Report – Lab 1

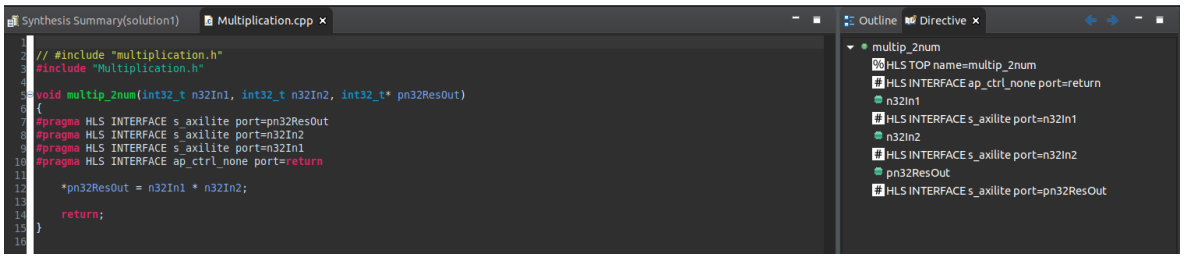
R13943009 鄭至盛

Overview of the HLS Design:

The C code

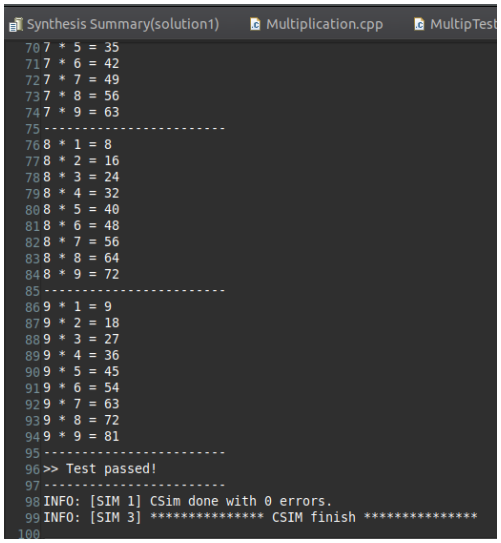
The .h file simply defines our top function (multip_2num) & data type (int32_t).

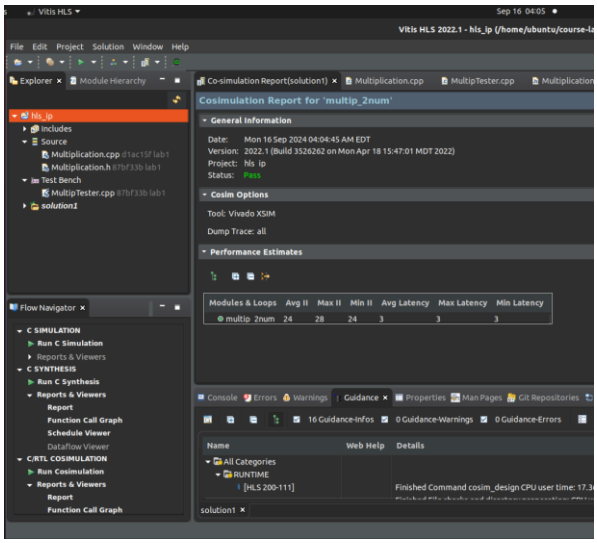
Let's move on to the .cpp file below. At first, the .cpp file defines the top function with 2x int32_t inputs & 1 int32_t output (address). Then, it defines the AXI-Lite interface for IO with pragma & the corresponding control protocol (ap_ctrl_none: eliminating handshake signals). Last, it defines the hardware description (simply multiple the 2 inputs & store them back to the assigned address).



Finally, the tester simply computes different $i \times j$ with multip_2num & pure software. Then it would compare both outputs & determine whether the functional verification is passed.

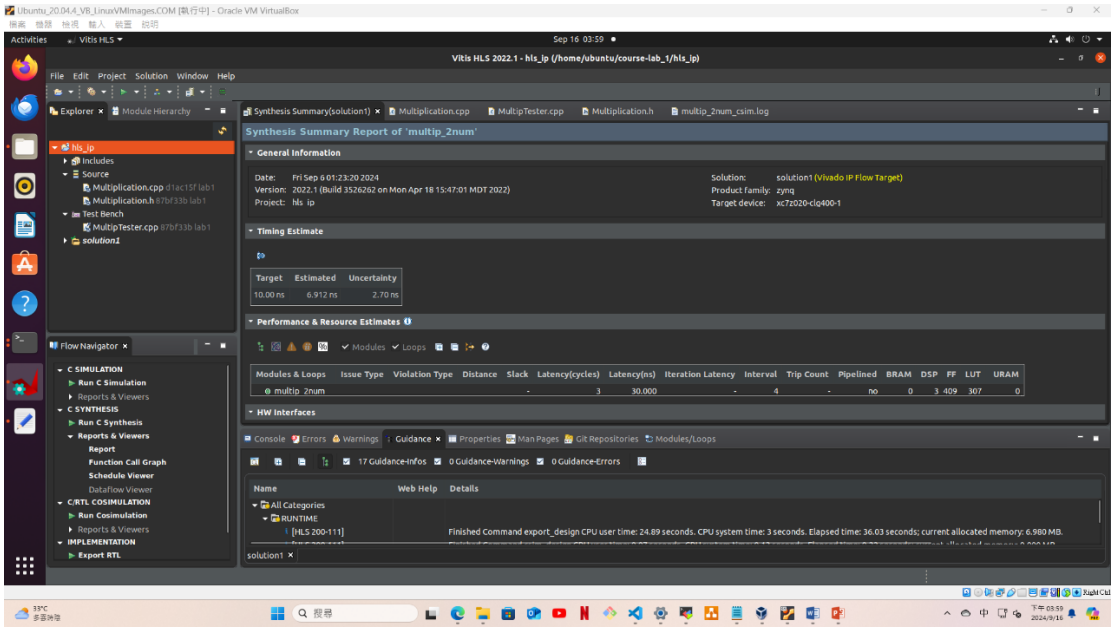
C-simulation (left) & Co-simulation (right)





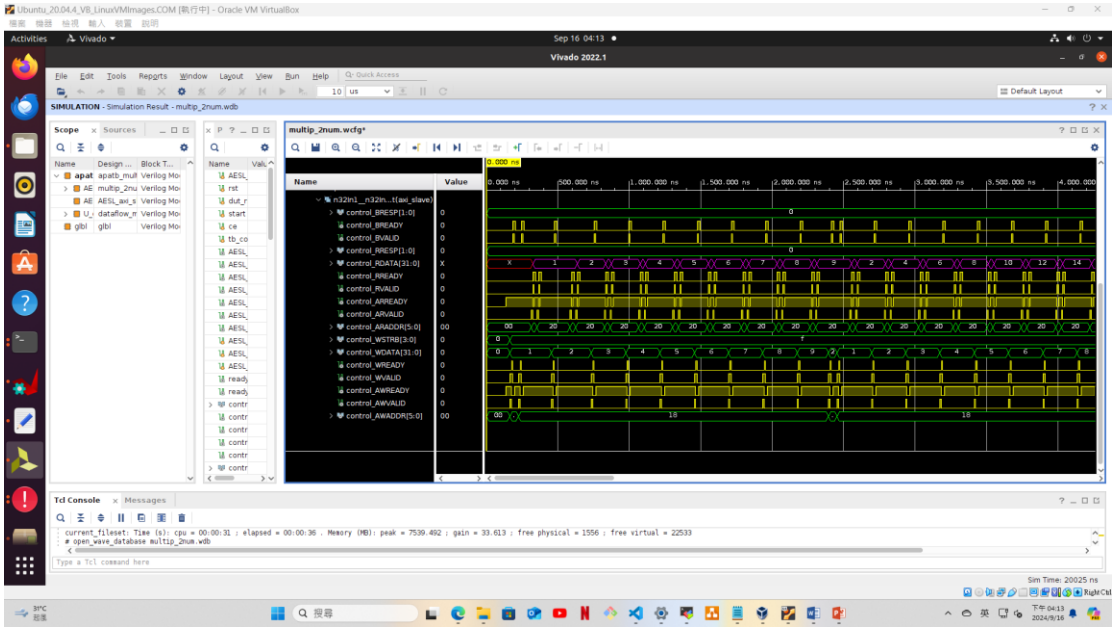
C-synthesis

The design is synthesized with clock period=10ns



Generated Waveform

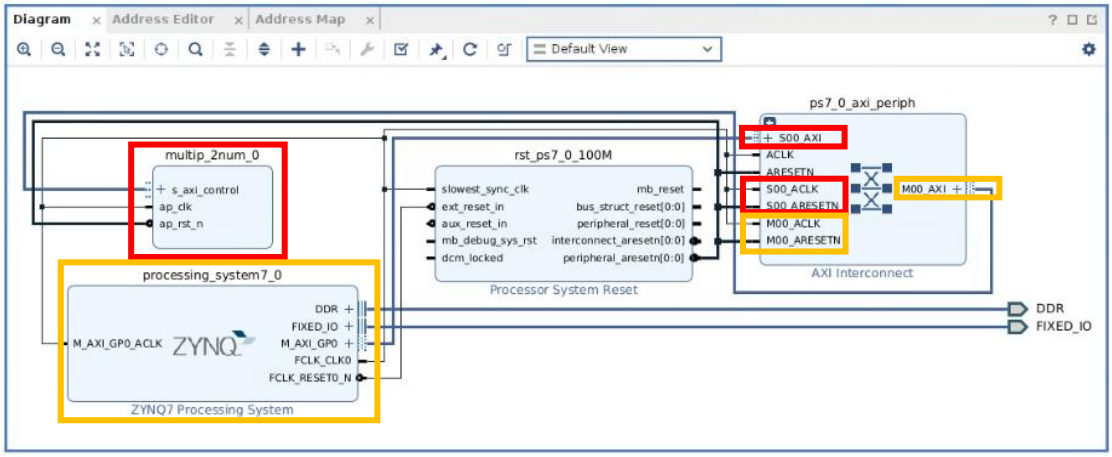
The purple part is the output signal sent to AXI-bus by RTL part.



Overview of the Hardware Wrapper:

Block Diagram

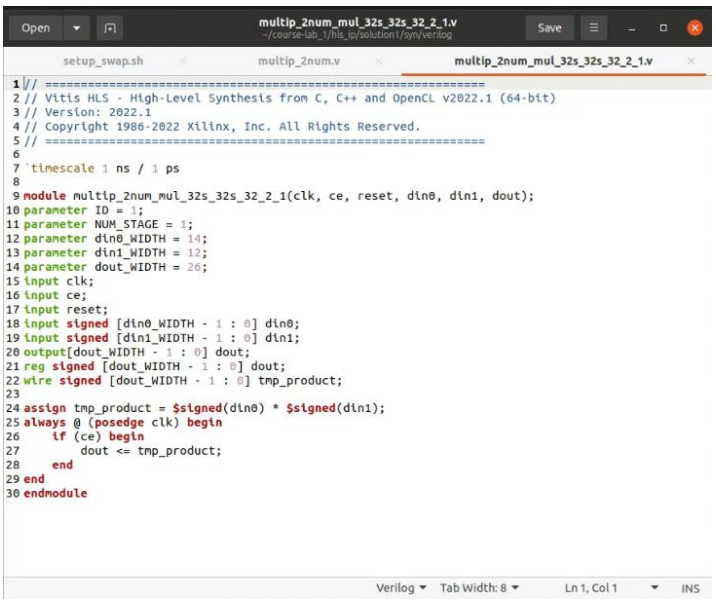
In the block diagram, we can see an AXI peripheral is connected between generated HLS IP (multip_2num_0) & ZYNQ7 Processing System. With the protocol of AXI-Lite, the connection between ZYNQ (Master) & the peripheral is only M00_AXI signals, and the connection between HLS IP (Slave) & the peripheral is the S00_AXI signals.



Verilog Code

The generated Verilog code can be divided into 2 parts. The computation unit is mainly composed of 1 FF & 1 multiplier. On the other hand, the controlling unit deals with the AXI-Lite protocol.

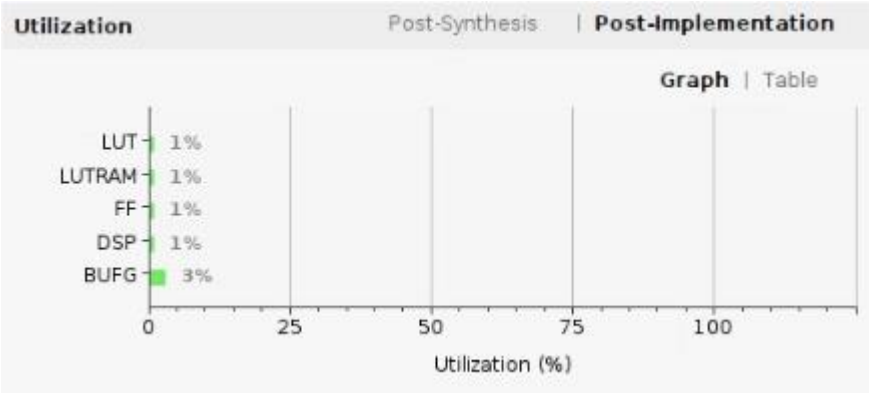
The bit-width of din0, din1, & dout would overwrite to 32, 32, & 64 respectively by its parent.



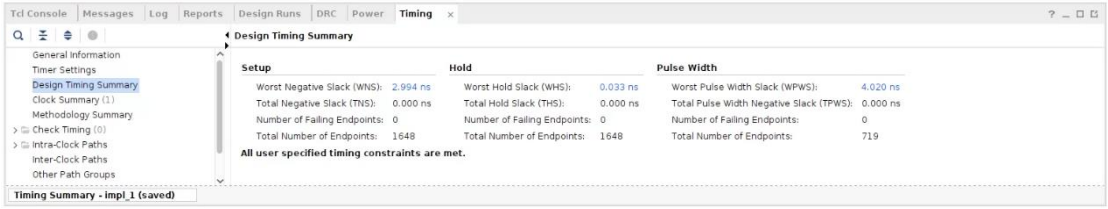

```verilog
// ==============================================================
// Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2022.1 (64-bit)
// Version: 2022.1
// Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
// ==============================================================

`timescale 1 ns / 1 ps

module multip_2num_mul_32s_32s_32_2_1(clk, ce, reset, din0, din1, dout);
parameter ID = 1;
parameter NUM_STAGE = 1;
parameter din0_WIDTH = 14;
parameter din1_WIDTH = 12;
parameter dout_WIDTH = 26;
input clk;
input ce;
input reset;
input signed [din0_WIDTH - 1 : 0] din0;
input signed [din1_WIDTH - 1 : 0] din1;
output[dout_WIDTH - 1 : 0] dout;
reg signed [dout_WIDTH - 1 : 0] dout;
wire signed [dout_WIDTH - 1 : 0] tmp_product;

assign tmp_product = $signed(din0) * $signed(din1);
always @ (posedge clk) begin
    if (ce) begin
        dout <= tmp_product;
    end
end
endmodule
```

Hardware Utilization

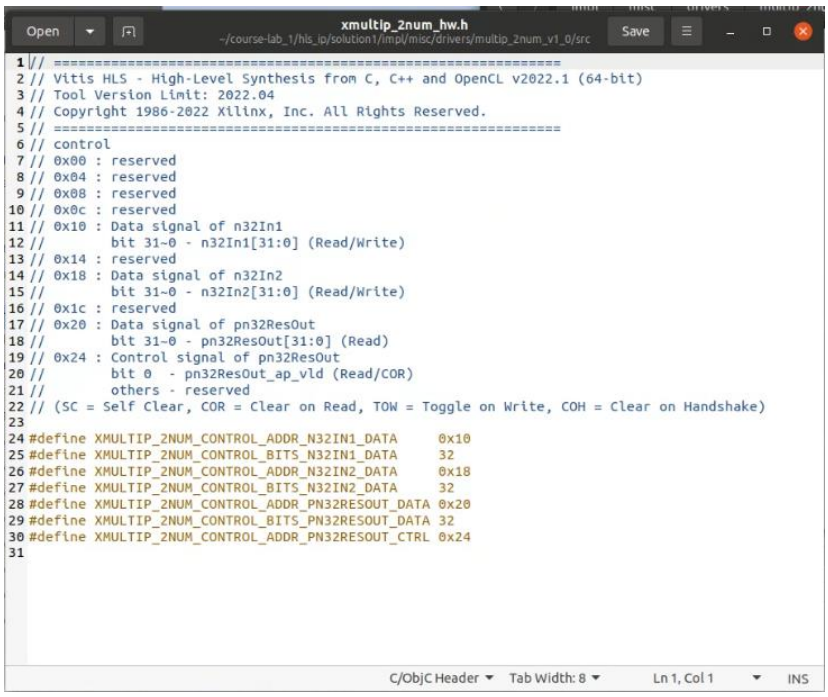
From the aspect of DSP, the 32x32 multiplier requires 3 DSP48E1 (25x18 for each). On the other hand, the whole HLS IP & the AXI peripheral require ~200 / ~400 FFs respectively.



Under 100MHz clock signals, both setup / hold slack is met.



In the driver files of MISC, we can find the address of different control signals which PYNQ APIs can further exploit.




```
// ==============================================================
// Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2022.1 (64-bit)
// Tool Version Limit: 2022.04
// Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
// ==============================================================
// control
// 0x00 : reserved
// 0x04 : reserved
// 0x08 : reserved
// 0x0c : reserved
// 0x10 : Data signal of n32In1
//        bit 31~0 - n32In1[31:0] (Read/Write)
// 0x14 : reserved
// 0x18 : Data signal of n32In2
//        bit 31~0 - n32In2[31:0] (Read/Write)
// 0x1c : reserved
// 0x20 : Data signal of pn32ResOut
//        bit 31~0 - pn32ResOut[31:0] (Read)
// 0x24 : Control signal of pn32ResOut
//        bit 0  - pn32ResOut_ap_vld (Read/COR)
//        others - reserved
// (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)

#define XMULTIP_2NUM_CONTROL_ADDR_N32IN1_DATA     0x10
#define XMULTIP_2NUM_CONTROL_BITS_N32IN1_DATA     32
#define XMULTIP_2NUM_CONTROL_ADDR_N32IN2_DATA     0x18
#define XMULTIP_2NUM_CONTROL_BITS_N32IN2_DATA     32
#define XMULTIP_2NUM_CONTROL_ADDR_PN32RESOUT_DATA 0x20
#define XMULTIP_2NUM_CONTROL_BITS_PN32RESOUT_DATA 32
#define XMULTIP_2NUM_CONTROL_ADDR_PN32RESOUT_CTRL 0x24
```

Overview of the PS-side (PYNQ)

Python code

We can see the Python code simply loads the bitstream file, selects IP, and writes to & reads from the address defined in the driver files.

```python
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")
```

Final results

As shown in the notebook, the computed results from HLS IP are correct.

jupyter Multip2Num Last Checkpoint: 3 分鐘前 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Voilà

In [1]:

```python
# 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
```

In [ ]:

In [ ]: