



# **Jini Intelligent Computing**

## **Workbook of Lab. #1**



## Preamble

在Lab. #1的檔案中主要有下列專案目錄：

- hls\_Multiplication  
Vitis HLS乘法器原始碼檔案
- vvd\_Multip2Num  
範例乘法器Vivado Design Suite參考檔案
  - design\_1.tcl  
範例乘法器Block Design完成Generate Bitstream後匯出之TCL Script檔
  - MakeBit.bat  
範例乘法器完成Generate Bitstream後，將.bit/.hwh拷貝至專案根目錄之批次檔
- ipy\_Multip2Num  
範例乘法器系統程式Python原始碼檔及Jupyter Notebook原始碼編輯檔

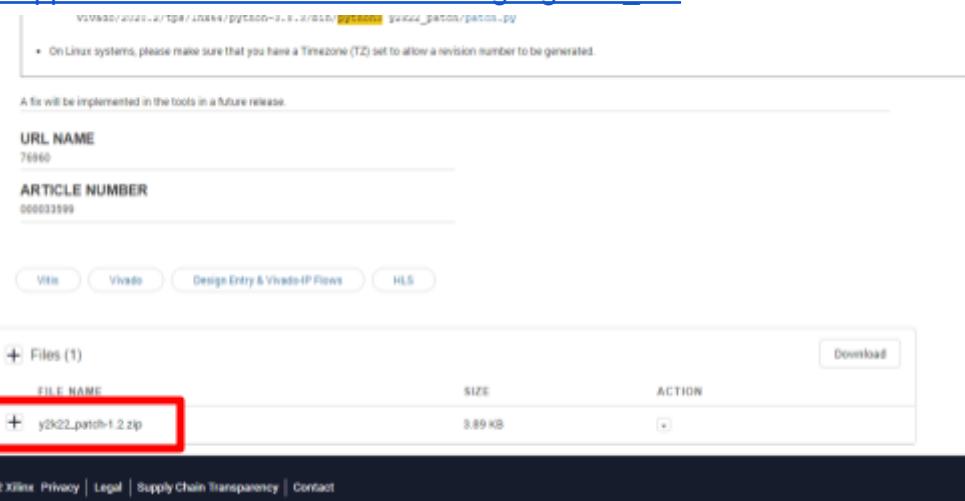
# 1. 前置作業

## 1.1. Time overflow patch update

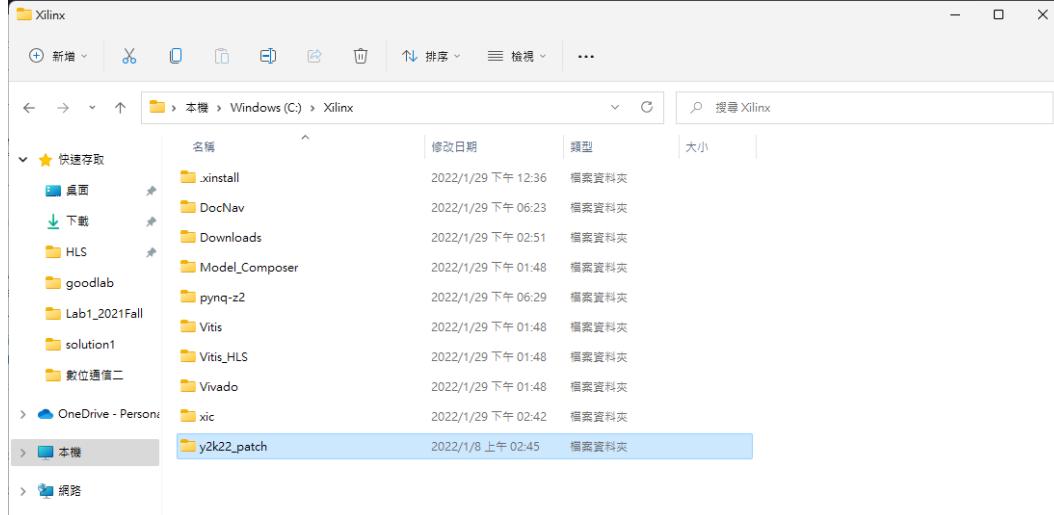
由於vitis2021.2版本有time overflow問題, 若直接使用會無法generate ip (vitis hls及vivado都無法輸出), 需下載patch才可使用。

先到以下網站下載 [y2k22\\_patch-1.2.zip](https://support.xilinx.com/s/article/76960?language=en_US), 解壓縮並把資料丟到 C:/Xilinx

[https://support.xilinx.com/s/article/76960?language=en\\_US](https://support.xilinx.com/s/article/76960?language=en_US)

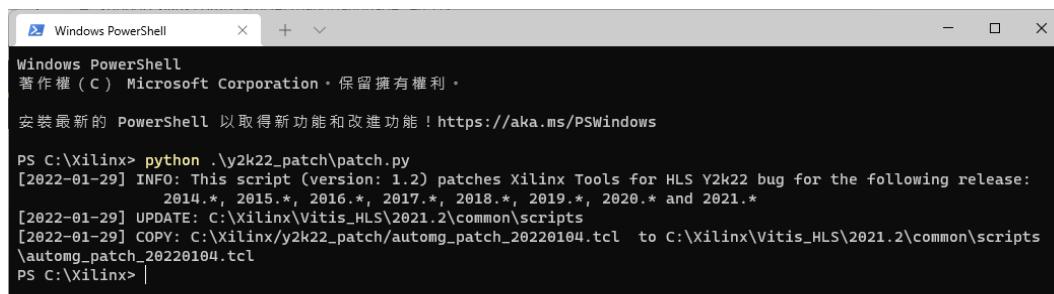


The screenshot shows the Xilinx Support Article page for article 76960. It includes a note about timezone settings, a URL name field containing '76960', an article number field containing '000033599', and a download section with a red box highlighting the file 'y2k22\_patch-1.2.zip'.

The screenshot shows the Windows File Explorer with the path 'Windows (C) > Xilinx'. A red box highlights the 'y2k22\_patch' folder, which contains several subfolders and files related to Xilinx tools like xinstall, DocNav, Downloads, Model\_Composer, pynq-z2, Vitis, Vitis\_HLS, Vivado, and xic.

在此位置打開terminal, 輸入 `python .\y2k22_patch\patch.py`。這樣即完成更新。



```

Windows PowerShell
著作權 (C) Microsoft Corporation • 保留擁有權利。

安裝最新的 PowerShell 以取得新功能和改進功能！https://aka.ms/PSWindows

PS C:\Xilinx> python .\y2k22_patch\patch.py
[2022-01-29] INFO: This script (version: 1.2) patches Xilinx Tools for HLS Y2k22 bug for the following release:
  2014.*, 2015.* , 2016.* , 2017.* , 2018.* , 2019.* , 2020.* and 2021.*
[2022-01-29] UPDATE: C:\Xilinx\Vitis_HLS\2021.2\common\scripts
[2022-01-29] COPY: C:\Xilinx\y2k22_patch\automp_patch_20220104.tcl to C:\Xilinx\Vitis_HLS\2021.2\common\scripts
\automp_patch_20220104.tcl
PS C:\Xilinx>

```

## 2. Implement Flow

### 2.1. Vitis HLS/IP Design

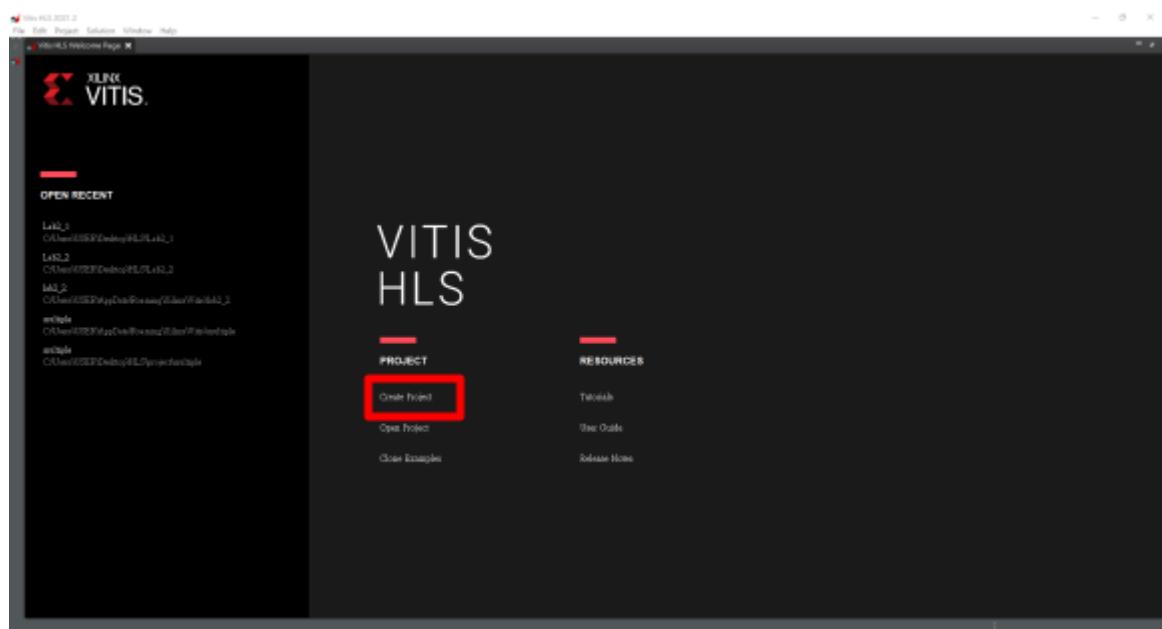
【施作環境為Windows】

啟動Vitis HLS開發套件。



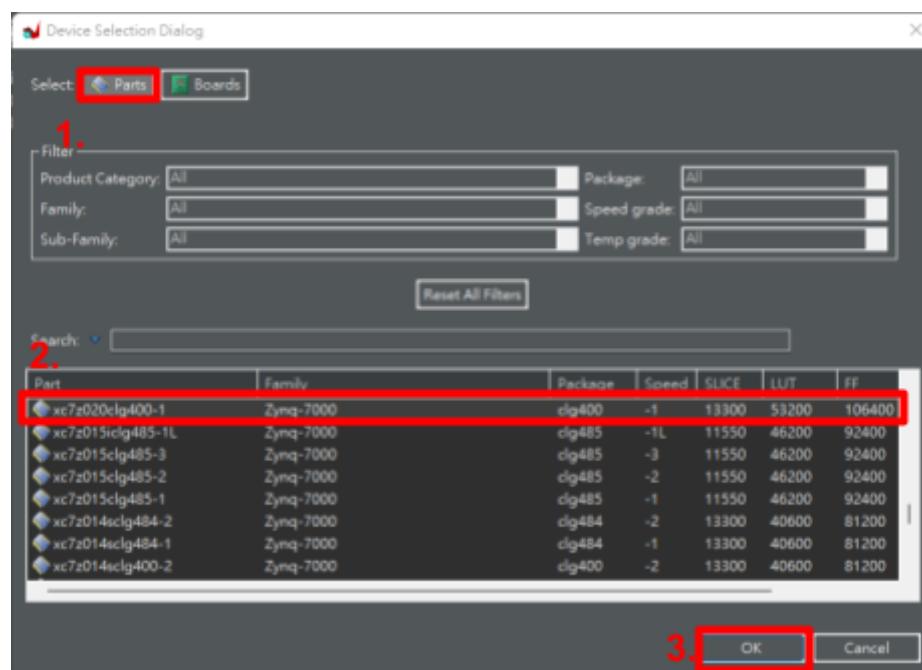
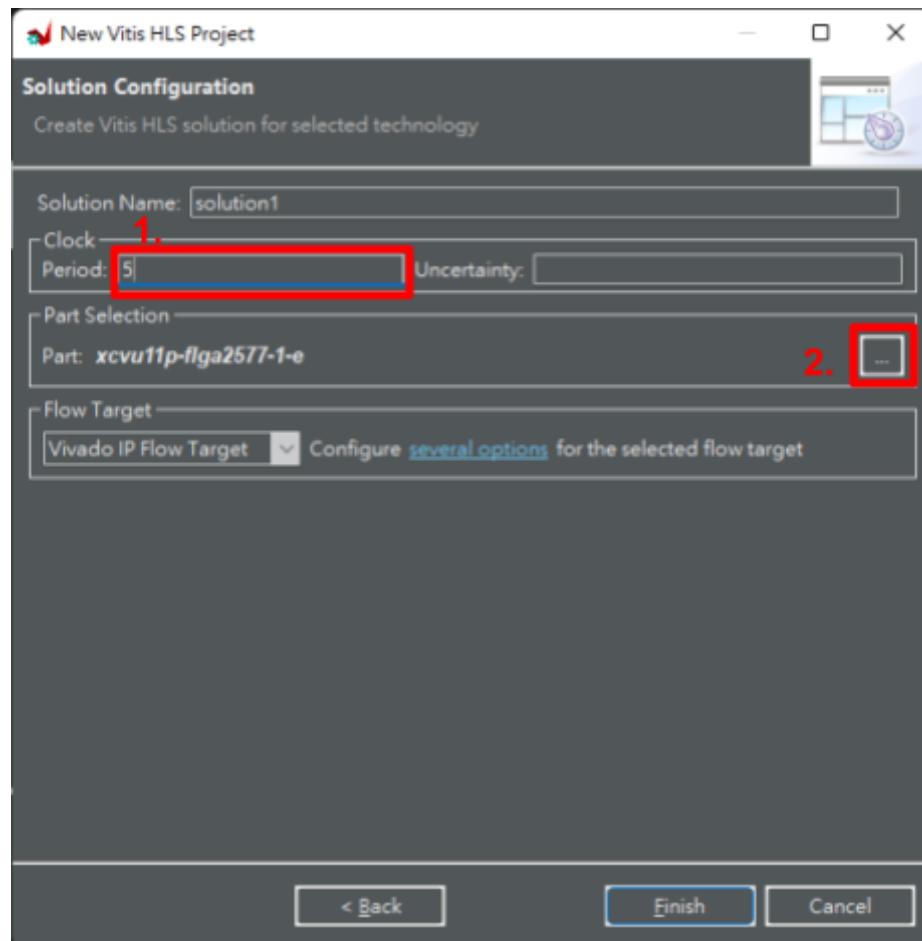
#### 2.1.1. Create Source Project

開啟新的專案，設定好專案存放路徑：





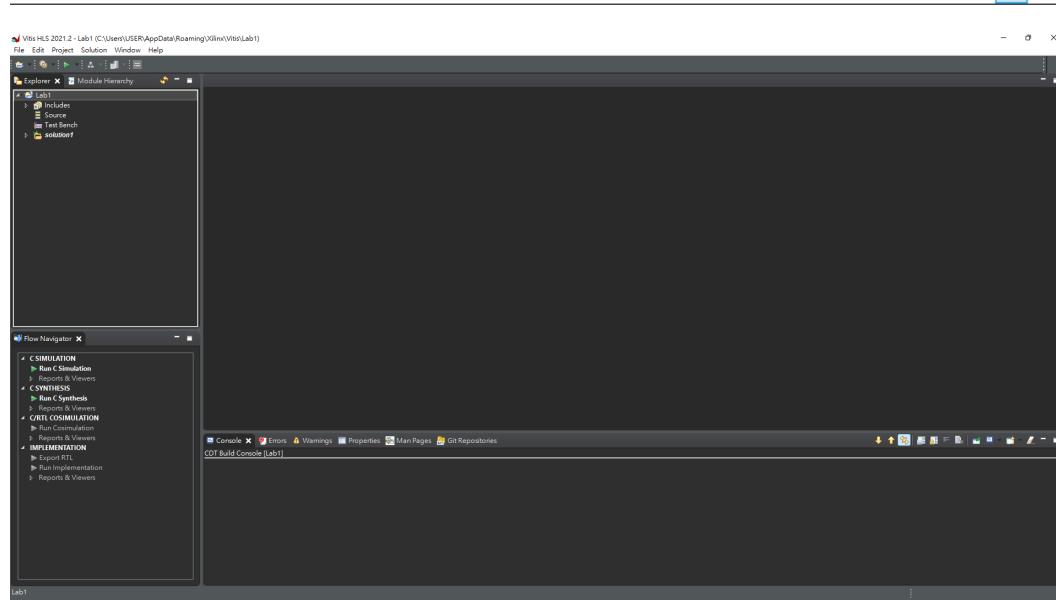
1. 若PYNQ-Z2的FPGA可運行於時脈200MHz以上，可以選擇週期為5 ns。
2. 因vitis 2021.2版本讀不到pynq z2 board file，所以選用與pynq z2相同Part的  
**xc7z020clg400-1**





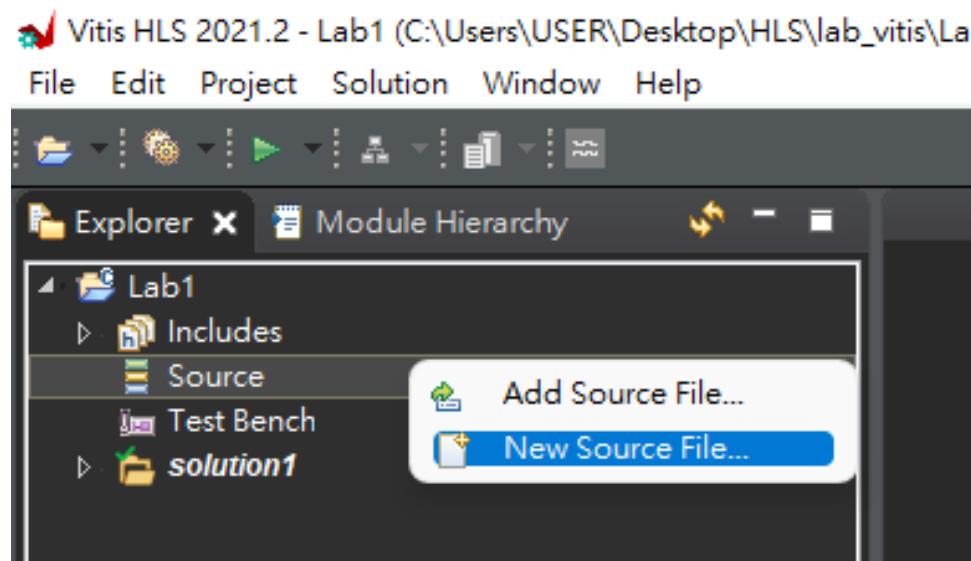
接著將hls\_Multiplication中三個檔案放到vitis hls專案對應的位置

名稱	修改日期	類型	大小
.apc	2022/1/27 下午 05:10	檔案資料夾	
.settings	2022/1/27 下午 05:10	檔案資料夾	
solution1	2022/1/27 下午 05:11	檔案資料夾	
.cproject	2022/1/27 下午 05:10	CPROJECT 檔案	29 KB
.project	2022/1/27 下午 05:10	PROJECT 檔案	2 KB
.vitis_hls_log_all.xml	2022/1/27 下午 05:11	XML Document	1 KB
hls.app	2022/1/27 下午 05:10	APP 檔案	1 KB
Multiplication.cpp	2021/11/16 下午 05:47	CPP 檔案	1 KB
Multiplication.h	2021/11/16 下午 05:40	C Header 來源檔案	1 KB
MultipTester.cpp	2020/7/21 上午 04:24	CPP 檔案	1 KB

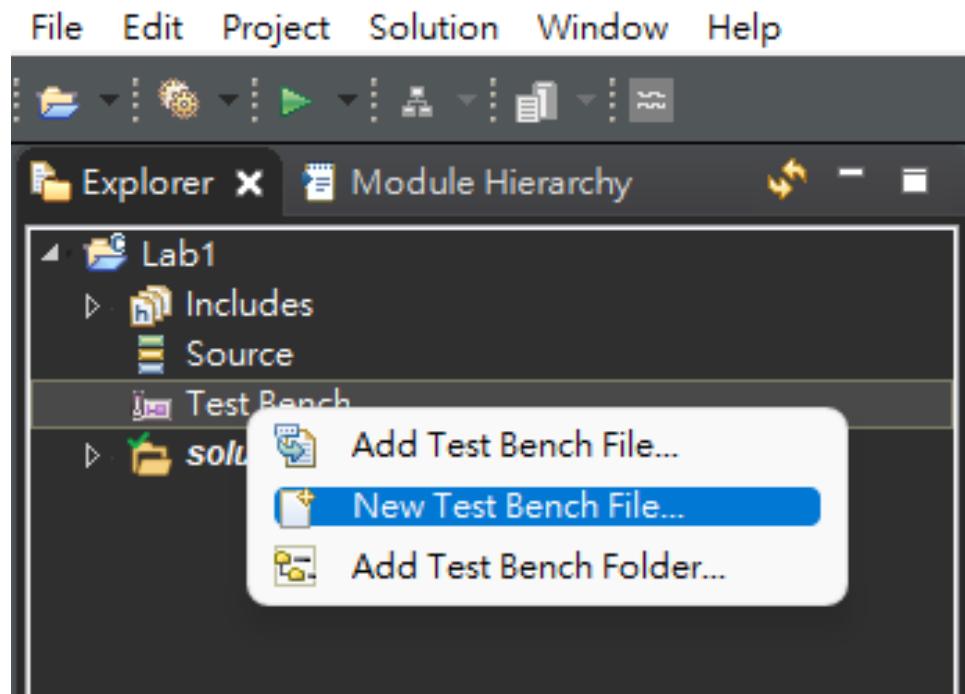


進入Vitis HLS專案IDE畫面後，加入Source/Test Bench的原始碼檔。

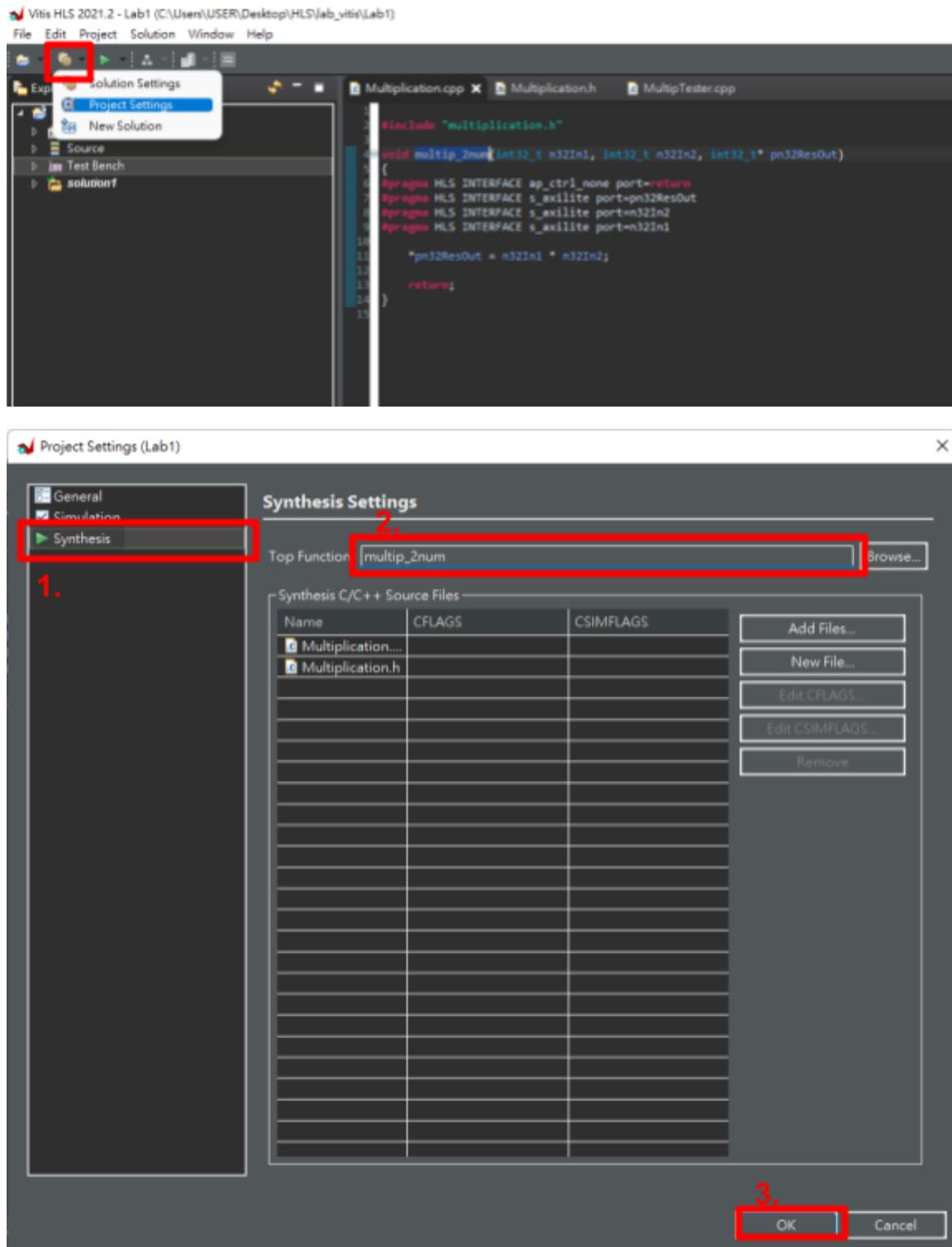
在專案Source按滑鼠右鍵加入新檔，在Source選擇Multiplication.cpp及Multiplication.h檔。（#Include部分注意大小寫問題。）



在專案Test Bench按滑鼠右鍵加入新檔，在Test Bench選擇MultipTester.cpp檔。



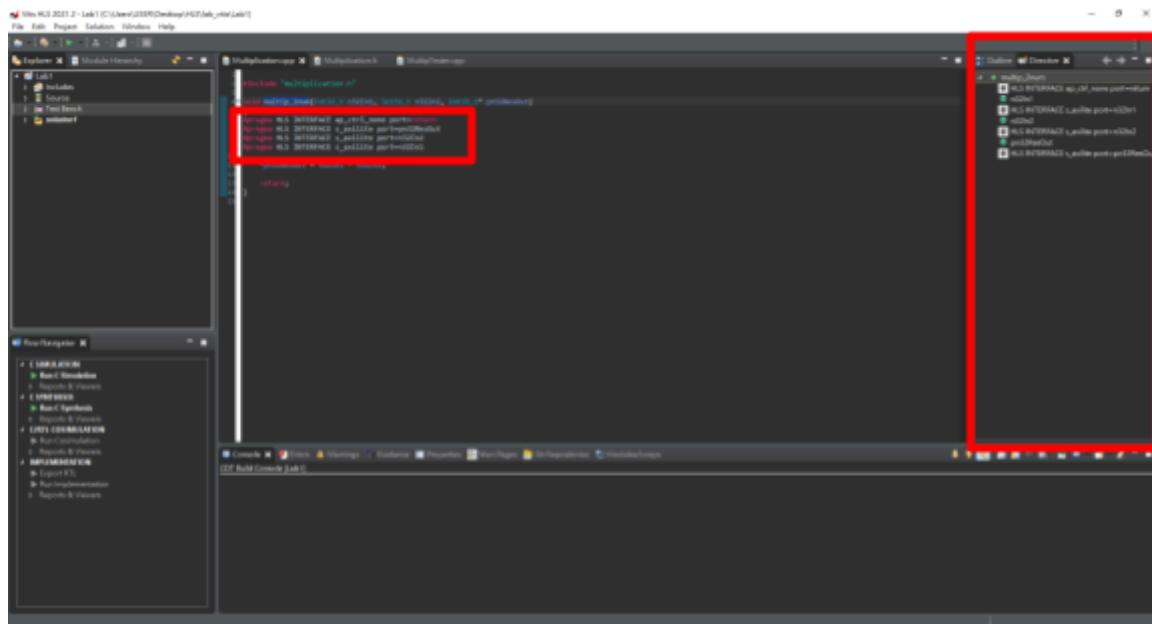
專案必須指明IP匯出的Top Function，在工具列設定圖加入Top Function名稱。(Top Function名稱即為 Multiplication.cpp Function 名稱)



## 2.1.2. Directives Control

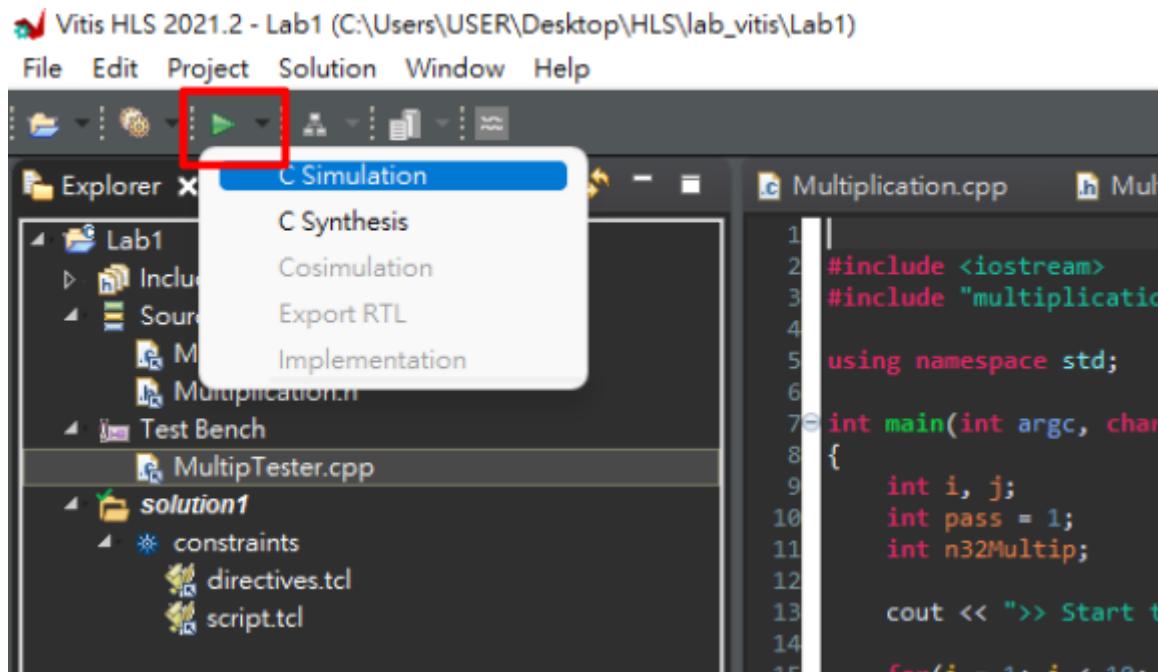
在開始Synthesis之前必須先加入directive描述。

加入directive有兩種方式，一種是inline方式(使用#pragma)，另一種是以 directives.tcl來控制。Lab 1是使用第一種方式，所以不需另外增加directives。



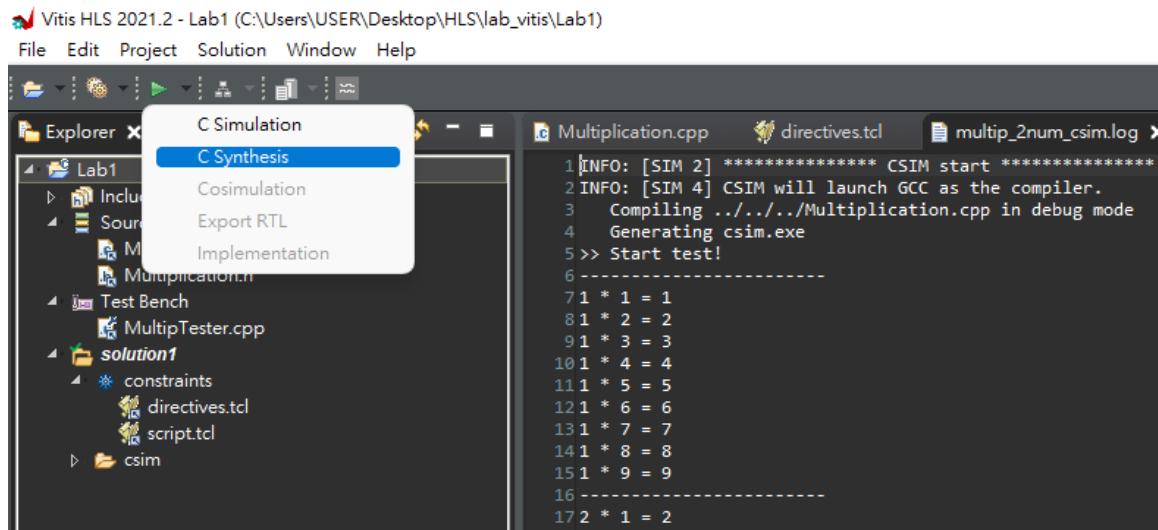
## 2.1.3. C Simulation

在Lab. #1裡提供Test Bench的C Simulation原始碼檔，已經有匯入專案。可執行C Simulation來驗證IP的執行結果。結果會顯示在下方console欄。

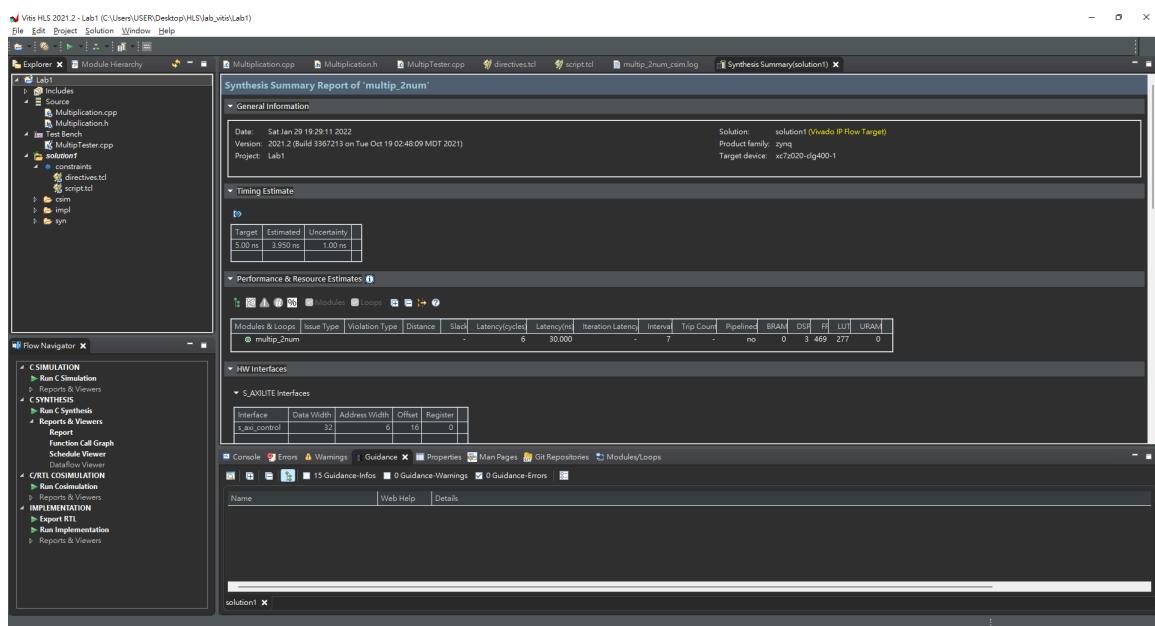


## 2.1.4. Synthesis

在Vitis HLS的功能列執行C Synthesis:



彈出視窗按ok進行C synthesis



完成C Synthesis會在主視窗回報synthesis report。

若有slack / time violation, 可調整period。

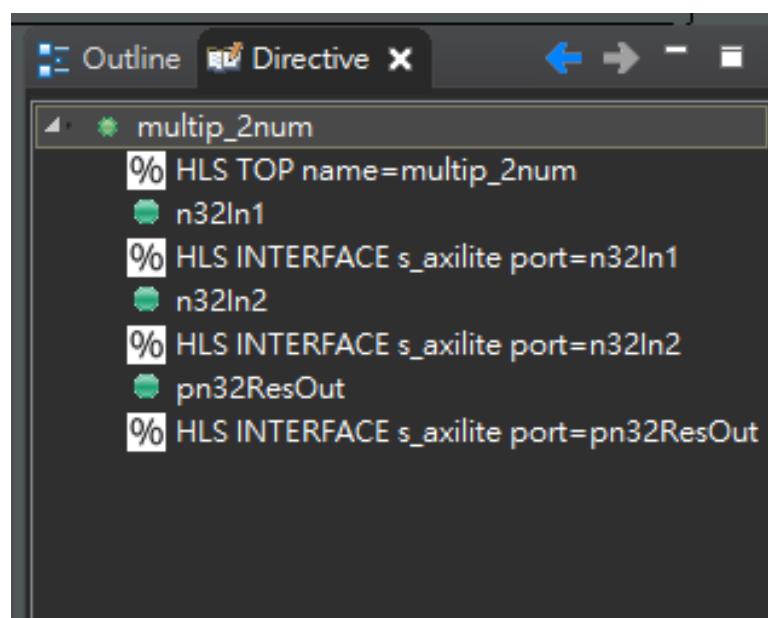
## 2.1.5. Cosimulation

當進行Cosimulation時directive的組態需改成如下圖，將ap\_ctrl\_none那行拿掉，不然會出現錯誤無法完成Cosimulation。在執行Cosimulation必須重新Synthesis。

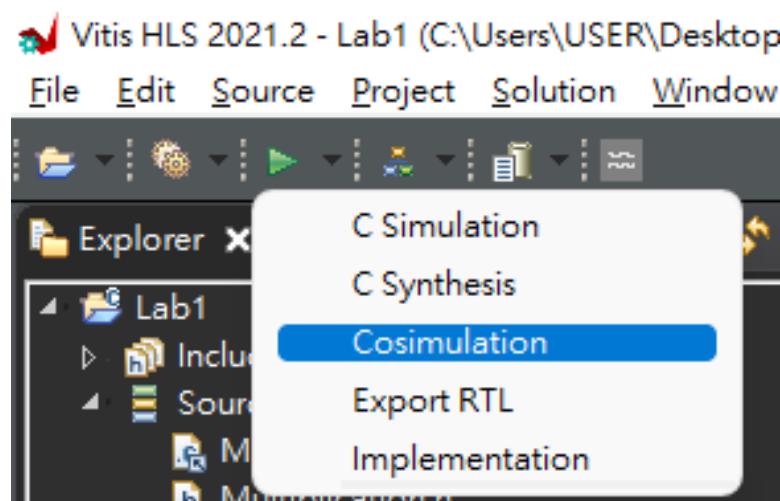
參考網址：

[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2021\\_1/ug871-vivado-high-level-synthesis-tutorial.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug871-vivado-high-level-synthesis-tutorial.pdf)

<https://forums.xilinx.com/t5/High-Level-Synthesis-HLS/Using-ap-memory-with-ap-control-none/td-p/681187>

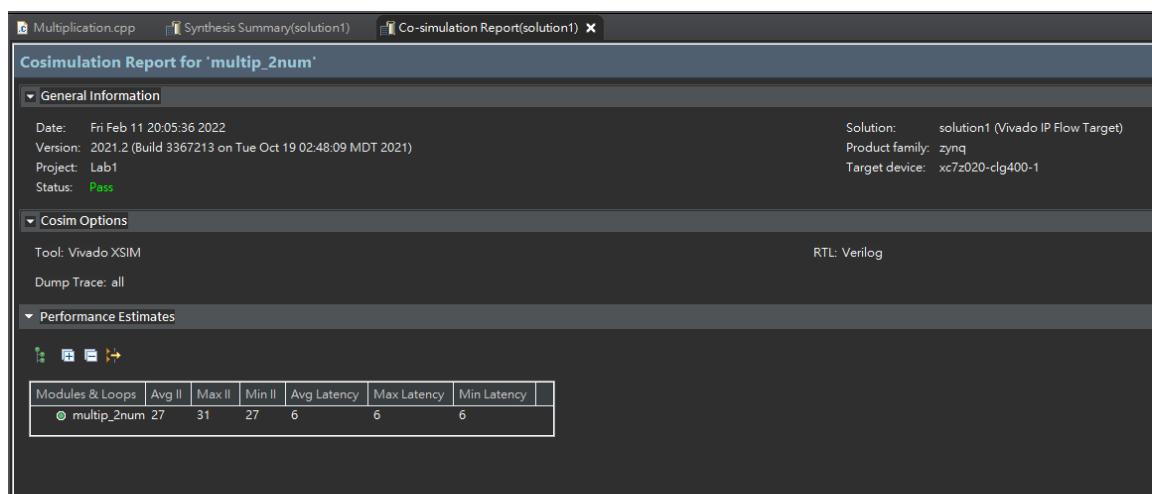
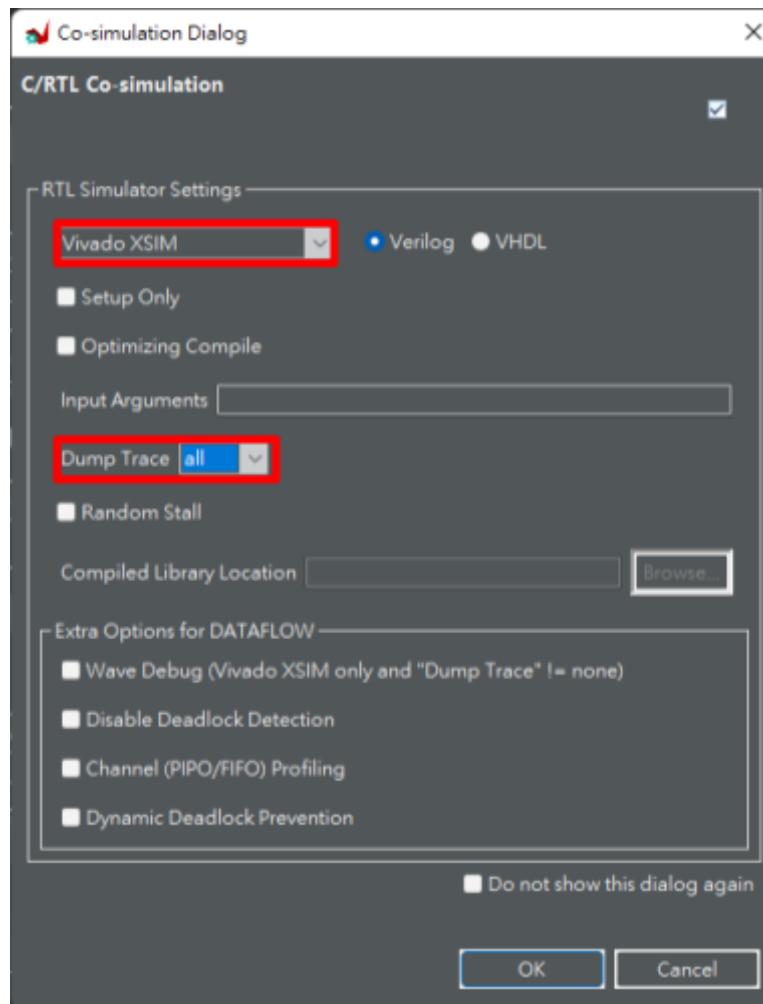


按下Cosimulation鍵來驗證設計。

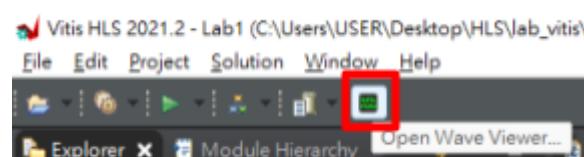




按下Cosimulation鍵後會彈出對話視窗，選擇Vivado XSIM，將dump trace選擇all

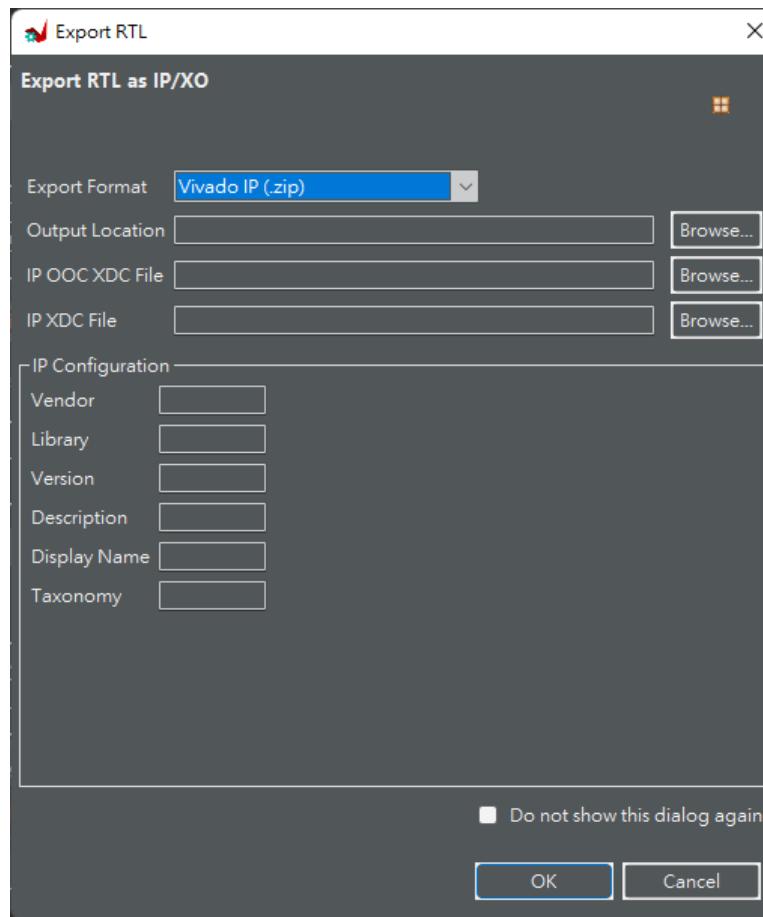
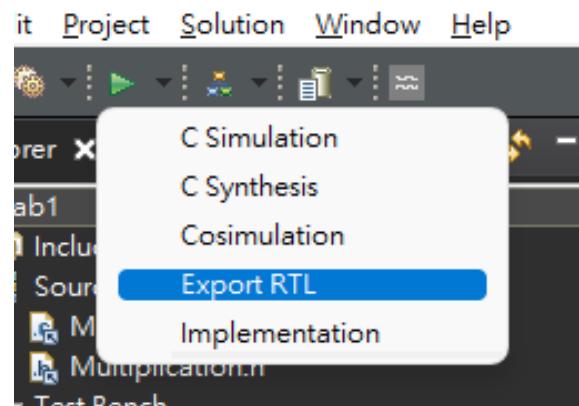


在輸出視窗檢視。要檢視波形可執行Open Wave Viewer。



## 2.1.6. Export IP

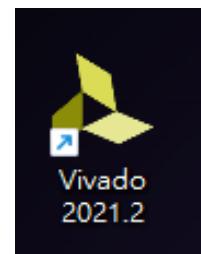
完成IP的設計後，先從Vitis HLS匯出IP，之後在Vivado Design Suite的Vivado還需要匯入由Vitis HLS匯出的IP。現階段操作Vitis HLS匯出RTL功能，在匯出RTL前先還原原本的directive，之後重新Synthesis才是做出的設計：



## 2.2. Vivado/Implement Flow

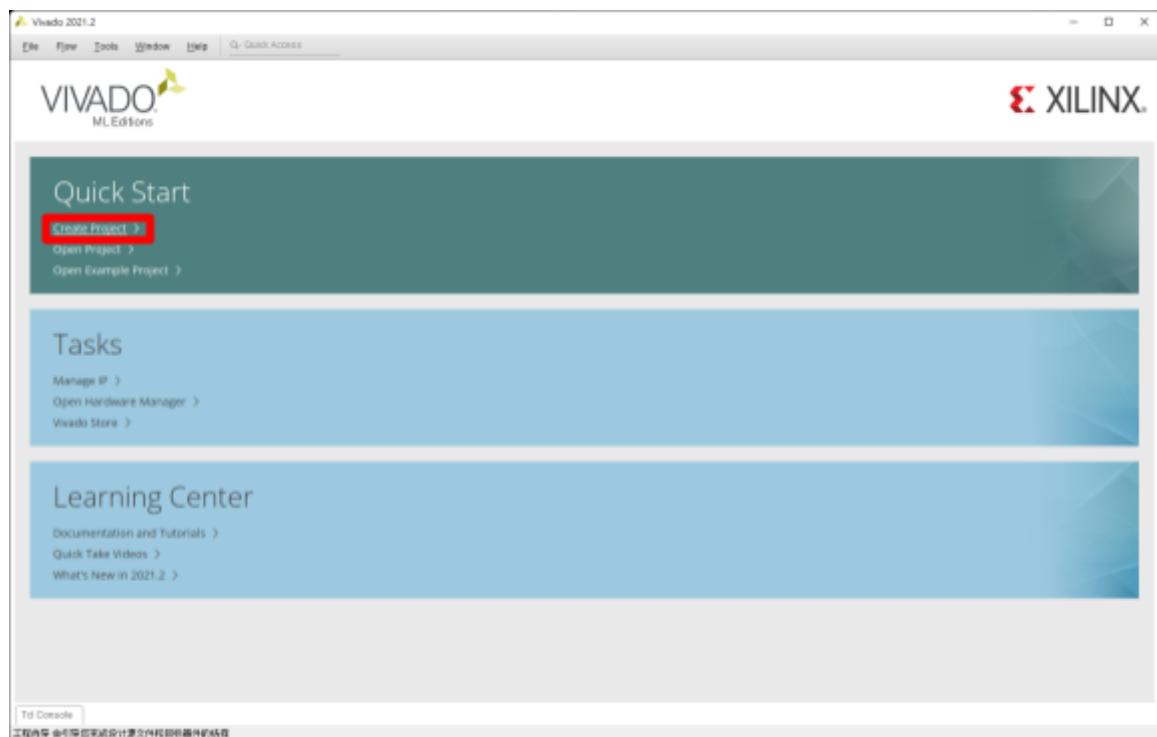
【施作環境為在使用者PC/laptop/notebook (Windows Base)。】

啟動Vivado Design Suite的Vivado開發套件。



### 2.2.1. Create Design Project

開啟新的專案，設定好專案存放路徑：



選擇好專案儲存路徑。RTL Project、sources 跟 constraints 留空之後再加。



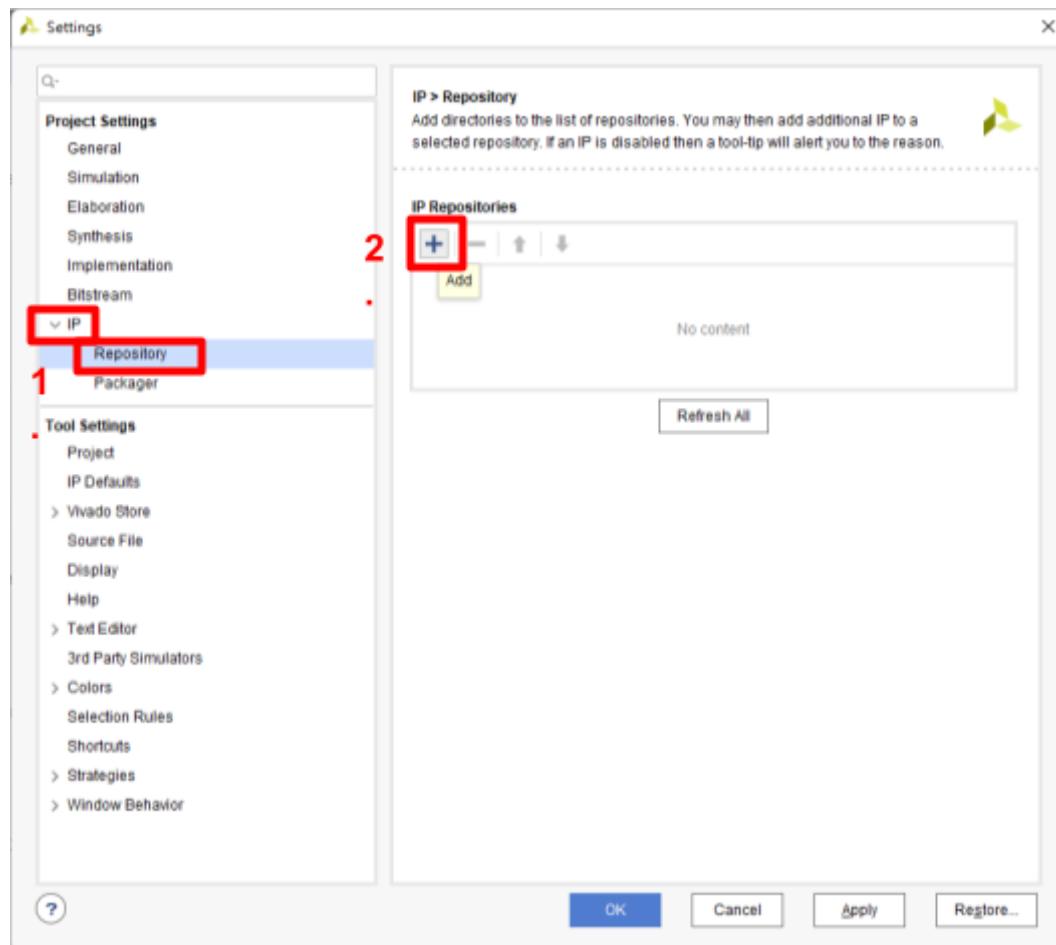
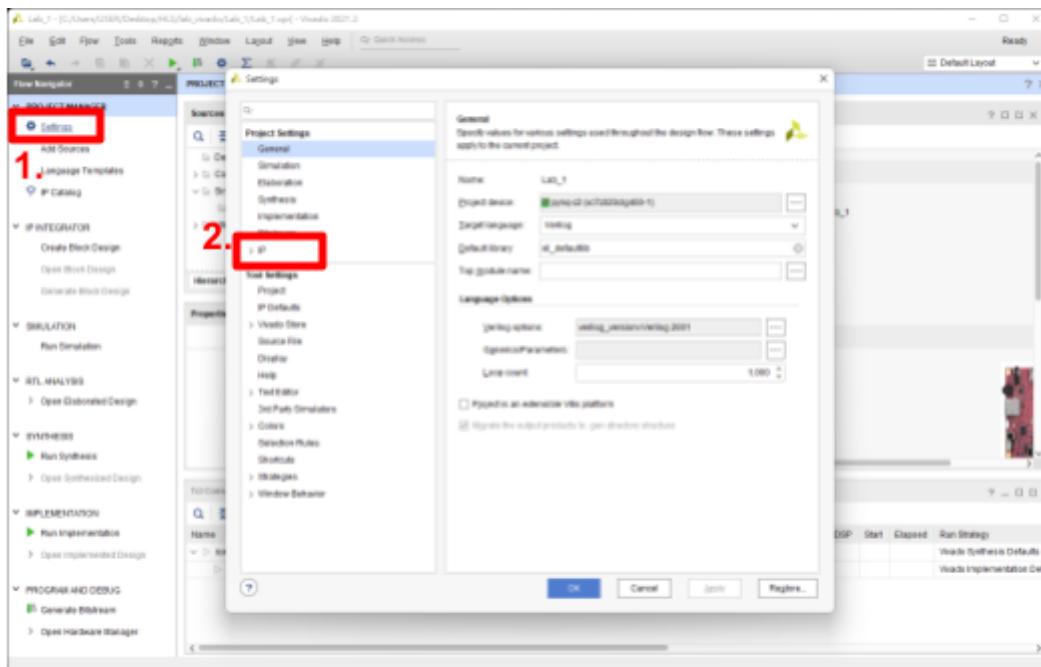
Parts選擇xc7z020clg400-1。

The screenshot shows the 'Default Part' configuration window in the Xilinx Project Manager. The 'Parts' tab is selected. A search bar at the top contains the text 'xc7z020clg400-1'. Below the search bar are several filter dropdowns: Category (All), Family (All), Package (All), Speed (All), Temperature (All), and Static power (All). The search results table has columns: Part, I/O Pin Count, Available IOBs, LUT Elements, FlipFlops, Block RAMs, Ultra RAMs, DSPs, Gb Transceivers, and C. One row is highlighted in blue, corresponding to the search term: xc7z020clg400-1, which has 400 I/O pins, 125 available IOBs, 53200 LUT elements, 106400 flip-flops, 140 block RAMs, 0 ultra RAMs, 220 DSPs, 0 Gb transceivers, and 0 C. At the bottom of the window are buttons for '?', '< Back' (disabled), 'Next >', 'Finish', and 'Cancel'.

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	C
xc7z020clg400-1	400	125	53200	106400	140	0	220	0	0

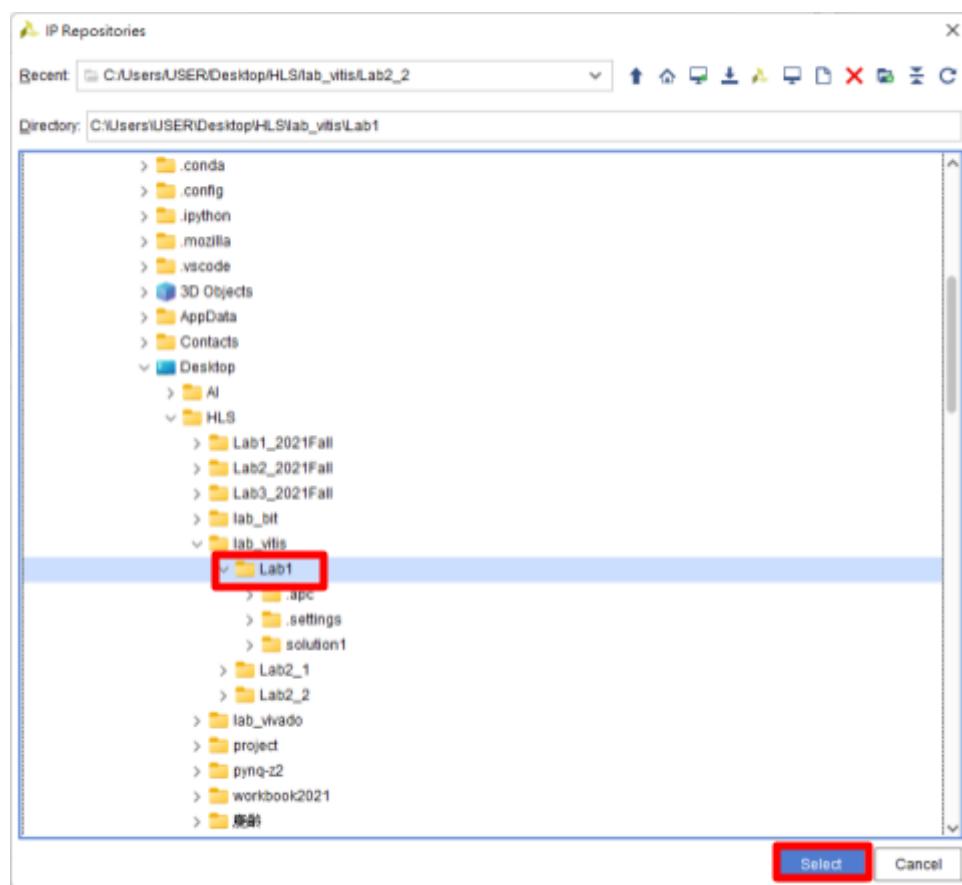
## 2.2.2. Import IP

進入Vivado專案IDE畫面後，第一步驟是匯入由Vitis HLS所產生的IP，在專案管理點擊Settings選項。



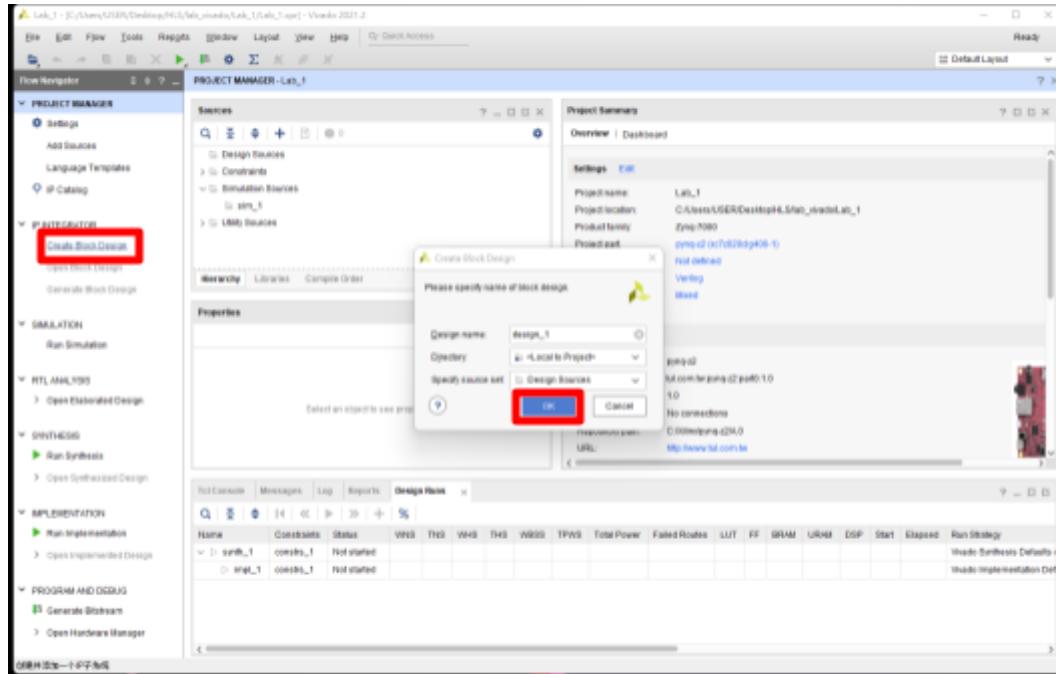
將IP Repositories指定到Vitis HLS專案目錄，下一步會匯入由Vitis HLS專案開發的IP

o

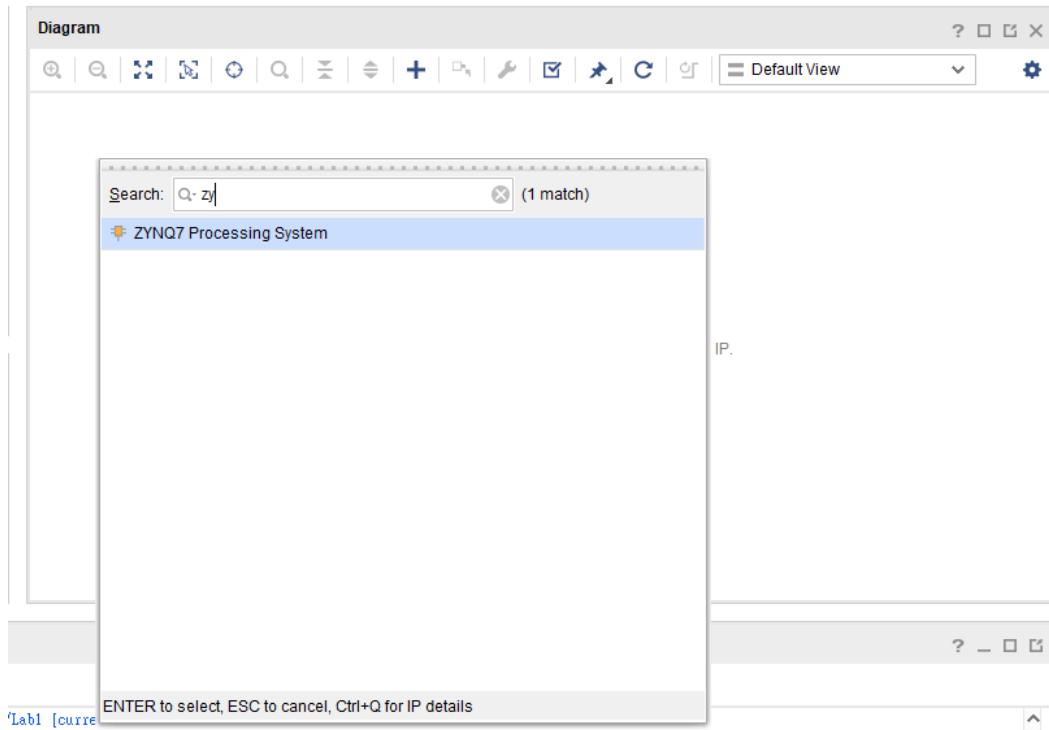


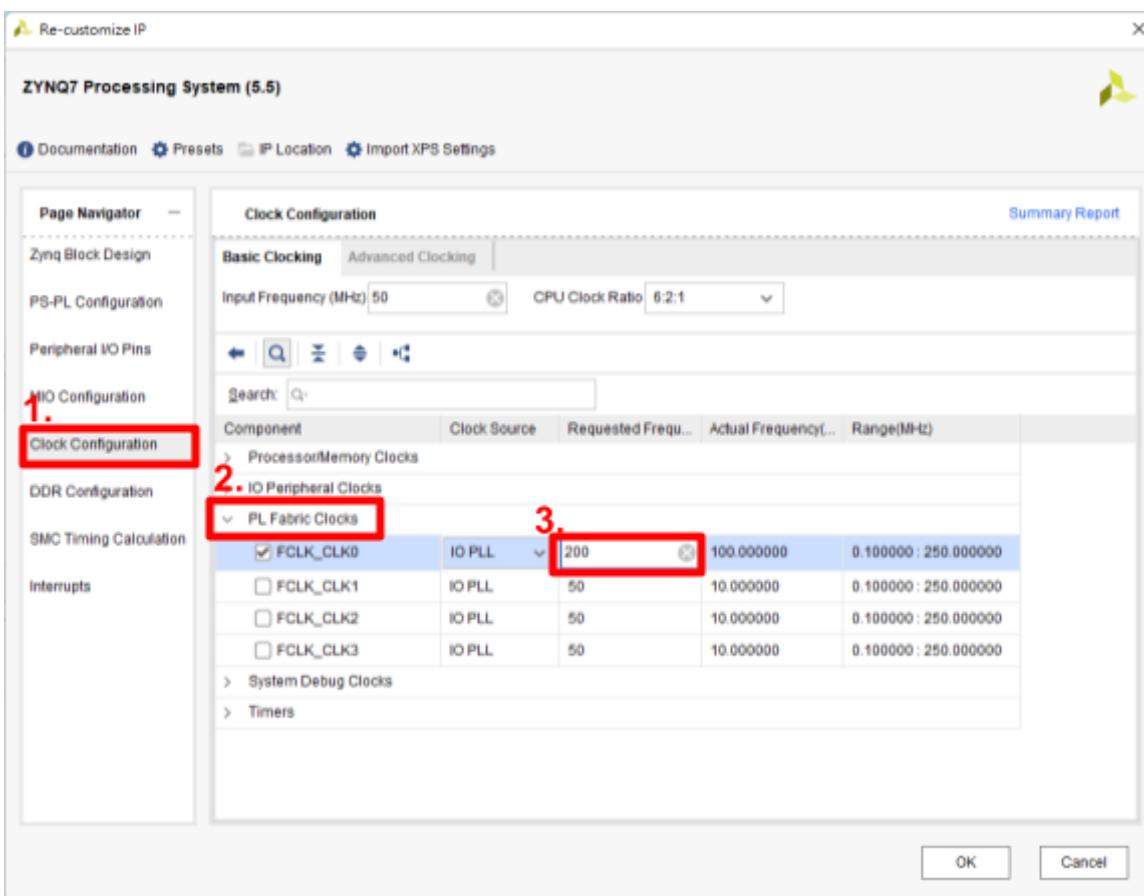
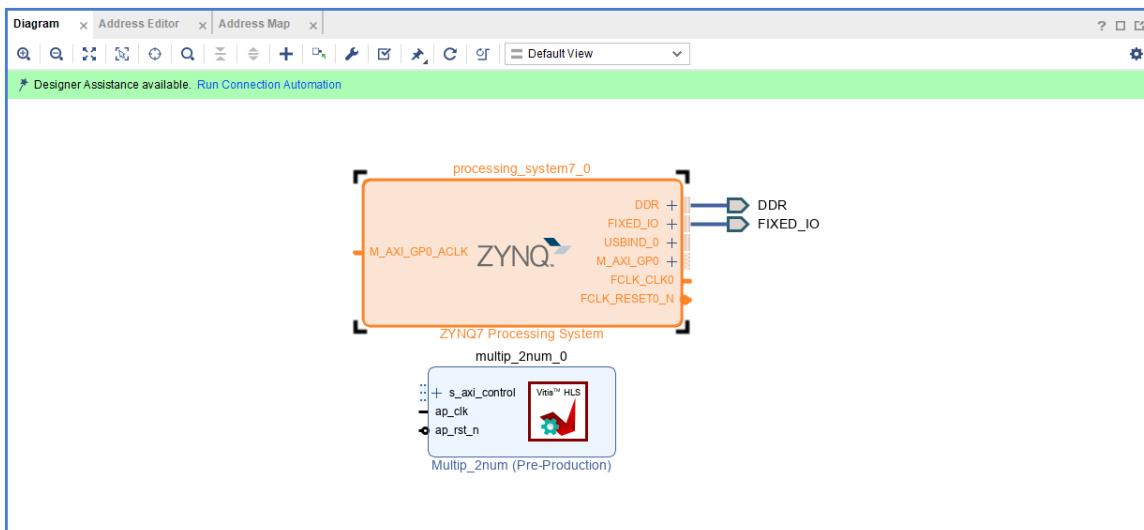
## 2.2.3. Block Design

回到Vivado專案IDE畫面，在專案管理點擊Create Block Design選項。

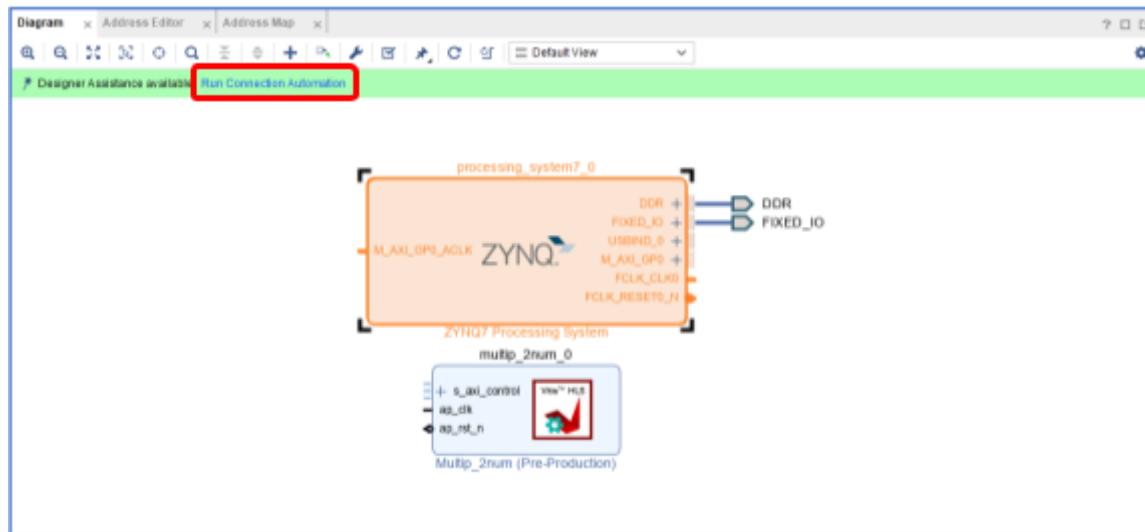


在Diagram tab視窗加入components(ZYNQ7 Processing System及multip\_2num\_0),  
 並在Run Block Automation後用滑鼠左鍵雙擊processing system block, 將PLL Fabric  
 clock設定為200MHz。

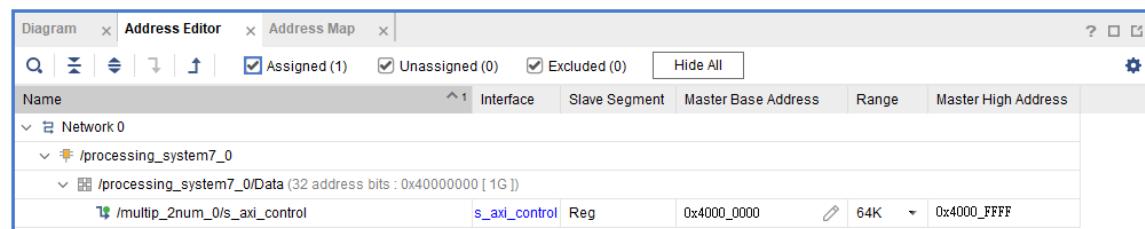
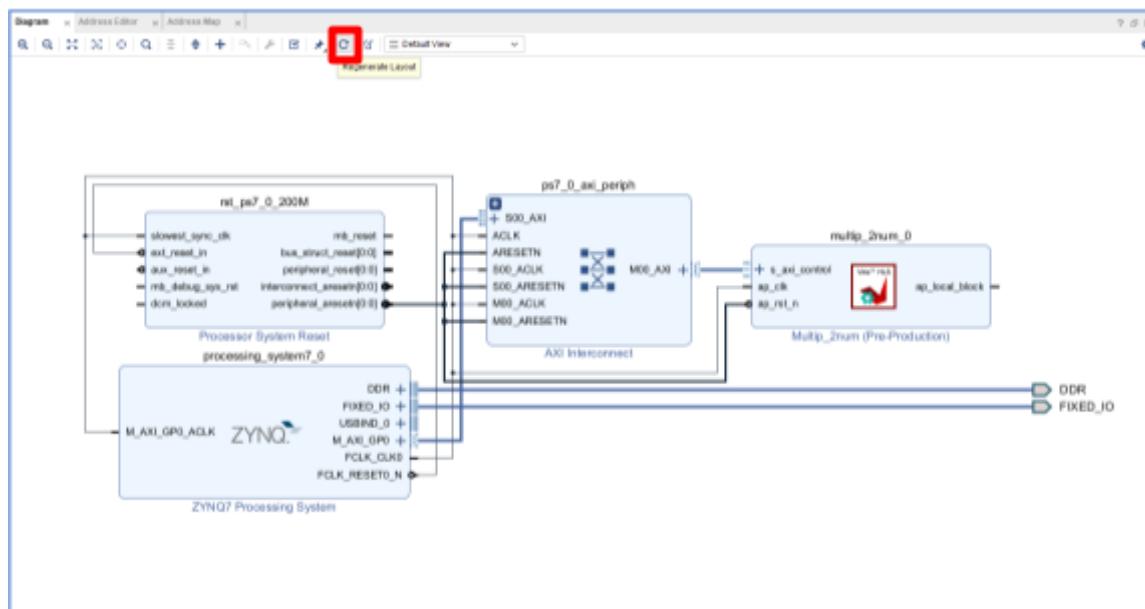




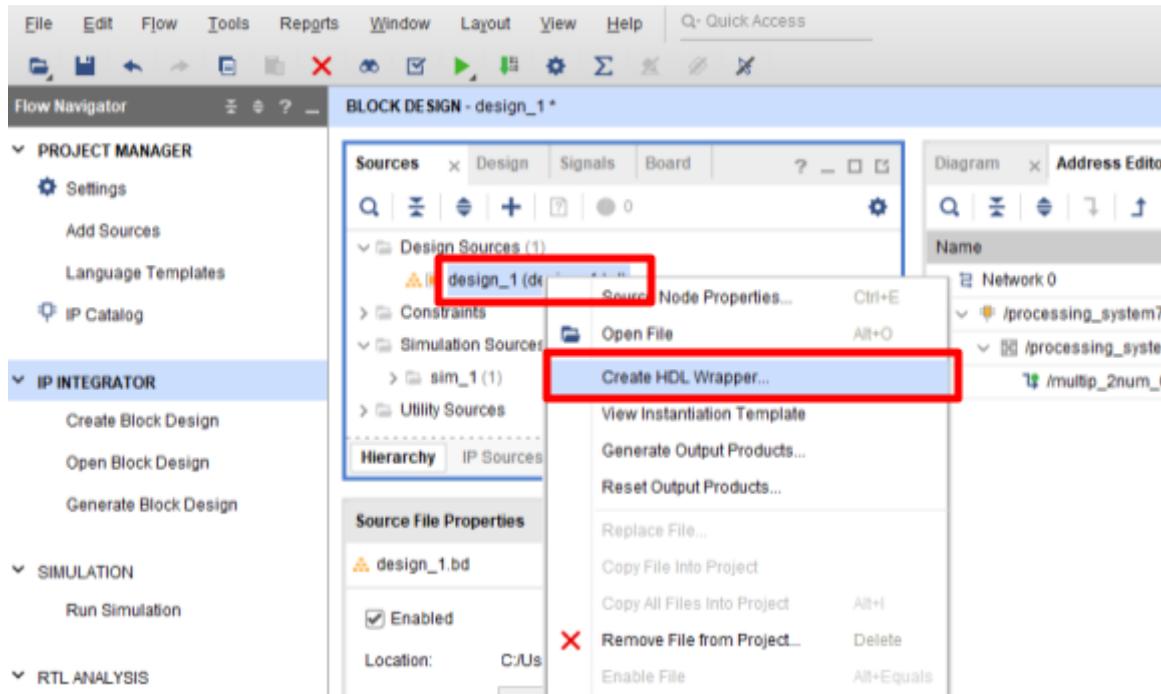
因為block design只有1個IP的設計，連線完成可由系統自動完成，直接執行Run Connection Automation即可。



完成後整個完整的diagram圖，可點選上方regenerate layout，檢查接線是否有誤，接著可切換到Address Editor tab檢視memory map：



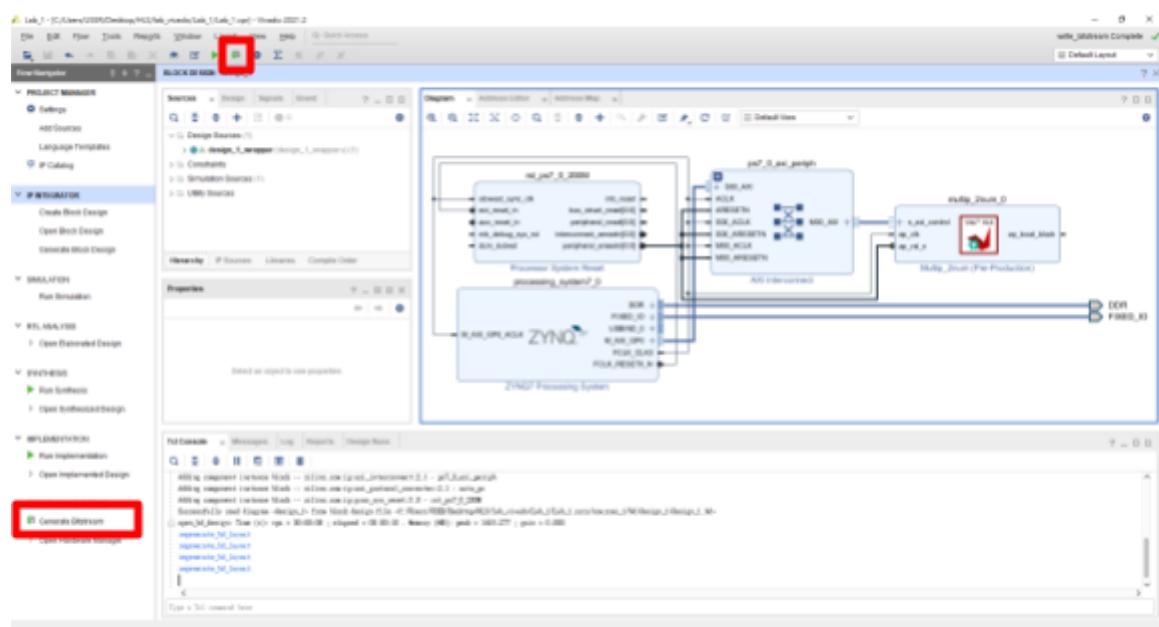
接下來進行HDL Wrapper動作：在Design子視窗Source tab頁面Design Sources的design\_1.bd點擊滑鼠右鍵進行HDL Wrapper：



## 2.2.4. Synthesis/Placement/Routing/Generate

### Bit-stream

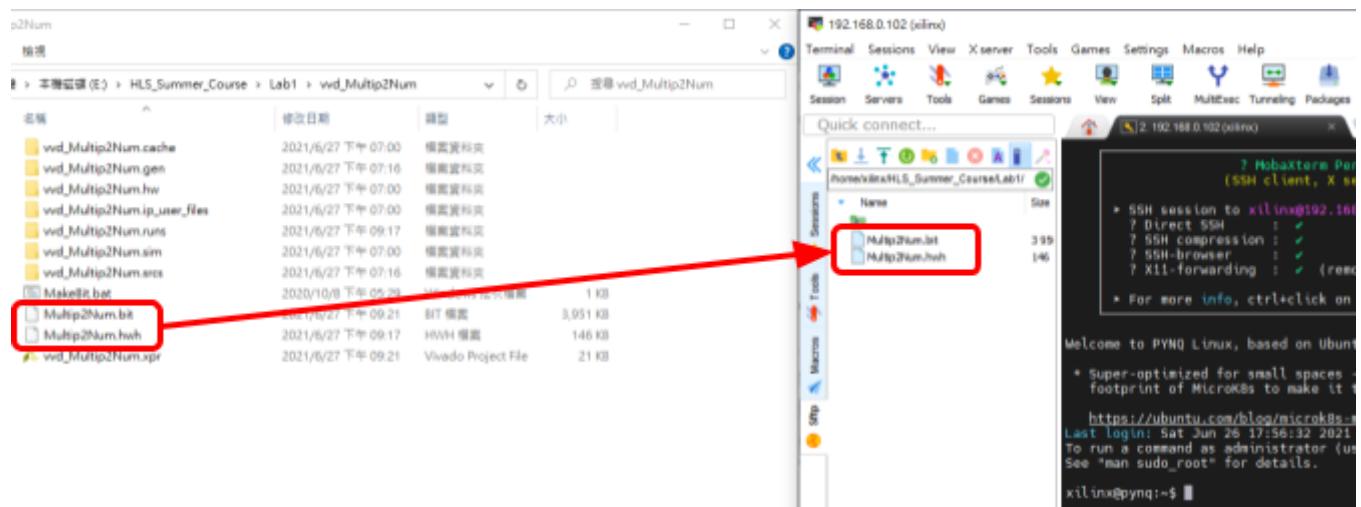
產生FPGA所需要的bit-stream file，由於要產生bit-stream file需要經過Synthesis/Placement/Routing的步驟，但可以省去單步執行的流程以一鍵執行由Vivado IDE自動完成所有流程。在Vivado專案IDE畫面，在專案管理點擊Generate Bitstream選項或由工具列按下Generate Bitstream按鍵。



Launch Runs的Numbers of jobs 可以選多一點，會快一些。

## 2.2.5. Bit-stream Transfer from Development Kit to Device

本實作已建立一個批次檔MakeBit.bat將FPGA運行時所需要.bit/.hwh拷貝到專案根目錄，此時只要將.bit/.hwh藉由MobaXterm或Samba傳送到PYNQ-Z2即可。提醒！開發者建立的專案名稱可能與MakeBit.bat內的專案名稱不相符，請自行修改.bat內專案路徑名稱。



bit檔位置 \Lab\_1\Lab\_1.runs\impl\_1\design\_1\_wrapper.bit

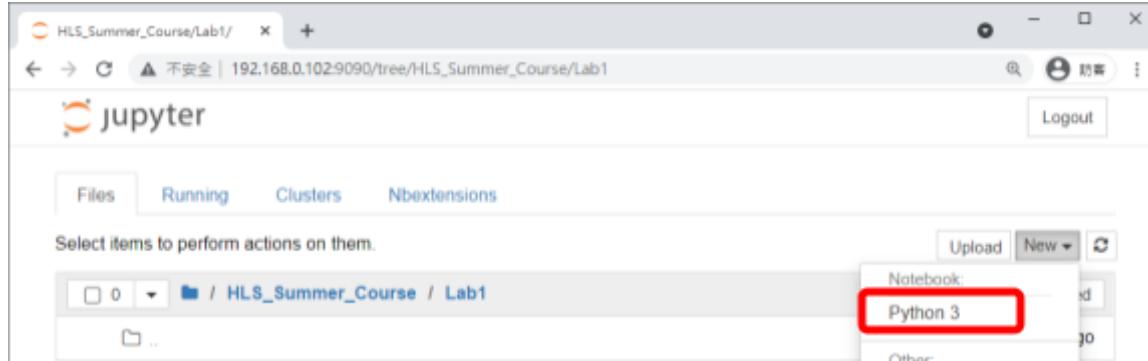
hwh檔位置 \Lab\_1\Lab\_1.gen\sources\_1\bd\design\_1\hw\_handoff\design\_1.hwh

更改檔名，並記得存放在pynq版上的位置。

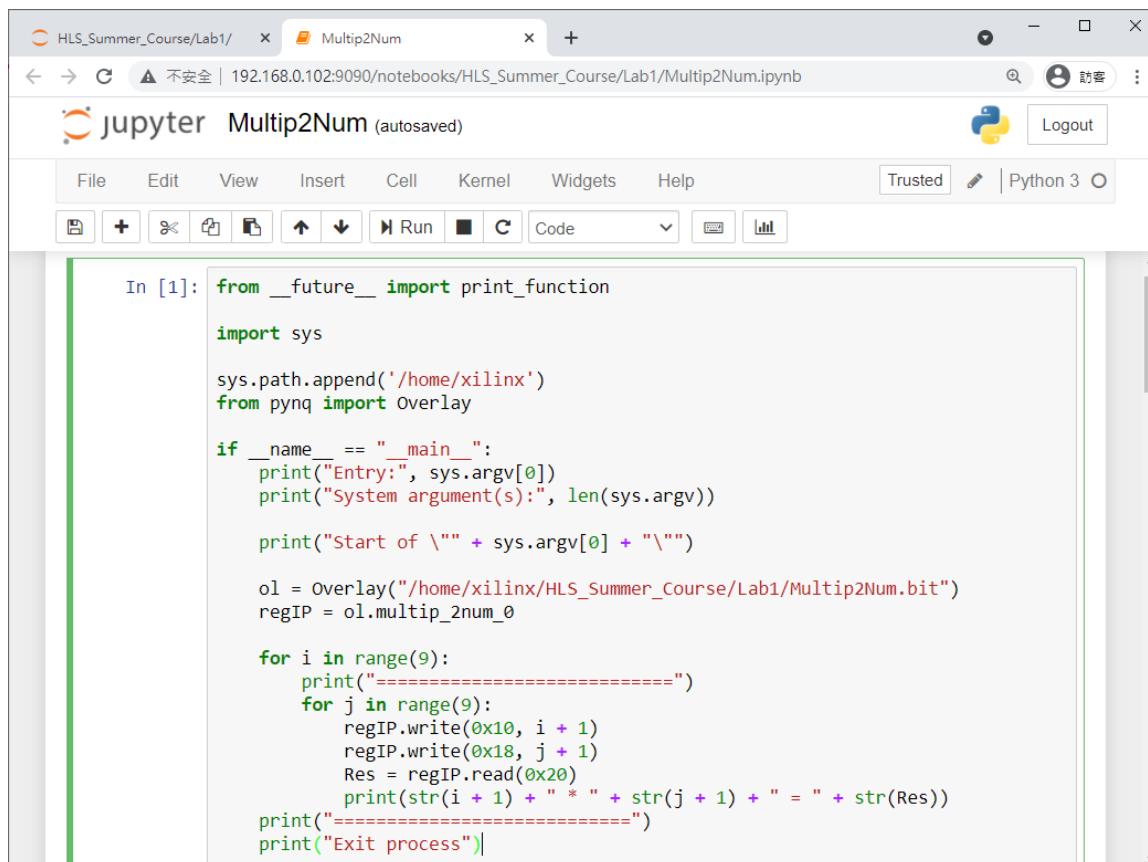
## 2.3. PYNQ/Host Program

### 2.3.1. Jupyter Notebook Browser Remote Editor

1. 啟動Jupyter Notebook，並開啟一個新的Python 3檔案。



2. 將Lab. #1提供的host program .py內容拷貝到瀏覽器編輯視窗，並運行後檢視結果。(記得檢查python code裡用到的檔案位置是否正確)



```

In [1]: from __future__ import print_function
import sys
sys.path.append('/home/xilinx')
from pynq import Overlay
if __name__ == "__main__":
    print("Entry:", sys.argv[0])
    print("System argument(s):", len(sys.argv))
    print("Start of \"{}\" + sys.argv[0] + \"\"")
    ol = Overlay("/home/xilinx/HLS_Summer_Course/Lab1/Multip2Num.bit")
    regIP = ol.multip_2num_0
    for i in range(9):
        print("====")
        for j in range(9):
            regIP.write(0x10, i + 1)
            regIP.write(0x18, j + 1)
            Res = regIP.read(0x20)
            print(str(i + 1) + " * " + str(j + 1) + " = " + str(Res))
    print("====")
    print("Exit process")

```

Note:

Kernel的register address offset可在以下檔案中找到：

hls\_Multiplication\solution1\impl\misc\drivers\multip\_2num\_v1\_0\src\xmultip\_2num\_hw.h



## 2.3.2. Understanding PYNQ

PYNQ是建立在Xilinx platform上的API模組，可運行Python程式碼的環境，PYNQ提供Python語言模組可進行對FPGA組態建立及流程控制。參考連結：

<http://www.pynq.io/board.html>

PYNQ open source: <https://github.com/xilinx/pynq>