

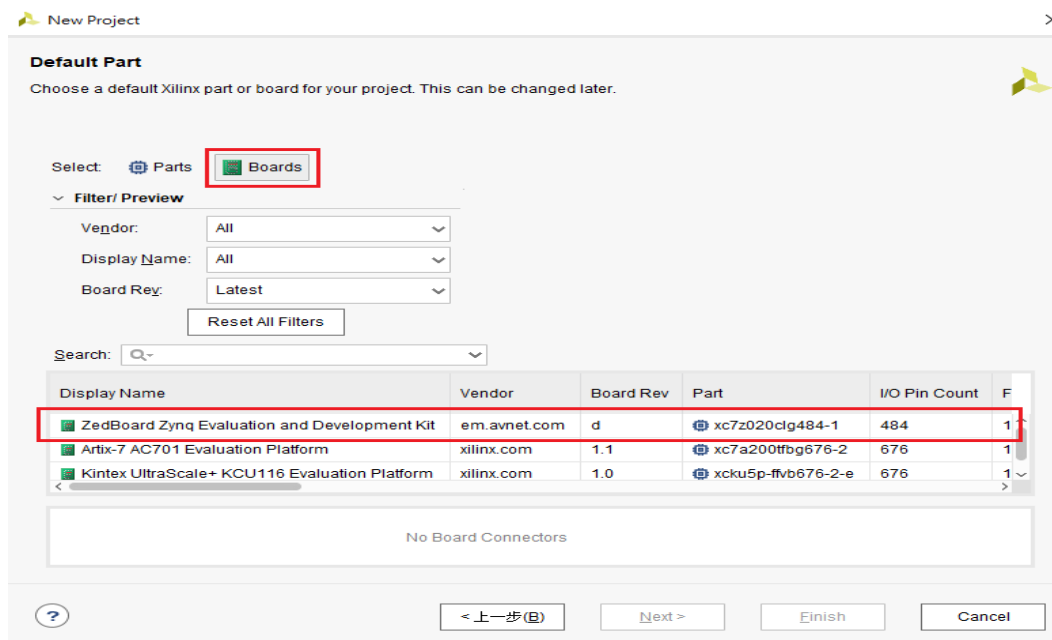
ZedBoard Training Lab2 – Vivado On Board Design

A、使用軟體：Vivado 2017.2

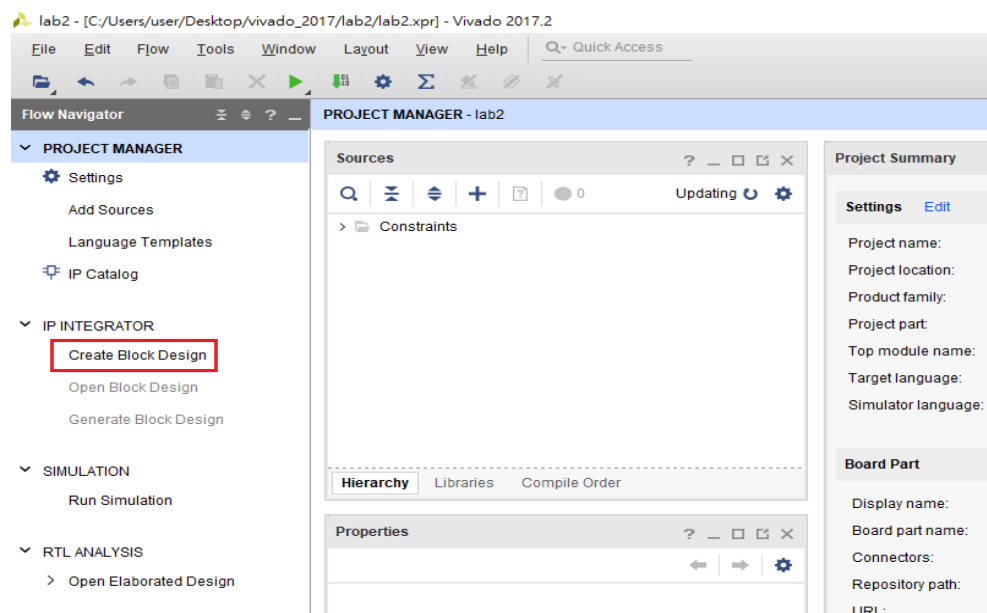
B、練習內容：使用 Vivado 提供的 GPIO IP，讓 CPU 以 AXI 溝通方式對板子上的設備(指撥開關、按鈕、LED 等等)做操作。

C、Vivado 軟體開發教學：

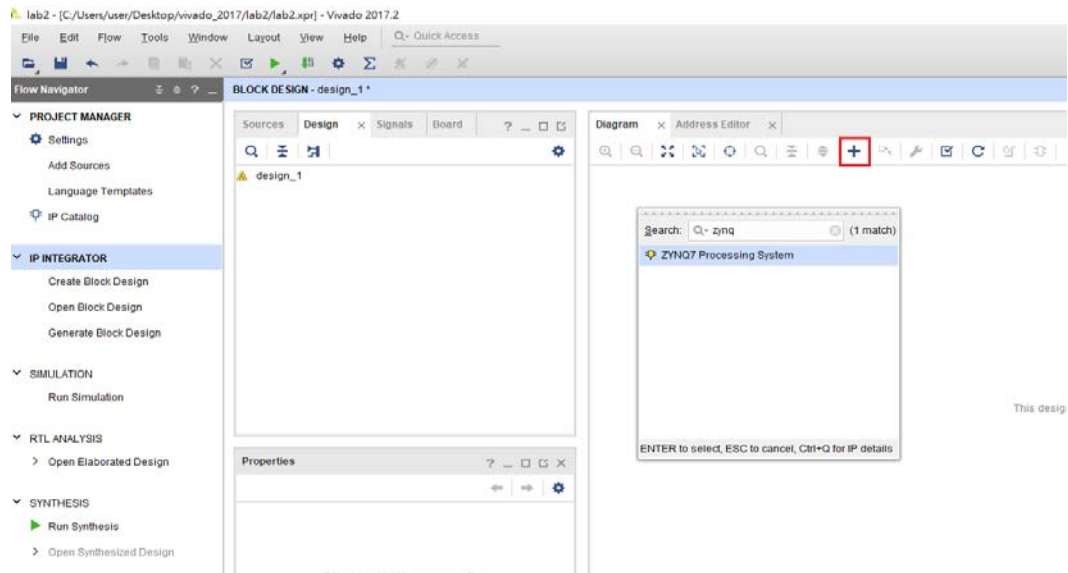
1、創立新專案，勾選(Do not specify sources at this time)，板子選擇 ZedBoard。



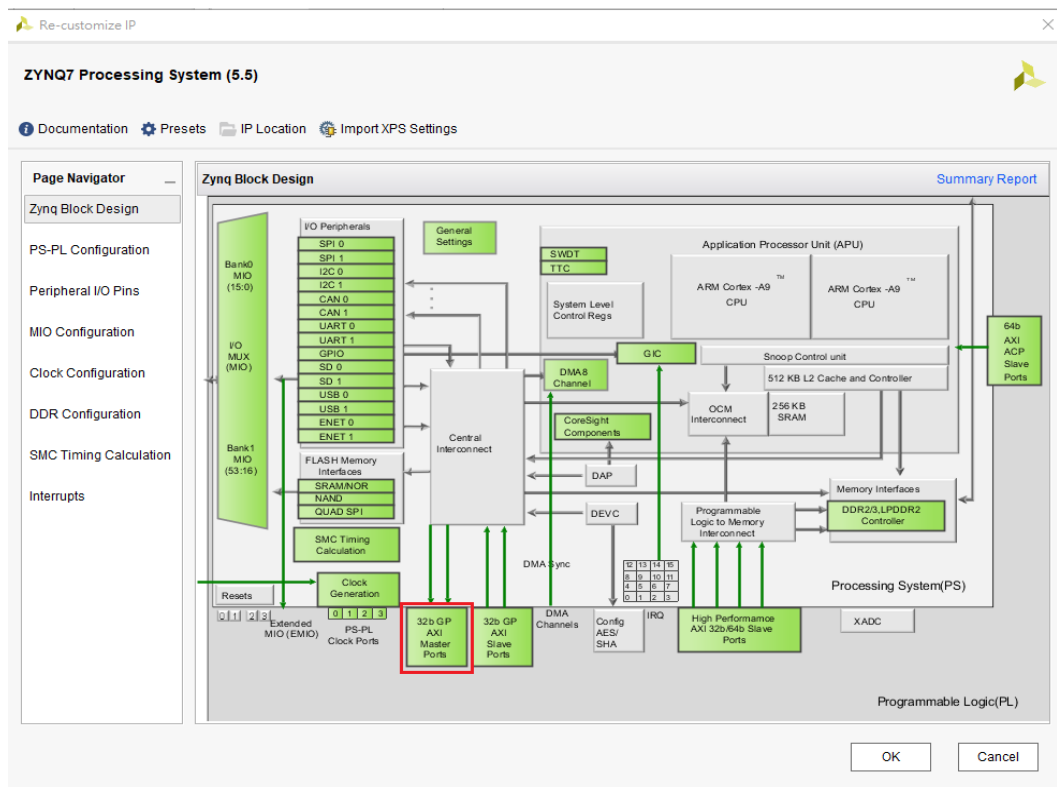
2、開啟專案後，點選 Create Block Design 後按 OK 產生一個空白的 Block Design。



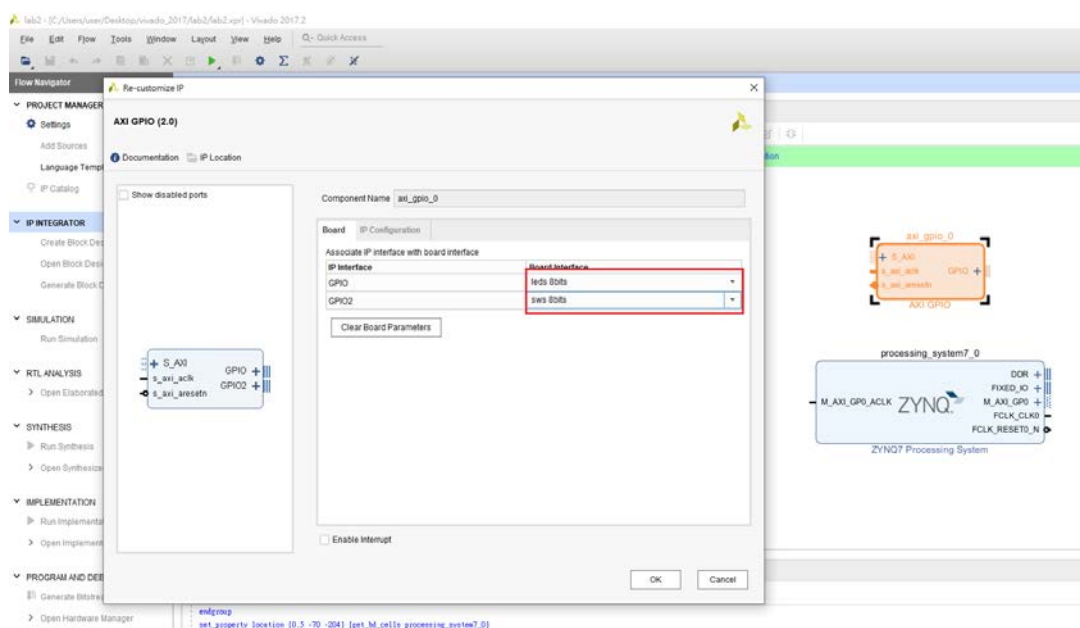
- 3、點選 Block Design 上方的十加入 Zynq processor，為什麼要加入這類 ZYNQ7 Processor System，原因是當我們需要用到 Zynq 上的 Processing System(PS 端)，例如 ARM CPU、on chip memory 等等，以及 Prommable Logic(PL 端)，也就是我們平常說的 FPGA，可供編成的資源。ZYNQ7 Processor System 提供了軟體的 interface，也就是說當我們需要用到軟體來操作硬體架構的時候，就需要這類 ZYNQ7 Processor System。



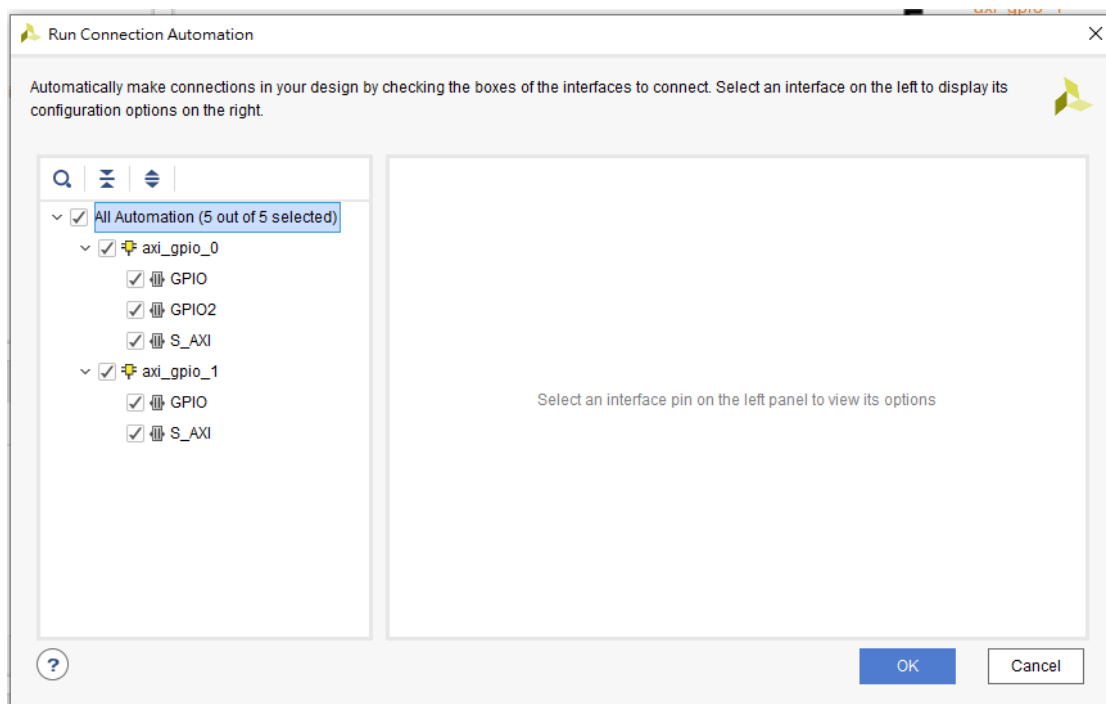
點開 ZYNQ7 Processor System IP，可以看到 Vivado 已經幫我們把 PS 與 PL 端整合起來，而這次我們用到的是 AXI GPIO，屬於 ARM CPU 的 Slave，所以需要開啟 Master Port 去連接，點開發現已經開啟，點擊 Cancel 關閉。



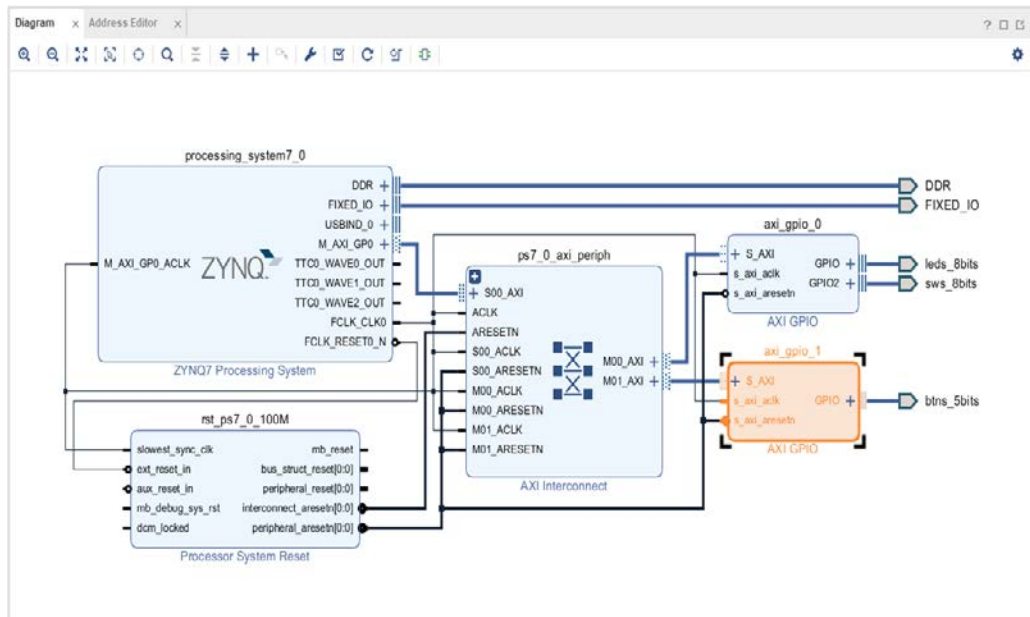
4、同樣點擊+加入2個 AXI GPIO，點開後分別在 Board Interface 加入 leds，sws 以及 btns。



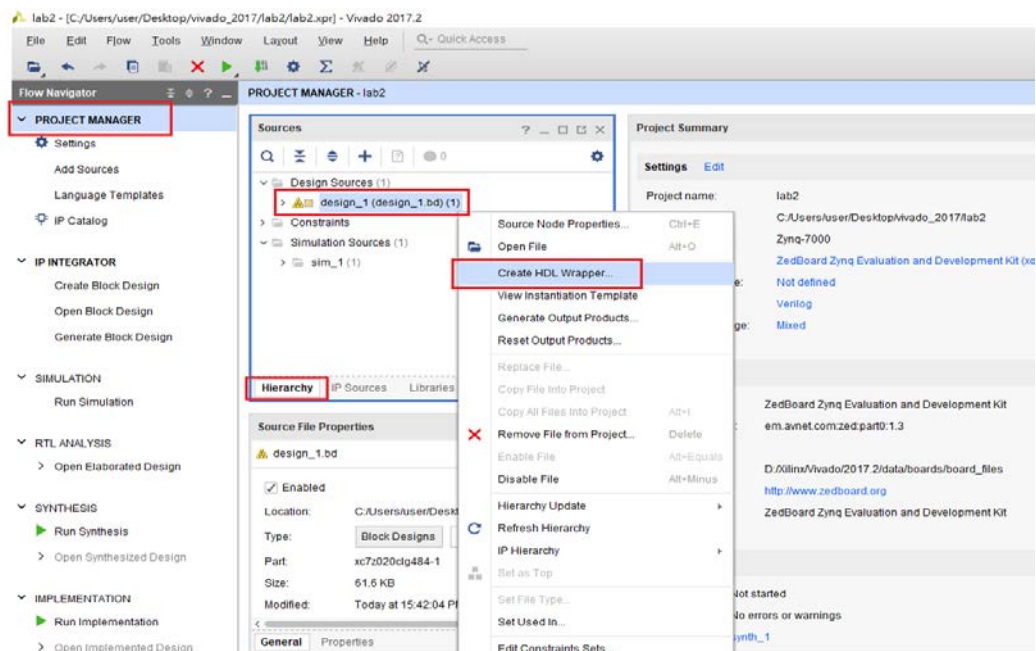
5、IP 呼叫完畢後點選上方 Run Bloc Automation 與 Run Connection Automation 完成 IP 之間的繞線。



- 6、完成後的 Block Diagram 如下圖，其中 processing system 接到 DDR 與 FIX_IO 才可以讓軟體端對 memory 做操作。

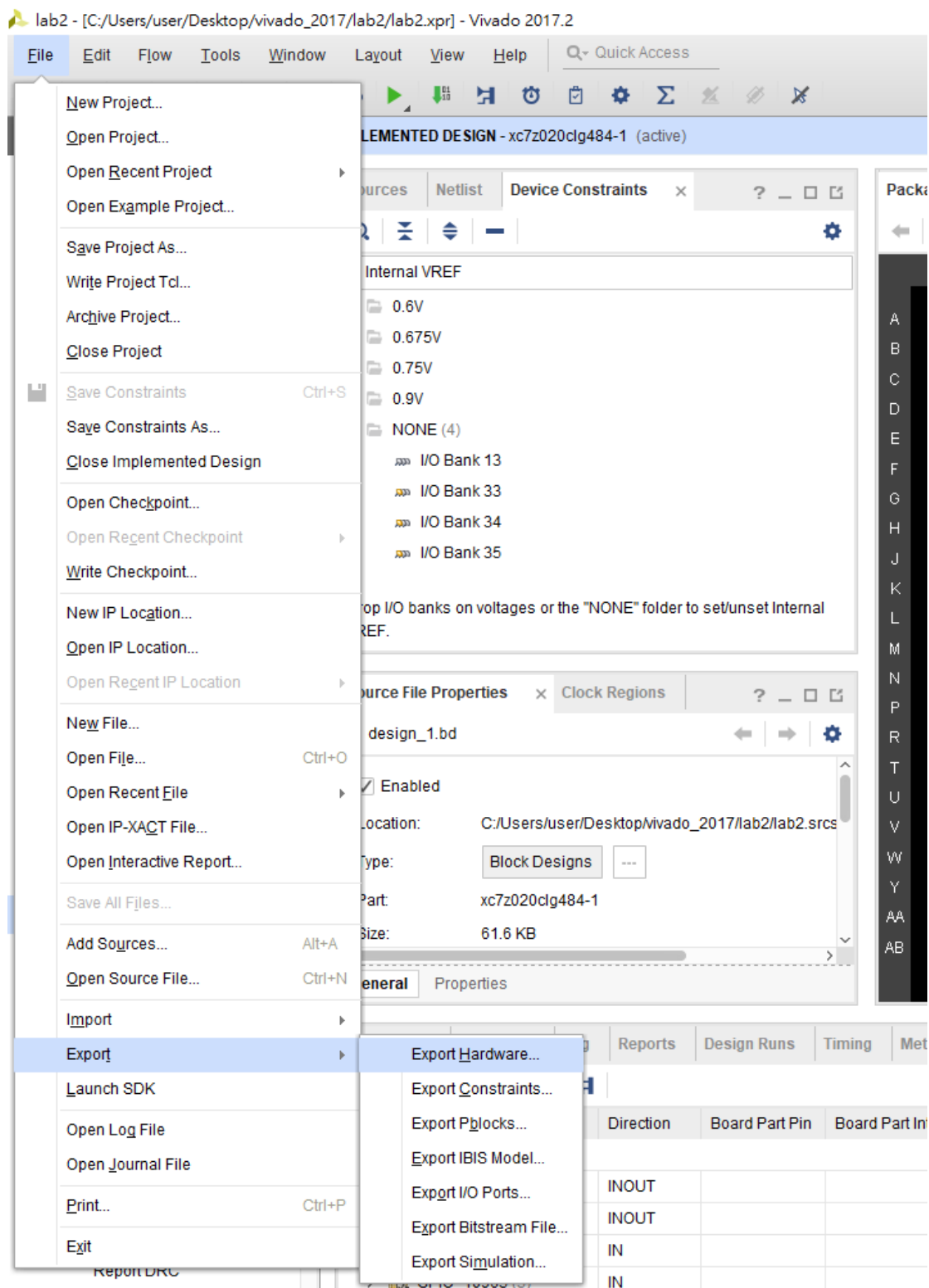


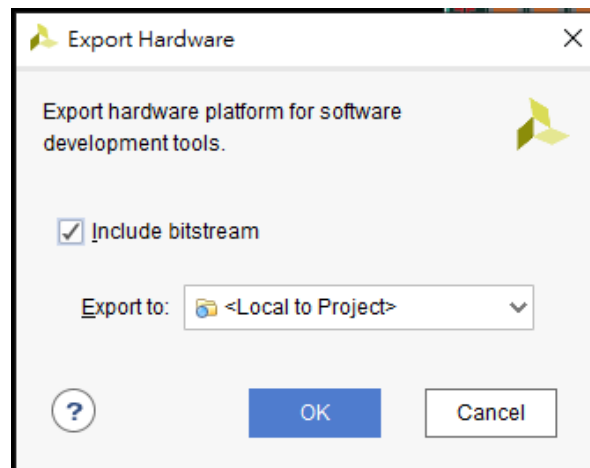
- 7、做完 Block Design 後需要將這些 IP 用 TOP Module 包起來，這樣 Vivado 才知道要以哪個 module 下去做合成。點擊 Project Manager 內 Hierarchy 中你的 design，右鍵點選 Create HDL Wrapper，讓 Vivado 幫我們自動生成最頂層的 verilog code。



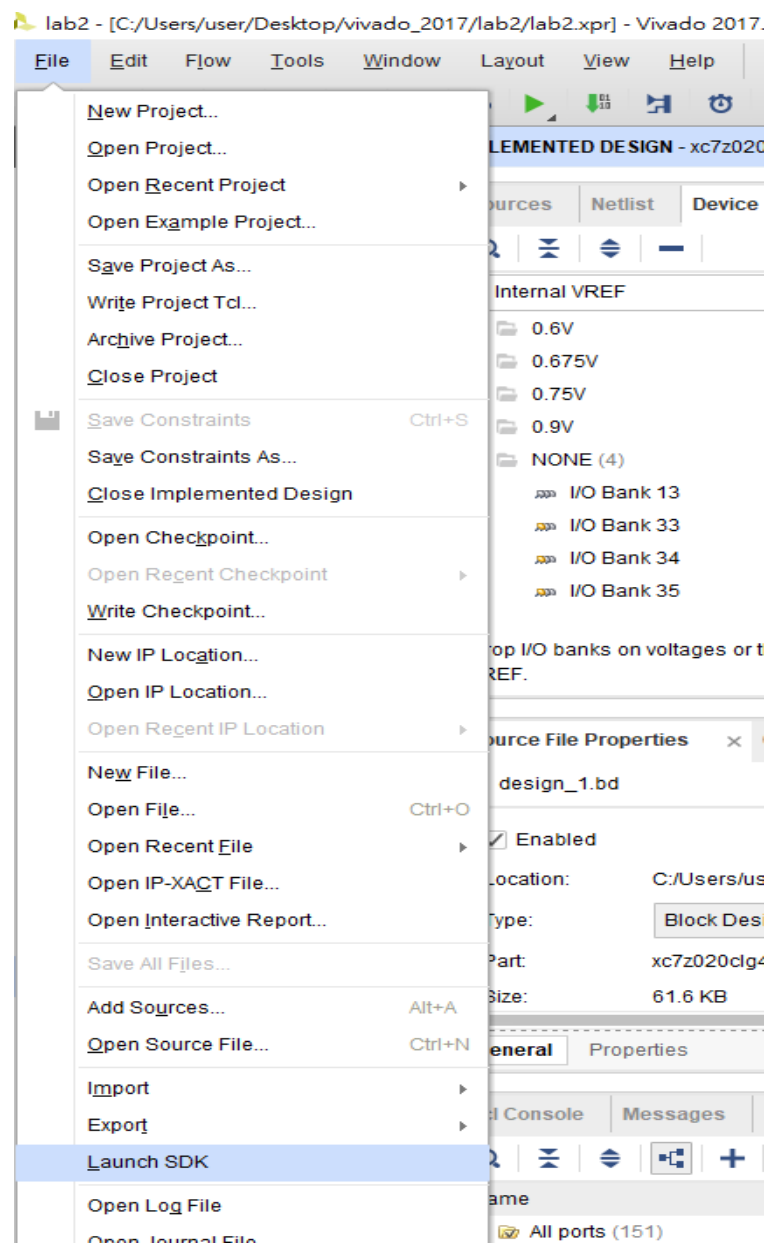
- 8、由於這次是利用軟體端對硬體做操作，所以不需要做 I/O Planning，直接 Generate Bitstream 即可。

9、Bit 檔產生完成後，點選上方 File 中 Export 的 Export Hardware，注意 **Include bitstream** 一定要打勾！





10、之後點選 Launch SDK，按 OK 開啟已經含有我們設計過硬體資訊的 SDK 專案。



- 11、SDK 開啟後如下所示，可以看到我們在 Vivado 使用到的硬體 IP 都被分別對應到一個專屬的 memory 位置，這種方式是 memory map 的方式，也就是說如果我們要去操作 GPIO，只要對它對應到的 memory 位置做操作就可以達成。

system.hdf

design_1_wrapper_hw_platform_0 Hardware Platform Specification

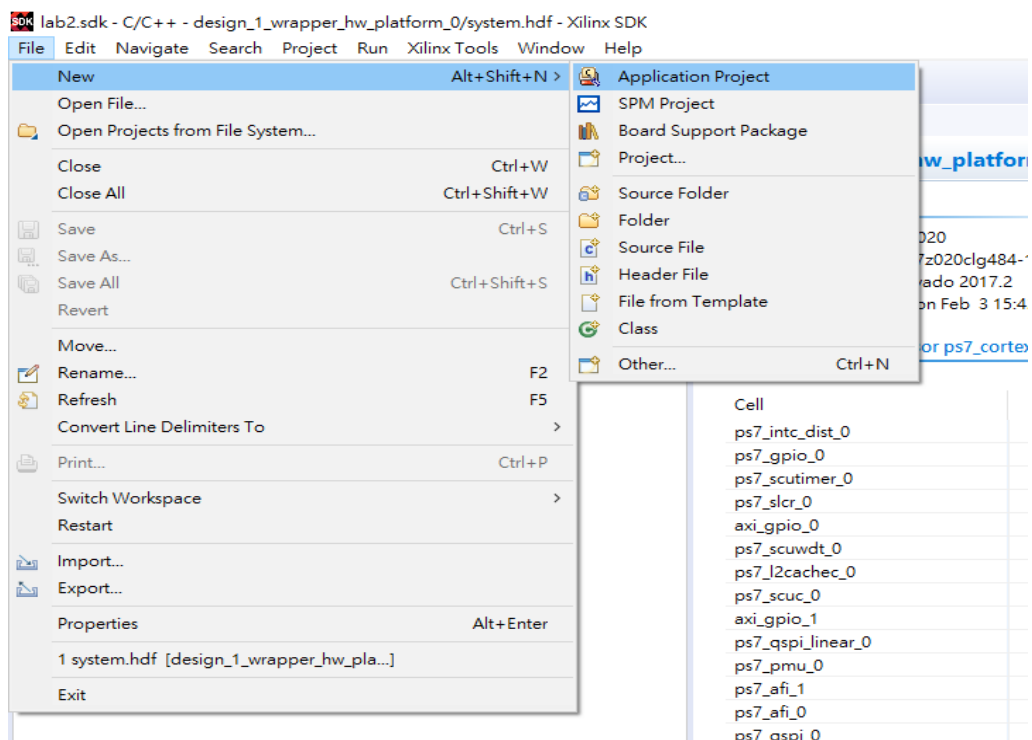
Design Information

Target FPGA Device: 7z020
 Part: xc7z020clg484-1
 Created With: Vivado 2017.2
 Created On: Mon Feb 3 15:42:20 2020

Address Map for processor ps7_cortexa9_0-1]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
axi_gpio_0	0x41200000	0x4120ffff	S_AXI	REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
axi_gpio_1	0x41210000	0x4121ffff	S_AXI	REGISTER
ps7_qspi_linear_0	0xfc000000	0xfcffffff		FLASH
ps7_pmu_0	0xf8893000	0xf8893fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_qspi_0	0xe000d000	0xe000dfff		REGISTER
ps7_usb_0	0xe0002000	0xe0002fff		REGISTER
ps7_afi_3	0xf800b000	0xf800bfff		REGISTER
ps7_afi_2	0xf800a000	0xf800afff		REGISTER
ps7_globaltimer_0	0xf8f00200	0xf8f002ff		REGISTER
ps7_dma_s	0xf8003000	0xf8003fff		REGISTER
ps7_iop bus config 0	0xe0200000	0xe0200fff		REGISTER

- 12、左邊的 hw_platform 是我們從 Vivado 載入的 hardware 檔案，這時我們要再加入 application 寫入操控硬體的軟體程式。



13、Project name 可以隨便取，其他地方都不用改，但要注意 OS Platform 是 standalone，Processor 要是 cortexa_9_0。點選 Next 後選擇 Empty Application 按 Finish，此時我們就有一個空白的 application 了，而產生的 bsp 資料夾裡面放的是 xilinx 根據我們在 Vivado 產生的 hardware 檔自動幫我們生成硬體相關的 library，其中包含每個 IP 對應的 address 以及 ID 等等。

SDK New Project

Application Project
Create a managed make application project.

Project name:

☒ Use default location
Location:
Choose file system:

OS Platform:

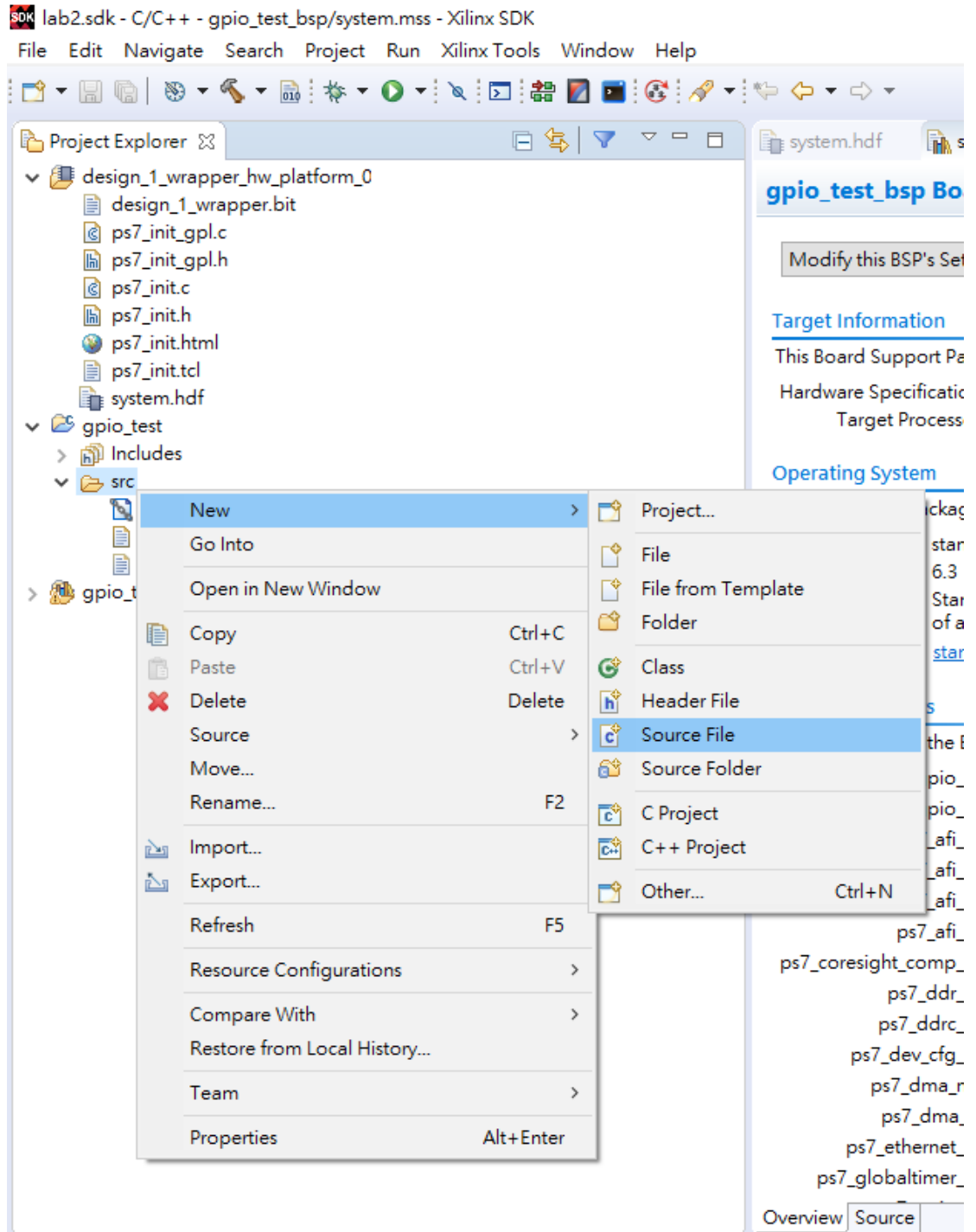
Target Hardware

Hardware Platform:
Processor:

Target Software

Language: ☒ C ☐ C++
Compiler:
Hypervisor Guest:
Board Support Package: ☒ Create New
☐ Use existing

14、接下來在 application 資料夾內的 src 資料夾中加入 C Source File，名字一樣可以隨便取。



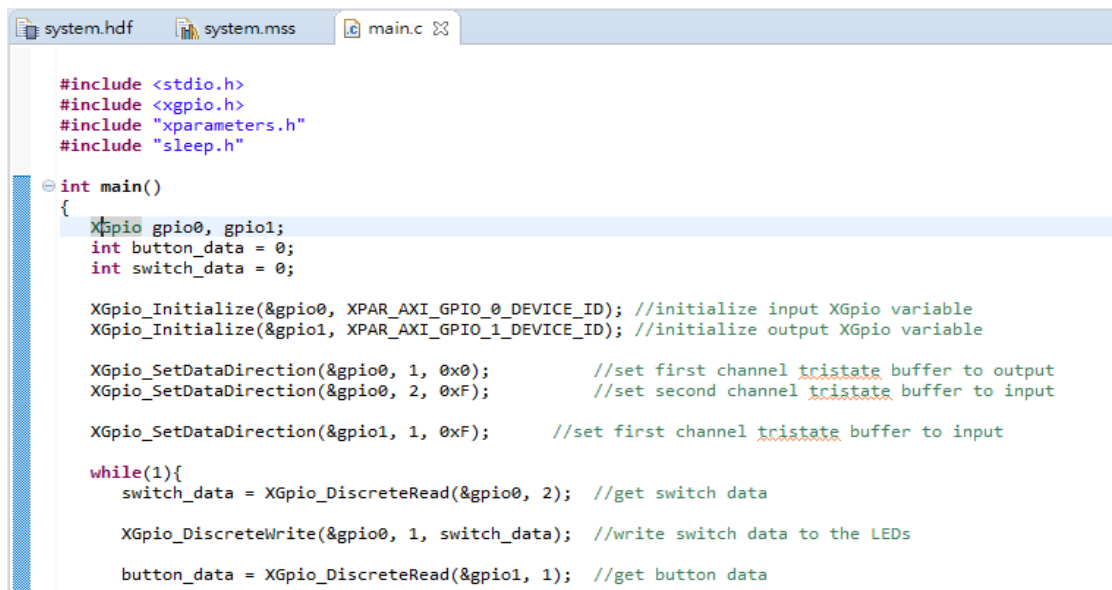
15、接下來就是要編寫控制硬體 GPIO 的 C 程式碼，這邊導入的<xgpio.h>以及<xparameter.h>都是在 bsp 中 xilinx 幫我們做好的 library，讓我們可以更方便去做功能的設計。

這邊簡單說明程式用到的 function 功能，有興趣詳細了解的話可以去閱讀 xilinx 官方的 GPIO 文件，裡面有更詳細的說明。

XGpio_Initialize：將前面宣告的 GPIO 與 vivado 的 GPIO IP 做對應。

XGpio_SetDataDirection：告訴 CPU 每個 GPIO port 的方向，1、2 代表 GPIO IP 的第幾個 port，0x0 表示這個 port 要作為 output 使用，0xF 表示作為 input。

XGpio_DiscreteRead/XGpio_DiscreteWrite：對 GPIO 做讀寫值。



```
#include <stdio.h>
#include <xgpio.h>
#include "xparameters.h"
#include "sleep.h"

int main()
{
    XGpio gpio0, gpio1;
    int button_data = 0;
    int switch_data = 0;

    XGpio_Initialize(&gpio0, XPAR_AXI_GPIO_0_DEVICE_ID); //initialize input XGpio variable
    XGpio_Initialize(&gpio1, XPAR_AXI_GPIO_1_DEVICE_ID); //initialize output XGpio variable

    XGpio_SetDataDirection(&gpio0, 1, 0x0); //set first channel tristate buffer to output
    XGpio_SetDataDirection(&gpio0, 2, 0xF); //set second channel tristate buffer to input

    XGpio_SetDataDirection(&gpio1, 1, 0xF); //set first channel tristate buffer to input

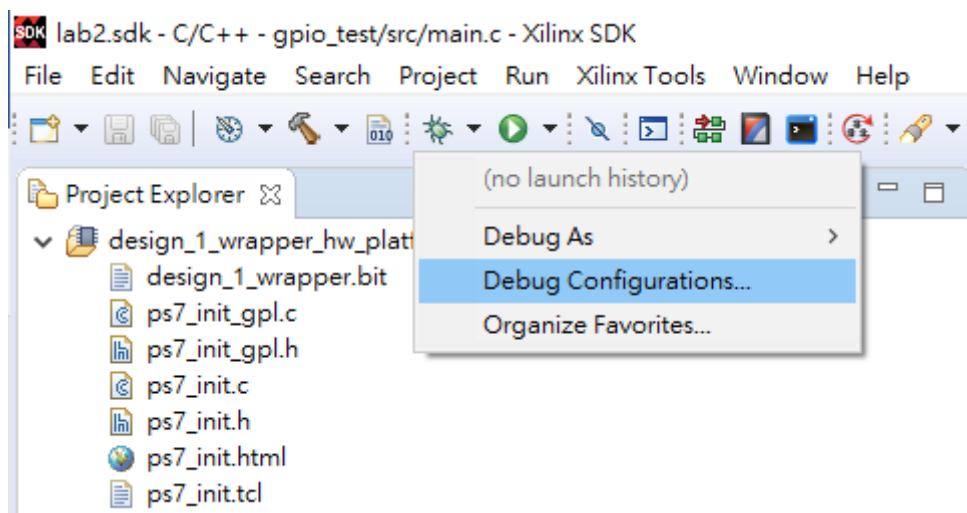
    while(1){
        switch_data = XGpio_DiscreteRead(&gpio0, 2); //get switch data

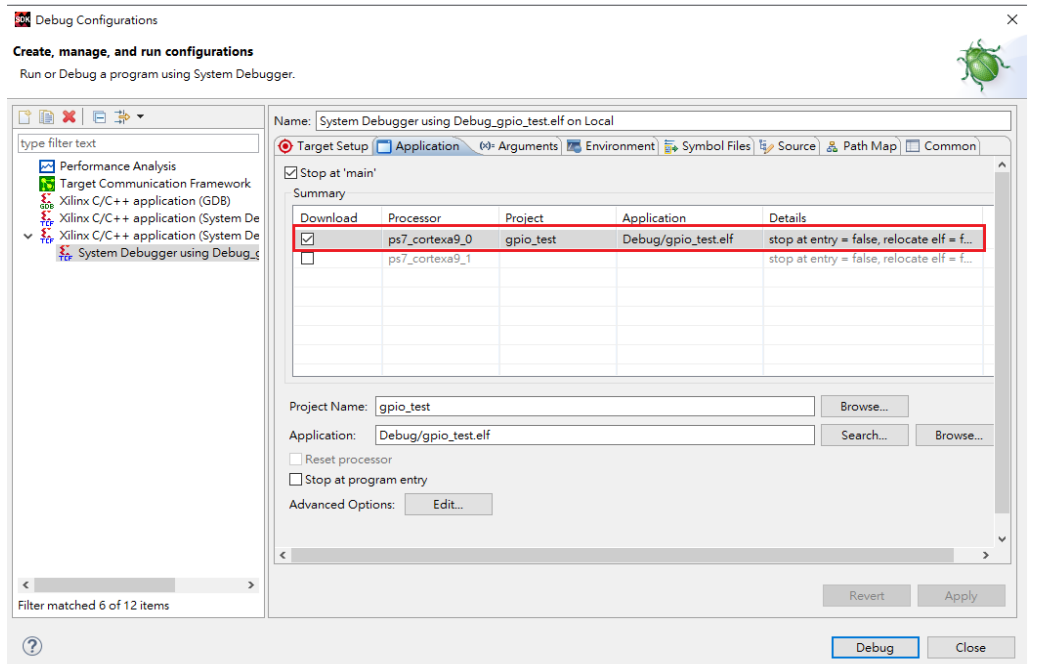
        XGpio_DiscreteWrite(&gpio0, 1, switch_data); //write switch data to the LEDs

        button_data = XGpio_DiscreteRead(&gpio1, 1); //get button data
```

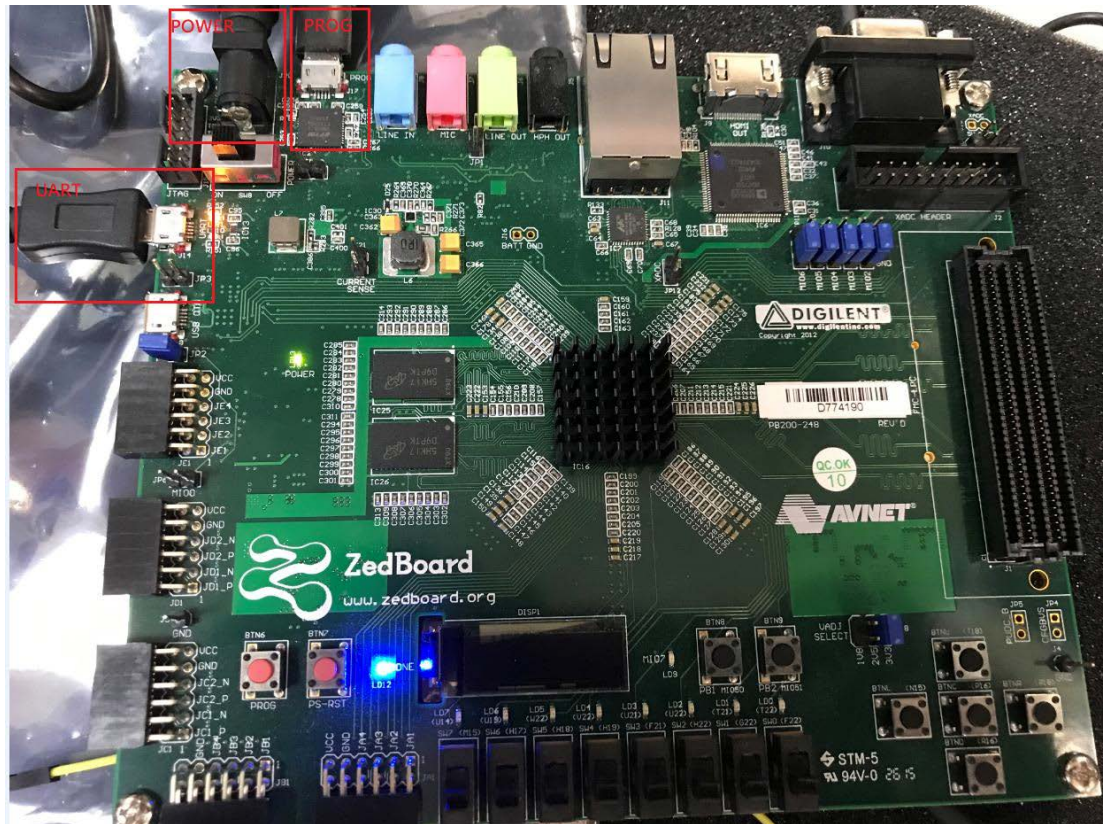
16、完成軟體撰寫後便可以燒入板子做測試，因為我們只是要確認功能是否正確，因此採用 Debug 模式做燒入即可(電路不會永久保存在板子中)。選擇上方

 下拉選單，點選 Debug Configurations。

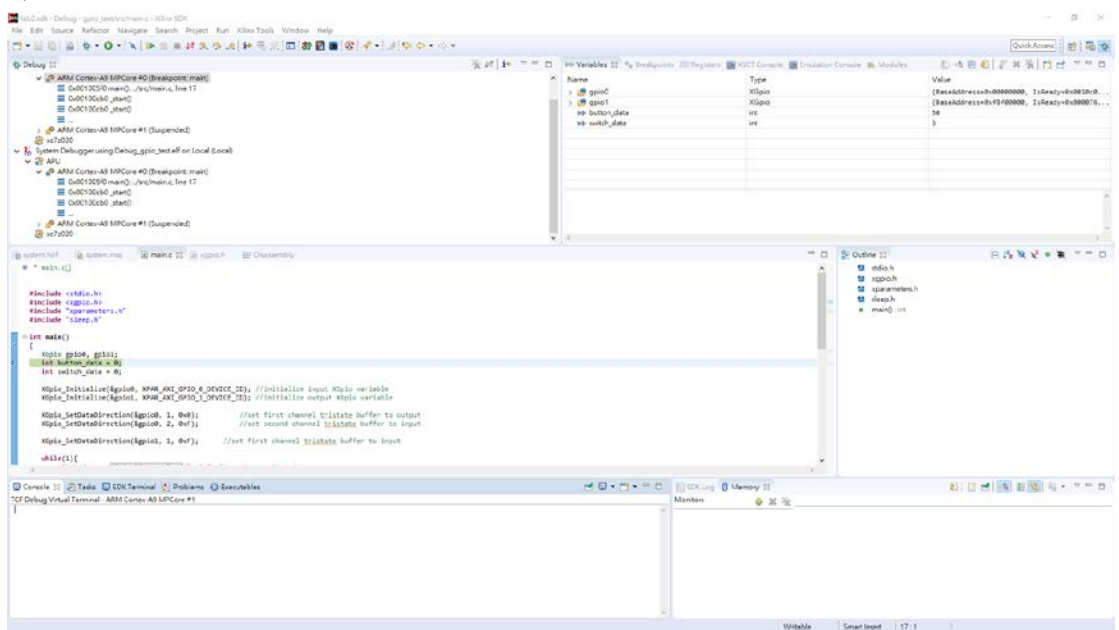




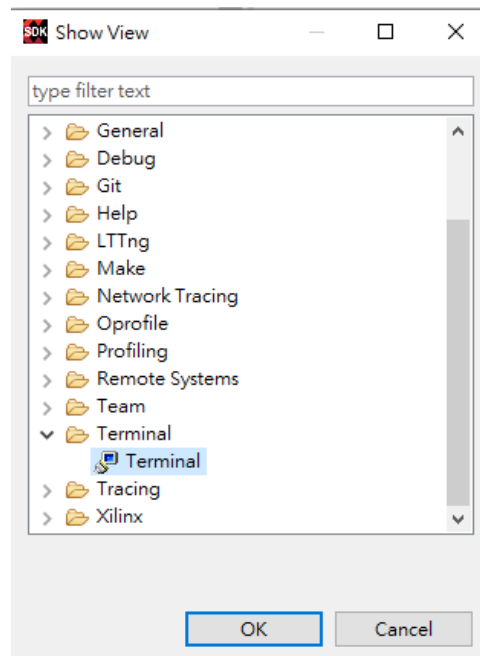
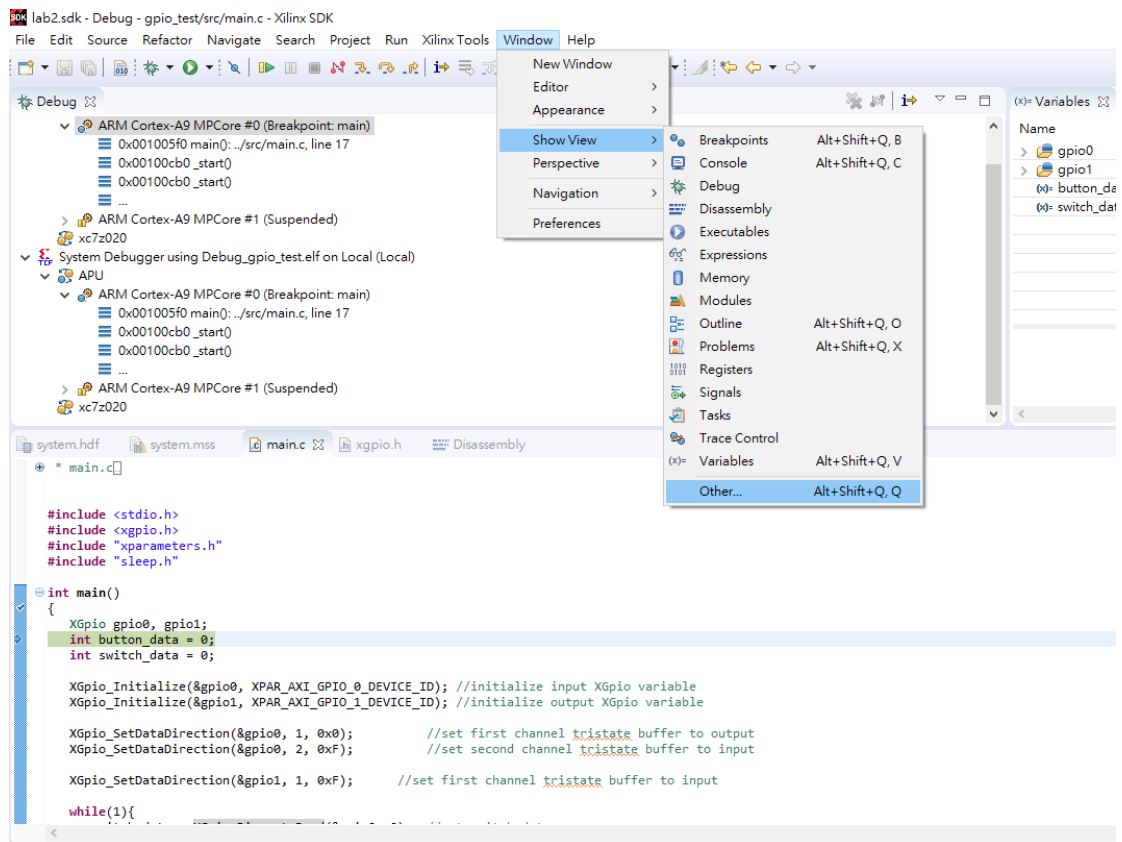
18、接上板子，此次 Lab 因為會從 GPIO 讀入訊息，所以需要將 UART 也接到電腦上。




19、接上且開啟電源後，按下 Debug 將我們設計的電路以及控制的軟體程式一起燒入 ZedBoard 內，此時 SDK 會詢問是否切換到 Debug 介面，點選 OK，畫面會如下所示。

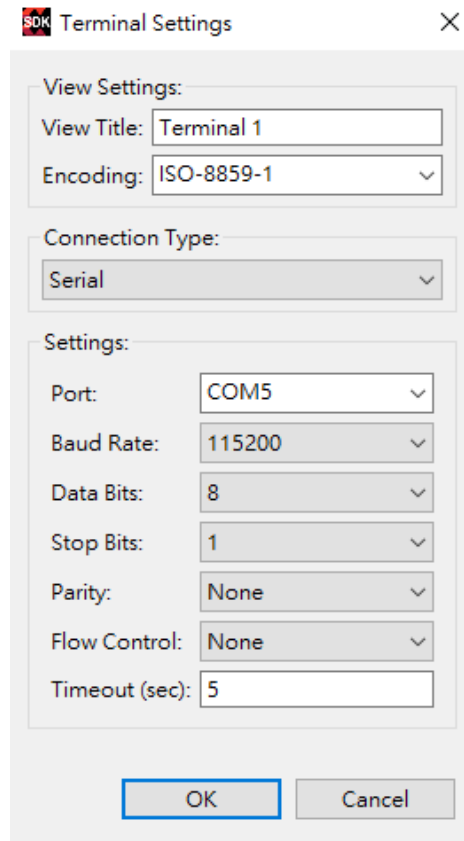



20、接下來我們要開啟終端機去接收從 UART 讀出的資料，這邊可以選擇使用 putty 連接，或是用 SDK 內部提供的終端機。位置在上方 Window，Show View 內的 Other。

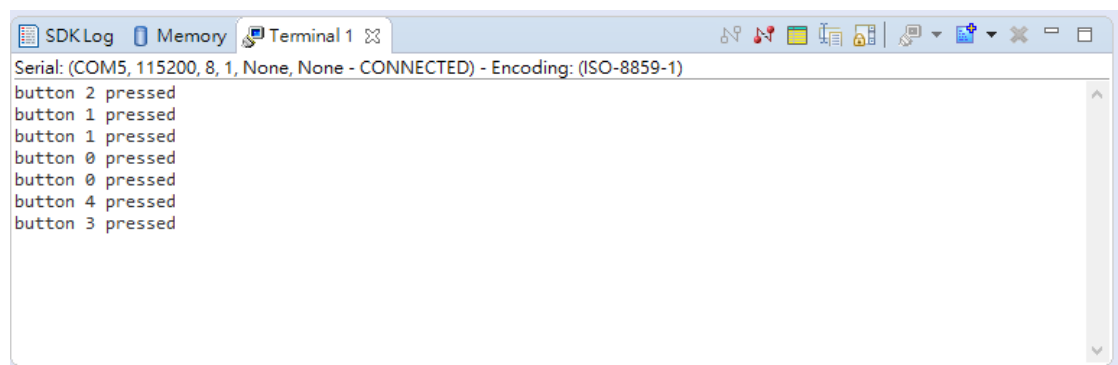


21、開啟後 Terminal 會出現在畫面右下角，點擊做連線設定，其中

Connection Type 選擇 Serial，Port 選擇連接到 UART 的那條線，Baud Rate 固定 115200。



22、點擊上方圖示，我們撰寫的 C code 便會開始執行，這時去撥動板子上的指撥開關即可看到 LED 跟著亮暗，原因是當我們對板子上的開關動作時，AXI GPIO 將這些訊號轉成 AXI 形式傳到 ARM Processor，然後又因為 ZYNQ7 Processor System 提供 software interface，讓我們能透過 C code 來操控這些 IO，因此便可以在讀入開關的訊號後去改變 LED 的值，使得即便沒有在 I/O Planning 做 IO 的 mapping，我們依然透過 CPU 藉由 AXI 訊號來完成 IO 的控制。



D、練習題

與 Lab1 相似，請利用 AXI-GPIO 搭配 SDK 設計一個簡易紅綠燈，其中以 led7 做為紅燈，led6 為黃燈，led5 為綠燈。led0~led4 為倒數計時用 5bit 二進位方式顯示。Sw0 為 0 時，當系統啟動時預設綠燈運作時間 15 秒，黃燈 1 秒，紅燈 16 秒，Sw0 為 1 時，綠燈運作時間 7 秒，黃燈 1 秒，紅燈 8 秒。(此練習不可使用 I/O Planning)