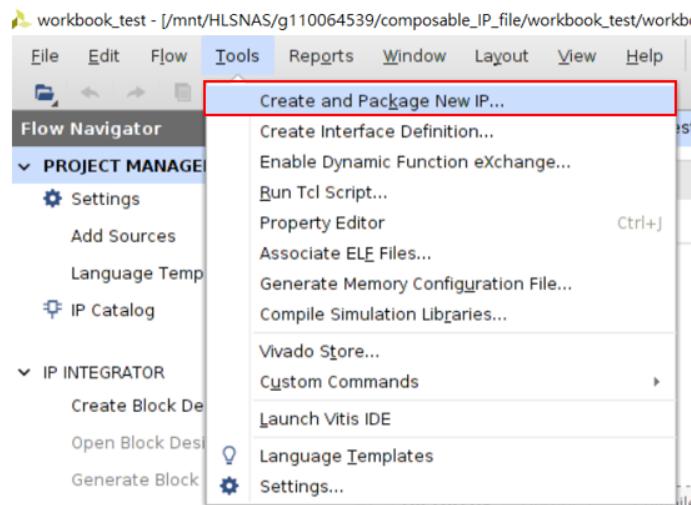
其中state=0是在parameter stream, 因此RTL不做事, 當s_axis_reg[63:48] == 16'hffff時, 表示parameter stream傳完, 接下來進入state=1, 開始做這個IP的功能。

以下是設計16-QAM IP:

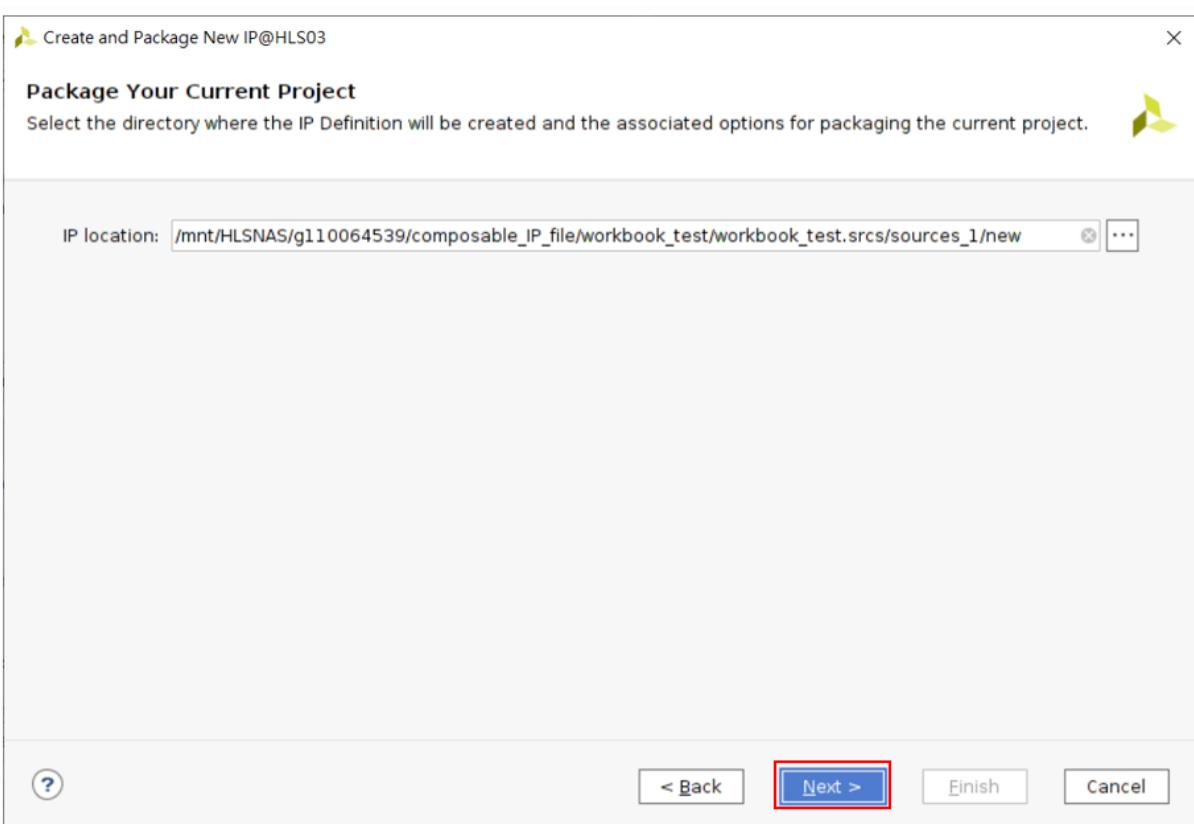
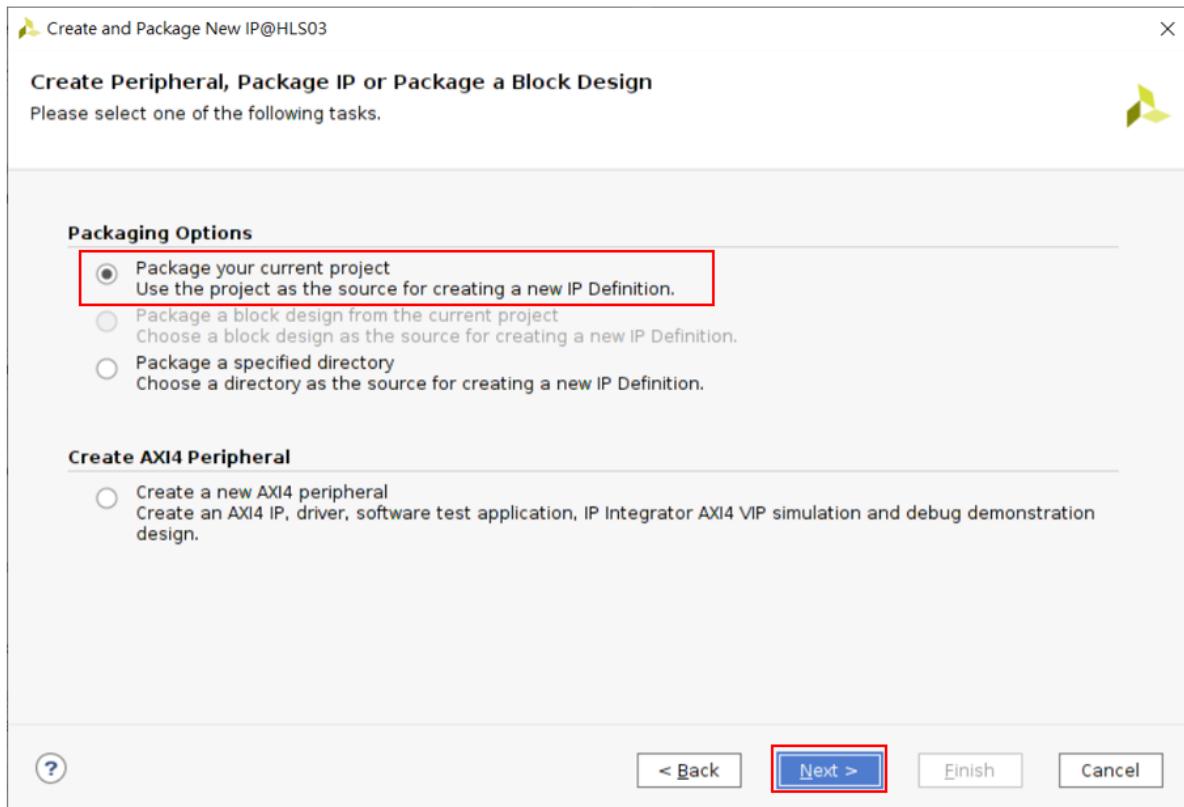
```
90      always @(*) begin
91          if(s_axis_valid_reg) begin
92              case(cs)
93                  1'd0: begin
94                      n_m_axis_data = s_axis_reg;
95                      if(s_axis_reg[63:48] == 16'hffff) begin
96                          ns = 1'd1;
97                      end
98                      else begin
99                          ns = 1'd0;
100                     end
101                 end
102             default: begin
103                 ns = cs;
104                 case(s_axis_reg[1:0])
105                     2'd0: begin //0
106                         n_m_axis_data[31:0] = 32'b00000000000011111010000000000000000; // -7'd3;
107                     end
108                     2'd1: begin //1
109                         n_m_axis_data[31:0] = 32'b000000000000111111100000000000000000; // -7'd1;
110                     end
111                     2'd2: begin //2
112                         n_m_axis_data[31:0] = 32'b000000000000000011000000000000000000; // 7'd3;
113                     end
114                     default: begin //3
115                         n_m_axis_data[31:0] = 32'b000000000000000000001000000000000000000; // 7'd1;
116                     end
117                 endcase
118                 case(s_axis_reg[3:2])
119                     2'd0: begin //0
120                         n_m_axis_data[63:32] = 32'b000000000000111110100000000000000; // -7'd3;
121                     end
122                     2'd1: begin //1
123                         n_m_axis_data[63:32] = 32'b000000000000111111100000000000000; // -7'd1;
124                     end
125                     2'd2: begin //2
126                         n_m_axis_data[63:32] = 32'b000000000000000011000000000000000; // 7'd3;
127                     end
128                     default: begin //3
129                         n_m_axis_data[63:32] = 32'b000000000000000000001000000000000000000; // 7'd1;
130                     end
131                 endcase
132             end
133         end
134     endcase
135     else begin
136         n_m_axis_data = m_axis_data;
137         ns = cs;
138     end
139 end
140 end
141
142 endmodule
```

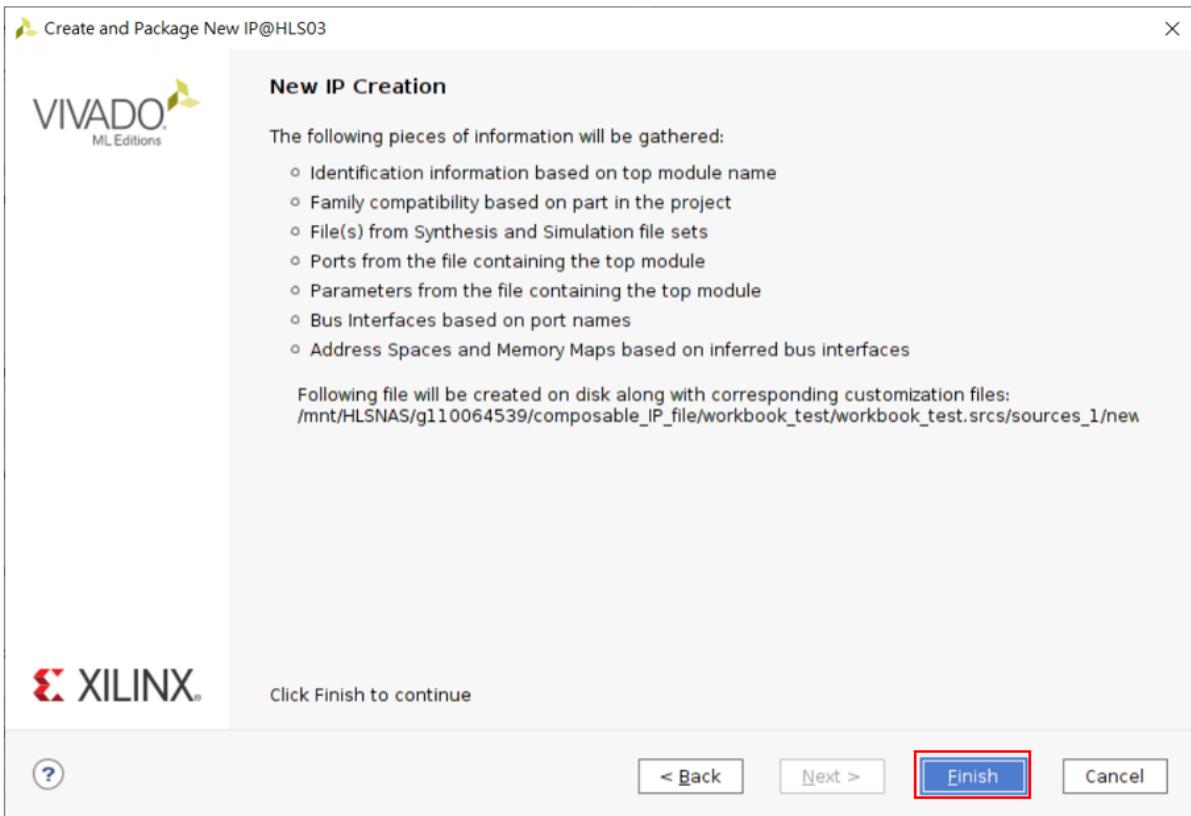
Package IP

驗證完RTL功能後，將RTL包成IP，首先在Tools選單中點選Create and Package New IP...



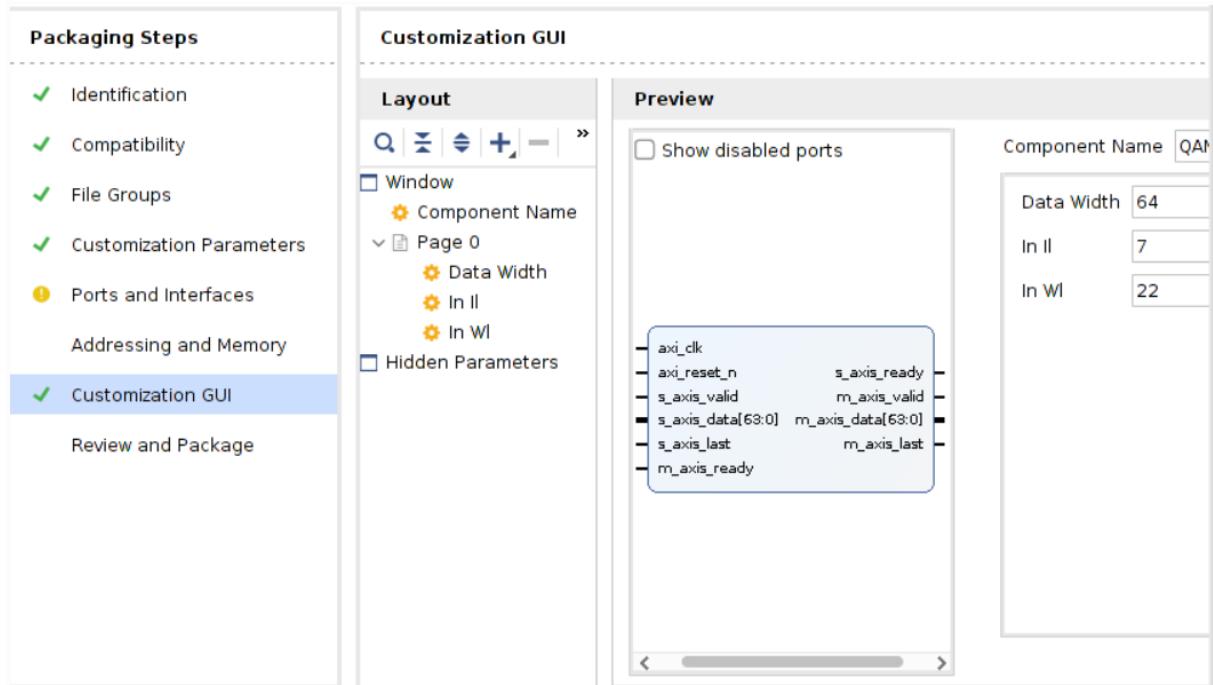
接下來按Next >, 會到下圖, 選擇Package your current project, 按Next >





會生成IP-XACT -> component.xml檔

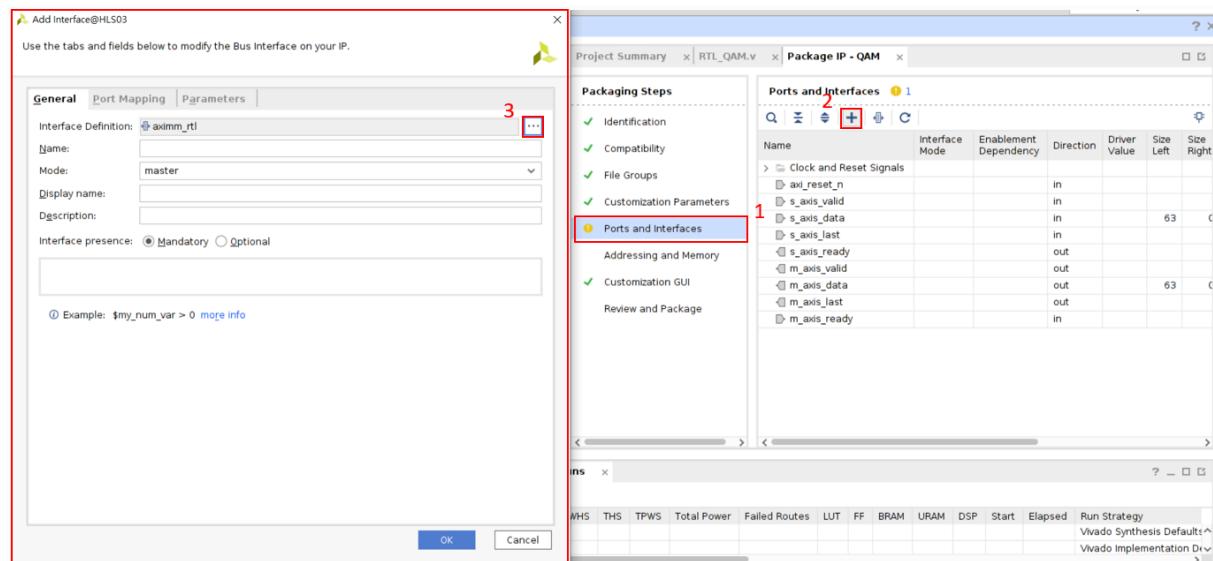
可以在Customization GUI中看到包出來的IP的ports:



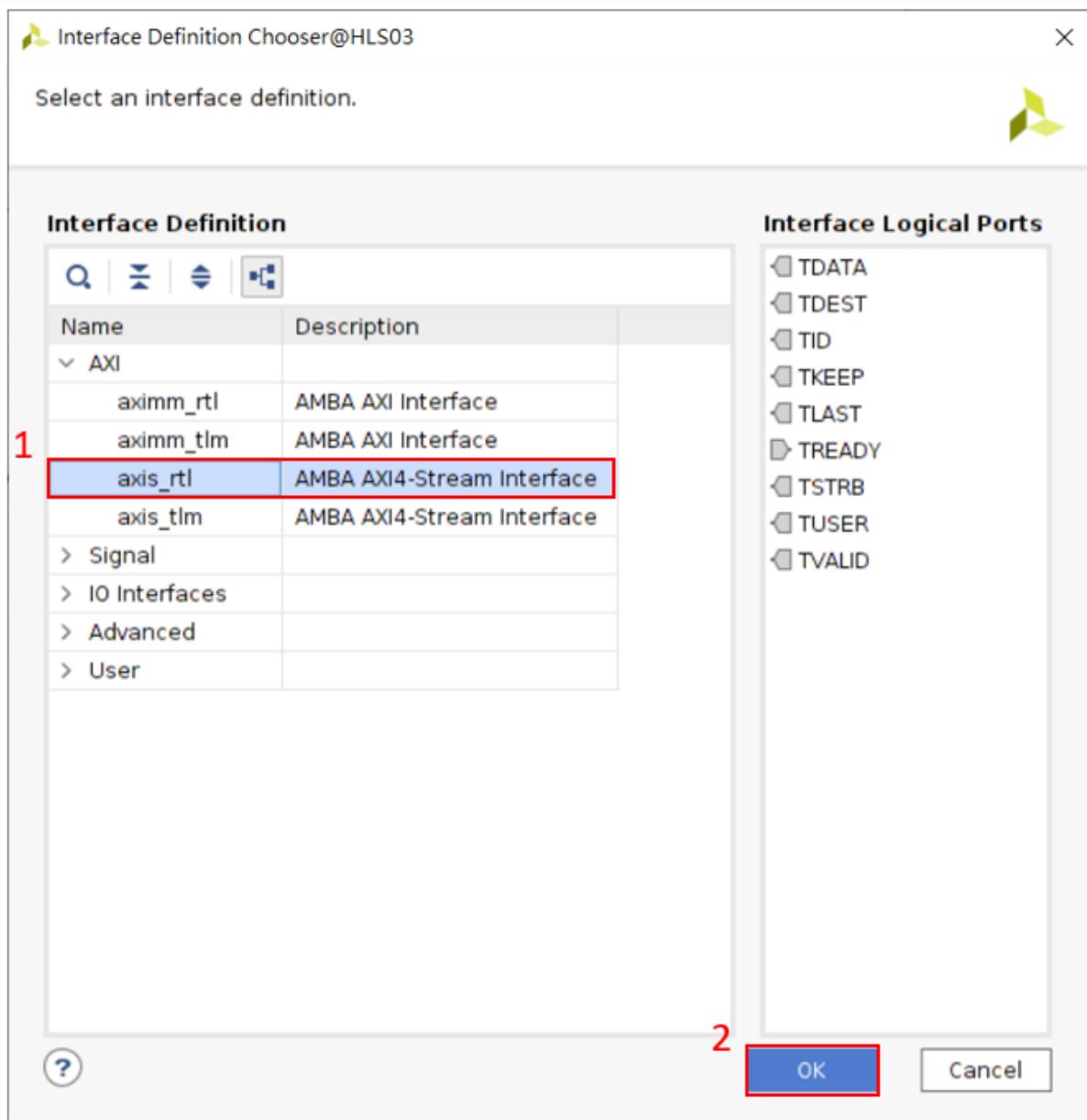
接下來我們將AXI4-stream的ports包起來:

點選Ports and Interfaces，點 ，會跳出Add interface視窗

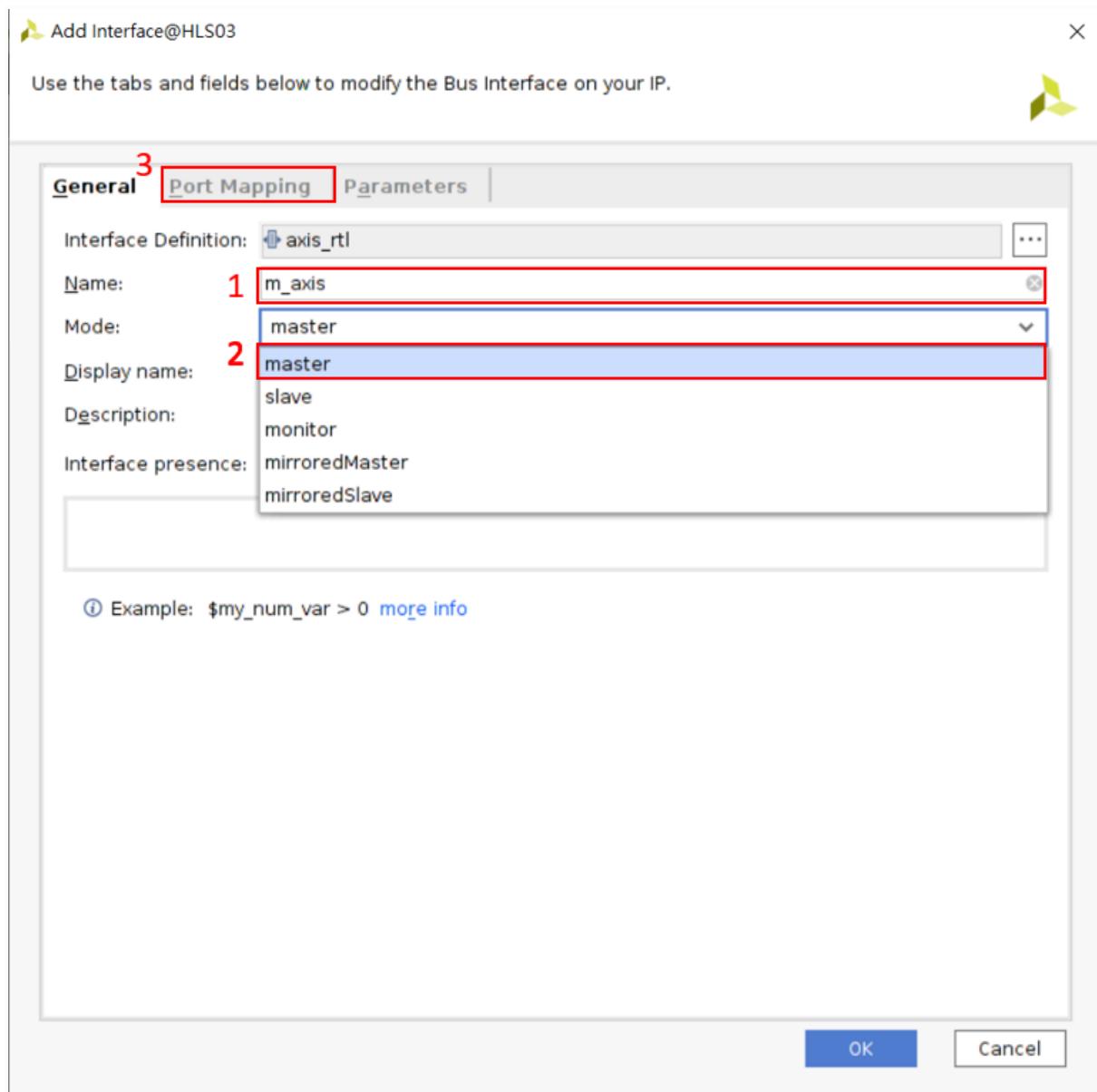
點選Interface Definition後面的 。



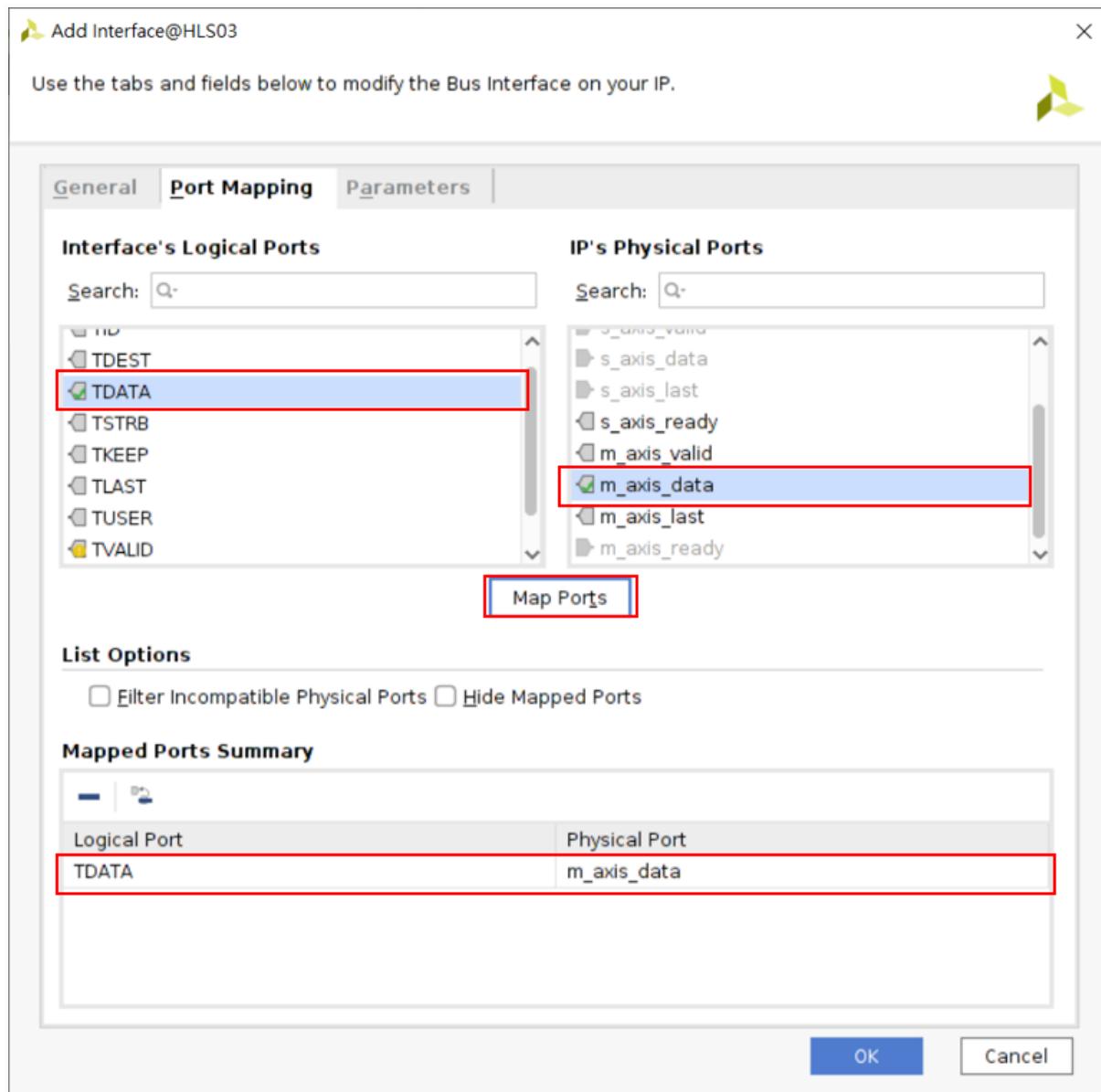
將interface改成axi4-stream的型態



先設定master stream, 將Name設定為"m_axis"(可以自己定義), Mode需要選"master", 接下來點Port Mapping將master stream的ports包在m_axis裡。



將Interface's Logical Ports與對應的IP's Physical Ports用map起來
點Interface's Logical Ports的"TDATA", 並點選IP's Physical Ports的"m_axis_data
再點選"Map Ports", 將這兩個map起來, 會在下面的Mapped Ports Summary看到結果。



將 TDATA、TVALID、TREADY、TLAST 都 mapping 好，如下圖，完成後按 OK，會看到 m_axis 將四個 ports 包起來。

 Add Interface@HLS03 X

Use the tabs and fields below to modify the Bus Interface on your IP.



General Port Mapping Parameters

Interface's Logical Ports

Search:

- TDATA
- TSTRB
- TKEEP
- TLAST
- TUSER
- TVALID
- TREADY

IP's Physical Ports

Search:

 s_axis_data

 s_axis_last

 s_axis_ready

m_axis_valid

m_axis_data

m_axis_last

 m_axis_ready

Map Ports

List Options

Filter Incompatible Physical Ports Hide Mapped Ports

Mapped Ports Summary

Logical Port	Physical Port
TDATA	m_axis_data
TVALID	m_axis_valid
TREADY	m_axis_ready
TLAST	m_axis_last

OK **Cancel**

Project Summary | RTL_QAM.v | Package IP - QAM

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- Ports and Interfaces
- Addressing and Memory
- ✓ Customization GUI
- Review and Package

Ports and Interfaces 2

Name	Interface Mode	Enablement Dependency	Direction	Driver Value	Size Left	Size Right
↳ m_axis	master					
m_axis_data			out		63	0
m_axis_valid			out			
m_axis_last			out			
m_axis_ready			in			
↳ Clock and Reset Signals						
axi_reset_n			in			
s_axis_valid			in			
s_axis_data			in		63	0
s_axis_last			in			
s_axis_ready			out			

接下來用相同的方式將s_axis_data、s_axis_last、s_axis_valid、s_axis_ready包起來在s_axis裡，如下圖：

Project Summary | RTL_QAM.v | Package IP - QAM

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- Ports and Interfaces
- Addressing and Memory
- ✓ Customization GUI
- Review and Package

Ports and Interfaces 3

Name	Interface Mode	Enablement Dependency	Direction	Driver Value	Size Left	Size Right
↳ m_axis	master					
m_axis_data			out		63	0
m_axis_valid			out			
m_axis_last			out			
m_axis_ready			in			
↳ s_axis	slave					
s_axis_data			in		63	0
s_axis_last			in			
s_axis_valid			in			
s_axis_ready			out			
↳ Clock and Reset Signals						
axi_clk	slave					
axi_reset_n			in			

設定clk，對axi_clk左鍵雙擊，會跳出Edit Interface, "Parameters"選單，選"ASSOCIATED_BUSIF"，按 ➔

2

3 ASSOCIATED_BUSIF

4

1

會看到剛選的ASSOCIATED_BUSIF跑到右邊，在Value那欄中填寫剛剛包好的ports name，因此填寫"m_axis:s_axis"，完成後按"OK"

1

2

3 ASSOCIATED_BUSIF

4

Value: m_axis:s_axis

OK

完成後可以看Customization GUI圖：

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- Addressing and Memory
- ✓ Customization GUI

Review and Package

Customization GUI

Layout

Window

- Component Name
- Page 0
 - Data Width
 - In IL
 - In WL
- Hidden Parameters

Preview

Show disabled ports

Block diagram preview showing port connections:

```

  graph TD
    subgraph s_axis [s_axis]
      s_axis_data[63:0]
      s_axis_last
      s_axis_valid
      s_axis_ready
    end
    subgraph m_axis [m_axis]
      m_axis_data[63:0]
      m_axis_valid
      m_axis_last
      m_axis_ready
    end
    axi_clk
    axi_reset_n
    s_axis_data --> m_axis_data
    s_axis_last --> m_axis_valid
    s_axis_valid --> m_axis_last
    s_axis_ready --> m_axis_ready
    axi_clk --> m_axis
    axi_reset_n --> m_axis
  
```

選到Review and Package, 將”Delete project after packaging”取消, 按”Apply”, 最後按”Package IP”, 完成RTL package成IP。

Project Summary | **RTL_QAM.v** | **Package IP - QAM**

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- Addressing and Memory
- ✓ Customization GUI

Review and Package

Summary

Display name: QAM_v1_0
Description: QAM_v1_0
Root directory: /mnt/HLSNAS/g110064539/compos

After Packaging

An archive will not be generated. Use the setting:
IP will be made available in the catalog using the
mnt/HLSNAS/g110064539/composable_IP_file/wo
[Edit packaging settings](#)

Package IP

Settings@HLS03

Project Settings

- General
- Simulation
- Elaboration
- Synthesis
- Implementation
- Bitstream
- IP
- Repository
- Packager**

Tool Settings

- Project
- IP Defaults
- Vivado Store
- Source File
- Display
- WebTalk
- Help
- Text Editor
- 3rd Party Simulators
- Colors
- Selection Rules
- Shortcuts
- Strategies
- Remote Hosts
- Window Behavior

IP > Packager

Specify settings related to IP Packager.

Default Values

The following values will be automatically applied after finishing the IP Packager Wizard.

Vendor:	HLSNIS
Library:	user
Category:	/UserIP
IP location:	./ip_repo

Automatic Behavior

After Packaging

- Create archive of IP
- Add IP to the IP Catalog of the current project
- Close IP Packager window
- Include Source project archive

Edit IP in IP Packager

Delete project after packaging

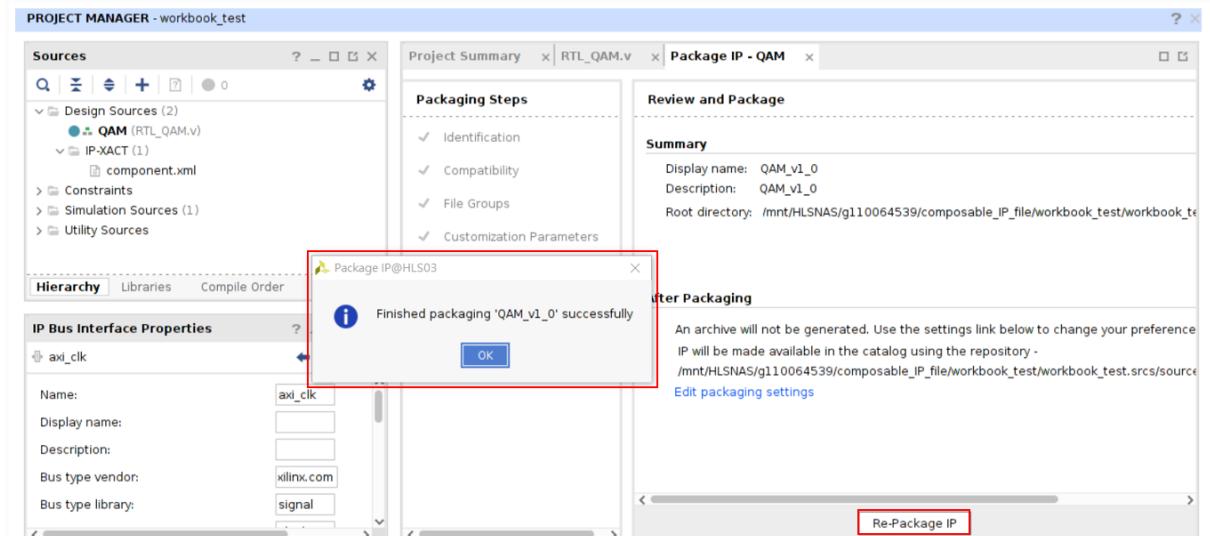
File Extensions to Filter on Add Directory

Create a list of file extensions that will be automatically filtered when adding a directory to a File Group.

+ ---

No content

OK Cancel Apply Restg...



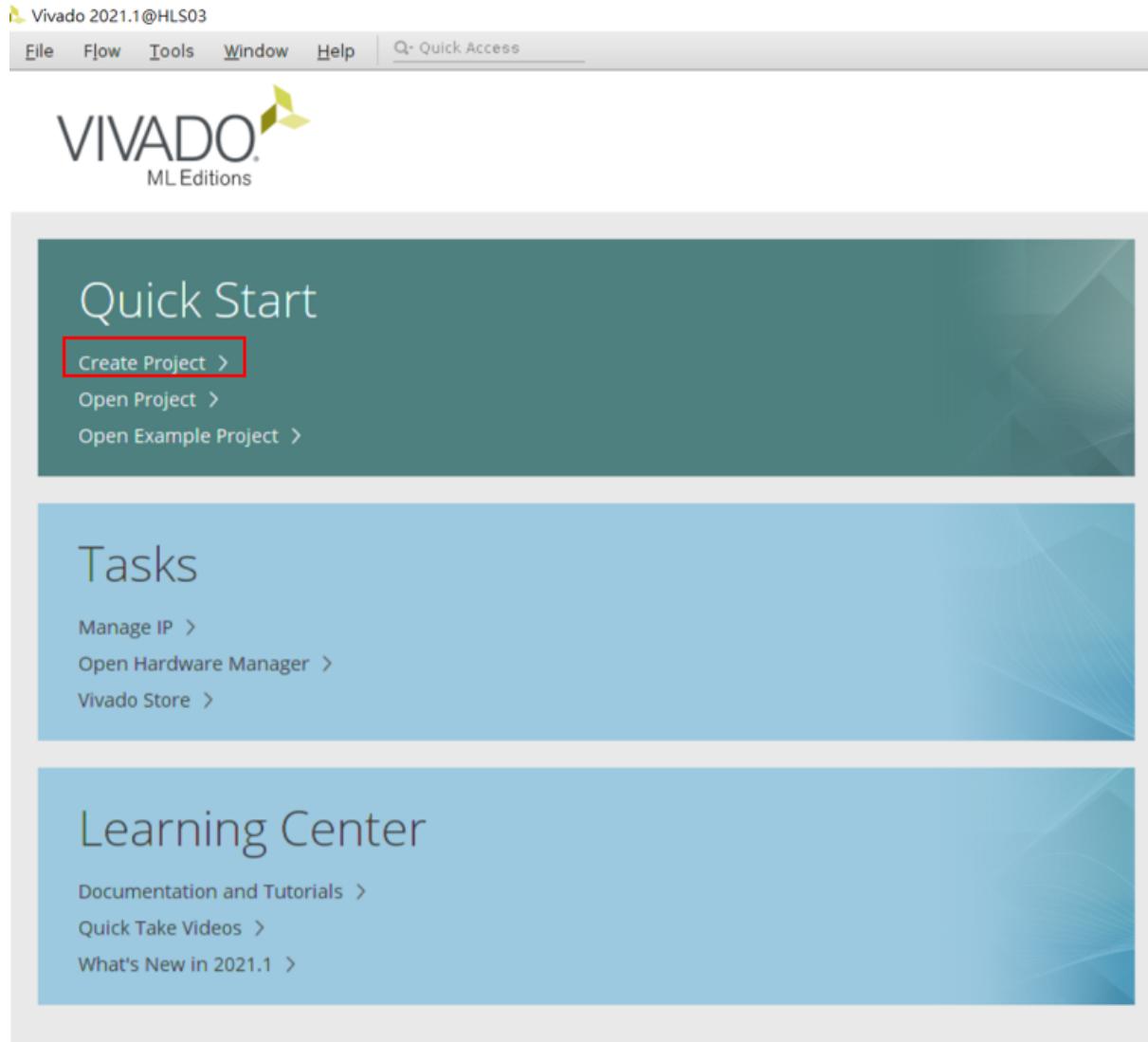
之後用vivado做Block design時，匯入IP只要選擇此project，就可以讀到IP了。

> Composable Pipeline Connection in Vivado

◦ Create vivado project

啟動vivado開發套件

開啟新的專案，設定好專案存放的路徑：



選擇RTL Project, 勾選Do not specify sources at this time, 接著按Next>:

Project Type

Specify the type of project to create.



- RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
 - Do not specify sources at this time
 - Project is an extensible Vitis platform
- Post-synthesis Project**
You will be able to add sources, view device resources, run design analysis, planning and implementation.
 - Do not specify sources at this time
- I/O Planning Project**
Do not specify design sources. You will be able to view part/package resources.
- Imported Project**
Create a Vivado project from a Synplify, XST or ISE Project File.
- Example Project**
Create a new Vivado project from a predefined template.



< Back

Next >

Finish

Cancel

選擇Borads, 並在下方選 Kria KV260 Vision AI Started Kit, 點選Next>:

Default Part

Choose a default Xilinx part or board for your project.



Parts **Boards**

To fetch the latest available boards from git repository, click on 'Refresh' button. [Dismiss](#)

[Reset All Filters](#)

Vendor: All

Name: All

Board Rev: Latest



Search: kv260 (1 match)

Display Name



Preview Status Vendor File Version Part

Kria KV260 Vision AI Starter Kit
Add Companion Card [Connections](#)

Installed xilinx.com 1.1

Som Vision Platform Boar

Refresh



< Back

Next >

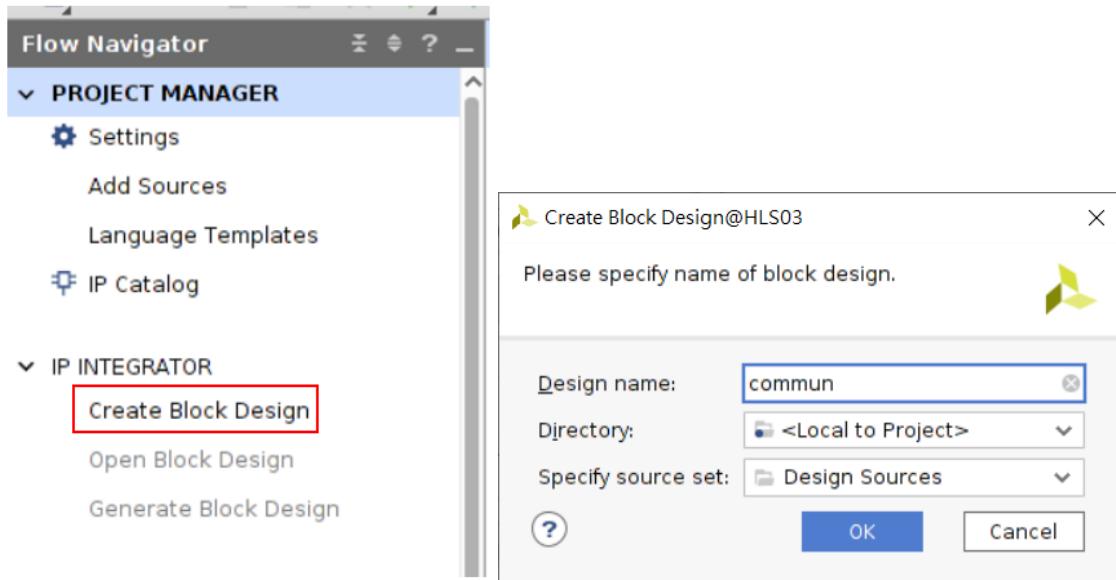
Finish

Cancel

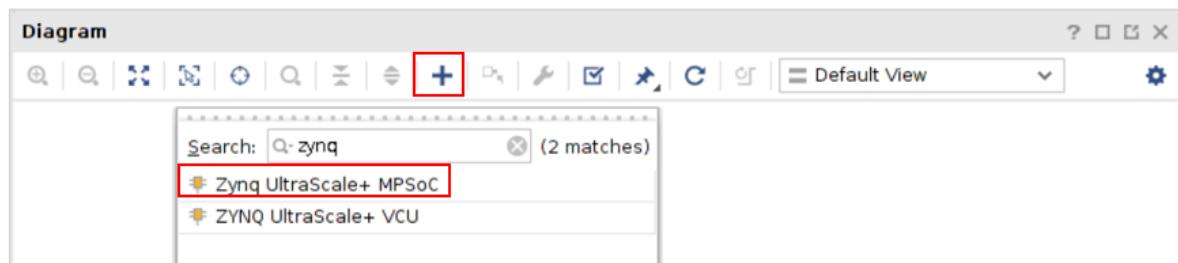
最後點選Finish, 完成Create Project。

◦ Design Block Design

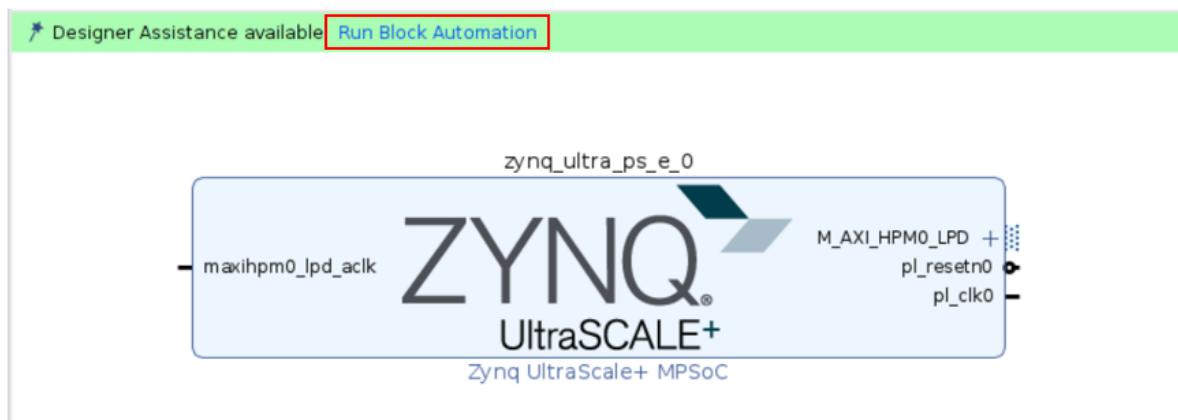
選擇Create Block Design, 設定block design名字

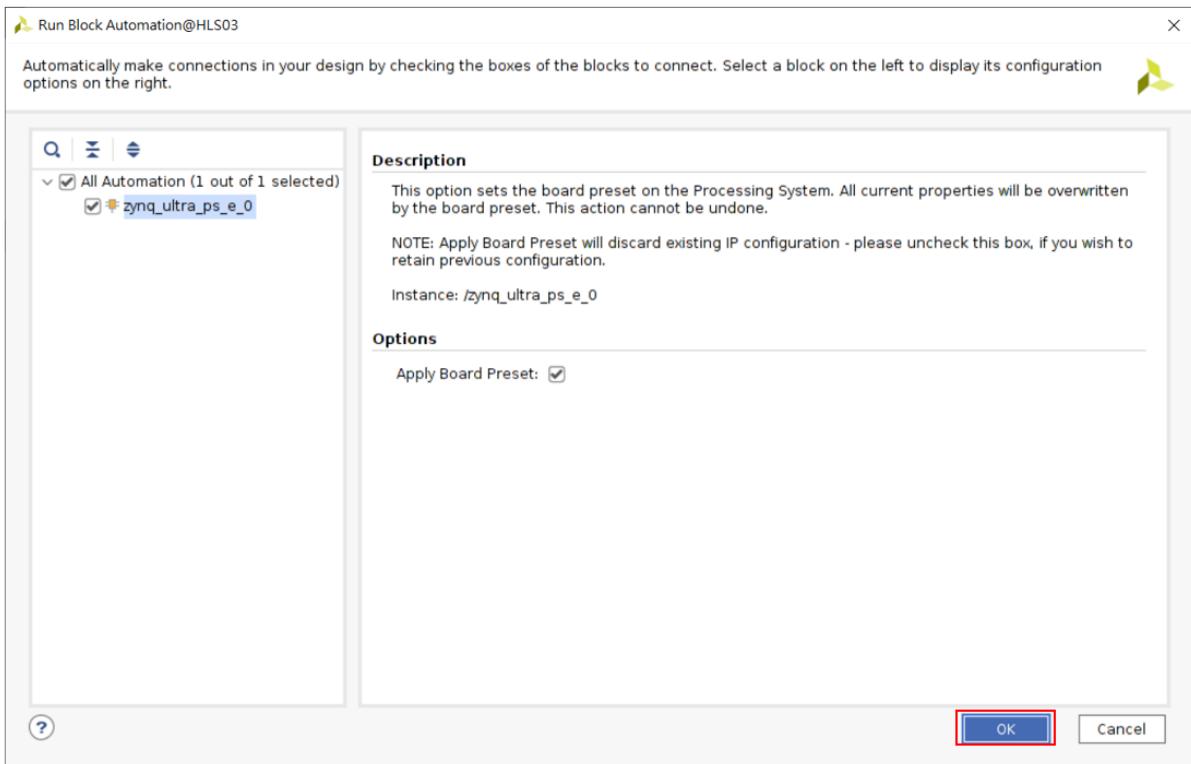


在Diagram的工具列中點選 來新增IP, 搜尋到Zynq UltraScale+ MPSoC並左鍵點選兩下

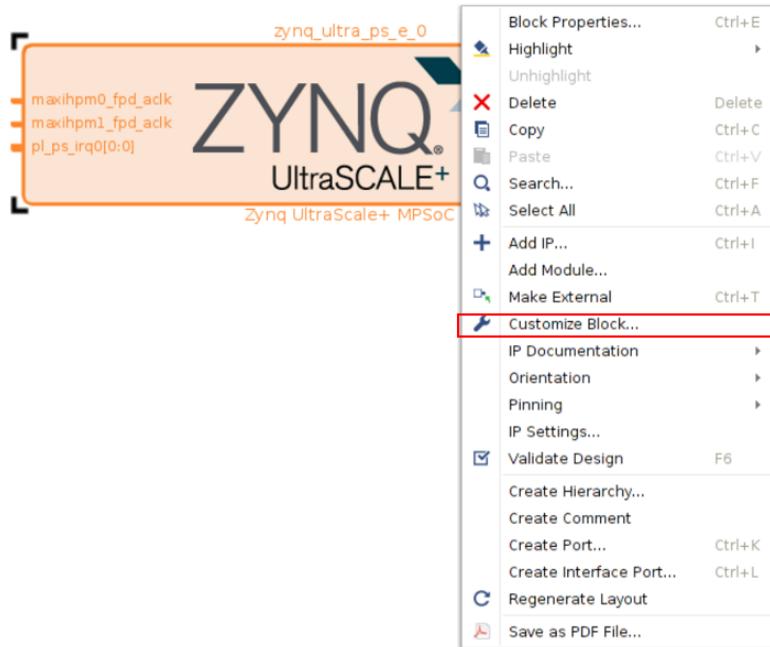


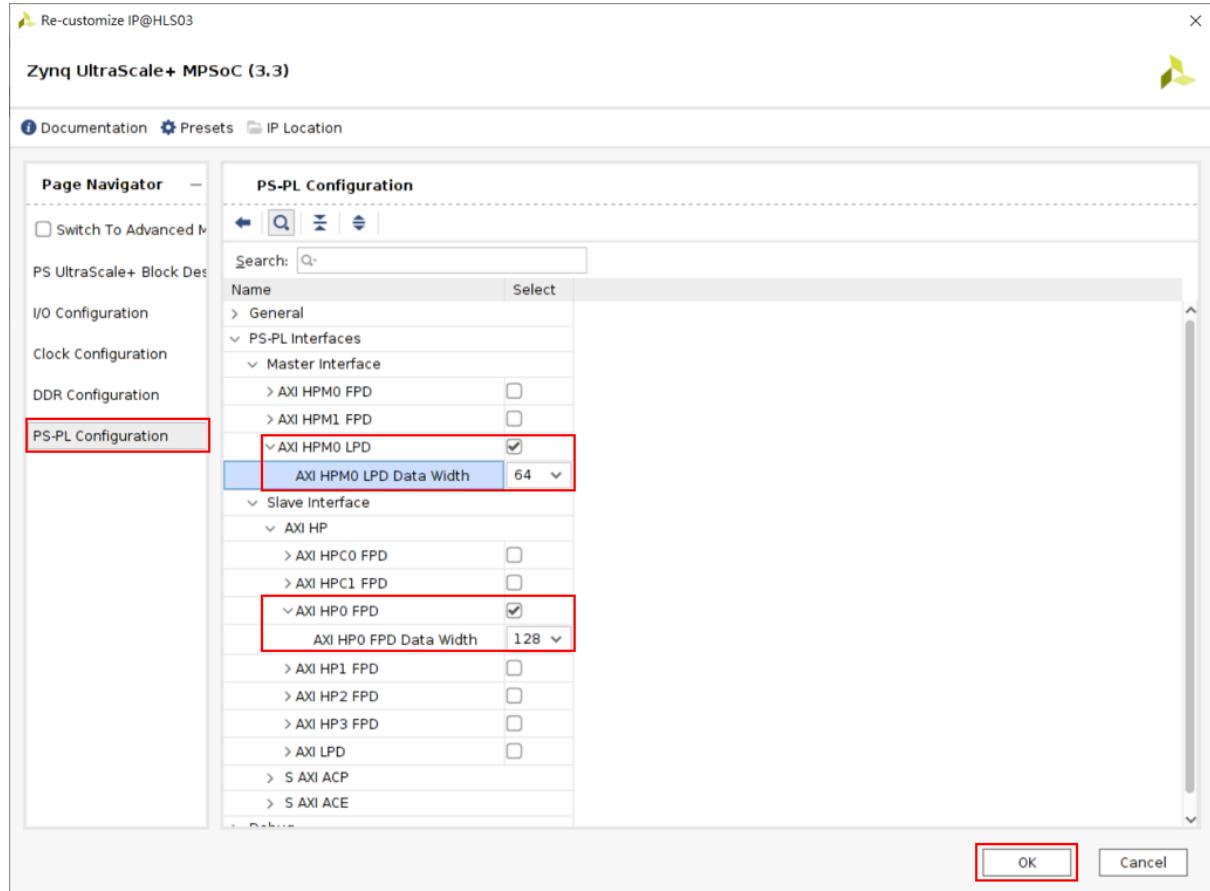
會跑出Zynq UltraScale+ MPSoC, 點選Run Block Automation, 跳出視窗按OK



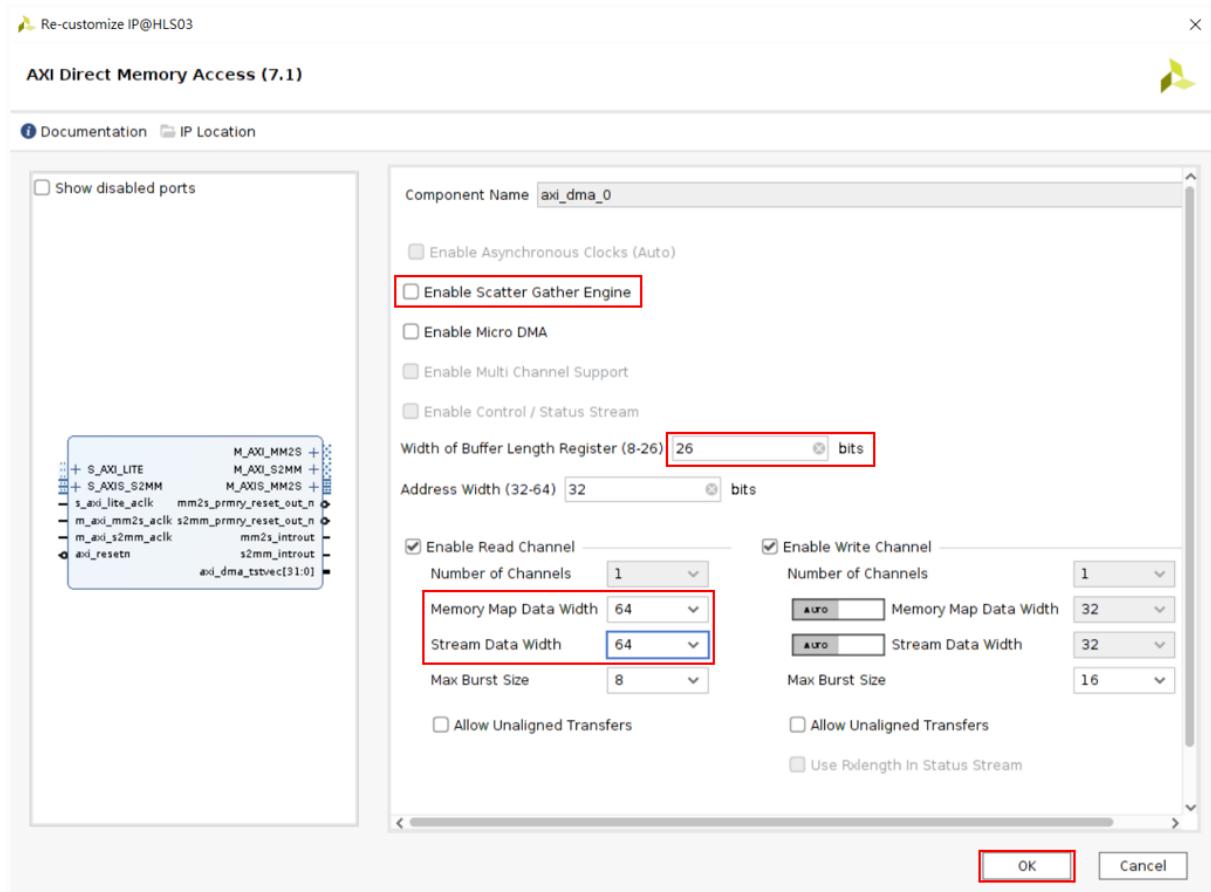


對Zynq UltraScale+ MPSoC按右鍵，選擇Customize Block...，在跳出視窗的左邊欄位選PS-PL Configuration，展開PS-PL interface，設定好Master interface及Slave interface如下圖，其中AXI HPM0 LPD Data Width要設定為64，因為我們設計中加了parameter stream後的規格是64 bits。





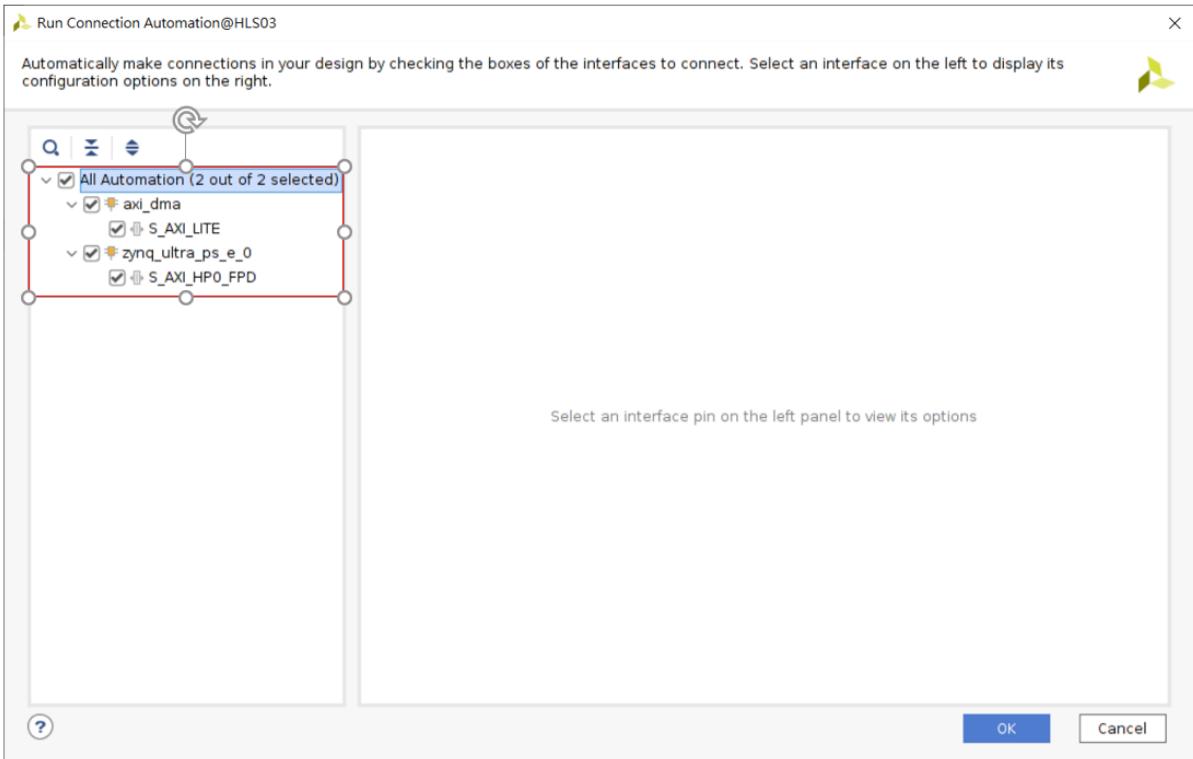
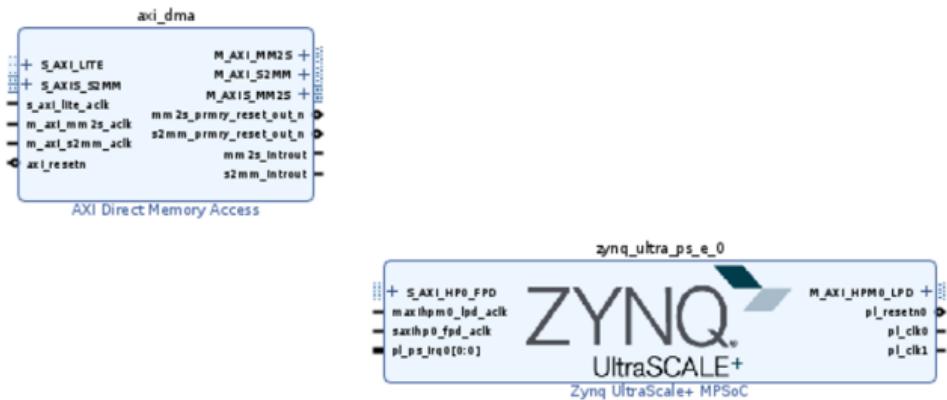
在Diagram的工具列中點選 來新增IP, 搜尋到AXI Direct Memory Access並左鍵點選兩下, 再對生成出來的AXI Direct Memory Access左鍵點兩下進入設定, Enable Scatter Gather Engine取消打勾, Width of Buffer Length Register設為26, 在Read channel中把Memory data Width和Stream data Width都設為64 bits, 按OK, 最後將IP Block名字改為axi_dma。



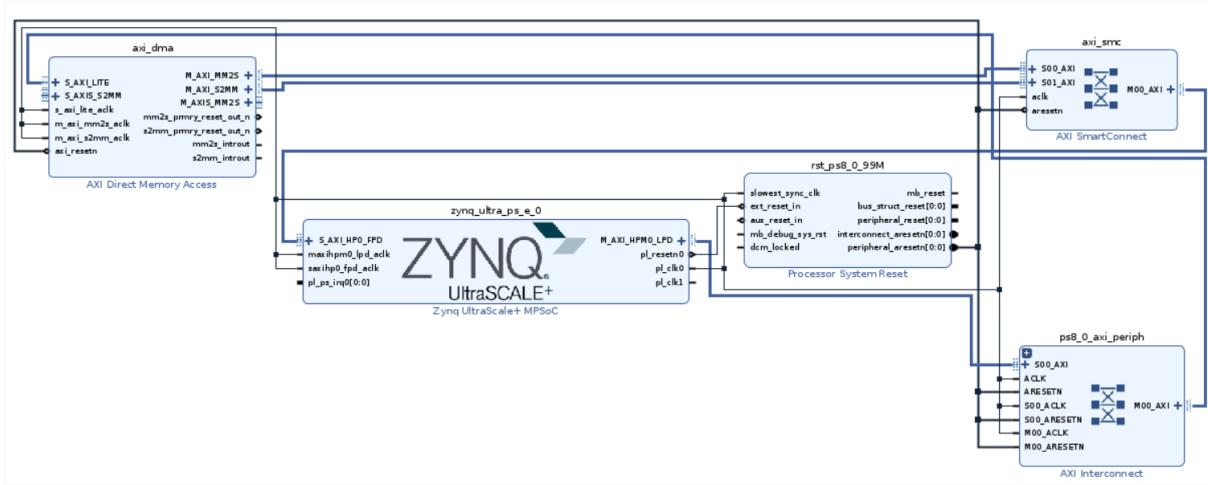
點選Run Block Automation, 跳出視窗勾選All automation按OK, 重複做兩次。

* Designer Assistance available

[Run Connection Automation](#)

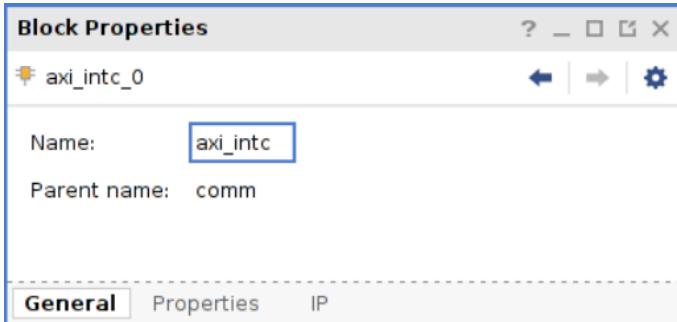


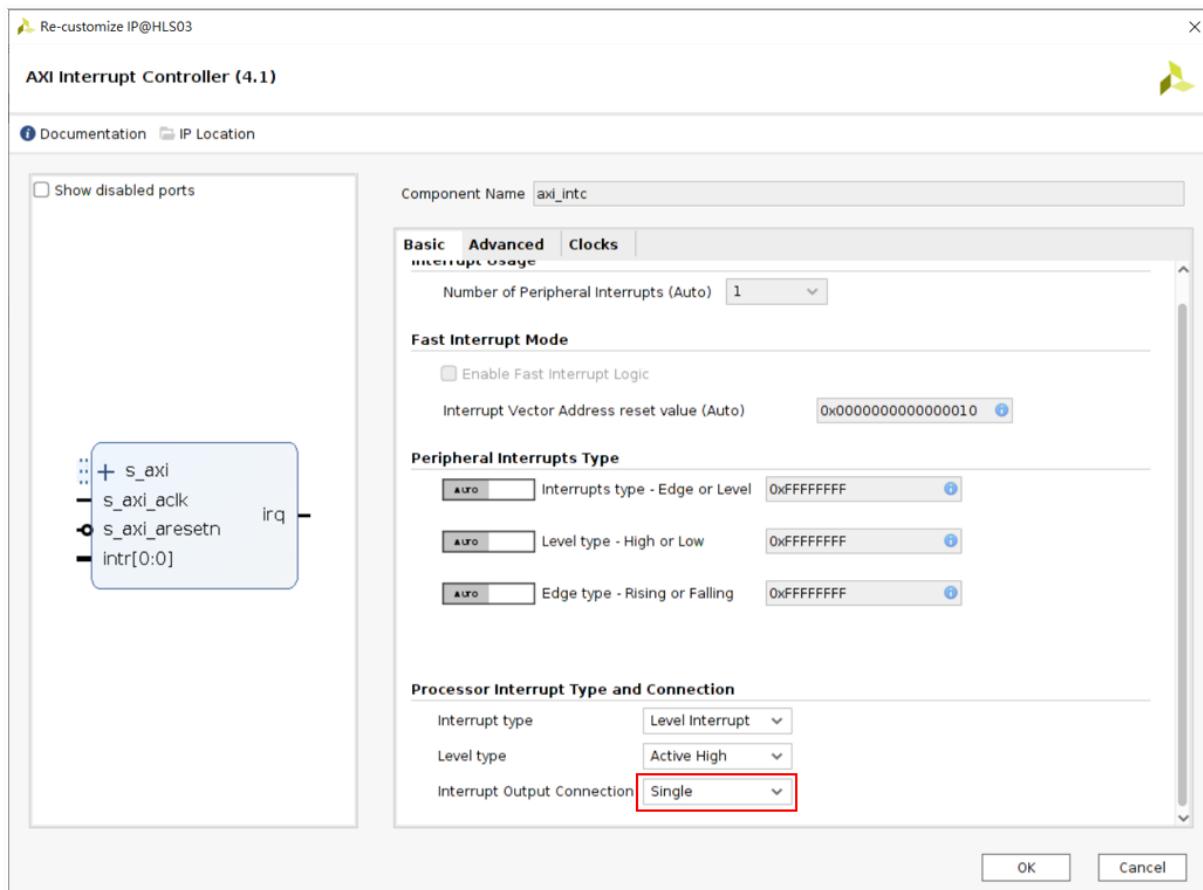
最後會會是下圖所示



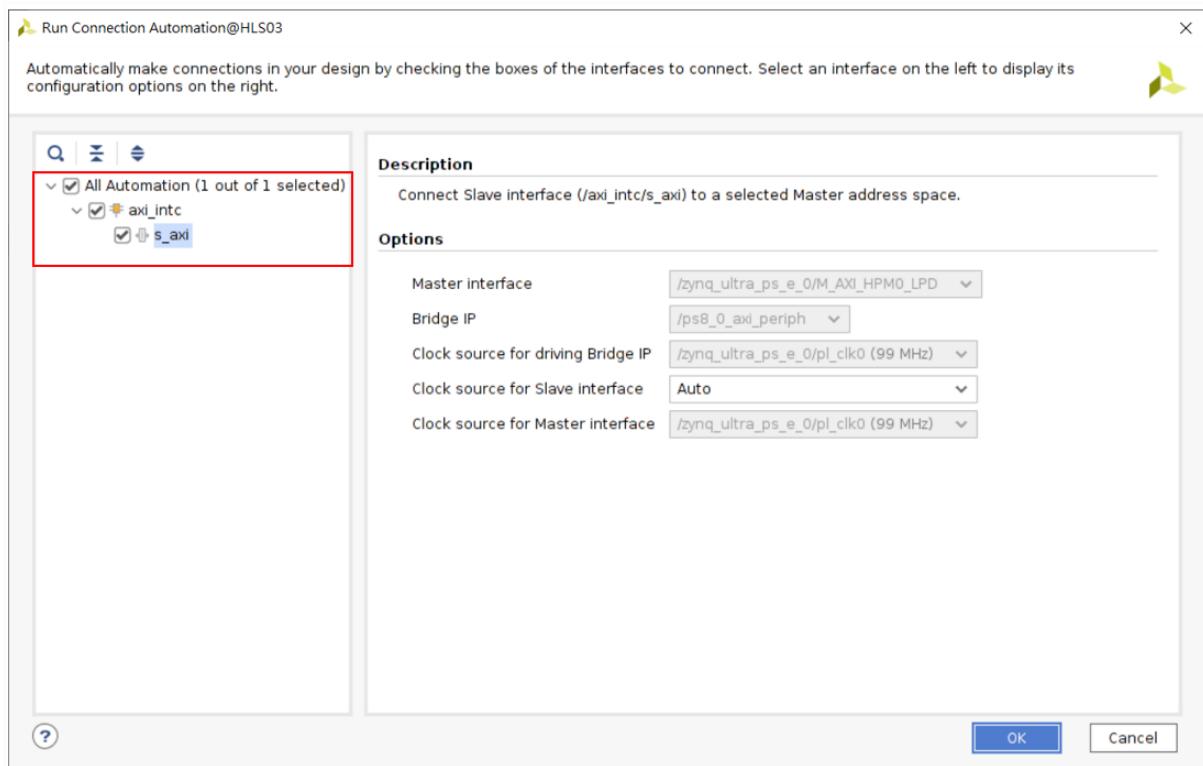
在Diagram的工具列中點選 來新增IP, 搜尋到AXI Interrupt Controller並左鍵點選兩下將IP block加入。

將IP block名字改成axi_intc, 在對IP block左鍵點兩下進入設定, 把Interrupt Output Connection選Single, 按OK。



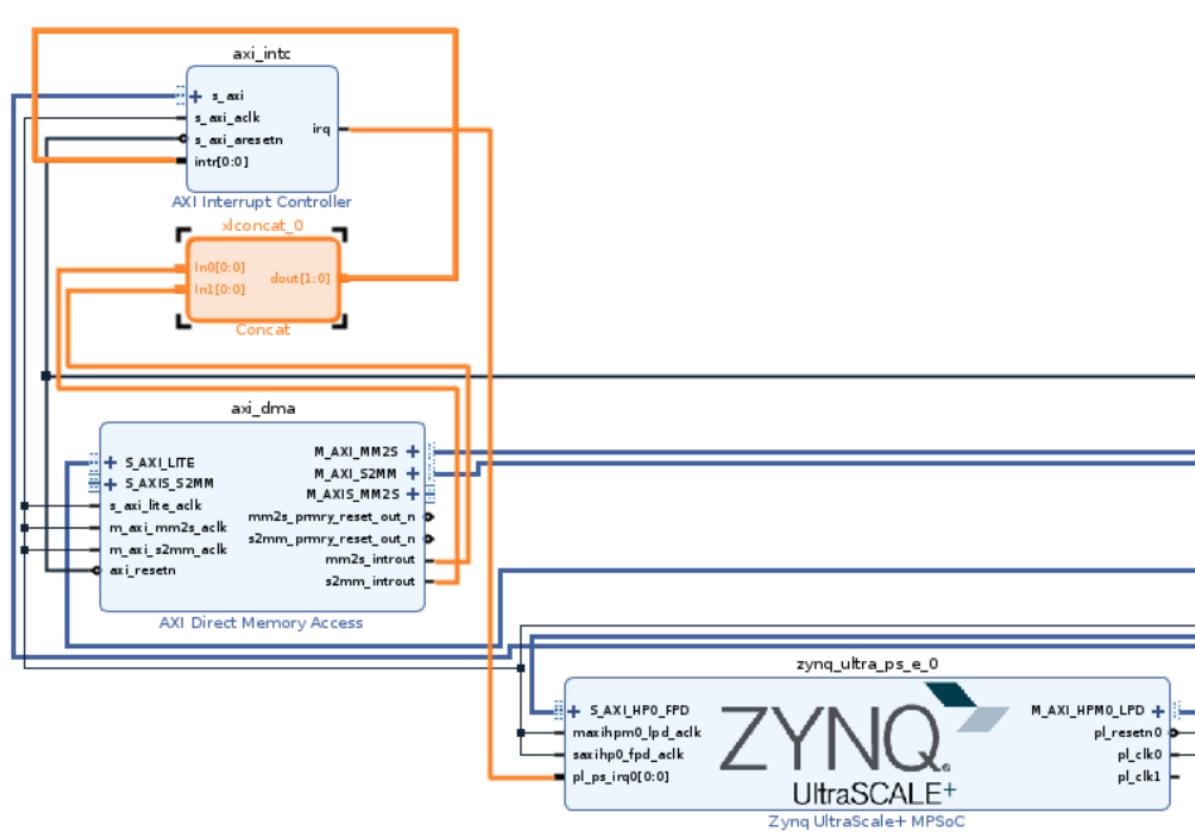


點選Run Block Automation, 跳出視窗勾選All automation按OK



手動將axi_intc的irq pin角與zynq_ultra_ps_e_0的pl_ps_irq0連接, 再新增Concat的IP

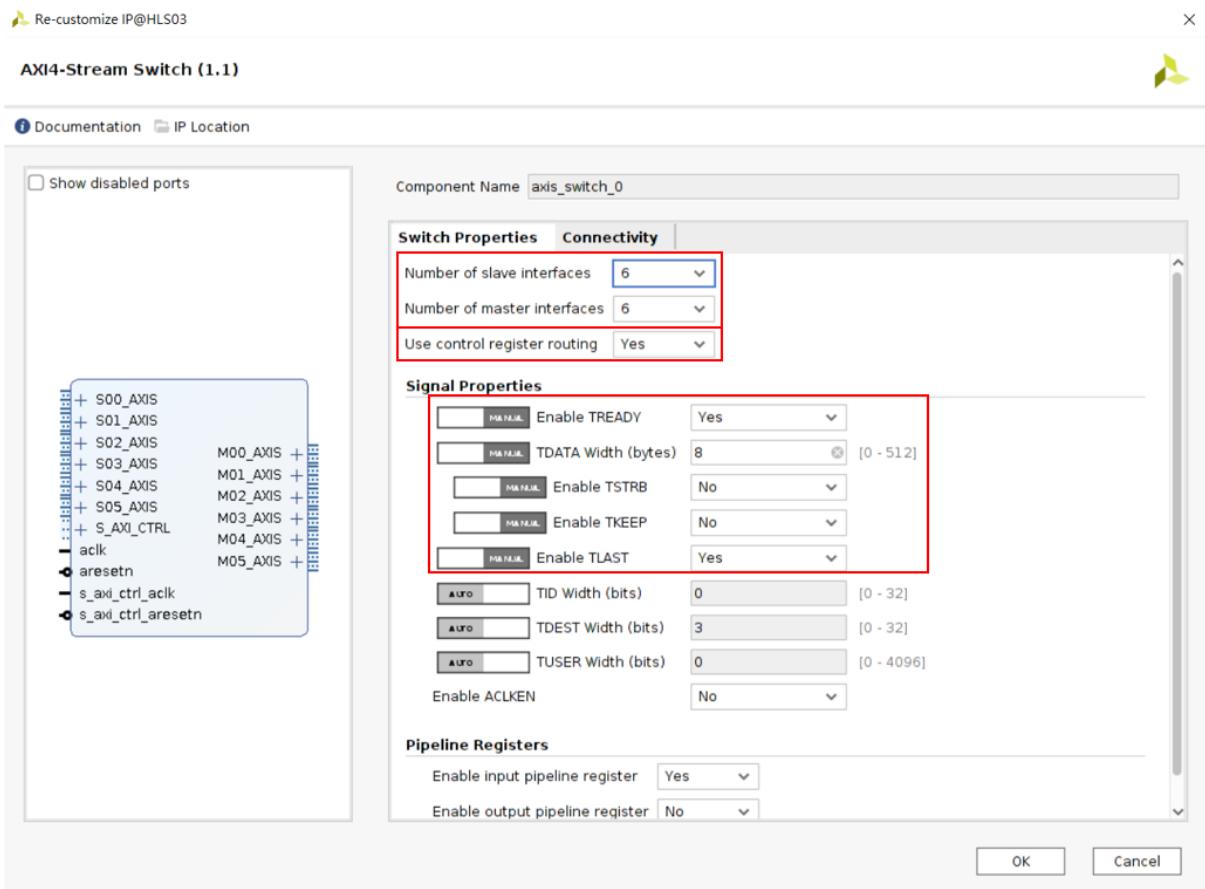
將axi_dma的mm2s_introut與xlconcat_0的input連接, axi_dma的s2mm_introut與xlconcat_0的另一個input連接, 與xlconcat_0的dout與axi_intc的intr連接。



◦ Make the overlay composable

在Diagram的工具列中點選 來新增IP, 搜尋到AXI4-Stream Switch並左鍵點選兩下將IP block加入, 對加入的IP block左鍵點兩下進入設定。

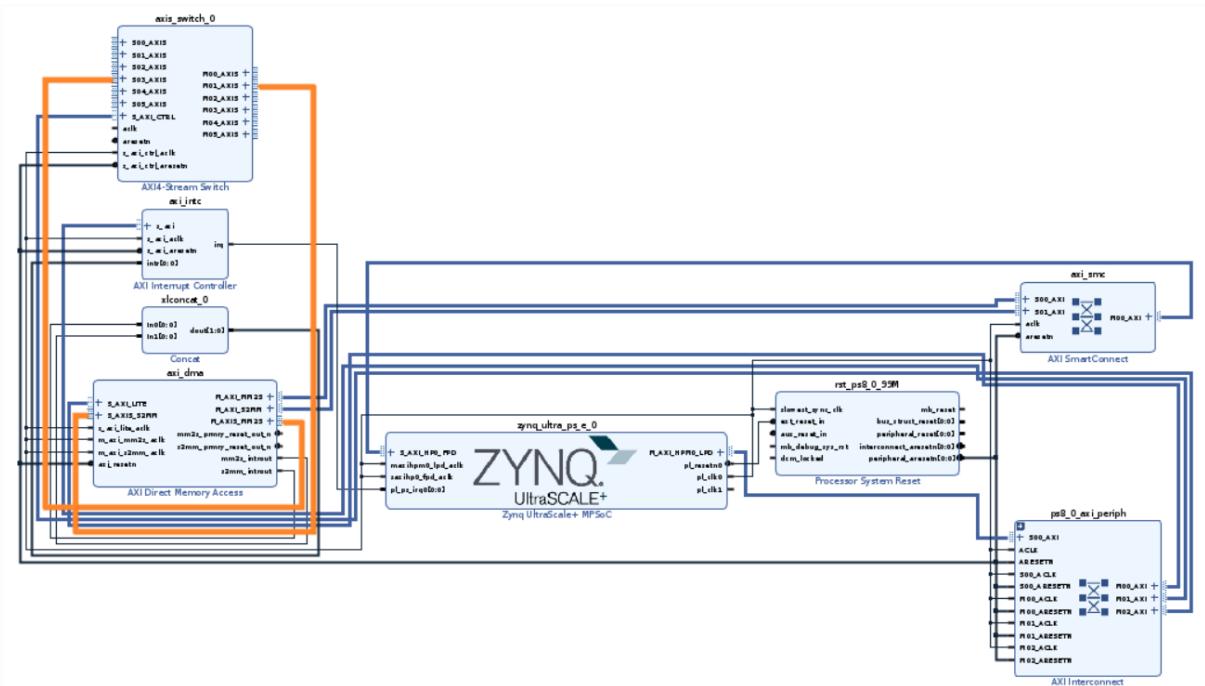
Number of slave interface及Number of master interface根據需要的控制的composable pipeline的個數決定, 這邊先設定為6, Use control register routing設Yes, TEADY、TSTRB、TKEEP、TLAST設定為下圖所示。由於我們訊號是64 bites, 所以TDATA設為8 bytes。按OK。

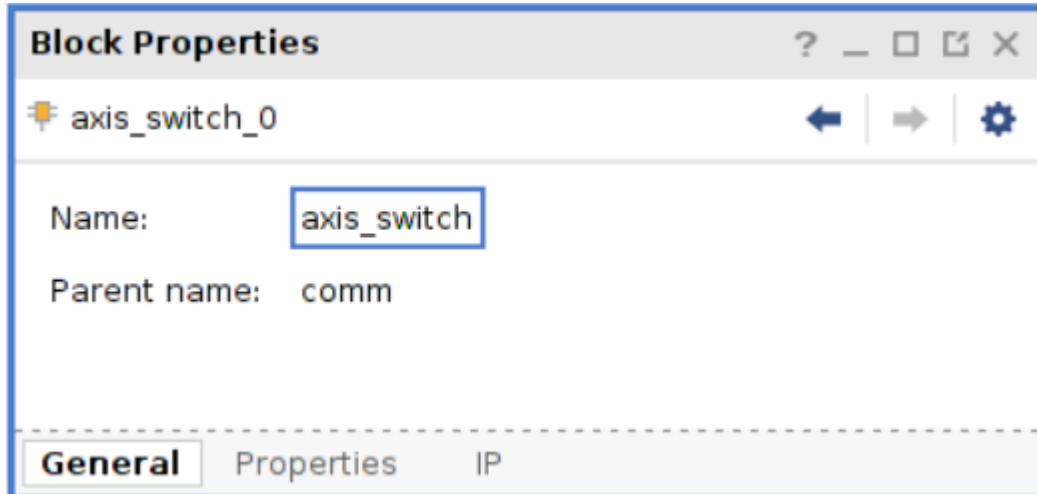


將axi_dma的M_AXIS_MM2S接到axis_switch_0的S03_AXIS, axis_switch_0的M01_AXIS接到axi_dma的S_AXIS_S2MM。

點選Run Block Automation, 跳出視窗勾選All automation按OK。

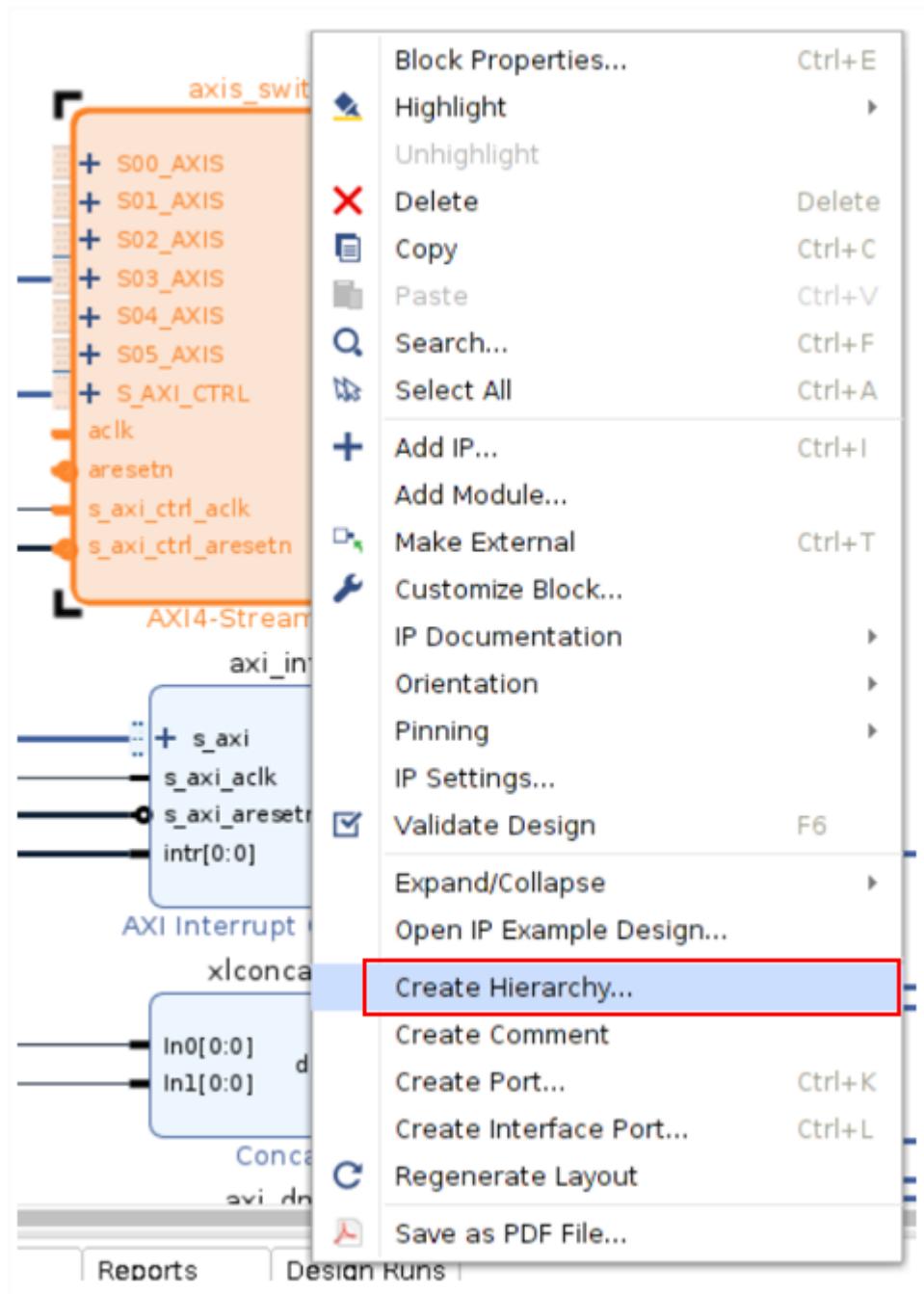
將axis_switch_0名字改為axis_switch

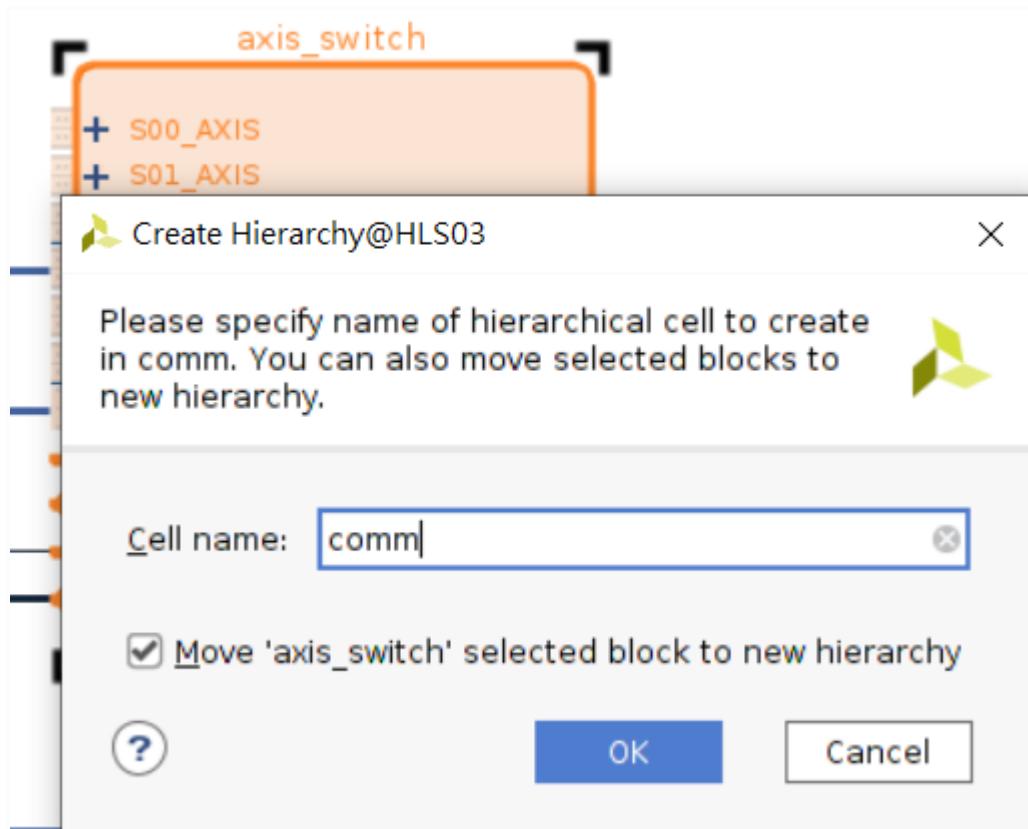




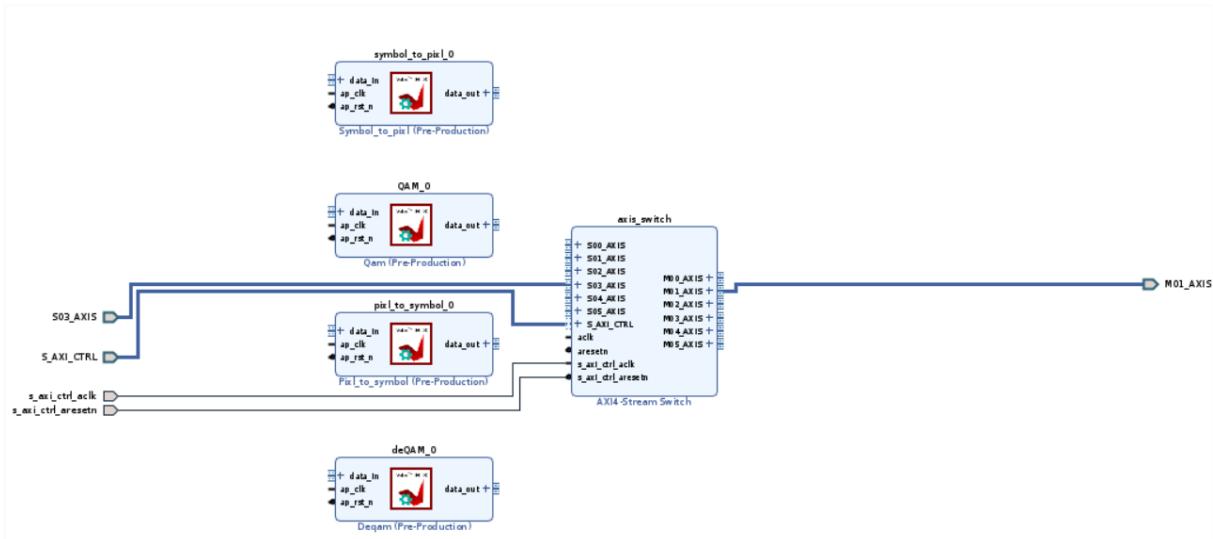
對axis_switch_0按右鍵, 點選Create Hierarchy...

Cell name設為comm, 按OK

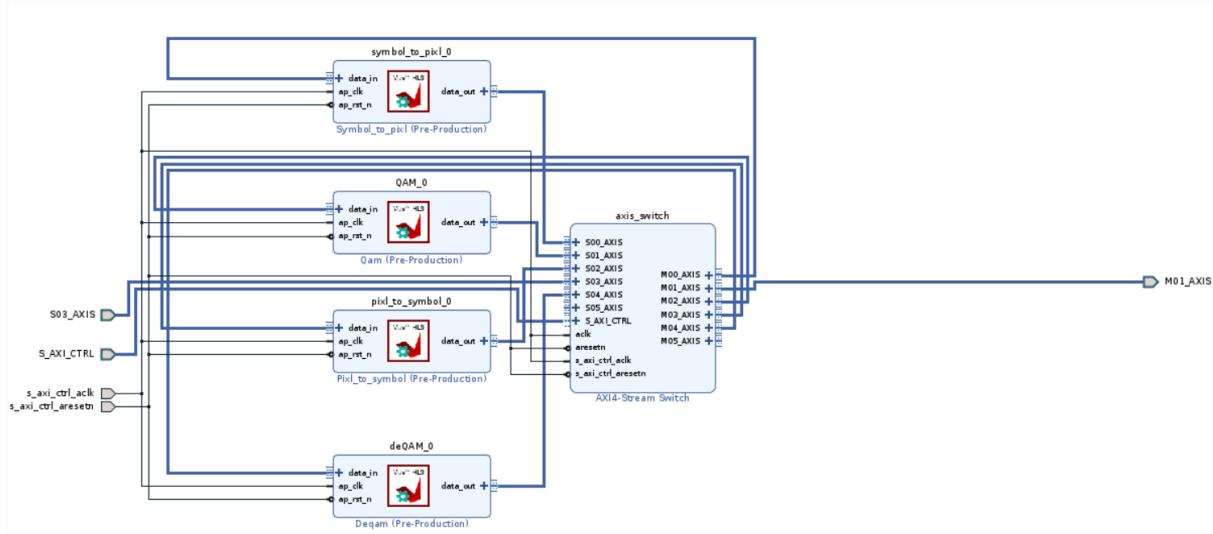




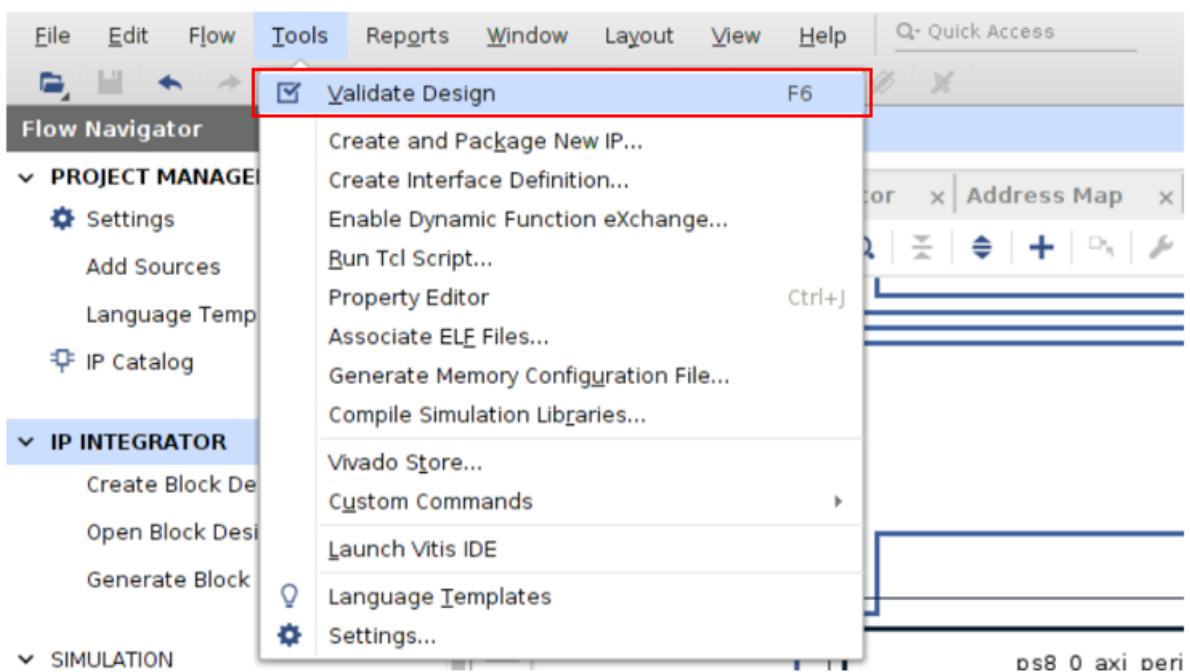
再對comm左鍵點兩下開啟，將想要加入的IP加進去



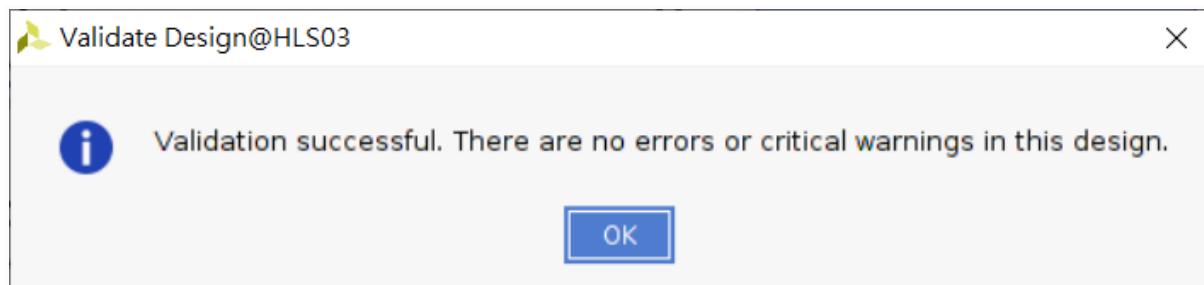
將想加入的IP output接到axi_switch的input, axi_switch的output接到IP 的input, 點選Run Block Automation, 跳出視窗勾選All automation按OK。



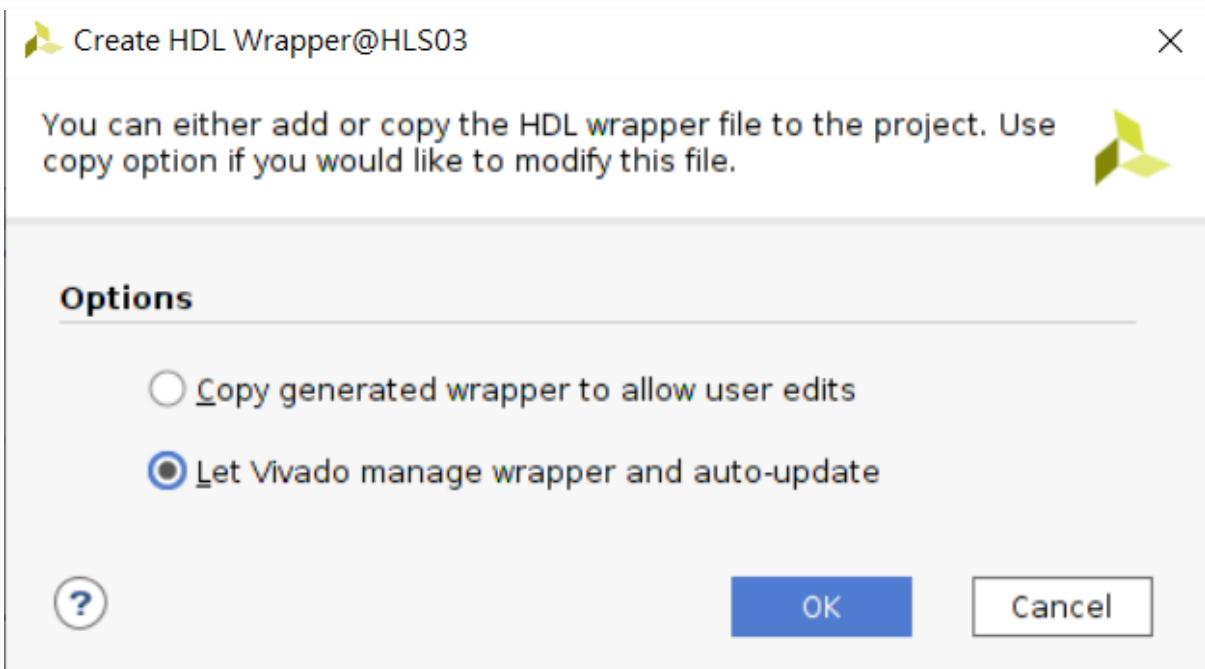
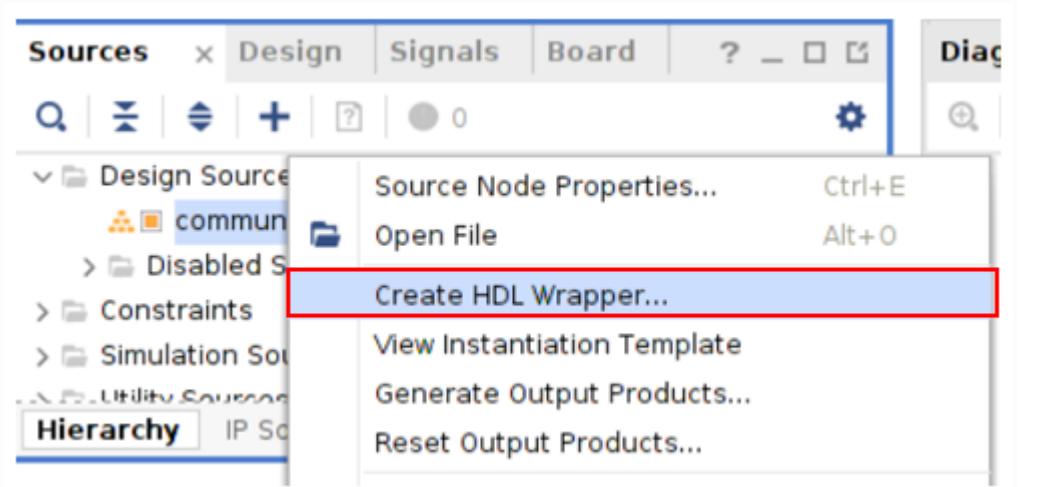
按CTRL+S儲存Block Design,
在Tools中點選Validate Design,



若跳出Validation successful, 表示沒有問題。



在Design sources的commun按右鍵選Create HDL Wrapper..., 跳出視窗按OK。



最後可以點選Generate Bitstream



◦ Jupyter Notebook Host Code

將commun.bat、commun.hwh、commun_path.json及communication.ipynb放到jupyter notebook

其中commun_path.json程式碼如下圖所示，其中"port" : 3、"port" : 1分別對應到axi_switch接到axi_dma的ports(S03_AXIS、M01_AXIS)。

```

1   {
2       "comm": {
3           "ps": {
4               "ci": {
5                   "port": 3,
6                   "Description": "Stream In"
7               },
8               "pi": {
9                   "port": 1,
10                  "Description": "Stream Out"
11             }
12         }
13     }
14 }
```

communication.ipynb程式碼為host code, 先將commun.bat檔Overlay進來, 並用cfilter物件等於comm的Hierarchy, 並print出Hierarchy中的IP。

```

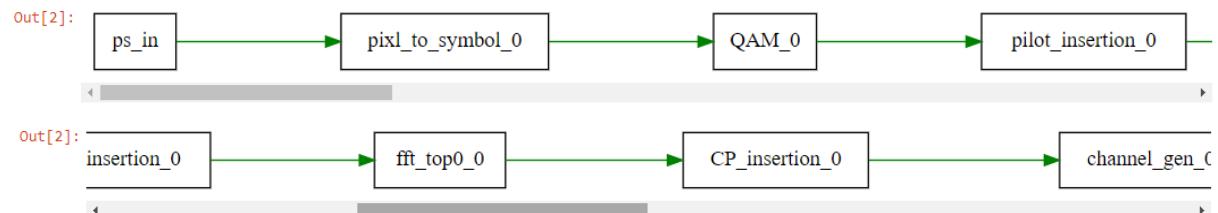
from pynq import Overlay, allocate
import matplotlib.pyplot as plt
import pynq_composable
import numpy as np
import math
from PIL import Image

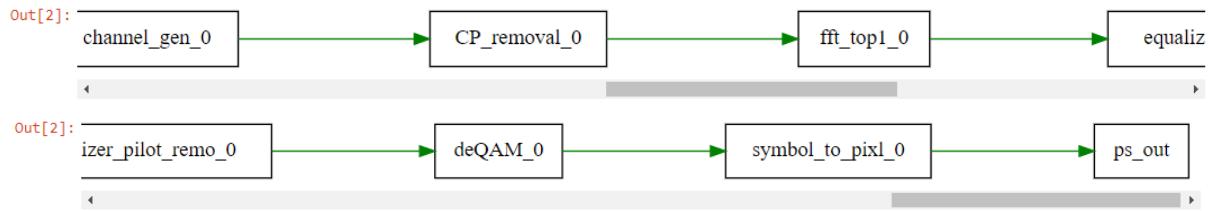
ol = Overlay("commun.bit")
cfILTER = ol.comm
print(f"Parent class: {cfILTER.__class__.__bases__}, driver: {type(cfILTER)}")
cfILTER.c_dict
```

並且將想要的IP串起來, 使用cfILTER.compose()把想要的IP連起來, 如下圖

```
# channel_total => ok
cfILTER.compose([cfILTER.ps_in, cfILTER.pixel_to_symbol_0, cfILTER.QAM_0, cfILTER.pilot_insertion_0, cfILTER.fft_top0_0, cfILTER.CP_insertion_0, cfILTER.channel_gen_0, cfILTER.CP_removal_0, cfILTER.fft_top1_0, cfILTER.equalizer_pilot_removal_0, cfILTER.deQAM_0, cfILTER.symbol_to_pixel_0, cfILTER.ps_out])
cfILTER.graph
```

回印出想串起來的IP圖, 如下圖





設定module_id以及parameter所需的值,

#-----ORIGINAL PIC-----#

將所要傳送的data讀進來, 這邊是使用cat_punch.jpg做舉例

#-----READ DATA OK-----#

將各個module id所需要的parameter設定在parameter_id

#-----PARA SET-----#

將要傳送的input_buffer根據parameter stream規格, 把parameter_id放進的
input_buffer

#-----DATA SET-----#

將要傳送的input_buffer根據parameter stream規格, 把data放進的input_buffer

#-----KERNEL START-----#

將input_buffer傳送進dma, 資料進入通訊系統, 並用output_buffer接dma出來的值

#-----KERNEL END-----#

將經過通訊系統傳送回來的值與原資料做比較, 計算bit error rate, 並將傳送回來的資
料印出圖片來

#-----PROCESS END-----#

```

dma_send = ol.axi_dma.sendchannel
dma_recv = ol.axi_dma.recvchannel

module_id = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12])
len_module_id = len(module_id)
len_parameter_id = [4,4,4,3,4,6,4,3,5,4,4,4] # # of parameters in each module
parameter_sum = sum(len_parameter_id)

qam_num = 16
sym_num = 2
pilot_width = 4
CP_length = 16
TAPS_NUM = 1
SNR = 20
FFT_len = 64

RGB_val = 3
print("#-----ORIGIANL PIC-----#")
cat_pic = Image.open("cat_punch.jpg")
plt.imshow(cat_pic)
plt.show()
cat_pic = cat_pic.rotate(270)
cat_pic = cat_pic.transpose(Image.FLIP_LEFT_RIGHT)
pic_array = allocate(shape=((cat_pic.size[0]*cat_pic.size[1]*3)), dtype=np.uint8)
for inx1 in range(cat_pic.size[0]):
    for inx2 in range(cat_pic.size[1]):
        pix = cat_pic.getpixel((inx1,inx2))
        pic_array[(inx1)*(cat_pic.size[1])*RGB_val+(inx2)*RGB_val+0] = pix[0]
        pic_array[(inx1)*(cat_pic.size[1])*RGB_val+(inx2)*RGB_val+1] = pix[1]
        pic_array[(inx1)*(cat_pic.size[1])*RGB_val+(inx2)*RGB_val+2] = pix[2]
print("#-----READ DATA OK-----#")

numSamples = cat_pic.size[0]*cat_pic.size[1]*3
input_compensate = (int)((FFT_len-(FFT_len/pilot_width))/sym_num)-(numSamples%((FFT_len-(FFT_len/pilot_width))/sym_num))
data_len = numSamples + input_compensate
total_input_num = data_len + parameter_sum + 1
in_num_sample = parameter_sum + 1 + data_len
out_num_sample = parameter_sum + 1 + data_len

input_buffer = allocate(shape=(in_num_sample,), dtype=np.uint64)
output_buffer = allocate(shape=(out_num_sample,), dtype=np.uint64)

BER = [0]
out_pic = allocate(shape=((cat_pic.size[0]*cat_pic.size[1]*3)), dtype=np.uint64)

parameter_id = np.array([[data_len,qam_num,sym_num,pilot_width], # module_id = 0 pixel2sym
                        [data_len,qam_num,sym_num,pilot_width], # module_id = 1 encoder
                        [data_len,qam_num,sym_num,pilot_width], # module_id = 2 QAM
                        [data_len,sym_num,pilot_width,CP_length], # module_id = 3 pilot insert
                        [data_len,sym_num,pilot_width], # module_id = 4 IFFT
                        [data_len,sym_num,pilot_width,CP_length], # module_id = 5 CP insert
                        [data_len,sym_num,pilot_width,CP_length,TAPS_NUM,SNR], # module_id = 6 channel
                        [data_len,sym_num,pilot_width,CP_length], # module_id = 7 CP remove
                        [data_len,sym_num,pilot_width], # module_id = 8 FFT
                        [data_len,sym_num,pilot_width,CP_length, TAPS_NUM], # module_id = 9 pilot remove
                        [data_len,qam_num,sym_num,pilot_width], # module_id = 10 deQAM
                        [data_len,qam_num,sym_num,pilot_width], # module_id = 11 decoder
                        [data_len,qam_num,sym_num,pilot_width]],dtype=object)# module_id = 12 sym2pixel

```

```

print("#-----PARA SET-----#")
temp = allocate(shape=(1,), dtype=np.uint64)
cnt_para_in = 0
for i in range(len_module_id):
    temp[0] = module_id[i] * pow(2,16)
    for j in range(len_parameter_id[i]):
        input_buffer[cnt_para_in+j] = temp[0]
        input_buffer[cnt_para_in+j] = input_buffer[cnt_para_in+j] + j
        input_buffer[cnt_para_in+j] = input_buffer[cnt_para_in+j] * pow(2,32)
        input_buffer[cnt_para_in+j] = input_buffer[cnt_para_in+j] + parameter_id[i][j]
    cnt_para_in = cnt_para_in+len_parameter_id[i]
input_buffer[parameter_sum] = (pow(2,16)-1)*pow(2,48)

print("#-----DATA SET-----#")
for k in range(data_len):
    if (k<numSamples):
        input_buffer[parameter_sum+k+1] = pic_array[k]
    else:
        input_buffer[parameter_sum+k+1] = 0

print("#-----KERNEL START-----#")
dma_send.transfer(input_buffer)
dma_recv.transfer(output_buffer)
dma_send.wait()
dma_recv.wait()
print("#-----KERNEL END-----#")

para_out_cnt = 0
while (output_buffer[para_out_cnt]/math.pow(2,48) != 65535):
    para_out_cnt = para_out_cnt + 1

ber = 0
pixl_length = 8

for k in range(numSamples):
    err = '{:08b}'.format(output_buffer[k+para_out_cnt+1]^input_buffer[k+parameter_sum+1]).count('1')
    ber = ber + err
    out_pic[k] = output_buffer[k+para_out_cnt+1]

BER = ber/numSamples/pixl_length*100
print("BER = ", BER, "%");

out_pic_matrix = out_pic.reshape(cat_pic.size[0],cat_pic.size[1],3)
plt.imshow(out_pic_matrix)
plt.show()
print("#-----PROCESS END-----#")

```

最終會印出下圖:

```

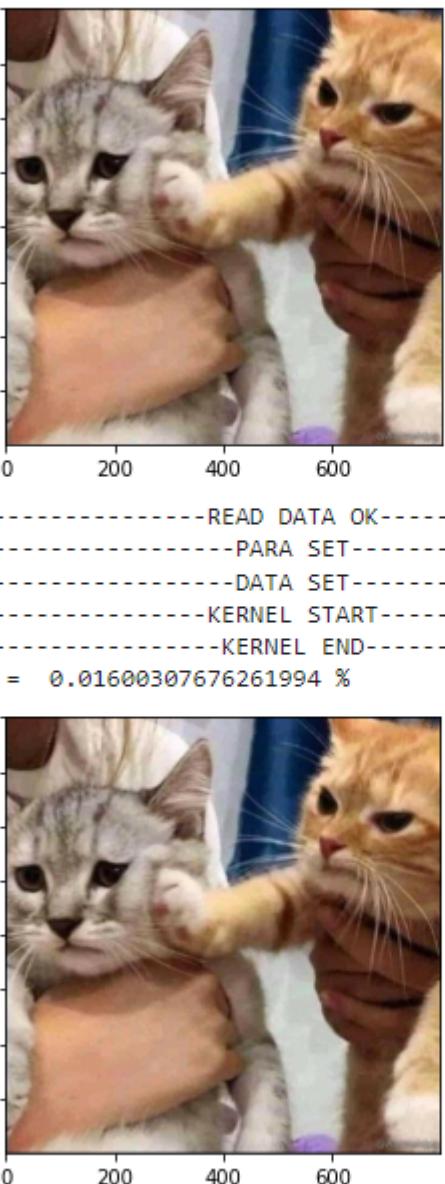
#-----ORIGIANL PIC-----#
0
100
200
300
400
500
600
700
0 200 400 600

#-----READ DATA OK-----#
#-----PARA SET-----#
#-----DATA SET-----#
#-----KERNEL START-----#
#-----KERNEL END-----#
BER = 0.01600307676261994 %

0
100
200
300
400
500
600
700
0 200 400 600

#-----PROCESS END-----#

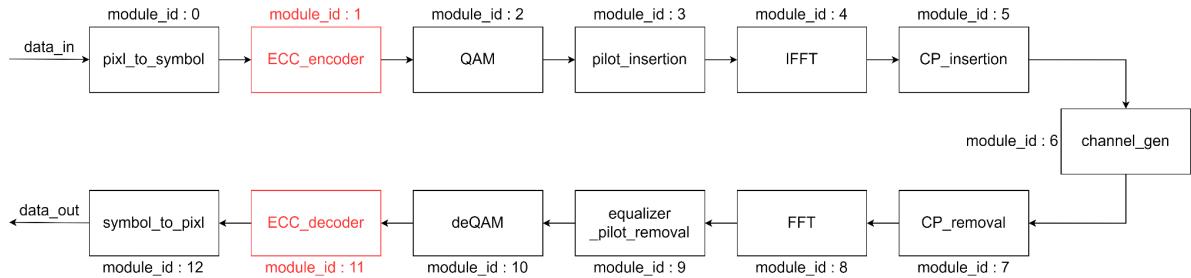
```



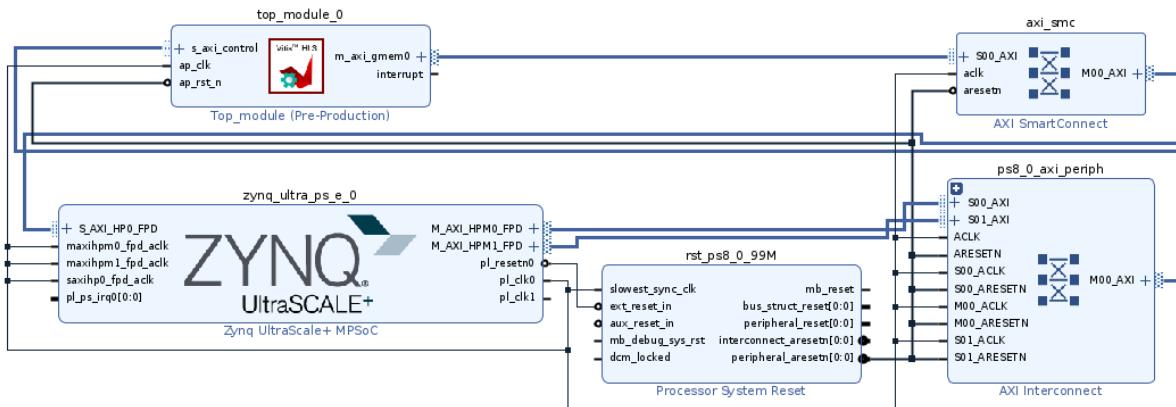
Lab

> Error Correction Code

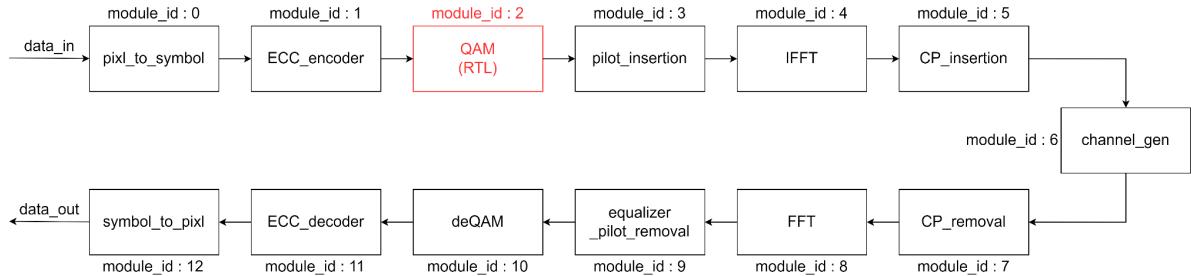
此實驗目的是希望可以透過實作將Error Correction Code的source code加入通訊系統中，並將完成的系統在FPGA上面進行實現，來讓同學了解在Composable Pipeline中新增Kernel的流程，以下為加入ECC後的Block Diagram：



請根據前面講解的C++ kernel的方法在composable pipeline中加入ECC encoder和decoder的IP，並在Jupyter Notebook host端比較加入ECC前後的BER (SNR在0到30每五為一個單位)，並繪製曲線圖進行比較。



> RTL Kernel



請根據前面講解的RTL kernel的方法在composable pipeline中加入QAM的RTL IP，並在Jupyter Notebook host端確認使用C++ kernel和RTL kernel的結果一致。