Term project #1: Matrix-vector multiplication w/ half-precision input data

05/10/2018

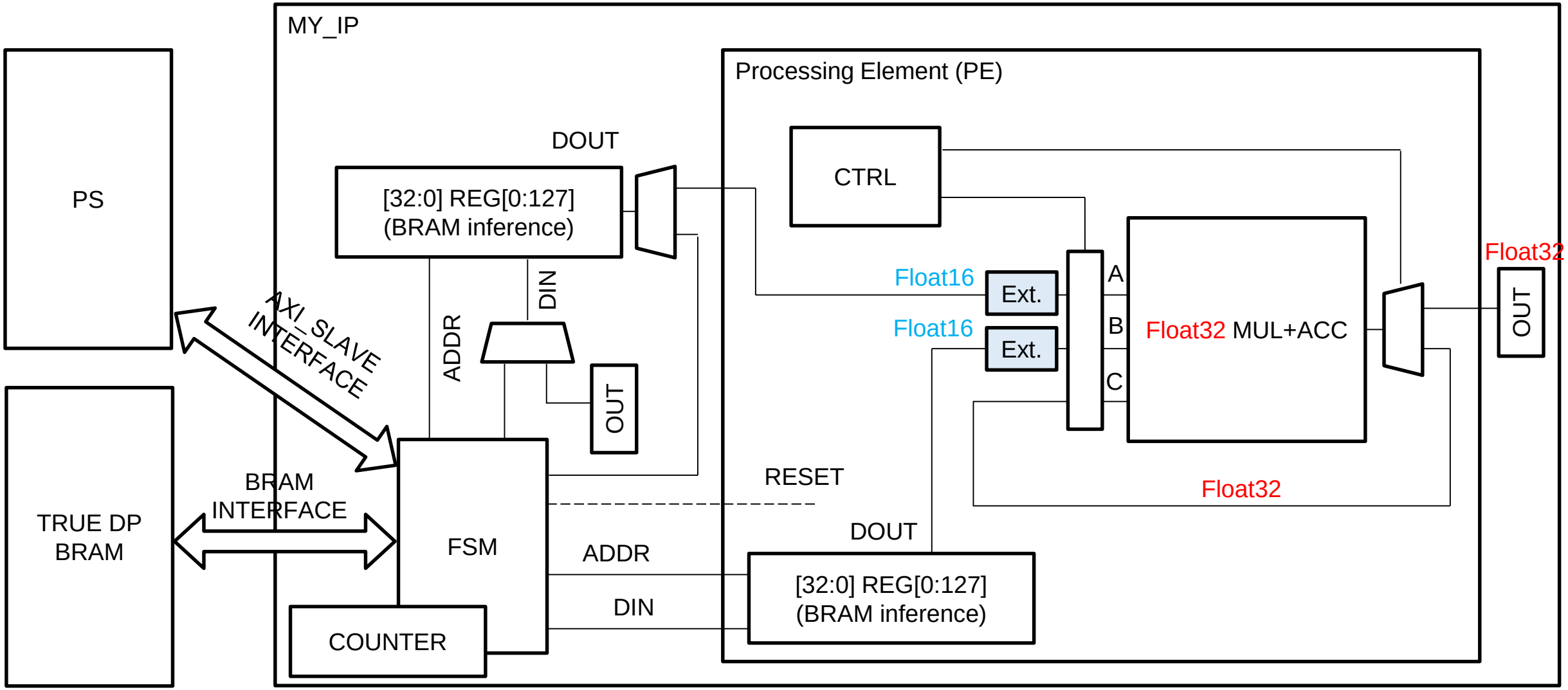
4190.309A: Hardware System Design
(Spring 2018)

Introduction to term project #1

- Matrix and vector multiplication
  - Multiply matrix and vector to produce vector
  - 4x4 MV multiplier example:

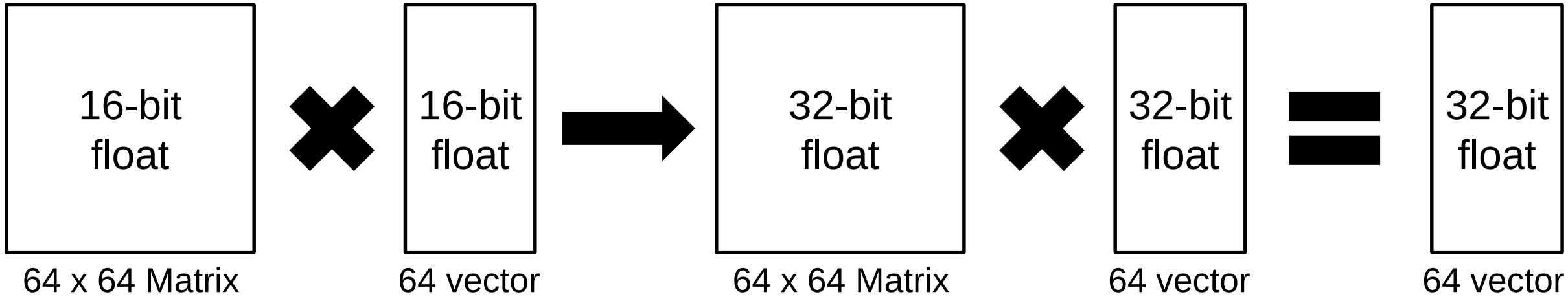
$$\begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix} \times \begin{bmatrix} b1 \\ b2 \\ b3 \\ b4 \end{bmatrix} = \begin{bmatrix} c1 \\ c2 \\ c3 \\ c4 \end{bmatrix}$$

Overview: Matrix-vector multiplication IP



Specification of Term Project #1

- **Goal**: Implement 64x64 MV multiplication IP with matrix and vectors described in *half-precision data format*
  - Input / output format is provided (See the next page)
  - Convert FLOAT16 into FLOAT32 (See p3, Ext. logic)
  - Calculate 64x64 matrix – 64 vector multiplication.



Specification of Term Project #1

- **Scoring policy**
  - *No scoring for performance now*, but we will perform an optional project
    - e.g. Performance tuning with a fixed point format
  - Scoring of IP will be performed as follows
    - By precision lost of conversion, error less than 1% will be accepted as correct answer.
    - Penalties for incorrect results (e.g. missing partial results, …)
    - Penalties for non-general functionality (e.g. work only once, …)
    - Show your demo results in front of TAs
    - You must show tests at least 3 times with randomly generated inputs
    - **Due date**: **Lab time at 5/31 Thr. and 6/1 Fri.**

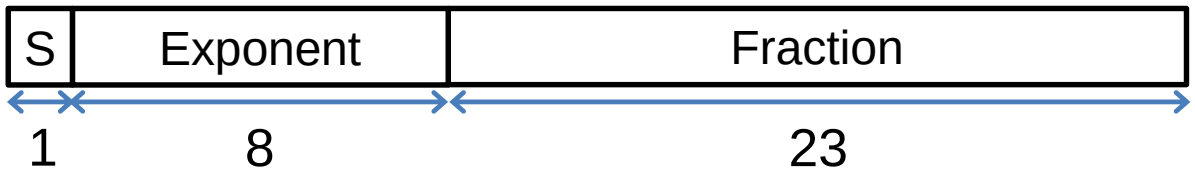
Specification of Term Project #1

- File description for software parts
  - `input.txt`
  - `generator.c` → `input_generator (bin)`
  - `main.cpp, zynq.cpp, zynq.h` → `project1 (bin)`
  - `Makefile`

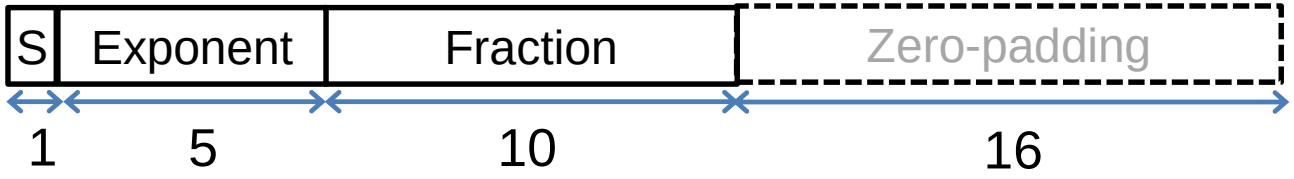
- File description for hardware parts (verilog)
  - Start from Lab 8
    - Design & edit IP implementation for 64x64 MV multiplication (pe_controller.v, etc.)
    - Generate bitstream
    - Execute *project1*

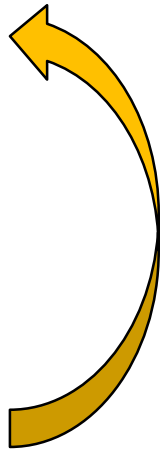
Specification of Term Project #1

- Data format
  - **FLOAT32**: IEEE754 single-precision (**used for computation**)
    - 32bit (1bit sign / 8bit exponent / 23bit fraction)



  - **FLOAT16**: IEEE754 half-precision (**used for BRAM**)
    - 16bit (1bit sign / 5bit exponent / 10bit fraction)
    - We'll use this format, but have to convert it into FLOAT32 when computing with floating-point MAC IP

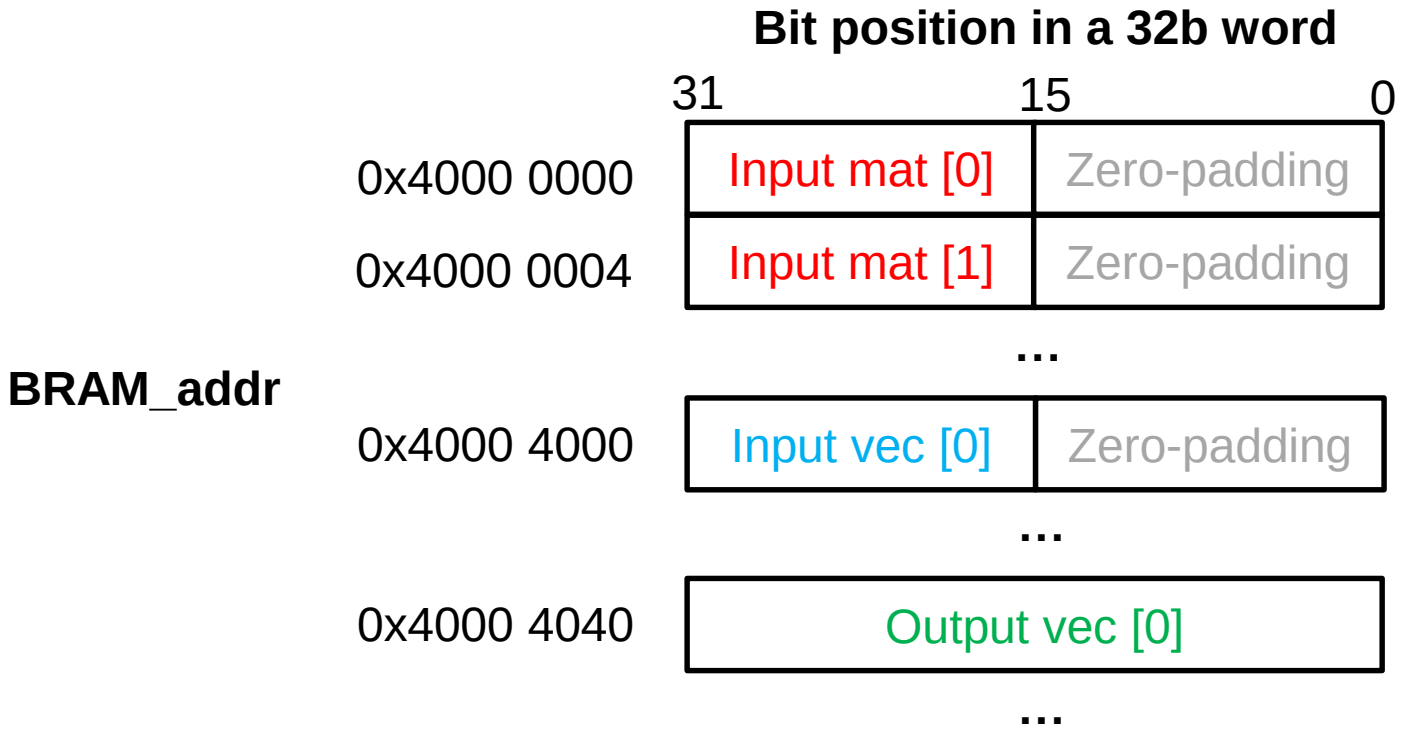




You must implement FLOAT16 to FLOAT32 converter for internal use in MV multiplication IP

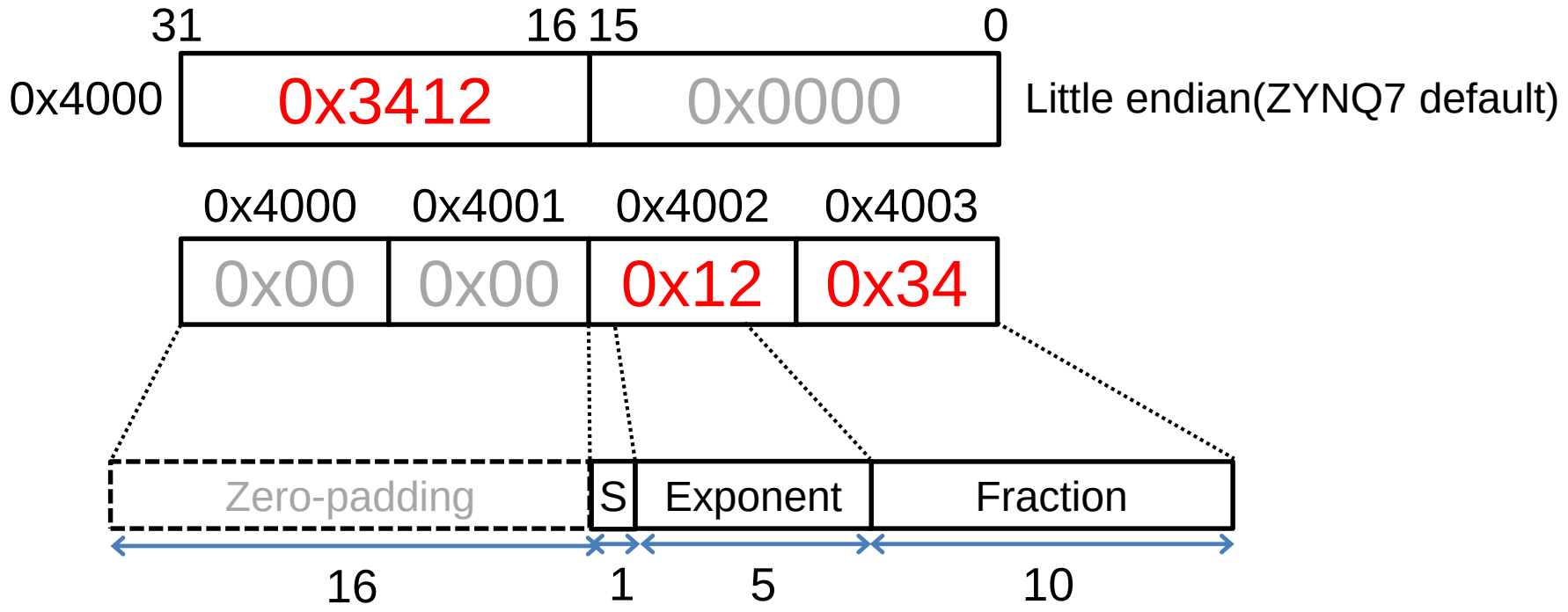
Specification of Term Project #1

- Input / output format
  - Input : 4 byte-aligned mini-float (16-bit data)
  - Valid data is stored at MSB [31:16]
  - Output should hold 32-bit floating point (Little-endian)



Specification of Term Project #1

- Closer look on input format
  - Example value : 0x1234
  - Looks like upper figure in C code
  - Actual byte order is like in second figure.



Specification of Term Project #1

- CPU-implemented code.
  - arm_calculate() function do the same thing (with general-purpose H/W).
  - Use it as reference for your hardware.

```
static inline float f16_to_f32(const uint32_t *input)
{
        char *iptc = (char*)input + 2;

        uint32_t half_precision = 0;
        memcpy(&half_precision, iptc + 1, 1);
        memcpy(((void*)&half_precision) + 1, iptc, 1);

        uint32_t opt    = ((half_precision & 0x8000) << 16)
                        | ((((half_precision & 0x7C00) >> 10) + 112) << 23)
                        | ((half_precision & 0x3FF) << 13);

        float casted_opt;
        memcpy(&casted_opt, &opt, sizeof(float));
        return casted_opt;
}
```

Specification of Term Project #1

- Write your C code.
  - Map BRAM to virtual memory (use `mmap()` system call).
  - Copy DRAM's input data to BRAM.
  - Run your IP after start time is measured.
  - It will be reported as cheating if you do it before time measurement starts.
  - Modify zynq.cpp file only.
- Input/output
  - Input matrix : `ipt_matrix_f16`
  - Input vector : `ipt_vector_f16`
  - Output vector : `your_vector_f32`
    - After calculation is done, write your data to `your_vector_f32`

Specification of Term Project #1

- Compile and run your code
  - Compile : `$> make`
  - All warnings will be considered as error (-Werror compiler option).
    - (i.e., C++ code with warning will not be accepted.)
  - Do not modify Makefile. If you want to add compile options, ask TA for it.
  - Run : `$> ./project1`
- Use input generator
  - Usage : `$> ./input_generator > input.txt`

Review of previous labs

- It is highly recommended to review Lab materials !
  - **Lab 3**: Processing Element (PE) and BRAM model
    - This BRAM is just for simulation, you will use BRAM IP in this term project
  - **Lab 4**: PE arrays and its controller for Inner Product (FSM)
  - **Lab 6**: Block design tutorial
  - **Lab 7**: Integration of example shifter IP and its runtime debugging
  - **Lab 8**: An example full system of Inner Product unit and its customization
    - Block design is pre-described by tcl scripts, you can just start from it !
    - See how *$LAB8/hw_2* is used for simulation.
    - See how *$LAB8/hw_3* is used to generate bitstream, edit IP (myip) and run on FPGA

Demo

- Check correct output of 64x64 matrix * 64 vector output
  - This is example results with sample inputs
  - Check error rate with baseline result (32-bit * 32-bit multiplication).