SOC LAB#5 Test (Labi)Report

Group no: 4

Members:

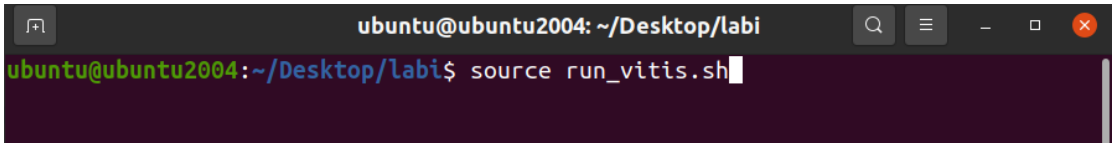
M11107410 羅善寬

M11107004 曹榮恩

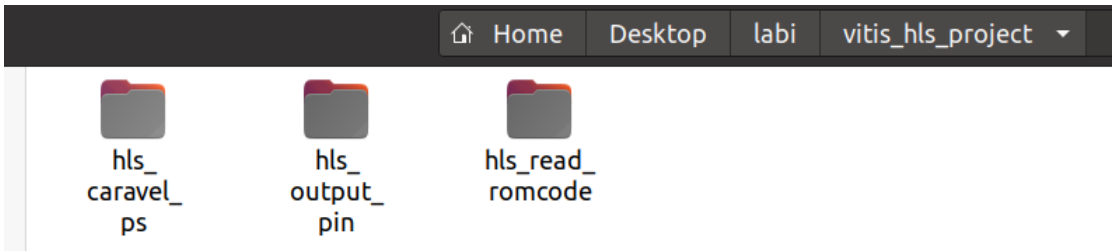
M11107409 陳昱碩

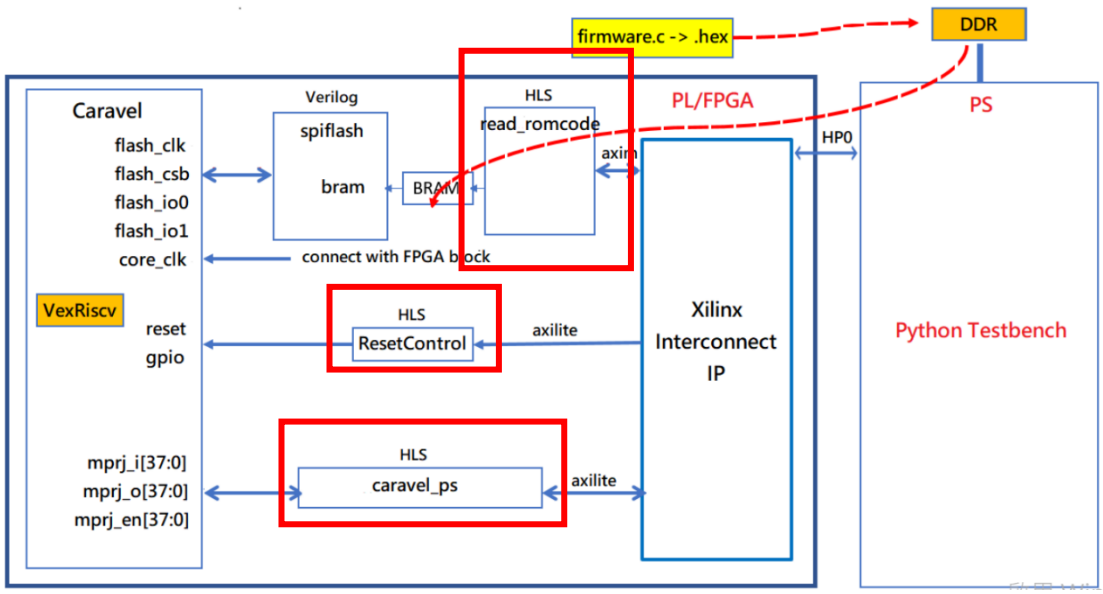
1. Run_vitis

1.1 Vitis HLS(生成 IP)



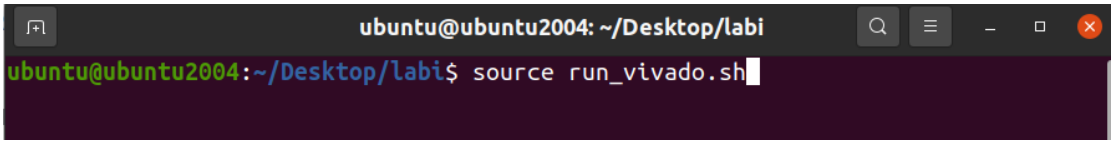
以下三個硬體由 vitis 產生 ip。



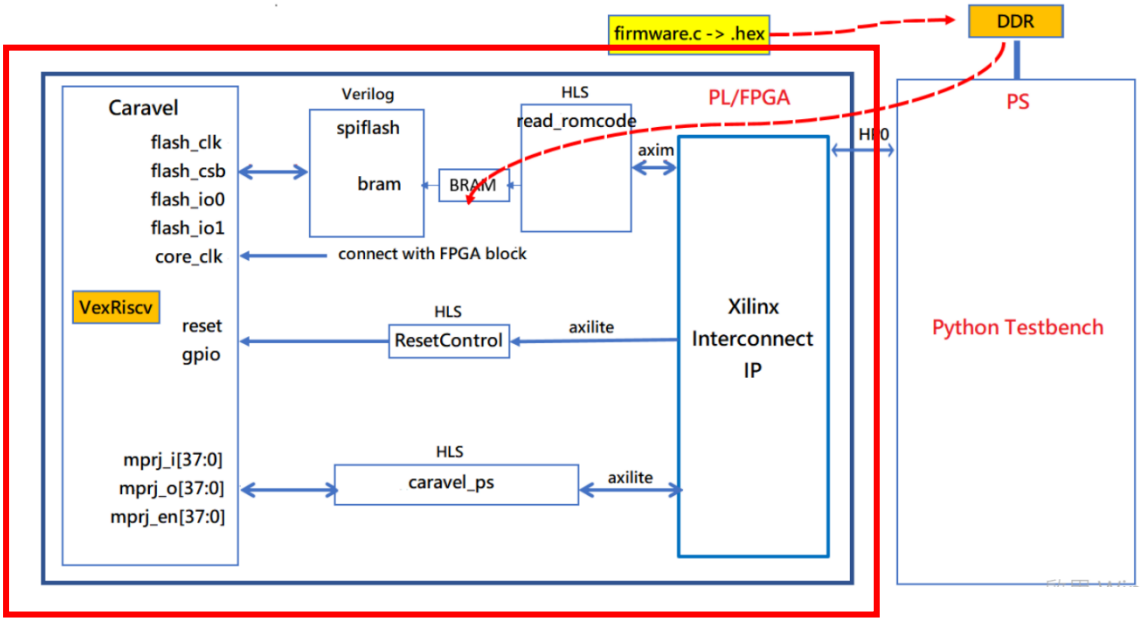


2. Run_vivado

2.1 Vivado (IP、SOC 整合)

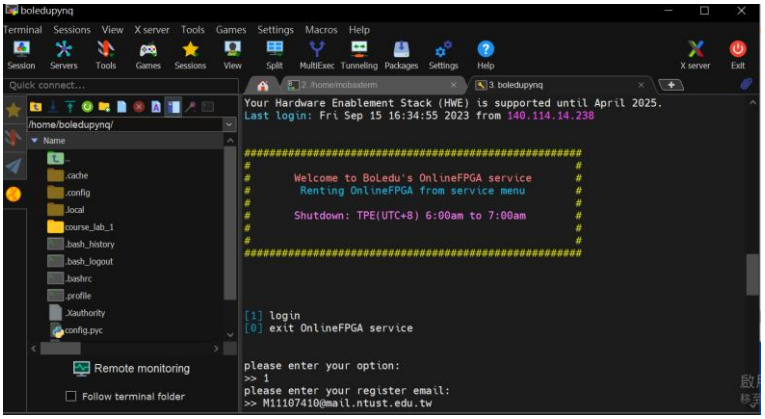


將整個系統整合起來。

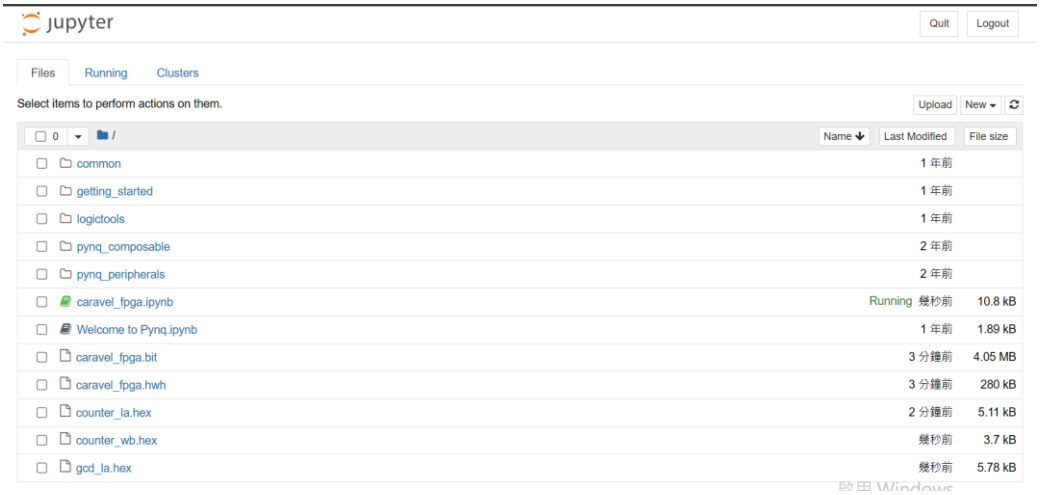


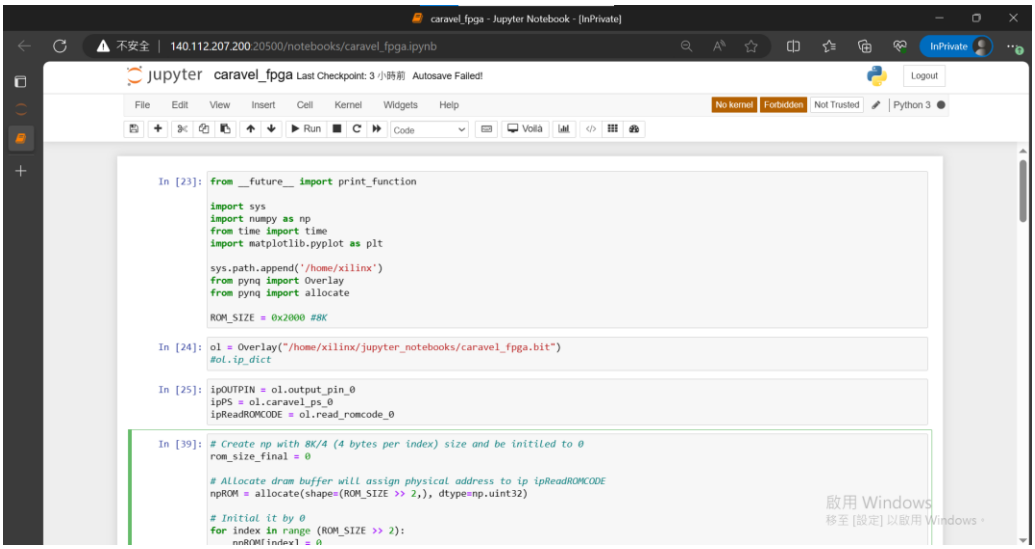
3. Online FPGA

租借一個線上的 FPGA 板子。

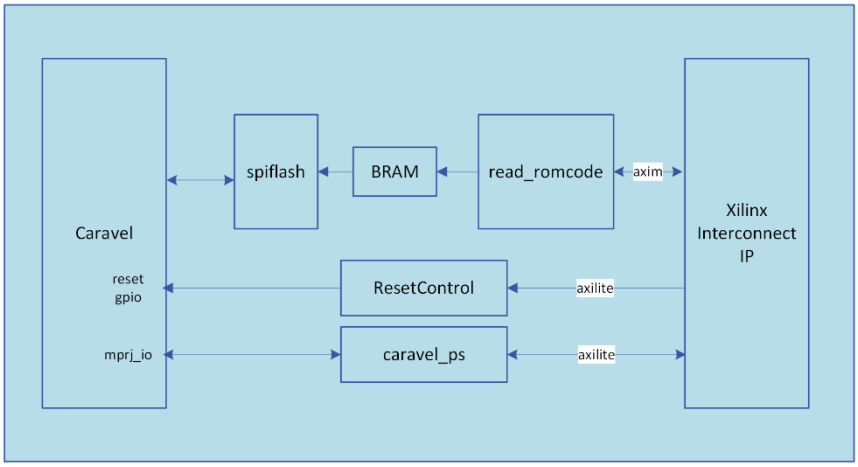


租借板子後在 jupyter 上用 python 驗證。





4. Block diagram



5. FPGA utilization

5.1 run_vivado

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	5327	0	0	53200	10.01
LUT as Logic	5149	0	0	53200	9.68
LUT as Memory	178	0	0	17400	1.02
LUT as Distributed RAM	18	0			
LUT as Shift Register	160	0			
Slice Registers	6051	0	0	106400	5.69
Register as Flip Flop	6051	0	0	106400	5.69
Register as Latch	0	0	0	106400	0.00
F7 Muxes	169	0	0	26600	0.64
F8 Muxes	47	0	0	13300	0.35

5.2 run_vivado_gcd

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	6457	0	0	53200	12.14
LUT as Logic	6279	0	0	53200	11.80
LUT as Memory	178	0	0	17400	1.02
LUT as Distributed RAM	18	0			
LUT as Shift Register	160	0			
Slice Registers	6082	0	0	106400	5.72
Register as Flip Flop	6082	0	0	106400	5.72
Register as Latch	0	0	0	106400	0.00
F7 Muxes	168	0	0	26600	0.63
F8 Muxes	47	0	0	13300	0.35

6. Explain the function of IP in this design

6.1 HLS

➢ read_romcode

一開始 .hex 檔是載入在 ps side 的 DDR 裡面，需要將.hex 載入到硬體 bram 裡面，所以要透過 read_romcode 這 ip 來傳遞 data。

IP read ROM code from DRAM	IP write ROM code to DRAM
• PS set m_axi_BUS0 base address • PS send read command • IP start read DRAM ROM code • PS wait for IP done	• PS set m_axi_BUS1 base address • PS send write command • IP start write ROM code to DRAM • PS wait for IP done

➢ ResetControl

當 ps side 要與 caravel 的 gpio、reset 溝通所需的 ResetControl ip。

執行開始前所需的 reset，我們從 python code 可對 ResetControl ip 設定 reset 信號，caravel 就開始跑，當 firmware 做完之後，可以將 reset release 掉。

• Output 1 or 0 signal, which used to assert/de-assert Caravel reset pin • Provide AXI LITE interface for PS CPU to control the output • Implement by HLS and export IP for Vivado project usage

➢ caravel_ps

當 ps side 與 caravel 的 mprj_io 溝通所需的 caravel_ps ip。

當 caravel 跑完之後，透過 mprj_io 經由 caravel_ps ip 送一些信息跟 python code 溝通，來驗證功能是否正確。

• Provide AXI Lite interface for PS CPU to read the MPRJ_IO/OUT/EN bits • Implement by HLS and export IP for Vivado project usage

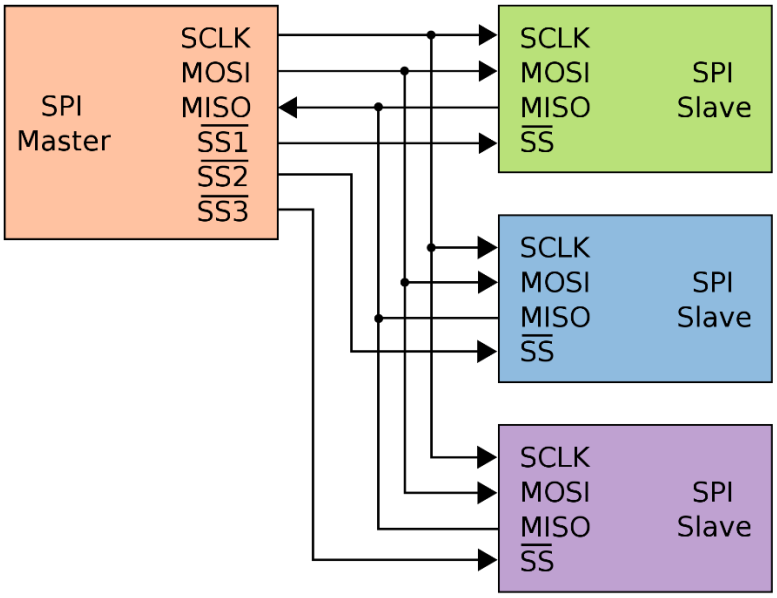
6.2 Verilog

➢ Spiflash

經由將 bram 裡".hex"的 data 儲存起來，並供給 caravel 使用。

• Implement SPI slave device, only support read command (0x03) • Return data from BRAM to Caravel

Master Output Slave Input (MOSI)	從 master 傳輸到 slave
Master Input Slave Output (MISO)	從 slave 傳輸到 master
Serial Clock (SCK)	Master 與 slave 的共用 clk
SS	選擇要傳輸的 slave



7. Run these workload on caravel FPGA & Screenshot of Execution result on all workload

7.1 counter_wb.hex

open “counter_wb.hex”。

```
npROM_index = 0
npROM_offset = 0
fiROM = open("counter_wb.hex", "r+")
#fiROM = open("counter_la.hex", "r+")
#fiROM = open("gcd_la.hex", "r+")
```

如果所執行的為“counter_wb.hex”，查看 mprj_io 的輸出，0x1c 腳位值為 0xab61。

In [9]:

```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab610008
0x20 =  0x2
0x34 =  0xfff7
0x38 =  0x37
```

7.2 counter_la.hex

open “counter_la.hex”。

```
npROM_index = 0
npROM_offset = 0
#fiROM = open("counter_wb.hex", "r+")
fiROM = open("counter_la.hex", "r+")
#fiROM = open("gcd_la.hex", "r+")
```

如果所執行的為“counter_la.hex”，查看 mprj_io 的輸出，0x1c 腳位值為 0xab51。

In [33]:
```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab518474
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

7.3 gcd_la.hex

open “gcd_la.hex”。

```
npROM_index = 0
npROM_offset = 0
#fiROM = open("counter_wb.hex", "r+")
#fiROM = open("counter_la.hex", "r+")
fiROM = open("gcd_la.hex", "r+")
```

如果所執行的為“gcd_la.hex”，查看 mprj_io 的輸出，0x1c 腳位值為 0xab51。

In [17]:
```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab510041
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

8. Screenshot of Execution result on all workload

open “.hex”。

In [1]:
```
from __future__ import print_function

import sys
import numpy as np
from time import time
import matplotlib.pyplot as plt

sys.path.append('/home/xilinx')
from pynq import Overlay
from pynq import allocate

ROM_SIZE = 0x2000 #8K
```

In [3]:
```
ol = Overlay("/home/xilinx/jupyter_notebooks/caravel_fpga.bit")
#ol.ip_dict
```

In [4]:
```
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
```

In [5]:
```
# Create np with 8K/4 (4 bytes per index) size and be initiled to 0
rom_size_final = 0

# Allocate dram buffer will assign physical address to ip ipReadROMCODE
npROM = allocate(shape=(ROM_SIZE >> 2,), dtype=np.uint32)

# Initial it by 0
for index in range (ROM_SIZE >> 2):
    npROM[index] = 0

npROM_index = 0
npROM_offset = 0
fiROM = open("counter_wb.hex", "r+")
#fiROM = open("counter_la.hex", "r+")
```

將“.hex”寫進去 bram。

“.hex”裡以”@”開頭。

```
#fiROM = open("counter_la.hex", "r+")
#fiROM = open("gcd_la.hex", "r+")

for line in fiROM:
    # offset header
    if line.startswith('@'):
        # Ignore first char @
        npROM_offset = int(line[1:].strip(b'\x00'.decode()), base = 16)
        npROM_offset = npROM_offset >> 2 # 4byte per offset
        #print (npROM_offset)
        npROM_index = 0
        continue
    #print (line)

    # We suppose the data must be 32bit alignment
    buffer = 0
    bytecount = 0
    for line_byte in line.strip(b'\x00'.decode()).split():
        buffer += int(line_byte, base = 16) << (8 * bytecount)
        bytecount += 1
        # Collect 4 bytes, write to npROM
        if(bytecount == 4):
            npROM[npROM_offset + npROM_index] = buffer
            # Clear buffer and bytecount
            buffer = 0
            bytecount = 0
            npROM_index += 1
            #print (npROM_index)
            continue
    # Fill rest data if not alignment 4 bytes
    if (bytecount != 0):
        npROM[npROM_offset + npROM_index] = buffer
        npROM_index += 1

fiROM.close()
```

```
rom_size_final = npROM_offset + npROM_index
#print (rom_size_final)

#for data in npROM:
#    print (hex(data))
```

In [6]:

```
# 0x00 : Control signals
#        bit 0  - ap_start (Read/Write/COH)
#        bit 1  - ap_done (Read/COR)
#        bit 2  - ap_idle (Read)
#        bit 3  - ap_ready (Read)
#        bit 7  - auto_restart (Read/Write)
#        others - reserved
# 0x10 : Data signal of romcode
#        bit 31~0 - romcode[31:0] (Read/Write)
# 0x14 : Data signal of romcode
#        bit 31~0 - romcode[63:32] (Read/Write)
# 0x1c : Data signal of length_r
#        bit 31~0 - length_r[31:0] (Read/Write)

# Program physical address for the romcode base address
ipReadROMCODE.write(0x10, npROM.device_address)
ipReadROMCODE.write(0x14, 0)
# Program length of moving data
ipReadROMCODE.write(0x1C, rom_size_final)


# ipReadROMCODE start to move the data from rom_buffer to bram
ipReadROMCODE.write(0x00, 1) # IP Start
while (ipReadROMCODE.read(0x00) & 0x04) == 0x00: # wait for done
    continue

print("Write to bram done")

```

```
Write to bram done
```

查看 mprj_io 的輸出。

In [7]:

```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))

```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0x8
0x20 =  0x0
0x34 =  0xfffffff7
0x38 =  0x3f
```

給 reset 為 1 的信號，caravel 會重新執行 firmware。

In [8]:

```
# Release Caravel reset
# 0x10 : Data signal of outpin_ctrl
#        bit 0  - outpin_ctrl[0] (Read/Write)
#        others - reserved
print (ipOUTPIN.read(0x10))
ipOUTPIN.write(0x10, 1)
print (ipOUTPIN.read(0x10))
```

```
0
1
```

執行後“.hex”，查看 mprj_io 的輸出，0x1c 腳位值。

In [9]:

```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab610008
0x20 =  0x2
0x34 =  0xfff7
0x38 =  0x37
```

9. Study caravel_fpga.ipynb, and be familiar with caravel SoC control flow

lab4 的 test bench 由 simulator 來跑中間 interaction。

而 lab5 是由 python code 來跑驗證結果，而中間彼此的 interaction 則由實際的 ip 來完成 data 傳遞溝通。

一開始 .hex 檔是載入在 ps side 的 DDR 裡面，需要將.hex 載入到硬體 bram 裡面，所以要透過 read_romcode 這 ip 來傳遞 data。

其中還有 ps side 要與 caravel 的 gpio、reset 溝通所需的 ResetControl ip，以及 ps side 與 caravel 的 mprj_io 溝通所需的 caravel_ps ip。

執行開始前所需的 reset，我們從 python code 可對 ResetControl ip 設定 reset 信號，caravel 就開始跑，當 firmware 做完之後，可以將 reset release 掉。

當 caravel 跑完之後，透過 mprj_io 經由 caravel_ps ip 送一些信息跟 python code 溝通，來驗證功能是否正確。