系統晶片設計
SOC Design

LAB5

組員:郭銘宸、郭紘碩

學號:311511082、3115120656

系所:電機所碩二、電控所碩二

一、 Block Diagram

- 實驗目的

  先前的 lab4 實驗模擬 Caravel SOC 是用軟體進行(testbench)，再這次實驗則是將整個 Caravel SOC simulation 環境放入 FPGA，使其模擬速度加快。為了完成這個模擬任務則需要設計一些 IP(老師提供)，並將其接線整合在一起，在與 Caravel SOC 一同燒入至 FPGA 進行模擬。可以看到下圖(1)是建立 block design 所使用的 TCL script，裡面有將架構圖會用的 IP 與 Caravel SOC 加入 vivado。

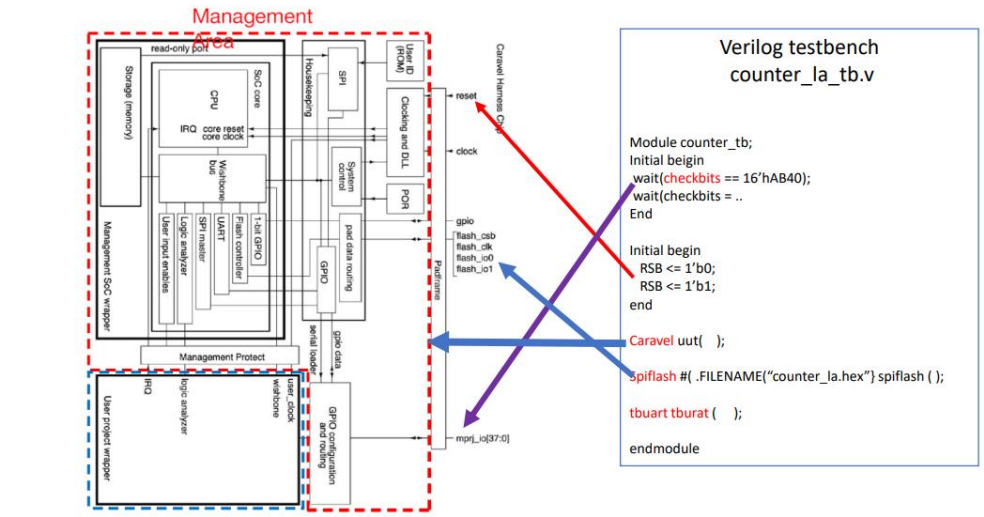
```
"[file normalize "$origin_dir/vvd_srcs/spiflash.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/vip/RAM128.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/vip/RAM256.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/VexRiscv_MinDebugCache.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/chip_io.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/gpio_control_block.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/gpio_defaults_block.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/housekeeping.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/housekeeping_spi.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/mgmt_core.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/mgmt_core_wrapper.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/mprj_io.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/user/user_proj_example.counter.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/user/user_project_wrapper.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/soc/caravel.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/header/user_defines.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/rtl/header/defines.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/vip/spiflash.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/counter_wb/counter_wb_tb.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/vip/tbuart.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/counter_la/counter_la_tb.v"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/counter_wb/counter_wb.hex"]"\
"[file normalize "$origin_dir/vvd_srcs/caravel_soc/counter_la/counter_la.hex"]"\
 ]
```

圖(1) TCL script 所加入的檔案

- 軟體模擬架構圖(a.k.a.Lab4)

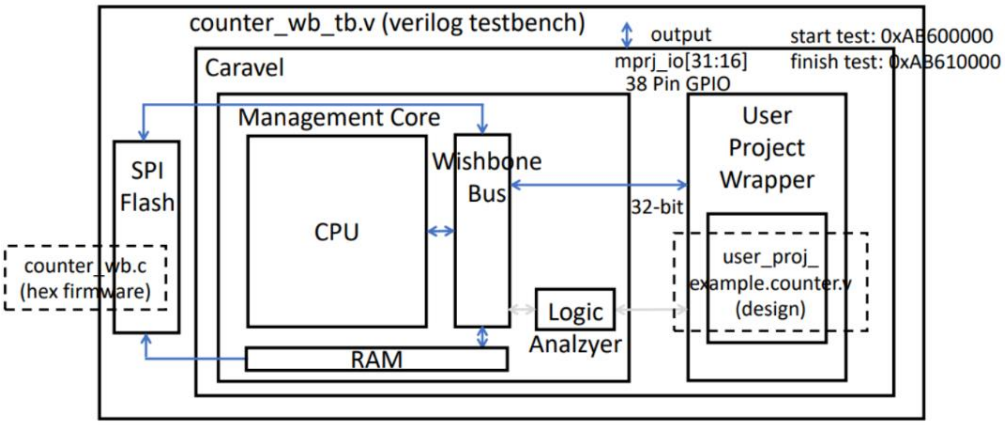
  可以透過下圖(2-1)與圖(2-2)了解，先前的是透過 TB 來模擬 Caravel SOC 的 verilog code，可以看到此時對 Caravel SOC 輸入的訊號都是軟體模擬，也就是直接在 TB 中直接對相對應的腳位賦值，若想知道 mprj 的資料也是直接讀取該腳位便可完成。

Caravel Simulation Verification System



圖(2-1) Caravel SOC 軟體驗證架構圖 1

Counter - WB



圖(2-2) Caravel SOC 軟體驗證架構圖 2

- 硬體模擬架構圖(a.k.a.Lab5)

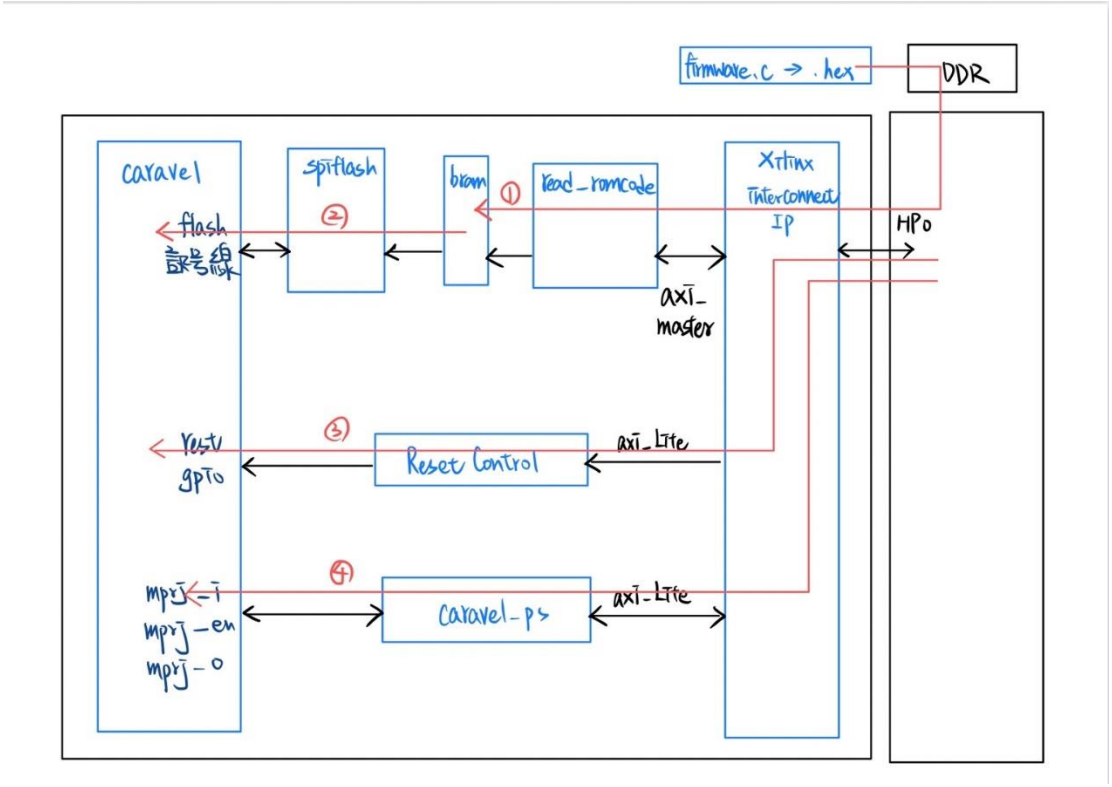
(1)若現在想將 Caravel SOC 與模擬環境整個移植到 FPGA 上，此時需要額外的 IP 來幫助完成模擬，因為在硬體上沒辦法像 TB 一樣用 Y=mprj[31:0]這樣來讀取 Caravel 的資料,也沒辦法使用 RSB<=1’b0 來對 Caravel 輸入訊號，這些工作需要仰賴特定硬體來完成。

(2)可以透過下面圖(3)架構圖來了解整個硬體模擬流程。

a. 可以透過 read_romcode IP 來完成將 firmware code 從 PS side 的 DDR 搬到 bram 中，如圖紅線 1 路徑所示。

b. 再來 Caravel SOC 可以透過 spiflash IP 來像 bram 讀取需要的 firmware 來執行，可以把 spiflash IP 當作兩這之間的橋樑，如圖紅線 2 徑所示

c. 若需要將 Caravel SOC 發送 reset 訊號，可以透過 ResetControl IP 來完成，如圖紅線 3 徑所示。
d. 若想對 Caravel SOC 的 mprj 進行讀寫，則透過 caravel_ps IP 來完成，如圖紅線 4 徑所示



圖(3) Caravel SOC 硬體驗證架構圖

二、 FPGA Utilization

Utilization Design Information

Table of Contents

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

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
282	Yes	-	Set
943	Yes	-	Reset
111	Yes	Set	-
4715	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice	2303	0	0	13300	17.32
SLICEL	1625	0			
SLICEM	678	0			
LUT as Logic	5149	0	0	53200	9.68
using O5 output only	0				
using O6 output only	4205				
using O5 and O6	944				
LUT as Memory	178	0	0	17400	1.02
LUT as Distributed RAM	18	0			
using O5 output only	0				
using O6 output only	2				
using O5 and O6	16				
LUT as Shift Register	160	0			
using O5 output only	41				
using O6 output only	81				
using O5 and O6	38				
Slice Registers	6051	0	0	106400	5.69
Register driven from within the Slice	2815				
Register driven from outside the Slice	3236				
LUT in front of the register is unused	1978				
LUT in front of the register is used	1258				
Unique Control Sets	312		0	13300	2.35

3. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	6	0	0	140	4.29
RAMB36/FIFO*	3	0	0	140	2.14
RAMB36E1 only	3				
RAMB18	6	0	0	280	2.14
RAMB18E1 only	6				

4. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	0	0	0	220	0.00

5. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	0	0	0	125	0.00
Bonded IPADs	0	0	0	2	0.00
Bonded IOPADs	130	130	0	130	100.00
PHY_CONTROL	0	0	0	4	0.00
PHASER_REF	0	0	0	4	0.00
OUT_FIFO	0	0	0	16	0.00
IN_FIFO	0	0	0	16	0.00
IDELAYCTRL	0	0	0	4	0.00
IBUFDS	0	0	0	121	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	200	0.00
ILOGIC	0	0	0	125	0.00
OLOGIC	0	0	0	125	0.00

6. Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	7	0	0	32	21.88
BUFIO	0	0	0	16	0.00
MMCME2_ADV	0	0	0	4	0.00
PLLE2_ADV	0	0	0	4	0.00
BUFMRCE	0	0	0	8	0.00
BUFHCE	0	0	0	72	0.00
BUFR	0	0	0	16	0.00

7. Specific Feature

Site Type	Used	Fixed	Prohibited	Available	Util%
BSCANE2	0	0	0	4	0.00
CAPTUREE2	0	0	0	1	0.00
DNA_PORT	0	0	0	1	0.00
EFUSE_USR	0	0	0	1	0.00
FRAME_ECCE2	0	0	0	1	0.00
ICAPE2	0	0	0	2	0.00
STARTUPE2	0	0	0	1	0.00
XADC	0	0	0	1	0.00

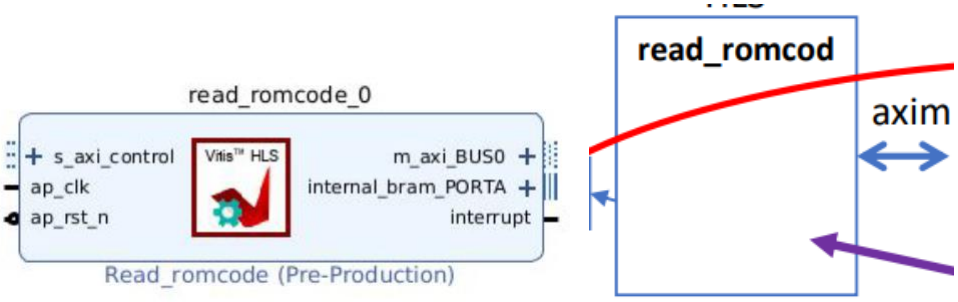
8. Primitives

Ref Name	Used	Functional Category
FDRE	4715	Flop & Latch
LUT6	2131	LUT
LUT4	1150	LUT
LUT5	1125	LUT
LUT3	957	LUT
FDCE	943	Flop & Latch
LUT2	501	LUT
FDPE	282	Flop & Latch
LUT1	229	LUT
CARRY4	216	CarryLogic
MUXF7	169	MuxFx
SRL16E	134	Distributed Memory
BIBUF	130	IO
FDSE	111	Flop & Latch
SRLC32E	64	Distributed Memory
MUXF8	47	MuxFx
RAMD32	26	Distributed Memory
RAMS32	8	Distributed Memory
BUFG	7	Clock
RAMB18E1	6	Block Memory
RAMB36E1	3	Block Memory
PS7	1	Specialized Resource

圖(4) FPGA Utilization

三、 Explain the function of IP in this design

- read_romcode(HLS)
  - (1) 可以將 firmware 從 PS side 的 DDR 送到 bram。
  - (2) 限制 bram 大小為 8K。
  - (3) 可透過下圖(5-1)知道此 IP 的 port，對 PS side 所連接的 axi_master，適合傳輸 firmware 這種大量的資料(具 burst 能力)。
  - (4) 可透過下圖(5-2)知道此 IP 具有 pineline 功能。

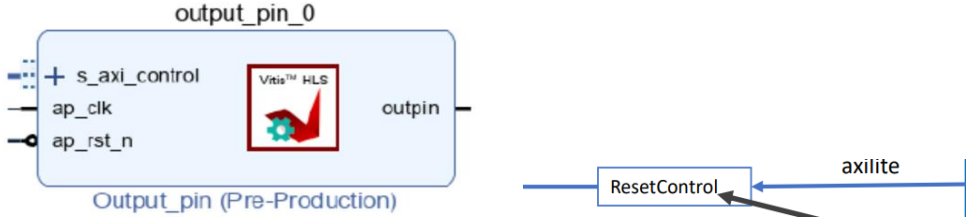


圖(5-1) read_romcode IP port & 對應架構圖位置

```
#define CODE_SIZE   2048*4

void read_romcode(
// PS side interace
    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;
}
```

圖(5-2) read_romcode IP 程式

- RestControl(HLS)因為接收資料
  (1) 會輸出 0 或 1 來 assert/de-assert Caravel SOC 的 reset pin。
  (2) 如下圖(6-1)所示，提供 AXI Lite interface 給 PS side 的 CPU 來控制此 IP。
  (3) 如下圖(6-2)所示，可以看到此 IP 將由 AXI lite interface 的 outpin_crtl 傳給 outpin。此 outpin 將會接到 Caravel SOC。



圖(6-1) RestControl IP port & 對應架構圖位置

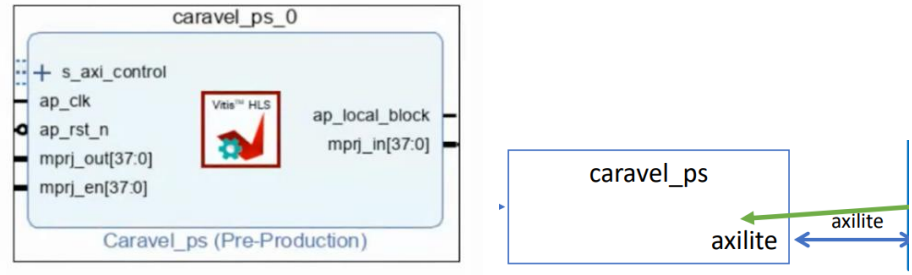
```
void output_pin(
        bool outpin_ctrl,
        bool& outpin)
{
    #pragma HLS INTERFACE s_axilite port=outpin_ctrl
    #pragma HLS INTERFACE ap_none port=outpin
    #pragma HLS INTERFACE ap_ctrl_none port=return

    outpin = outpin_ctrl;

    return;
}
```

圖(6-2) RestControl IP 程式

- caravel_ps(HLS)

(1) 如下圖(7-1)所示，caravel_ps IP 提供 AXI Lite interface 給 PS side 的 CPU 來讀取 mprj_in/out/en。

(2) 如下圖(7-2)所示，可以看到使用 unroll 將其硬體平行 38 套，也就是每一個 pin 一個硬體(mprj 共 38 pin)。

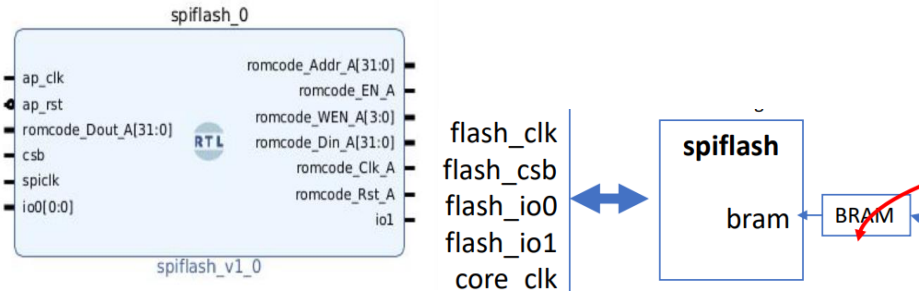


圖(7-1) caravel_ps IP port & 對應架構圖位置

```
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];
    }


}
```

圖(7-2) caravel_ps IP 程式

- Spiflash(Verilog)
  (1) 實現 SPI slave 功能，只提供 read command。
  (2) 會從 bram return data 到 Caravel SOC。
  (3) 可以透過圖(8)知道，此 IP 會連接到 Caravel SOC 的 spiflash 相關的 port 。



圖(8) Spiflash IP port & 對應架構圖位置

四、Run workloads on caravel FPGA & Screenshot of execution result

- Counter_wb.hex

```
# 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+")
```

```
# 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
```

```
# 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
```

- Counter_la.hex

```
# 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+")
```

```
# 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
```

```
# 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 =  0xab51a3ee
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

- gcd_la. hex

```
# 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+")
```

```
# 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 =  0xffffffff
0x38 =  0x3f
```

```
# 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 =  0xab5145d3
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

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

```
In [10]: 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
```

- Import 各個會用到的 library，並且將 ROM_SIZE 定義為 8k

```
In [11]: ol = Overlay("/home/xilinx/jupyter_notebooks/caravel_fpga.bit")
         #ol.ip_dict
```

- 將由三個 IP 產生的 bitstream 檔放到 fpga 上運行

```
In [12]: ipOUTPIN = ol.output_pin_0
         ipPS = ol.caravel_ps_0
         ipReadROMCODE = ol.read_romcode_0
```

- 把 Jupyter notebook 中對應到各 ip 的連接線接上，ipOUTPIN 對應到 ResetControl，ipPS 對應到 caravel_ps，ipReadROMCODE 對應到 read_romcode

In [13]:
```
# 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))
```

- 首先 npROM allocate 出 ROM_SIZE/4 的空間，接著把要寫進 bram 的 firmware code 命名為 fiROM，再來將.hex 檔中的資料讀到先前 allocate 好的 npROM 陣列中，其中包含了 npROM_offset 及 npROM_index 和 data buffer 等資訊

In [14]:
```
# 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")
```

- 先設定好 base address 以及 data length 後便開始傳輸資料，最後等到 bram 資料搬移結束即 print 出 Write to bram done

In [15]:
```
# 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)))
```

- 透過先前連接到 caravel_ps 的 ipPS 讀取 register map 中對應到的值，就能夠透過觀察其中的值知道 mprj 的輸入輸出狀態

```
In [16]: # 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))
```

- 在 release reset 後，caravel SoC 便能夠開始執行 spiflash 由 bram 拿取到的 firmware code 的內容

```
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)))
```

- 透過觀察 mprj_io 的輸入輸出狀態，能夠看到由 firmware code 控制的部分讀取出來的值為結束時所命的值，在 counter_wb 中為 0xAB610000，而在 counter_la 中為 0xAB510000，其中後面的 0000 部分可能會隨執行時間增加而增加，因為[16:0]並非為 GPIO_MODE_MGMT_STD_OUTPUT，而是 GPIO_MODE_USER_STD_OUTPUT，所以無法被使用者控制