Communication Training System

Workbook

Table of Contents

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應，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
使用。

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:

```
ap_axiu_64 para_in;
ap_uint<16> module_id;
ap_uint<16> para_id;
ap_uint<32> para_val;

module_id = QAM_module_num; //給定參數所屬模組的id
para_id = 0; //給定參數在所屬模組中的id
para_val = DATA_LEN; //給定參數數值
para_in.data = (module_id, para_id, para_val); //合併所有資訊
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。

若是要在現有的kernel中新增參數，只需要

。**Jupyter Notebook Host Code**

以下為在host端

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

data_len = 480
qam_num = 16
sym_num = 2
pilot_width = 4
CP_length = 16
TAPS_NUM = 1
SNR = 30
FFT_len = 64

parameter_id = np.array([[data_len,qam_num,sym_num,pilot_width], # module_id = 0 pixl2sym
                         [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]],dtype=object)# module_id = 12 sym2pixl
```

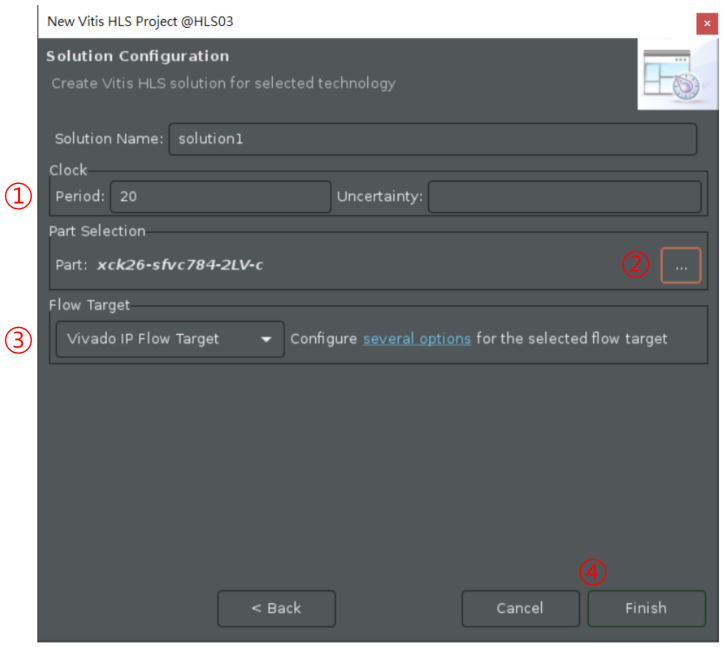
(a) module_id : 填入會使用到的module對應的module_id，擺放順序隨意。
(b) len_parameter_id : 每個module中所會用到的參數個數，順序需和module_id的位置對應。
(c) 給定會使用到的參數數值。
(d) parameter_id : module順序需與module_id對應，每一個module內部的參數順序需與

New IP Intergration

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

> C++ Kernel IP

。Project Setting
① Clock Period : 20ns
② Part : xck26-sfvc784-2LV-c
③ Flow Target : Vivado IP Flow Target
④ Finish



。CSIM

。Synthesis

在合成之前需要先對欲合成的Block設定Directive,
選擇Source File
data_in和data_out設定為axis
block type protocol設定為ap_ctrl_none

。COSIM

用於驗證合成完的
是否符合

。Export IP

最後將設計

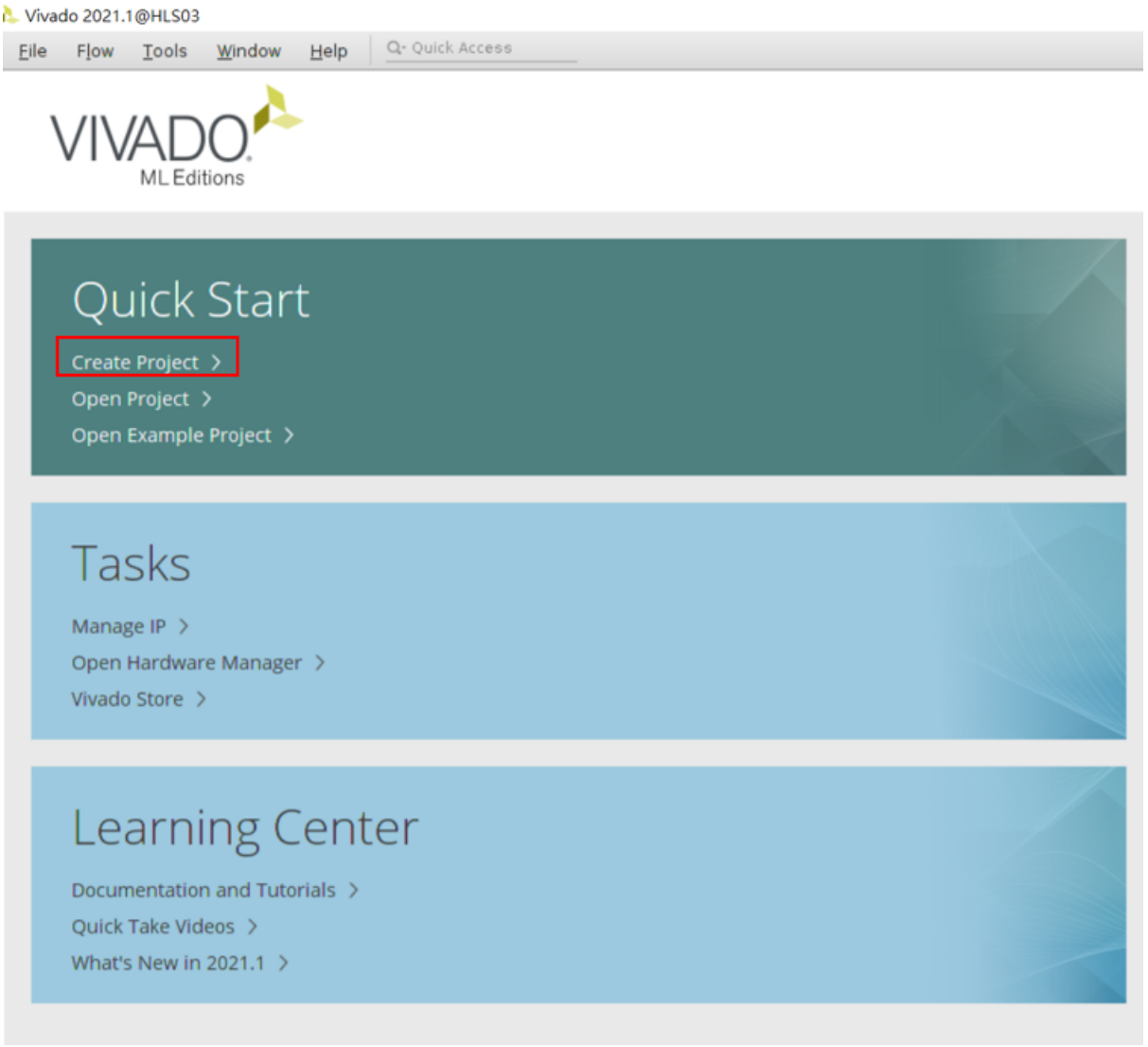
完成以上步驟之後便可以進入vivado進行composable pipeline的連接。

> Verilog Kernel IP

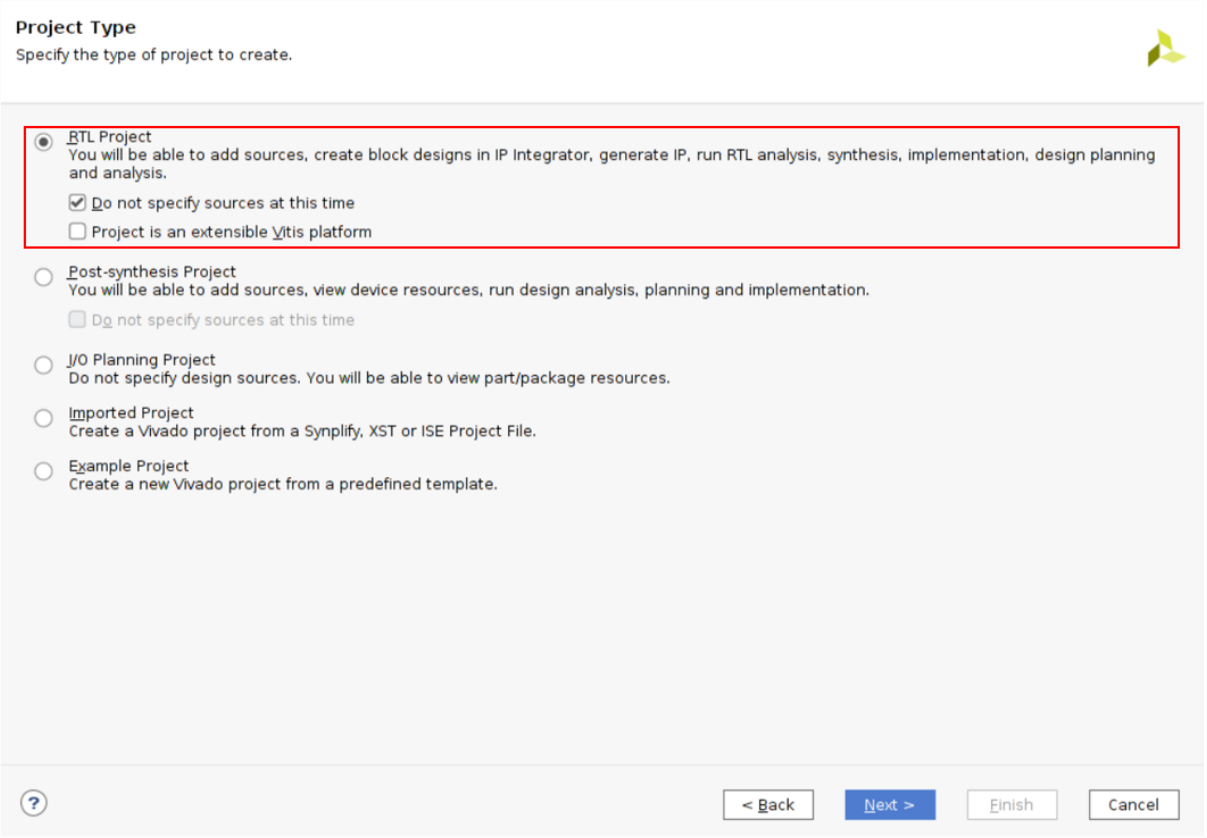
。Project Setting

。**Create vivado project**

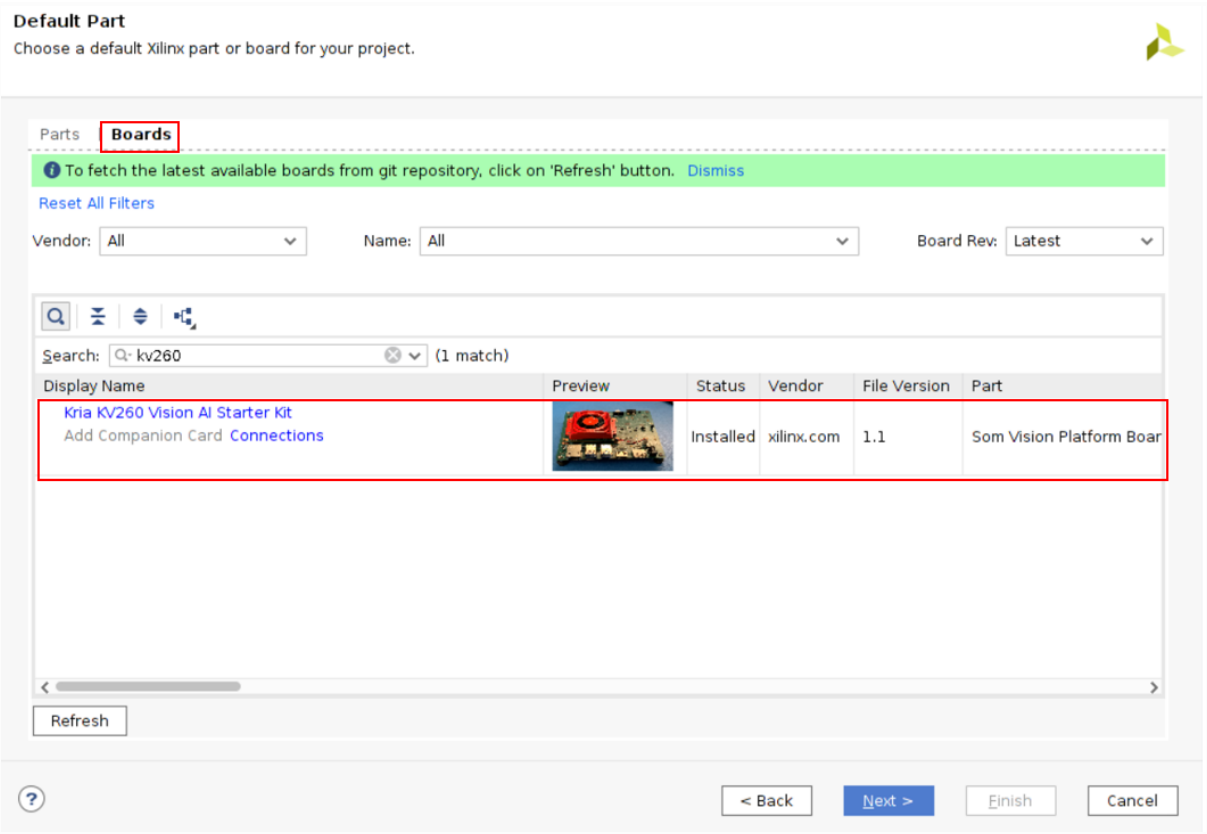
啟動vivado開發套件
開啟新的專案，設定好專案存放的路徑:



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



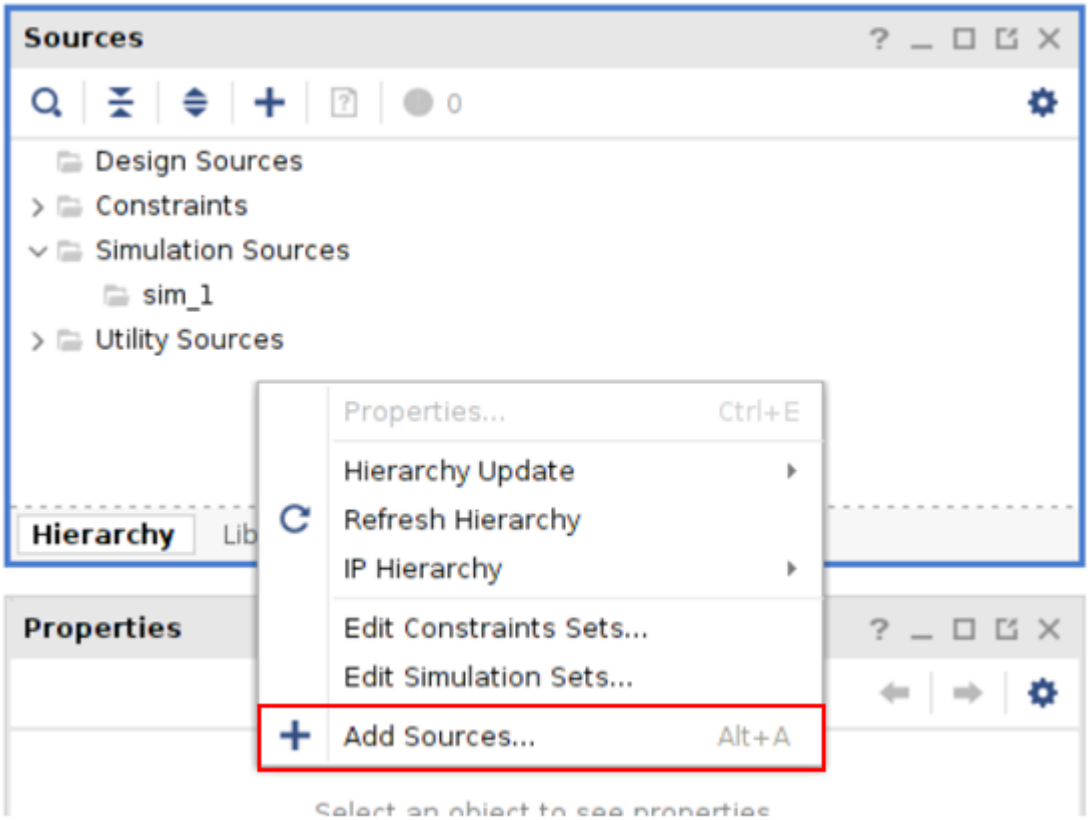
選擇Borads，並在下方選 Kria KV260 Vision AI Started Kit，點選Next>:



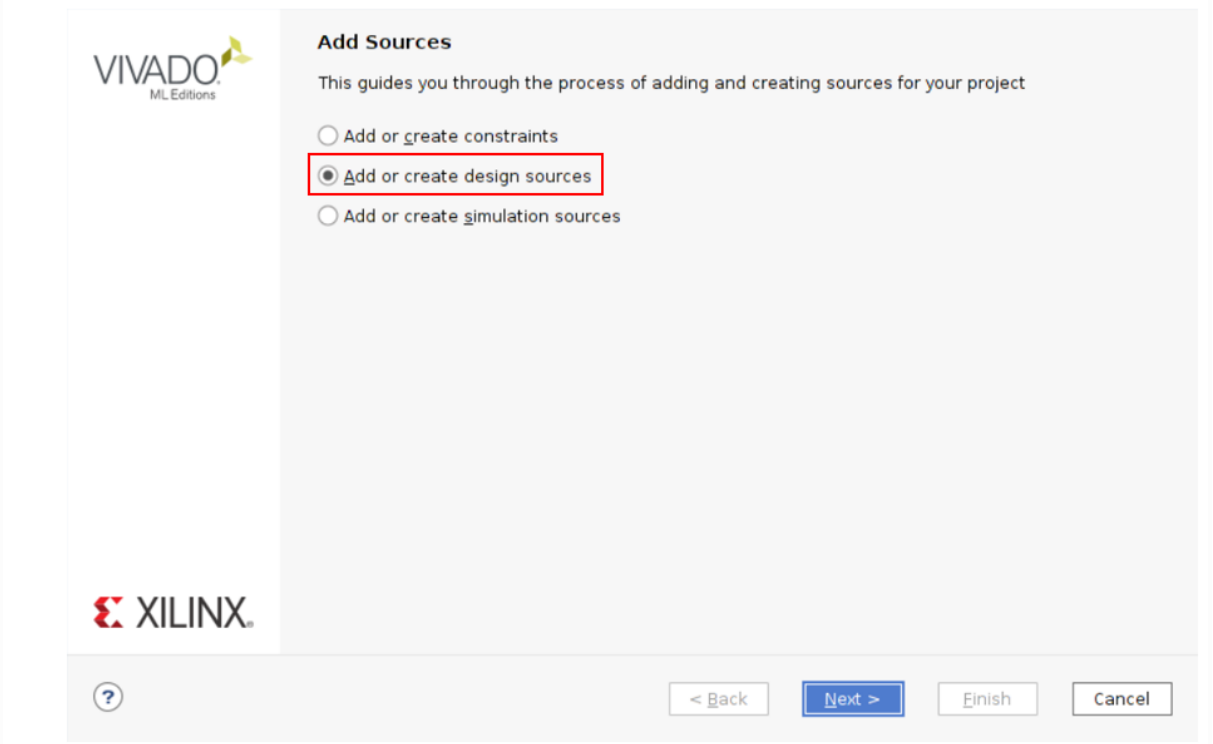
最後點選Finish，完成Create Project。

。Create design source code

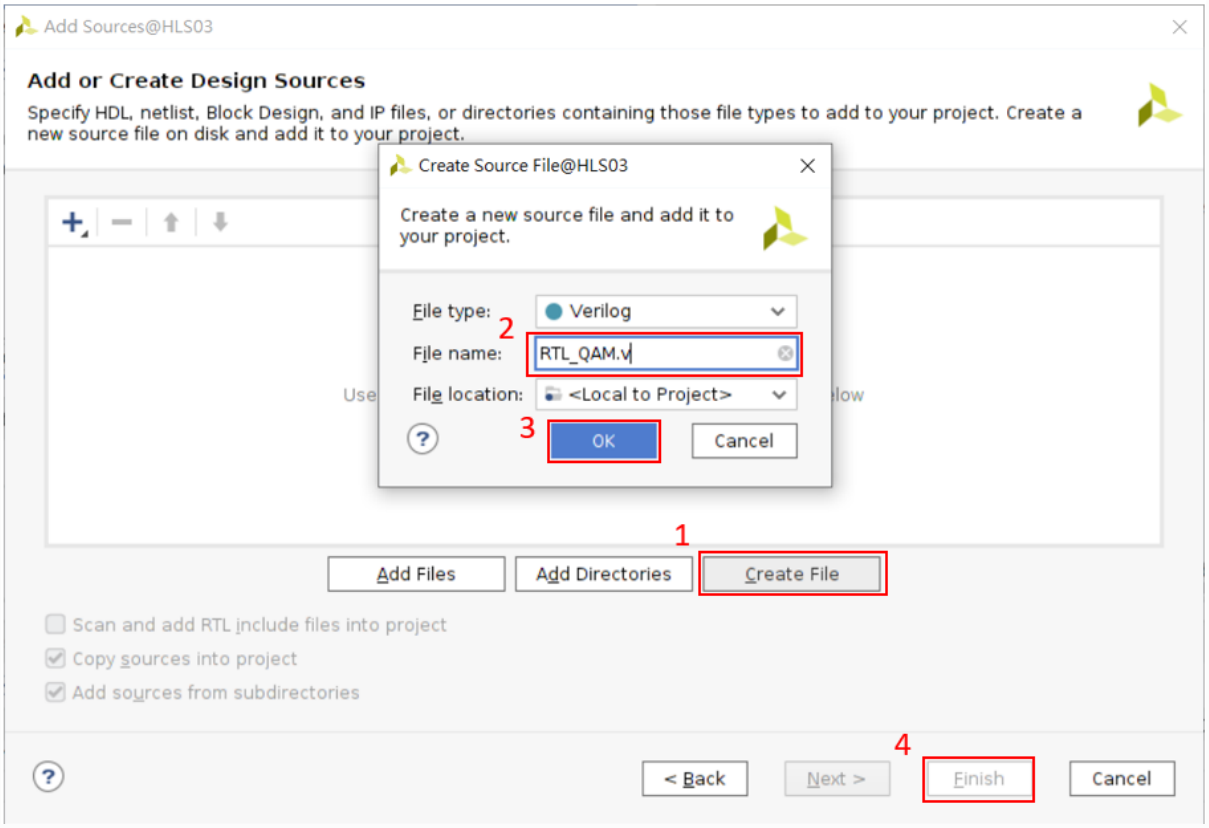
在Sources的窗口空白處按右鍵，點選Add Sources…

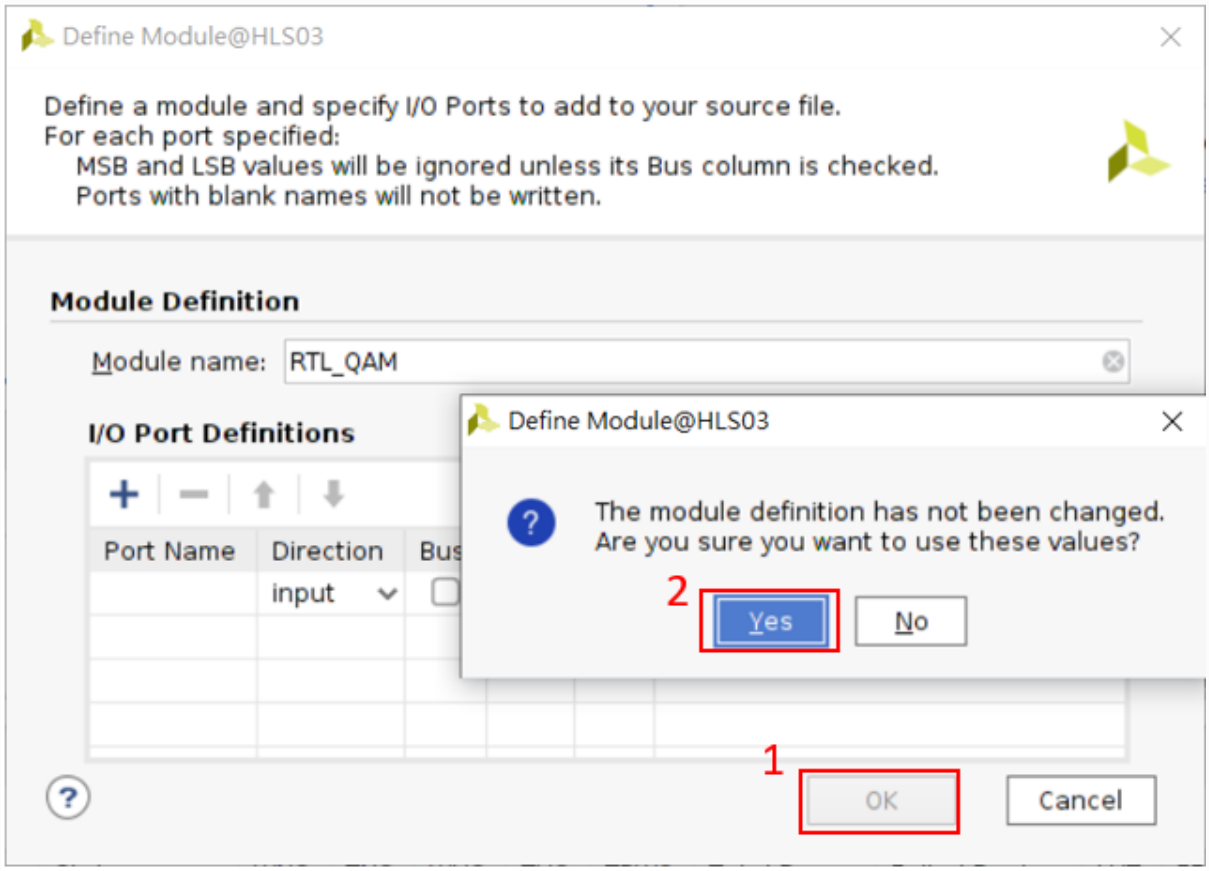


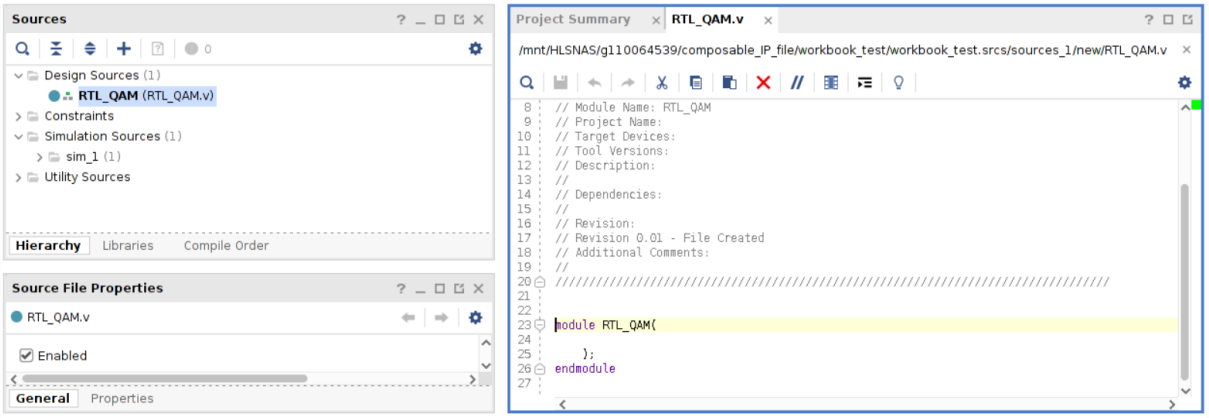
選擇Add or create design sources，點選Next >



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


Add Sources@HLS03
Add or Create Design Sources
Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.
Create Source File@HLS03
Create a new source file and add it to your project.
File type:
Verilog
2
File name:
RTL_QAM.v
File location:
<Local to Project>
3
OK
Cancel
1
Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
4
< Back
Next >
Finish
Cancel

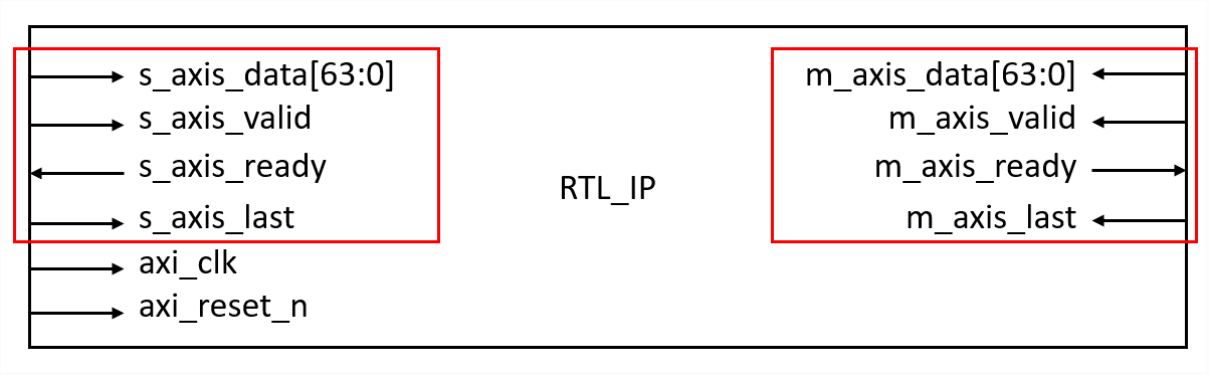

Define Module@HLS03
Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.
Module Definition
Module name:
RTL_QAM
I/O Port Definitions
Port Name
Direction
Bus
input
Define Module@HLS03
The module definition has not been changed.
Are you sure you want to use these values?
2
Yes
No
1
OK
Cancel



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

。Design verilog code

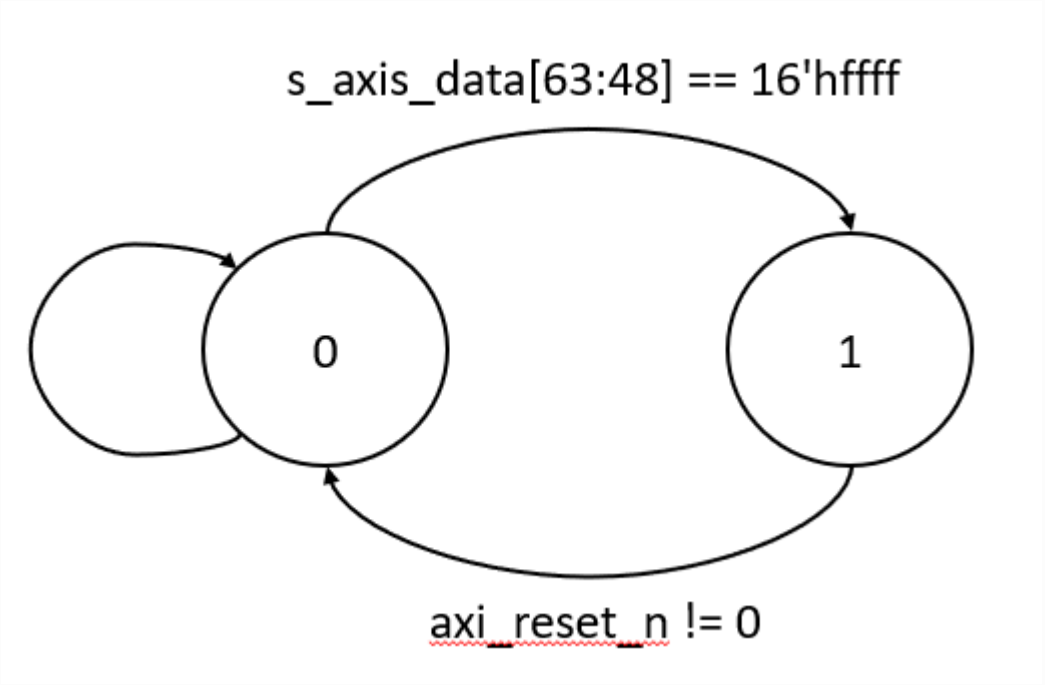
我們開始設計RTL，首先先確定好input及output的訊號，下圖是AXI4-stream 的ports:



因此verilog程式碼 input及output ports:

```
module RTL_QAM #(parameter DATA_WIDTH=64,
                 parameter IN_WL=22,
                 parameter IN_IL=7)
    (
     input    axi_clk,
     input    axi_reset_n,
     //AXI4-S slave i/f
     input    s_axis_valid,
     input [DATA_WIDTH-1:0] s_axis_data,
     input s_axis_last,
     output s_axis_ready,
     //AXI4-S master i/f
     output reg  m_axis_valid,
     output reg [DATA_WIDTH-1:0] m_axis_data,
     output reg m_axis_last,
     input    m_axis_ready
     );
```

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):



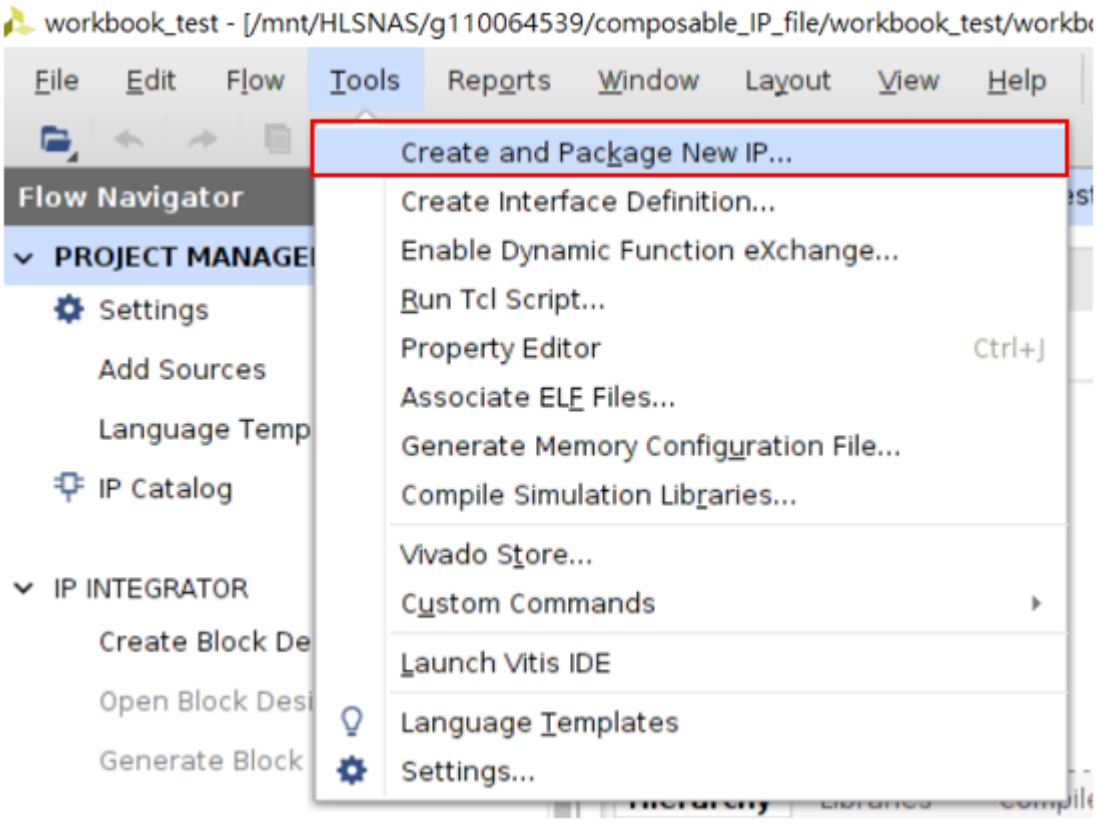
其中state=0是在parameter stream，因此RTL不做事，當s_axis_data[63:48] == 16'hffff時，表示parameter stream傳完，接下來進入state=1，開始做這個IP的功能。
以下是設計16-QAM IP:

```
41      reg [DATA_WIDTH-1:0] n_m_axis_data;
42      reg cs,ns;
43
44
45
46      always @(posedge axi_clk ) begin   //or negedge axi_reset_n
47          if(~axi_reset_n) begin
48              m_axis_data <= 0;
49              m_axis_last <= 0;
50              m_axis_valid <= 0;
51              cs <= 0;
52              //s_axis_ready <= 0;
53          end
54          else begin
55              m_axis_data <= n_m_axis_data;
56              m_axis_last <= s_axis_last;
57              m_axis_valid <= s_axis_valid;
58              cs <= ns;
59              //s_axis_ready <= m_axis_ready;
60          end
61      end
62      //----------------------------------------combination-----------------------------------//
63      assign s_axis_ready = m_axis_ready;
64      always @(*) begin
65         if(s_axis_valid & s_axis_ready) begin
66             case(cs)
67                  1'd0: begin
68                      n_m_axis_data = s_axis_data;
69                      if(s_axis_data[63:48] == 16'hffff) begin
70                          ns = 1'd1;
71                      end
72                      else begin
73                          ns = 1'd0;
74                      end
75                  end
76                  default: begin
```

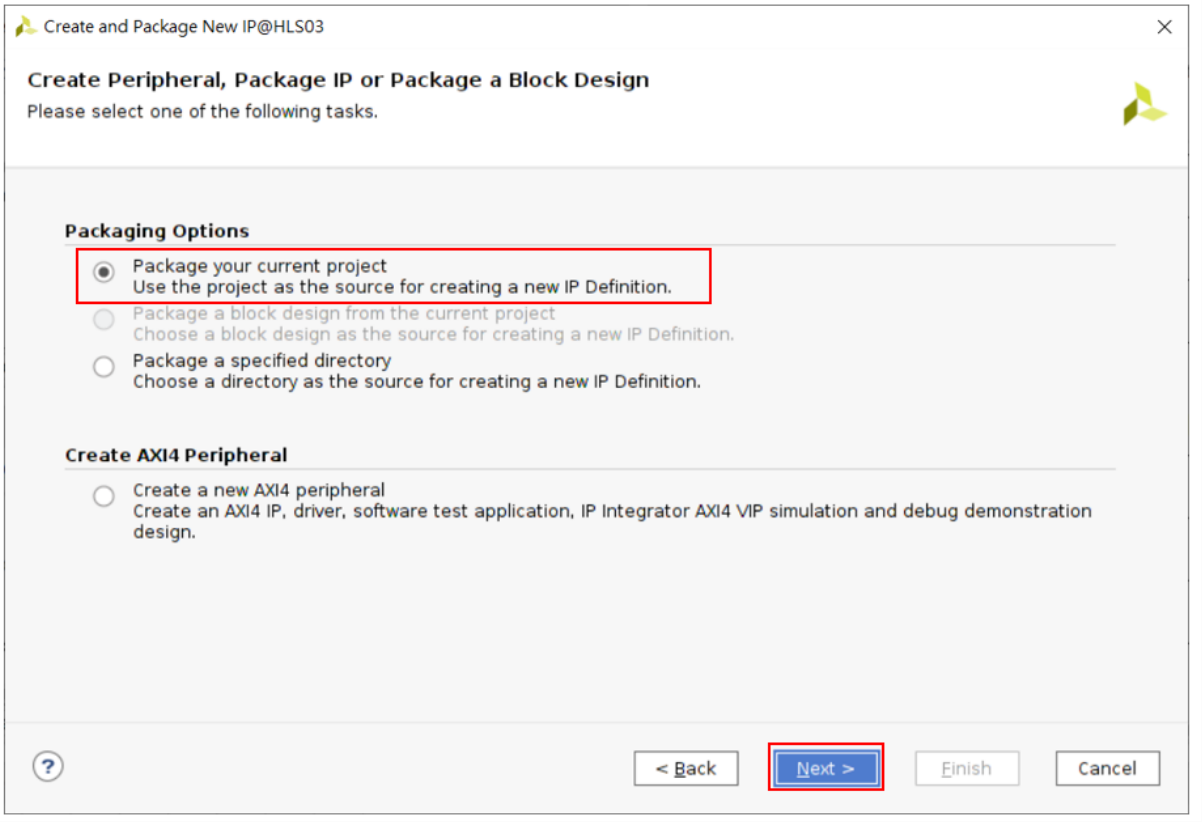
```
                    ns = cs;
                    case(s_axis_data[1:0])
                        2'd0: begin   //0
                            n_m_axis_data[IN_WL-1:IN_WL-IN_IL] = -7'd3;
                        end
                        2'd1: begin   //1
                            n_m_axis_data[IN_WL-1:IN_WL-IN_IL] = -7'd1;
                        end
                        2'd2: begin   //2
                            n_m_axis_data[IN_WL-1:IN_WL-IN_IL] = 7'd3;
                        end
                        default: begin //3
                            n_m_axis_data[IN_WL-1:IN_WL-IN_IL] = 7'd1;
                        end
                    endcase
                    case(s_axis_data[3:2])
                        2'd0: begin   //0
                            n_m_axis_data[32+IN_WL-1:32+IN_WL-IN_IL] = -7'd3;
                        end
                        2'd1: begin   //1
                            n_m_axis_data[32+IN_WL-1:32+IN_WL-IN_IL] = -7'd1;
                        end
                        2'd2: begin   //2
                            n_m_axis_data[32+IN_WL-1:32+IN_WL-IN_IL] = 7'd3;
                        end
                        default: begin //3
                            n_m_axis_data[32+IN_WL-1:32+IN_WL-IN_IL] = 7'd1;
                        end
                    endcase
                end
            endcase
        end
        else begin
            n_m_axis_data = m_axis_data;
            ns = cs;
        end
    end

endmodule
```

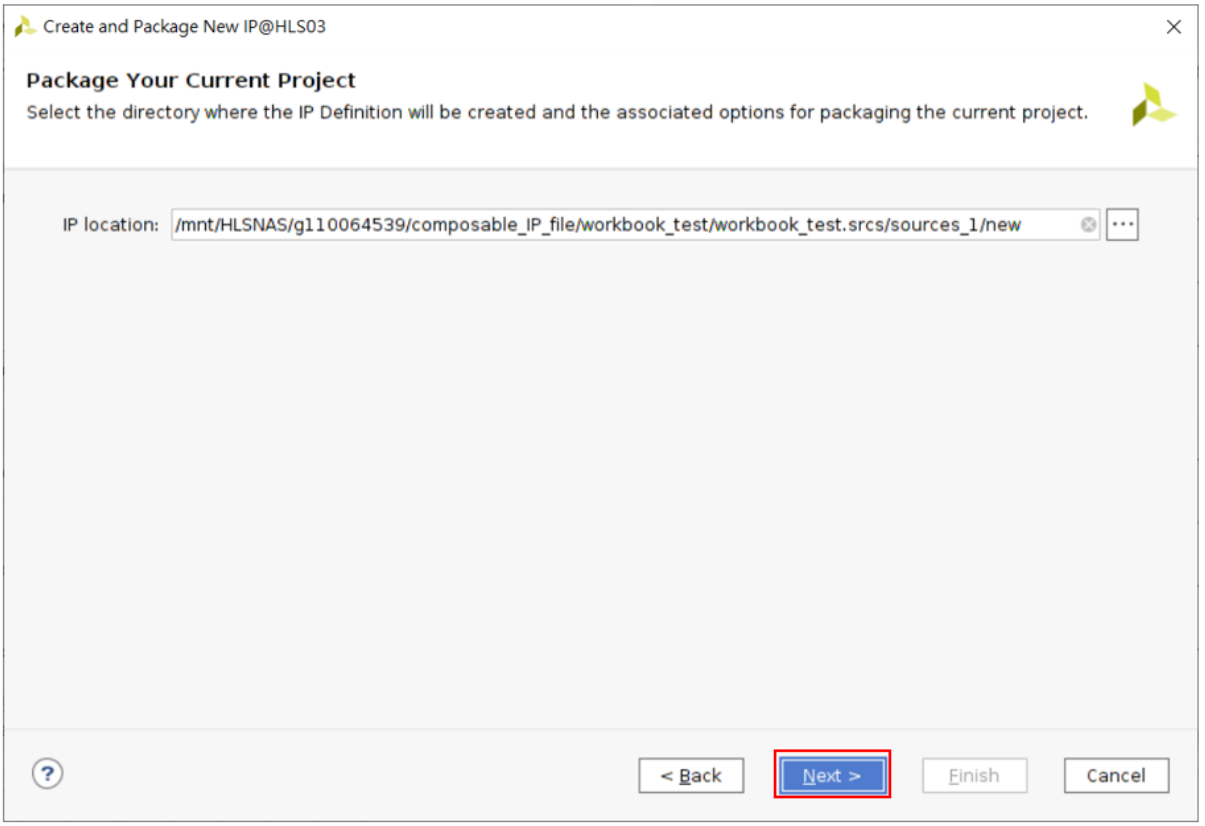
。**Package IP**

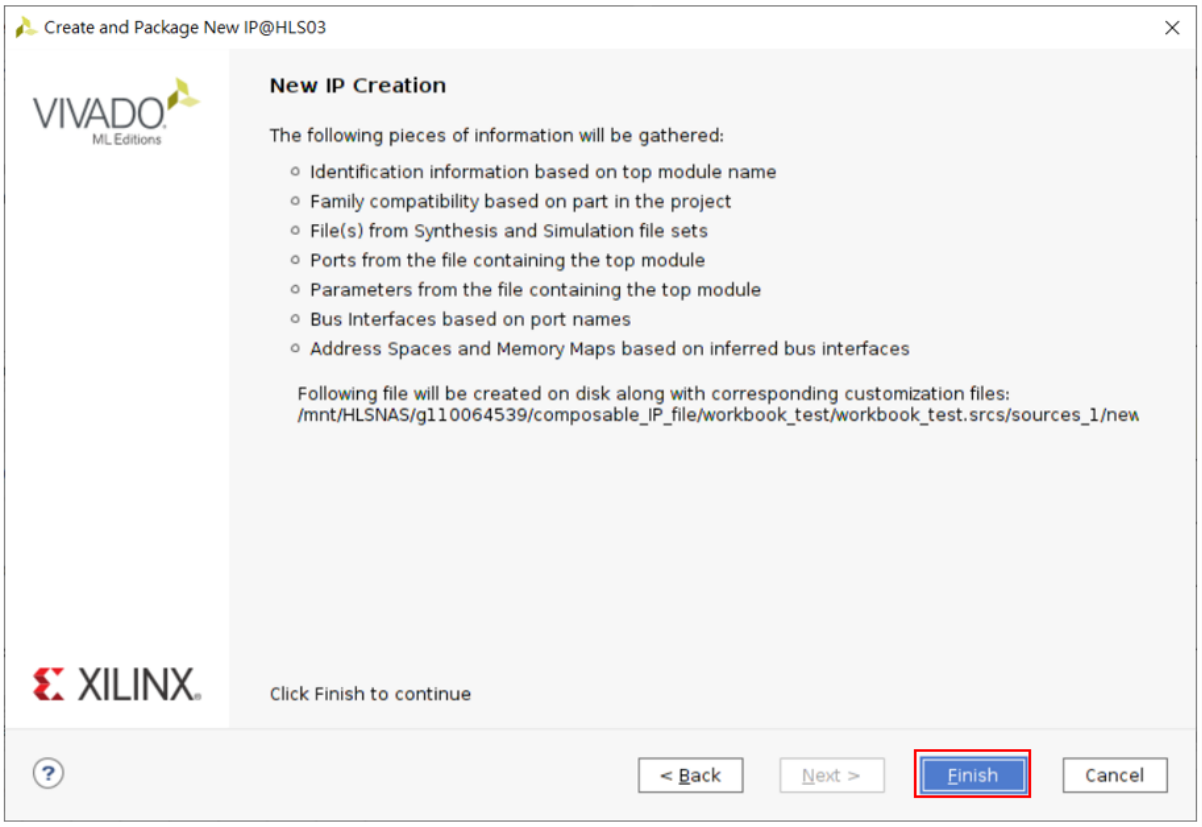
驗證完RTL功能後，將RTL包成IP，首先在Tools選單中點選Create and Package New IP…



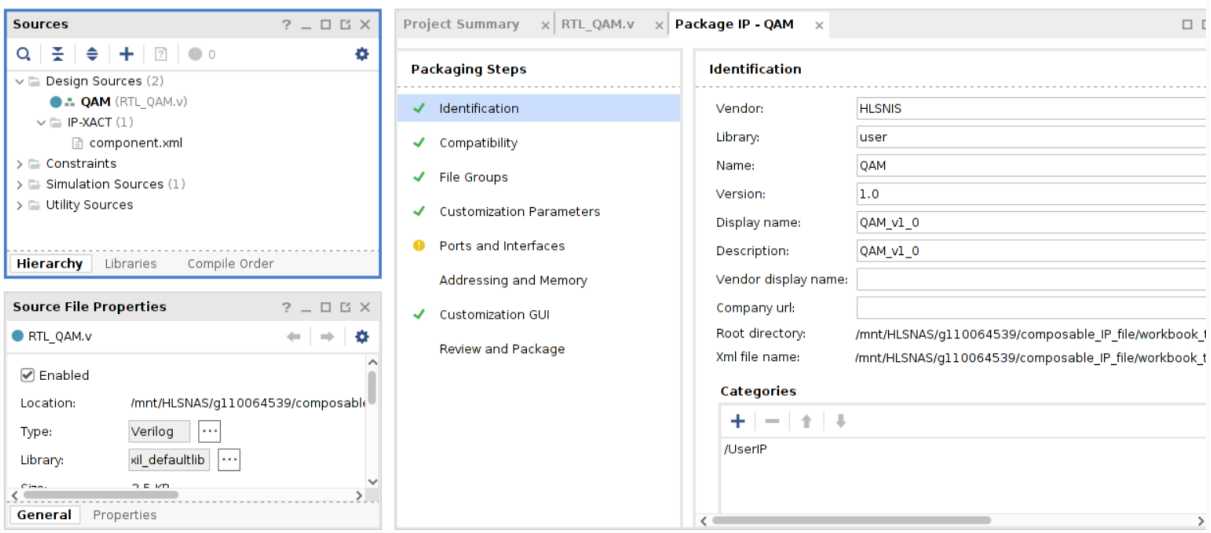
接下來按Next >，會到下圖，選擇Package your current project，按Next >



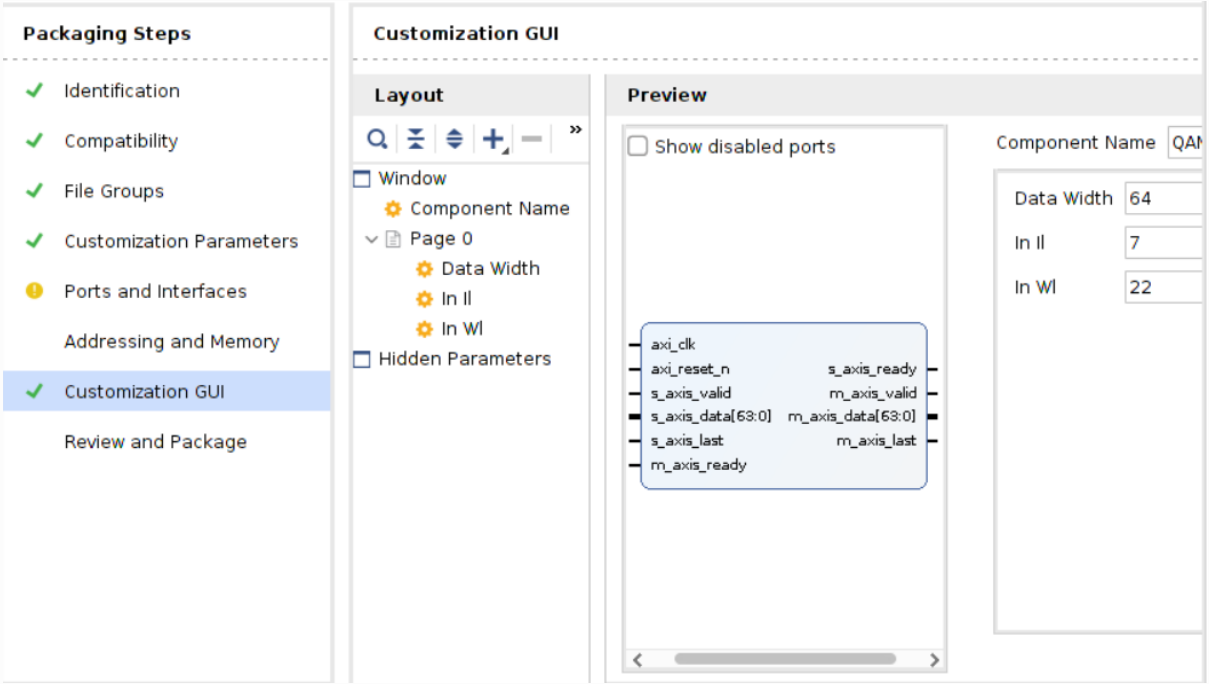




會生成IP-XACT -> component.xml檔



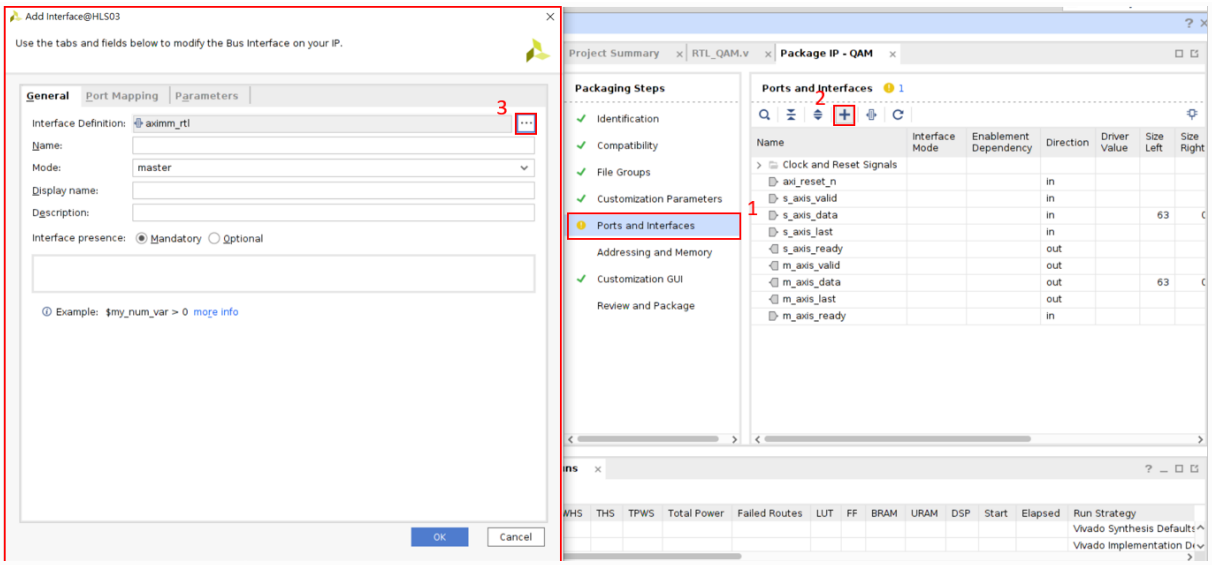
可以在Customization GUI中看到包出來的IP的ports:



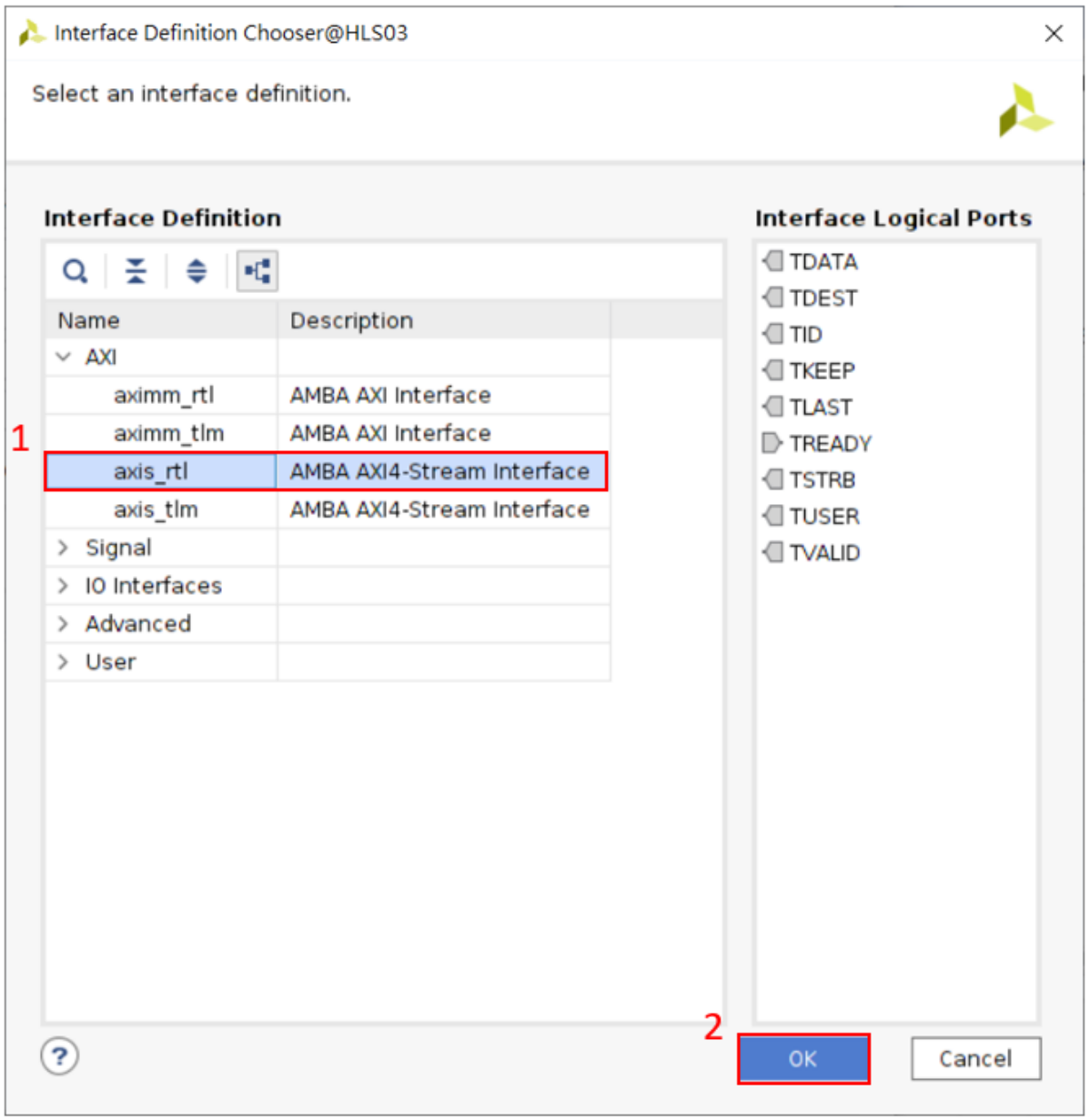
接下來我們將AXI4-stream的ports包起來:

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

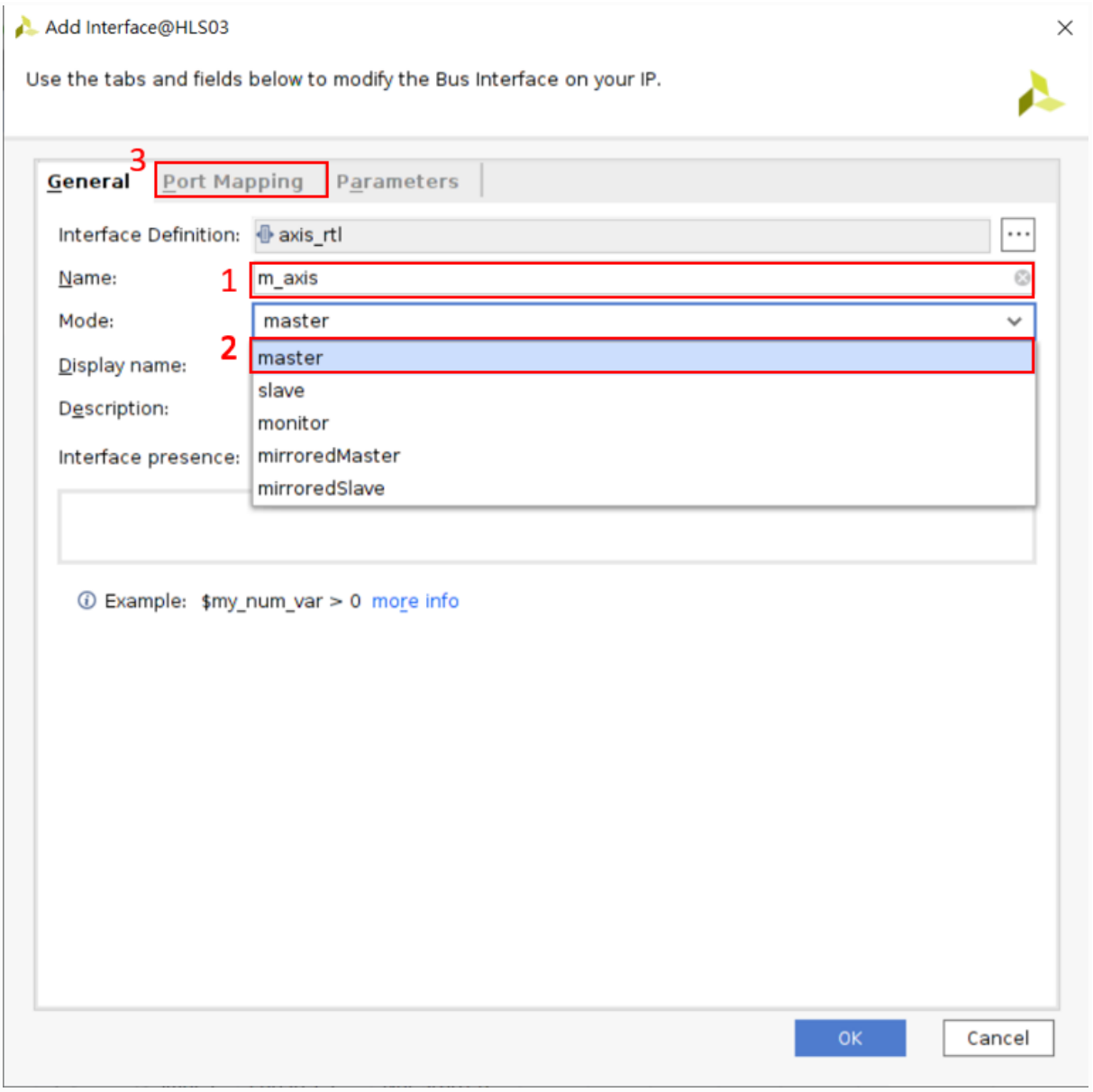
點選Interface Definition後面的  。



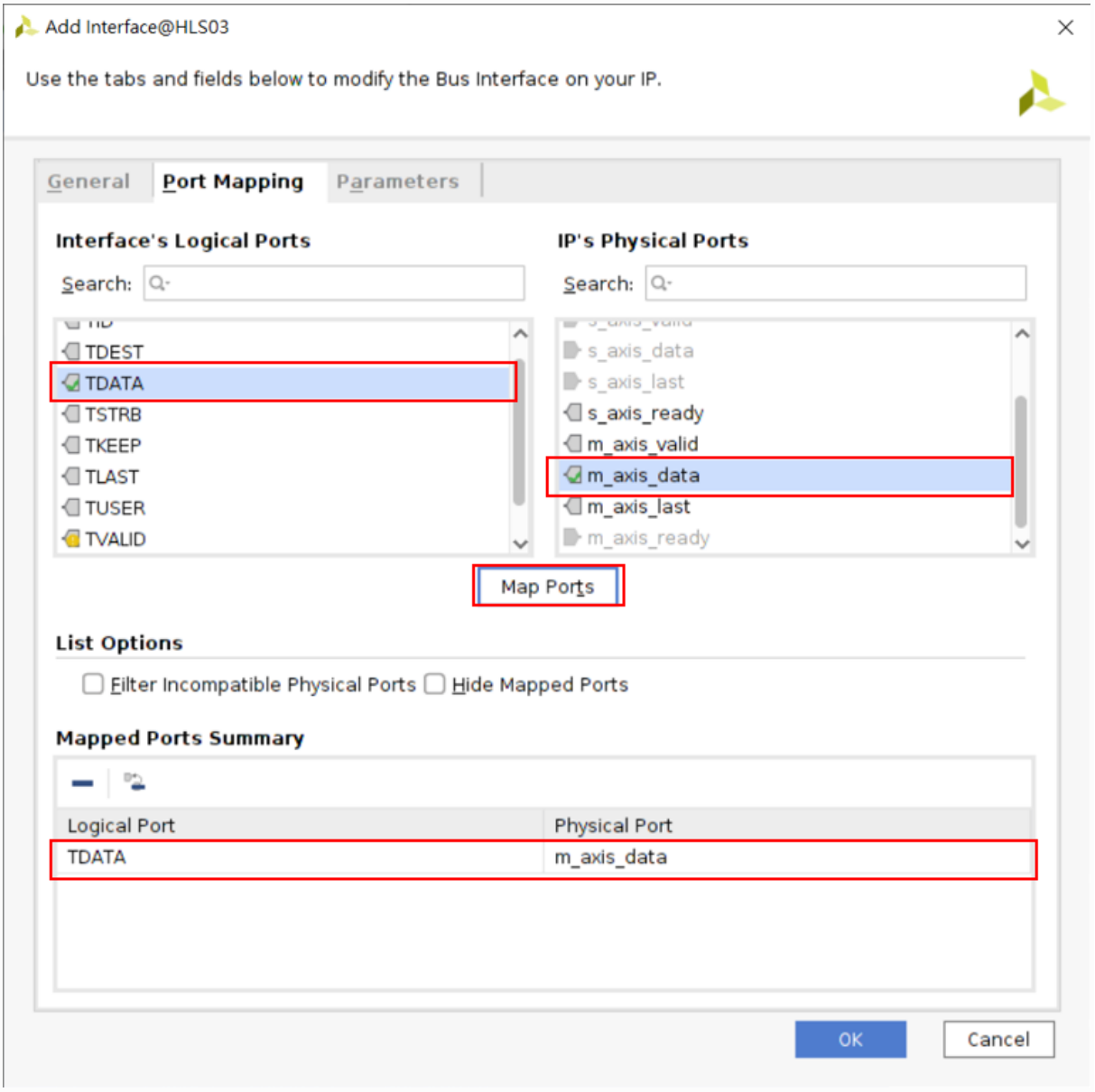
将interface改成axi4-stream的型態



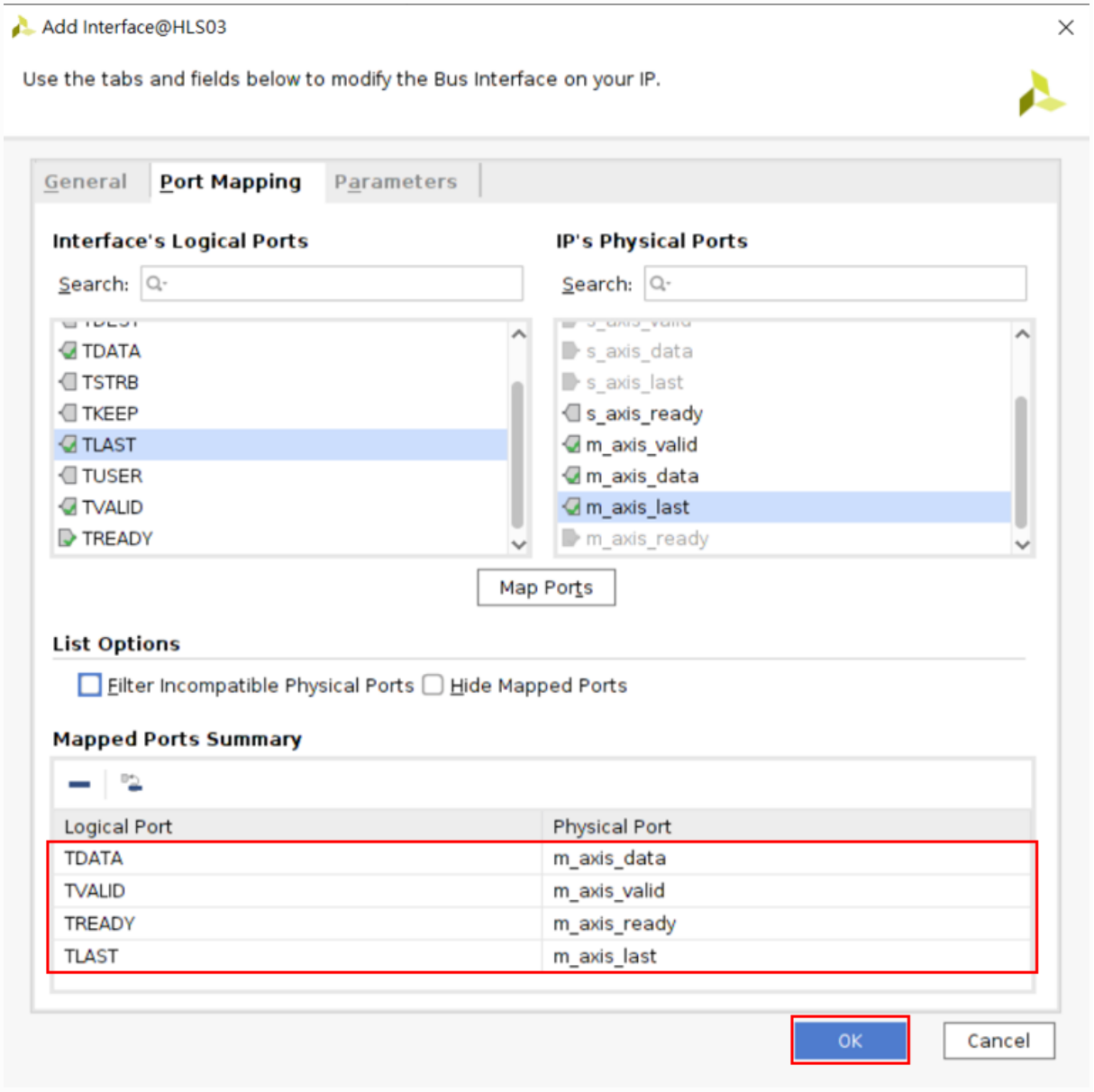
先設定master stream，將Name設定為”m_axis”(可以自己定義)，Mode需要選”master”，接下來點Port Mapping將master stream的ports包在m_axis裡。

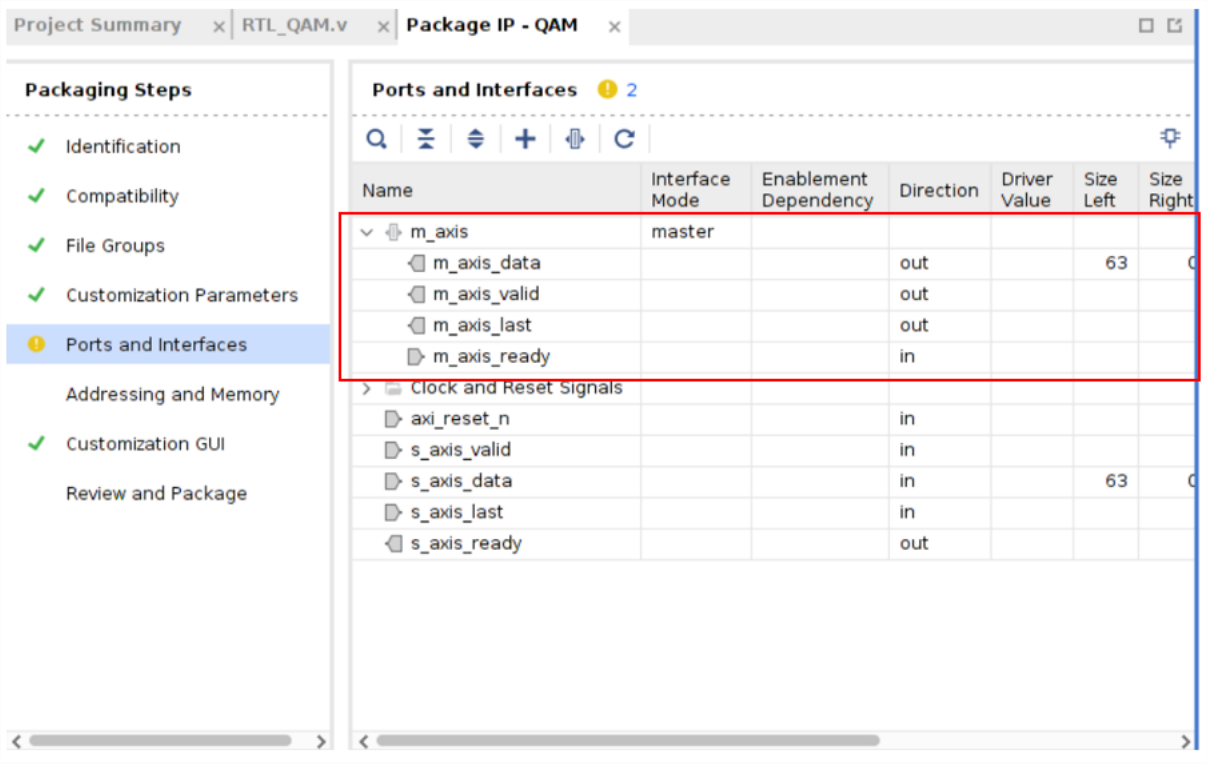


將Interface’s Logical Ports與對應的IP’s Physical Ports用map起來
點Interface’s Logical Ports的"TDATA"，並點選IP’s Physical Ports的”m_axis_data
再點選"Map Ports”，將這兩個map起來，會在下面的Mapped Ports Summary看到結果。

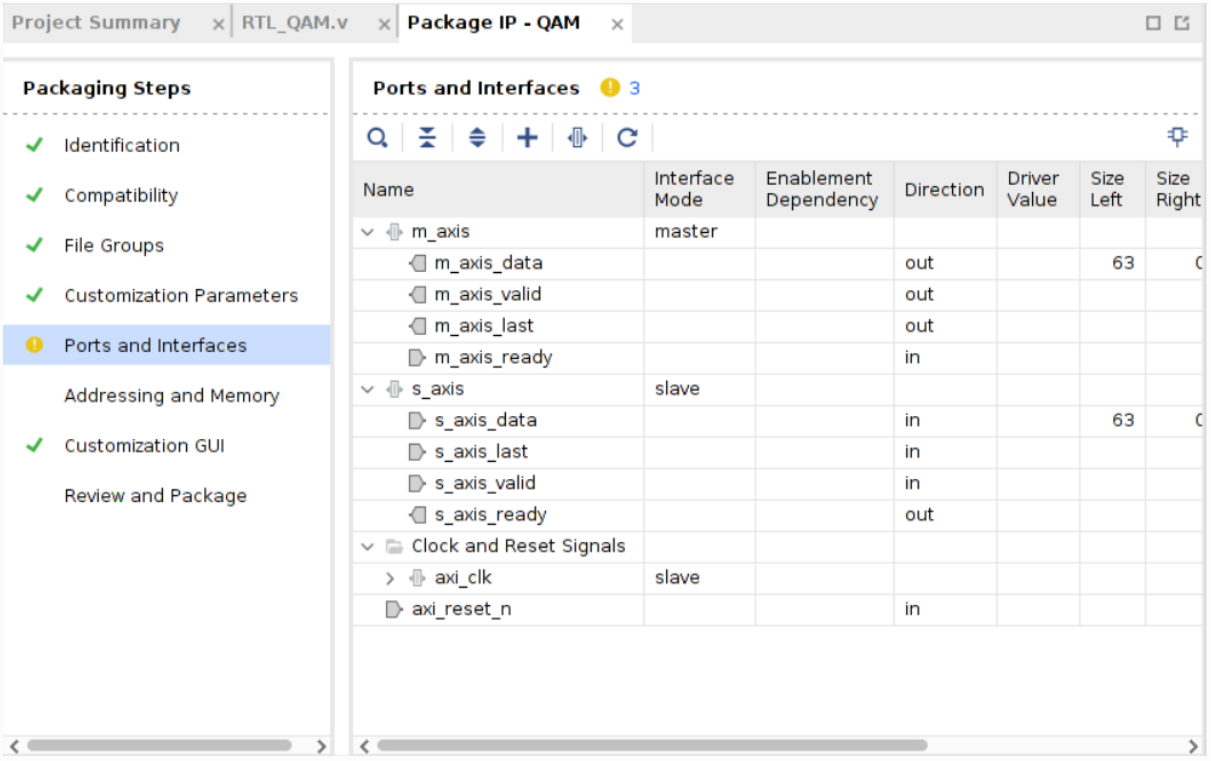


將TDATA、TVALID、TREADY、TLAST都mapping好，如下圖，完成後按OK，會看到m_axis將四個ports包起來。

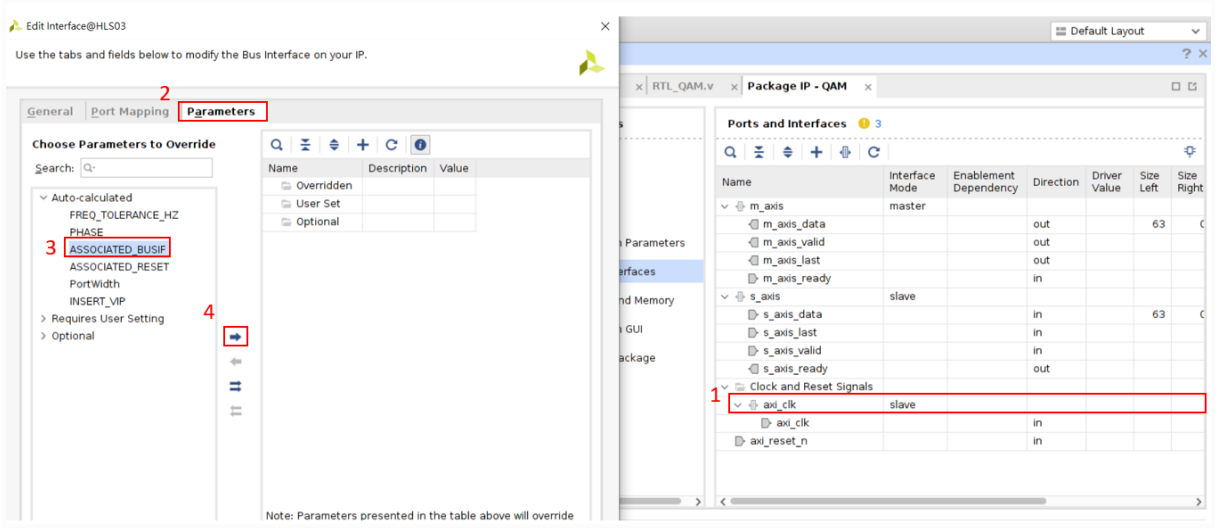

Add Interface@HLS03
Use the tabs and fields below to modify the Bus Interface on your IP.
General
Port Mapping
Parameters
Interface's Logical Ports
Search:
TDATA
TSTRB
TKEEP
TLAST
TUSER
TVALID
TREADY
IP's Physical Ports
Search:
s_axis_data
s_axis_last
s_axis_ready
m_axis_valid
m_axis_data
m_axis_last
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



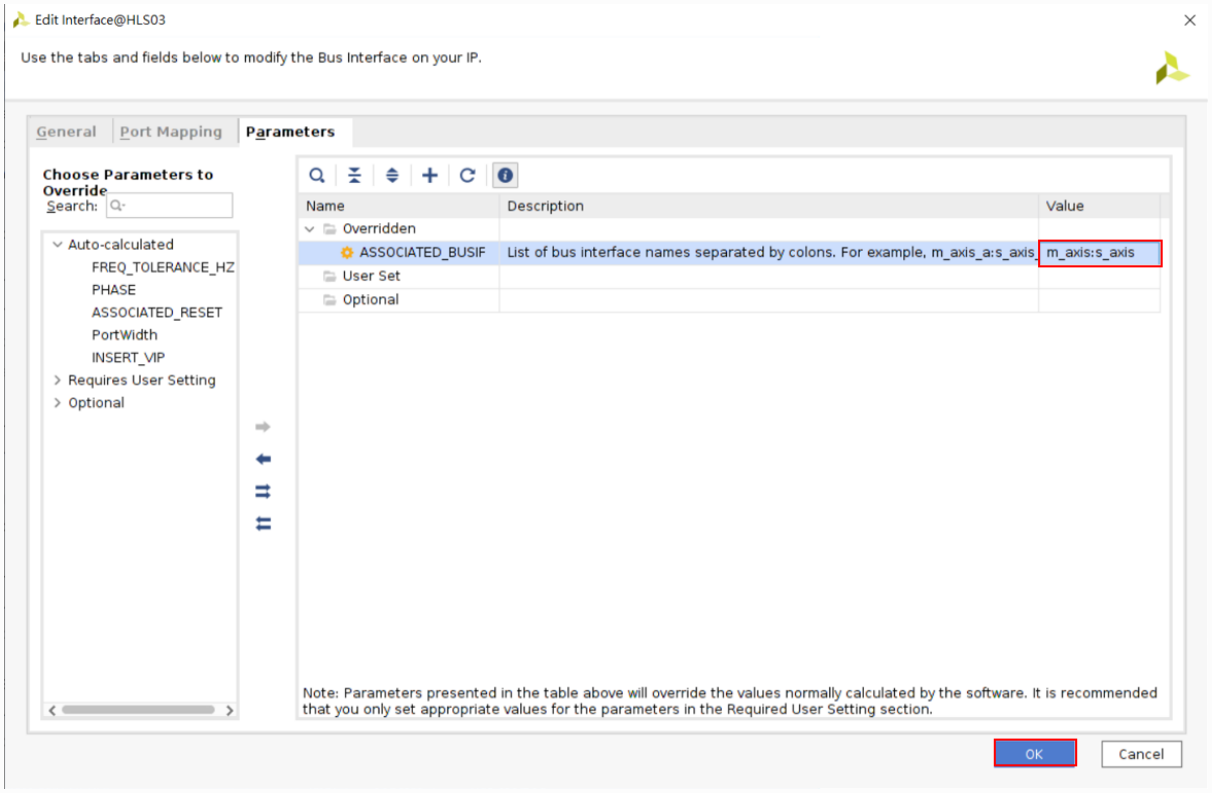
接下來用相同的方式將s_axis_data、s_axis_last、s_axis_valid、s_axis_ready包起來在s_axis裡，如下圖:



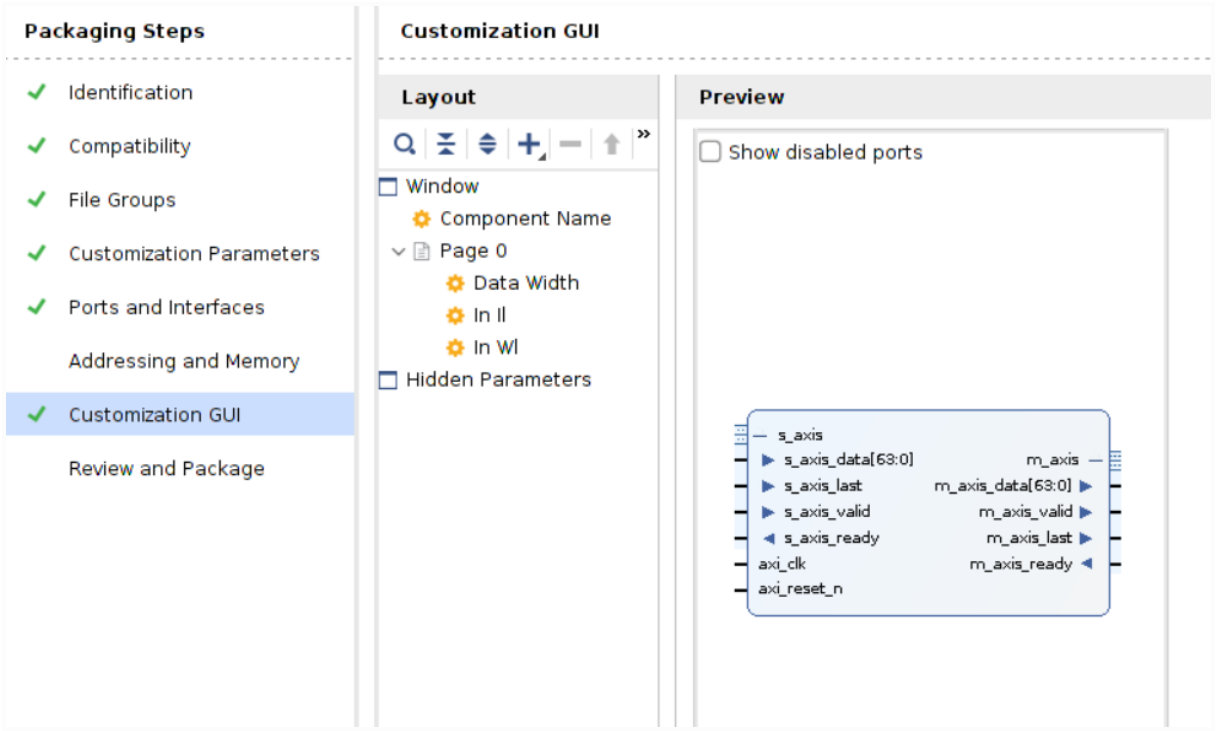
設定clk，對axi_clk左鍵雙擊，會跳出Edit Interface，"Parameters”選單，選"ASSOCIATED_BUSIF"，按 ➡



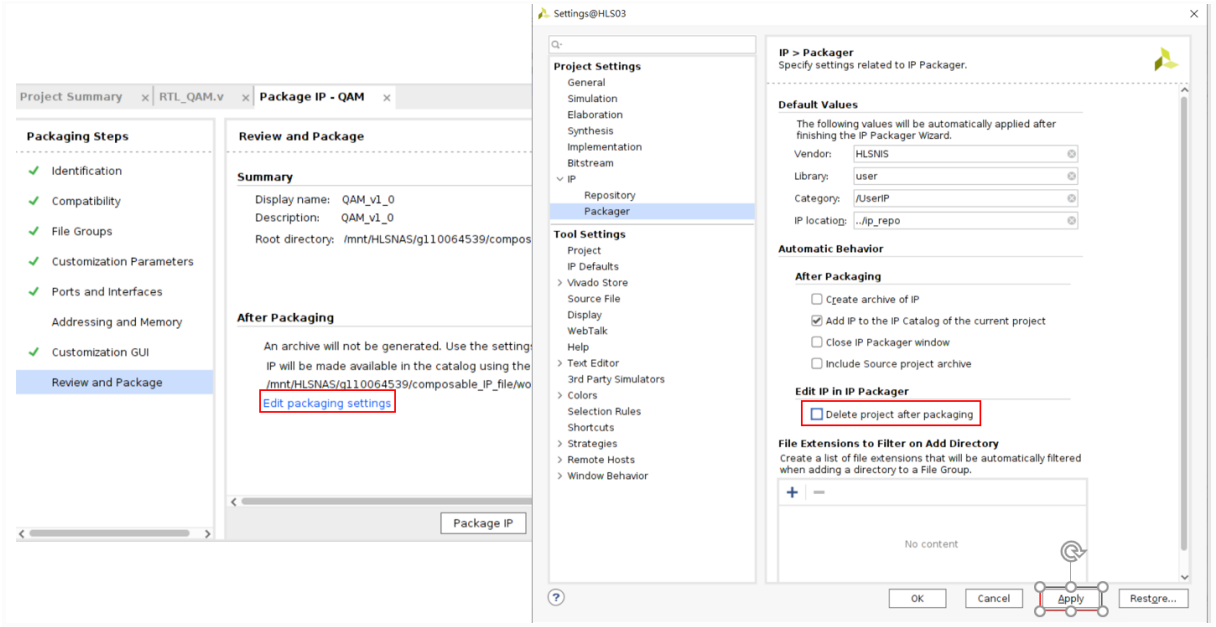
會看到剛選的ASSOCIATED_BUSIF跑到右邊，在Value那欄中填寫剛剛包好的ports name，因此填寫"m_axis:s_axis”，完成後按"OK"

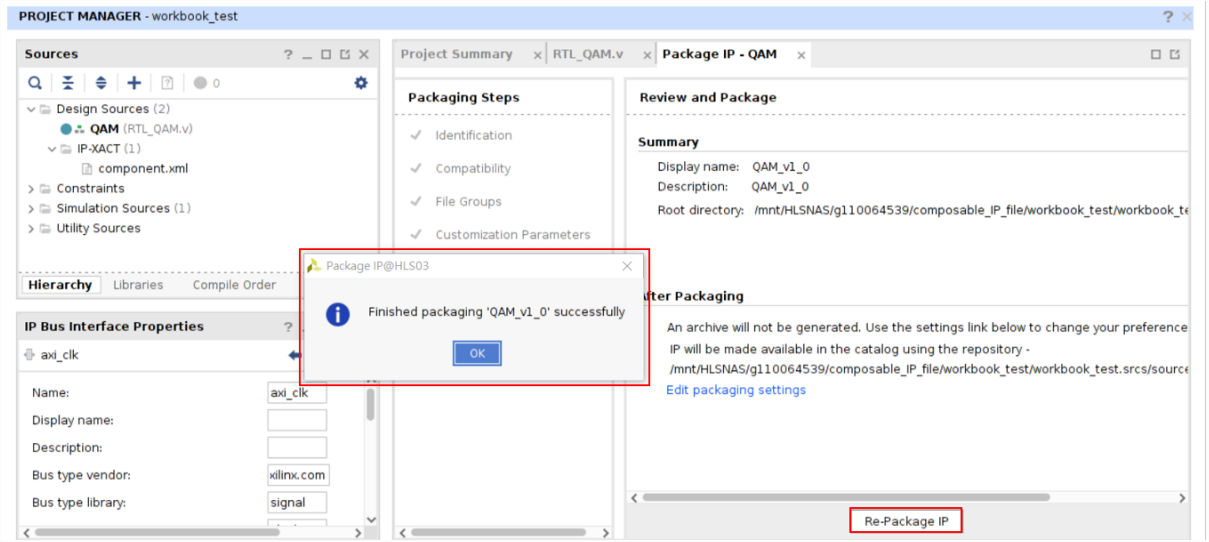


完成後可以看Customization GUI圖:



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。





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

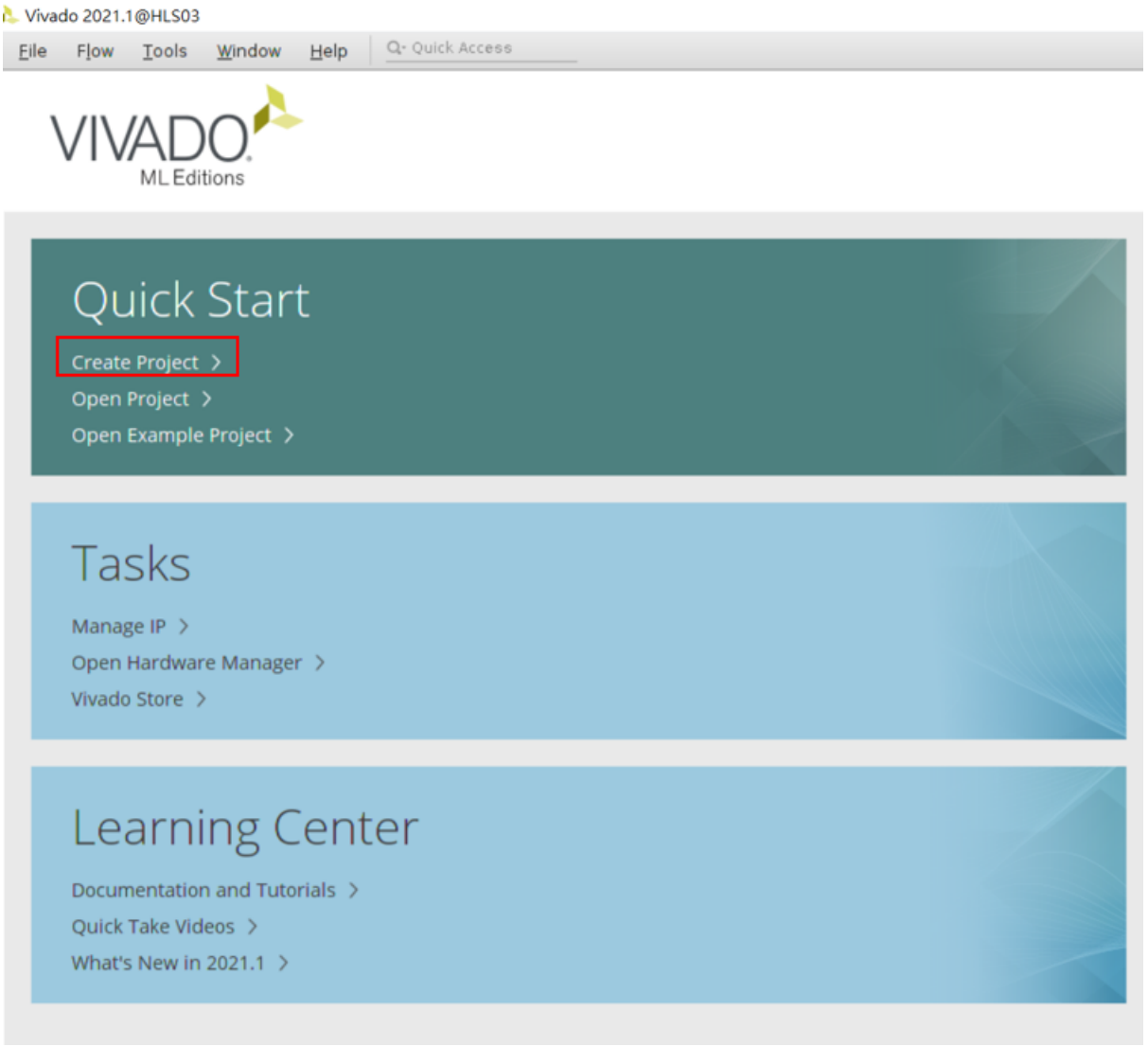
> Composable Pipeline Connection in Vivado

。Project Setting

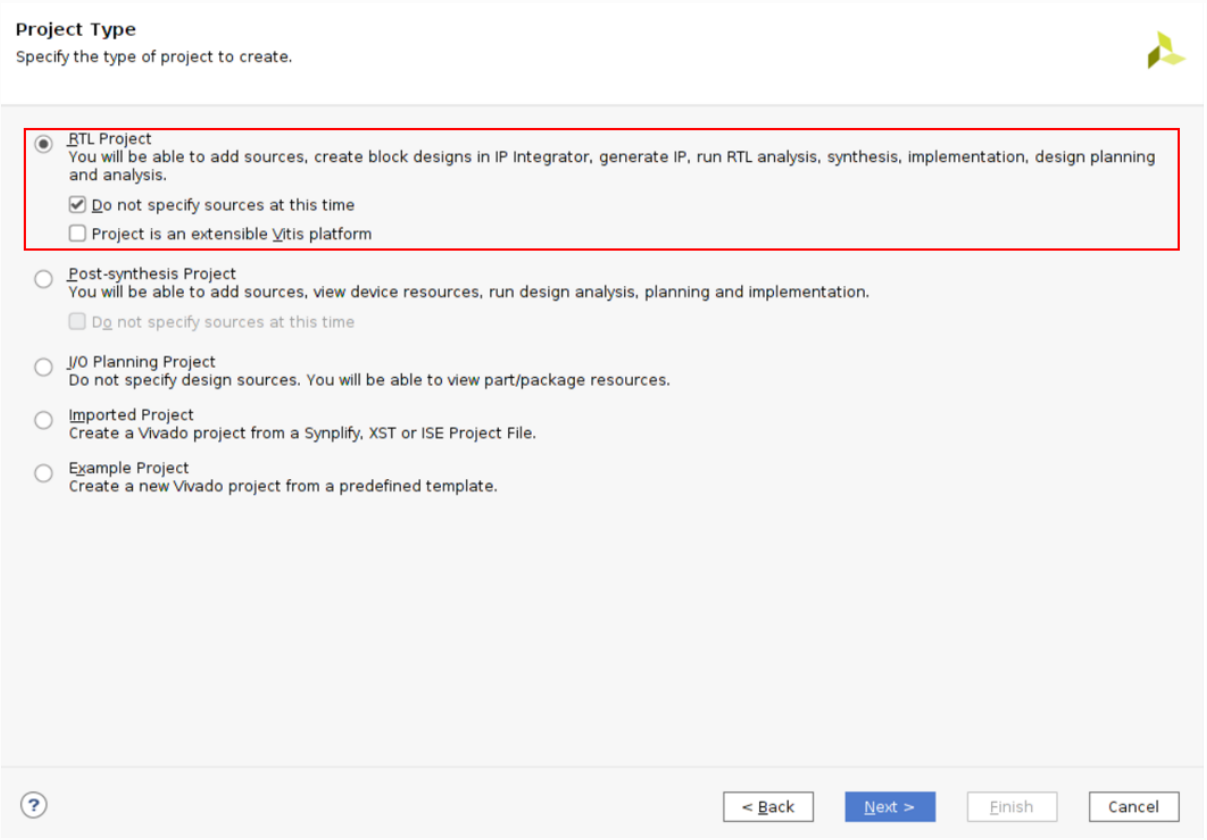
。**Create vivado project**

啟動vivado開發套件

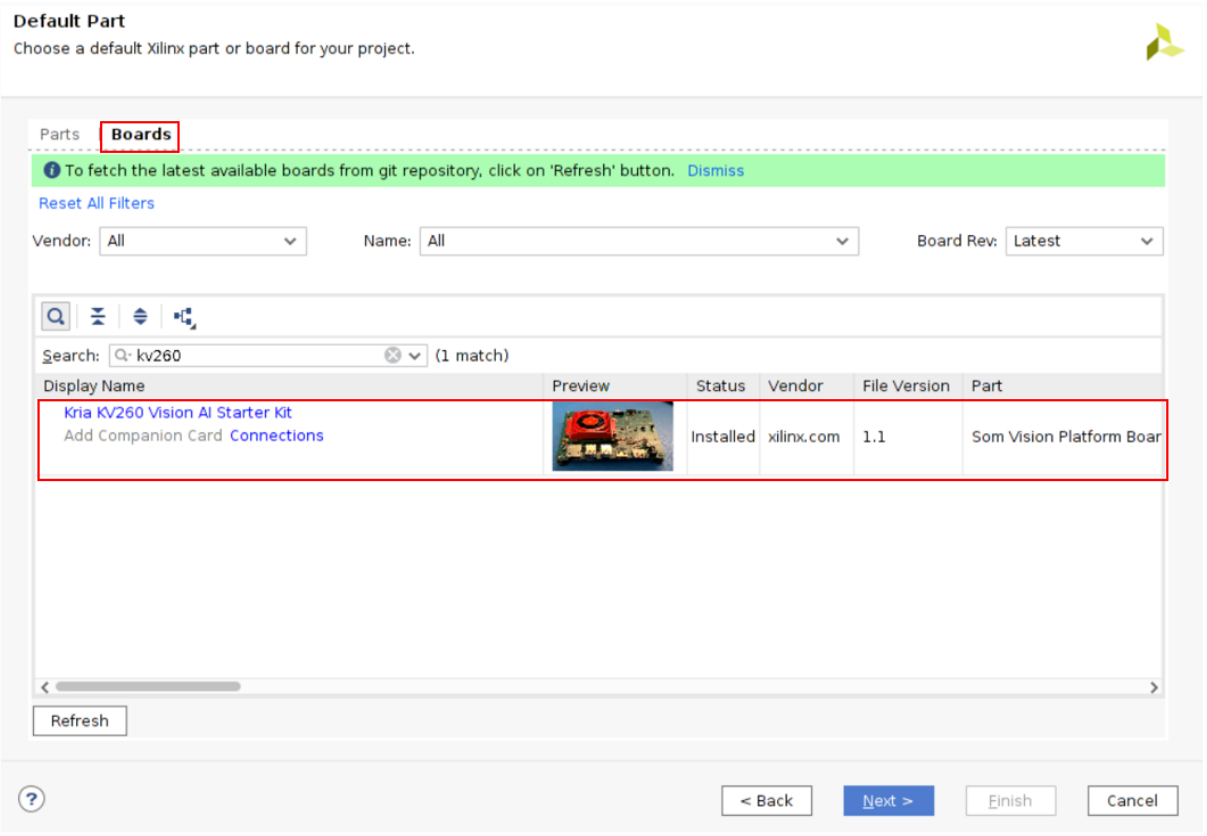
開啟新的專案，設定好專案存放的路徑:



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



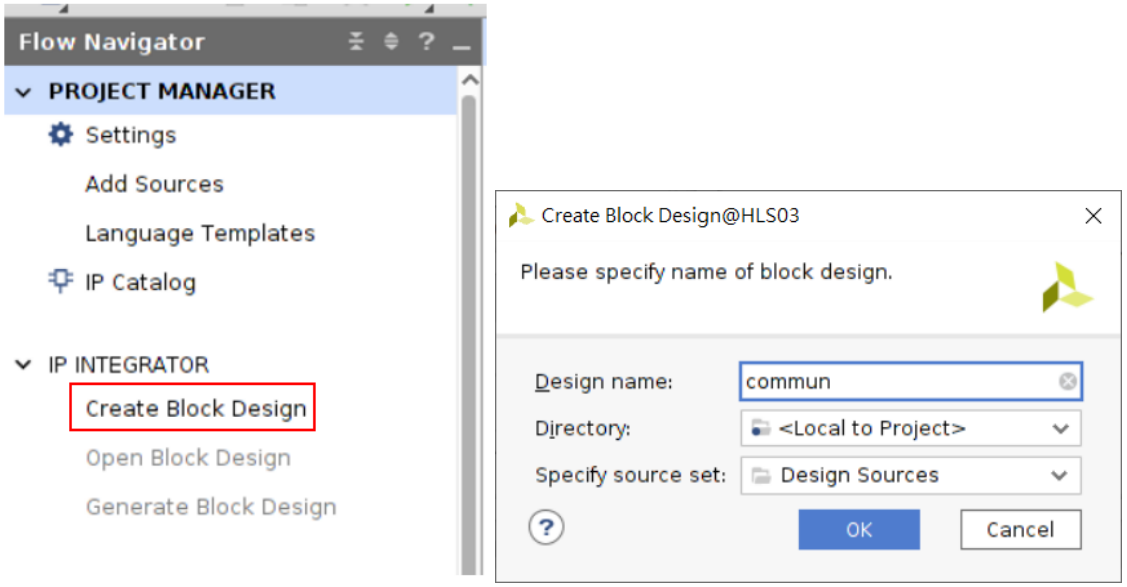
選擇Borads，並在下方選 Kria KV260 Vision AI Started Kit，點選Next>:



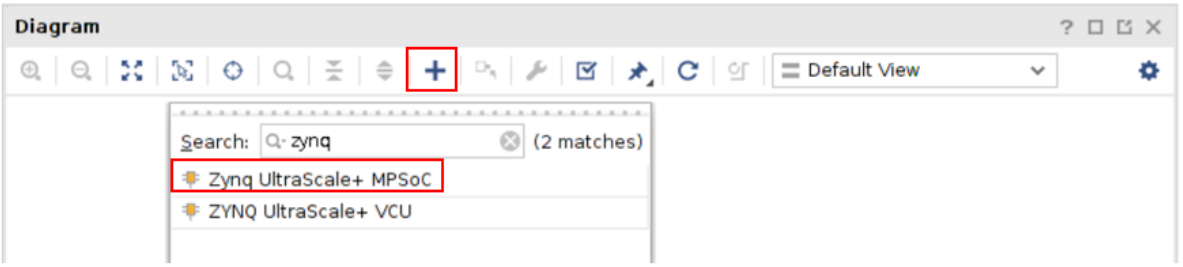
最後點選Finish，完成Create Project。

。Design Block Design

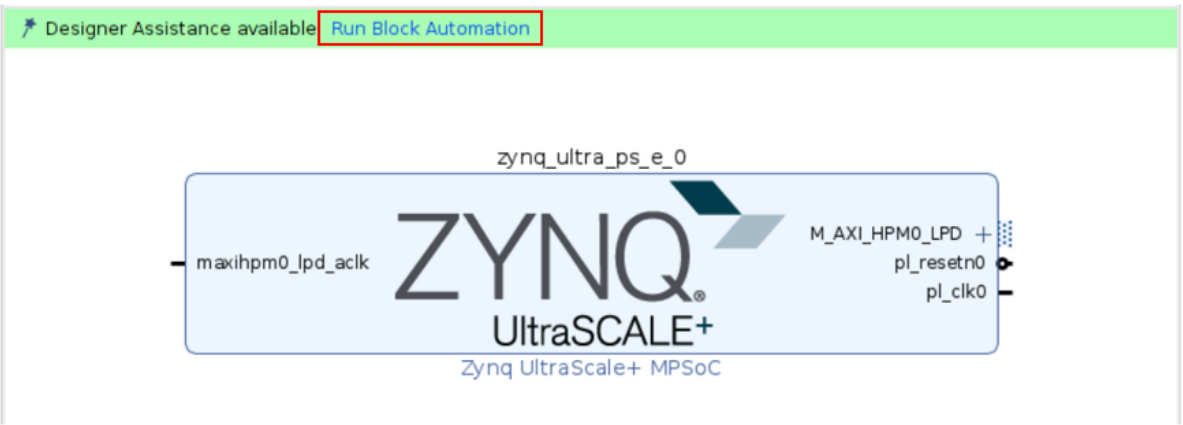
選擇Create Block Design，設定block design名字

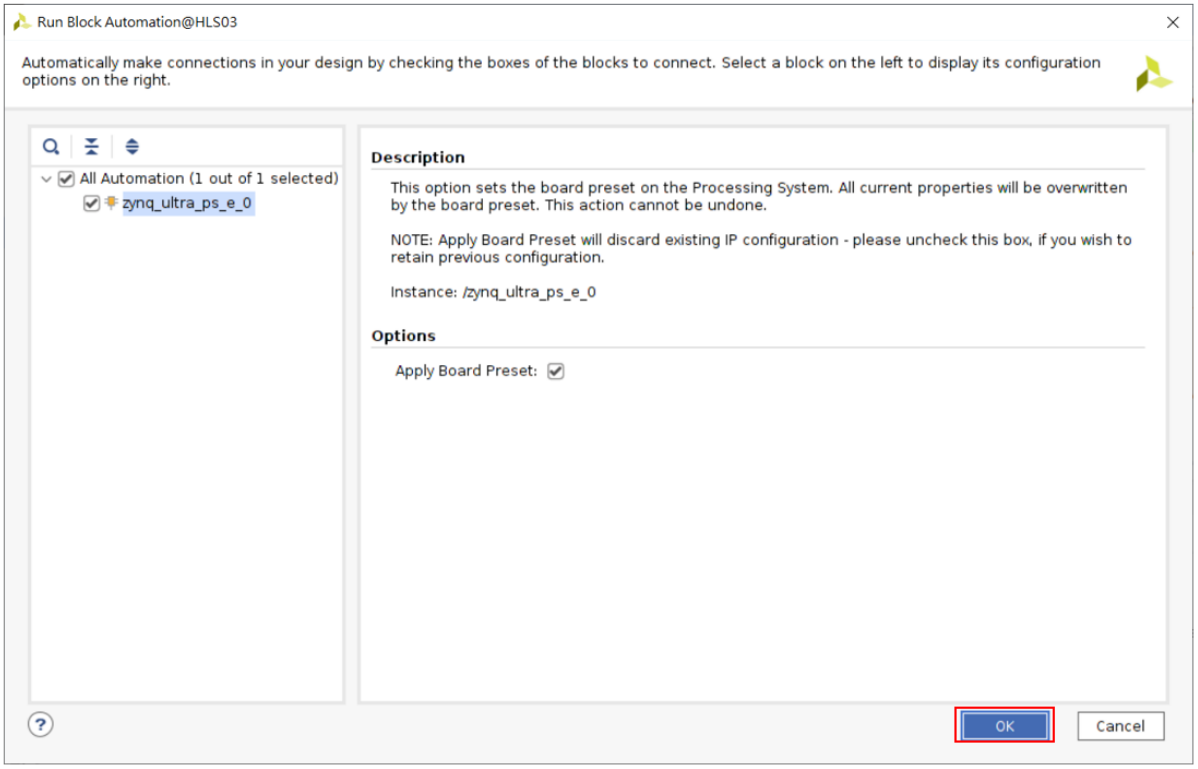


在Diagram的工具列中點選 ＋ 來新增IP，搜尋到Zynq UltraScale+ MPSoC並左鍵點選兩下

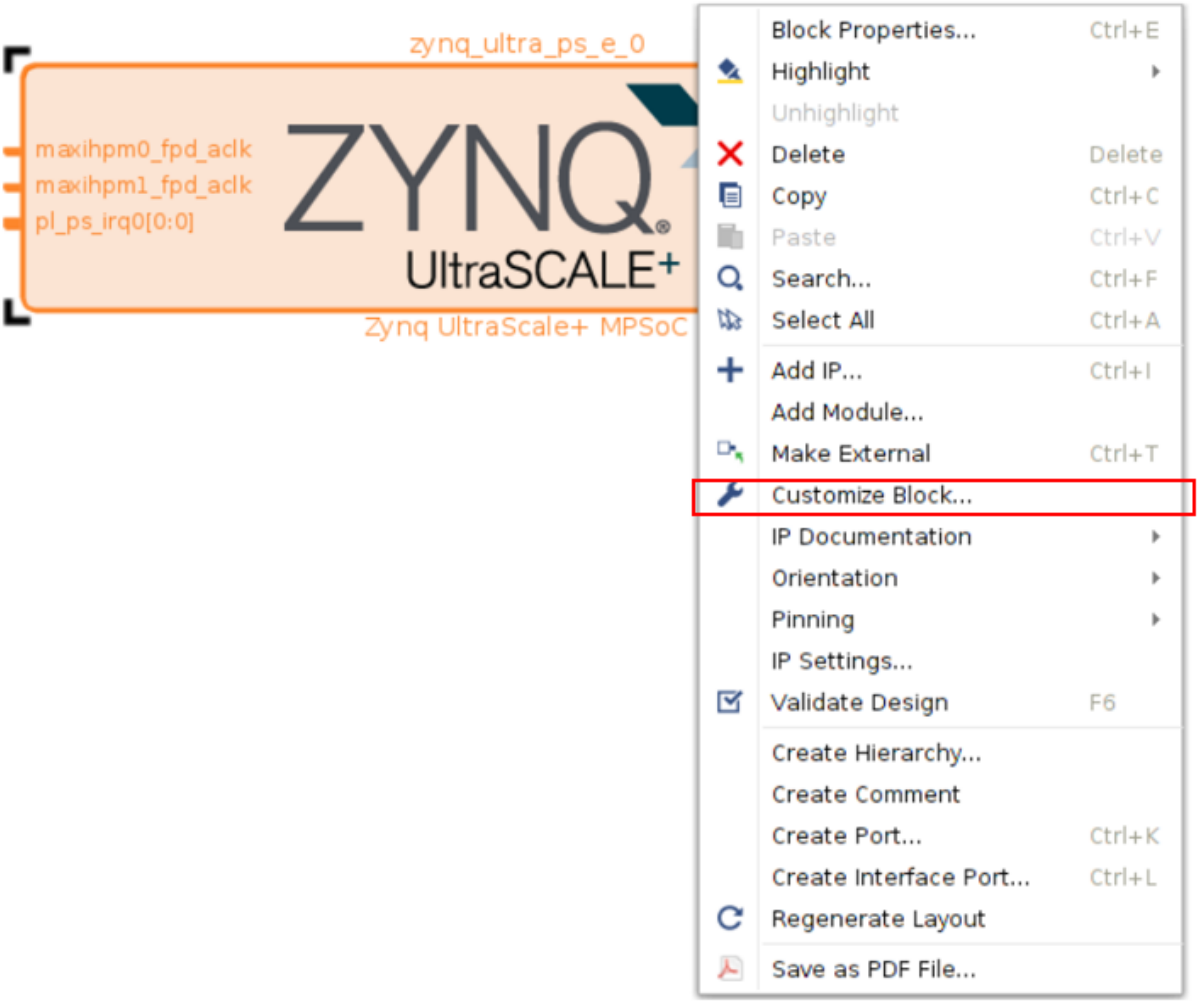


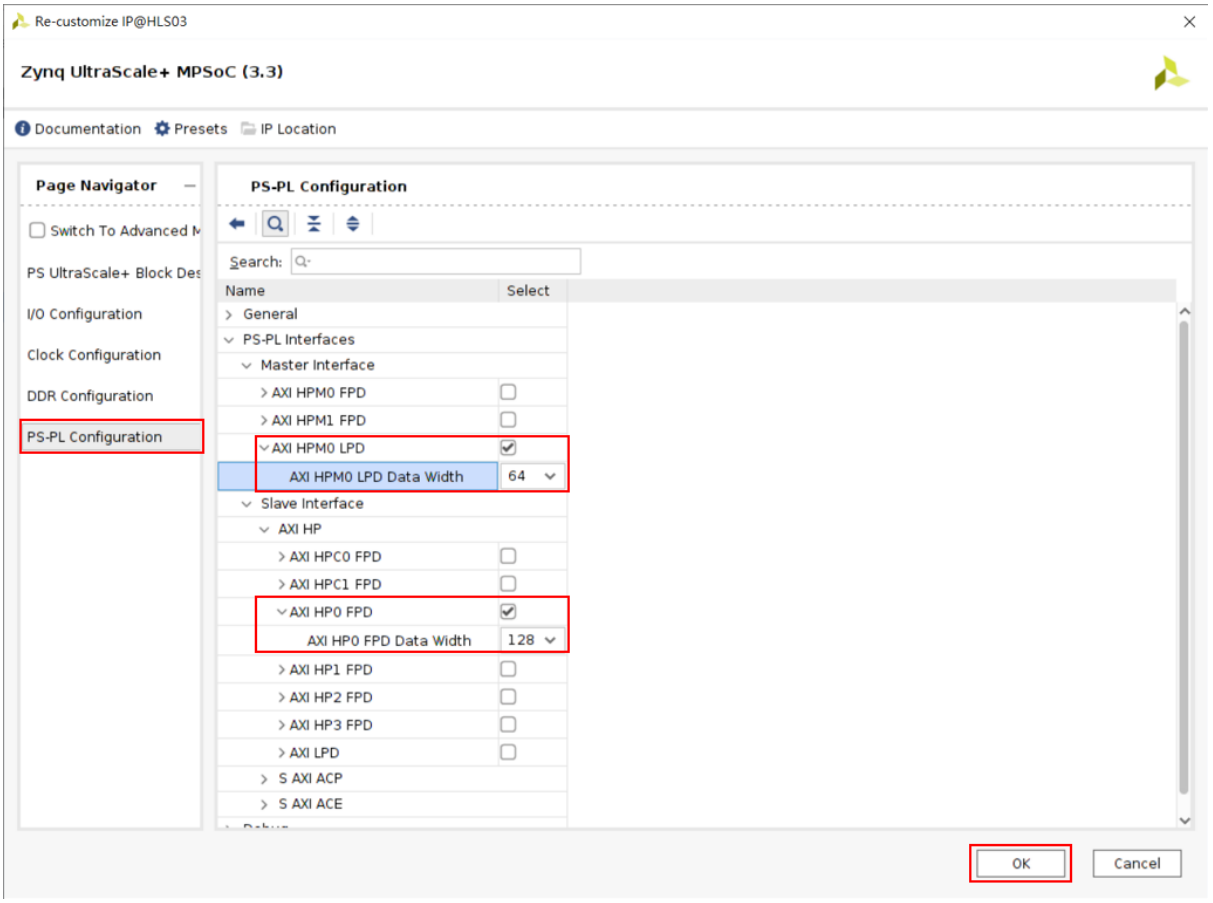
會跑出Zynq UltraScale+ MPSoC，點選Run Block Automation，跳出視窗按OK



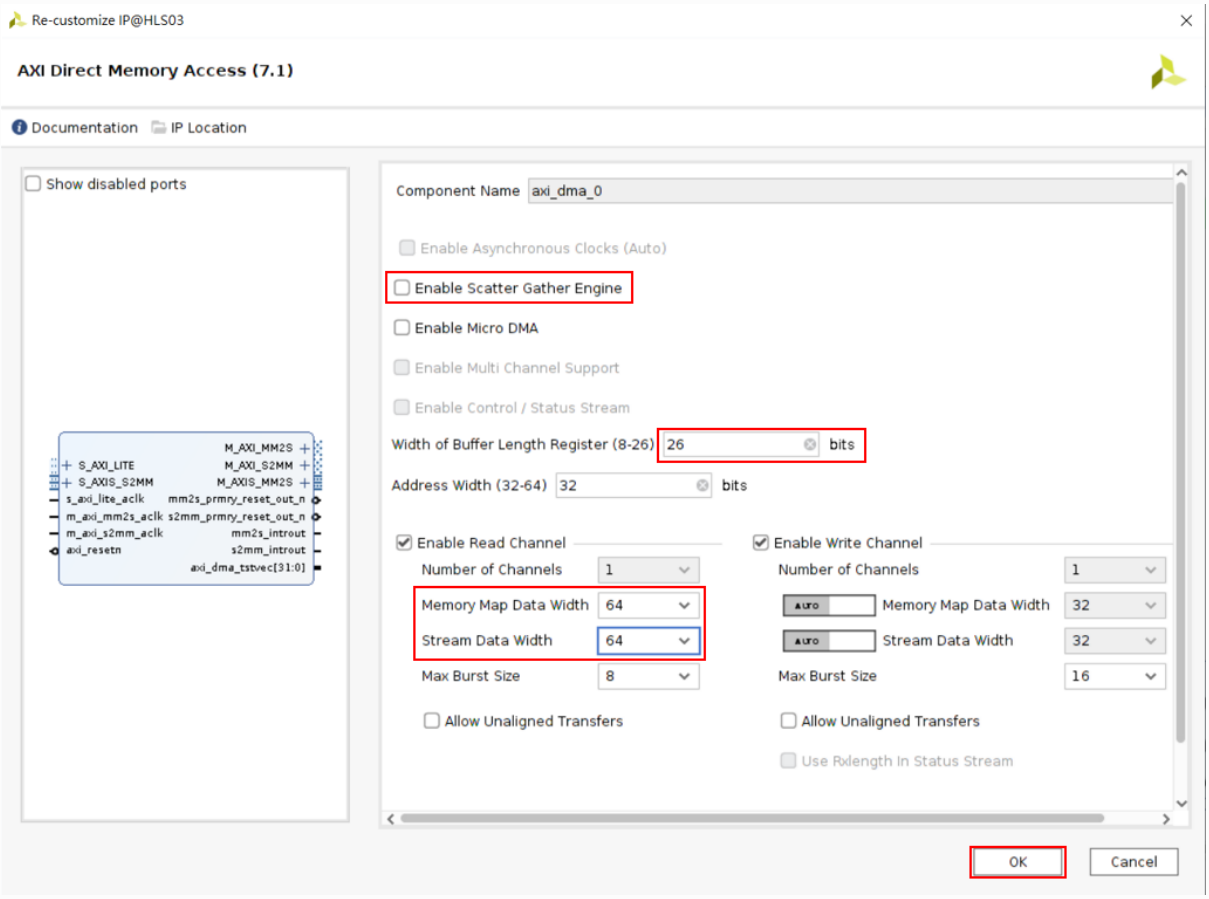


對Zynq UltraScale+ MPSoC按右鍵，選擇Customize Block…，在跳出視窗的左邊欄位選PS-PL Configuration，展開PS-PL interface，設定好Master interface及Slave interface如下圖，其中AXI HPM0 LPD Data Width要設定為64，因為我們設計中加了parameter stream後的規格是64 bits。

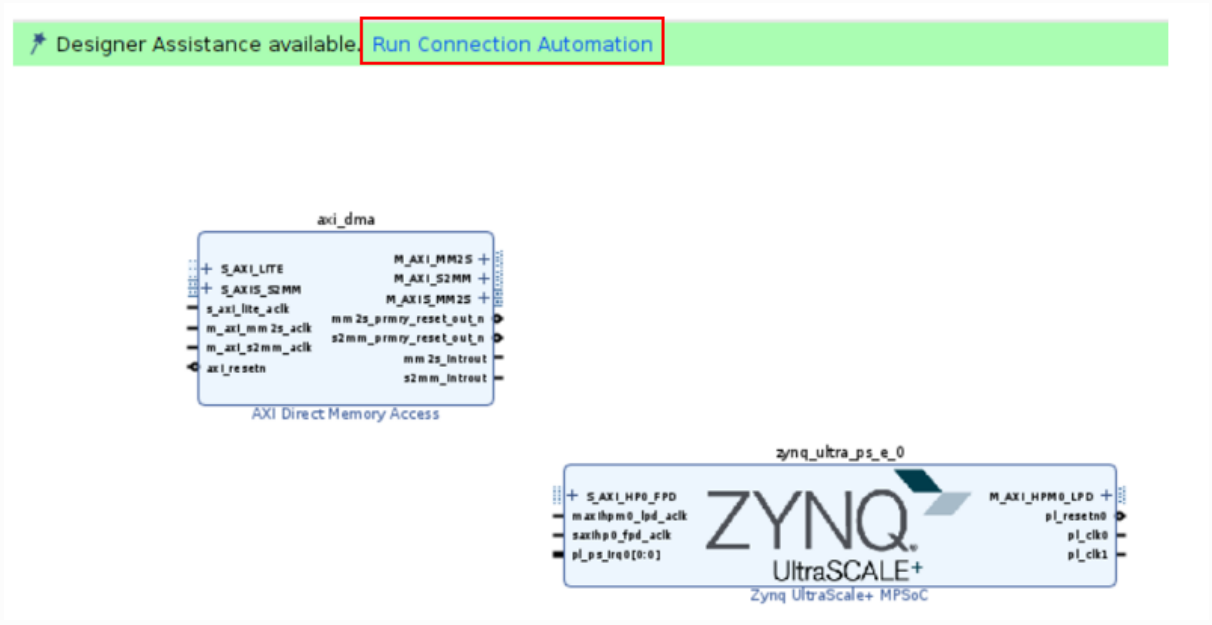

zynq_ultra_ps_e_0
maxihpm0_fpd_aclk
maxihpm1_fpd_aclk
pl_ps_irq0[0:0]
ZYNQ
UltraSCALE+
Zynq UltraScale+ MPSoC
Block Properties... Ctrl+E
Highlight
Unhighlight
Delete Delete
Copy Ctrl+C
Paste Ctrl+V
Search... Ctrl+F
Select All Ctrl+A
Add IP... Ctrl+I
Add Module...
Make External Ctrl+T
Customize Block...
IP Documentation
Orientation
Pinning
IP Settings...
Validate Design F6
Create Hierarchy...
Create Comment
Create Port... Ctrl+K
Create Interface Port... Ctrl+L
Regenerate Layout
Save as PDF File...

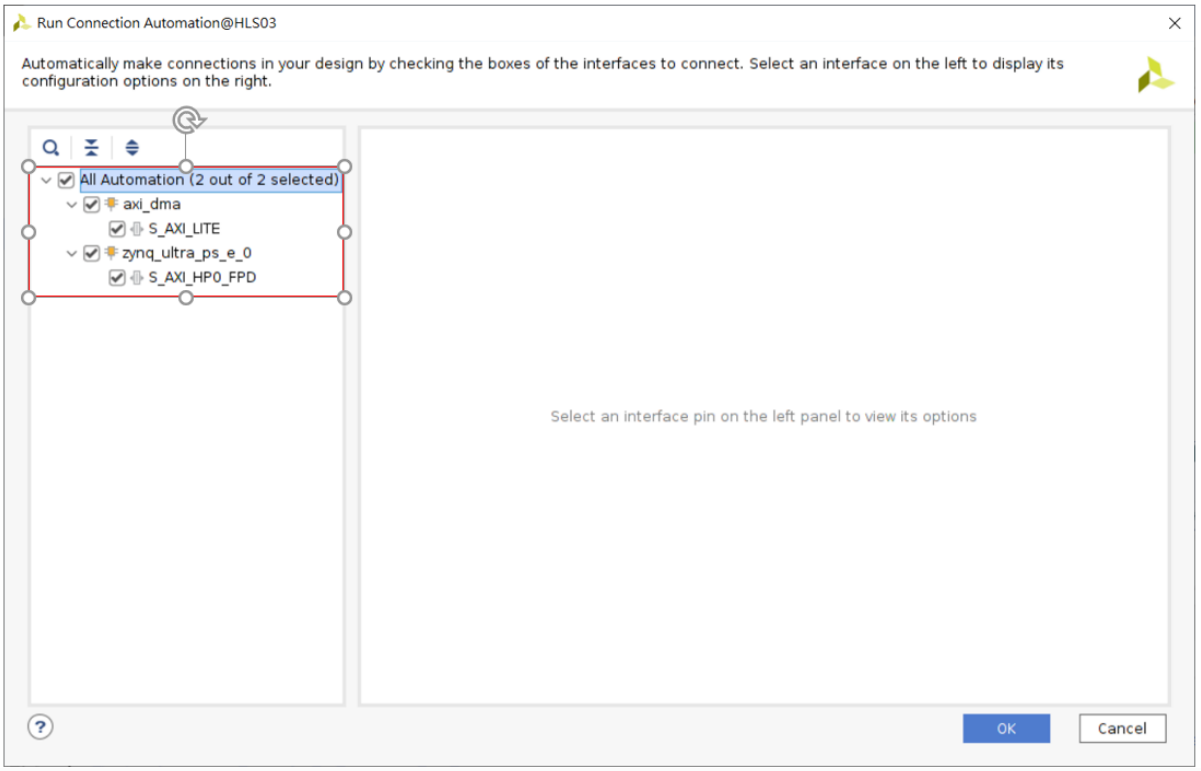


在Diagram的工具列中點選 ➕ 來新增IP，搜尋到AXI Direct Memory Access並左鍵點選兩下，再對生成出來的AXI Direct Memory Access左鍵點兩下進入設定，Enable Scatter Gather Engine取消打勾，Width of Buffer Length Register設為26，在Read channel中把Memory data Width和Stream data Width都設為64 bits，按OK，最後將IP Block 名字改為axi_dma。

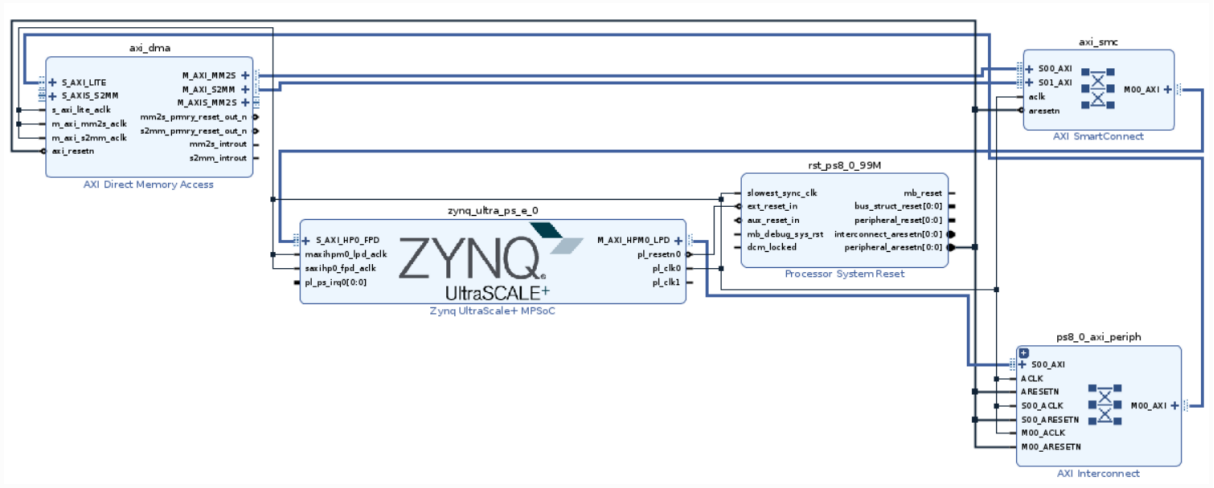


點選Run Block Automation，跳出視窗勾選All automation按OK，重複做兩次。


Designer Assistance available.
Run Connection Automation
axi_dma
S_AXI_LITE
S_AXIS_S2MM
s_axi_lite_aclk
m_axi_mm2s_aclk
m_axi_s2mm_aclk
axi_resetn
M_AXI_MM2S
M_AXI_S2MM
M_AXIS_MM2S
mm2s_prmry_reset_out_n
s2mm_prmry_reset_out_n
mm2s_introut
s2mm_introut
AXI Direct Memory Access
zynq_ultra_ps_e_0
S_AXI_HP0_FPD
maxihpm0_lpd_aclk
saxihp0_fpd_aclk
pl_ps_irq0[0:0]
ZYNQ
UltraSCALE+
M_AXI_HPM0_LPD
pl_resetn0
pl_clk0
pl_clk1
Zynq UltraScale+ MPSoC

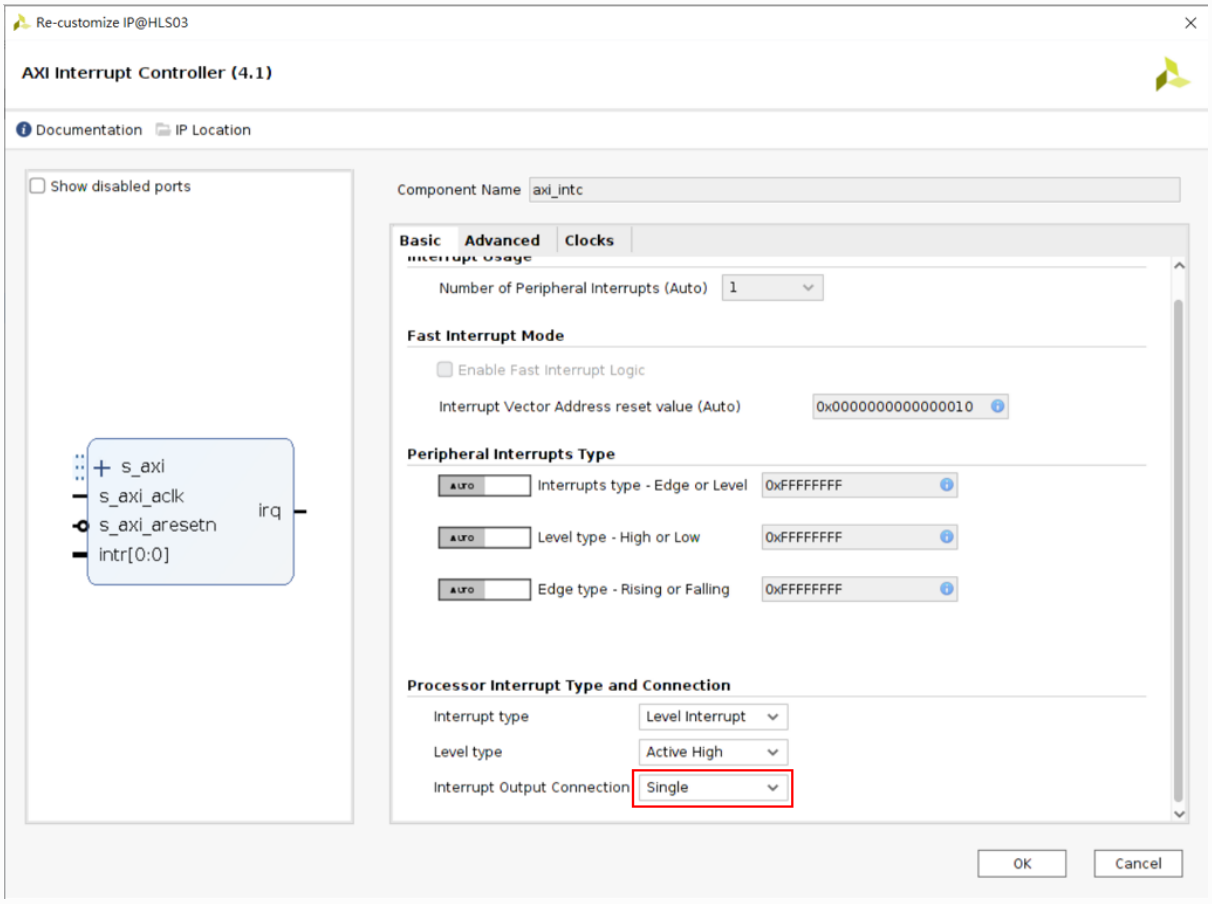

Run Connection Automation@HLS03
Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.
All Automation (2 out of 2 selected)
axi_dma
S_AXI_LITE
zynq_ultra_ps_e_0
S_AXI_HP0_FPD
Select an interface pin on the left panel to view its options
OK
Cancel

最後會會是下圖所示

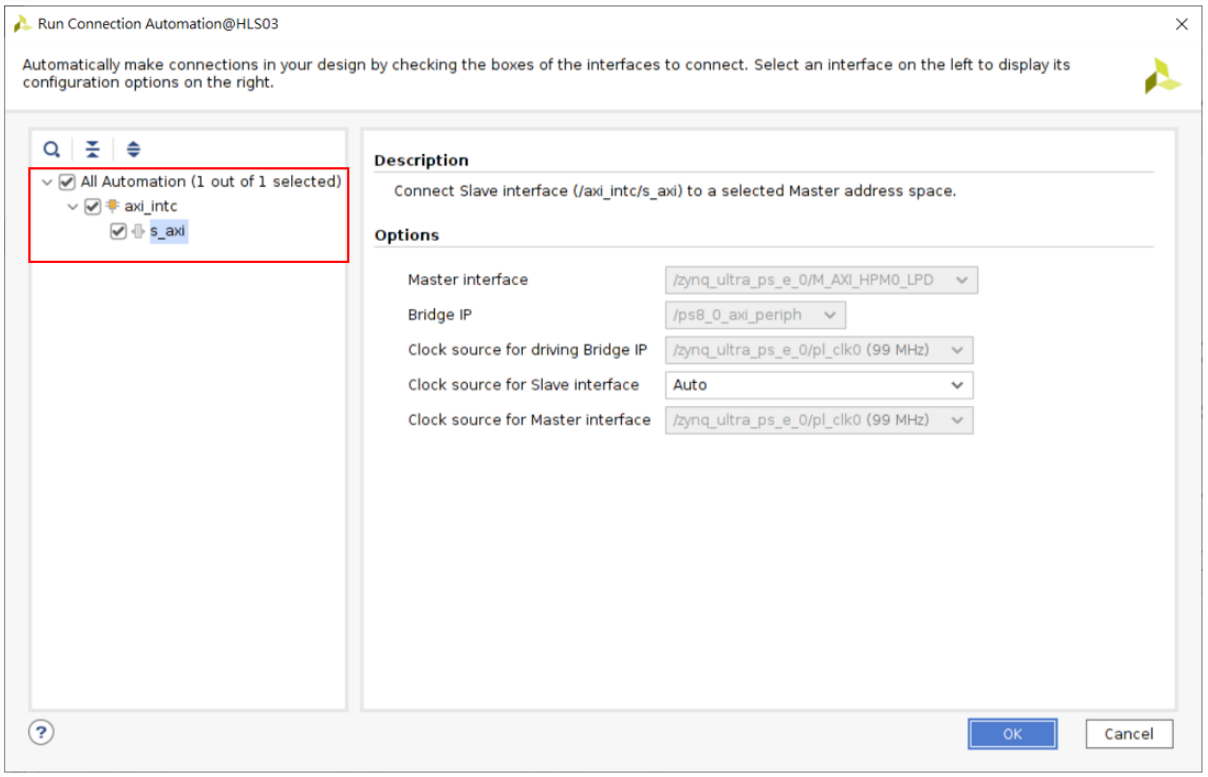


在Diagram的工具列中點選 + 來新增IP，搜尋到AXI Interrupt Controller並左鍵點選兩下將IP block加入。

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。

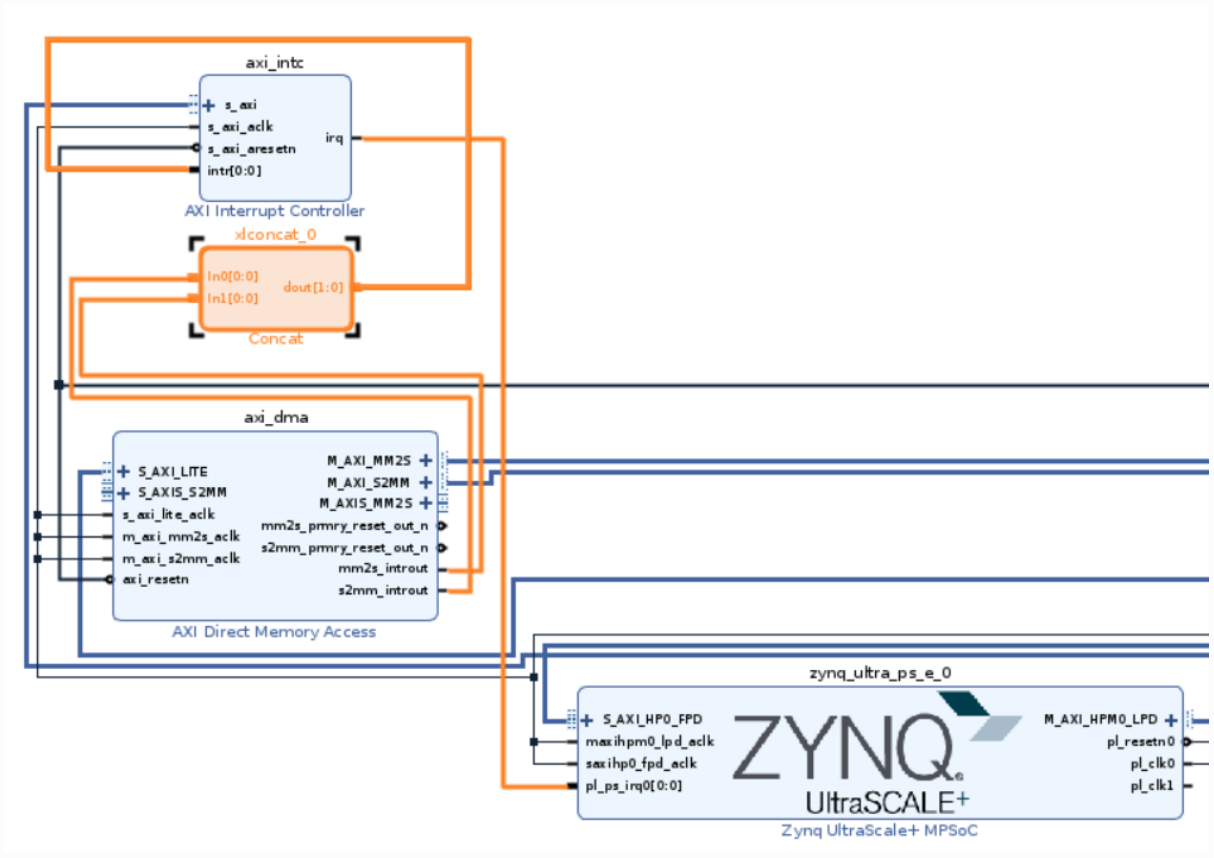


點選Run Block Automation，跳出視窗勾選All automation按OK



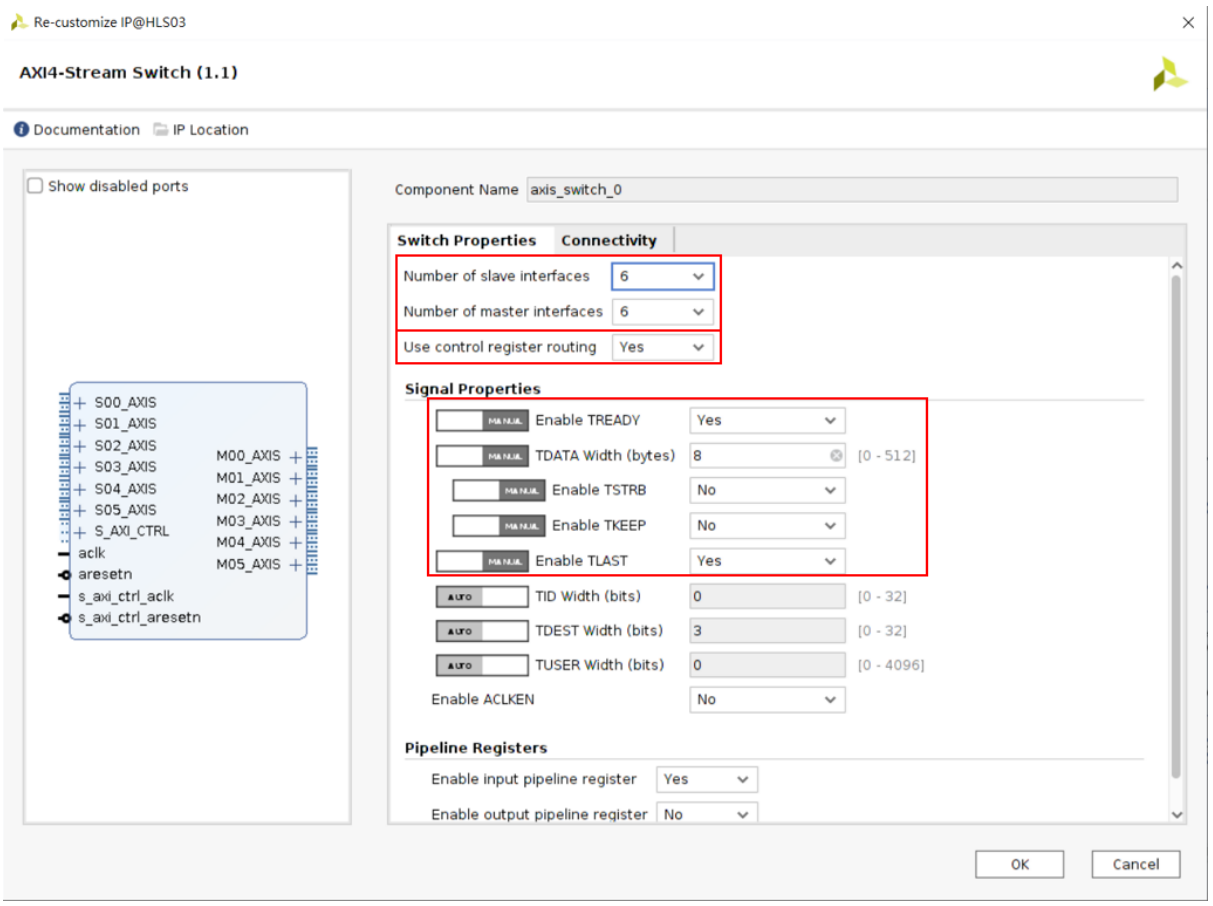
手動將axi_intc的irq pin角與zynq_ultra_ps_e_0的pl_ps_irq0連接，再新增Concat的IP

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**

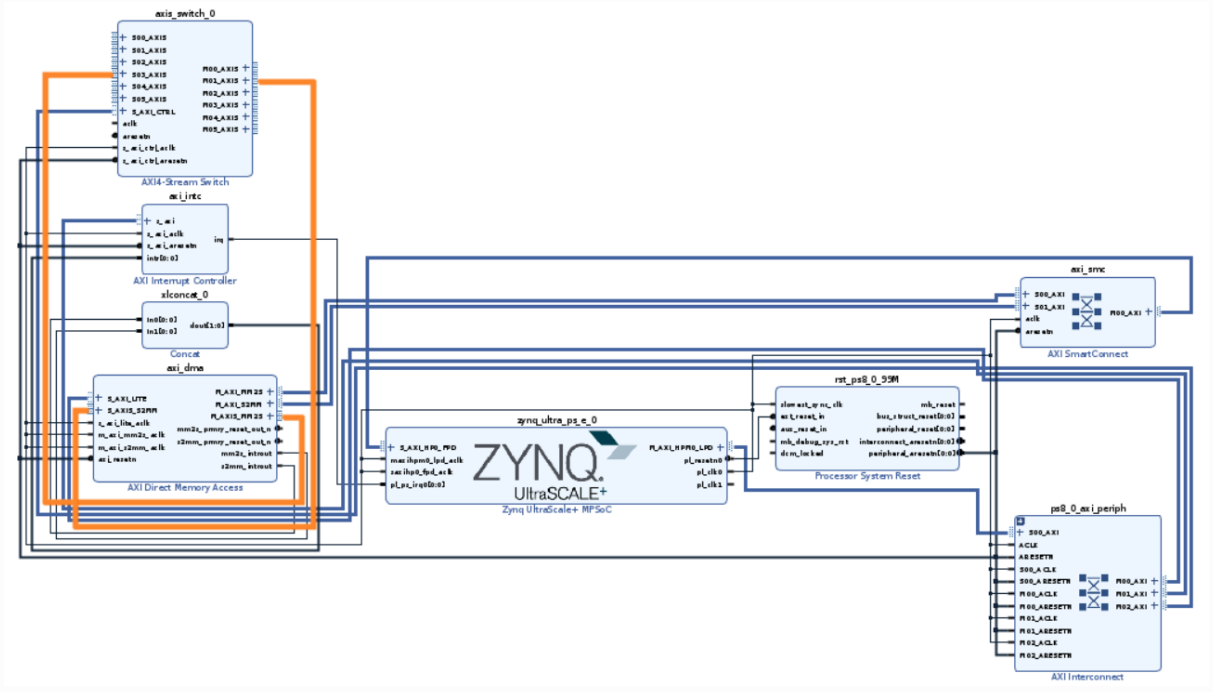
在Diagram的工具列中點選 ＋ 來新增IP，搜尋到AXI4-Stream Switch並左鍵點選兩下將IP block加入，對加入的IP block左鍵點兩下進入設定。
Number of slave interface及Number of master interface根據需要的控制的composable pipeline的個數決定，這邊先設定為6，Use control register routing設Yes，TEADY、TSTRB、TKEEP、TLAST設定為下圖所示。由於我們訊號是64 bites，所以TDATA設為8 bytes。按OK。



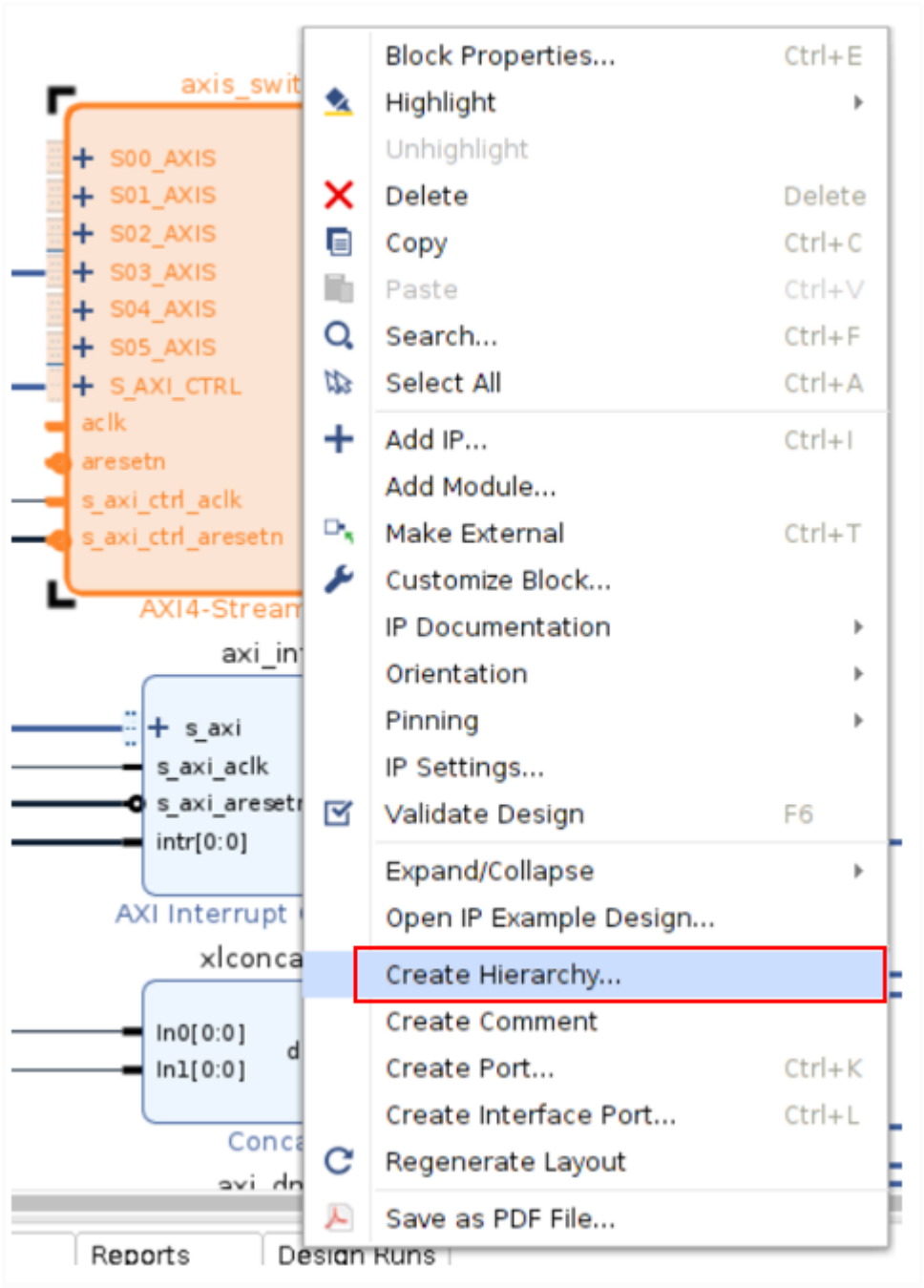
將axi_dma的M_AXIS_MM2S接到axis_switch_0的S03_AXIS，axis_switch_0的M01_AXIS接到axi_dma的S_AXIS_S2MM。

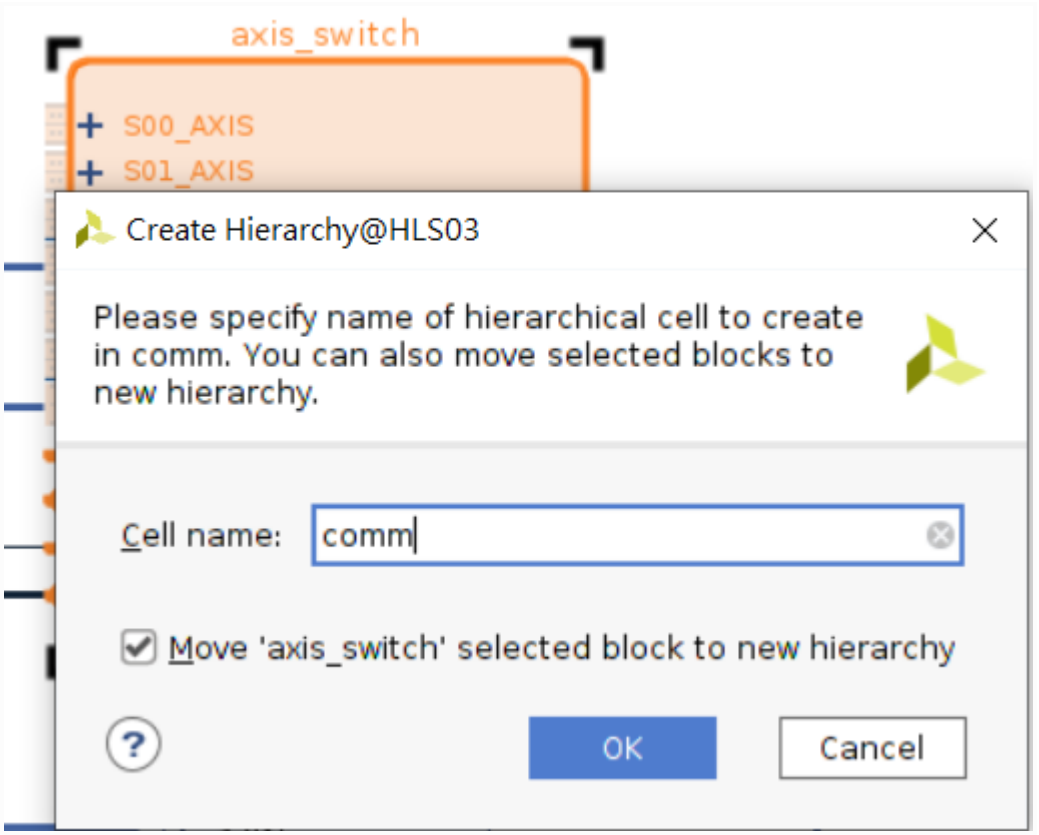
點選Run Block Automation，跳出視窗勾選All automation按OK。

將axis_switch_0名字改為axis_switch

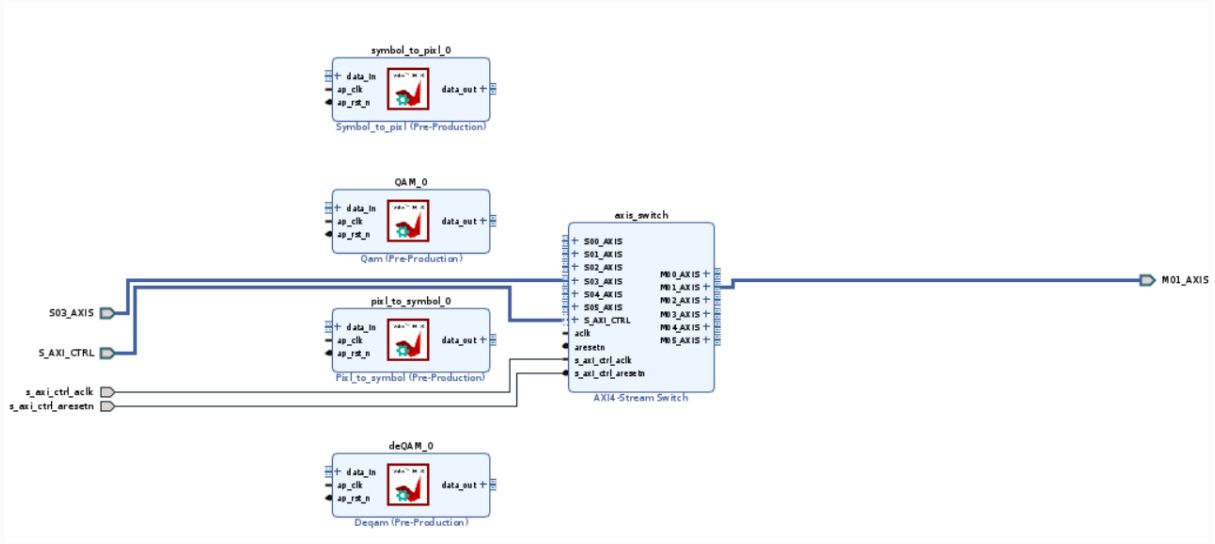


對axis_switch_0按右鍵，點選Create Hierarchy…
Cell name設為comm，按OK

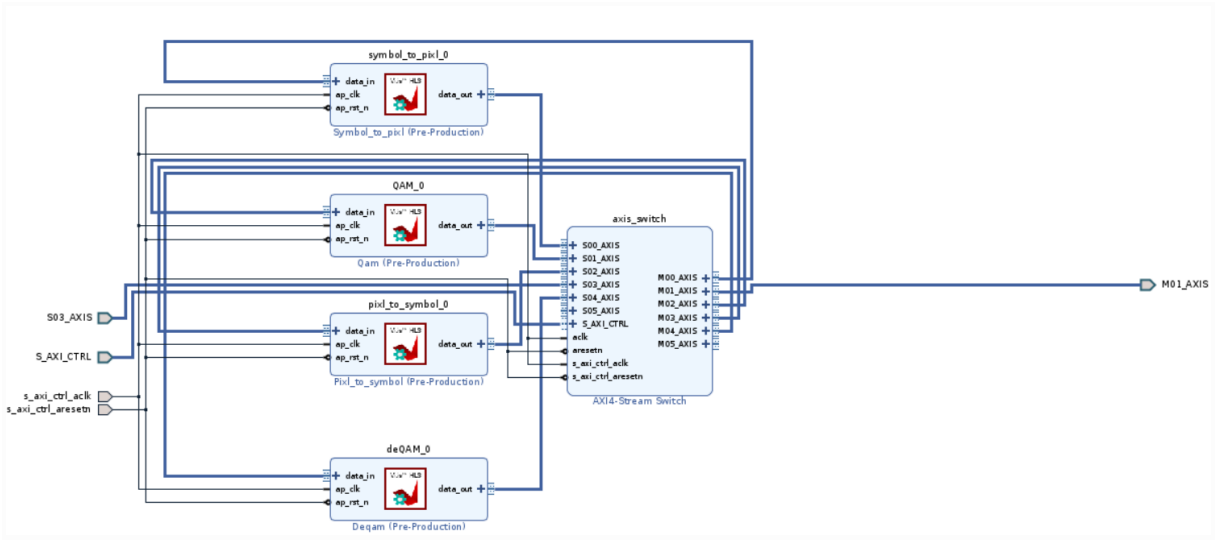

axis_swit
S00_AXIS
S01_AXIS
S02_AXIS
S03_AXIS
S04_AXIS
S05_AXIS
S_AXI_CTRL
aclk
aresetn
s_axi_ctrl_aclk
s_axi_ctrl_aresetn
AXI4-Stream
axi_int
s_axi
s_axi_aclk
s_axi_aresetn
intr[0:0]
AXI Interrupt
xlconca
In0[0:0]
In1[0:0]
Conca
axi_dp
Reports
Design Runs
Block Properties... Ctrl+E
Highlight
Unhighlight
Delete Delete
Copy Ctrl+C
Paste Ctrl+V
Search... Ctrl+F
Select All Ctrl+A
Add IP... Ctrl+I
Add Module...
Make External Ctrl+T
Customize Block...
IP Documentation
Orientation
Pinning
IP Settings...
Validate Design F6
Expand/Collapse
Open IP Example Design...
Create Hierarchy...
Create Comment
Create Port... Ctrl+K
Create Interface Port... Ctrl+L
Regenerate Layout
Save as PDF File...



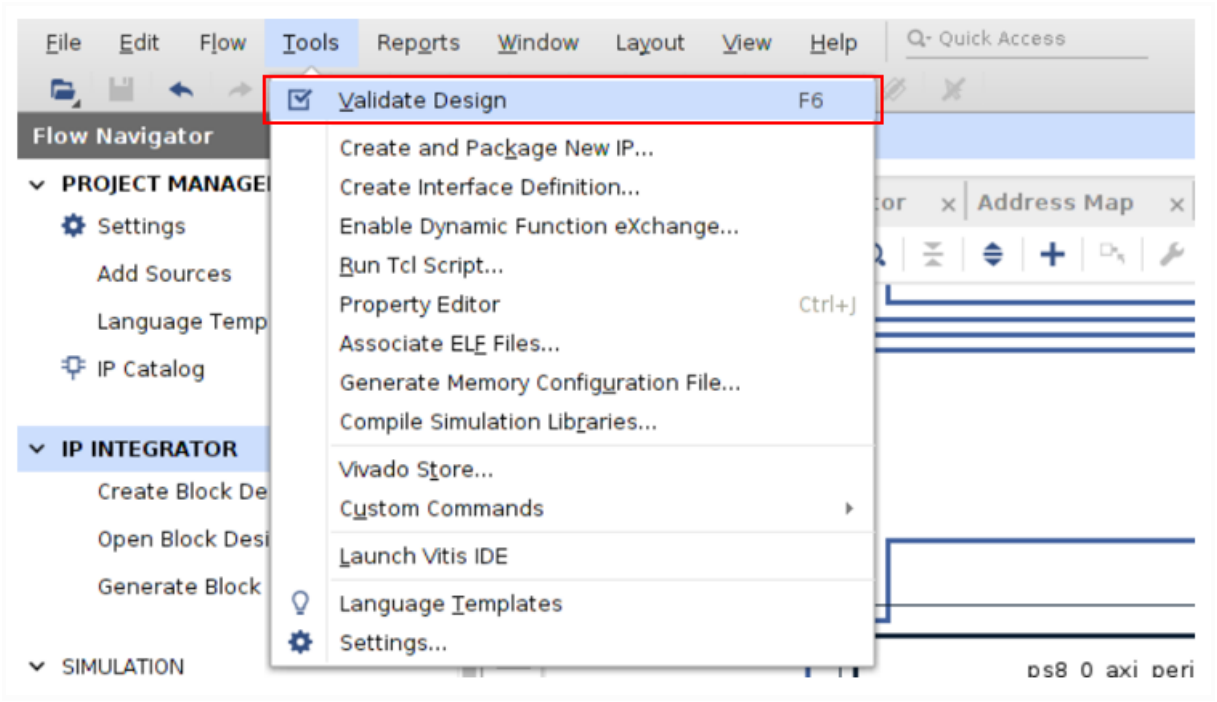
再對comm左鍵點兩下開啟，將想要加入的IP加進去



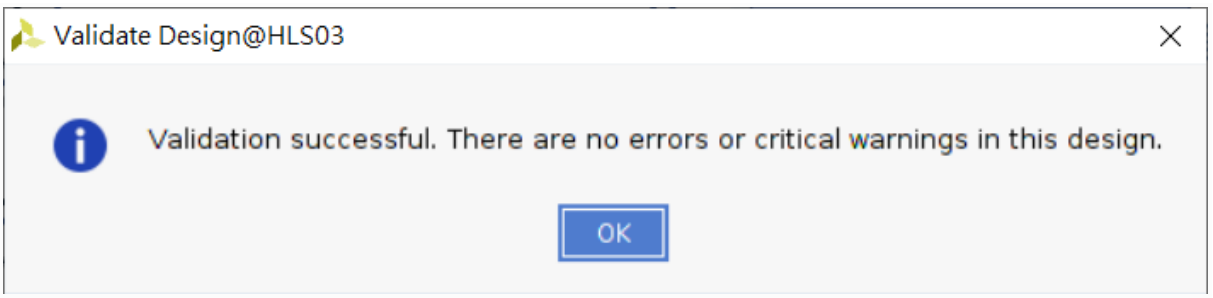
將想加入的IP output接到axi_switch的input，axi_switch的output接到IP 的input，點選Run Block Automation，跳出視窗勾選All automation按OK。



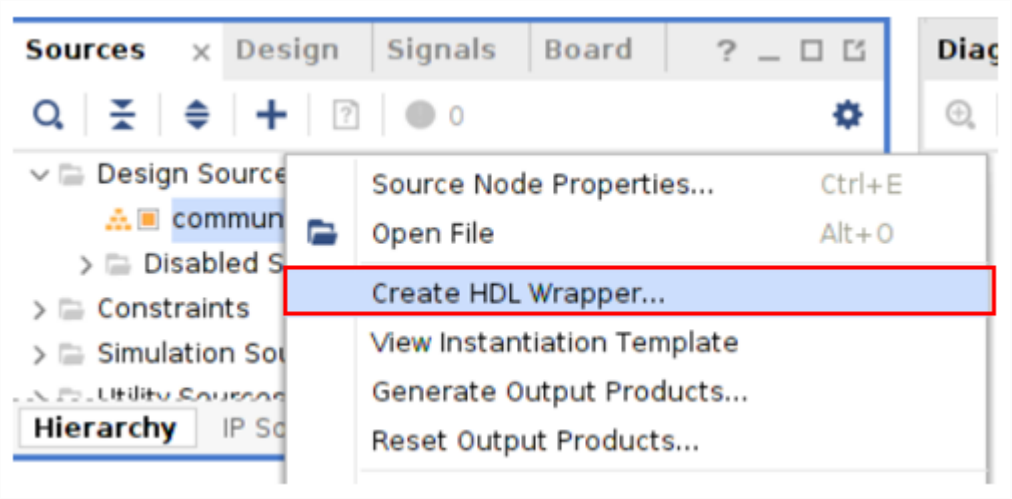
按CTRL+S儲存Block Design,
在Tools中點選Validate Design,

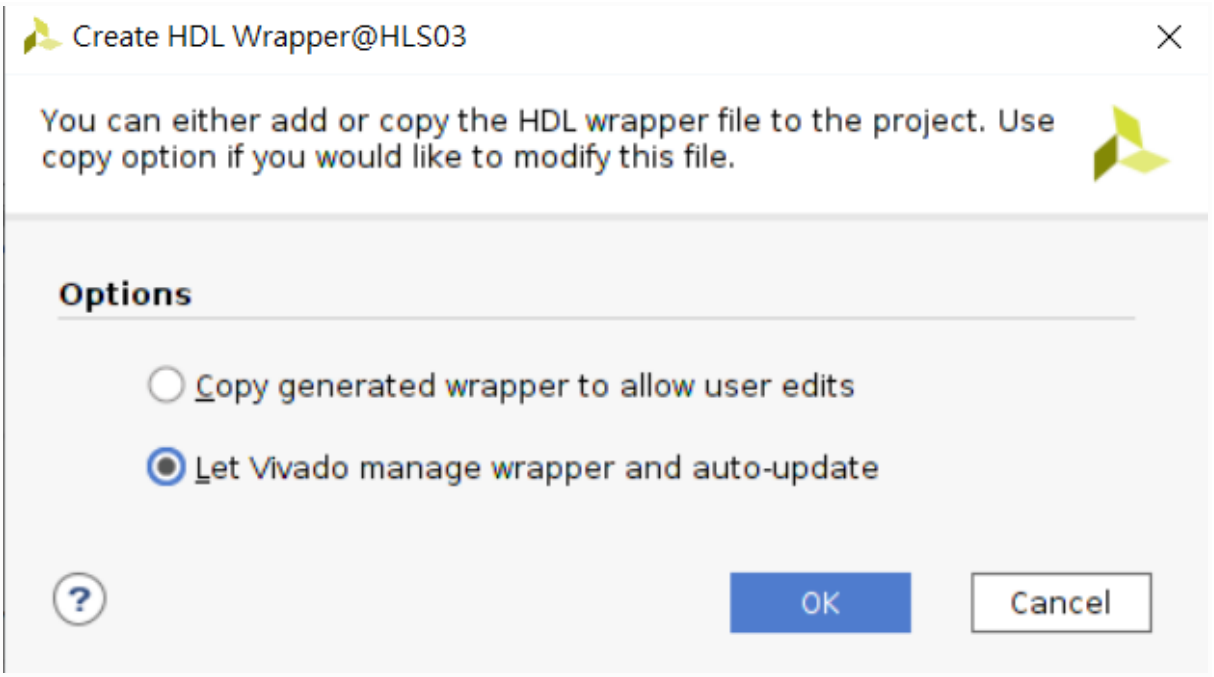


若跳出Validdation successful，表示沒有問題。

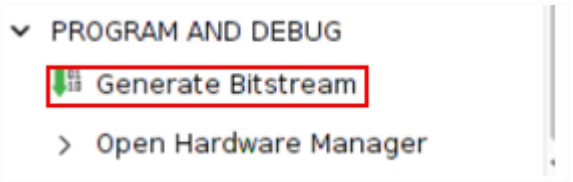


在Design sources的commun按右鍵選Create HDL Wrapper…，跳出視窗按OK。





最後可以點選Generate Bitstream



。**Jupyter Notebook Host Code**
將commun.bat、commun.hwh、commun_path.json及communcation.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)。

```
{
    "comm": {
        "ps": {
            "ci": {
                        "port": 3,
                        "Description": "Stream In"
                },
            "pi": {
                        "port": 1,
                        "Description": "Stream Out"
            }
        }
    }
}
```

communcation.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
```

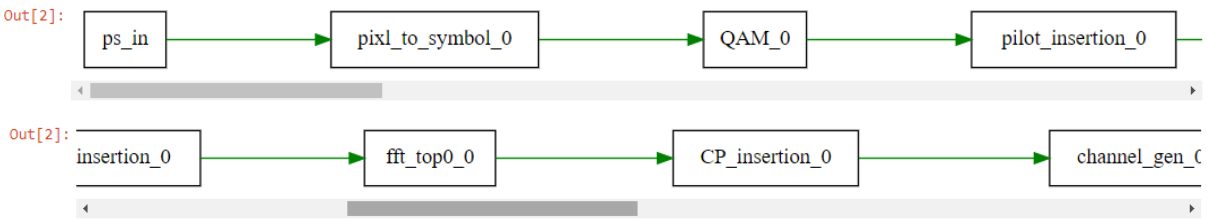
並且將想要的IP串起來，使用cfilter.compose()把想要的IP連起來，如下圖

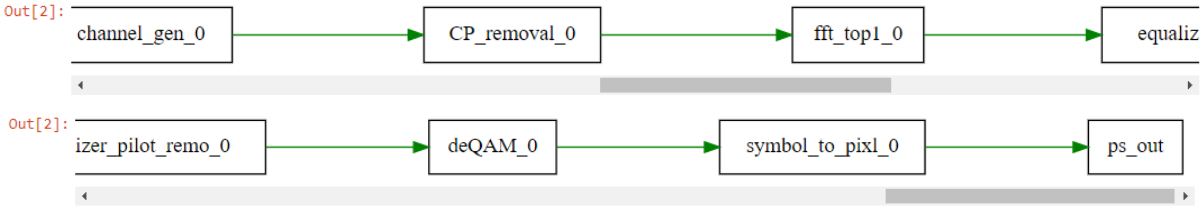
```
# channel_total => ok
cfilter.compose([cfilter.ps_in, cfilter.pixl_to_symbol_0, cfilter.QAM_0, cfilter.pilot_insertion_0, cfilter.fft_top0_0, cfilter.(
cfilter.graph
```

```
cfilter.fft_top0_0, cfilter.CP_insertion_0, cfilter.channel_gen_0, cfilter.CP_removal_0, cfilter.fft_top1_0, cfilter.equalizer_p
```

```
ter.CP_removal_0, cfilter.fft_top1_0, cfilter.equalizer_pilot_remo_0, cfilter.deQAM_0, cfilter.symbol_to_pixl_0, cfilter.ps_out])
```

回印出想串起來的IP圖，如下圖





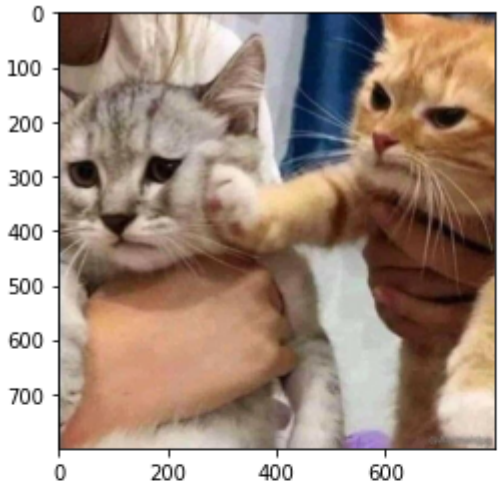
設定module_id以及parameter所需的值，
#-------ORIGIANL PIC------#
將所要傳送的data讀進來，這邊是使用cat_punch.jpg做舉例
#-------READ DATA OK------#
將各個module id所需要的parameter設定在parameter_id
#-------PARA SET------#
將要傳送的input_buffer根據parameter stream規格，把parameter_id放進的input_buffer
#-------DATA SET------#
將要傳送的input_buffer根據parameter 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 = [4,4,4,4,3,4,6,4,3,5,4,4,4] # # 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 pixl2sym
                         [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 sym2pixl
```

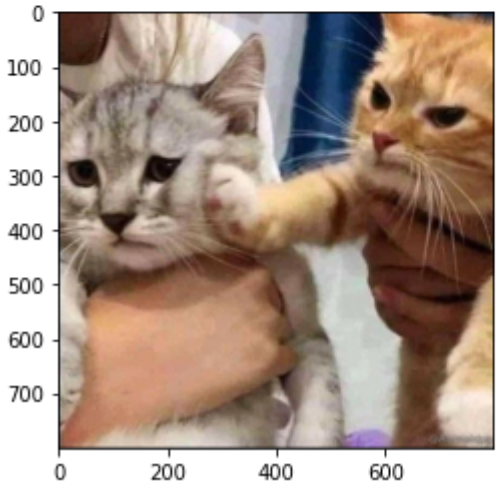
```
print("#-------------------PARA SET----------------------#")
temp = allocate(shape=(1,), dtype=np.uint64)
cnt_para_in = 0
for i in range(len_module_id):
    temp[0] = module_id[i] * pow(2,16)
    for j in range(len_parameter_id[i]):
        input_buffer[cnt_para_in+j] = temp[0]
        input_buffer[cnt_para_in+j]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




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

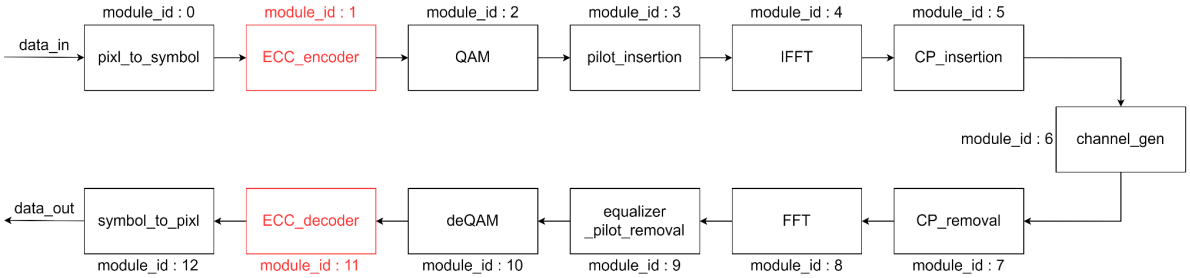



```
#-----------------PROCESS END---------------------#
```

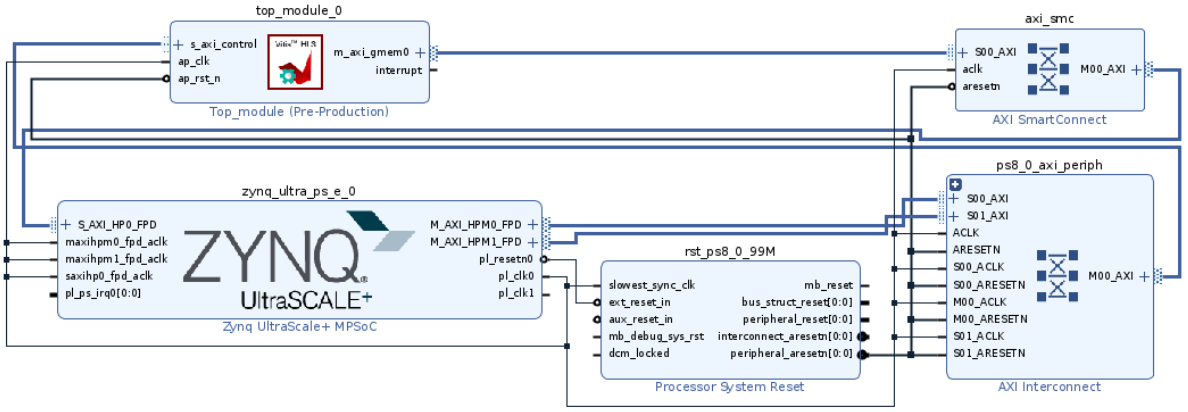
Lab

> Error Correction Code

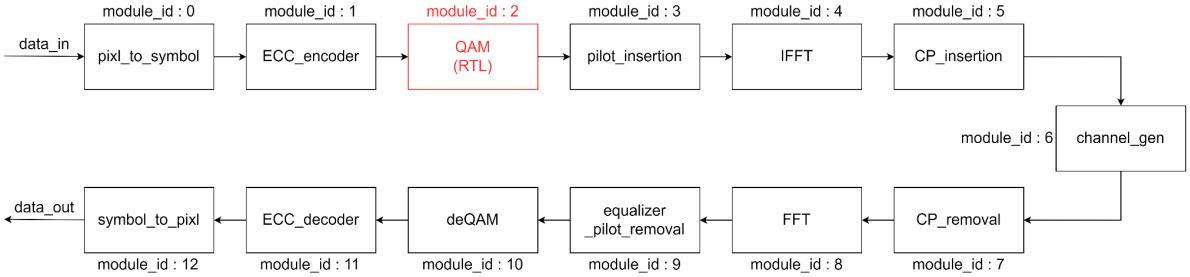
此實驗目的是希望可以透過實作將Error Correction Code的source code加入通訊系統中，並將完成的系統在FPGA上面進行實現，來讓同學了解在Composable Pipeline中新增Kernel的流程，以下為加入ECC後的Block Diagram :



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。