

Hardware System Design Final Exam

2019. 6. 11

(5 questions, 50 points, 12 pages)

ID :

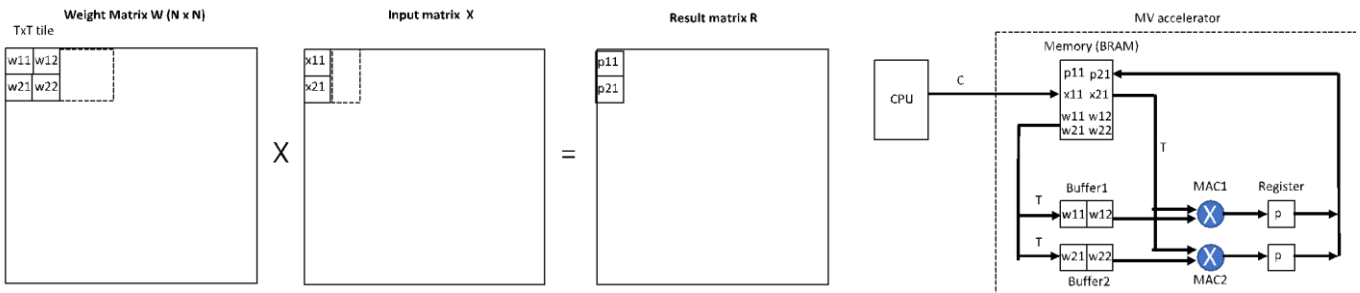
Name :

Q1. (Matrix-vector multiplication in hardware)

The following figure (left) shows a tile-based matrix-matrix multiplication. Assume that we use $T \times T$ weight tile and $1 \times T$ input tile. The figure shows the case that $T = 2$.

Assume that we use our MV accelerator shown in the following figure (right). The figure shows the number of execution cycles on each operation for the multiplication of a weight tile and an input tile. The communication (for the transfer of a pair of weight and input tiles) from CPU to the FPGA memory (BRAM) of MV accelerator takes C cycles. Weight transfer from the FPGA memory to each PE buffer takes T cycles, which gives T^2 cycles for the transfer of entire $T \times T$ weight tile from the FPGA memory to T buffers. Broadcast of input data from the FPGA memory to MAC units and computation takes T cycles to consume a $1 \times T$ input tile.

Each of weight (W), input (X) and result (R) matrices is a $N \times N$ matrix. We assume that N is a multiple of T and returning the result from MV accelerator to CPU does not take any cycle.



Problem 1-1. Case A (3 points)

Assume the CPU sends a pair of weight tile and input tile to the MV accelerator for each tile-based computation where the data transfer from CPU to MV accelerator, and FPGA memory to buffers occur and broadcast/computation is performed. Calculate the total execution cycle of matrix multiplication of weight W and input X to obtain matrix R . (Hint: # of transfers of each weight tile is N for the entire $W \times X$ multiplication due to $1 \times T$ input tile)

Problem 1-2. Case B (5 points)

A weight tile can be reused by the MV accelerator over multiple tile-based MV computations. When a weight tile is reused, it can be stored in the FPGA memory (BRAM and buffers in the figure) of MV accelerator.

Assume the communication cycle is constant, i.e., C cycles whether both weight and input tiles are transferred or only input tile is transferred. We also assume that only a pair of weight tile and input tile can be stored in the local memory of MV accelerator. Calculate the minimum total execution cycle of $W \times X$ matrix multiplication when the weight tile is reused.

Problem 1-3. Execution cycle comparison (2 points)

Assume $N = 100$, $T = 10$, and $C = 1$. Compare the total execution cycles of Cases (A) and (B).

Q2. (Low Precision Computation)

Consider a HW accelerator with floating point 32-bit (fp32) mechanism. To exploit 8-bit Quantization mechanism, int8 fma (Fused Multiply & Add) is needed. In order to reuse fp32 PE and PEarray module, we are going to make int8 fma which has similar i/o configuration to fp32 fma.

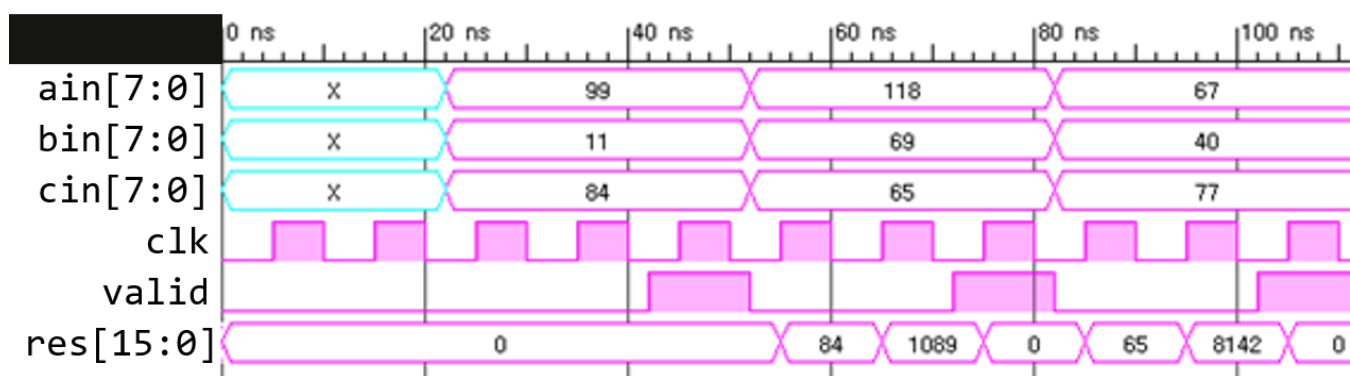
The followings show the I/O configuration and waveform of an **UNKNOWN** int8 multiply-adder. INT8 fma will consist of this unknown INT8 multiply-adder and used in parent module like fp32 PE & PE array instead of fp32 fma. In order to verify the function of this unknown INT8 multiply-adder, we ran behavioral simulation and found that we need to modify the Verilog code in order to ensure the desired behavior.

- Module I/O:

```
assign a_val = valid ? ain : 8'b0;
assign b_val = valid ? bin : 8'b0;
assign c_val = valid ? cin : 8'b0;
```

```
unknown_int UUT_int8(
    .CLK(clk),
    .CE(1'b1),
    .SCLR(rst),
    .A(a_val),
    .B(b_val),
    .C(c_val),
    .P(res)
);
```

- int8 multiply-adder waveform



(valid signal on simulation is given from test environment)

(Pay attention to the input data, input valid and **RESULT** data value)

(Hint. $99 \times 11 + 84 = 1173$)

(Hint. $99 \times 11 = 1089$)

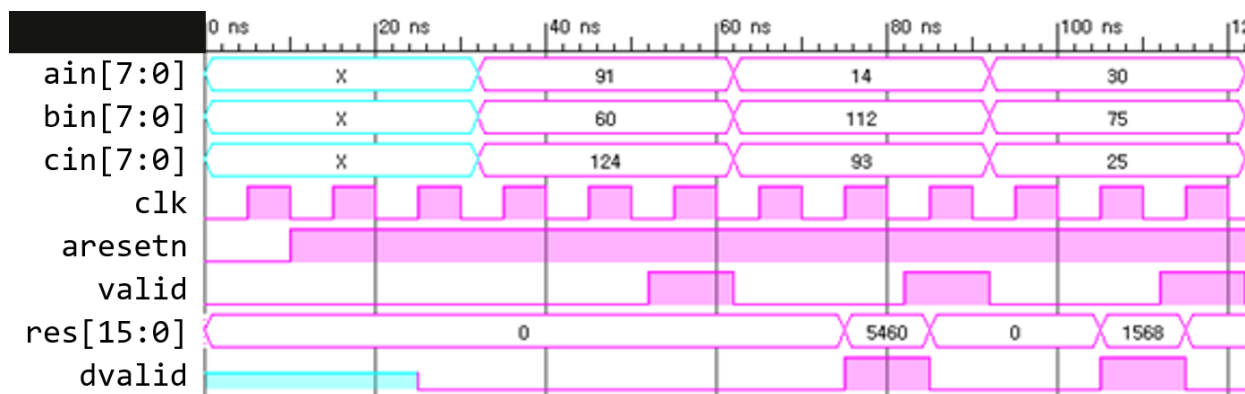
Problem 2-1. INT8 Fused Multiply-Adder (4 points)

Design INT8 fused multiply-adder with the following format by exploiting the above “unknown int8” module.

- **Module I/O:**

```
fma_int8 U_INT8_FMA(
    .clk(),
    .aresetn(),
    .ain(),
    .bin(),
    .cin(),
    .valid(),
    .res(),
    .dvalid()
);
```

- **Simulation Result :**



(Hint. $91 \cdot 60 + 124 = 5584$)

(Hint. $91 \cdot 60 = 5460$)

- **Code**

```
module fma_int8(
    input [8-1:0] ain, bin, cin,
    input clk, valid, aresetn,
    output [16-1:0] res,
    output dvalid
);
    wire [8-1:0] a_val, b_val, c_val;
```

```
assign a_val = valid ? ain : 8'b0;
```

```
unknown_int UUT_int8(  
    .CLK(clk),  
    .CE(1'b1),  
    .SCLR(!aresetn),  
    .A( a_val ),  
    .B(          ),  
    .C(          ),  
    .P(          )  
);  
endmodule
```

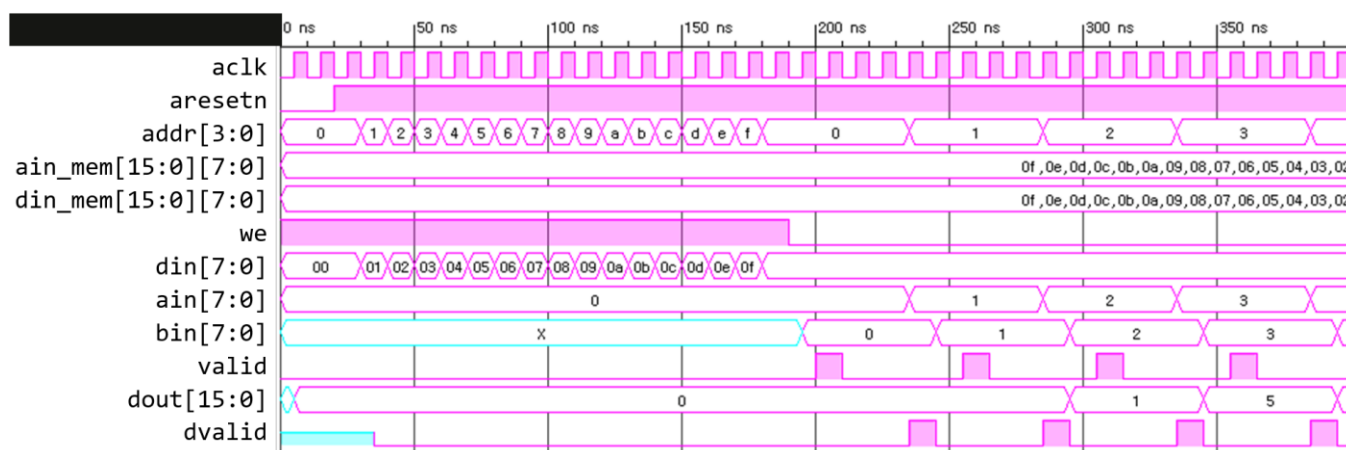
Problem 2-2. INT8 Processing Element (6 points)

Design INT8 PE with the following format by exploiting the above “fma_int8” module.

- Module I/O:

```
my_pe_int8 #(
  4
) PE_int8 (
  .aclk(),
  .aresetn(),
  .ain(),
  .din(),
  .addr(),
  .we(),
  .valid(),
  .dvalid(),
  .dout()
);
```

- Simulation Result :



- Code

```
module my_pe_int8 #(
  parameter L_RAM_SIZE = 6
)
(
  // clk/reset
  input aclk,
  input aresetn,
  // port A
  input [7:0] ain,
  // peram -> port B
  input [7:0] din,
  input [L_RAM_SIZE-1:0] addr,
  input we,
  input valid,
  output reg dvalid,
  output reg [15:0] dout
);
```

```

// peram: PE's local RAM -> Port B
reg [7:0] bin;
(* ram_style = "block" *) reg [7:0] peram [0:2**L_RAM_SIZE - 1];

always @(posedge aclk)
    if (we)
        peram[addr] <= din;
    else
        bin <= peram[addr];

always @(posedge aclk)
    if(!aresetn) begin

end

else begin

end

end

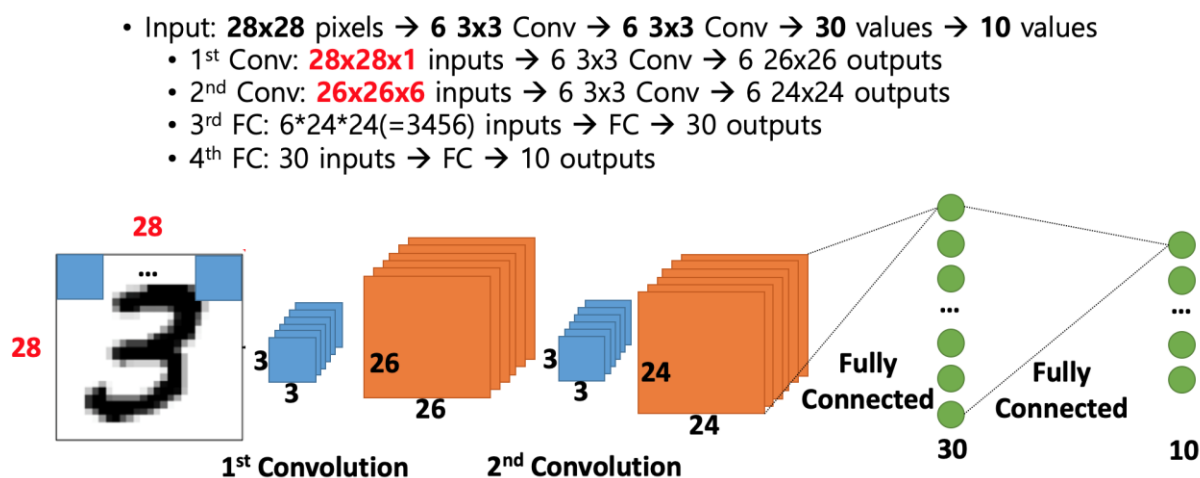
fma_int8 U_INT8_FMA(
    .clk(aclk),
    .aresetn(aresetn),
    .ain(          ),
    .bin(          ),
    .cin(          ),
    .valid(        ),
    .res(          ),
    .dvalid(       )

);
endmodule

```

Q3. (Neural Network Operations)

Here is an example of neural network that classifies handwritten digits. As shown below, the network consists of two convolution layers and two fully connected layers. Note that other details of each operation follow our previous practices.



Problem 3-1 (4 points)

Calculate the total number of block operations where $M=16$, $V=16$.

Problem 3-2 (6 points)

Propose three optimization methods to accelerate the execution of the network on FPGA, and explain why each method improves the inference time (maximum three sentences).

Q4. (Controller Simulation, FSM)

See the provided code of `pearray.v`.

Problem 4-1. (3 points)

Complete the waveform on answer sheet 4.1.

Problem 4-2. (1 points)

What is the function simulated in Problem 4-1 (Answer sheet 4.1)?

Problem 4-3. (1 points)

Where should each of M and V be stored, `pe_global_ram` or the `pe_(local_)ram`? Explain the benefit of such data placement in terms of memory utilization and latency.

Problem 4-4. (3 points)

Complete the waveform on answer sheet 4.4.

Problem 4-5. Deal with Zero data (2 points)

Due to the weight pruning and ReLU activation, a large amount of zero data can be obtained in the matrix or vector. Since the multiplication with zero input is meaningless, it is recommended to skip zero-input computation as much as possible. Therefore, we want to skip the redundant cycle dealing with zero data in LOAD (Problem 4.1) state and CALC (Problem 4.4) state.

Problem 4-5-1. (0.5 points)

Where should M and V be stored, `pe_global_ram` or the `pe_(local_)ram` in the above case?

Problem 4-5-2 (1.5 points)

If 40% of the matrix elements are zero, 20% of the vector elements are zero, what is the expected speedup for each state? (How much redundant cycle can you reduce? Think after dividing the entire operation by V LOAD, M LOAD and CALC. **hint:** `pe_global_ram` and `pe_(local_)ram` are constant in size, regardless of zero ratio.)

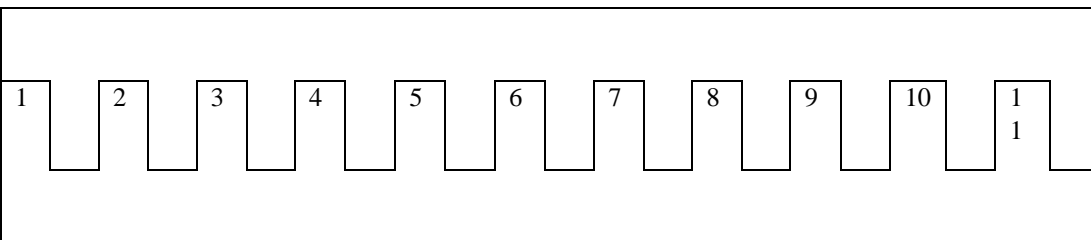
Q5. (BRAM modelling)

Our M*V accelerator uses Xilinx BRAM modules. In order to verify how BRAM module works, we designed a vector write/read example. In this example, we write 4 sample data on BRAM and read 4 written data from BRAM.

Input sample data)

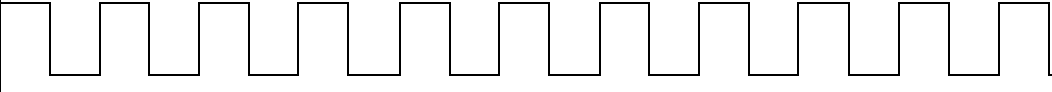
Time	1	2	3	4	5	6	7	8
Type	WR	WR	WR	WR	RD	RD	RD	RD
ADDR	0x30	0x34	0x38	0x3C	0x30	0x34	0x38	0x3C
DIN	0x1234	0x5678	0x9ABC	0xDEFO	0x0808	0x0808	0x0808	0x0808

After writing sample data with WE, we assert ADDR to read written data from BRAM. When ADDR is given on Xilinx BRAM module, DOUT appears on @posedge CLK

CLK											
ADDR	0000	0030	0034	0038	003C	0030	0034	0038	003C	003C	003C
DIN	0000	1234	5678	9ABC	DEFO	0808	0808	0808	0808	0808	0808
DOUT	0000	0000	0000	5678	9ABC	DEFO	0808	5678	9ABC	DEFO	DEFO
WE	0	F	F	F	F	0	0	0	0	0	0

Problem 5-1. Is the verification result correct? Explain the verification result. (4 points)

Suppose we use new CLK, which is a 180 degree inversion of the original clock. With the new inverted clock, DOUT appears on @**negedge CLK**.

CLK												
ADDR	0000	0030	0034	0038	003C	0030	0034	0038	003C	003C	003C	
DIN	0000	1234	5678	9ABC	DEF0	0808	0808	0808	0808	0808	0808	
DOUT		0000	0000	1234	5678	9ABC	DEF0	1234	5678	9ABC	DEF0	
WE	0	F	F	F	F	0	0	0	0	0	0	

Problem 5-2. Is the verification result with the inverted clock correct? Explain the verification result. Explain why the result is different from the above example in Problem 5-1. (6 points)