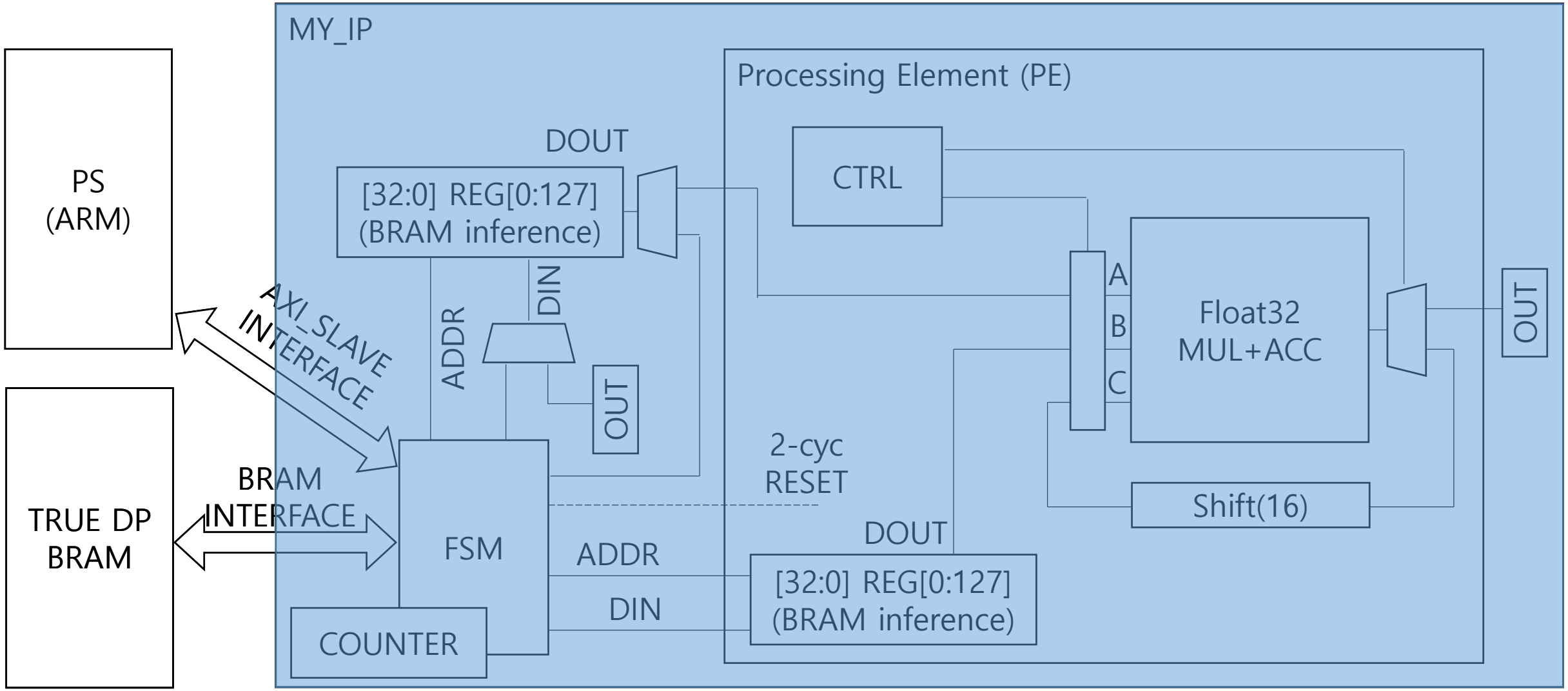
# Practice 6

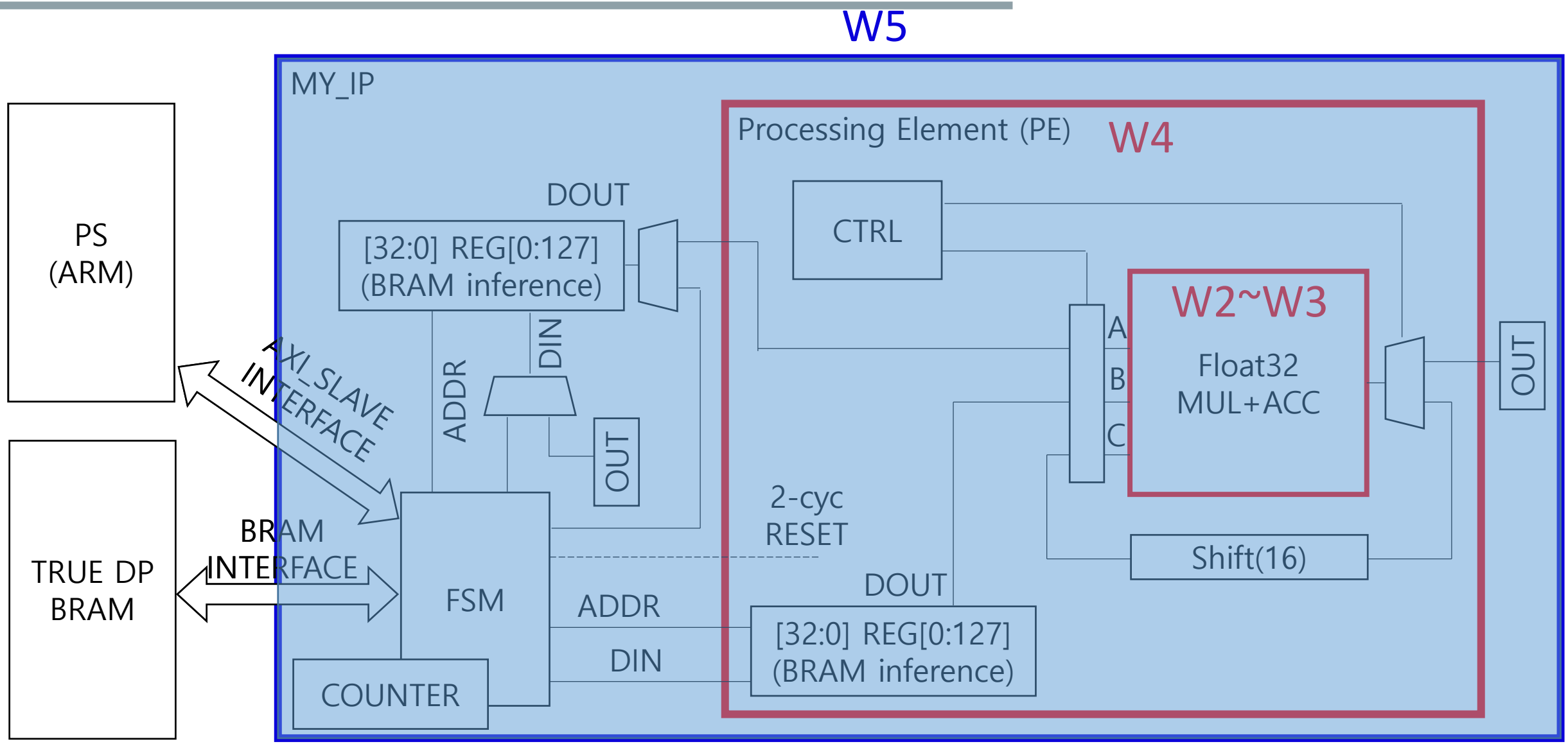
- **BRAM to PE controller**

Computing Memory Architecture Lab.

# Final Project Overview: Matrix Multiplication IP



# Final Project Overview: Matrix Multiplication IP



# Final Project Overview: Matrix Multiplication IP

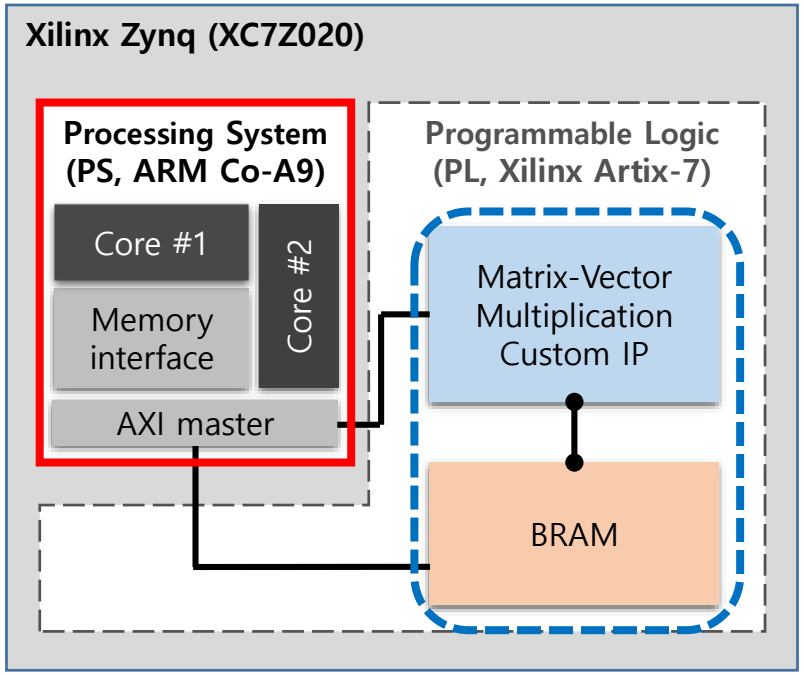
- MLP is our application
  - Each layer is matrix-vector multiplication, e.g., 256x1024 matrix * 1024-d vector ➔ 256-d vector
- ARM CPU runs the main function which calls your MV IP on PL
  - MV for 64x64 weight matrix * 64-entry input vector multiplication
- BRAM is used for data transfer between SW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

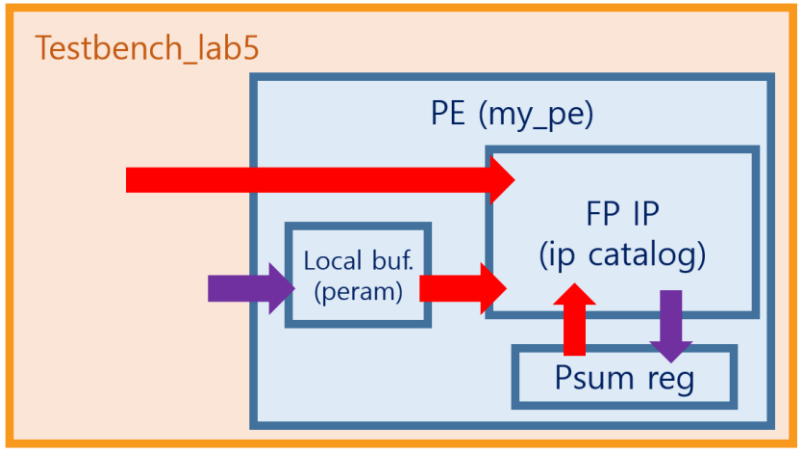
Vector-Vector Multiplication



# Main Practice

# From Lab 5 to Lab 6

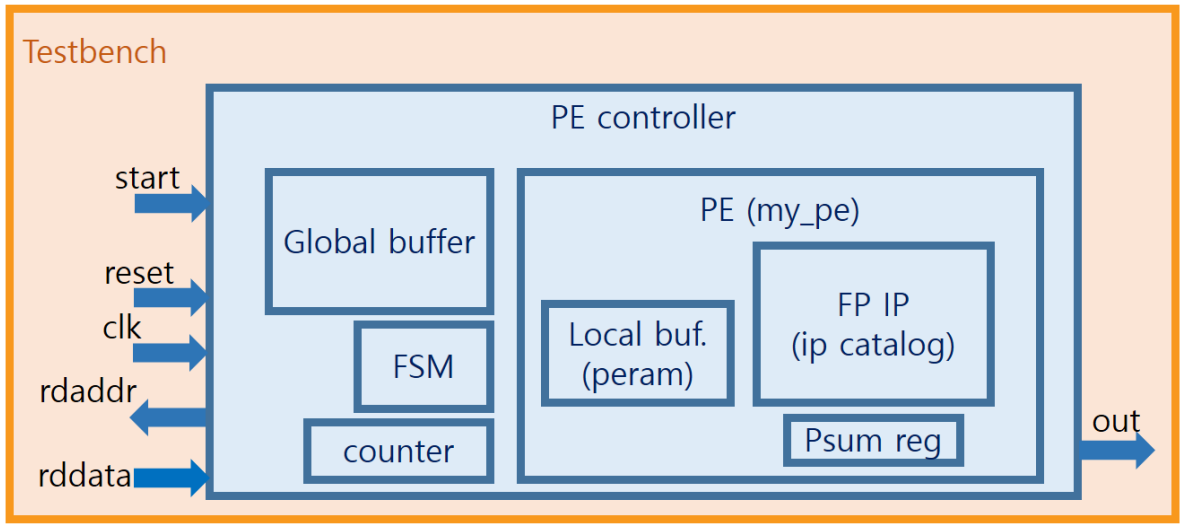
- **Lab 5:**
  - We implemented PE with local registers and tested MAC(multiply & accumulate) operation.
    - with 16 sequential inputs from testbench and 16 data on local registers.




- **Lab 6:**
  - PE controller
    - Controlling PE using FSM

# Practice

- **Implementing PE controller**
  - Implement PE controller based on PE you made in Lab5.
  - PE Controller
    - PE controller consists of PE and FSM.
    - FSM controls PE to calculate inner product (e.g., data[0]*data[16] + data[1]*data[17]+ … + data[15]*data[31]) with several states.
      - Inner product can be made with MAC operation of PE

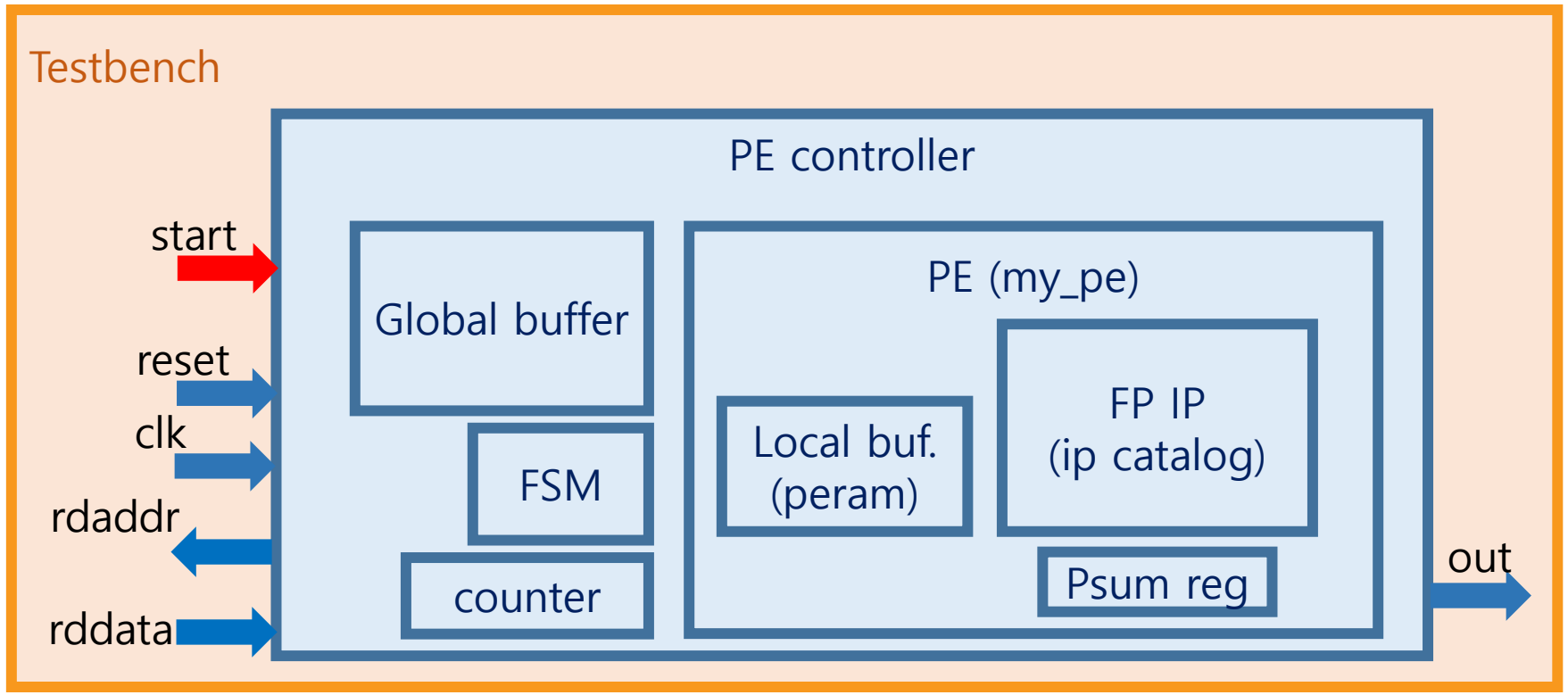


# Practice

- **PE Controller FSM states**
  - **S_IDLE:**
    - Idle state. Starts operation with state transition to S_LOAD, when input ‘start==1’.
  - **S_LOAD:**
    - Loads 16 data into local buffer in PE and load 16 data into global BRAM.
    - After completion of data loading, state moves to S_CALC.
  - **S_CALC:**
    - Calculate “inner product” with 16 data from local memory of PE, 16 data from global memory.
    - After calculation, state moves to S_DONE.
  - **S_DONE:**
    - Gives ‘done’ as 1.
    - After a few cycles(ex. 5 cycles), state moves to S_IDLE

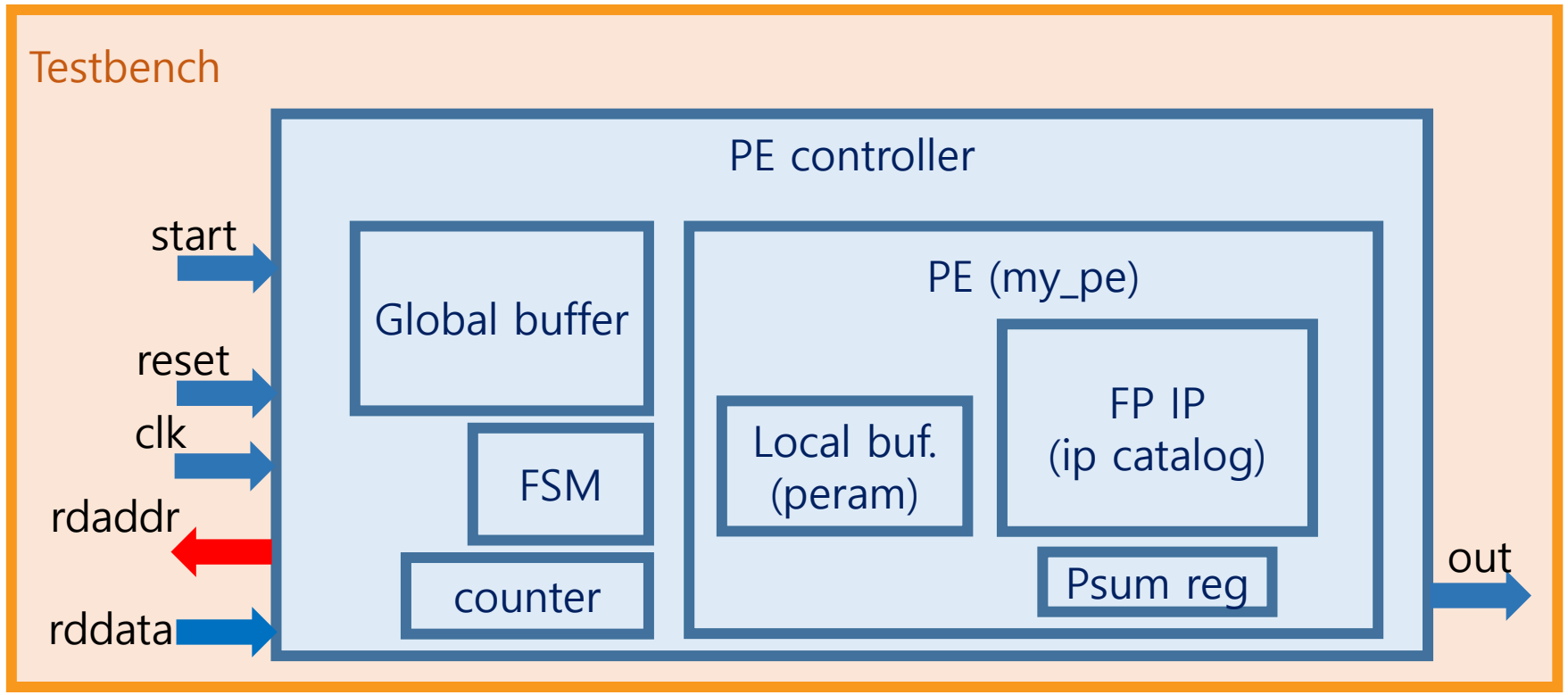
# Idle state

- **S_IDLE:**
  - Idle state. Starts operation with state transition to S_LOAD, when input ‘start==1’.



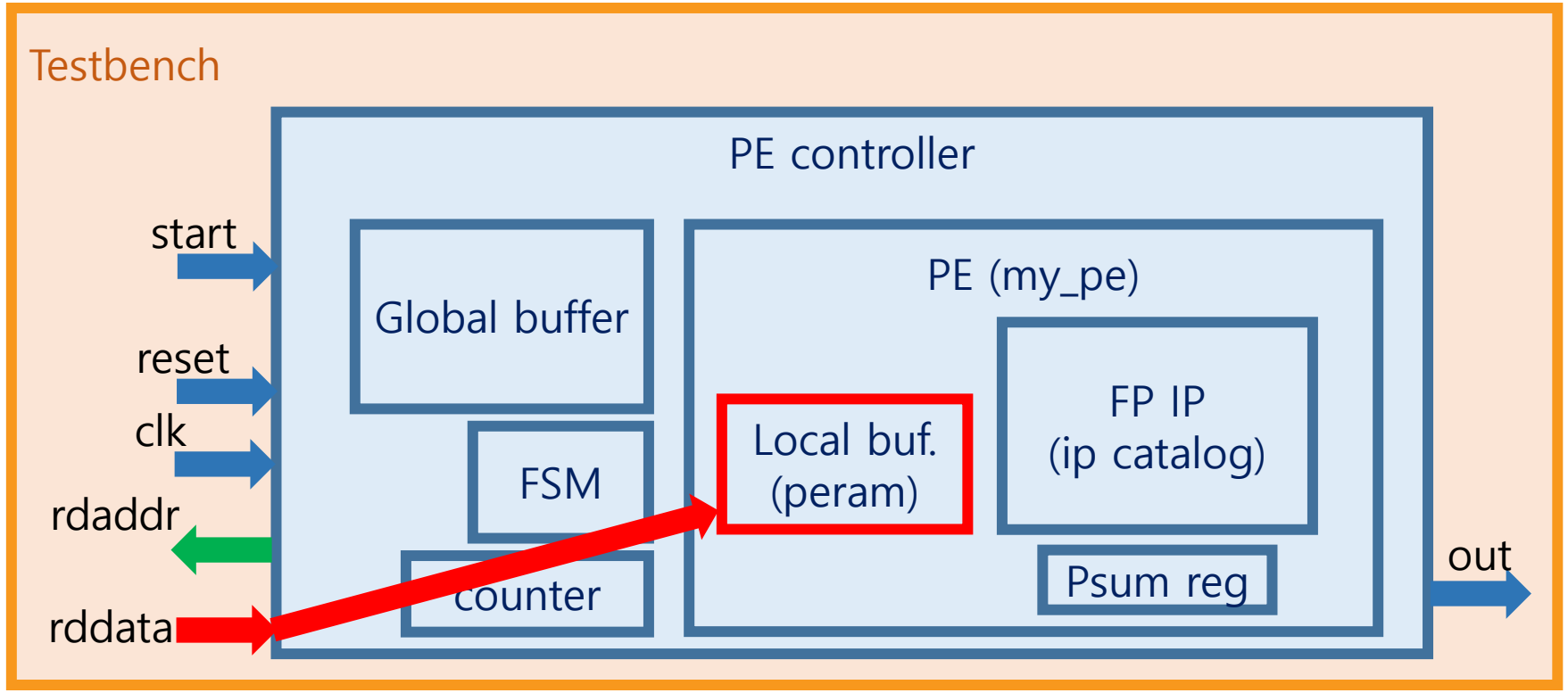
# Load state

- **S_LOAD:**
  - Loads 16 data into local buffer in PE and load 16 data into global BRAM.
  - After completion of data loading, state moves to S_CALC.



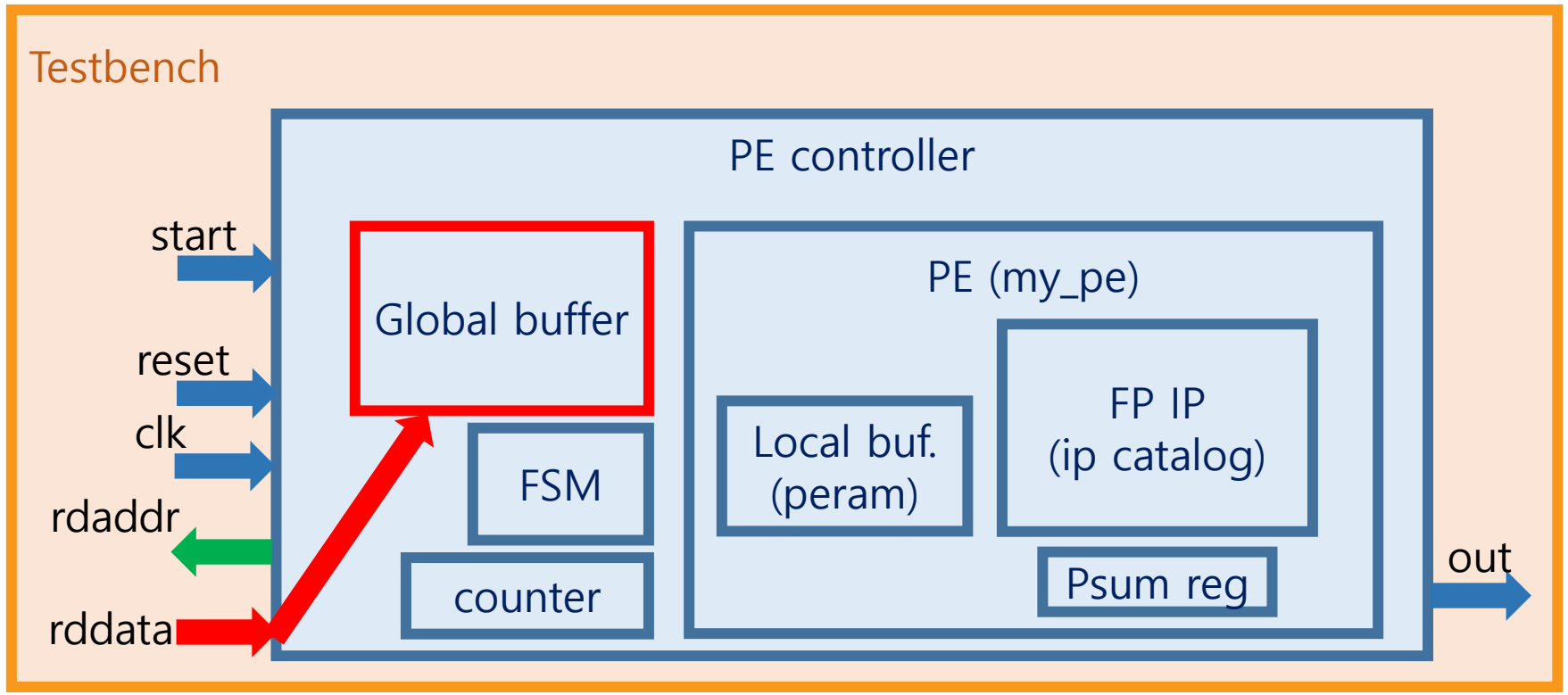
# Load state

- **S_LOAD:**
  - Loads 16 data into local buffer in PE and load 16 data into global BRAM.
  - After completion of data loading, state moves to S_CALC.



# Load state

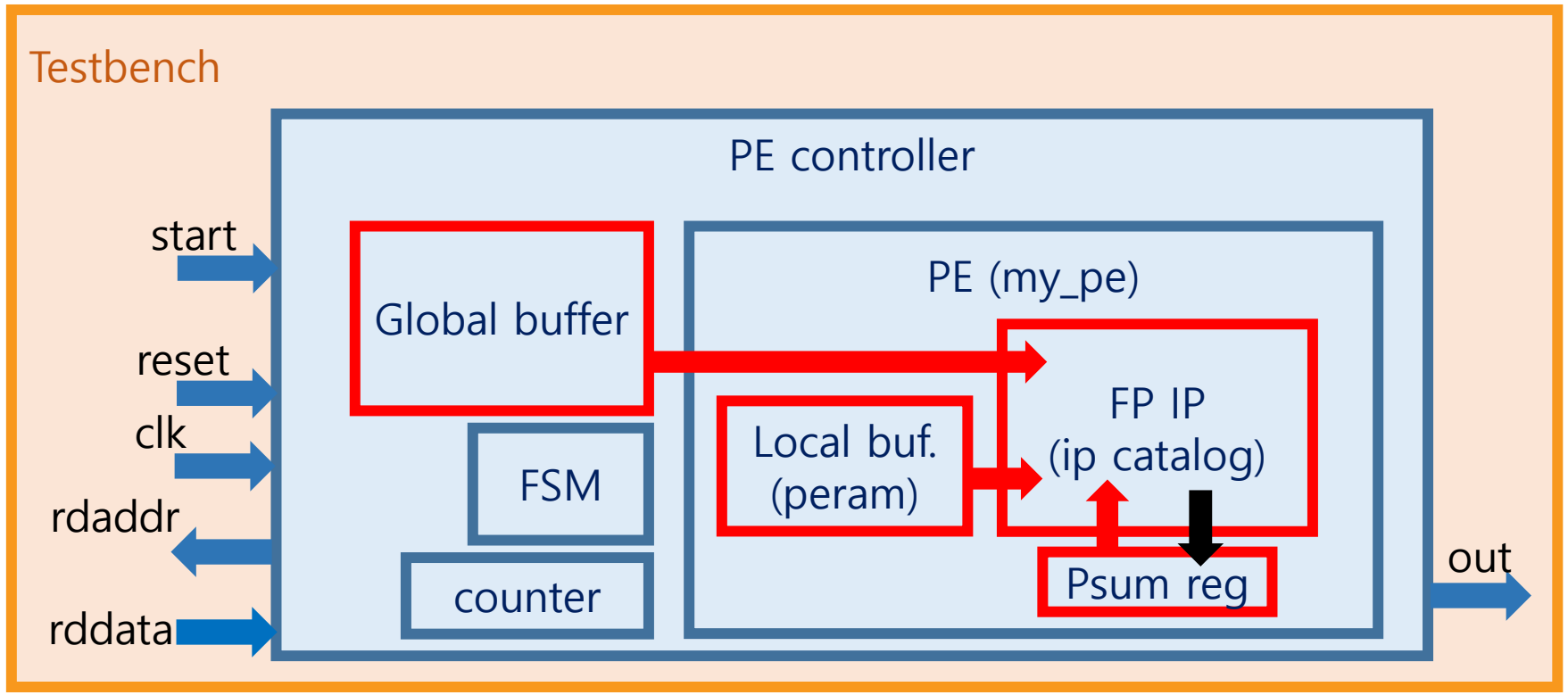
- **S_LOAD:**
  - Loads 16 data into local buffer in PE and load 16 data into global BRAM.
  - After completion of data loading, state moves to S_CALC.



# Calc state

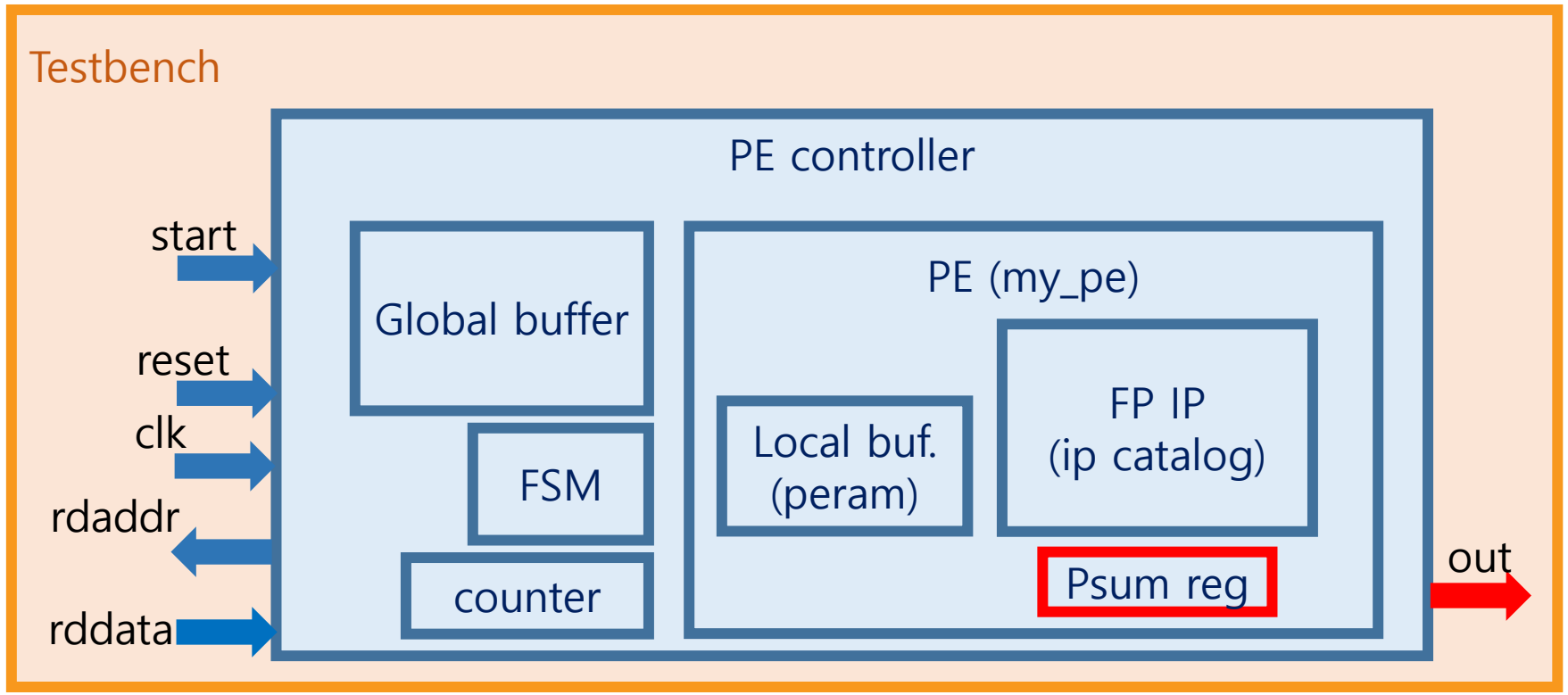
- **S_CALC:**
  - Calculate “inner product” with 16 data from local memory of PE, 16 data from global memory.
  - After calculation, state moves to S_DONE.

$$result = (a_{in} * b_{in}) + result$$



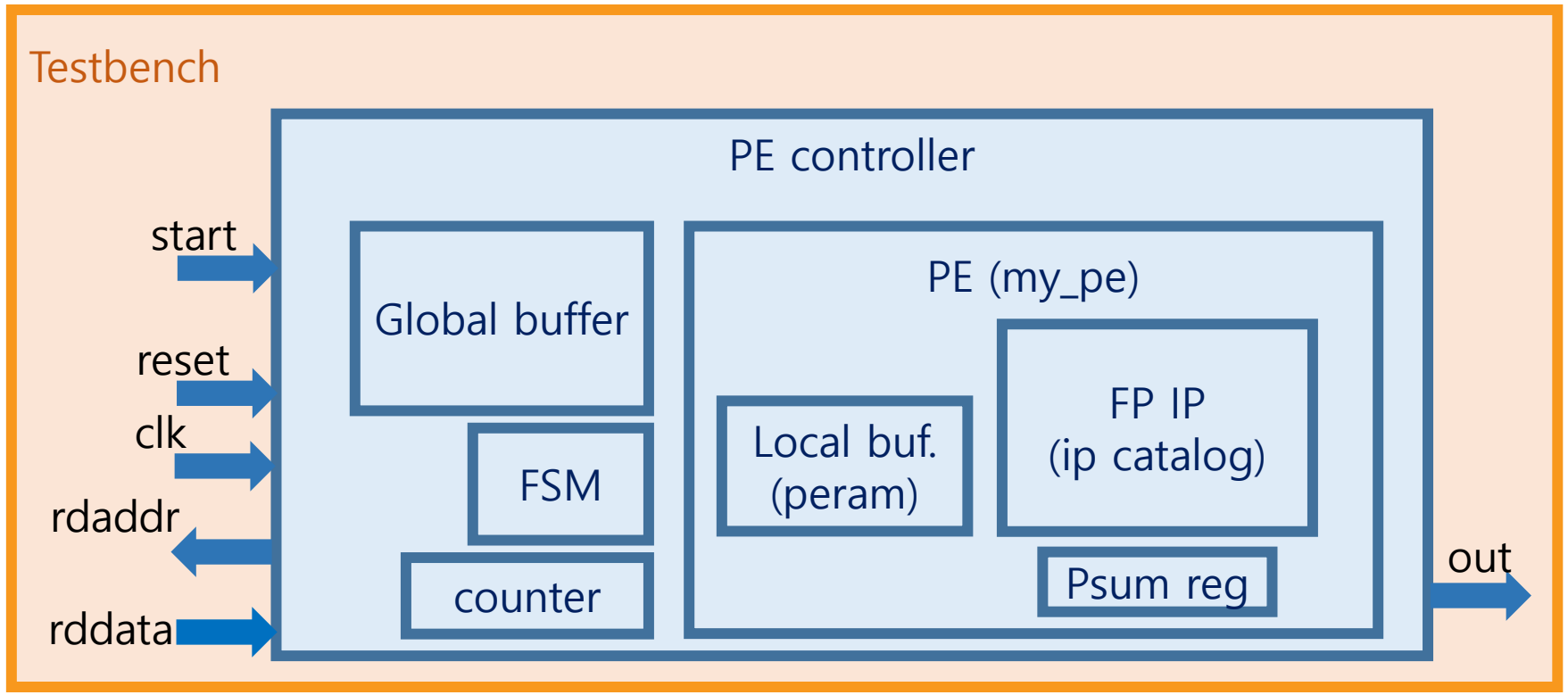
# Done state

- **S_DONE:**
  - Gives ‘done’ as 1.
  - After 5 cycles, state moves to S_IDLE



# Done state

- **S_DONE:**
  - Gives ‘done’ as 1.
  - After 5 cycles, state moves to S_IDLE



# Homework

- Requirements
  - Result
    - Attach your project folder with all your verilog codes (e.g., PE Controller, test bench)
    - Attach your PE Controller waveform(simulation result) with [student_number, name]
      - Test the scenario in slide[9~14].
      - The correct waveform should be shown to confirm the operation of your code.
      - Refer to Practice3 about screenshot.
  - Report
    - Explain operation of PE Controller 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 : [Lab06]name.zip (ex : [Lab06]홍길동.zip)
  - Due: 5/5(TUE) 23:59
    - **No Late Submission**

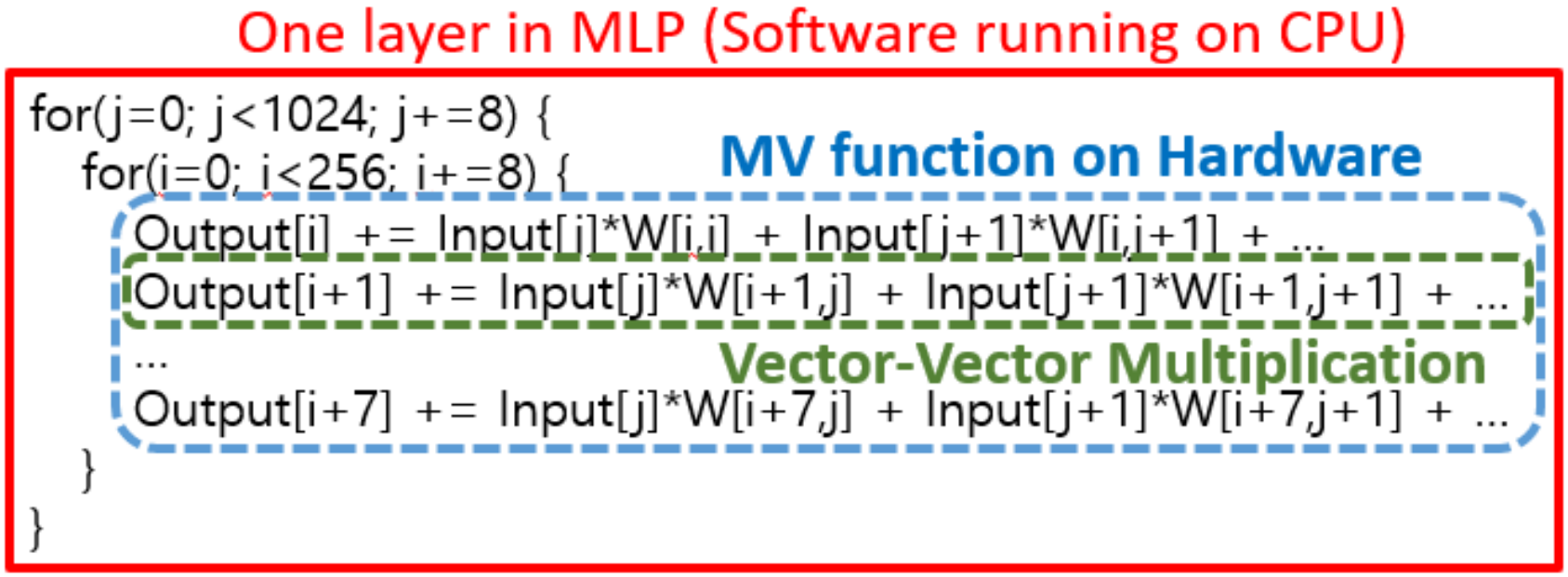
# Future Work

This is not included in lab6.

But let’s think about the future work that we have to do finally.

# VV to MV multiplication

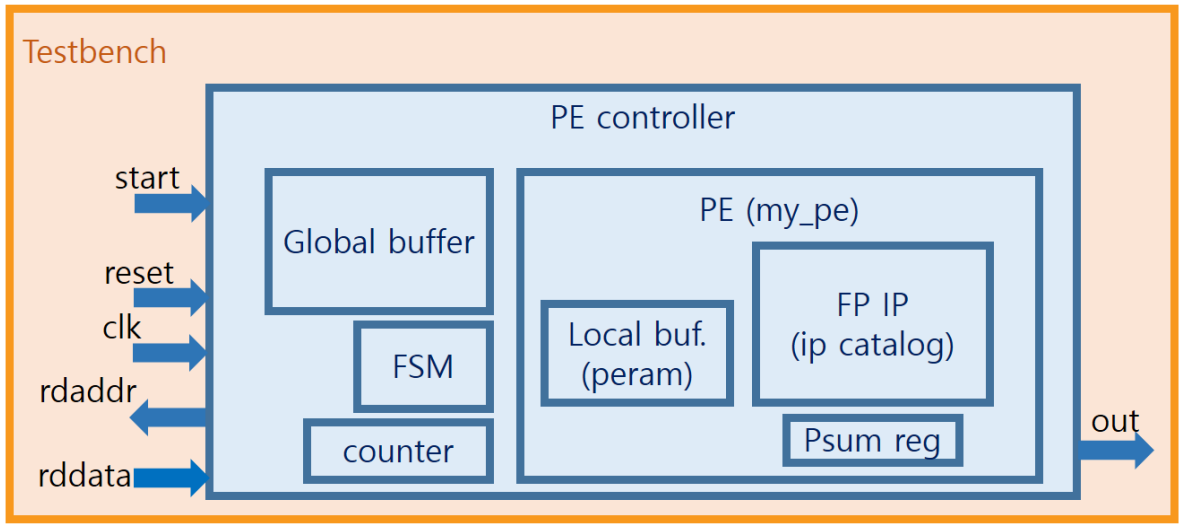
- **Our Final Application (red box)**
  - MLP
- **What we did by lab6 (green box)**
  - Vector-Vector(VV) multiplication
- **What we're going to do in the near future (blue box)**
  - Matrix-Vector(MV) multiplication



# VV to MV multiplication

- **PE & PE Controller for matrix multiplication**
  - What do we have to do?

- Pe controller on lab6




- Our application

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

Vector-Vector Multiplication