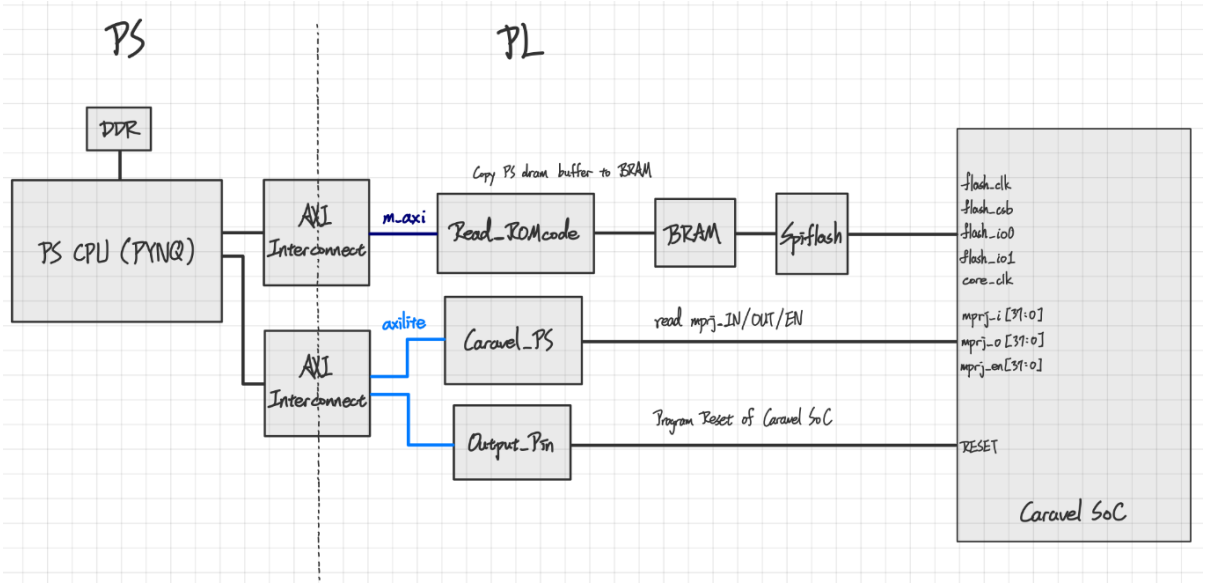
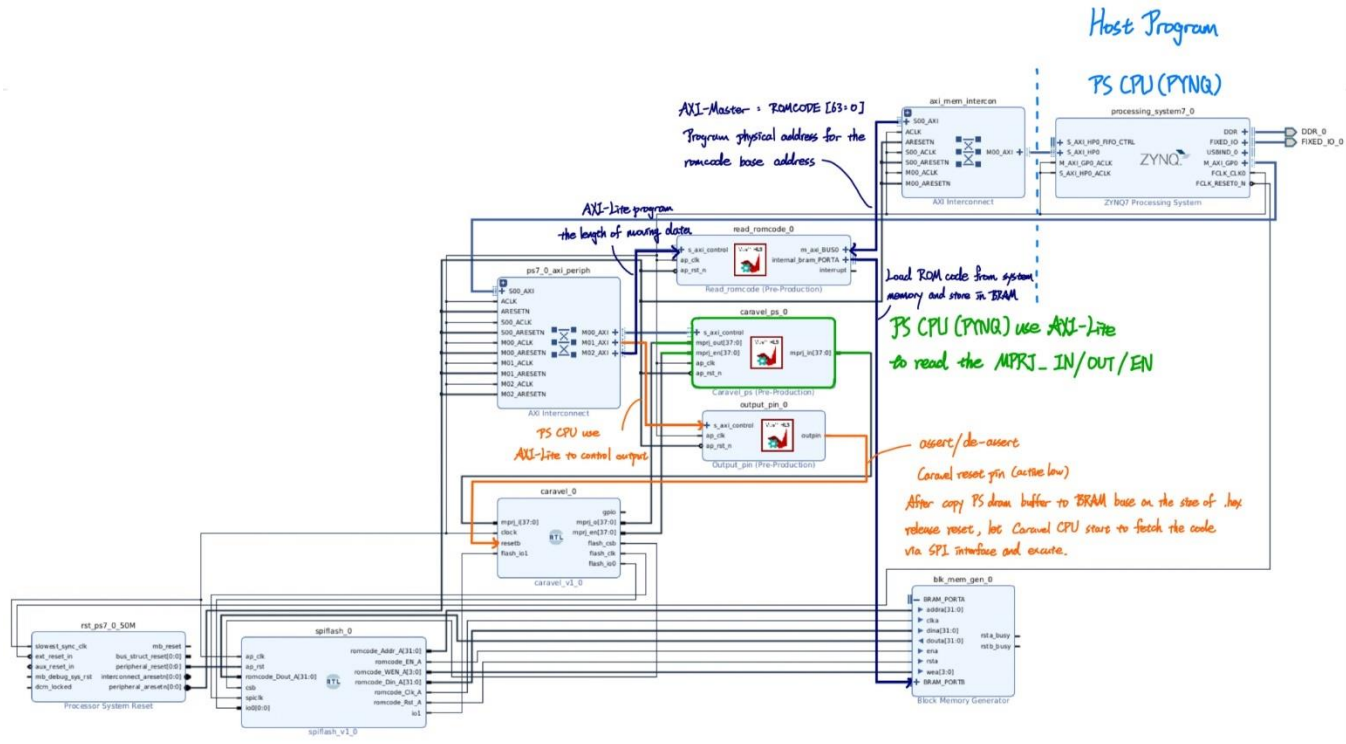
SOC Lab5 Report

110590022 陳冠晰 110000107 陳柏翰 110061217 王彥智

1. Block diagram



PS PL interconnect



Detailed Datapath with IP and interconnect (important wire are marked and explained)

2. The function of IP in this design

a. HLS

i. Read_romcode

```
#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;
}
```

Pragma HLS INTERFACE:

Use AXI-Lite as port-level protocol.

Provide AXI-Master for port romcode to read the binary file from host-side.

Specific port internal_bram as BRAM, which means this port will connect to BRAM

Provide AXI-Lite for PS CPU to load the length of moving data.

Function:

After checking the length of code size, load romcode from PS side to BRAM in PL side.

ii. Output_pin (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;
}
```

Pragma HLS INTERFACE:

Ap_ctrl_none for block level protocol.

Provide AXI-Lite interface for PS CPU to control the signal.

Pure wire ap_none for outpin, which will connect to Caravel reset pin.

Function:

Very simple, just transfer control signal to outpin by using AXI-Lite.

iii. Caravel_ps

```
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];
        }


}
```

Pragma HLS INTERFACE:

Provide AXI-Lite interface for PS CPU to read mprj_in/out/en.

Ap_ctrl_none for block-level protocol.

Pure wires mprj_in/out/en which are connected to Caravel.

Function:

PS CPU can directly read the mprj_out/en by using AXI-Lite.

If mprj_en from Caravel is 0, Caravel mprj_in will accept the value of PS CPU.

b. Verilog: spiflash

(https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/labi/vvd_srcs/spiflash.v)

```
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] ;
```

Load the data from romcode BRAM.

```
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
```

Count the bit and byte, bytecount + 1 when bitcount count to 7.

Get the SPI command and address.

```
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
```

First, check the command, if READ 03, then load the address.

Second, after finish addressing (bytecount >= 4),

transfer the data from SPI memory to buffer. (Return data from BRAM to Caravel.)

```
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
```

Use another shift buffer for output, notice that it uses the negative edge of spiclk.

3. FPGA utilization

a. ReadROMcode

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

Since we use **HLS INTERFACE bram** to decide the port type of internal_bram, it generates a memory.

b. OutputPin

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

Only some simple logic.

c. Caravel_PS

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

Provide AXI-Lite interface for PS side to read mprj_IN/OUT/EN from Caravel.

d. Spiflash

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

Using registers for shifter registers, in order to load the data.

e. CaravelSoC

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	3842	0	0	53200	7.22
LUT as Logic	3788	0	0	53200	7.12
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

f. Block RAM

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

4. **Run these workloads on caravel FPGA**
   Done and the results are showed below.

5. **Screeshot of Execution result**
   a. counter_wb

```
Write to bram done
0x10 =  0x0
0x14 =  0x0
0x1c =  0x8
0x20 =  0x0
0x34 =  0xffffffff7
0x38 =  0x3f
0
1
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab610008
0x20 =  0x2
0x34 =  0xfff7
0x38 =  0x37
```

   b. counter_la

```
Write to bram done
0x10 =  0x0
0x14 =  0x0
0x1c =  0x8
0x20 =  0x0
0x34 =  0xffffffff7
0x38 =  0x3f
0
1
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab514ff2
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

c. gcd_la

```
Write to bram done
0x10 =  0x0
0x14 =  0x0
0x1c =  0x8
0x20 =  0x0
0x34 =  0xfffffff7
0x38 =  0x3f
0
1
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab40afe0
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

6. caravel_fpga.ipynb

a. Initial ROM_SIZE (Max 8k)
b. ol = Overlay .bit file
c. Allocate ReadROMCODE to dram buffer and initialize by 0.
d. Open the firmware code (.hex)
e. Calculate the np_ROM_offset and npROM_index to determine the length of the code. (32bit alignment)

f. Start to write the firmware code in to bram.

```
# 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")
```

g. Check the MPRJ_IO pin before the execution.

```
# 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)))
```

h. Print the ipOUTPIN.read(0x10).

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))
```

```
0
1
```

0x10 is for reset signal control. Reset is negative active. Therefore, it's 0 before the execution starts.

For the caravel SoC to start executing, we write the reset to 1 and read if we successfully write the value into 0x10.

i. The last part is to see the result of the execution.

```
# 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
```

we just print them out to check!

```
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)))
```