하드웨어 시스템 설계 5주차 실습 보고서

2017-12751 컴퓨터공학부 이동학

Goal: Implement BRAM model, Processing Element & test bench according to scenario.

Code:

My_bram.v

```
module my_bram #(
    parameter integer BRAM_ADDR_WIDTH = 15,
    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;
```

```
reg [31:0] readREG[1:0];
reg readREG_V[1:0];
reg [BRAM_ADDR_WIDTH-3:0] addrREG;
reg [31:0] writeREG;
reg [3:0] writeREG_V;
wire [31:0] mask;

 initial begin
     if (INIT_FILE != "") begin
         $readmemh(INIT_FILE, mem);
     end
     wait (done) begin
         $writememh(OUT_FILE, mem);
     end
 end

 assign mask = {{8{writeREG_V[0]}}, {8{writeREG_V[1]}},{8{writeREG_V[2]}}, {8{writeREG_V[3]}}};
 always @(posedge BRAM_CLK) begin
     if(BRAM_RST == 1) begin
         BRAM_RDDATA <= 0;
     end
     else begin
         if(readREG_V[1] == 1) begin
             BRAM_RDDATA = readREG[1];
         end
```

```
            mem[addrREG] = (mem[addrREG] & (~mask)) | (writeREG & mask);


            readREG_V[1] = readREG_V[0];
            readREG[1] = readREG[0];
            readREG_V[0] = BRAM_EN & (!BRAM_WE);
            readREG[0] = mem[addr];
            writeREG_V = {4{BRAM_EN}} & BRAM_WE;
            addrREG = addr;
            writeREG = BRAM_WRDATA;
        end
    end
endmodule
```

tb_bram.v

```
module tb_bram(

    );
    reg BRAM_CLK;
    parameter BRAM_ADDR_WIDTH = 15;
    reg [BRAM_ADDR_WIDTH-1:0] BRAM_ADDR;
    wire [31:0] BRAM_RDDATA1;
    wire [31:0] BRAM_RDDATA2;
    reg BRAM_EN1;
    reg BRAM_EN2;
    reg BRAM_RST;
    reg [3:0] BRAM_WE;
```

```
reg done;


integer i;
initial begin
	BRAM_CLK <= 0;
	BRAM_EN1 <= 0;
	BRAM_EN2 <= 0;
	BRAM_RST <= 0;
	done <= 0;
	BRAM_WE <= 0;

	for(i=0;i<8192;i=i+1) begin
		#10;
		BRAM_ADDR <= {i, 2'b00};
		BRAM_EN1 <= 1;
		BRAM_EN2 <= 0;
		#10;
		BRAM_EN1 <= 0;
		#20;
		BRAM_EN2 <= 1;
		BRAM_WE <= 4'b1111;
		#10;
		BRAM_EN2 <= 0;
		BRAM_WE <= 0;
		#10
		BRAM_EN2 <= 1;
```

```
        end
        #10
        #100;
        BRAM_EN2 <= 0;
        done <= 1;
        $finish;
    end

    always #5 BRAM_CLK = ~BRAM_CLK;

    my_bram read_BRAM(
        .BRAM_ADDR(BRAM_ADDR),
        .BRAM_CLK(BRAM_CLK),
        .BRAM_WRDATA(32'b0),
        .BRAM_RDDATA(BRAM_RDDATA1),
        .BRAM_EN(BRAM_EN1),
        .BRAM_RST(BRAM_RST),
        .BRAM_WE(4'b0),
        .done(0)
    );

    my_bram #(BRAM_ADDR_WIDTH, "", "output.txt") write_BRAM(
        .BRAM_ADDR(BRAM_ADDR),
        .BRAM_CLK(BRAM_CLK),
        .BRAM_WRDATA(BRAM_RDDATA1),
        .BRAM_RDDATA(BRAM_RDDATA2),
```

```
        .BRAM_EN(BRAM_EN2),
        .BRAM_RST(BRAM_RST),
        .BRAM_WE(BRAM_WE),
        .done(done)
    );
Endmodule
```

My_pe.v

```
module my_pe #(
    parameter L_RAM_SIZE = 6
    )
    (
    input aclk,
    input aresetn,
    input [31:0] ain,
    input [31:0] din,
    input [L_RAM_SIZE-1:0] addr,
    input we,
    input valid,
    output dvalid,
    output [31:0] dout
    );

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

    reg [31:0] psum;
```

```
    wire [31:0] buffer;

    always @(posedge aclk) begin
        if(we == 1) peram[addr] <= din;
        if(~aresetn) psum <= 0;
        if(dvalid == 1) psum = buffer;
    end

    assign dout = (dvalid == 1) ? psum : 32'b0;

    floating_point_0 mac(
        .aclk(aclk),
        .aresetn(aresetn),
        .s_axis_a_tdata(ain),
        .s_axis_a_tvalid(valid),
        .s_axis_b_tdata(peram[addr]),
        .s_axis_b_tvalid(valid),
        .s_axis_c_tdata(psum),
        .s_axis_c_tvalid(valid),
        .m_axis_result_tdata(buffer),
        .m_axis_result_tvalid(dvalid)
    );
Endmodule
```

Tb_pe.v

```
module tb_pe(
```

```
);
parameter L_RAM_SIZE = 6;

reg aclk;
reg aresetn;
reg [31:0] ain;
reg [31:0] din;
reg [31:0] mem [15:0];

reg [L_RAM_SIZE - 1:0] addr;
reg we;
reg valid;
wire dvalid;
wire [31:0] dout;

integer i;

initial begin
	aclk <= 0;
	aresetn <= 0;
	we <= 1;
	valid <= 0;
	$readmemh("din.txt", mem);
	#10;
	for(i = 0; i < 16; i = i + 1) begin
```

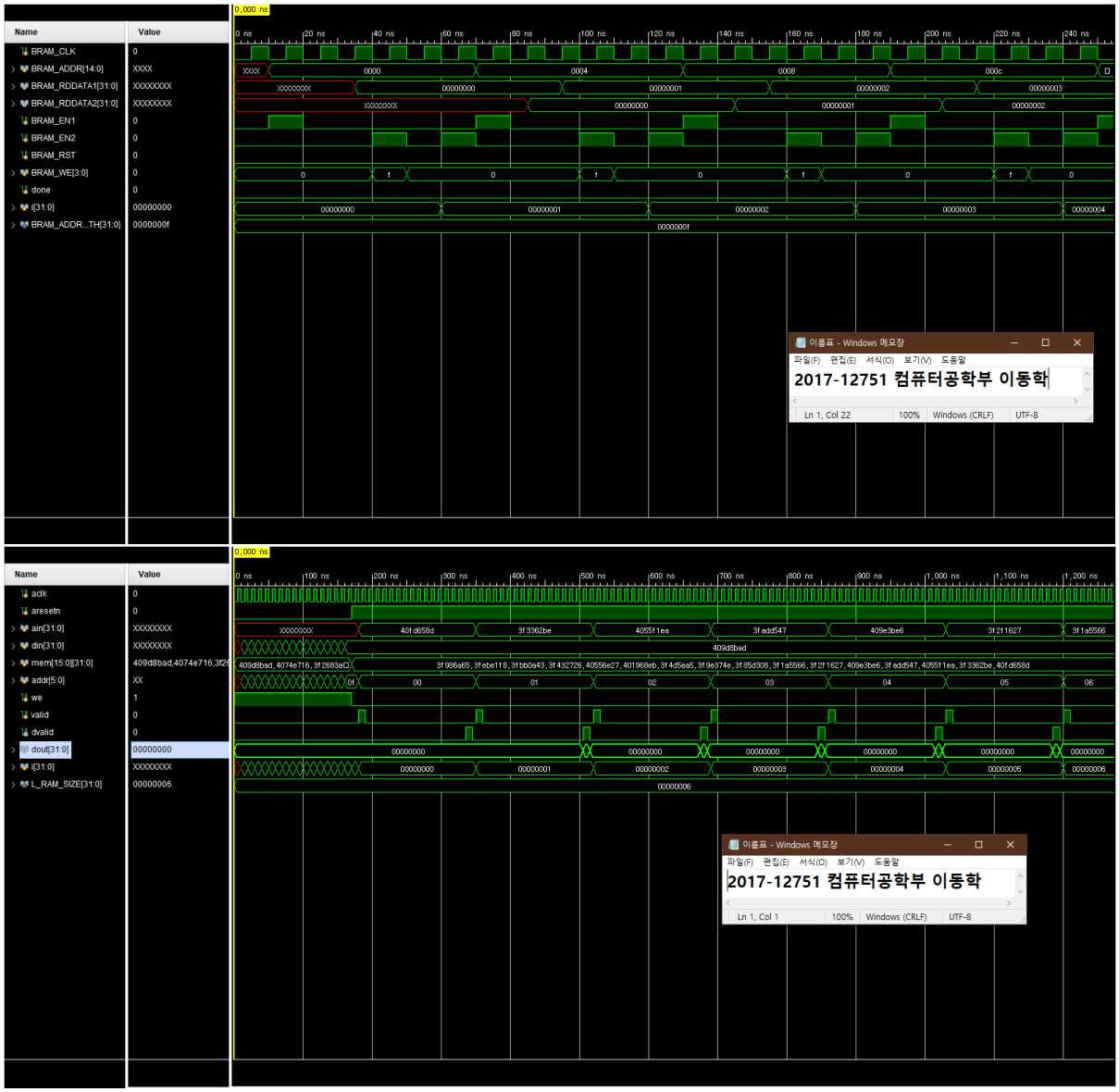
```
            addr <= i;
            din <= mem[i];
            #10;
        end
        $readmemh("ain.txt", mem);
        we <= 0;
        aresetn <= 1;
        #10;
        for(i = 0; i < 16; i = i + 1) begin
            addr <= i;
            ain <= mem[i];
            valid <= 1;
            #10;
            valid <= 0;
            #160;
        end
        $finish;
    end

    always #5 aclk = ~aclk;

    my_pe PE(
    .aclk(aclk),
    .aresetn(aresetn),
    .ain(ain),
    .din(din),
```

```
        .addr(addr),
        .we(we),
        .valid(valid),
        .dvalid(dvalid),
        .dout(dout)
);
endmodule
```

Result: my_bram / my_pe



Discussion: my_bram 의 경우 read BRAM 은 input.txt 에서 값을 불러와 initialize 하였고, write BRAM 은 read BRAM 의 모든 값을 복사하도록 하였습니다. Write 에 1cycle, read 에 2cycle 이 사용되도록 read register 2 개와 write register 1 개를 선언하고 posedge clk 마다 한칸씩 밀리도록 구현하였습니다. 모든 과정이 끝난 후엔 output.txt 에 write BRAM 에 저장된 값을 출력하도록 하였습니다.

My_pe의 경우 처음엔 din.txt에서 16개의 값을 불러와 local register의 알맞은 주소에 저장하고, 그 이후 ain.txt에서 하나씩 값을 읽어서 IP catalog를 이용해 MAC operation을 수행하고, 중간 결과를 저장하여 다음 연산에 사용하도록 하였습니다.