

Report – Lab 1

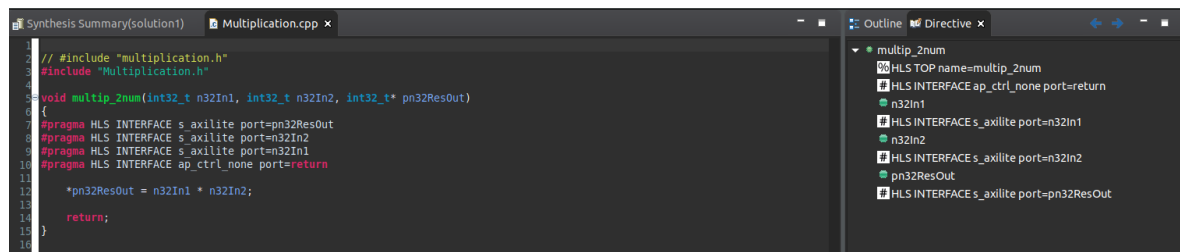
R13943009 鄭至盛

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).



```
// #include "multiplication.h"
#include "Multiplication.h"

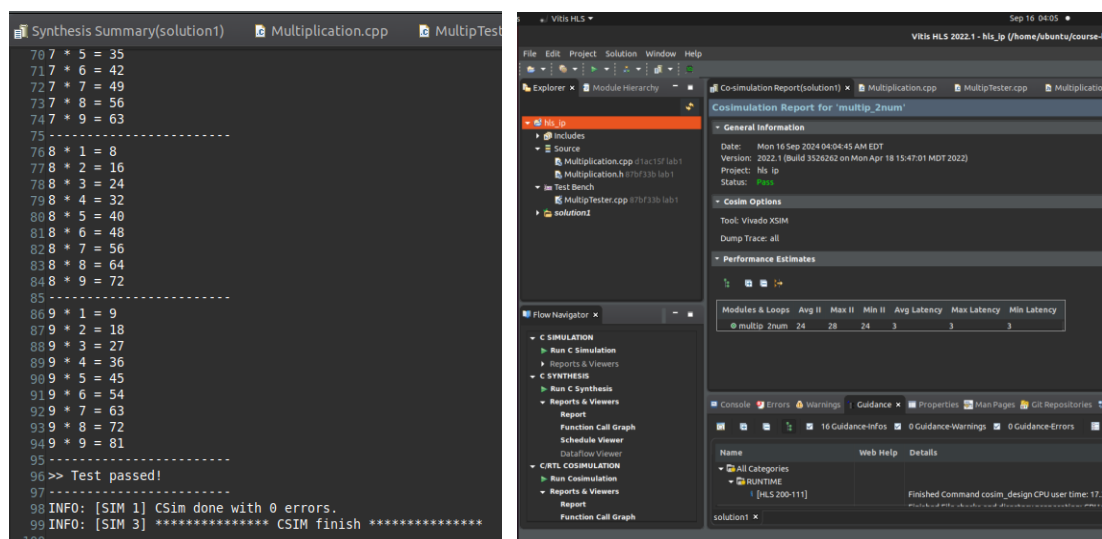
void multip_2num(int32_t n32In1, int32_t n32In2, int32_t* pn32ResOut)
{
    #pragma HLS INTERFACE s_axilite port=pn32ResOut
    #pragma HLS INTERFACE s_axilite port=n32In2
    #pragma HLS INTERFACE s_axilite port=n32In1
    #pragma HLS INTERFACE ap_ctrl_none port=return

    *pn32ResOut = n32In1 * n32In2;

    return;
}
```

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-simulation (left): The screenshot shows the MultipTest.cpp file with a list of multiplication results. The results are as follows:

i	j	Result
70	7	5 = 35
71	7	6 = 42
72	7	7 = 49
73	7	8 = 56
74	7	9 = 63
75	8	1 = 8
76	8	2 = 16
77	8	3 = 24
78	8	4 = 32
79	8	5 = 40
80	8	6 = 48
81	8	7 = 56
82	8	8 = 64
83	8	9 = 72
84	8	1 = 9
85	8	2 = 18
86	8	3 = 27
87	8	4 = 36
88	8	5 = 45
89	8	6 = 54
90	8	7 = 63
91	8	8 = 72
92	8	9 = 81

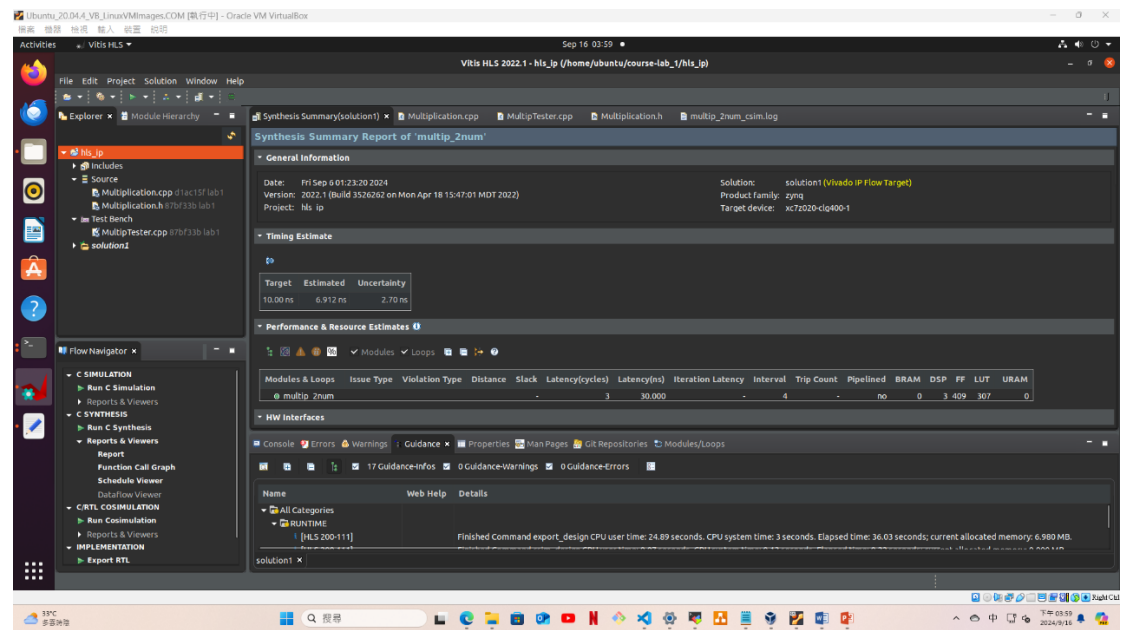
The test passed! The C-simulation was done with 0 errors.

Co-simulation (right): The screenshot shows the Co-simulation Report for 'multip_2num'. The report includes the following information:

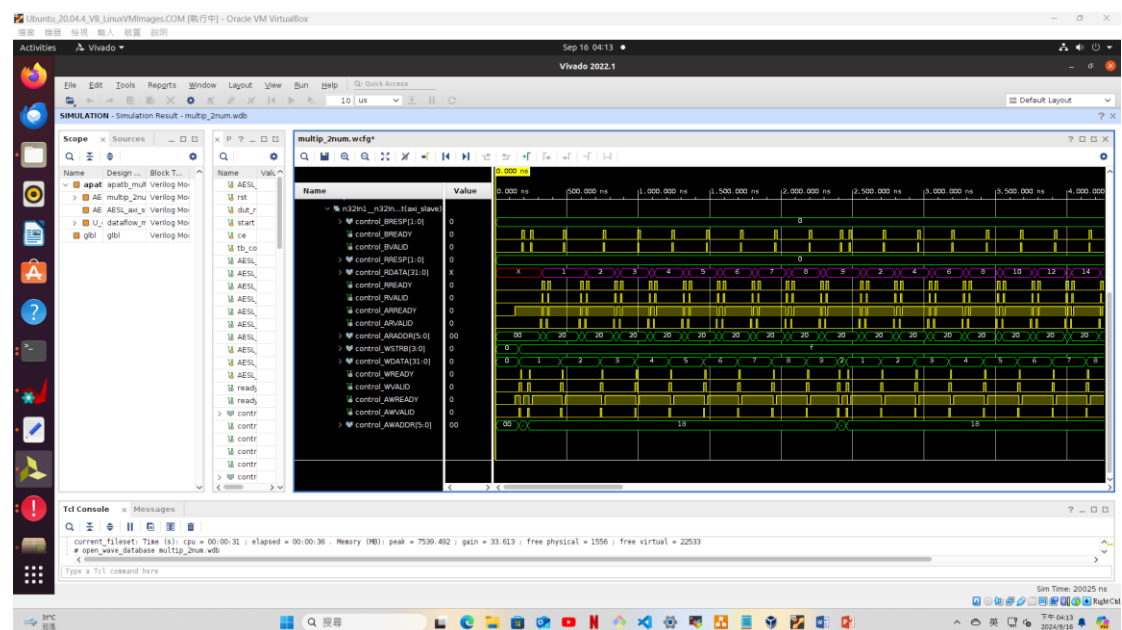
- General Information: Date: Mon 16 Sep 2024 04:04:45 AM EDT, Version: 2022.1 (Build 3326262 on Mon Apr 18 15:47:01 MDT 2022), Project: No IP, Status: Pass.
- Cosim Options: Tool: Vivado XSIM, Dump Trace: all.
- Performance Estimates: A table showing the performance of the co-simulation.

Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
• multip_2num	24	28	24	3	3	3

The design is synthesized with clock period=10ns



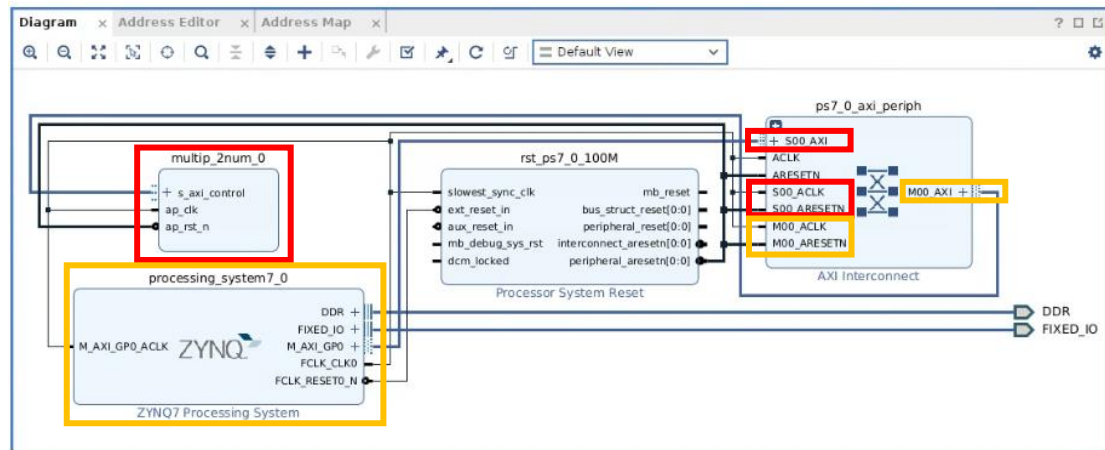
The purple part is the output signal sent to AXI-bus by RTL part.



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.

```
Open multip_2num_mul_32s_32s_32_2_1.v Save
~/course/lab_1/hls_ip/solution1/gpn/verilog

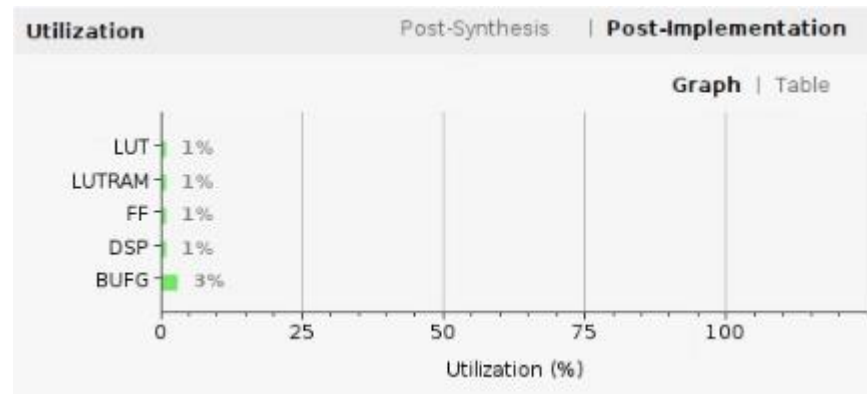
setup_swap.sh multip_2num.v multip_2num_mul_32s_32s_32_2_1.v

1 // =====
2 // Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2022.1 (64-bit)
3 // Version: 2022.1
4 // Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 // =====
6
7 `timescale 1 ns / 1 ps
8
9 module multip_2num_mul_32s_32s_32_2_1(clk, ce, reset, din0, din1, dout);
10 parameter ID = 1;
11 parameter NUM_STAGE = 1;
12 parameter din0_WIDTH = 14;
13 parameter din1_WIDTH = 12;
14 parameter dout_WIDTH = 26;
15 input clk;
16 input ce;
17 input reset;
18 input signed [din0_WIDTH - 1 : 0] din0;
19 input signed [din1_WIDTH - 1 : 0] din1;
20 output [dout_WIDTH - 1 : 0] dout;
21 reg signed [dout_WIDTH - 1 : 0] dout;
22 wire signed [dout_WIDTH - 1 : 0] tmp_product;
23
24 assign tmp_product = $signed(din0) * $signed(din1);
25 always @ (posedge clk) begin
26   if (ce) begin
27     dout <= tmp_product;
28   end
29 end
30 endmodule

Verilog Tab Width: 8 Ln 1, Col 1 INS
```

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.

The Design Timing Summary window displays the following data:

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 2.994 ns	Worst Hold Slack (WHS): 0.033 ns	Worst Pulse Width Slack (WPWS): 4.020 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1648	Total Number of Endpoints: 1648	Total Number of Endpoints: 719

All user specified timing constraints are met.

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

```
1 // =====
2 // Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2022.1 (64-bit)
3 // Tool Version Limit: 2022.04
4 // Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
5 // =====
6 // control
7 // 0x00 : reserved
8 // 0x04 : reserved
9 // 0x08 : reserved
10 // 0x0c : reserved
11 // 0x10 : Data signal of n32In1
12 // bit 31-0 - n32In1[31:0] (Read/Write)
13 // 0x14 : reserved
14 // 0x18 : Data signal of n32In2
15 // bit 31-0 - n32In2[31:0] (Read/Write)
16 // 0x1c : reserved
17 // 0x20 : Data signal of pn32ResOut
18 // bit 31-0 - pn32ResOut[31:0] (Read)
19 // 0x24 : Control signal of pn32ResOut
20 // bit 0 - pn32ResOut_ap_vld (Read/COR)
21 // others - reserved
22 // (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)
23
24 #define XMULTIP_2NUM_CONTROL_ADDR_N32IN1_DATA 0x10
25 #define XMULTIP_2NUM_CONTROL_BITS_N32IN1_DATA 32
26 #define XMULTIP_2NUM_CONTROL_ADDR_N32IN2_DATA 0x18
27 #define XMULTIP_2NUM_CONTROL_BITS_N32IN2_DATA 32
28 #define XMULTIP_2NUM_CONTROL_ADDR_PN32RESOUT_DATA 0x20
29 #define XMULTIP_2NUM_CONTROL_BITS_PN32RESOUT_DATA 32
30 #define XMULTIP_2NUM_CONTROL_ADDR_PN32RESOUT_CTRL 0x24
31
```

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.

```
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 分钟前 (autosaved) Python 3
File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3
In [1]:
1 # coding: utf-8
2 # In[ ]:
3
4 from __future__ import print_function
5 import sys, os
6
7 sys.path.append('/home/xilinx')
8 os.environ['XILINX_XRT'] = '/usr'
9 from pynq import Overlay
10
11 if __name__ == '__main__':
12     print("Entry:", sys.argv[0])
13     print("System argument(s):", len(sys.argv))
14
15     print("Start of " + sys.argv[0] + "\n")
16
17     oi = Overlay("/home/xilinx/jupyter_notebooks/Multip2Num.bit")
18     regIP = oi.multip_2num_0
19
20     for i in range(9):
21         print("=====")
22         for j in range(9):
23             regIP.write(0x10, i + 1)
24             regIP.write(0x18, j + 1)
25             Res = regIP.read(0x20)
26             print(str(i + 1) + " * " + str(j + 1) + " = " + str(Res))
27         print("=====")
28     print("Exit process")
29
30 Entry: /usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py
31 System argument(s): 3
32 Start of "/usr/local/share/pynq-venv/lib/python3.8/site-packages/ipykernel_launcher.py"
33
34 1 * 1 = 1
35 1 * 2 = 2
36 1 * 3 = 3
37 1 * 4 = 4
38 1 * 5 = 5
39 1 * 6 = 6
40 1 * 7 = 7
41 1 * 8 = 8
42 1 * 9 = 9
43
44 2 * 1 = 2
45 2 * 2 = 4
46 2 * 3 = 6
47 2 * 4 = 8
48 2 * 5 = 10
49 2 * 6 = 12
50 2 * 7 = 14
51 2 * 8 = 16
52 2 * 9 = 18
53
54 3 * 1 = 3
55 3 * 2 = 6
56 3 * 3 = 9
57 3 * 4 = 12
58 3 * 5 = 15
59 3 * 6 = 18
60 3 * 7 = 21
61 3 * 8 = 24
62 3 * 9 = 27
63
64 4 * 1 = 4
65 4 * 2 = 8
66 4 * 3 = 12
67 4 * 4 = 16
68 4 * 5 = 20
69 4 * 6 = 24
70 4 * 7 = 28
71 4 * 8 = 32
72 4 * 9 = 36
73
74 5 * 1 = 5
75 5 * 2 = 10
76 5 * 3 = 15
77 5 * 4 = 20
78 5 * 5 = 25
79 5 * 6 = 30
80 5 * 7 = 35
81 5 * 8 = 40
82 5 * 9 = 45
83
84 6 * 1 = 6
85 6 * 2 = 12
86 6 * 3 = 18
87 6 * 4 = 24
88 6 * 5 = 30
89 6 * 6 = 36
90 6 * 7 = 42
91 6 * 8 = 48
92 6 * 9 = 54
93
94 7 * 1 = 7
95 7 * 2 = 14
96 7 * 3 = 21
97 7 * 4 = 28
98 7 * 5 = 35
99 7 * 6 = 42
100 7 * 7 = 49
101 7 * 8 = 56
102 7 * 9 = 63
103
104 8 * 1 = 8
105 8 * 2 = 16
106 8 * 3 = 24
107 8 * 4 = 32
108 8 * 5 = 40
109 8 * 6 = 48
110 8 * 7 = 56
111 8 * 8 = 64
112 8 * 9 = 72
113
114 9 * 1 = 9
115 9 * 2 = 18
116 9 * 3 = 27
117 9 * 4 = 36
118 9 * 5 = 45
119 9 * 6 = 54
120 9 * 7 = 63
121 9 * 8 = 72
122 9 * 9 = 81
123
124 Exit process
125
126 In [ ]: 1
127 In [ ]: 1
```