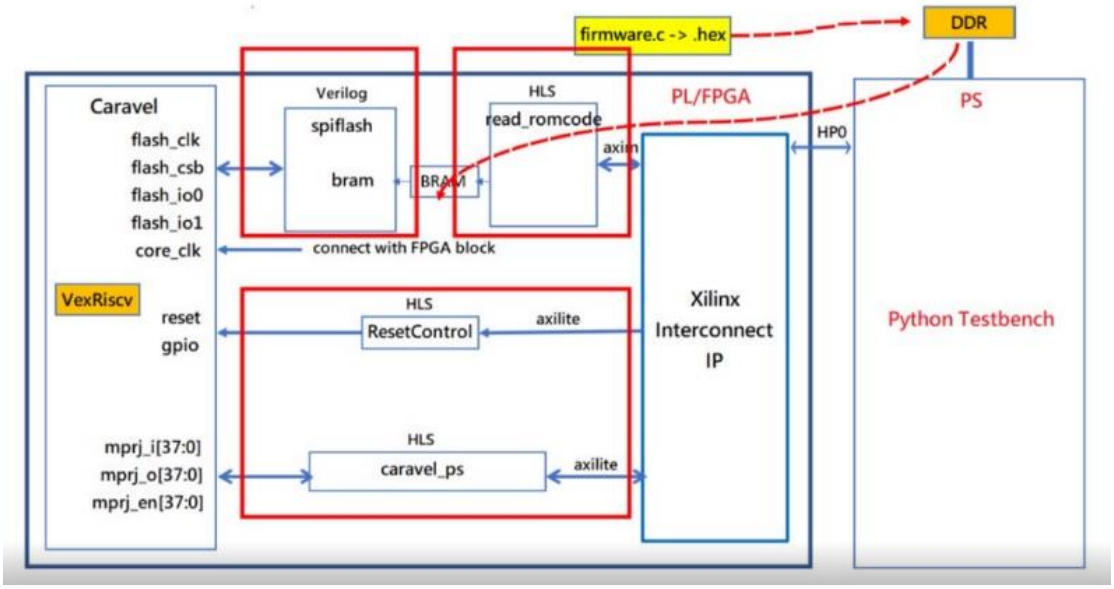
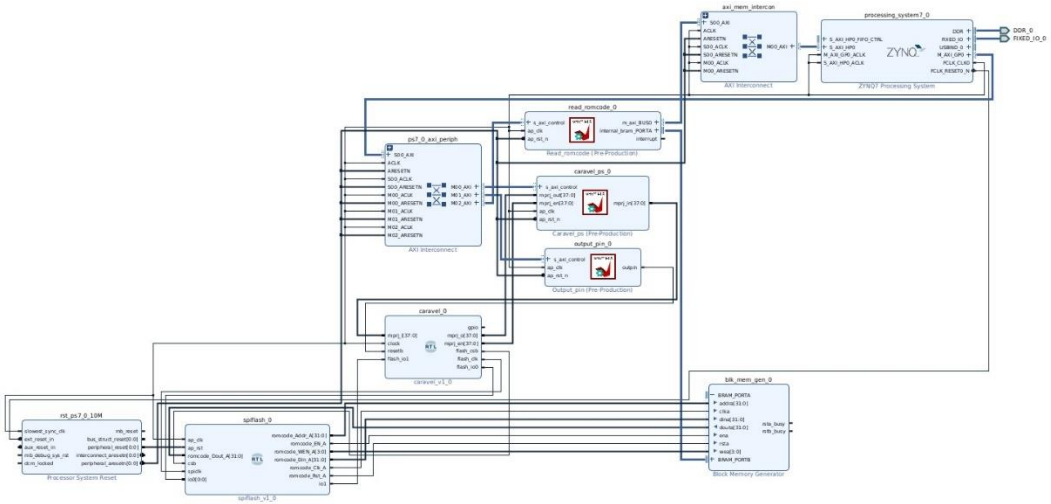
SOC Lab 5 Caravel Soc FPGA Integration

組員：王證皓、丁緒翰、潘金生

1. Block diagram
   A. Object: 把 Caravel SOC simulation 環境放到 FPGA 上
   B. Step:
      1. Load firmware code 檔案參數給 PS side DDR，要 load 進硬體裡面透過 AXI-master protocol 藉由 read_romcode 硬體放到 BRAM 裡
      2. Load 完後，ResetControlu 硬體把 Reset 放掉
      3. 透過 Caravel_ps 傳遞 mprj 反映在 axilite





圖一 Block diagram

2. FPGA utilization
   a. Caravel utilization

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	4967	0	0	53200	9.34
LUT as Logic	4913	0	0	53200	9.23
LUT as Memory	54	0	0	17400	0.31
LUT as Distributed RAM	16	0			
LUT as Shift Register	38	0			
Slice Registers	3976	0	0	106400	3.74
Register as Flip Flop	3901	0	0	106400	3.67
Register as Latch	75	0	0	106400	0.07
F7 Muxes	168	0	0	26600	0.63
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				

圖二 Caravel utilization

b. Read_romcode utilization

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	739	0	0	53200	1.39
LUT as Logic	664	0	0	53200	1.25
LUT as Memory	75	0	0	17400	0.43
LUT as Distributed RAM	0	0			
LUT as Shift Register	75	0			
Slice Registers	1100	0	0	106400	1.03
Register as Flip Flop	1100	0	0	106400	1.03
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

圖三 Read_romcode utilization

c. Caravel_ps utilization

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

圖四 Caravel_ps utilization

d. Output_pin utilization

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

圖五 output pin utilization

3. Explain the function of IP in this design

HLS：read_romcode, ResetControl, caravel_ps

Read_romcode:

A. IP 從 DRAM 讀取 ROM 代碼
   - PS 設定 m_axi_BUS0 基底位址
   - PS 發送讀取命令
   - IP 開始讀取 DRAM ROM 程式碼
   - PS 等待 IP 完成

B. IP 將 ROM 代碼寫入 DRAM
   - PS 設定 m_axi_BUS1 基底位址
   - PS 發送寫入命令
   - IP 開始將 ROM 代碼寫入 DRAM
   - PS 等待 IP 完成

```
/*
 * 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 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;
}
```

圖六 read_romcode

ResetControl:

- 輸出 1 或 0 訊號，用於 assert/ de-assert Caravel reset pin
- 提供 AXI LITE 介面供 PS CPU 控制輸出
- 透過 HLS 實作並匯出 IP 以供 Vivado 專案使用

```
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;
}
```

圖七 ResetControl

Caravel_ps:

- 提供 PS CPU AXI Lite 介面以讀取 MPRJ_IO/OUT/EN bits
- 透過 HLS 實作並匯出 IP 以供 Vivado 專案使用。

```
#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];
        }


}
```

圖八 caravel_ps

Verilog ：spiflash

- 實作 SPI slave device，僅支援讀取指令（0x03）
- 將資料從 BRAM 返回 Caravel

```
// io1 output
    // assign io1 = buffer[7];
    assign io1 = outbuf[7];

// BRAM Interface
   assign romcode_Addr_A = {8'b0, spi_addr};
   assign romcode_Din_A = 32'b0;
   assign romcode_EN_A = (bytecount >= 4);
   assign romcode_WEN_A = 4'b0;
   assign romcode_Clk_A = ap_clk;
   assign romcode_Rst_A = ap_rst;

       // 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] ;

       task spi_action;
               begin

                       if (bytecount == 0) begin
                               spi_cmd <= buffer;
                       end

                       if (spi_cmd == 'h 03) begin     // only support READ 03
                               if (bytecount == 2)
                                       spi_addr[23:16] <= buffer;

                               if (bytecount == 3)
                                       spi_addr[15:8] <= buffer;

                               if (bytecount == 4)
                                       spi_addr[7:0] <= buffer;

                               if (bytecount >= 4) begin
                                       buffer <= memory;
                                       spi_addr <= spi_addr + 1;
                               end
                       end
               end
       endtask
```

```
  // 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

  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
          end
              end
      end
```

圖九 spiflash

4. Run these workload on caravel FPGA
   A. Select one firmware binary to load into BRAM as SPIROM data
      - fiROM = open("counter_wb.hex", "r+")
      - fiROM = open("counter_la.hex", "r+")
      - fiROM = open(“gcd_la.hex", "r+")

B. Release reset to make Caravel-soc start execute firmware code

• ipOUTPIN.write(0x10, 1)

Note: If you want to load different firmware binary without restart FPGA, you need to assert & release reset signal for the caravel-soc to make CPU execute new firmware.

1. Load counter_wb.hex
2. Release reset by ipOUTPIN.write(0x10, 1), check mprj_i/o/en
3. Load counter_la.hex
4. Assert reset by ipOUTPIN.write(0x10, 0)
5. Release reset by ipOUTPIN.write(0x10, 1), check mprj_i/o/en

```
# 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()
```

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

5. Screenshot of Execution result on all workload

• counter_wb.hex

如附圖 C code，可以看到結束時(counter = 0x2B3D)，0x1c 在 61 的位置。

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))
```

```
0
1
```

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
```

```
 /* Apply configuration */
reg_mprj_xfer = 1;
while (reg_mprj_xfer == 1);

reg_la2_oenb = reg_la2_iena = 0x00000000;    // [95:64]

// Flag start of the test
reg_mprj_datal = 0xAB600000;

reg_mprj_slave = 0x00002710;
reg_mprj_datal = 0xAB610000;
if (reg_mprj_slave == 0x2B3D) {
    reg_mprj_datal = 0xAB610000;
}
```

• counter_la.hex

如附圖 C code，當 counter 數到 0x1F4 時，位置為 41。

```
while (1) {
        if (reg_la0_data_in > 0x1F4) {
                reg_mprj_datal = 0xAB410000;
                break;
        }
}
```

```
Write to bram done
1
0
1
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab41fa3f
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

• gcd_la.hex

如附圖 C code，當 checkbits 等於 16’hAB40 時，gcd 完成。

```
initial begin
        wait(checkbits == 16'hAB40);
        $display("LA Test seq_gcd(10312050, 29460792)=138 started");
```

```
Write to bram done
1
0
1
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab40f267
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

6. Study caravel_fpga.ipynb, and be familiar with caravel SoC control flow
   a. 把 hex 檔讀進 ROM 裡

```
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)
```

   b. 把 ROM code 放進 bram

```
# 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")
```

c. 讀 mprj

```
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. Reset

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

e. 讀 mprj

```
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)))
```

f. 重複執行上述步驟。