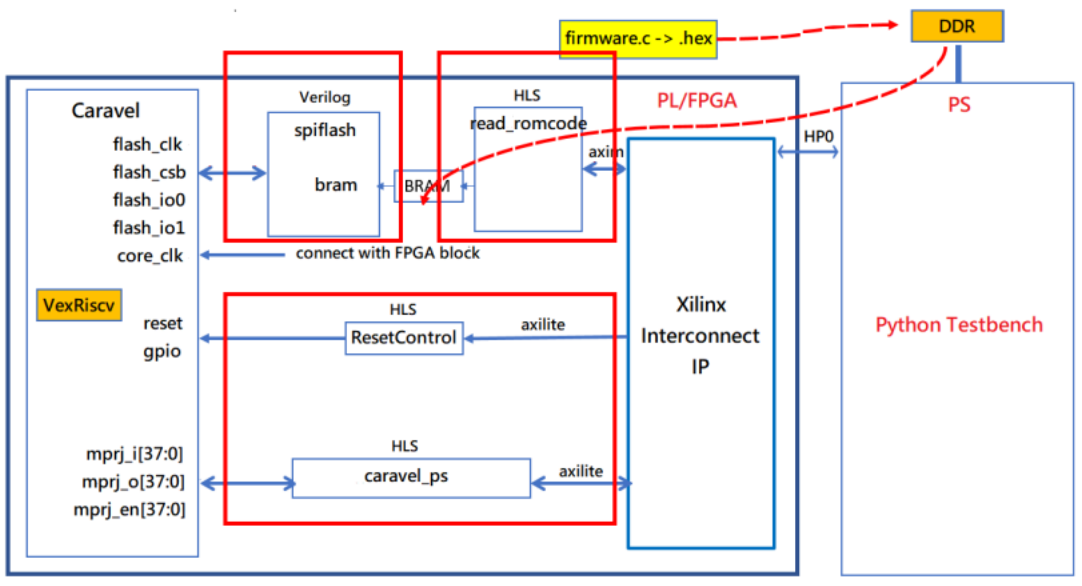
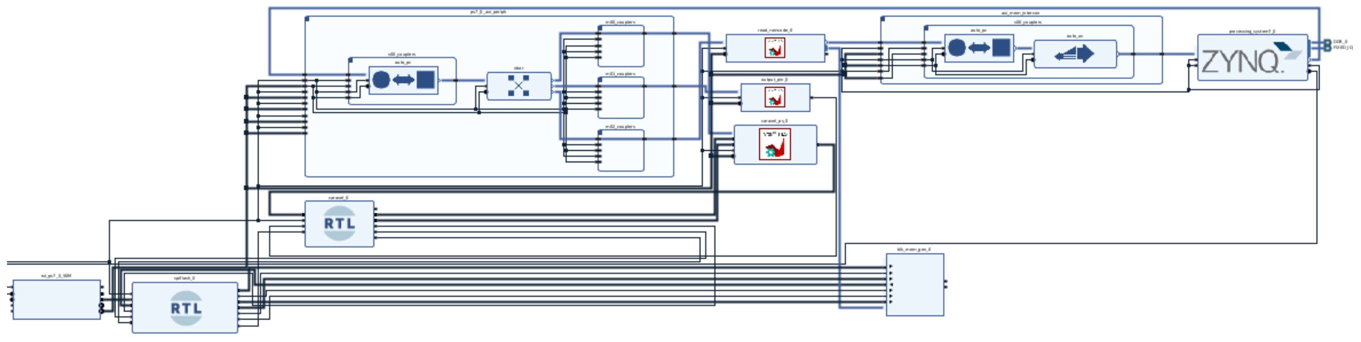
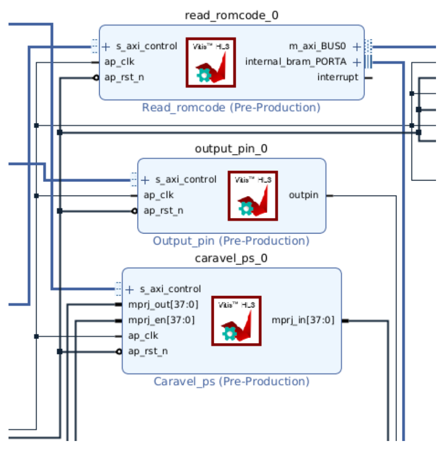
NYCU 電子研究所 系統晶片設計 LAB5

呂紹愷 311510187 江尹凡 311510207 廖智偉 311510216

- **Block diagram**

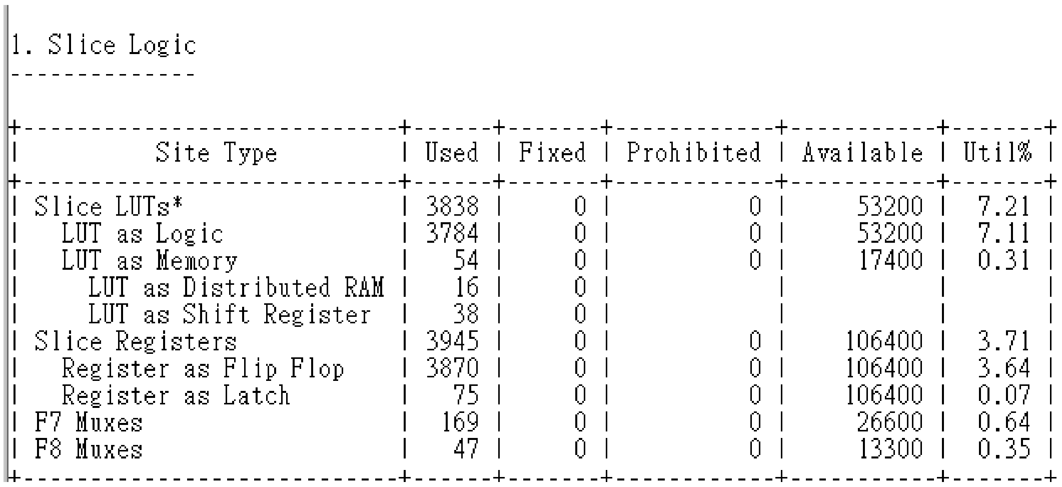




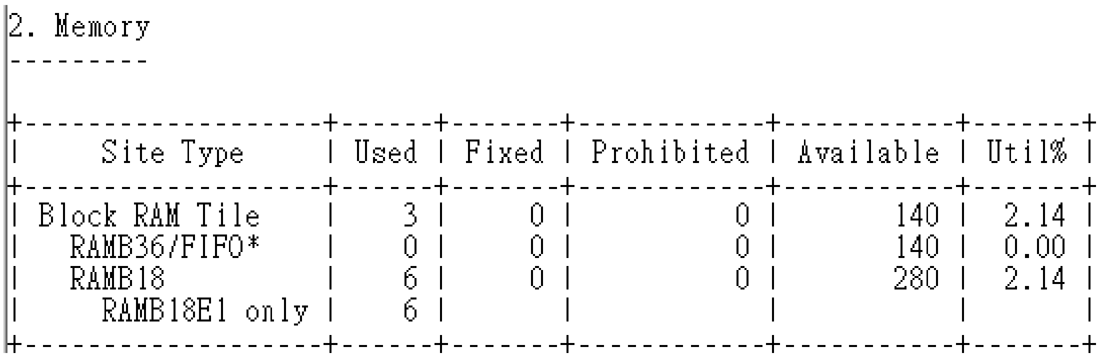


- **FPGA utilization**

Carvel

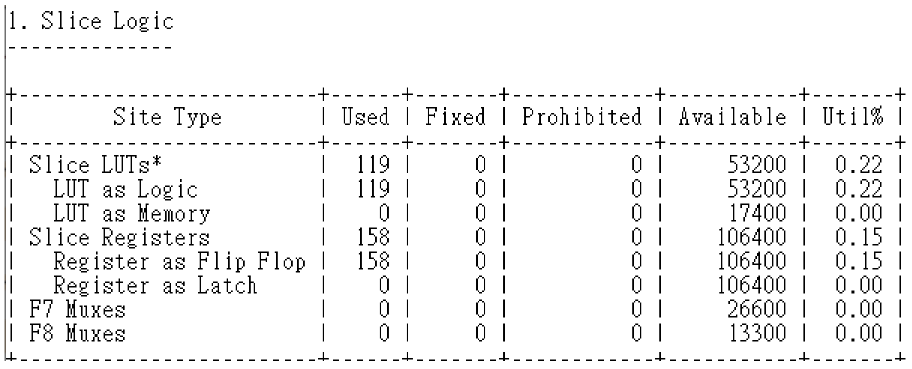


```
1. Slice Logic
--------------

+----------------------------+------+-------+------------+-----------+-------+
|         Site Type          | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*                | 3838 |     0 |          0 |     53200 |  7.21 |
|   LUT as Logic             | 3784 |     0 |          0 |     53200 |  7.11 |
|   LUT as Memory            |   54 |     0 |          0 |     17400 |  0.31 |
|     LUT as Distributed RAM |   16 |     0 |            |           |       |
|     LUT as Shift Register  |   38 |     0 |            |           |       |
| Slice Registers            | 3945 |     0 |          0 |    106400 |  3.71 |
|   Register as Flip Flop    | 3870 |     0 |          0 |    106400 |  3.64 |
|   Register as Latch        |   75 |     0 |          0 |    106400 |  0.07 |
| F7 Muxes                   |  169 |     0 |          0 |     26600 |  0.64 |
| F8 Muxes                   |   47 |     0 |          0 |     13300 |  0.35 |
+----------------------------+------+-------+------------+-----------+-------+
```

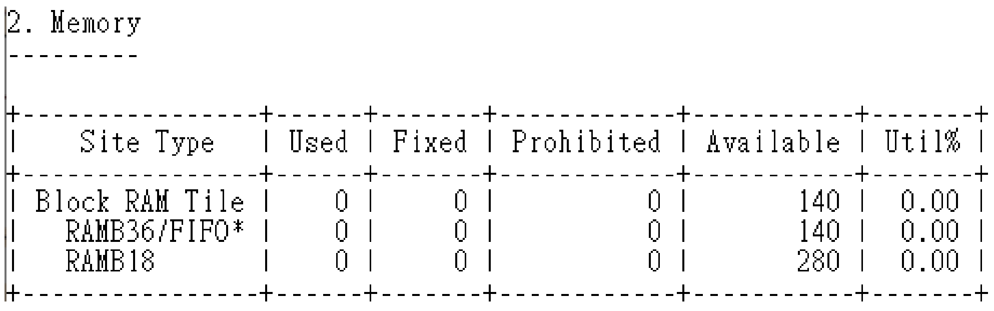


```
2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    3 |     0 |          0 |       140 |  2.14 |
|   RAMB36/FIFO*    |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB18          |    6 |     0 |          0 |       280 |  2.14 |
|     RAMB18E1 only |    6 |       |            |           |       |
+-------------------+------+-------+------------+-----------+-------+
```

caravel_ps

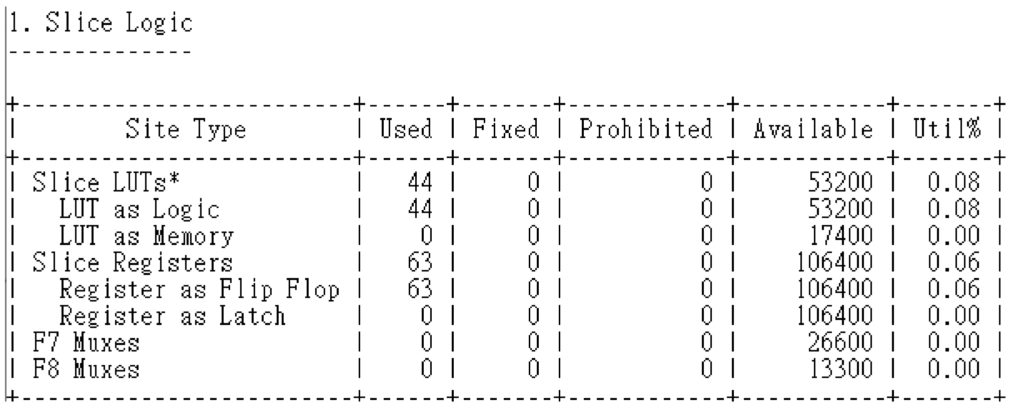


```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |  119 |     0 |          0 |     53200 |  0.22 |
|   LUT as Logic          |  119 |     0 |          0 |     53200 |  0.22 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |  158 |     0 |          0 |    106400 |  0.15 |
|   Register as Flip Flop |  158 |     0 |          0 |    106400 |  0.15 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

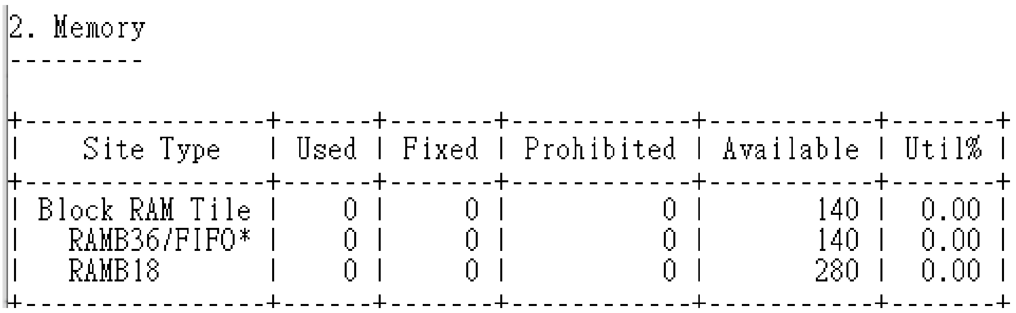


```
2. Memory
---------

+----------------+------+-------+------------+-----------+-------+
|    Site Type   | Used | Fixed | Prohibited | Available | Util% |
+----------------+------+-------+------------+-----------+-------+
| Block RAM Tile |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB18       |    0 |     0 |          0 |       280 |  0.00 |
+----------------+------+-------+------------+-----------+-------+
```

spiflash

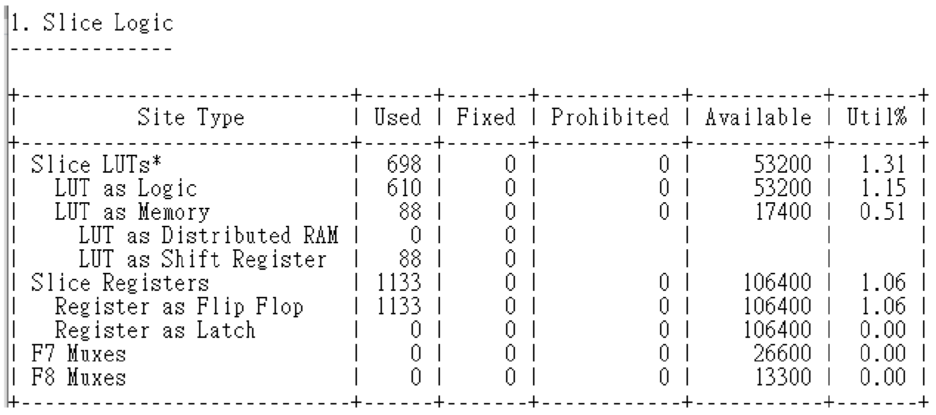


```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   44 |     0 |          0 |     53200 |  0.08 |
|   LUT as Logic          |   44 |     0 |          0 |     53200 |  0.08 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |   63 |     0 |          0 |    106400 |  0.06 |
|   Register as Flip Flop |   63 |     0 |          0 |    106400 |  0.06 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

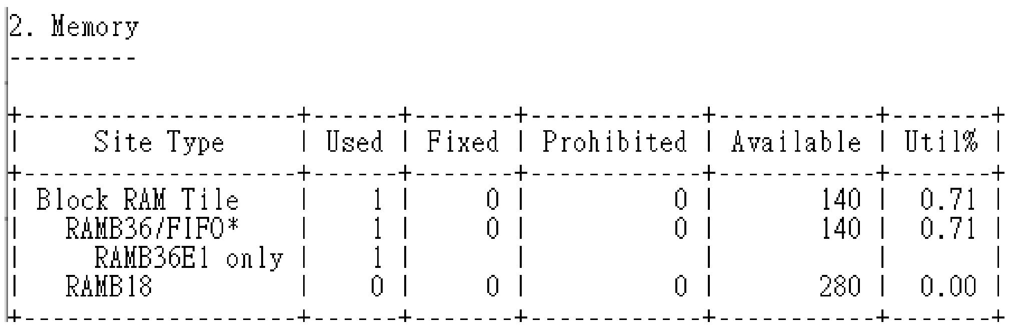


```
2. Memory
---------

+----------------+------+-------+------------+-----------+-------+
|    Site Type   | Used | Fixed | Prohibited | Available | Util% |
+----------------+------+-------+------------+-----------+-------+
| Block RAM Tile |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB18       |    0 |     0 |          0 |       280 |  0.00 |
+----------------+------+-------+------------+-----------+-------+
```

read_romcod

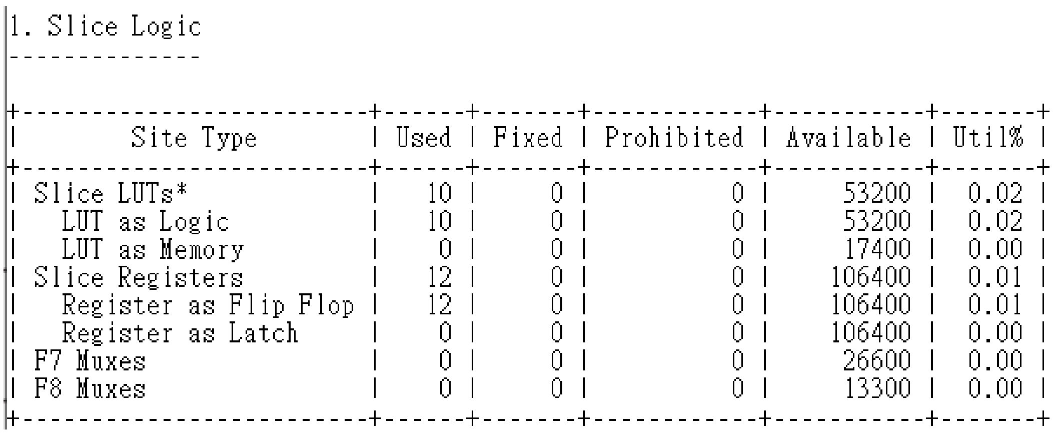


```
1. Slice Logic
--------------

+----------------------------+------+-------+------------+-----------+-------+
|          Site Type         | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*                |  698 |     0 |          0 |     53200 |  1.31 |
|   LUT as Logic             |  610 |     0 |          0 |     53200 |  1.15 |
|   LUT as Memory            |   88 |     0 |          0 |     17400 |  0.51 |
|     LUT as Distributed RAM |    0 |     0 |            |           |       |
|     LUT as Shift Register  |   88 |     0 |            |           |       |
| Slice Registers            | 1133 |     0 |          0 |    106400 |  1.06 |
|   Register as Flip Flop    | 1133 |     0 |          0 |    106400 |  1.06 |
|   Register as Latch        |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                   |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                   |    0 |     0 |          0 |     13300 |  0.00 |
+----------------------------+------+-------+------------+-----------+-------+
```

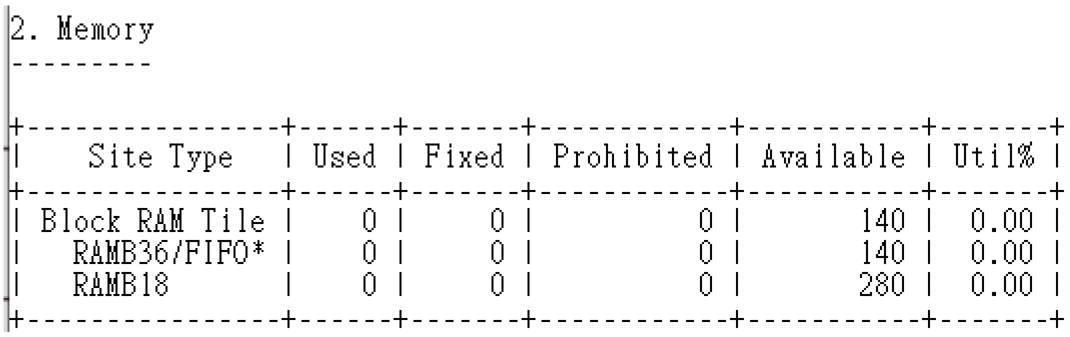


```
2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    1 |     0 |          0 |       140 |  0.71 |
|   RAMB36/FIFO*    |    1 |     0 |          0 |       140 |  0.71 |
|     RAMB36E1 only |    1 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |       280 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```

ResetControl

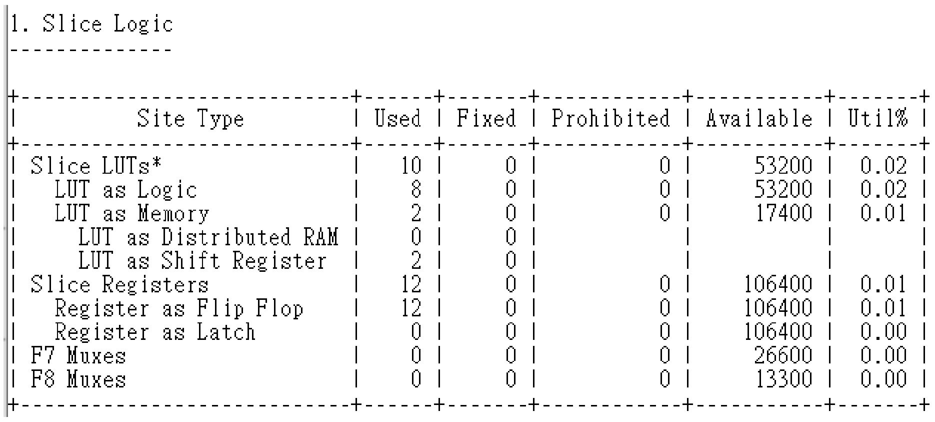


```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |   10 |     0 |          0 |     53200 |  0.02 |
|   LUT as Logic          |   10 |     0 |          0 |     53200 |  0.02 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |   12 |     0 |          0 |    106400 |  0.01 |
|   Register as Flip Flop |   12 |     0 |          0 |    106400 |  0.01 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

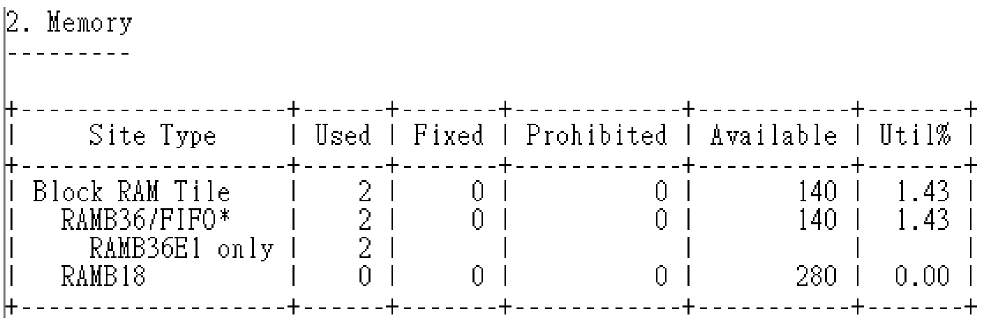


```
2. Memory
---------

+----------------+------+-------+------------+-----------+-------+
|    Site Type   | Used | Fixed | Prohibited | Available | Util% |
+----------------+------+-------+------------+-----------+-------+
| Block RAM Tile |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB18       |    0 |     0 |          0 |       280 |  0.00 |
+----------------+------+-------+------------+-----------+-------+
```

BRAM



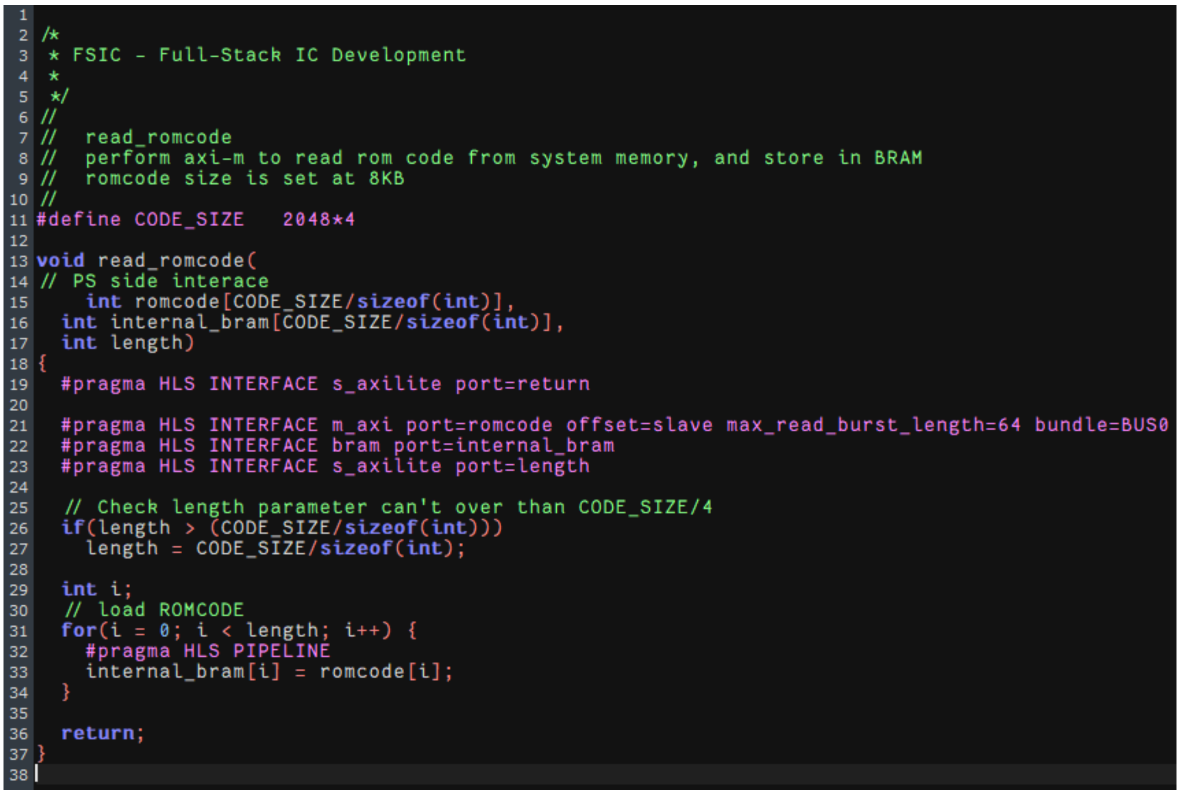
```
1. Slice Logic
--------------

+----------------------------+------+-------+------------+-----------+-------+
|          Site Type         | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*                |   10 |     0 |          0 |     53200 |  0.02 |
|   LUT as Logic             |    8 |     0 |          0 |     53200 |  0.02 |
|   LUT as Memory            |    2 |     0 |          0 |     17400 |  0.01 |
|     LUT as Distributed RAM |    0 |     0 |            |           |       |
|     LUT as Shift Register  |    2 |     0 |            |           |       |
| Slice Registers            |   12 |     0 |          0 |    106400 |  0.01 |
|   Register as Flip Flop    |   12 |     0 |          0 |    106400 |  0.01 |
|   Register as Latch        |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                   |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                   |    0 |     0 |          0 |     13300 |  0.00 |
+----------------------------+------+-------+------------+-----------+-------+
```



```
2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    2 |     0 |          0 |       140 |  1.43 |
|   RAMB36/FIFO*    |    2 |     0 |          0 |       140 |  1.43 |
|     RAMB36E1 only |    2 |       |            |           |       |
|   RAMB18          |    0 |     0 |          0 |       280 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
```

- **Explain the function of IP in this design**
    - **HLS：read_romcode, ResetControl, caravel_ps**

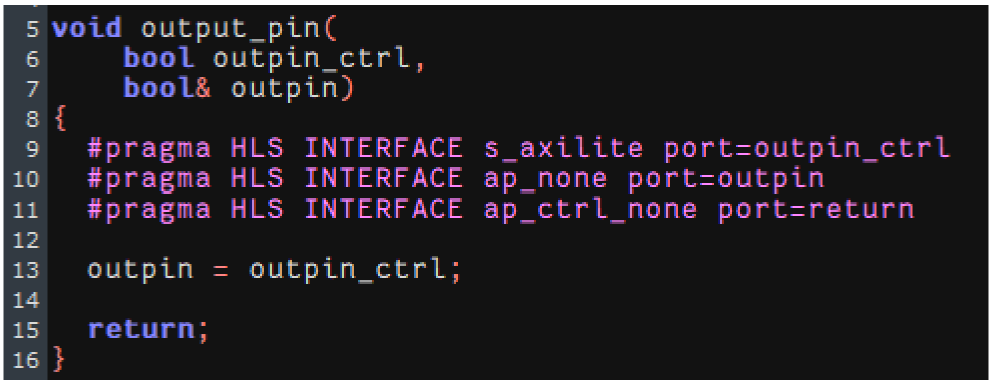
      read_romcode:



```

/*
 * FSIC - Full-Stack IC Development
 *
 */
//
//   read_romcode
//   perform axi-m to read rom code from system memory, and store in BRAM
//   romcode size is set at 8KB
//
#define CODE_SIZE   2048*4

void read_romcode(
// PS side interface
    int romcode[CODE_SIZE/sizeof(int)],
  int internal_bram[CODE_SIZE/sizeof(int)],
  int length)
{
  #pragma HLS INTERFACE s_axilite port=return

  #pragma HLS INTERFACE m_axi port=romcode offset=slave max_read_burst_length=64 bundle=BUS0
  #pragma HLS INTERFACE bram port=internal_bram
  #pragma HLS INTERFACE s_axilite port=length

  // Check length parameter can't over than CODE_SIZE/4
  if(length > (CODE_SIZE/sizeof(int)))
    length = CODE_SIZE/sizeof(int);

  int i;
  // load ROMCODE
  for(i = 0; i < length; i++) {
    #pragma HLS PIPELINE
    internal_bram[i] = romcode[i];
  }

  return;
}

```

此段 code 為透過 AXI-M interface 來讀取系統內的 ROM code，然後再將其存在 BRAM 中，而 AXI-Lite 介面則用於控制信號。首先會先檢查輸入的長度參數是否超過 ROM 的大小，如果是則修正成合法數值，最後透過 for 迴圈，把整個 ROM 複製到 BRAM 中。

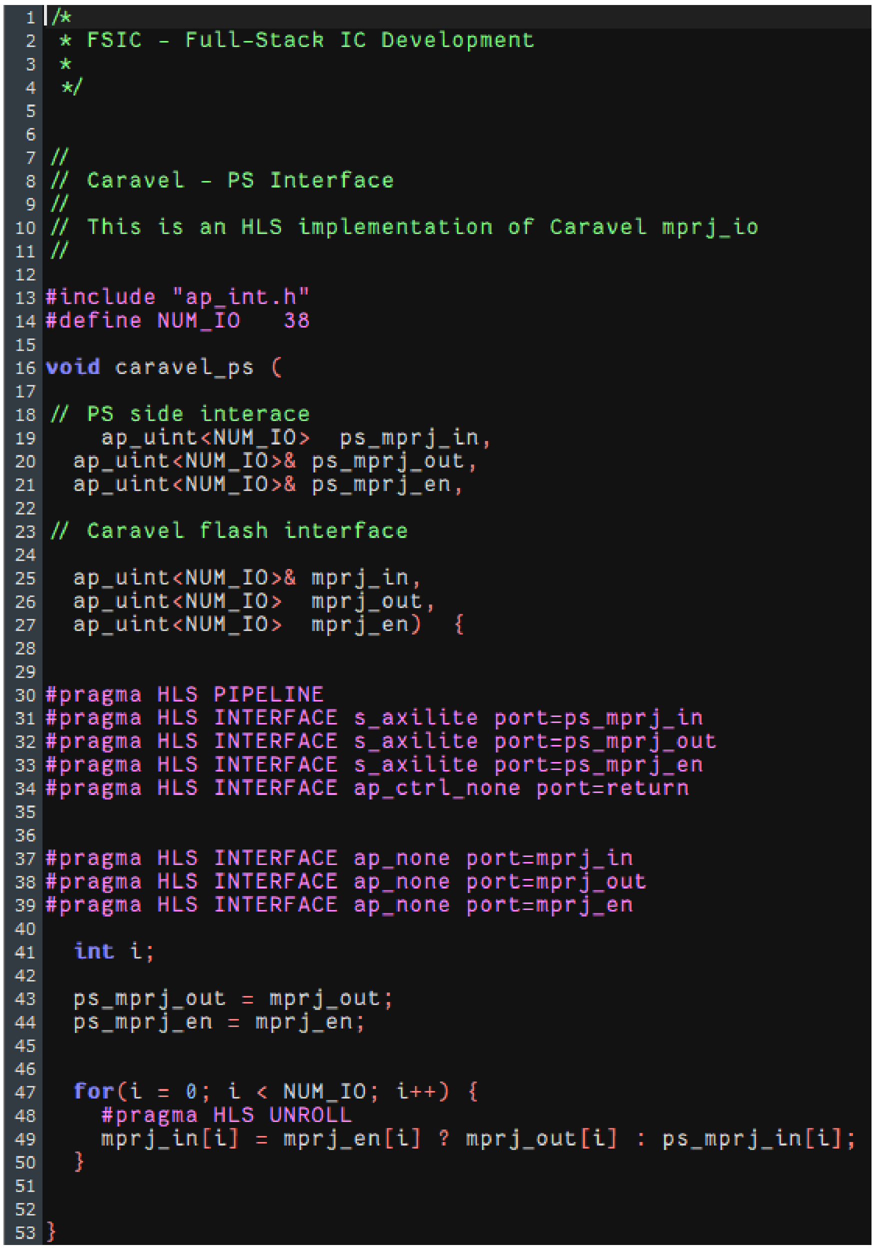
ResetControl:



```
void output_pin(
      bool outpin_ctrl,
      bool& outpin)
{
  #pragma HLS INTERFACE s_axilite port=outpin_ctrl
  #pragma HLS INTERFACE ap_none port=outpin
  #pragma HLS INTERFACE ap_ctrl_none port=return

  outpin = outpin_ctrl;

  return;
}
```

此段 code 為 reset 訊號的控制，將控制信號的值直接給輸出訊號的變數。

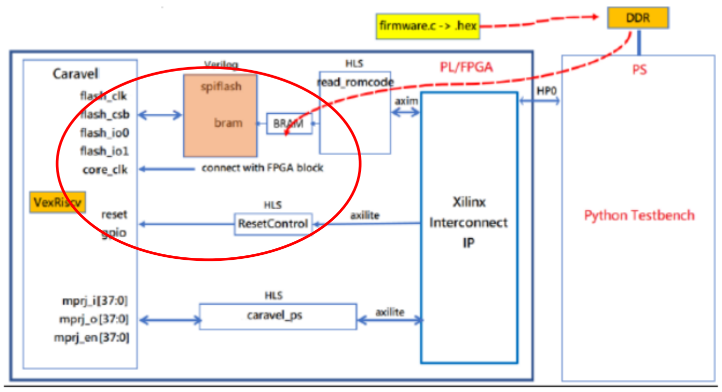
caravel_ps:

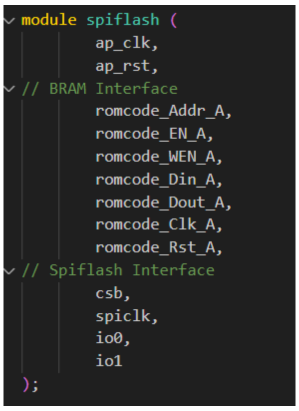


```
/*
 * FSIC - Full-Stack IC Development
 *
 */


//
// Caravel - PS Interface
//
// This is an HLS implementation of Caravel mprj_io
//

#include "ap_int.h"
#define NUM_IO   38

void caravel_ps (

// PS side interace
    ap_uint<NUM_IO>  ps_mprj_in,
  ap_uint<NUM_IO>& ps_mprj_out,
  ap_uint<NUM_IO>& ps_mprj_en,

// Caravel flash interface

  ap_uint<NUM_IO>& mprj_in,
  ap_uint<NUM_IO>  mprj_out,
  ap_uint<NUM_IO>  mprj_en)  {


#pragma HLS PIPELINE
#pragma HLS INTERFACE s_axilite port=ps_mprj_in
#pragma HLS INTERFACE s_axilite port=ps_mprj_out
#pragma HLS INTERFACE s_axilite port=ps_mprj_en
#pragma HLS INTERFACE ap_ctrl_none port=return


#pragma HLS INTERFACE ap_none port=mprj_in
#pragma HLS INTERFACE ap_none port=mprj_out
#pragma HLS INTERFACE ap_none port=mprj_en

  int i;

  ps_mprj_out = mprj_out;
  ps_mprj_en = mprj_en;


  for(i = 0; i < NUM_IO; i++) {
    #pragma HLS UNROLL
    mprj_in[i] = mprj_en[i] ? mprj_out[i] : ps_mprj_in[i];
  }


}
```

此段 code 實現了對 Caravel mprj 訊號的 PS 介面的設計，其中 PS 端的訊號有 ps_mprj_in, ps_mprj_out, ps_mprj_en；Caravel 端的訊號為 mprj_in, mprj_out, mprj_en。 使用 for 迴圈來跑，藉由 mprj_en 來控制，決定輸入的訊號為何。

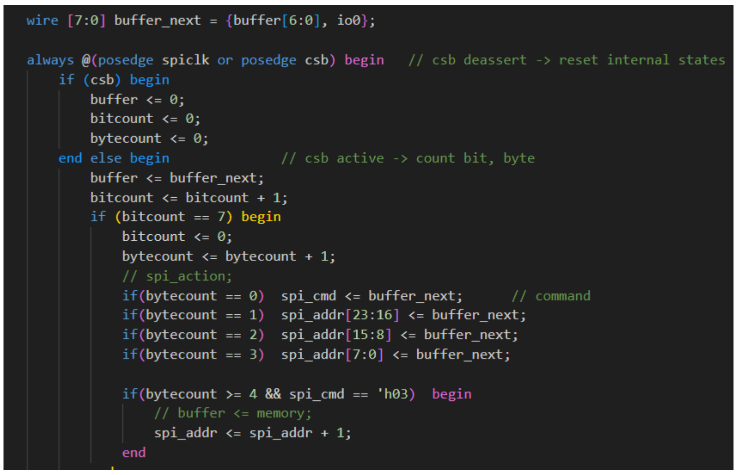
➢ Verilog ：spiflash





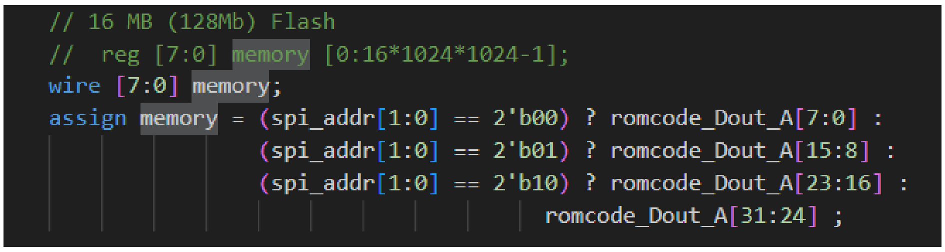
```
module spiflash (
        ap_clk,
        ap_rst,
// BRAM Interface
        romcode_Addr_A,
        romcode_EN_A,
        romcode_WEN_A,
        romcode_Din_A,
        romcode_Dout_A,
        romcode_Clk_A,
        romcode_Rst_A,
// Spiflash Interface
        csb,
        spiclk,
        io0,
        io1
);
```

把前面 python 寫進 bram 的指令，利用這個 verilog 寫的 spiflash 中繼站傳 data 回 bram，他會從 bram 讀出來，carvel 依狀況跟 spiflash 讀取資料。



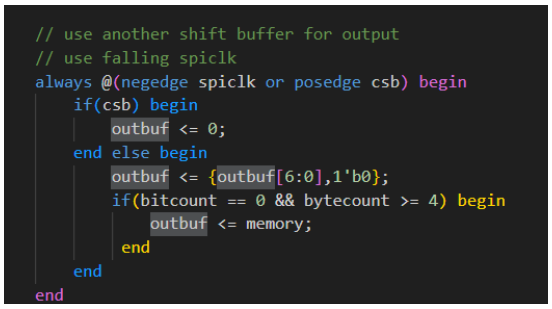
```
wire [7:0] buffer_next = {buffer[6:0], io0};

always @(posedge spiclk or posedge csb) begin   // csb deassert -> reset internal states
    if (csb) begin
        buffer <= 0;
        bitcount <= 0;
        bytecount <= 0;
    end else begin               // csb active -> count bit, byte
        buffer <= buffer_next;
        bitcount <= bitcount + 1;
        if (bitcount == 7) begin
            bitcount <= 0;
            bytecount <= bytecount + 1;
            // spi_action;
            if(bytecount == 0)  spi_cmd <= buffer_next;       // command
            if(bytecount == 1)  spi_addr[23:16] <= buffer_next;
            if(bytecount == 2)  spi_addr[15:8] <= buffer_next;
            if(bytecount == 3)  spi_addr[7:0] <= buffer_next;

            if(bytecount >= 4 && spi_cmd == 'h03)  begin
                // buffer <= memory;
                spi_addr <= spi_addr + 1;
            end
```

carvel 會將指令根要讀的 bram_addr(spi_addr)由 spi 介面的 io0，1bit、1bit 輸入給 spiflash ，spiflash 會慢慢讀近來後 decoder 指令(cmd)跟地址



```
// 16 MB (128Mb) Flash
//  reg [7:0] memory [0:16*1024*1024-1];
wire [7:0] memory;
assign memory = (spi_addr[1:0] == 2'b00) ? romcode_Dout_A[7:0] :
                (spi_addr[1:0] == 2'b01) ? romcode_Dout_A[15:8] :
                (spi_addr[1:0] == 2'b10) ? romcode_Dout_A[23:16] :
                                           romcode_Dout_A[31:24] ;
```

再根據 spi_addr 從 bram 讀出的 data 為 memory

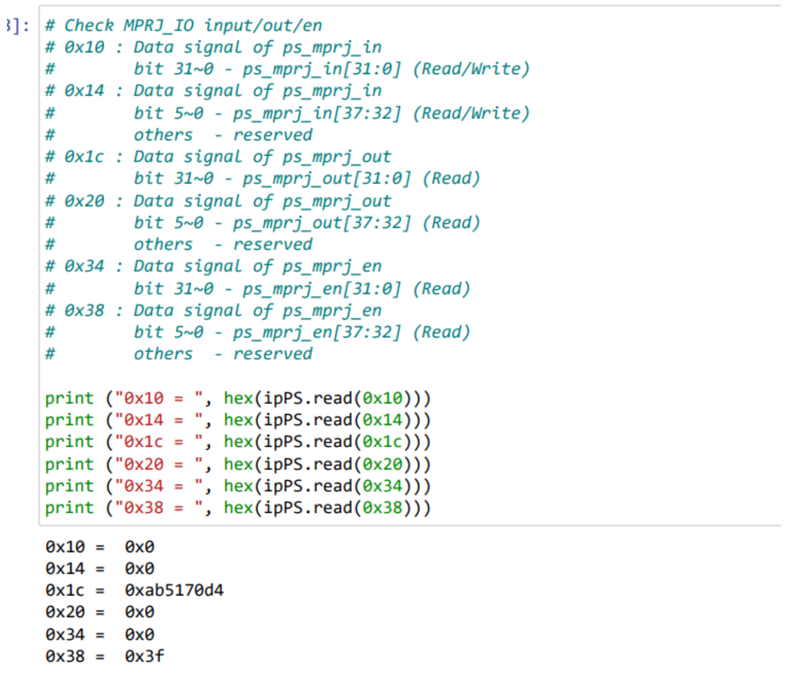


```
// use another shift buffer for output
// use falling spiclk
always @(negedge spiclk or posedge csb) begin
    if(csb) begin
        outbuf <= 0;
    end else begin
        outbuf <= {outbuf[6:0],1'b0};
        if(bitcount == 0 && bytecount >= 4) begin
            outbuf <= memory;
        end
    end
end
```

memory 再給 outbuf，依序輸出給 carvel

- **Run these workload on caravel FPGA**
- **Screenshot of Execution result on all workload**

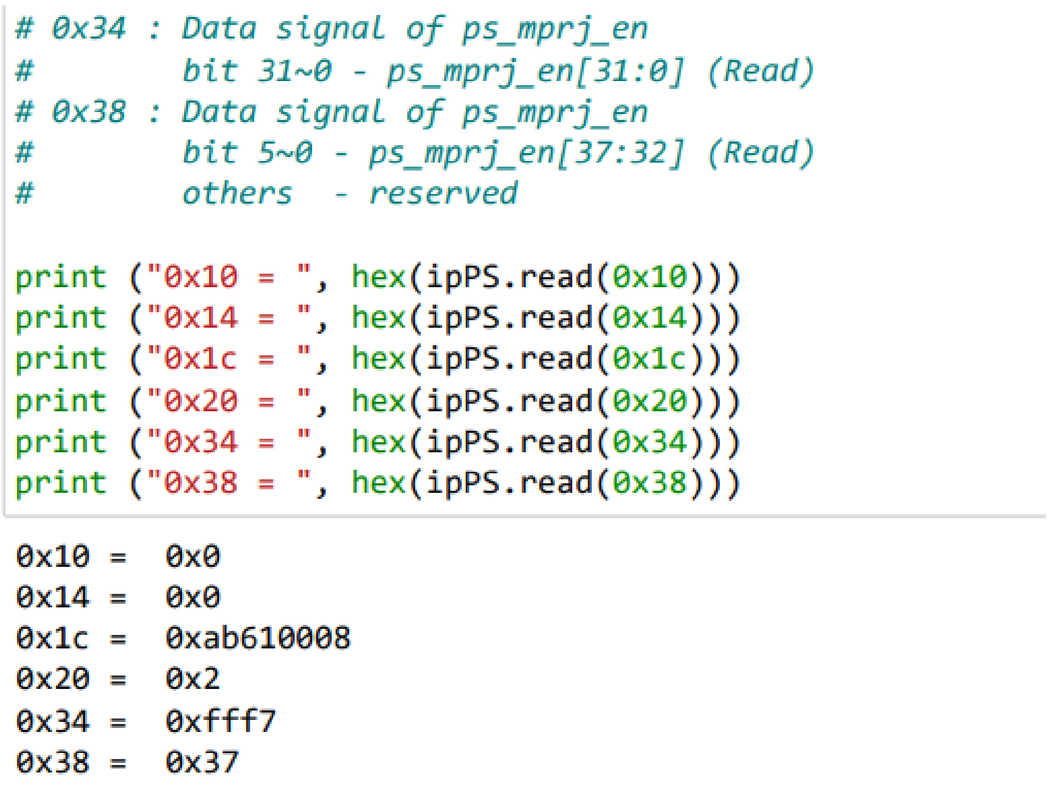
Couter_la



```
]: # 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 =  0xab5170d4
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

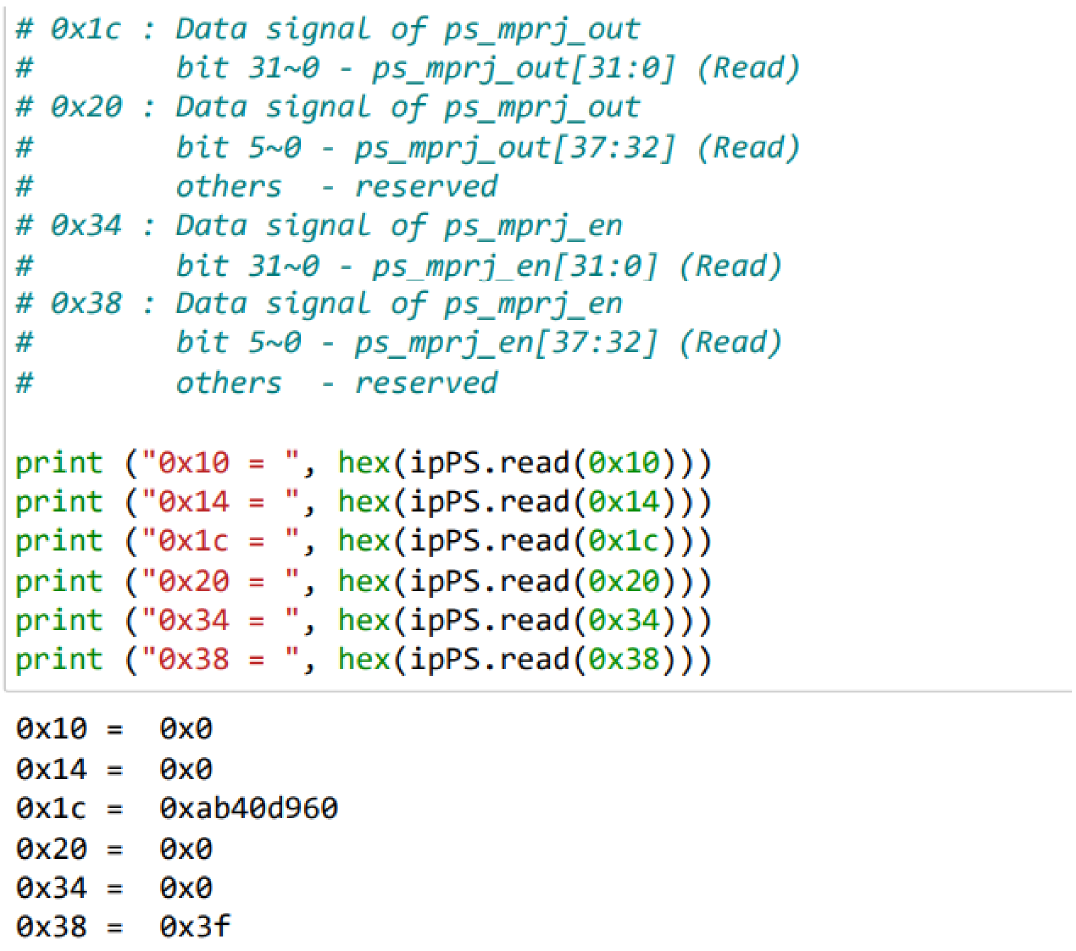
Couter_wb



```
# 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
```

gcd_la



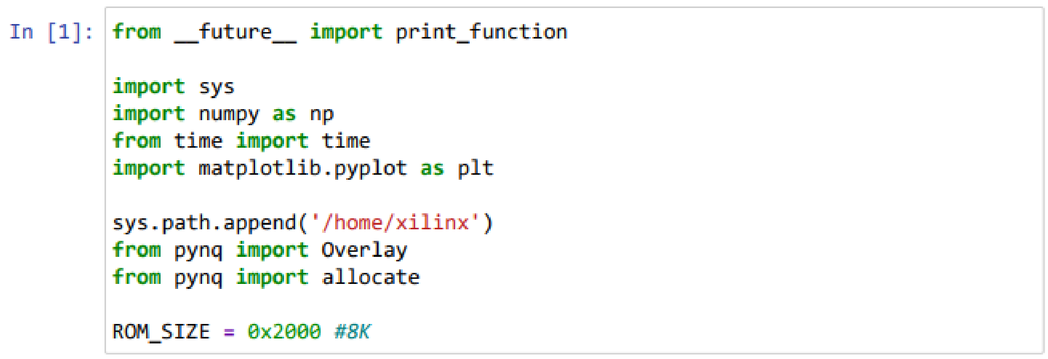
```
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        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 =  0xab40d960
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

- **Study caravel_fpga.ipynb, and be familiar with caravel SoC control flow**
  這次 Lab 和 Lab1 和 Lab2 一樣，需要遠端到 FPGA 測試版上使用 Jupyter Notebook Python 來驗證。

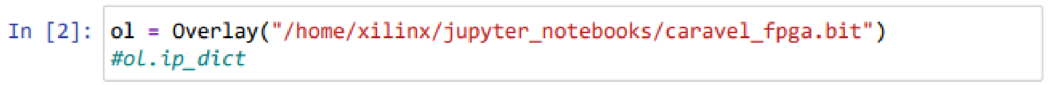
  需要的檔案:
  1. firmwares:.hex
  2. FPGA bitstream:.bit
  3. HWH file:.hwh
  4. Jupyter Notebook example code:.ipynq

  流程:
  Step1:由於 Jupyter Notebook 上是使用 Python code，因此需要先 import 一些 python 內的 model、function 等內建資料，以利後續 code 執行。



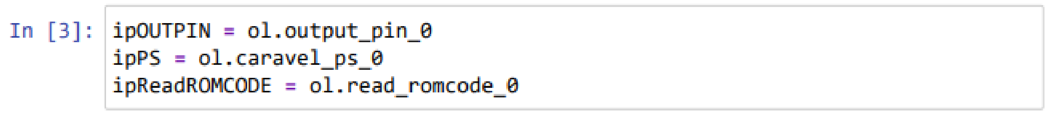
```
In [1]: from __future__ import print_function

        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
```

  Step2:匯入.bit 檔



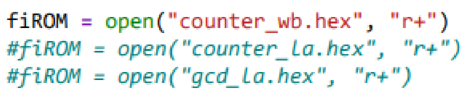
```
In [2]: ol = Overlay("/home/xilinx/jupyter_notebooks/caravel_fpga.bit")
        #ol.ip_dict
```

  Step3:透過.bit 檔轉換成 ip 相關資訊



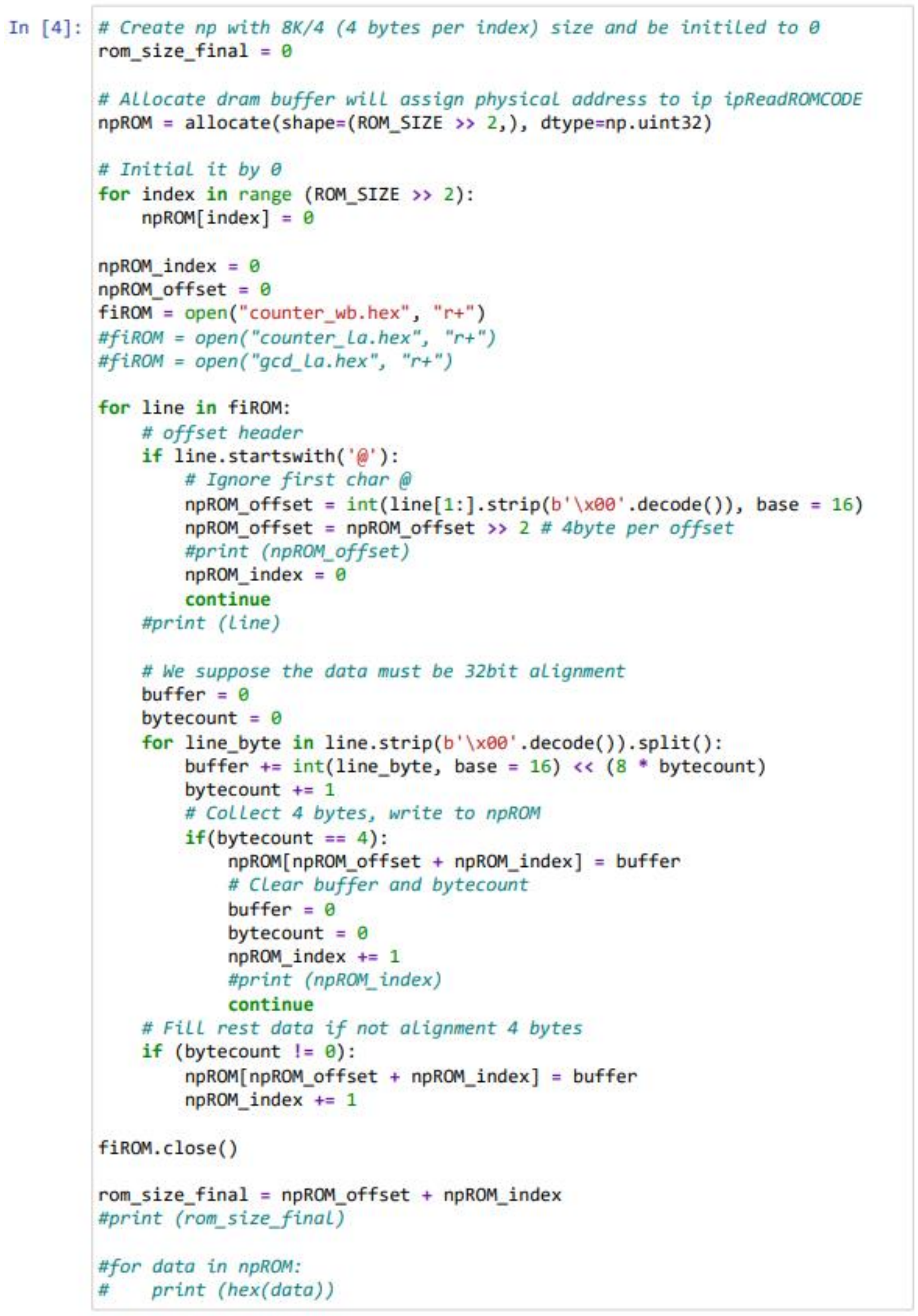
```
In [3]: ipOUTPIN = ol.output_pin_0
        ipPS = ol.caravel_ps_0
        ipReadROMCODE = ol.read_romcode_0
```

  Step4:這次 Lab 需要驗證 counter_wb 和 counter_la，可以自行選擇要註解哪一行來執行要得 firmwave



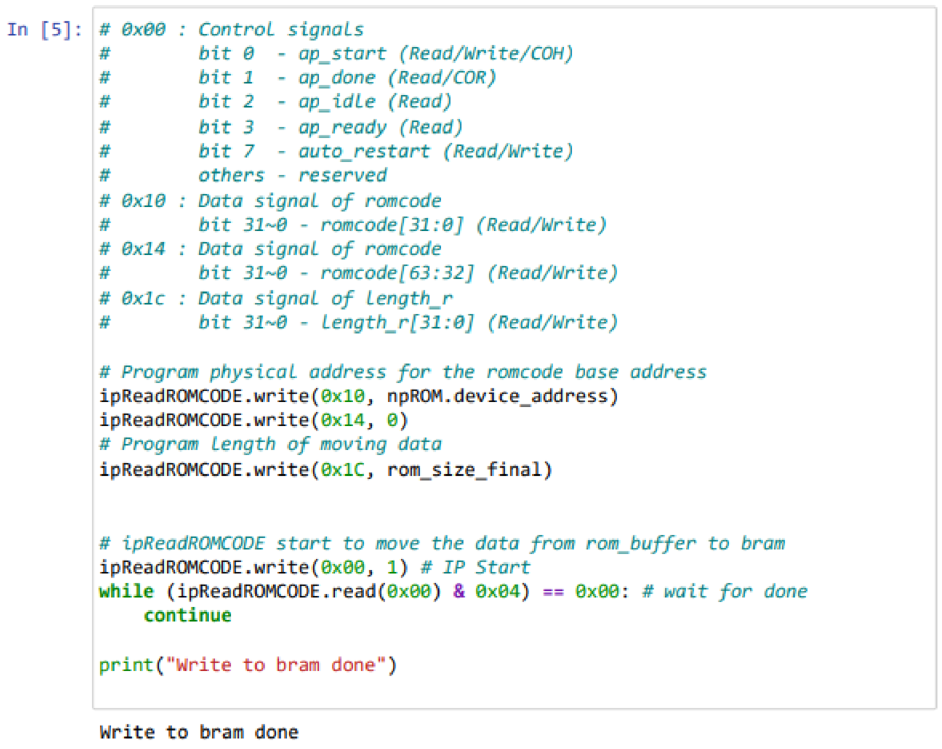
```
fiROM = open("counter_wb.hex", "r+")
#fiROM = open("counter_la.hex", "r+")
#fiROM = open("gcd_la.hex", "r+")
```

  Step5:計算 ROM size



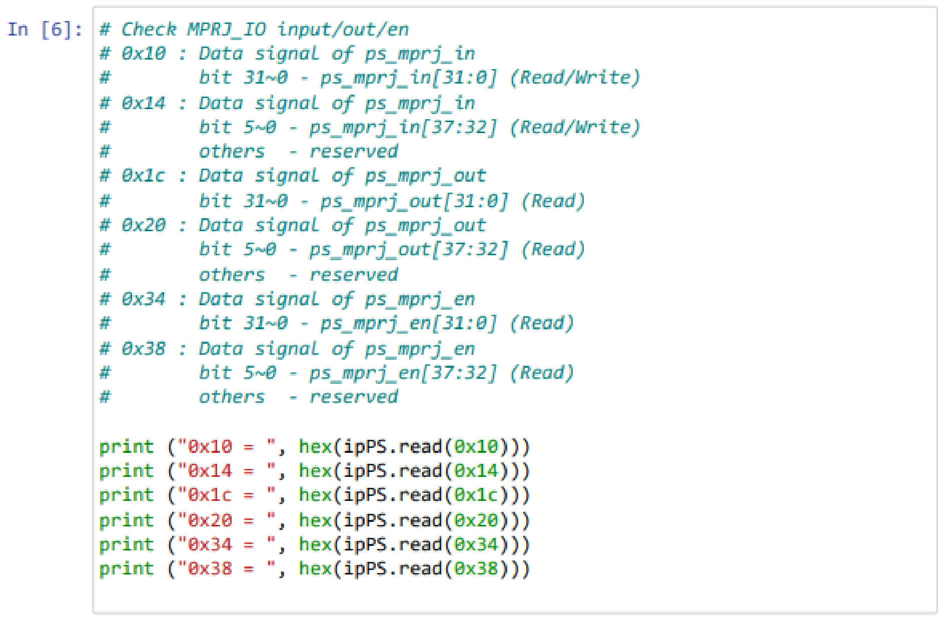
```
In [4]: # Create np with 8K/4 (4 bytes per index) size and be initiled to 0
        rom_size_final = 0

        # 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+")
        #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))
```

Step6:將資料寫進 Bram



```
In [5]: # 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
```

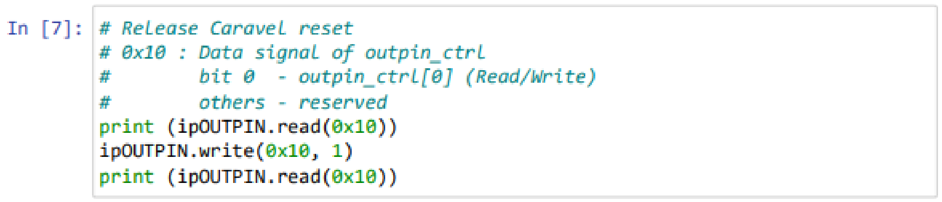
Step7:查看尚未執行 firmwave code 前，mprj_io[31:0]內的 data



```
In [6]: # 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)))
```

D

Step8:Release Caravel reset 訊號，並開始執行 firmwave code



```
In [7]: # 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))
```

Step9: 查看執行 firmwave code 後，mprj_io[31:0]內的 data

```
In [8]: # 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
```