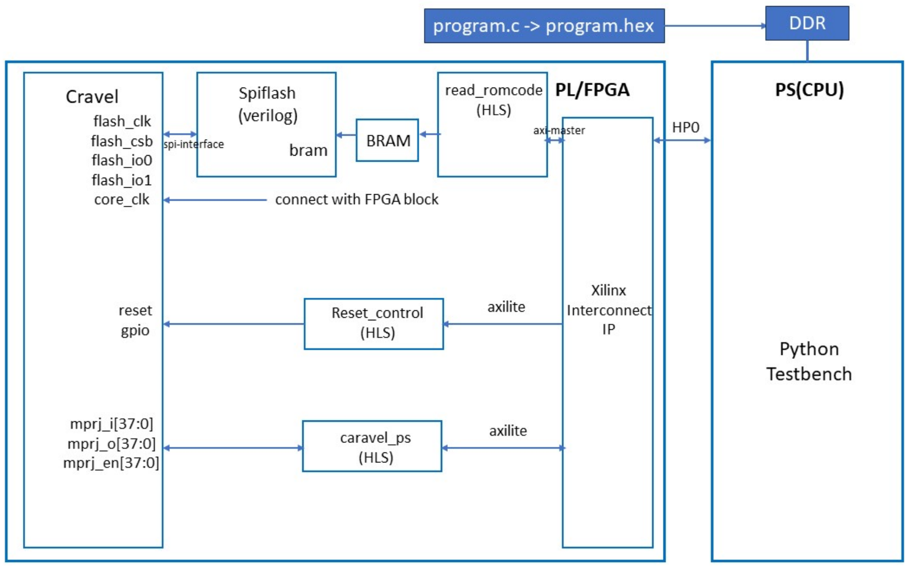
112061603 李柏叡
112061620 郭邑哲
112061621 貢暐家

SOC Design Lab5

- Block diagram:



- Utilization:

```
+----------------------------+------+-------+------------+-----------+-------+
|         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 |
+----------------------------+------+-------+------------+-----------+-------+
```

- Explain the function of IP in this design:

1. HLS:

   I. Read_romcode:
   將 program.hex 從 DDR 透過 axi master interface 傳到此 IP 後，以 pipeline 的形式將資料輸出到 BRAM，因為 axi master interface 不能直接和 BRAM 作互動，因此透過 Read_romcode 做連結。

   II. Reset_control(Output pin):
   PS 透過 axilite 控制 output pin 決定 Caravel 是否 RESET (0:reset 1:execute)

   III. Caravel_ps:
   主要功用為 PS 針對 Caravel 的 mprj port 做 read or write 指令的媒介。可接收 ps 以 axilite 傳輸的 write data 轉換為 mprj_in[37:0]，再傳給 Caravel，也可 read Caravel mprj_out[37:0]，轉換為 axilite interface 再傳

回 ps。如果遇到 mprj_en 為 1，代表這個 pin 是 output pin，所以直接把 mprj_out 接回去給 mprj_in。

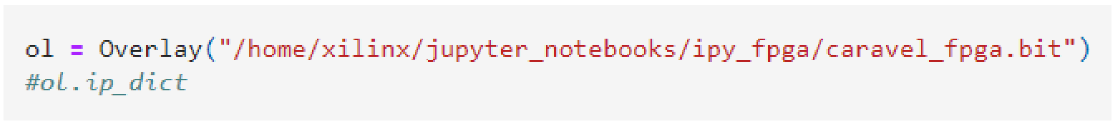
2. Verilog:

    IV. Spiflash:將已儲存的 program.hex 從 BRAM 取出後，利用內部的 shift register 將資料透過 spi interface 傳給 Caravel，由於此次實驗設計，因此只支援 read 功能。

- Caravel SOC control flow:

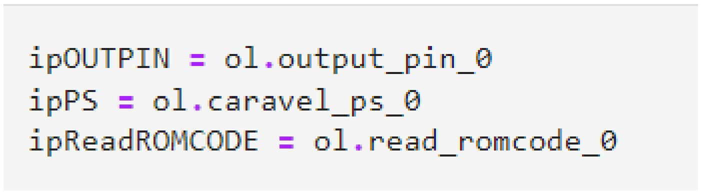
Caravel_fpga.ipynb control flow:

1. 讀取指定的 bitstream 檔



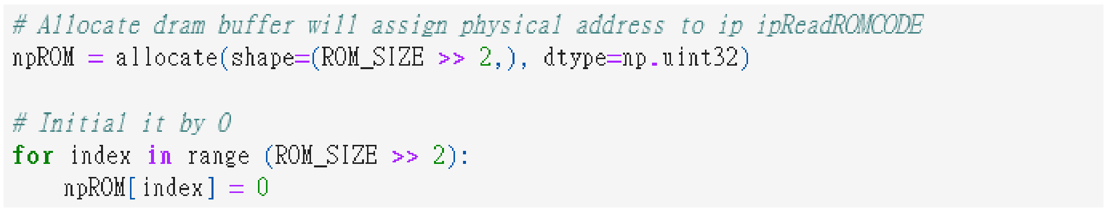
```python
ol = Overlay("/home/xilinx/jupyter_notebooks/ipy_fpga/caravel_fpga.bit")
#ol.ip_dict
```

2. 將需要 PS 端讀取或寫入的 IP 連接



```python
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
```

3. 分配一個 8K byte 的 dram buffer，並將其初始化為 0



```python
# 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
```

4. 然後將開啟的 hex 檔以 4byte assignment 的形式存入

    hex 檔中會有一些以@開頭的 offset header，需要讀取並更新 npROM_offset

```
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
```

5. 由於資料會用 axi_master interface 傳給 Read_romcode，因此也要記錄 rom 的大小

```
rom_size_final = npROM_offset + npROM_index
```

6. 將 dram 中的資料傳給 Read_romcode 後再寫入 BRAM
   (PS 會先把 npROM 的地址和 rom 的大小告訴 read_romcode，然後給出 ap_start 讓 read_romcode 開始運作。之後持續 polling 等到 ap_idle 後再 print 完成訊息)

```
# 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")
```

7. 將 Output_pin(Reset_control)設為 1，使 Caravel 開始藉由 spiflash 這個 ip 從 BRAM 中讀取指令來執行(由於 Caravel 的 reset 為 low active，因此 Output_pin

(0:reset 1:execute)。

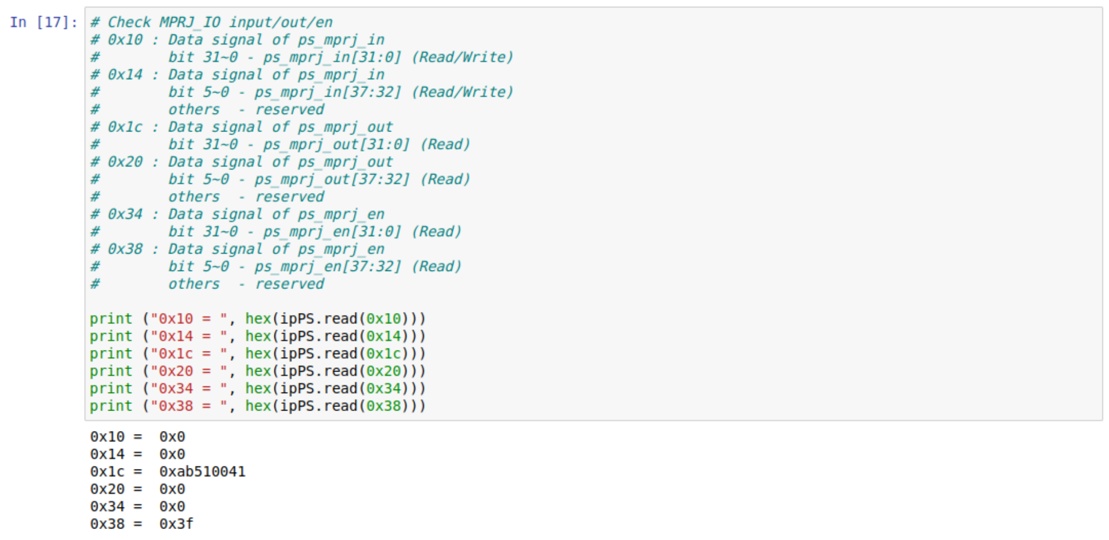
```
print (ipOUTPIN.read(0x10))
ipOUTPIN.write(0x10, 1)
print (ipOUTPIN.read(0x10))
```

8. 藉由讀取 Caravel_ps 的 mprj_out 訊號，檢查運算結果是否正確

```
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)))
```

- Screenshot of Execution result on all workload

1. Counter_la result:

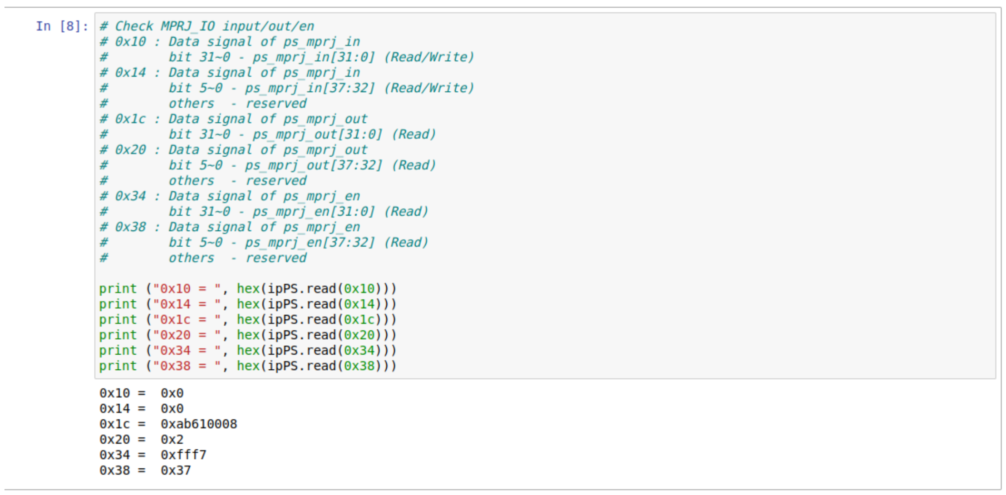


```
In [17]: # Check MPRJ_IO input/out/en
         # 0x10 : Data signal of ps_mprj_in
         #        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
         # 0x14 : Data signal of ps_mprj_in
         #        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
         #        others  - reserved
         # 0x1c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

2. Counter_wb result:

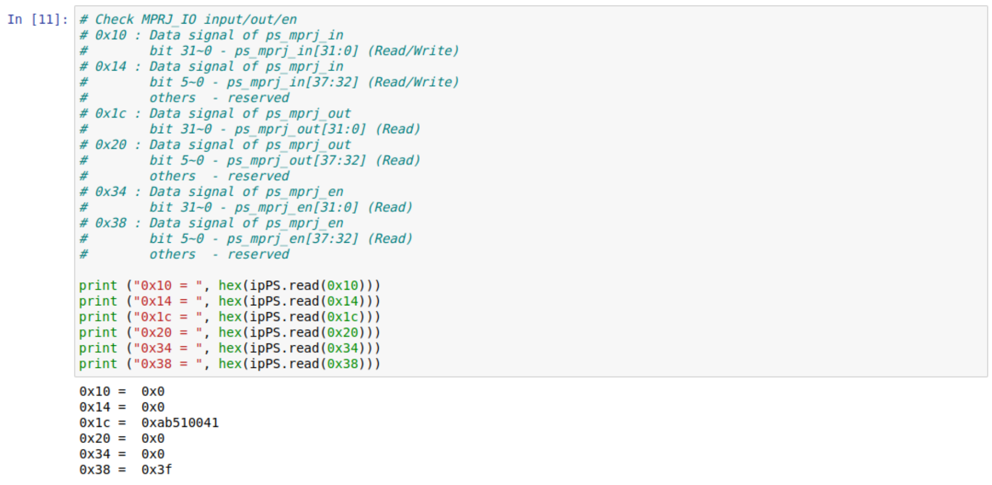


```
In [8]: # Check MPRJ_IO input/out/en
        # 0x10 : Data signal of ps_mprj_in
        #        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
        # 0x14 : Data signal of ps_mprj_in
        #        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
        #        others  - reserved
        # 0x1c : Data 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
```

3. gcd_la result:



```
In [11]: # 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
```