[SoC Lab] Lab1

tags: SoC Lab, SOC Design

Student ID	Name
311551095	林聖博

附上此篇Hackmd Link：https://hackmd.io/@Sheng08/H1hkiwMl6

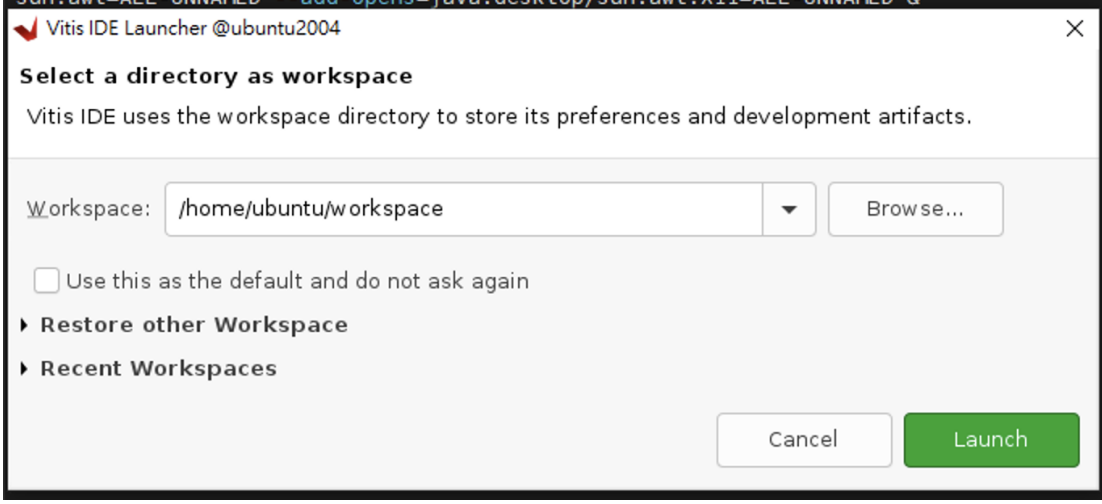
Lab 1

- https://github.com/bol-edu/course-lab_1

Tools Install Result

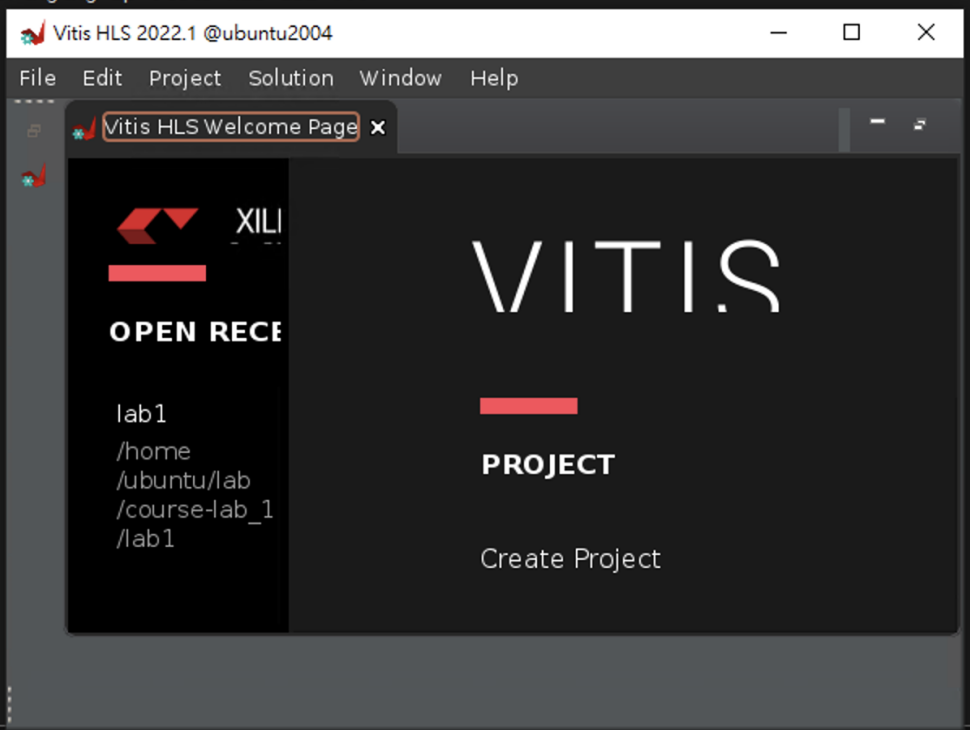
- vitis

```
ubuntu@ubuntu2004:~$ vitis

****** Xilinx Vitis Development Environment
****** Vitis v2022.1 (64-bit)
  **** SW Build 3524922 on 2022-04-14-18:00:18
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

Launching Vitis with command /tools/Xilinx/Vitis/2022.1/eclipse/lnx64.o/eclipse -vmargs -Xms64m -
Xmx1024m -Dorg.eclipse.swt.internal.gtk.cairoGraphics=false -Dosgi.configuration.area=@user.home/
.Xilinx/Vitis/2022.1 --add-modules=ALL-SYSTEM --add-opens=java.base/java.nio=ALL-UNNAMED --add-op
ens=java.desktop/sun.swing=ALL-UNNAMED --add-opens=java.desktop/javax.swing=ALL-UNNAMED --add-ope
ns=java.desktop/javax.swing.tree=ALL-UNNAMED --add-opens=java.desktop/javax.swing.plaf.basic=ALL-
UNNAMED --add-opens=java.desktop/javax.swing.plaf.synth=ALL-UNNAMED --add-opens=java.desktop/com.
sun.awt=ALL-UNNAMED --add-opens=java.desktop/sun.awt.X11=ALL-UNNAMED &
```

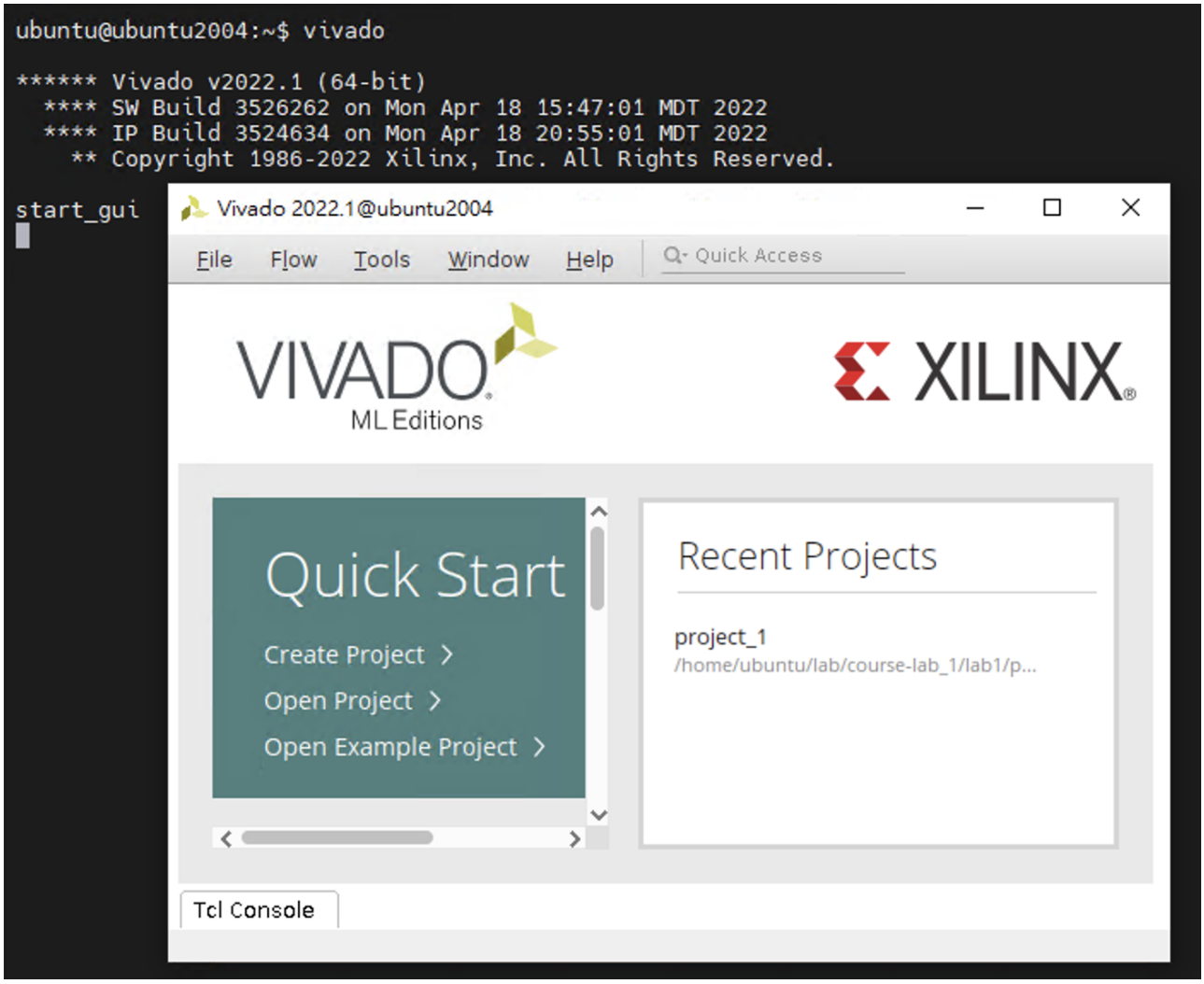


- vitis_hls

```
ubuntu@ubuntu2004:~$ vitis_hls

****** Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2022.1 (64-bit)
  **** SW Build 3526262 on Mon Apr 18 15:47:01 MDT 2022
  **** IP Build 3524634 on Mon Apr 18 20:55:01 MDT 2022
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

source /tools/Xilinx/Vitis_HLS/2022.1/scripts/vitis_hls/hls.tcl -notrace
INFO: [HLS 200-10] Running '/tools/Xilinx/Vitis_HLS/2022.1/bin/unwrapped/lnx64.o/vitis_hls'
INFO: [HLS 200-10] For user 'ubuntu' on host 'ubuntu2004.linuxvmimages.local' (Linux_x86_64 versi
on 5.15.0-84-generic) on Thu Sep 28 06:14:53 EDT 2023
INFO: [HLS 200-10] On os Ubuntu 20.04.4 LTS
INFO: [HLS 200-10] In directory '/home/ubuntu'
INFO: [HLS 200-10] Bringing up Vitis HLS GUI ...
```

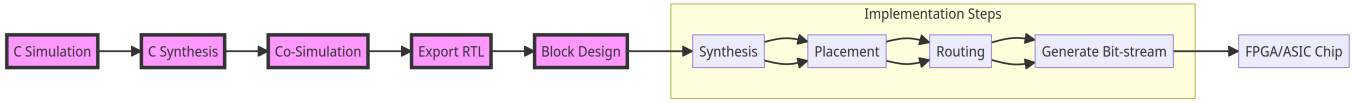


- vivado



Introduction about the overall system

從實作 Lab1 過程中，完成跑過整個簡單的實驗流程，可將個階段步驟整合成以下流程：



Step

1. C Simulation:
   - 利用 C/C++ 撰寫欲達成目標邏輯算法 → 運行C Simulation工具 → 驗證程式碼正確性
2. C Synthesis:
   - 選擇合成設置 → 運行合成(Synthesis)工具 → 產生合成後的 report
3. Co-Simulation:
   - 使用C模擬的輸入/輸出 → 運行合成的HDL模擬 → 比較結果
4. Export RTL:
   - 從HLS工具中選擇輸出或導出RTL → 獲得HDL文件
5. Implementation:

> 在這個階段，HDL 被轉換成 FPGA 或 ASIC 的實際物理結構。 選擇特定的 FPGA/ASIC → 運行 Implementation

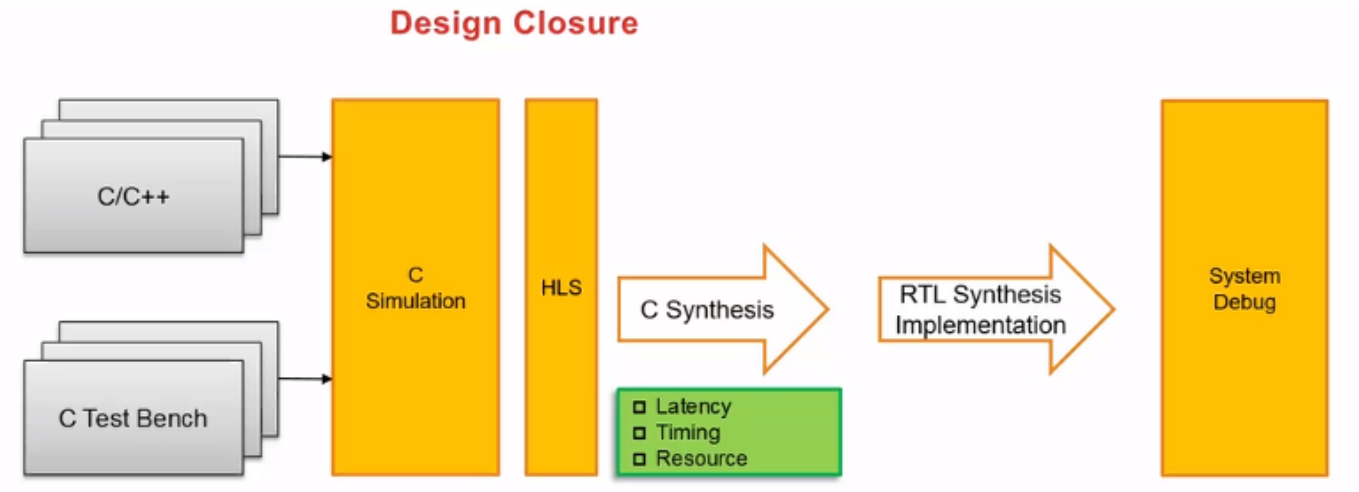
1. Synthesis (within Implementation):
    - 將HDL 轉換為一個 Netlist，這個 Netlist 包含邏輯閘、Flip-flop等之間的連接
2. Placement:
    - 決定 Netlist 中，每個元件在 FPGA 或 ASIC 物理布局上的位置
3. Routing:
    - 確定如何在 FPGA 或 ASIC 中連接這些元件，以形成完整的設計
4. Generate Bit-stream:
    - 創建一個代表整個 FPGA 配置的文件，可被下載到 FPGA 上執行

而 "Export RTL" 後，可進行 Block Design (系統集成)，使用圖形化介面連接不同的 IP 模組，也可以包含從 HLS 或其他來源導入的自定義 IP (Intellectual Property)。

系統概述

Lab1 使用的工具為 Vitis-HLS (Vitis High-Level Synthesis)，為基於 C++ 的開發工具，可以高階語言轉換為 RTL語言。

使用此設計流程，開發者可直接利用 C/C++ 來描述演算法及Test Bench，執行 C Simulation 驗證設計的正確性。接著，透過高階合成工具，完成 C/C++ 轉換到 RTL。同時也需要考慮延時、時序以及資源的問題。然後，依照 RTL 設計流程，使用軟體工具進行 RTL 電路的合成、佈局以及佈線。最終，在系統層面上對整體設計進行驗證和修正。

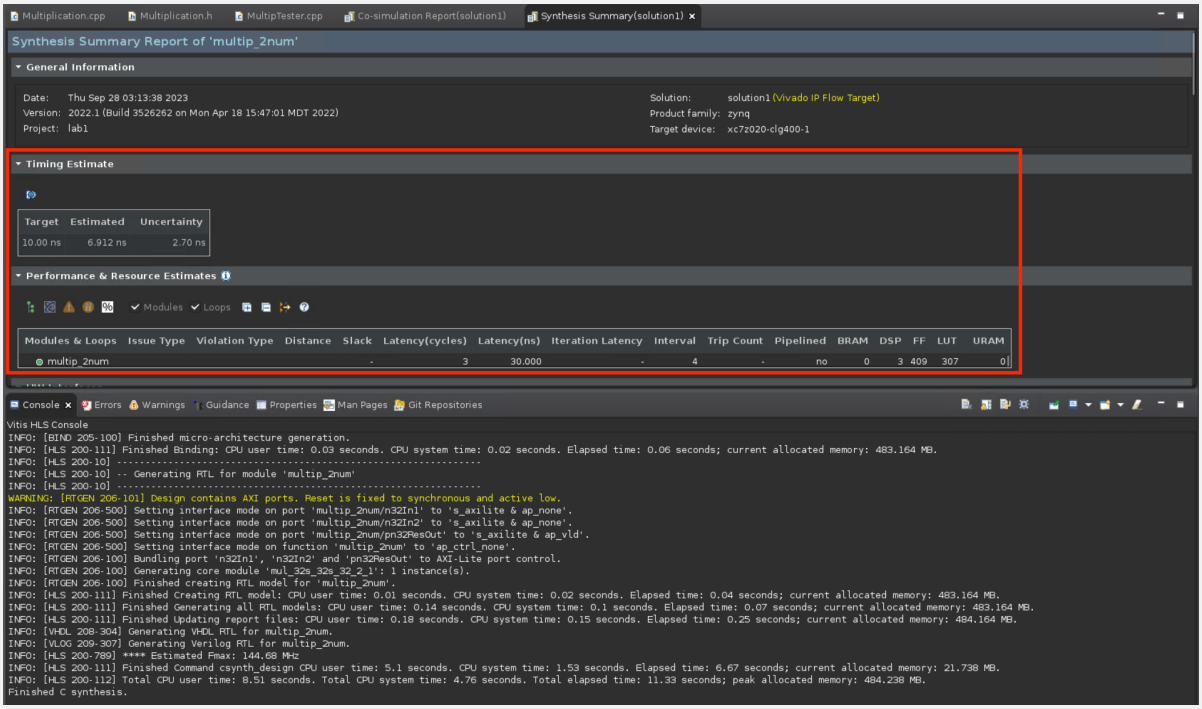


> Ref: https://xilinx.eetrend.com/blog/2022/100560242.html

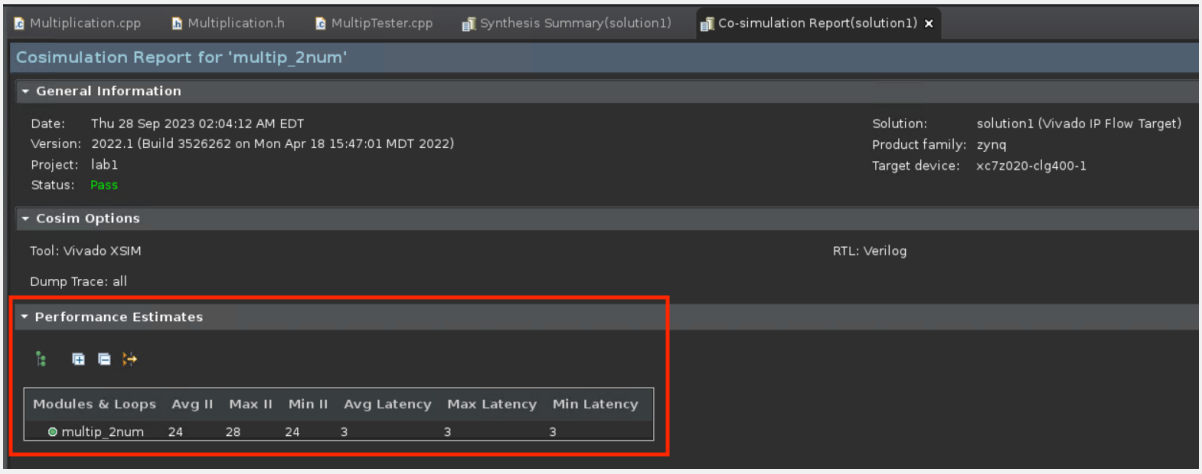
Screen dump

- Performance

C Synthesis



Co-Simulation



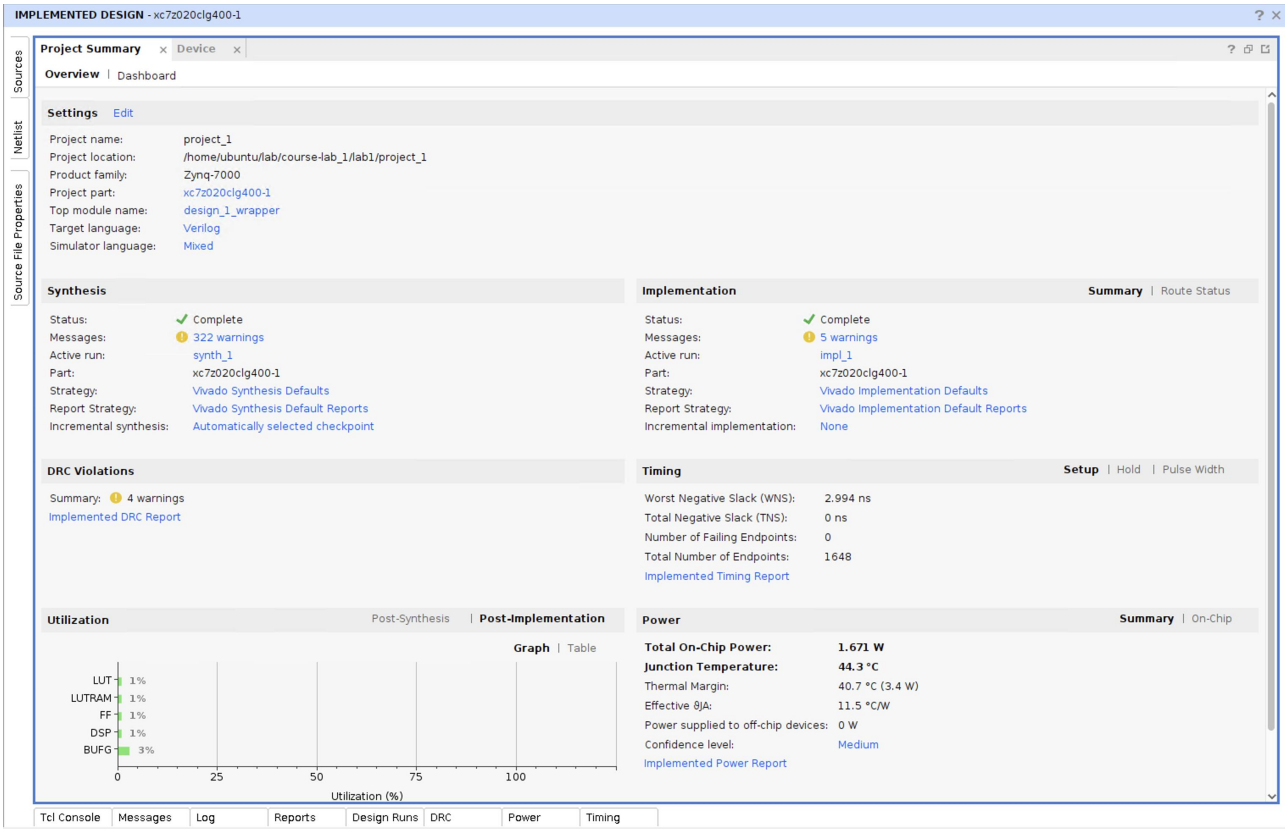
csynth.rpt

```
+ Performance & Resource Estimates:

    PS: '+' for module; 'o' for loop; '*' for dataflow
    +-----------------+-----+-----+----------+---------+----------+---------+-----+----------+-----+
    |    Modules      | Issue|      | Latency | Latency| Iteration|         | Trip |          |     |
    |    & Loops      | Type | Slack| (cycles)|  (ns)  |  Latency | Interval| Count| Pipelined| BRAM |
    +-----------------+-----+-----+----------+---------+----------+---------+-----+----------+-----+-
    |+ multip_2num    |     -|  0.39|         3|  30.000|         -|        4|     -|        no|     -|
    +-----------------+-----+-----+----------+---------+----------+---------+-----+----------+-----+-
```

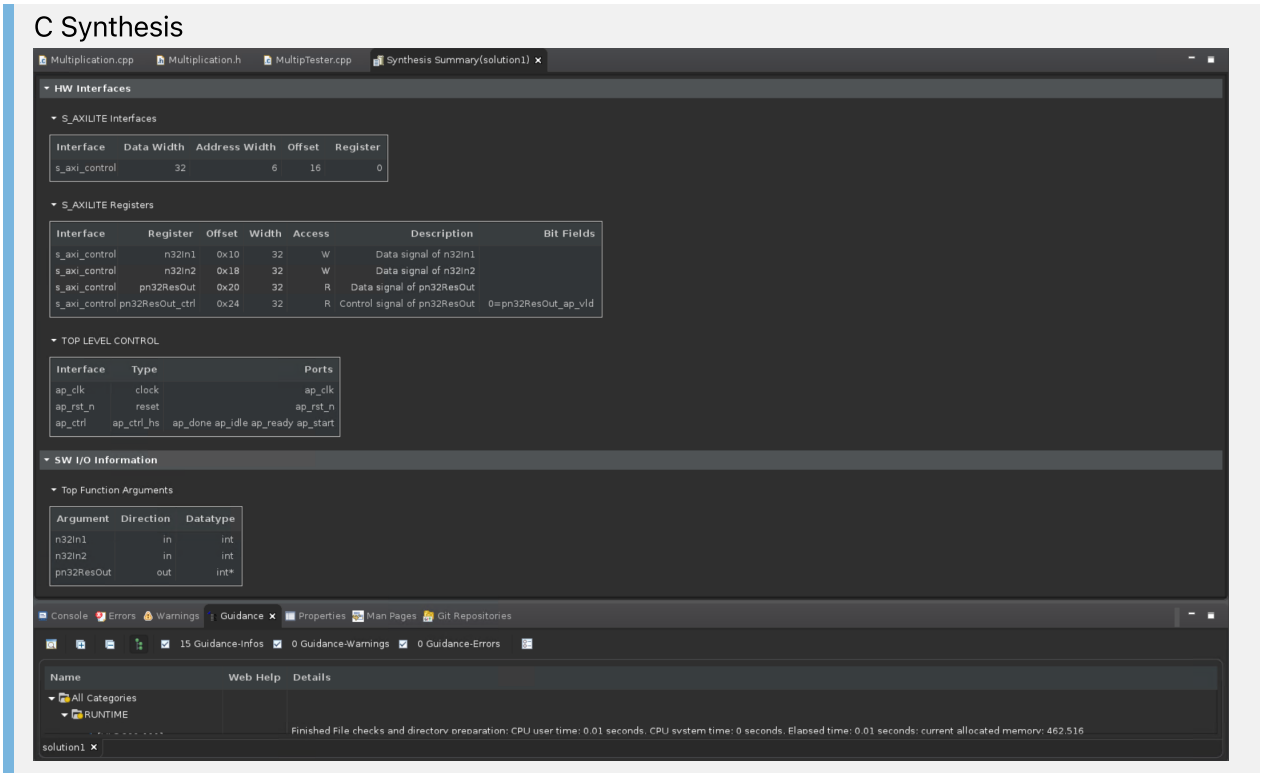
multip_2num_csynth.rpt

```
================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+----------+------------+
    |  Clock |  Target  | Estimated| Uncertainty|
    +--------+----------+----------+------------+
    |ap_clk  |  10.00 ns|  6.912 ns|     2.70 ns|
    +--------+----------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+-----+-----+---------+
    |  Latency (cycles) |   Latency (absolute)  |  Interval | Pipeline|
    |   min   |   max   |    min    |    max    | min | max |   Type  |
    +---------+---------+-----------+-----------+-----+-----+---------+
    |        3|        3|  30.000 ns|  30.000 ns|    4|    4|       no|
    +---------+---------+-----------+-----------+-----+-----+---------+

    + Detail:
        * Instance:
        N/A

        * Loop:
        N/A
```

- Utilization



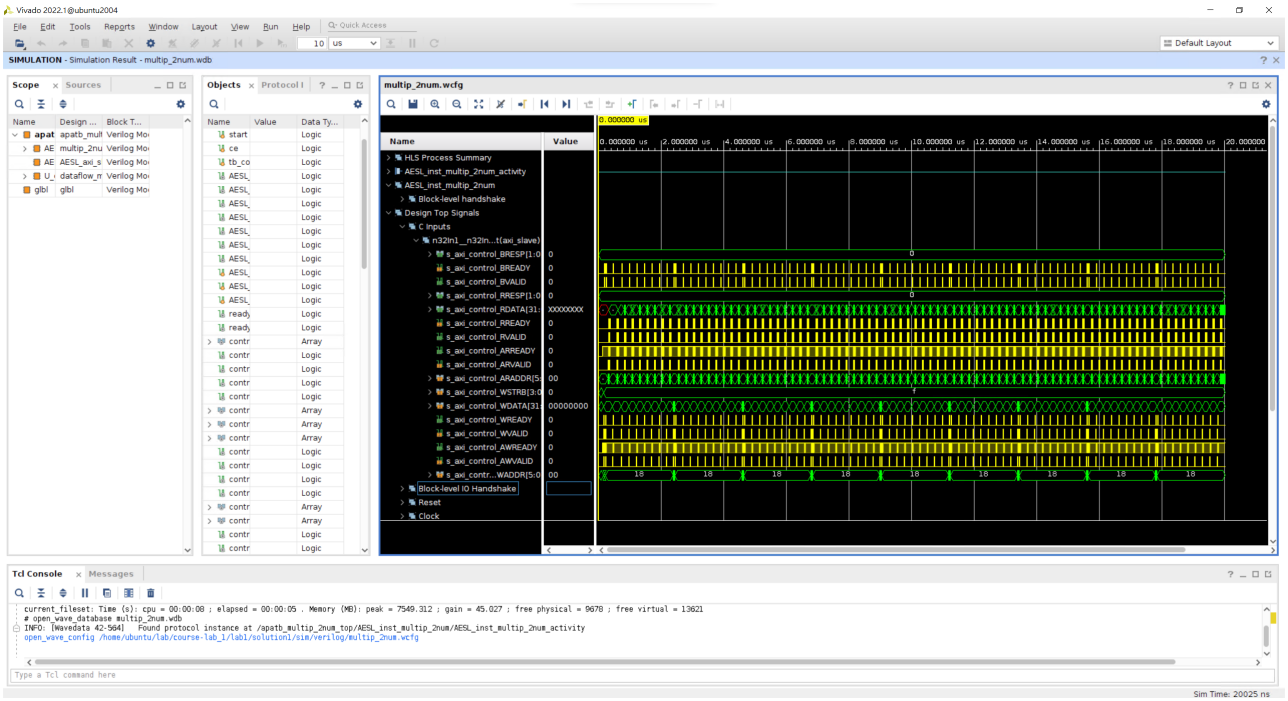
- Interface



multip_2num_csynth.rpt

```
================================================================
== Interface
================================================================
* Summary:
+-----------------------+-----+----+--------------+-------------+--------------+
|       RTL Ports       | Dir | Bits|   Protocol   | Source Object|    C Type    |
+-----------------------+-----+----+--------------+-------------+--------------+
|s_axi_control_AWVALID  |   in|    1|         s_axi|      control|       pointer|
|s_axi_control_AWREADY  |  out|    1|         s_axi|      control|       pointer|
|s_axi_control_AWADDR   |   in|    6|         s_axi|      control|       pointer|
|s_axi_control_WVALID   |   in|    1|         s_axi|      control|       pointer|
|s_axi_control_WREADY   |  out|    1|         s_axi|      control|       pointer|
|s_axi_control_WDATA    |   in|   32|         s_axi|      control|       pointer|
|s_axi_control_WSTRB    |   in|    4|         s_axi|      control|       pointer|
|s_axi_control_ARVALID  |   in|    1|         s_axi|      control|       pointer|
|s_axi_control_ARREADY  |  out|    1|         s_axi|      control|       pointer|
|s_axi_control_ARADDR   |   in|    6|         s_axi|      control|       pointer|
|s_axi_control_RVALID   |  out|    1|         s_axi|      control|       pointer|
|s_axi_control_RREADY   |   in|    1|         s_axi|      control|       pointer|
|s_axi_control_RDATA    |  out|   32|         s_axi|      control|       pointer|
|s_axi_control_RRESP    |  out|    2|         s_axi|      control|       pointer|
|s_axi_control_BVALID   |  out|    1|         s_axi|      control|       pointer|
|s_axi_control_BREADY   |   in|    1|         s_axi|      control|       pointer|
|s_axi_control_BRESP    |  out|    2|         s_axi|      control|       pointer|
|ap_clk                 |   in|    1|  ap_ctrl_none|   multip_2num|  return value|
|ap_rst_n               |   in|    1|  ap_ctrl_none|   multip_2num|  return value|
+-----------------------+-----+----+--------------+-------------+--------------+
```

- Co-simulation transcript/waveform



- Jupyter Notebook execution result

In [1]:

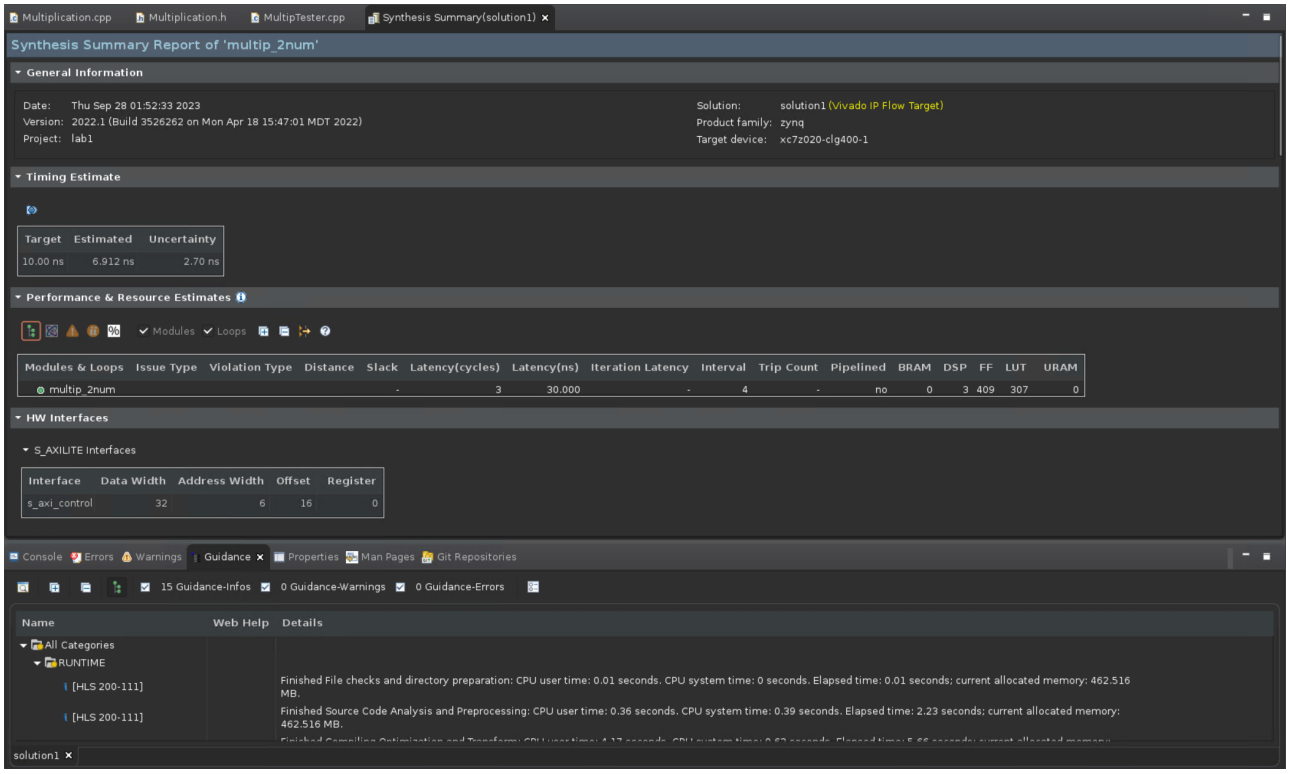
```
# coding: utf-8

# In[ ]:


from __future__ import print_function

import sys, os

sys.path.append('/home/xilinx')
os.environ['XILINX_XRT'] = '/usr'
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/jupyter_notebooks/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")
```

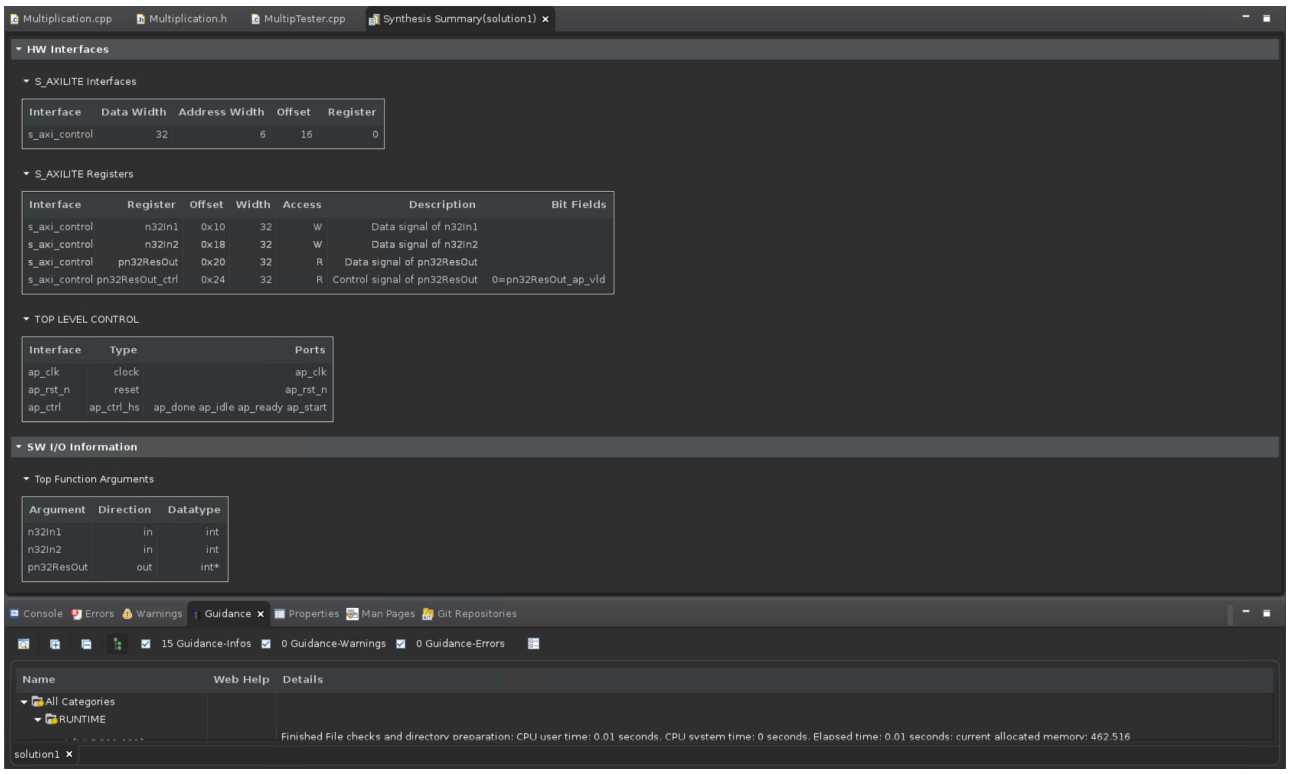
```
Entry: /usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py
System argument(s): 3
Start of "/usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py"
============================
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 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
```

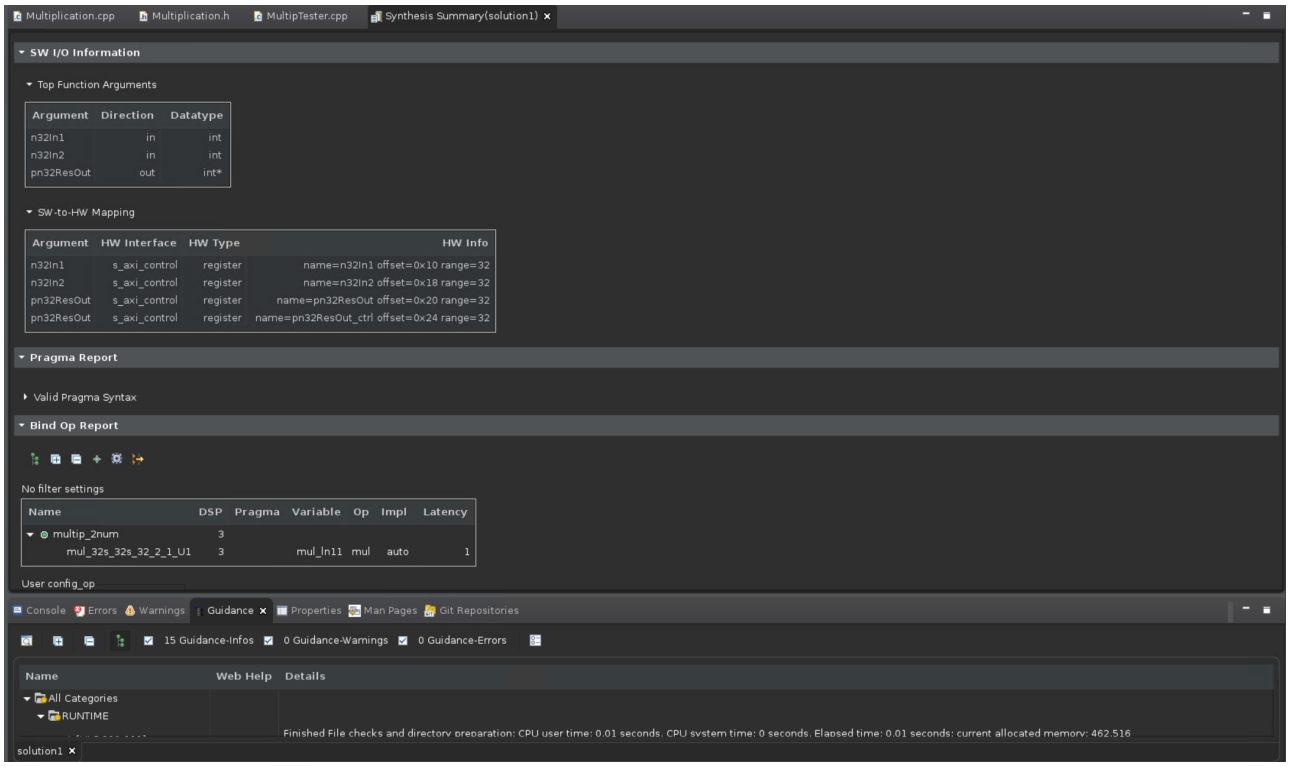
> 由於截圖範圍大，因此解析度有點損失

Screen dump (Other)

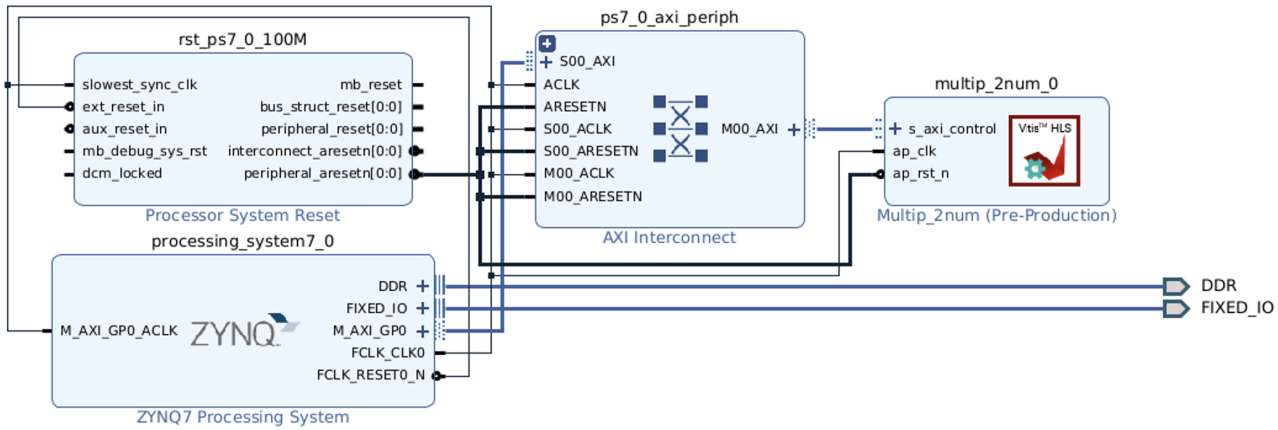
- C Synthesis

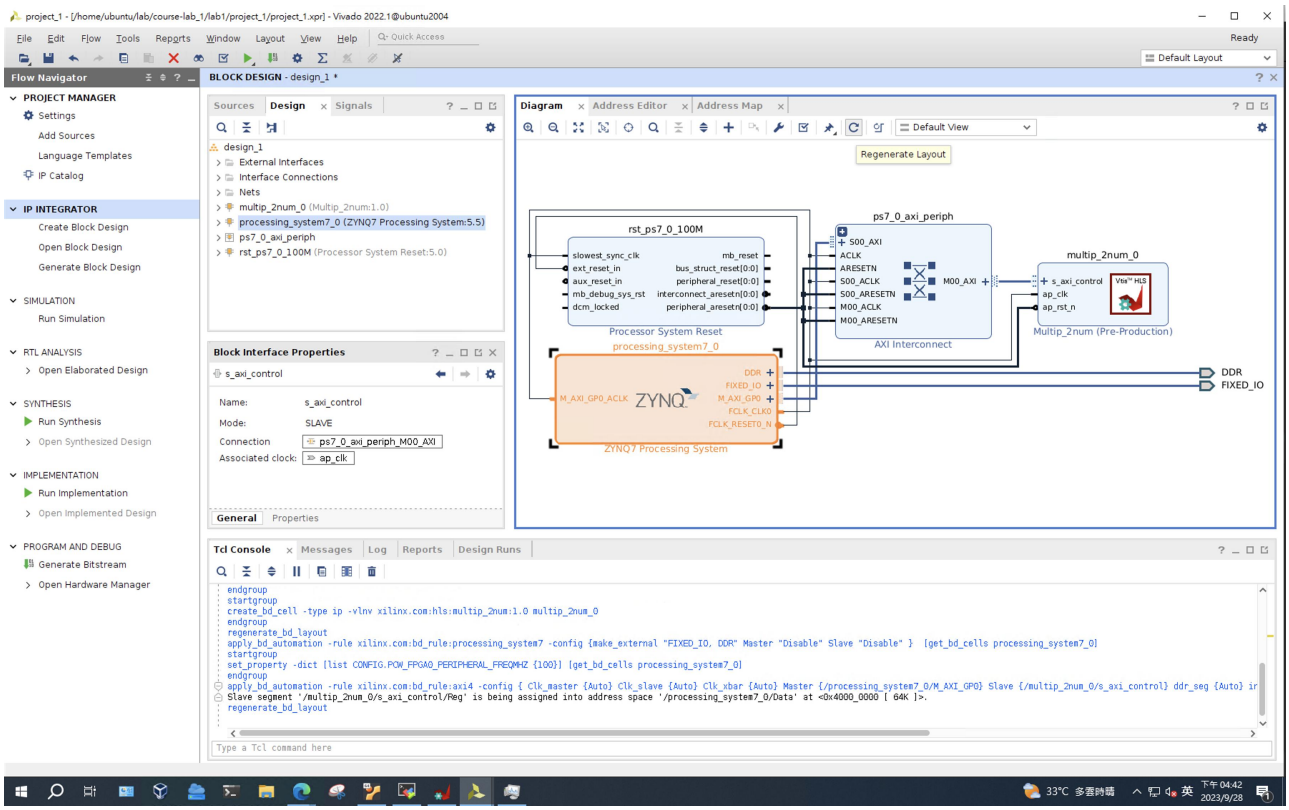




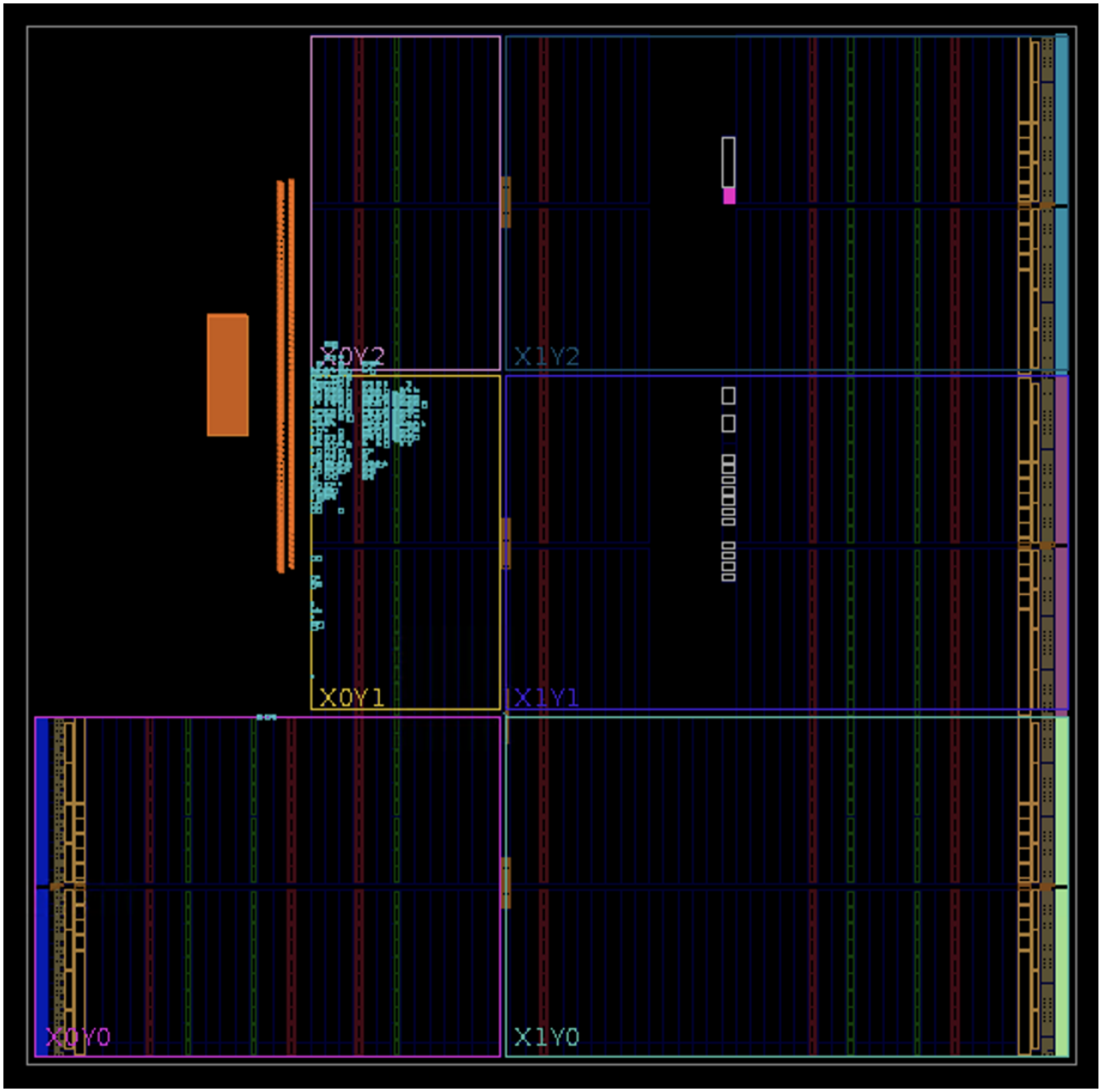


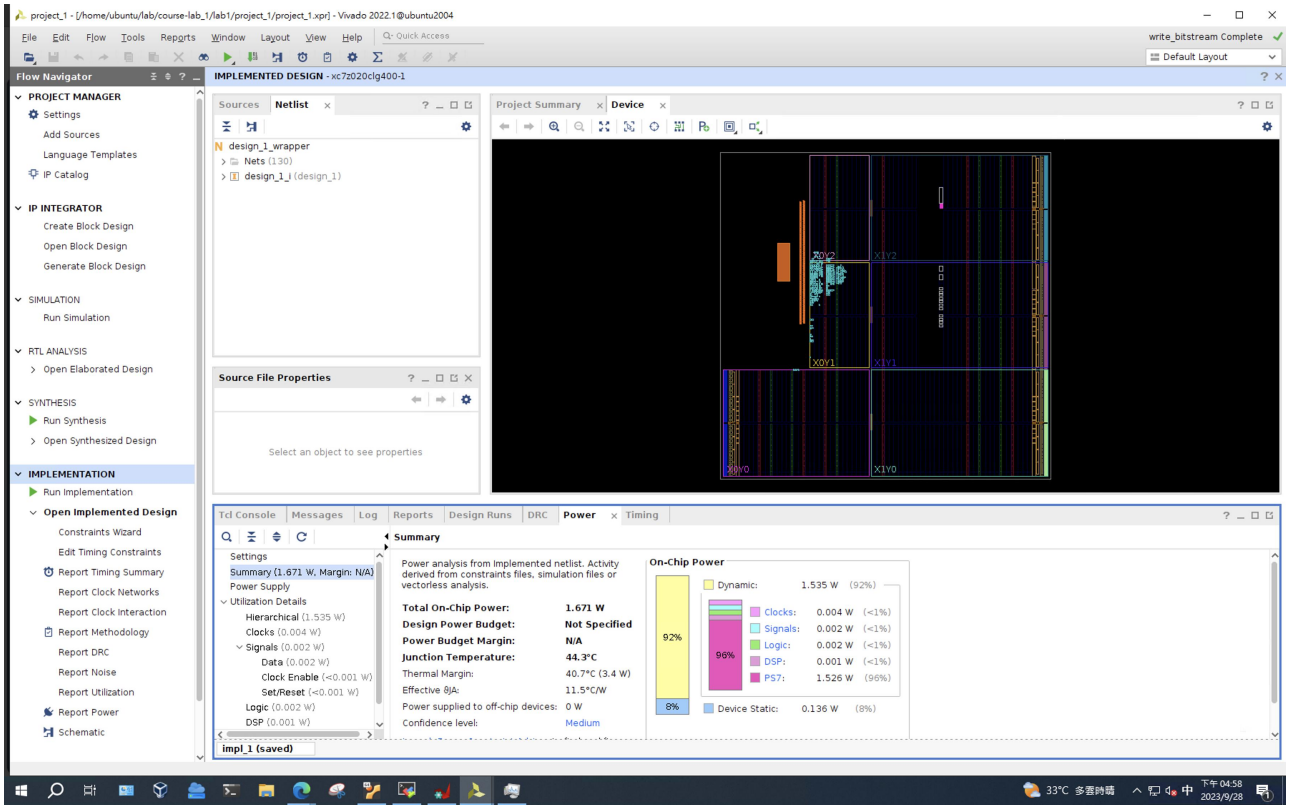
- Block Design




project_1 - [/home/ubuntu/lab/course-lab_1/lab1/project_1/project_1.xpr] - Vivado 2022.1@ubuntu2004
File Edit Flow Tools Reports Window Layout View Help
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
BLOCK DESIGN - design_1 *
Sources Design Signals
design_1
External Interfaces
Interface Connections
Nets
multip_2num_0 (Multip_2num:1.0)
processing_system7_0 (ZYNQ7 Processing System:5.5)
ps7_0_axi_periph
rst_ps7_0_100M (Processor System Reset:5.0)
Block Interface Properties
s_axi_control
Name: s_axi_control
Mode: SLAVE
Connection ps7_0_axi_periph_M00_AXI
Associated clock: ap_clk
General Properties
Diagram Address Editor Address Map
Default View
Regenerate Layout
rst_ps7_0_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
multip_2num_0
s_axi_control
ap_clk
ap_rst_n
Vitis™ HLS
Multip_2num (Pre-Production)
processing_system7_0
M_AXI_GP0_ACLK
ZYNQ
DDR
FIXED_IO
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
DDR
FIXED_IO
Tcl Console Messages Log Reports Design Runs
endgroup
startgroup
create_bd_cell -type ip -vlnv xilinx.com:hls:multip_2num:1.0 multip_2num_0
endgroup
regenerate_bd_layout
apply_bd_automation -rule xilinx.com:bd_rule:processing_system7 -config {make_external "FIXED_IO, DDR" Master "Disable" Slave "Disable" } [get_bd_cells processing_system7_0]
startgroup
set_property -dict [list CONFIG.PCW_FPGA0_PERIPHERAL_FREQMHZ {100}] [get_bd_cells processing_system7_0]
endgroup
apply_bd_automation -rule xilinx.com:bd_rule:axi4 -config { Clk_master {Auto} Clk_slave {Auto} Clk_xbar {Auto} Master {/processing_system7_0/M_AXI_GP0} Slave {/multip_2num_0/s_axi_control} ddr_seg {Auto} ir
Slave segment '/multip_2num_0/s_axi_control/Reg' is being assigned into address space '/processing_system7_0/Data' at <0x4000_0000 [64K]>.
regenerate_bd_layout
Type a Tcl command here
33°C 多雲時晴
英
下午 04:42
2023/9/28

- Implement Design





Note

名詞解釋

- **C Simulation:**
  - 目的: 驗證C/C++的算法邏輯正確性
  - 說明: 為初步的模擬階段，只涉及C/C++程式，**不涉及任何硬體描述語言**。基本上，這階段確保C/C++程式在未進行任何合成或硬體轉換的情況下運行正確性
  - 結果: 若發現錯誤，需要於這階段修改C/C++程式
- **C Synthesis:**
  - 目的: 將轉換C/C++到硬體描述語言 (HDL)，如 VHDL 或 Verilog。
  - 說明: 高階合成 (HLS) 工具會嘗試**將C/C++程式碼轉化成硬體結構**。在這階段，可能需要進行優化或改善，以達到所需的**clock速度**、**資源使用或功耗**
  - 結果: 得到合成後的報告(`csynth.rpt`)，包括估計的資源使用、性能(Performance)等
- **Co-Simulation:**
  - 目的: 驗證合成後的HDL程式碼與C/C++原始碼的**行為**是否一致。
  - 說明: 在這階段，原始C/C++程式碼和其合成的HDL程式碼將**同時模擬**。這是確保合成後的HDL實現與原始演算邏輯在功能上保持一致
  - 結果: 若發現差異或問題，需回到C Synthesis階段，調整優化指令或更改C/C++程式，然後重新合成和模擬
- **Export RTL:**
  - 目的: 生成代表特定設計的硬體描述語言 (HDL)，如 VHDL 或 Verilog。
  - 說明: 在完成高階合成 (HLS) 後，已從其C/C++ 轉換為硬體結構。而 "Export RTL" 將硬體結構生成為可用於標準 FPGA 或 ASIC toolchain 的 HDL 程式碼的過程
  - 結果: 生成的 HDL 可以被整合到更大的系統中，或者直接使用在 FPGA 或 ASIC 設計中
- **IP (Intellectual Property):**
  - 目的: 封裝特定的設計或功能以便**重複使用**

    - 說明: IP 是指一個已經設計好的、可以重複使用的功能或子系統。IP 可為自己設計，亦可為第三方供應商提供。當設計被合成並驗證無誤後，可被封裝成一個 IP，以便在其他設計中重複使用或共享
    - 結果: 生成的 IP 模組通常以標準格式提供，如：Xilinx 的 AXI4 接口，允許被整合到更大的系統中

觀念補充

1. 在 HLS 中，"pragmas" 和 "directives" 通常指同樣的東西，都是用於指導合成過程的指令。允許優化和指導合成過程，而不必大量修改 C/C++ 的原始碼。
2. reg (register) 為存儲單元。在 FPGA 設計中，register 是用來保存狀態或資料的基本元件，由`flip-flops`構成。
    - `regIP = ol.multip_2num_0`，代表 IP 核心的物件，可通過 Python 存取的 register
        - 每個在 FPGA overlay 中的 IP 都可利用其名稱於 Python 存取和控制
3. `Overlay`: 將 bitstream 載入到 FPGA，並將相應的硬體設計在 FPGA 上執行
    - MMIO (Memory-Mapped I/O): 允許 CPU 使用常規的記憶體存取指令，如：load 和 store
        - 特定 I/O 裝置的 register 或控制位會**映射(mapping)到系統的主記憶體地址空間中的某些位置**
    - 在 PYNQ 環境中，利用 MMIO 直接訪問 FPGA 上的 MMIO 地址空間，能夠讀取與寫入 FPGA 中的硬體 register
    - 使用 Overlay 載入 FPGA 設計時，各個 IP 核心可能有屬於自己的 MMIO 地址範圍，允許從 Python 存取和控制這些核心

心得

由於為資工系的學生，之前並未有機會接觸到SoC的相關知識和設計。在這學期的SoC課程中，首次深入接觸SoC的設計與相關知識。 於Lab1實作過程中，遇到了許多我不熟悉的專業名詞和實作原理。為了更好掌握這些新知識，因此投入時間學習和理解。而透過Lab1的初步實作更讓我體驗到工具的操作，如: vitis_hls，這都使我對SoC有了更深入的了解，從而加深了我的理解與實踐能力。

而本次實驗過程中，對於 OnlineFPGA 的租借機制與 infra 感到非常驚艷，利用所提供的平台服務，可快速建立驗證環境，並且可不侷限於理設備的實體實驗限制與配置的複雜性，利用遠端方式就可以快速完成 FPAG 的設計與驗證以及測試。

未來預期目標

- 目前可能有些名詞用語不夠精準，期望後續能逐漸了解其意義，並正確的使用名詞說明實作過程的體悟與想法

Troubleshooting

- None

🚀 補充

後續更多資訊會補充於以下HackMD

- https://hackmd.io/@Sheng08/HkzeBuDyp

📌 TODO

- [ ] 了解 ASIC
- [ ] 了解 正反器(Flip-flop) 與查找表(LUT)