

國立清華大學
系統晶片設計
SOC Design



國立清華大學
NATIONAL TSING HUA UNIVERSITY

Lab 1

系所級:電子所二年級

學號:111063548

姓名:蕭方凱

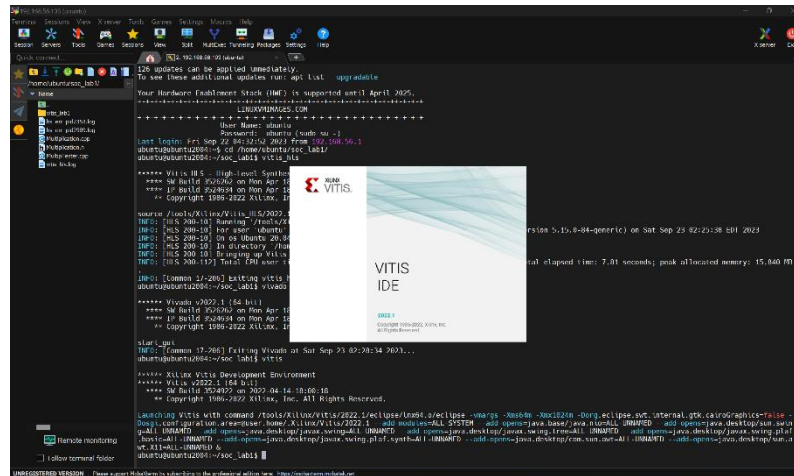
指導老師:賴瑾教授

目錄

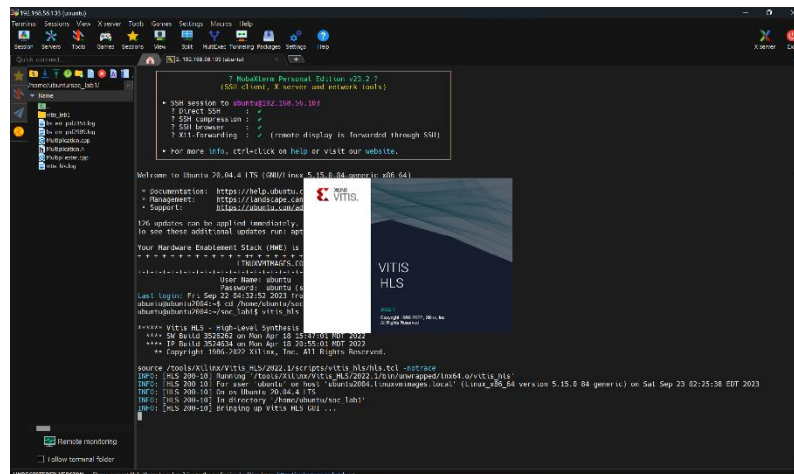
I.	Xilinx Tools Installation.....	3
1.	Vitis Installation	3
2.	Vitis_HLS Installation	3
3.	VIVADO Installation	3
II.	Vitis_HLS	4
1.	Multiplication.cpp	4
2.	Multiplication.h.....	4
3.	MultipTester.cpp	5
III.	VIVADO	7
1.	匯入 vitis_hls 的 IP (MULTIP_2NUM):.....	7
2.	加入 ZYNQ7 Processing System 並設定 PLL Fabric clock 為 100MHz:7	
3.	加入由 vitis_hls 匯出的 IP multip_2num , 並 run block automation.....	8
4.	Run Connection Automation.....	8
5.	Create HDL Wrapper	8
6.	Generate FPGA Bitstream.....	9
IV.	PYNQ/Host Program	11
1.	租借 online FPGA	11
2.	開啟 jupyter notebook , 並上傳 Multip2Num.bit, Multip2Num.hwh, Multip2Num.ipynb.....	11
3.	執行 Multip2Num.ipynb.....	12
V.	Discussion	13

I. Xilinx Tools Installation

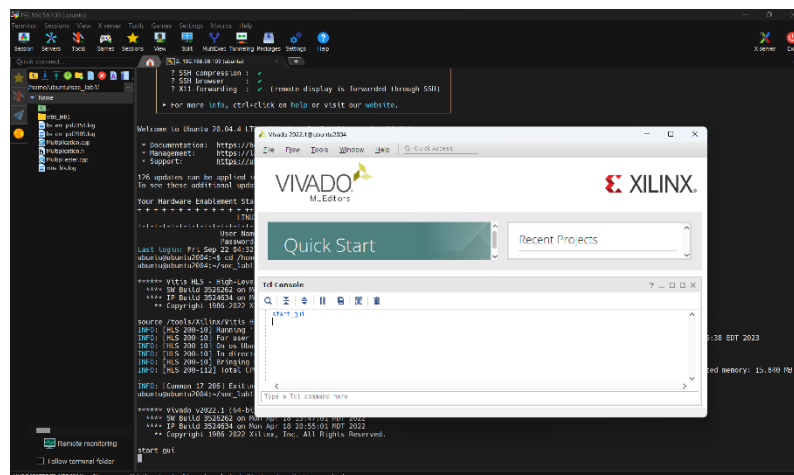
1. Vitis Installation



2. Vitis_HLS Installation

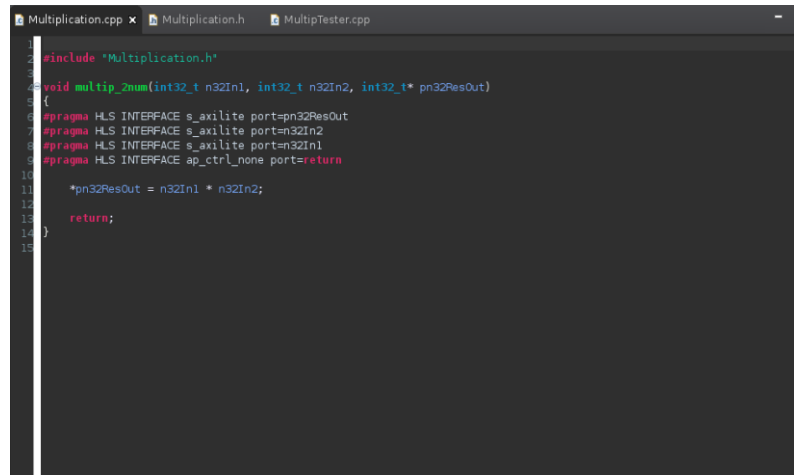


3. VIVADO Installation



II. Vitis_HLS

1. Multiplication.cpp

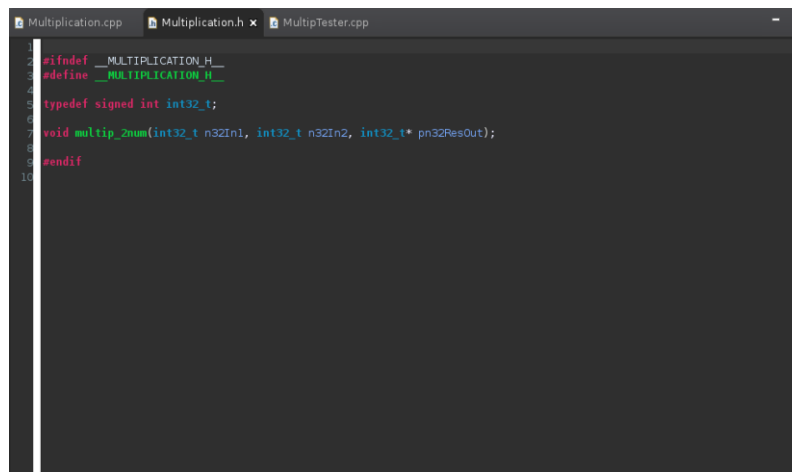


```
1 #include "Multiplication.h"
2
3 void multip_2num(int32_t n32In1, int32_t n32In2, int32_t* pn32ResOut)
4 {
5     #pragma HLS INTERFACE s_axilite port=pn32ResOut
6     #pragma HLS INTERFACE s_axilite port=n32In2
7     #pragma HLS INTERFACE s_axilite port=n32In1
8     #pragma HLS INTERFACE ap_ctrl_none port=return
9
10     *pn32ResOut = n32In1 * n32In2;
11     return;
12 }
```

這是一 c++ 語言，其功能為主要為將 `int32_t` 型別的輸入參數 `n32In1`, `n32In2` 做乘積運算，並將運算結果指向 `pn32ResOut` 所指向的記憶體位置，而不是只有指向指標 `pn32ResOut`。

而 `#pragma HLS INTERFACE s_axilite` 是一種介面類型，將函式連接到 AXILite 介面，用於 FPGA 與外部通信的協議。

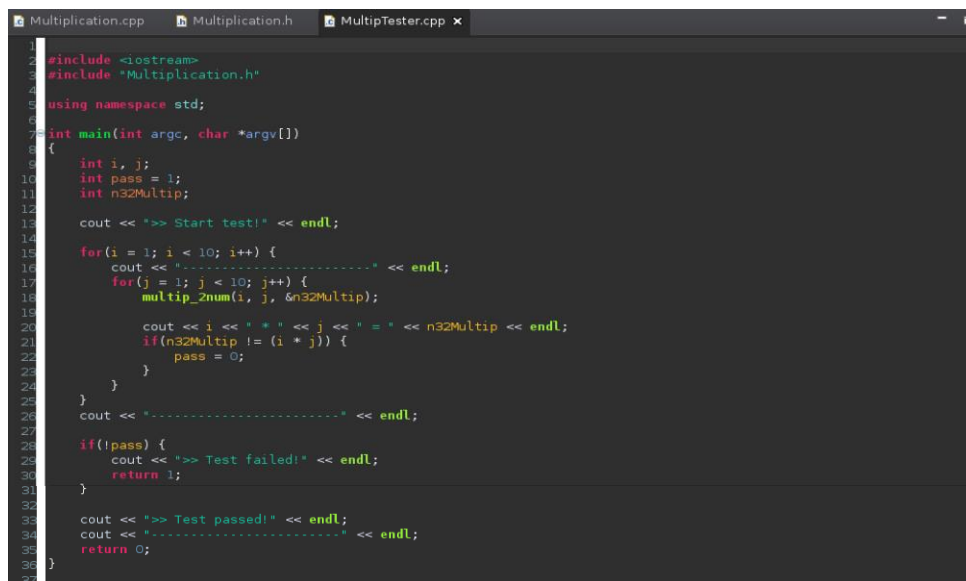
2. Multiplication.h



```
1 #ifndef __MULTIPLICATION_H__
2 #define __MULTIPLICATION_H__
3
4 typedef signed int int32_t;
5
6 void multip_2num(int32_t n32In1, int32_t n32In2, int32_t* pn32ResOut);
7
8 #endif
```

此圖標頭檔，標頭檔是程式與程式之間的介面，以便多個 .cpp 程式檔案可以共享和使用這些元素，而 `ifndef&define` 兩行是為確保 .h 標頭檔只會執行一次。

3. MultipTester.cpp



```
1
2 #include <iostream>
3 #include "Multiplication.h"
4
5 using namespace std;
6
7 int main(int argc, char *argv[])
8 {
9     int i, j;
10    int pass = 1;
11    int n32Multip;
12
13    cout << ">> Start test!" << endl;
14
15    for(i = 1; i < 10; i++) {
16        cout << "-----" << endl;
17        for(j = 1; j < 10; j++) {
18            multip_2num(i, j, &n32Multip);
19
20            cout << i << " * " << j << " = " << n32Multip << endl;
21            if(n32Multip != (i * j)) {
22                pass = 0;
23            }
24        }
25    }
26    cout << "-----" << endl;
27
28    if(!pass) {
29        cout << ">> Test failed!" << endl;
30        return 1;
31    }
32
33    cout << ">> Test passed!" << endl;
34    cout << "-----" << endl;
35    return 0;
36 }
```

此為 c++ 語言所寫出類似 testbench 的功能，透過兩個 for 迴圈搭配 i, j 變數得到輸入 pattern，接著去驗證 multip_2num 函式(輸入 i, j)的運算結果是否等於 i*j，只要有一個不相等，pass 就會變成 0，導致顯示 Test failed。

當三份檔案(multipicaiton.cpp, multiplication.h, multiptester.cpp)都準備好後，便可以開始執行 simulation→synthesis→cosimulation。要特別注意的是在跑 cosimulation 協同模擬時，必須要將「#pragma HLS INTERFACE ap_ctrl_none port=return」註解，查了一些資料以後，推測是因 return 有可能會被當成控制訊號，進而導致模擬結果錯誤。Cosimulation 只支持三種類型的 ap_ctrl_none 設計，如下：

- (1) combinational designs：純粹的組合邏輯設計，不涉及時序或流水線處理
- (2) pipelined design with II of 1：具有流水線架構的設計，其中 II 設置為 1，表示每個時鐘週期只有一個操作被執行。這可以用於一些需要時序控制的設計
- (3) designs with array streaming or hls_stream or AXI4 stream ports：
表示設計使用了特定的流式數據傳輸機制，例如 Vivado HLS 中的數組流或 hls_stream，或者是 AXI4 流介面

執行完 simulation→synthesis→cosimulation 後，會產生三個檔案，csynth.rpt, multip_2num_csim.log, multip_2num_csynth.rpt。

(1) csynth.rpt

此 report 內容為 C++ 代碼轉換為硬體的結果和相關資訊，包含 SW I/O Information, HW Interfaces 等，舉例來說，SW I/O Information 用於描述和記錄軟體的輸入和輸出（I/O）相關信息，HW Interfaces 則是硬體設計中的各種接口資訊。

(2) multip_2num_csim.log

此 log 檔類似執行完 testbench(multiptester.cpp)後的輸出，其主要是模擬輸出並檢查模擬是否正確。

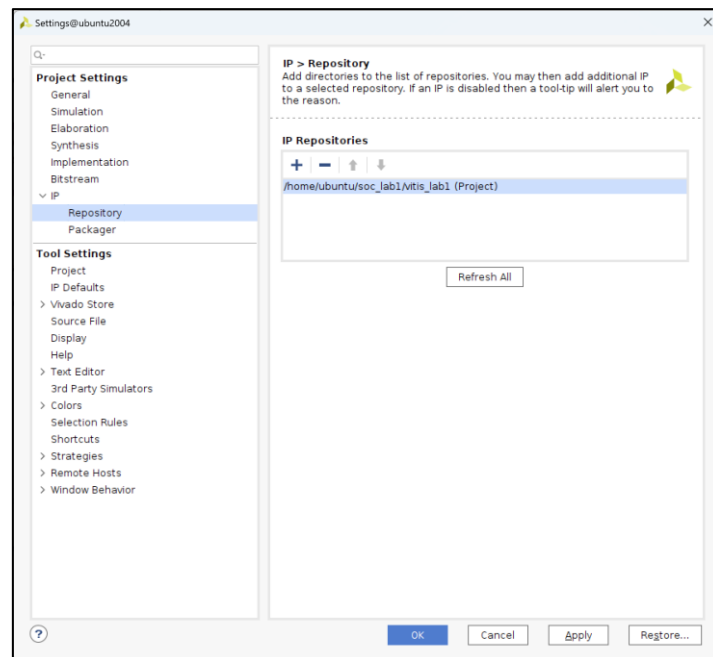
(3) multip_2num_csynth.rpt

與 csynth.rpt 不同，此 report 檔是針對特定函數 multip_2num 的報告，其內容包含時序分析(timing, latency)、硬體資源使用情況等。

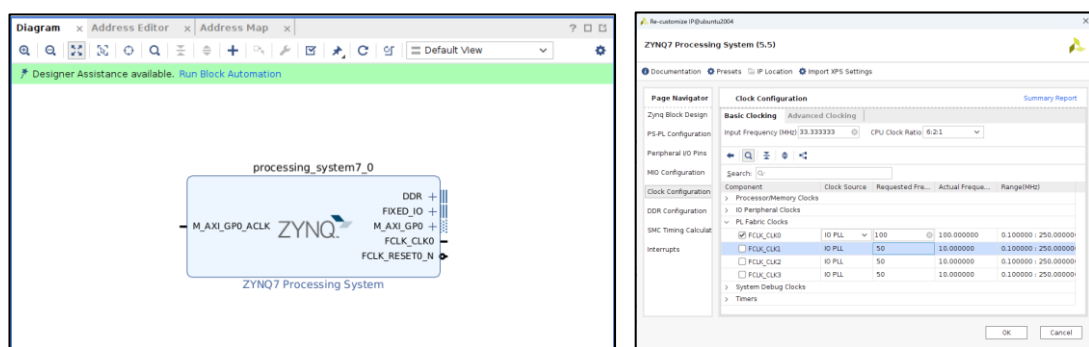
完成 IP 設計後，最後 export rtl 以供 vivado 做使用，Vivado 會需要匯入由 Vitis HLS 匯出的 IP，另外，由 C++ 語言轉換的 RTL code 會出現在 “solution1/impl/Verilog” 中的 multip_2num.v。

III. VIVADO

1. 匯入 vitis_hls 的 IP (MULTIP_2NUM):

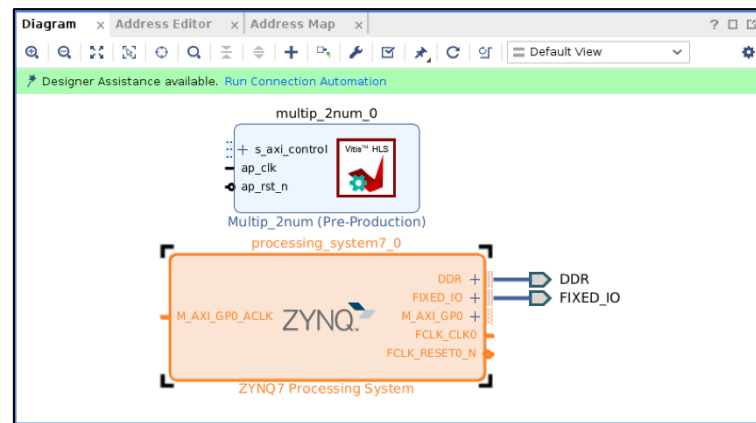


2. 加入 ZYNQ7 Processing System 並設定 PLL Fabric clock 為 100MHz:

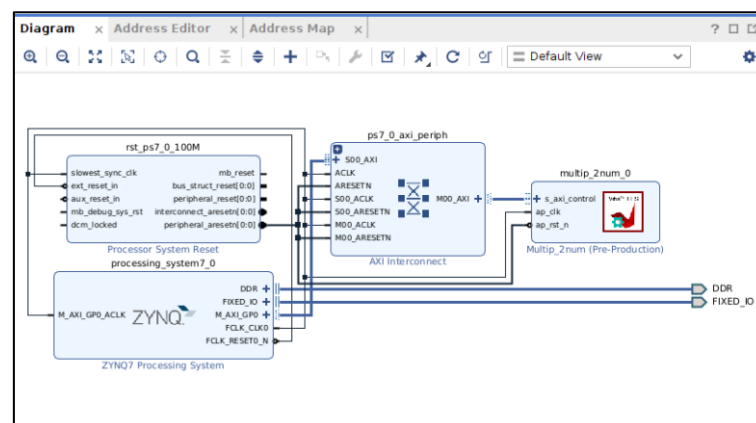


ZYNQ 是一種可編程 SOC，整合了 ARM 雙核 cortex-A9 處理器和 Xilinx 7 系列 FPGA 架構，內部包含 Processing System 及 Programmable Logic。至於上方設定的 PLL Fabric clock，是指可程式邏輯區域中的相位鎖定迴路所產生的時鐘訊號，由 processing system 產生並提供給 programming logic 使用，當提高 frequency 時候，在沒有 timing violation 的情況下，可提高 IP Core 在硬體上的運行速度。

3. 加入由 vitis_hls 匯出的 IP multip_2num，並 run block automation



4. Run Connection Automation



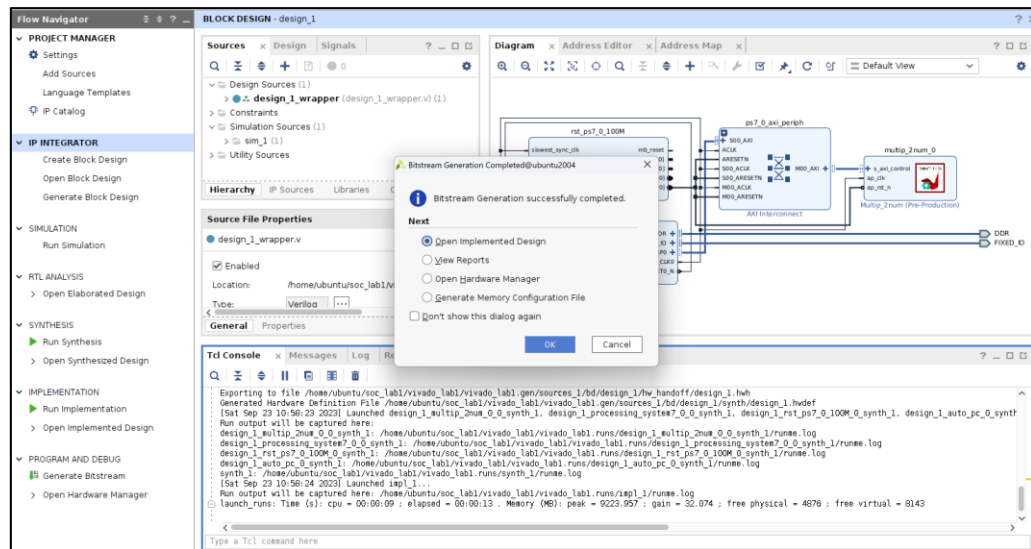
Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/multip_2num_0/s_axi_control	s_axi_control	Reg	0x4000_0000	64K	0x4000_FFFF

執行完 connection automation 後可切到 address editor 查看 Memory map

5. Create HDL Wrapper

這一部非常重要，Create HDL Wrapper 在 FPGA 或 ASIC 設計流程中創建一個 RTL 的封裝（wrapper）模塊，該封裝模塊將自行設計的模塊與 FPGA 或 ASIC 的外部界面進行連接。

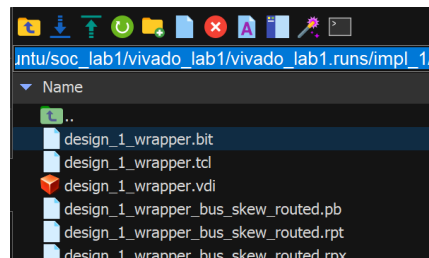
6. Generate FPGA Bitstream



Generate 完成後會產生兩個檔案，Multip2Num.bit 及 Multip2Num.hwh，以提供 FPGA 做使用。

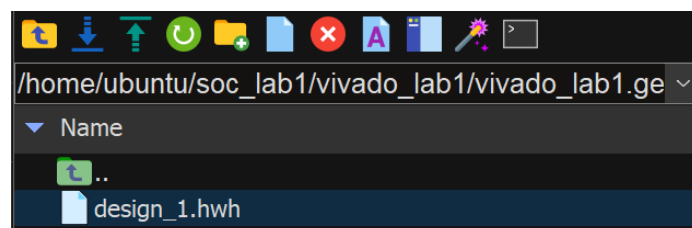
(1) Multip2Num.bit:

.bit 文件是 FPGA 的位流 (Bitstream) 文件，包含了將硬體設計配置到 FPGA 的資訊，包括邏輯閘配置、連接、clock 設定等。



(2) Multip2Num.hwh:

.hwh 文件是硬體描述文件 (Hardware Description File)，它包含了有關 FPGA 設計的元數據 (Metadata)，不包含實際數據。這些元數據可能包括設計的版本、配置信息、接口定義等。



```
cp ./vivado_lab1/vivado_lab1.runs/impl_1/design_1_wrapper.bit ./Multip2Num.bit
cp ./vivado_lab1/vivado_lab1.gen/sources_1/bd/design_1/hw_handoff/design_1.hwh .
./Multip2Num.hwh 後，會在 soc_lab1 目錄下得到兩檔案，分別為 Multip2Num.bit
及 Multip2Num.hwh，如下圖：
```

```
home/ubuntu/soc_lab1/
▼ Name
├── ..
├── vits_lab1
├── vivado_lab1
├── hls_err_psd2538.log
├── hls_err_psd2588.log
├── hls_err_psd2989.log
├── Multip2Num.bit
├── Multip2Num.hwh
├── Multiplication.cpp
├── Multiplication.h
├── MultipTester.cpp
├── vitsg_hls.log
├── vivado_jou
├── vivado.log
├── vivado_2661.backup.jou
├── vivado_2661.backup.log
└── vivado_psd201.str

? MobaXterm Personal Edition v23.1.2
(SSH client, X server and network tools)

▶ SSH session to ubuntu@192.168.56.103
  ? Direct SSH: ✓
  ? SSH compression: ✓
  ? SSH-browser: ✓
  ? X11-forwarding: ✓ (remote display is forwarded through SSH)

▶ For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.15.0-84-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

128 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

+++++
LINUXVMIMAGES.COM
+++++
User Name: ubuntu
Password: ubuntu (sudo su -)
ubuntu@ubuntu2004:~$ cd soc_lab1/
ubuntu@ubuntu2004:~/soc_lab1$ cp ./vivado_lab1/vivado_lab1.runs/impl_1/design_1_wrapper.bit ./Multip2Num.bit
ubuntu@ubuntu2004:~/soc_lab1$ cp ./vivado_lab1/vivado_lab1.gen/sources_1/bd/design_1/hw_handoff/design_1.hwh ./Multip2Num.hwh
ubuntu@ubuntu2004:~/soc_lab1$
```

IV. PYNQ/Host Program

1. 租借 online FPGA

```
[1] list all online device boards
[2] rent a specified device board
[3] return your rented device board
[0] exit OnlineFPGA service

please enter your option:
>> 2

[1] pynq
[2] kv260

please enter your option:
>> 1

[1] rent a device board by choice
[2] rent a device board by assignment

please enter your option:
>> 1
all online device boards
{'device': 'pynq_01', 'status': 'available'}
{'device': 'pynq_04', 'status': 'available'}
{'device': 'pynq_05', 'status': 'available'}
{'device': 'pynq_06', 'status': 'available'}
{'device': 'pynq_07', 'status': 'unknown'}
{'device': 'pynq_09', 'status': 'available'}
{'device': 'pynq_11', 'status': 'available'}
{'device': 'pynq_12', 'status': 'used'}
{'device': 'pynq_13', 'status': 'used'}
{'device': 'pynq_14', 'status': 'unknown'}
{'device': 'pynq_16', 'status': 'unknown'}
{'device': 'pynq_17', 'status': 'available'}
{'device': 'pynq_18', 'status': 'unknown'}

please enter pynq device name which you want to rent:
>> pynq_01
device pynq_01 is available
do you want to rent this device? (y/n)
>> y
user zeus950068@gmail.com rented device pynq_01 successfully
jupyter web ip port is 140.112.207.200:20100, web passwd is Plbzvm and timeup at 09/24/2023 03:14:58
```

2. 開啟 jupyter notebook，並上傳 Multip2Num.bit, Multip2Num.hwh, Multip2Num.ipynb

jupyter			Quit	Logout
Files	Running	Clusters		
Select items to perform actions on them.			Upload	New
<input type="checkbox"/>	0	/	Name	Last Modified
<input type="checkbox"/>	getting_started			File size
<input type="checkbox"/>	logictools			6 個月前
<input type="checkbox"/>	pynq_composable			7 天前
<input type="checkbox"/>	pynq_peripherals			1 年前
<input type="checkbox"/>	Multip2Num.ipynb		Running	2 年前
<input type="checkbox"/>	Multip2Num.bit		9 分鐘前	5.62 kB
<input type="checkbox"/>	Multip2Num.hwh		9 分鐘前	4.05 MB
<input type="checkbox"/>			9 分鐘前	150 kB

3. 執行 Multip2Num.ipynb

```
In [1]: 1 # coding: utf-8
2 # In[ ]:
3
4
5
6
7 from __future__ import print_function
8
9 import sys, os
10
11 sys.path.append('/home/xilinx')
12 os.environ['XILINX_XRT'] = '/usr'
13 from pynq import Overlay
14
15 if __name__ == "__main__":
16     print("Entry:", sys.argv[0])
17     print("System argument(s):", len(sys.argv))
18
19     print("Start of \"\" + sys.argv[0] + \"\"")
20
21     ol = Overlay("/home/xilinx/jupyter_notebooks/Multip2Num.bit")
22     regIP = ol.multip_2num_0
23
24     for i in range(9):
25         print("=====")
26         for j in range(9):
27             regIP.write(0x10, i + 1)
28             regIP.write(0x18, j + 1)
29             Res = regIP.read(0x20)
30             print(str(i + 1) + " * " + str(j + 1) + " = " + str(Res))
31         print("=====")
32     print("Exit process")
33
34
Entry: /usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py
System argument(s): 3
Start of "/usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py"
=====
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
=====
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
=====
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
=====
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
=====
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
=====
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
=====
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
=====
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
=====
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
=====
Exit process
```

V. Discussion

Q1. 為何#pragma HLS INTERFACE ap_ctrl_none port=return 不註解的話會無法 cosimulation?

ANS:

當使用 interface protocol ap_ctrl_none 時，不會將 block-level I/O protocol 添加到設計中，只會有 clock, reset, data ports。沒有 ap_done 信號，接收來自 ap_return port 的數據的 consumer block 將無法知道數據何時有效。

Q2. 為何 run block automation 後，ZYNQ7 會多出兩個 port 接腳(DDR, FIXED_IO)?

ANS:

DDR port 通常用於連接外部的 DDR SDRAM 存儲器，當運行 Run Block Automation 時，自動生成 DDR port 以支援 DDR 存儲器的連接和操作。

FIXED_IO port 通常與 FPGA 的固定引腳 (Fixed I/O Pins) 有關，此固定引腳通常用於連接到外部硬體設備，如顯示器、按鈕、LED 等，生成 FIXED_IO 端口的目的是確保可以設置這些引腳的功能並連接。

Q3. Run connection automation 後，多出來的 components 是什麼?

ANS:

多出來的 component 是 processor system reset 以及 AXI interconnect，Processor System Reset 是一個用於控制和管理處理器(通常是 ARM Cortex-A 等)的組件。

負責處理器的初始化和重置，以確保處理器能夠正確執行軟體。AXI Interconnect 是一個用於連接不同硬體模塊的組件，確保數據在不同 block 之間可靠地流動。

Q4. Bitstream 是什麼?.hwh 檔案是什麼?

ANS:

Bitstream 是 FPGA 設計的二進制配置檔案，其可將 FPGA 中的可編程邏輯開陣列配置為特定硬體功能，使 FPGA 得以實現特定的硬體功能。

Comment:

透過這次 lab1，我學會了一套從軟體語言(c++)到 FPGA 上模擬的流程，使用軟體而非硬體去編成的好處在於，當電路功能很大時用軟體去描述會比硬體來的更方便更快速一些，透過 vitis_hls 層層轉換輸出得到 IP，再匯入進去 vivado 配置 clock, connections, gate, etc.，最後產生 bitstream 及.hwh 檔案，即可在 FPGA 上面進行模擬觀察電路功能是否符合預期，有無正常運作。

在做 LAB 的過程有許多名詞是之前沒看過的，例如前面的 ap_ctrl_none，其用意是什麼?Processing System 及 Programming Logic 之間的關係又是什麼?有許多問題還是需要一定的基礎知識才較好理解，每個 step 的目的也希望能更加瞭解而不是只照個講義做，這堂課給予的實作機會讓我充分了解自己還有哪部分需要加強，希望未來可以不斷精進，更了解 SOC 的各種細節及設計概念。