NTHU 112 Fall Semester

系統晶片設計

SOC Design Laboratory

Lab5


國立清華大學

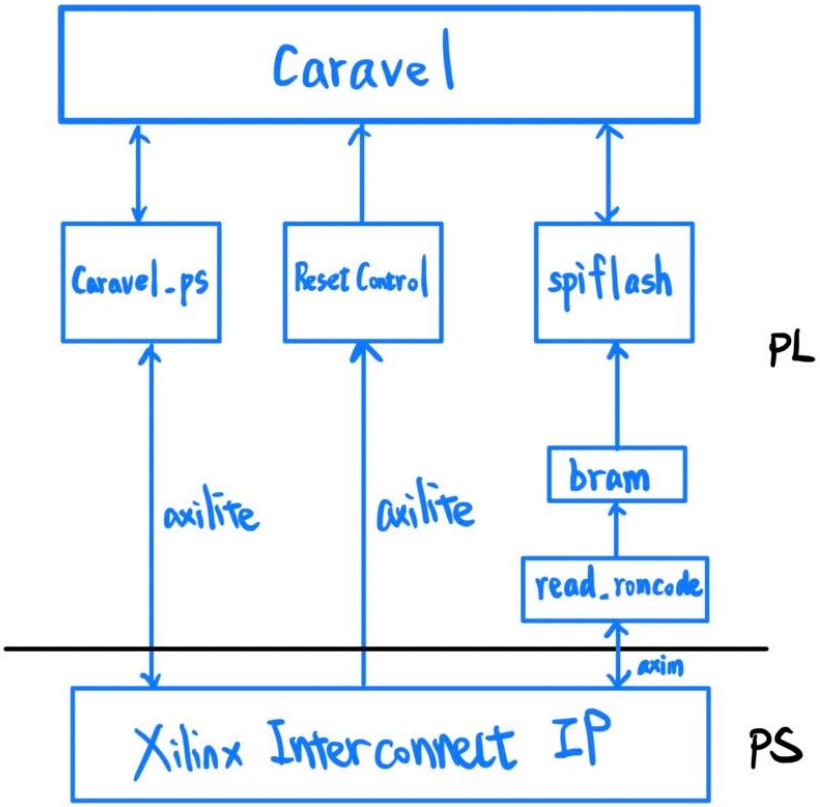
NATIONAL TSING HUA UNIVERSITY

組別: 第 18 組

學號: 111061647, 111064537, 112061576

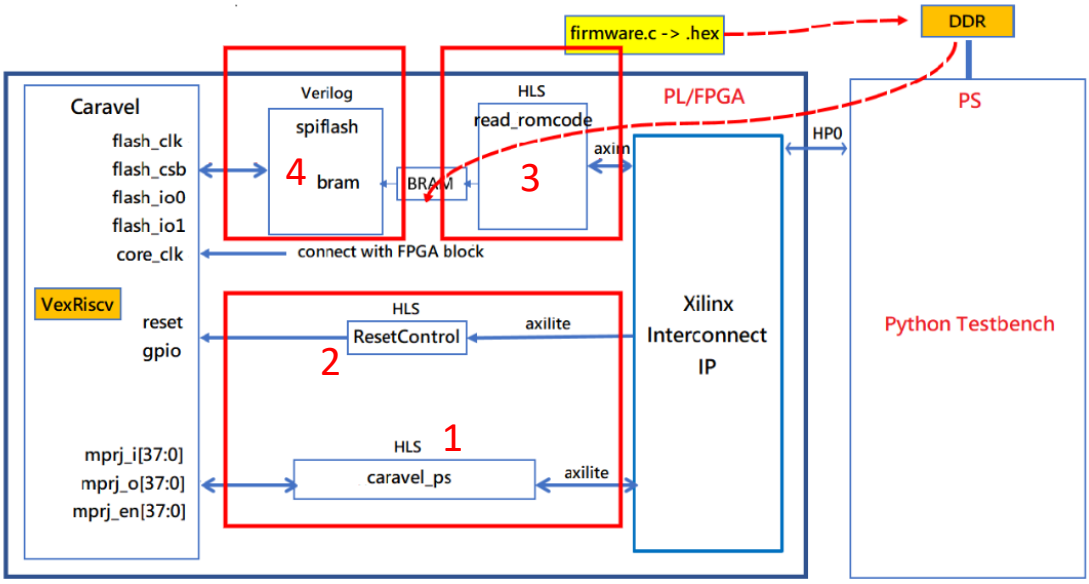
姓名: 盧人豪，林亭君，莊家政

**Block diagram:**




firmware code 一開始是存在 PS 這裡，剛開始 read_romcode 會將 firmware code 讀出放置在 bram。此時 ResetControl 傳送訊號給 Caravel，Caravel 就開始跑，跑的流程是 Caravel 會透過 spiflash 讀取 bram 裡的 firmware code。跑完後的值會丟到 mprj 的 pin 上，透過 carvel_ps，把這些值藉由 AXI-Lite 傳到 PS 那，才能讀取。

**Function:**




1. Caravel_ps:

使用 axilite interface 讓 PS CPU 能夠讀取 mprj_in/mprj_out/mprj_en bits

```
#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];
    }


}
```

將 Caravel (PL side)得到的值根據各自的 mprj_en 來決定要讓 mprj_in 接收自己的 output 或者要從 PS side 拿取數值。從 code 也可以看到，能透過此 ip 讓 PS side

讀取 PL side 之值(i.e. PL side 將值傳給 PS side)。並且從 pragma 的地方可以看到，interface 確實指定使用 axilite 進行傳輸。

## 2. Caravel GPIO (Reset Control):

### Output_pin

Program 0x10 這個位址，就可以對 caravel 做 reset。

```
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;
}
```

使用 AXI-Lite 將 outpin_ctrl 傳輸到 outpin。

## 3. Read_romcode:

Host 會從 main memory DDR 透過此 ip load program code。

```
#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;
}
```

將在 PS side 的 firmware code 從 DDR 那讀取出來，再放到 bram 裡面，而 bram size 限制為 8K。if 是判斷讀取的資料有幾筆，不能超過既有的數量。for 是將讀取出來的資料傳至 bram。

## 4. Spiflash:

Spiflash 可以 configurable 的 mmio 很少，能用的是 read command (0x30) 這個位置，它會從 BRAM 把 program 搬到 caravel 裡面。

```
    // 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] ;
```

將 readromcode 裡的值讀取出來放到 memory。

```
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
```

```
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
```

SPI 的 READ（讀取）命令為’h03 時，會根據不同的 bytecount 數值將 buffer 傳給 spi_addr 的各個 byte，當 bytecount>=4 時，memory 的資料給 buffer 且 spi_addr 加一。

```
// 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
```

使用 outbuf 緩存輸出，outbuf 在 spiclk 負緣觸發時更新。將 outbuf 每個位元左移一位，並將最低有效位設置為零，如果 bitcount 等於 0 且 bytecount 大於等於 4，outbuf 也會被更新為來自 memory 的數據。

**FPGA Utilization:**

Overall design (design_1_wrapper_utilization_synth.rpt) include:

| Ref Name | Used |
|---|---|
| design_1_xbar_0 | 1 |
| design_1_spiflash_0_0 | 1 |
| design_1_rst_ps7_0_50M_0 | 1 |
| design_1_read_romcode_0_0 | 1 |
| design_1_processing_system7_0_0 | 1 |
| design_1_output_pin_0_0 | 1 |
| design_1_caravel_ps_0_0 | 1 |
| design_1_caravel_0_0 | 1 |
| design_1_blk_mem_gen_0_0 | 1 |
| design_1_auto_us_0 | 1 |
| design_1_auto_pc_1 | 1 |
| design_1_auto_pc_0 | 1 |

Read ROM (design_1_read_romcode_0_0)

| 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 |

7. Primitives

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 1097 | Flop & Latch |
| LUT3 | 261 | LUT |
| LUT6 | 212 | LUT |
| LUT4 | 158 | LUT |
| LUT2 | 132 | LUT |
| SRL16E | 75 | Distributed Memory |
| LUT5 | 68 | LUT |
| CARRY4 | 63 | CarryLogic |
| LUT1 | 22 | LUT |
| FDSE | 3 | Flop & Latch |
| RAMB36E1 | 1 | Block Memory |

Output Pin (design_1_output_pin_0_0)

| 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 |

7. Primitives

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 12 | Flop & Latch |
| LUT5 | 4 | LUT |
| LUT4 | 4 | LUT |
| LUT6 | 1 | LUT |
| LUT2 | 1 | LUT |
| LUT1 | 1 | LUT |

Caravel PS (design_1_caravel_ps_0_0)

| 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 |

7. Primitives

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 158 | Flop & Latch |
| LUT3 | 79 | LUT |
| LUT6 | 46 | LUT |
| LUT2 | 8 | LUT |
| LUT4 | 4 | LUT |
| LUT5 | 1 | LUT |
| LUT1 | 1 | LUT |

SPI Flash (design_1_spiflash_0_0)

| 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 |

7. Primitives

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 32 | Flop & Latch |
| FDCE | 31 | Flop & Latch |
| LUT3 | 26 | LUT |
| LUT6 | 21 | LUT |
| CARRY4 | 10 | CarryLogic |
| LUT4 | 5 | LUT |
| LUT5 | 4 | LUT |
| LUT1 | 2 | LUT |
| LUT2 | 1 | LUT |

Caravel (design_1_caravel_0_0)

| 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 |

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 | | | | |

7. Primitives
---

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 2623 | Flop & Latch |
| LUT6 | 1753 | LUT |
| FDCE | 889 | Flop & Latch |
| LUT5 | 876 | LUT |
| LUT4 | 814 | LUT |
| LUT3 | 291 | LUT |
| FDPE | 271 | Flop & Latch |
| LUT2 | 262 | LUT |
| LUT1 | 184 | LUT |
| MUXF7 | 169 | MuxFx |
| CARRY4 | 134 | CarryLogic |
| FDSE | 87 | Flop & Latch |
| LDCE | 75 | Flop & Latch |
| MUXF8 | 47 | MuxFx |
| SRL16E | 38 | Distributed Memory |
| RAMD32 | 24 | Distributed Memory |
| RAMS32 | 8 | Distributed Memory |
| RAMB18E1 | 6 | Block Memory |

BRAM (design_1_blk_mem_gen_0_0)

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 |

| 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 |

7. Primitives
---

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 12 | Flop & Latch |
| LUT2 | 6 | LUT |
| SRL16E | 2 | Distributed Memory |
| RAMB36E1 | 2 | Block Memory |
| LUT4 | 2 | LUT |

Reset (design_1_rst_ps7_0_50M_0)

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|---|---|---|---|---|---|
| Slice LUTs* | 19 | 0 | 0 | 53200 | 0.04 |
| LUT as Logic | 18 | 0 | 0 | 53200 | 0.03 |
| LUT as Memory | 1 | 0 | 0 | 17400 | <0.01 |
| LUT as Distributed RAM | 0 | 0 | | | |
| LUT as Shift Register | 1 | 0 | | | |
| Slice Registers | 40 | 0 | 0 | 106400 | 0.04 |
| Register as Flip Flop | 40 | 0 | 0 | 106400 | 0.04 |
| 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 |

7. Primitives

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 36 | Flop & Latch |
| LUT2 | 9 | LUT |
| LUT4 | 6 | LUT |
| LUT1 | 5 | LUT |
| FDSE | 4 | Flop & Latch |
| LUT5 | 3 | LUT |
| SRL16E | 1 | Distributed Memory |
| LUT6 | 1 | LUT |
| LUT3 | 1 | LUT |

PS (design_1_processing_system7_0_0)

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|---|---|---|---|---|---|
| Slice LUTs* | 24 | 0 | 0 | 53200 | 0.05 |
| LUT as Logic | 24 | 0 | 0 | 53200 | 0.05 |
| LUT as Memory | 0 | 0 | 0 | 17400 | 0.00 |
| Slice Registers | 0 | 0 | 0 | 106400 | 0.00 |
| Register as Flip Flop | 0 | 0 | 0 | 106400 | 0.00 |
| 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 |

4. 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 | 0 | 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 |

5. Clocking
---

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|---|---|---|---|---|---|
| BUFGCTRL | 1 | 0 | 0 | 32 | 3.13 |
| 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. Primitives
---

| Ref Name | Used | Functional Category |
|---|---|---|
| BIBUF | 130 | IO |
| LUT1 | 24 | LUT |
| PS7 | 1 | Specialized Resource |
| BUFG | 1 | Clock |

Run these workloads on caravel FPGA:

counter_wb.hex:

```
# 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
```

```
# 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 =  0xab610008
0x20 =  0x2
0x34 =  0xfff7
0x38 =  0x37
```

```
// 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:

```
# 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 =  0xab51a261
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

```
// Flag start of the test
reg_mprj_datal = 0xAB400000;

// Set Counter value to zero through LA probes [63:32]
reg_la1_data = 0x00000000;

// Configure LA probes from [63:32] as inputs to disable counter write
reg_la1_oenb = reg_la1_iena = 0x00000000;

while (1) {
        if (reg_la0_data_in > 0x1F4) {
                reg_mprj_datal = 0xAB410000;
                break;
        }
}
//print("\n");
//print("Monitor: Test 1 Passed\n\n");  // Makes simulation very long!
reg_mprj_datal = 0xAB510000;
```

gcd_la.hex:

```
# 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 =  0xab40aa5e
0x20 =  0x0
0x34 =  0x0
0x38 =  0x3f
```

**Study caravel_fpga.ipynb , and be familiar with caravel SoC control flow:**

a.初始化 ROM_SIZE 為 8K
b.將 hex 檔和其對應的資料放在 fiROM 和 npROM
c.將 firmware code 寫到 bram
d.執行前先檢查 mprj pin
e. Caravel 執行從 bram 那讀到的 firmware code
f.將輸出結果放在 mprj pin 以便被讀取出來驗證