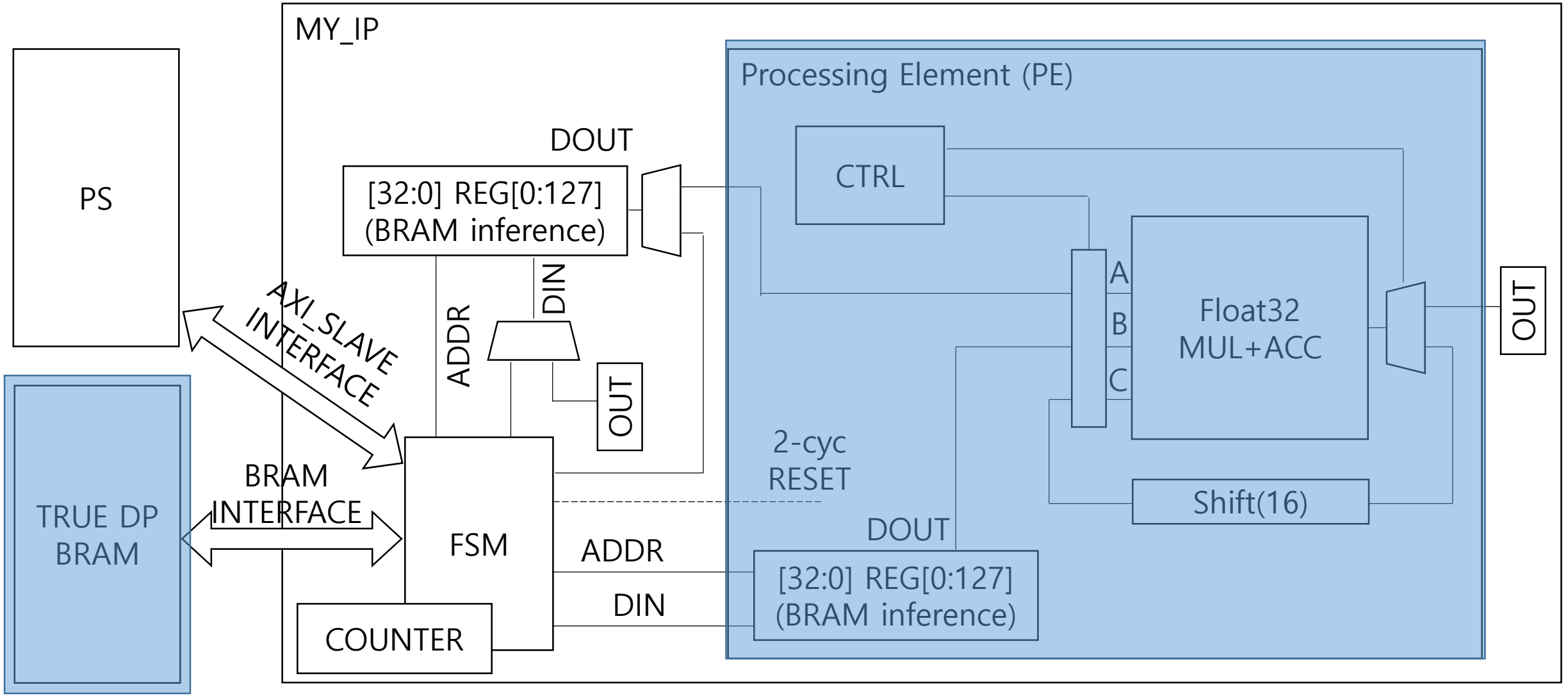
Practice 5

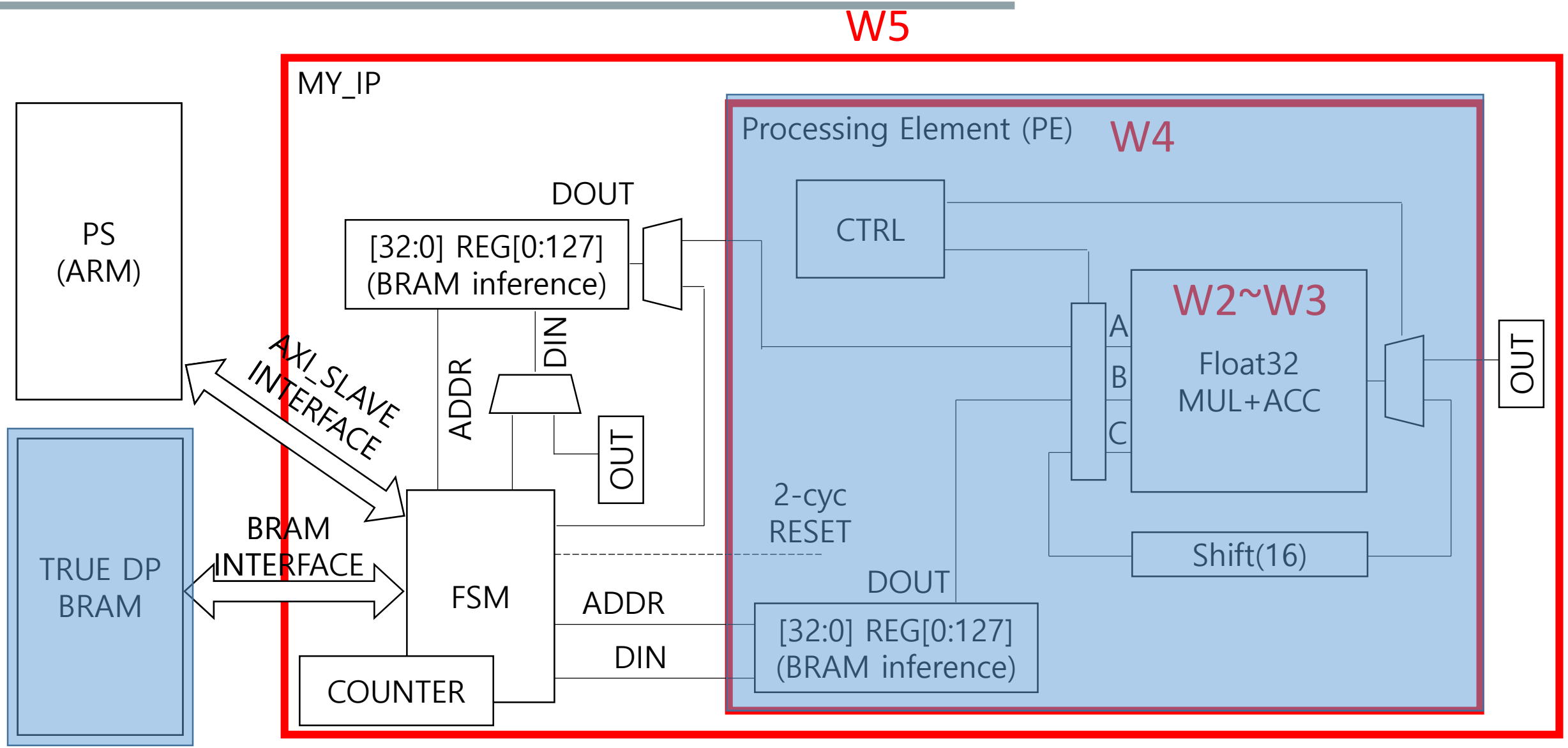
- **PE implementation & BRAM modeling**

Computing Memory Architecture Lab.

Final Project Overview: Matrix Multiplication IP



Final Project Overview: Matrix Multiplication IP



Final Project Overview: Matrix Multiplication IP

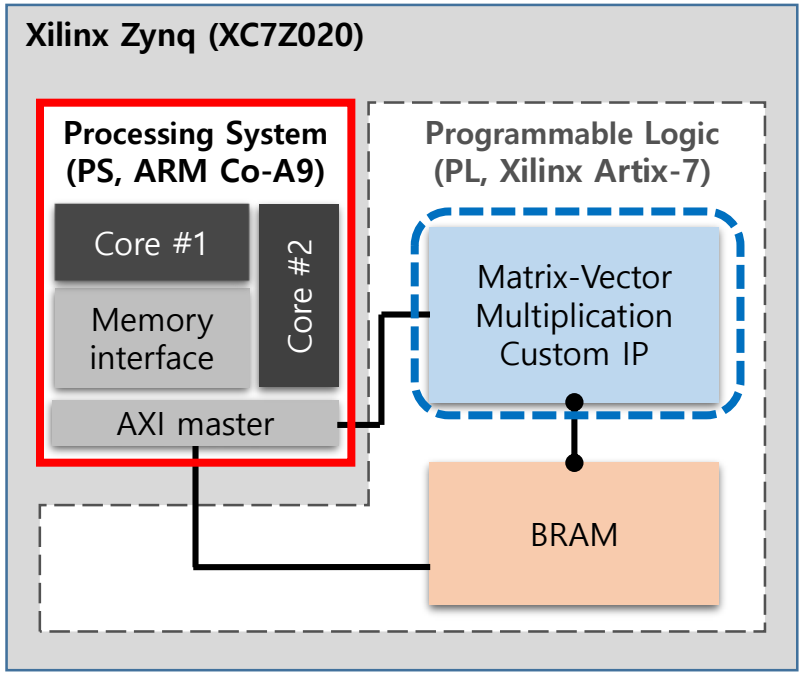
- MLP is our application
  - Each layer is matrix-vector multiplication, e.g., 256x1024 matrix * 1024-d vector ➔ 256-d vector
- ARM CPU runs the main function which calls your MV IP on PL
  - MV for 64x64 weight matrix * 64-entry input vector multiplication
- BRAM is used for data transfer between SW and HW

One layer in MLP (Software running on CPU)

```
for(j=0; j<1024; j+=8) {
   for(i=0; i<256; i+=8) {
      Output[i] += Input[j]*W[i,j] + Input[j+1]*W[i,j+1] + ...
      Output[i+1] += Input[j]*W[i+1,j] + Input[j+1]*W[i+1,j+1] + ...
      ...
      Output[i+7] += Input[j]*W[i+7,j] + Input[j+1]*W[i+7,j+1] + ...
   }
}
```

MV function on Hardware

MAC (Multiply-Accumulate)



Main Practice

Practice

1. Implementing BRAM model

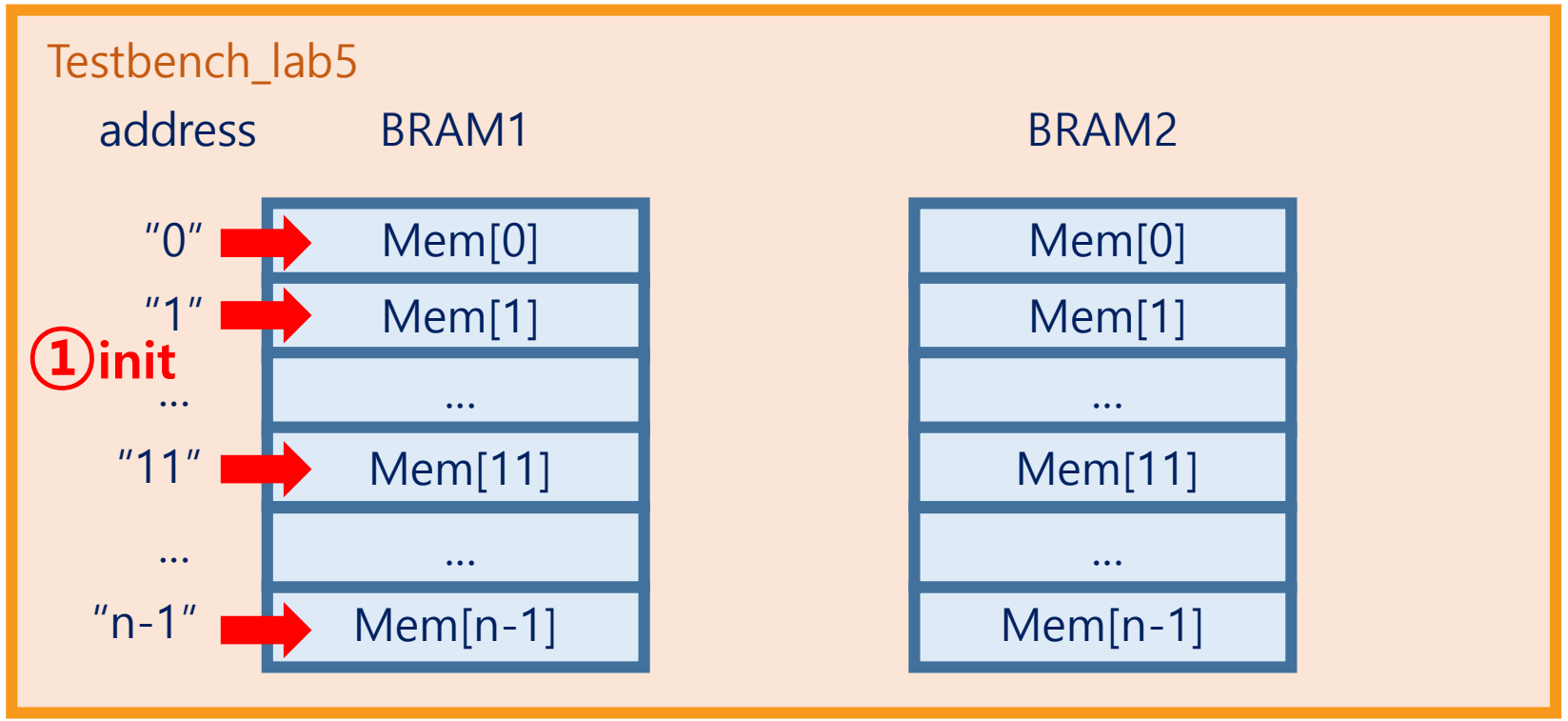
- Implement BRAM model & test-bench according to scenarios below.
  - BRAM costs 1 cycle for write and 2 cycles for read (i.e., BRAM returns value after next cycle when you assign address to read data).

- Scenario
  - ① **Make test-bench that instantiates two BRAMs and initialize one BRAM to store its address as data.**
    - (i.e., 'mem[0]' stores '0' and 'mem[1]' stores '1')
  - ② **Then, copy every data from the initialized BRAM to the other BRAM.**
    - Use I/O functions of verilog (e.g., '$readmemh' and '$writememh') & text-file (e.g., .txt) to initialize original BRAM and extract values from copied BRAM (access http://sunshowers.tistory.com/10 to get more information about I/O functions of Verilog).

Practice

① **Make test-bench that instantiates two BRAMs and initialize one BRAM to store its address as data.**

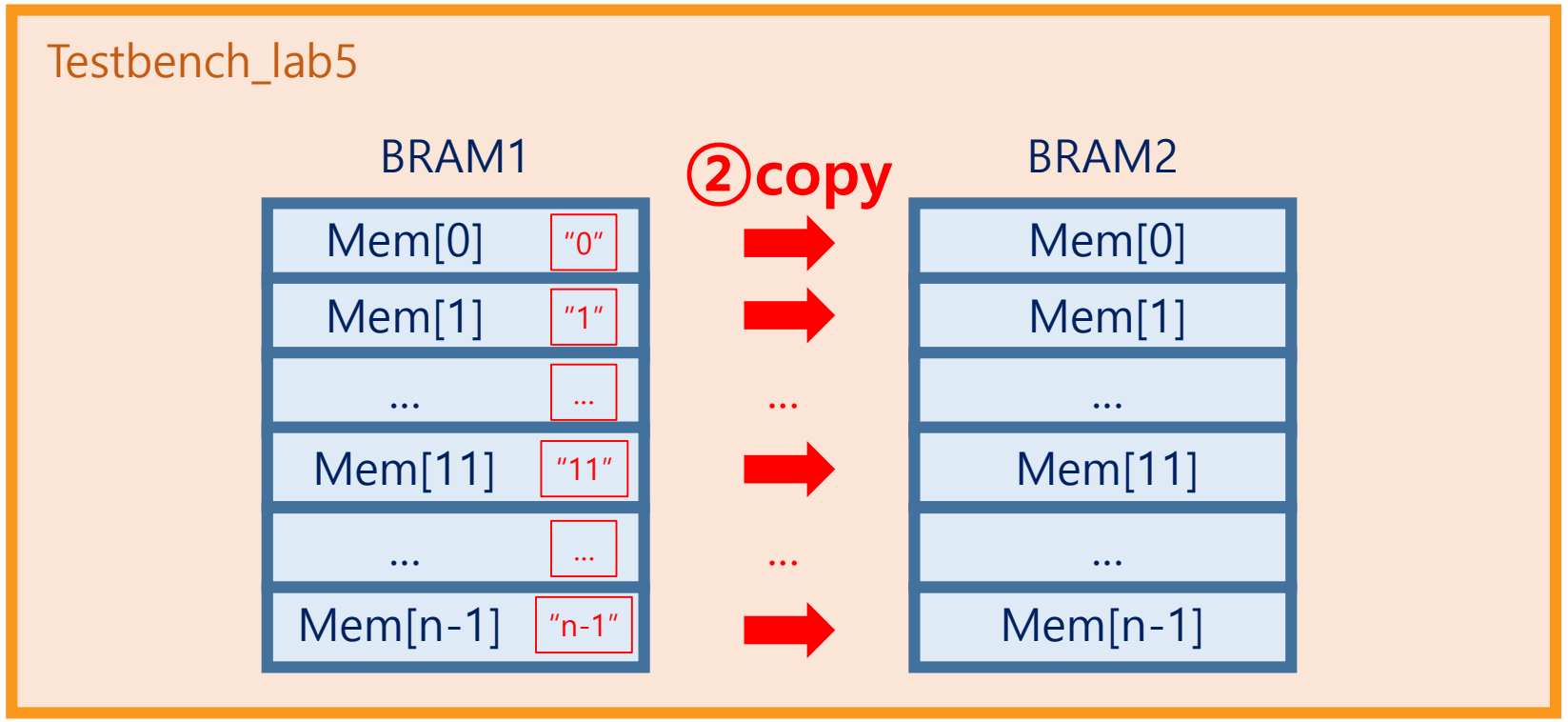
- (i.e., 'mem[0]' stores '0' and 'mem[1]' stores '1')
- Add source -> input.txt



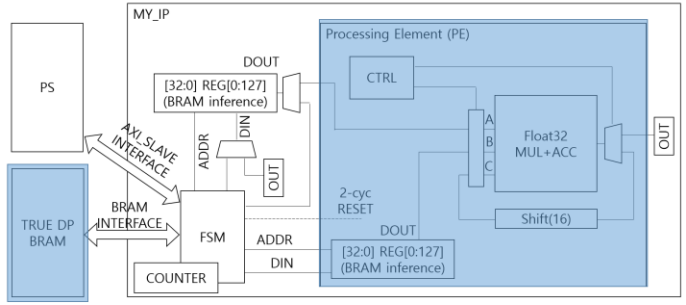
Practice

② **Then, copy every data from the initialized BRAM to the other BRAM.**

- Use I/O functions of verilog (e.g., '$readmemh' and '$writememh') & text-file (e.g., .txt) to initialize original BRAM and extract values from copied BRAM



Practice



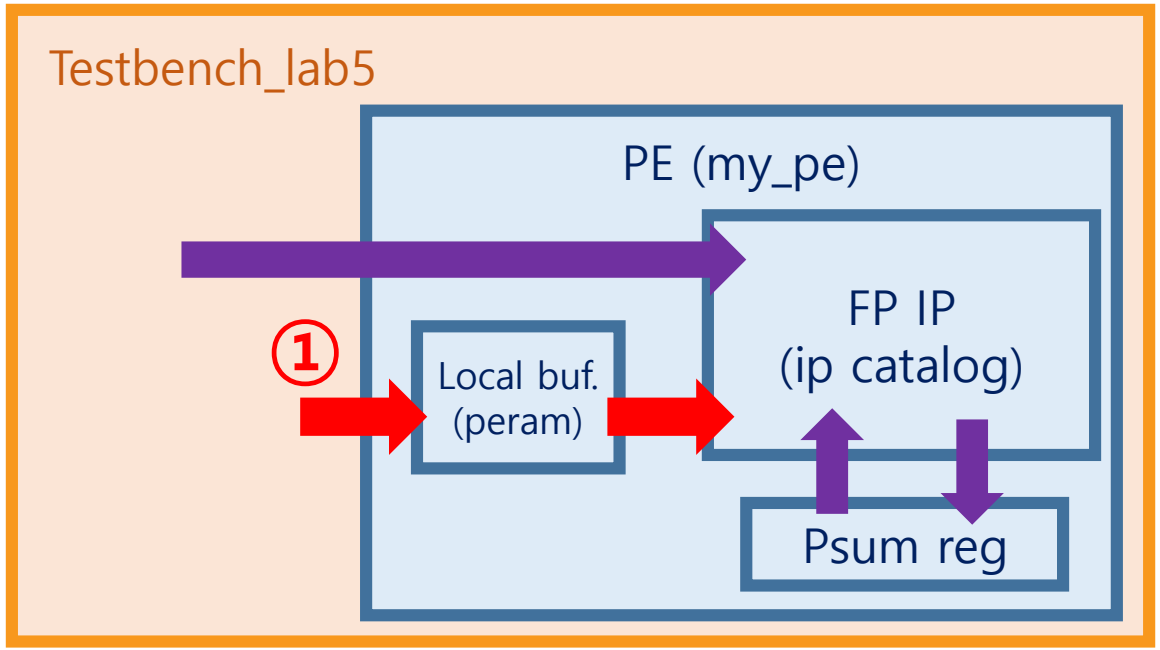
2. Implementing PE with floating-point fused multiply-adder

- Implement Processing Element (PE) & test-bench according to scenarios below.
- Processing Element (PE) consists of floating-point fused multiply-adder constructed with IP catalog (i.e., you implemented last week) and local register (i.e. block memory).

- Scenario
  - ① **PE first stores 16 data from outside(test-bench) into local register with 16 consecutive addresses, from address 0 to address 15.**
  - ② **Then, PE gets 16 new data from outside (test-bench) serially to perform MAC (Multiply-Accumulate) operations with the data stored in local register.**
    - Fused multiply-add: $result = (a_{in} * b_{in}) + c_{in}$
    - MAC (Multiply-Accumulate): $result = (a_{in} * b_{in}) + result$
  - ③ **Check 16 MAC results in testbench.**
    - Note that one MAC operation costs several cycles to compute result and it sets valid bit (i.e. dvalid) high when it finishes each operations.

PE schematic

① **PE first stores 16 data from outside into local register with 16 consecutive addresses, from address 0 to address 15.**

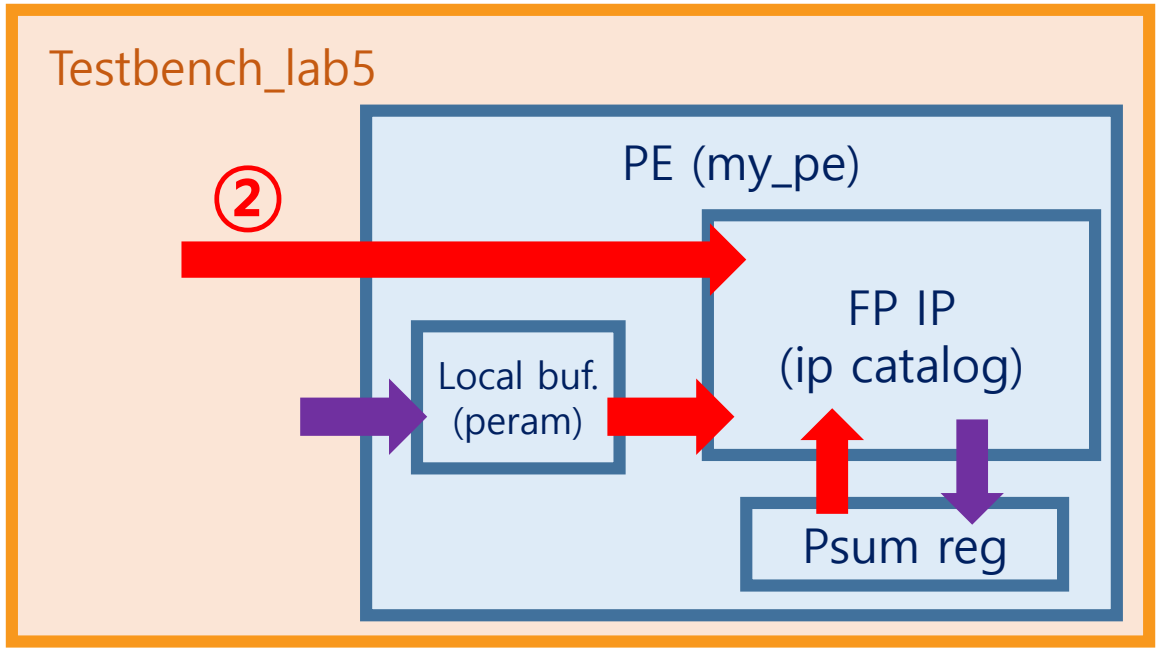


psum <= psum + ain * bin

PE schematic

② **Then, PE gets 16 new data from outside serially to perform MAC operations with the data stored in local register.**

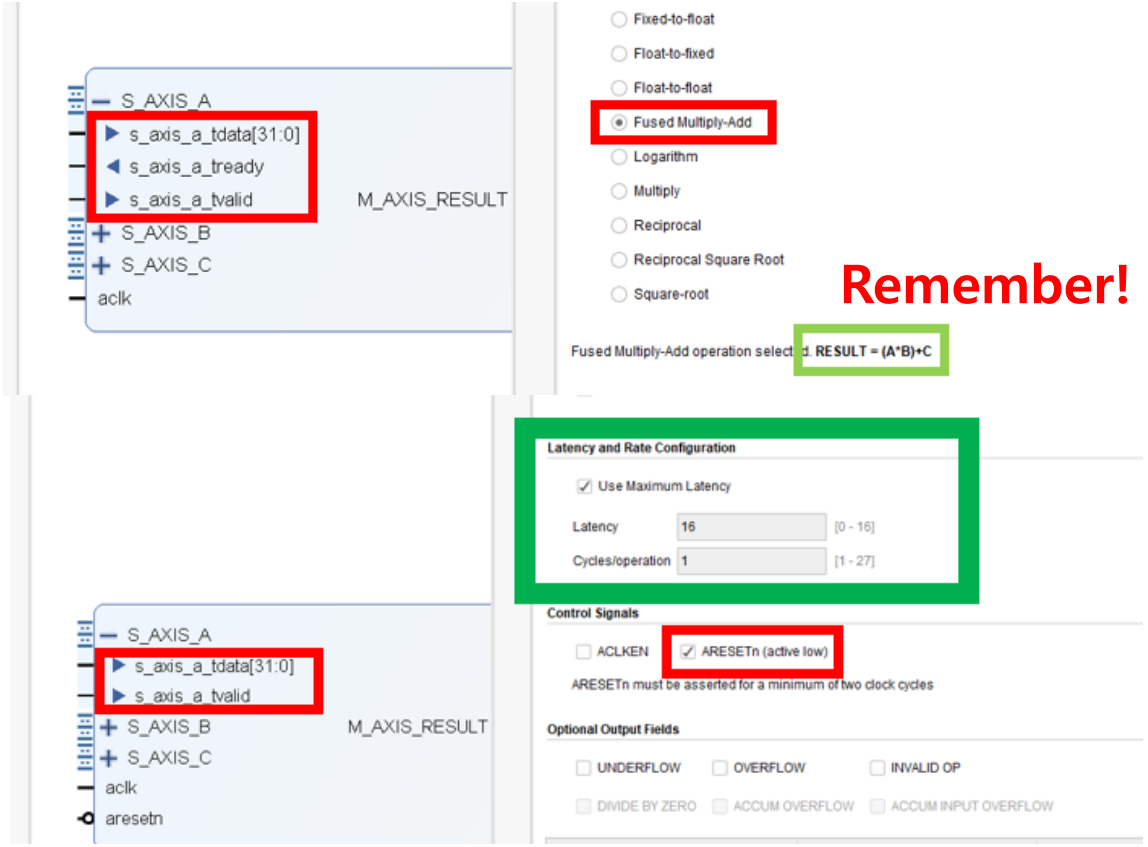
- MAC (Multiply-Accumulate): $result = (a_{in} * b_{in}) + result$

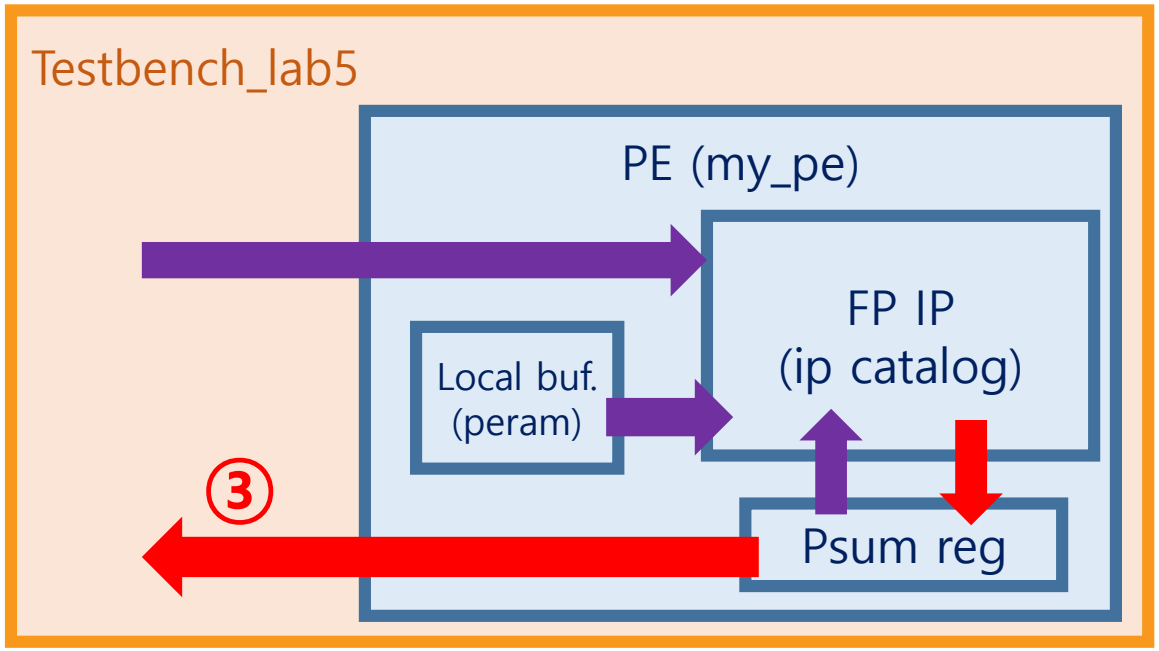


PE schematic

③ Check 16 MAC results.

- Note that one MAC operation costs several cycles to compute result and it sets valid bit (i.e. dvalid) high when it finishes each operations.





BRAM Model

```
module my_bram # (
    parameter integer BRAM_ADDR_WIDTH = 15, // 4x8192
    parameter INIT_FILE = "input.txt",
    parameter OUT_FILE = "output.txt"
  )(
    input wire [BRAM_ADDR_WIDTH-1:0] BRAM_ADDR,
    input wire BRAM_CLK,
    input wire [31:0] BRAM_WRDATA,
    output reg [31:0] BRAM_RDDATA,
    input wire BRAM_EN,
    input wire BRAM_RST,
    input wire [3:0] BRAM_WE,
    input wire done
  );
    reg [31:0] mem[0:8191];
    wire [BRAM_ADDR_WIDTH-3:0] addr = BRAM_ADDR[BRAM_ADDR_WIDTH-1:2];
    reg [31:0] dout;

    // code for reading & writing
    initial begin
      if (INIT_FILE != "") begin
          // read data from INIT_FILE and store them into mem
          ...
      end
      wait (done)
          // write data stored in mem into OUT_FILE
          ...
    end

   //code for BRAM implementation
    ...

endmodule
```

- **Memory configuration**
  - **Bit width of data:** 32 bits (4 bytes)
  - **Bit width of address:** 15 bits
  - **# of memory entries:** 8192 (== 2^{13})
  - Each external address of which size is 15 bits (i.e., **BRAM_ADDR**) is associated with 1 byte of data. Last two bits are masked and assigned to internal address of which size is 13 bits (i.e., **addr**). It is associated with each entry of **mem** (i.e. it is associated with 4 bytes of data).
- **BRAM_ADDR:** external address
- **BRAM_CLK:** clock signal
- **BRAM_WRDATA:** Data input port of BRAM.
- **BRAM_RDDATA:** Data output port of BRAM.
- **BRAM_EN:**
  - **(BRAM_EN==1): BRAM enabled. BRAM is available for read or write operation.**
  - If all of **BRAM_WE** is equal to 0 and **BRAM_EN** == 1, read **mem[addr]** into **BRAM_RDDATA**
- **BRAM_WE:**
  - (**BRAM_WE[i]**==1): **BRAM_WRDATA[8*(i+1)-1:8*i]** is stored into **mem[addr][8*(i+1)-1:8*i].**
- **BRAM_RST:** ==1, **BRAM_RDDATA** prints 0
- **done:** ==1, write data stored in mem into **"OUT_FILE"**

Processing Element (PE)

```
module my_pe #(
        parameter L_RAM_SIZE = 6
    )
    (
        // clk/reset
        input aclk,
        input aresetn,
        // port A
        input [31:0] ain,
        // peram -> port B
        input [31:0] din,
        input [L_RAM_SIZE-1:0]  addr,
        input we,
        // integrated valid signal
        input valid
        // computation result
        output dvalid,
        output [31:0] dout
    );

   (* ram_style = "block" *) reg [31:0] peram [0:2**L_RAM_SIZE - 1];  // local register

   ...

endmodule
```

- **Local register configuration**
  - **Bit width of data:** 32 bits (4 bytes)
  - **Bit width of address:** 6 bits
  - **# of memory entries:** 64 (==2^6)
  - Each address of which size is 6 bits (i.e., **addr**) is associated with 4 bytes of data.
- **aclk:** clock signal
- **aresetn:** negative reset; ==0, reset is activated;
- **ain:** input ports directly connected to MAC
- **din:** input ports connected to local register
- **addr:** input address
- **we:** write enable signal for local register;
  - ==1, **din** is stored into **peram[addr]**;
  - ==0, **peram[addr]** is assigned to one of inputs of MAC.
- **valid:** ==1, MAC gets inputs from its input ports and starts computation; It is divided into three valid signals and assigned to MAC (i.e., it is assigned to **s_axis_a_tvalid, s_axis_b_tvalid** and **s_axis_c_tvalid**).
- **dvalid:**
  - ==1, result data from MAC is valid;
  - ==0, result data from MAC is not valid; It is assigned to result valid signal of MAC (i.e., **m_axis_result_tvalid**).
- **dout:** if(dvalid==1), it prints result data from MAC; otherwise it prints 0.

Homework

- Requirements
  - Result
    - Attach your project folder with all your verilog codes (e.g., BRAM, PE, test bench)
    - Attach your BRAM waveform(simulation result) with [student_number, name]
      - Test the scenario in slide6.
      - The correct waveform should be shown to confirm the operation of your code.
      - Refer to Practice3 about screenshot.
    - Attach your PE waveform(simulation result) with [student_number, name]
      - Test the scenario in slide9.
      - The correct waveform should be shown to confirm the operation of your code.
      - Refer to Practice3 about screenshot.
  - Report
    - Explain operation of BRAM with waveform that you implemented
    - Explain operation of PE with waveform that you implemented
    - In your own words
    - Either in Korean or in English
    - # of pages does not matter
    - **PDF only!!**
  - **Result + Report to one .zip file**
- Upload (.zip) file on ETL
  - Submit one (.zip) file
    - zip file name : [Lab05]name.zip (ex : [Lab04]홍길동.zip)
  - Due: 4/14(TUE) 23:59
    - **No Late Submission**