tags: `MSOC`

MSOC Lab1 Report

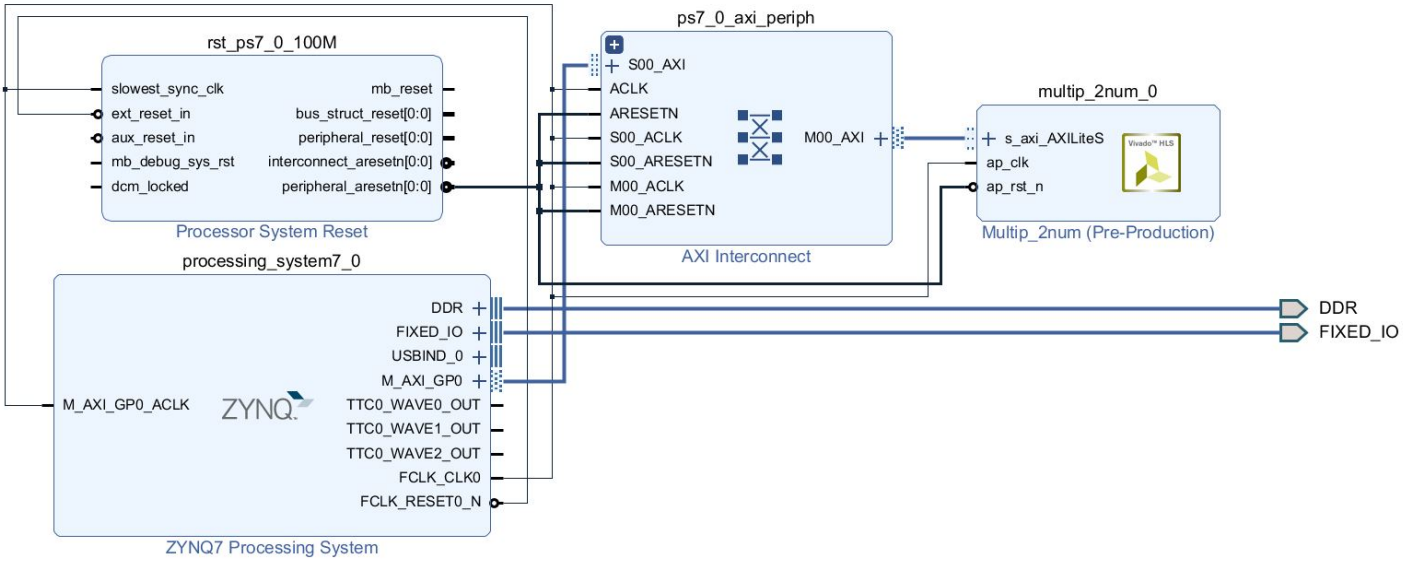
Objective

After completing this lab, we will be able to learn:

- Use Vivado-HLS API to generate a multiplier RTL IP with AXILiteS interface from System-C code.
- Configure the Processing system and generate implementation of the design.
- Build a software project and verify the multiplier design functionality in hardware.
- [Additional] Study the analysis report of Vivado-HLS.

System diagram

This lab intends to develop a **32-bit to 32-bit Multiplier** through the ASILiteS interface.



Interface

This design utilize the **AXILiteS interface** to transfer the data. The following table shows the usage for some address.

Addr. (+Base addr.)	Usage
0x10	Input data 1
0x18	Input data 2
0x20	Output data
0x24	AP_Valid

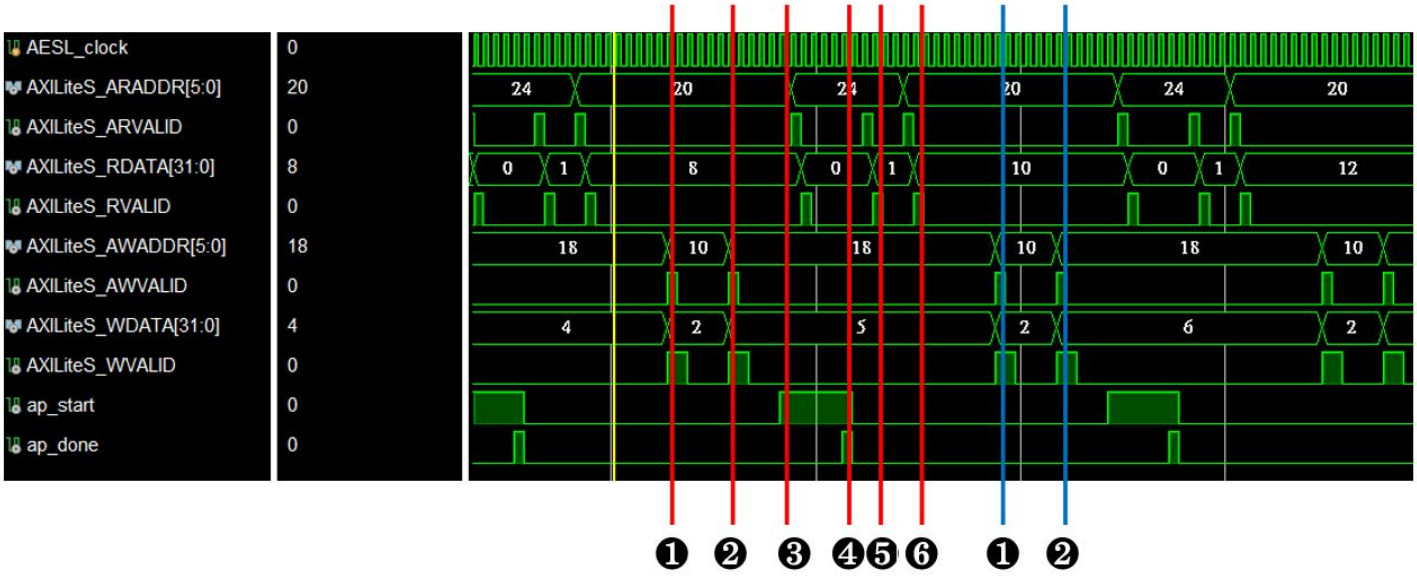
C code

Multiplication.cpp: multiply two 32-bit int. data and assign to a 32-bit int. pointer.

```
#include "multiplication.h"
void multip_2num(int32_t n32In1, int32_t n32In2, int32_t* pn32ResOut)
{
        *pn32ResOut = n32In1 * n32In2;
        return;
}
```

Cosim waveform

The following figure shows a portion of the co-sim waveform.



We can see that:

❶ AWValid = 1, AWADDR = 10, Wvalid = 1, WDATA = 2: Read data 2 into reg In1
❷ AWValid = 1, AWADDR = 18, Wvalid = 1, WDATA = 5: Read data 5 into reg In2
❸ ap_start = 1: The multiplier start its calculation
❹ ap_done = 1: The multiploer done calculations after 6 cycles
❺ RValid = 1 & ARADDR = 24 & RDATA = 1: The output of Kernel function is valid.
❻ RValid = 1 & ARADDR = 20 & RDATA = 10: The output data of Kernel function is 10.

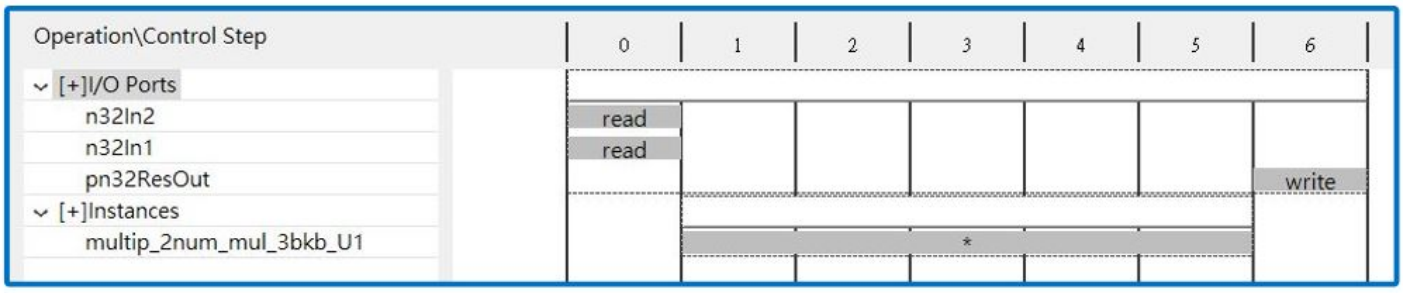
Performance and Analysis

We can see that it **consumes 6 cycles** for calculation and the **estimated clock rate is 3.95ns** from the report below:

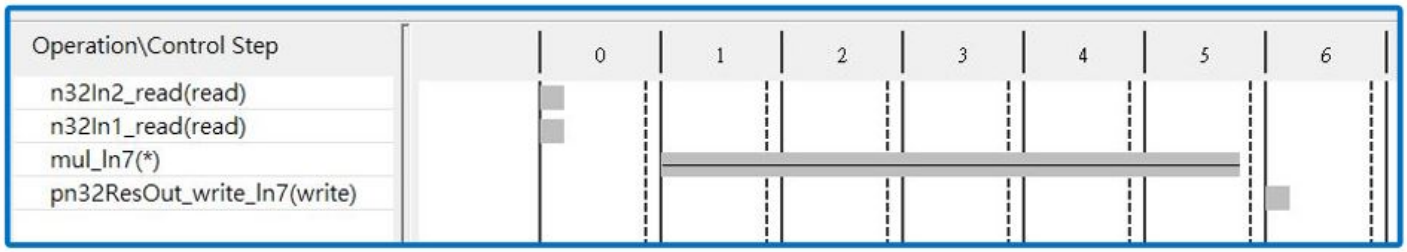
Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.950 ns	0.63 ns

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
6	6	30.000 ns	30.000 ns	6	6	none

The resource allocation plot is as follows:



The scheduling plot is as follows.



Analysis:

- Notice that **the dotted line indicates the clock uncertainty5th cycle, therefore, the estimated clock rate is 3.95ns** (default set to 12.5% of the cycle time).
- **The kernel only consumes 3.95ns in the 5th cycle, therefore, the estimated clock rate is 3.95ns**.
- **Notice that the kernels seems to consume 7 cycles; however, read and write can be done in the same cycle, so it is report to be done in 6 cycles** (refer to Vivado synthesize userguide).

Utilization

The utilization estimation is shown below:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	0	3	359	233	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	41	-
Register	-	-	103	-	-
Total	0	3	462	274	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0

Python code

```
1 from pynq import Overlay
2
3 ol = Overlay("/home/xilinx/IPBitFile/Multip2Num.bit")
4 regIP = ol.multip_2num_0
5
6 for i in range(9):
7     for j in range(9):
8         regIP.write(0x10, i + 1)
9         regIP.write(0x18, j + 1)
10        Res = regIP.read(0x20)
11        print(str(i + 1) + " * " + str(j + 1) + " = " + str(Res))
```

- Line 3: Build Overlay class and burn the bitstream file into the Board.
- Line 8: write value “i+1” to addr. BS+0x10 (In1 for AXILiteS)
- Line 9: write value “j+1” to addr. BS+0x18 (In2 for AXILiteS)
- Line 10: read data from addr. BS+0x20 (Out for AXILiteS)

Bugs encountered

- Remember to specify the board type when create the project
- Some anti-virus software would block cosim without notification.

Learnt

- Meanings of “Directives” in Vivado-HLS.
- Meanings for each step in the flow: C-sim -> C-syn -> Co-sim.
- The data transfer of the AXILiteS interface from Co-sim waveform.

- Connect the board and PC to the same local network.
- Transfer files and watch IP of the board through USB-UART cable.
- Some PYNQ package usage on python.
- The overall flow from HLS the RTL code, implememtation, burn on the board and run on python program.