Hardware System Design 4190.309A
Seoul National University

Practice #5. PE implementation & BRAM modeling
Jiwon Lee, Sangjun Son

Goal

- Implement BRAM model & test bench according to scenarios.
  - Make test bench that instantiates two BRAMs and initialize one BRAM to store address as data.
  - Copy every data from the initialized BRAM to the other BRAM.
- Implement PE with floating point fused multiply adder.

1 Implementation

이번 프로젝트는 Block Random Access Memory와 Processing Element를 각각 구현함으로써 추후 구현할 Matrix-Matrix Multiplication을 수행하기 위한 기본 모듈을 구성하는 것을 목적으로 한다. 아래는 코드 구현과 함께 간략한 아이디어 및 기능에 대한 설명이 (1) BRAM, (2) PE 순으로 진행된다.

1.1 Block Random Access Memory, BRAM

BRAM의 경우 크게 두 부분으로 이뤄져 있다. (1) 모듈이 실행이 되기 전 INIT_FILE에서 내부 메모리 `mem`를 읽는 부분과 done 신호가 주어졌을 때, OUT_FILE에 `mem`의 상태를 출력하는 부분과, (2) EN, RST, WE 신호가 들어왔을 때 경우에 따라 입력되는 데이터를 `mem`에 읽고 쓰는 역할을 한다.

- 아래에 첨부된 코드 중 21-28 라인의 외부 파일 입출력에 관한 구현 보면, `initial` 구문을 이용하여 모듈의 생성과 동시에 파일 입출력에 대한 실행 구문에 대한 scope를 지정한다. `$readmemh` 로 시작함으로써 INIT_FILE 파일을 읽어 `mem`에 저장한다. 그 후 `done` 신호 들어올 때 까지 대기하다가 신호가 들어오면 `$writememh` 함수를 사용해 OUT_FILE에 `mem` 데이터를 저장한다. 이 때 함수에 붙어있는 `$readmem`**h**와 `$writemem`**h**의 **h**는 hexadecimal로 파일에 저장하는 값을 16진수 형태로 저장하는 옵션을 의미한다.

- hello

MY_BRAM

```
1  `timescale 1ns / 1ps
2  module my_bram #(
3      parameter integer BRAM_ADDR_WIDTH = 15,
4      parameter INIT_FILE = "input.txt",
5      parameter OUT_FILE = "output.txt"
6  )(
7      input wire [BRAM_ADDR_WIDTH-1:0] BRAM_ADDR,
8      input wire BRAM_CLK,
9      input wire [31:0] BRAM_WRDATA,
10     output reg [31:0] BRAM_RDDATA,
11     input wire BRAM_EN,
12     input wire BRAM_RST,
13     input wire [3:0] BRAM_WE,
14     input wire done
15 );
16     reg [31:0] mem[0:8191];
17     wire [BRAM_ADDR_WIDTH-3:0] addr = BRAM_ADDR[BRAM_ADDR_WIDTH-1:2];
18     reg [31:0] dout, wdout;
19     integer rflag = 0, wflag = 0;
20
21     initial begin
22         if (INIT_FILE != "") begin
23             $readmemh(INIT_FILE, mem);
24         end
25         wait (done) begin
26             $writememh(OUT_FILE, mem);
```

```
        end
    end

    always @(posedge BRAM_CLK or posedge BRAM_RST) begin
        if (BRAM_RST) begin
            BRAM_RDDATA <= 0;
        end
        if (BRAM_EN) begin
            if (BRAM_WE) begin
                if (BRAM_WE[0]) mem[addr][7:0] <= BRAM_WRDATA[7:0];
                if (BRAM_WE[1]) mem[addr][15:8] <= BRAM_WRDATA[15:8];
                if (BRAM_WE[2]) mem[addr][23:16] <= BRAM_WRDATA[23:16];
                if (BRAM_WE[3]) mem[addr][31:24] <= BRAM_WRDATA[31:24];
            end
            else begin
                dout <= mem[addr];
                BRAM_RDDATA <= dout;
            end
        end
    end
endmodule
```

1.2 Integer fused multiply-adder

Floating point multiply-adder와 동일하게, 미리 구현된 IP catalog를 사용, testbench 에서 입력값을 조정하여 출력값을 확인하였다. IP catalog를 customize 할 때, Multiply와 Add가 동시에 수행되어 결과값을 나타내야 했기 때문에, P-A:B latency 와 P-C latency를 0으로 설정하였다.

IP catalog에는 입력값에 clk signal이 존재하지 않았다. 따라서, async한 방식으로 A,B,C (inputs)의 값이 변화할 때, P (output)이 계산되는 것을 볼 수 있었다. 또한, P값이 나오기까지는 delay가 존재하였는데, 이는 주어진 A, B, C값을 최종 결과값으로 나타내기까지 걸리는 시간 (latency)이다.

1.3 Processing Element, PE

MY_PE

```
`timescale 1ns / 1ps
module my_add #(
    parameter BITWIDTH = 32
)
(
    input [BITWIDTH-1:0] ain,
    input [BITWIDTH-1:0] bin,
    output [BITWIDTH-1:0] dout,
    output overflow
);
    // concatnate (overflow, dout) & detect overflow
    assign {overflow, dout} = ain + bin;
endmodule
```

상기된 모듈은 Lab03 adder 모듈이다. ain과 bin 길이의 입력값이

2 Result

2.1 Block Random Access Memory, BRAM

TB_MY_BRAM

아래의 코드는 BRAM의 구현의 Validity를 확인하기 위해 시나리오에 맞게 구현한 것이다.

```
`timescale 1ns / 1ps

module tb_my_bram #(
    parameter integer BRAM_ADDR_WIDTH = 15,
```

```
    parameter INIT_FILE = "input.txt"
)();
    reg [31:0] BRAM_INIT[0:8191];
    reg [BRAM_ADDR_WIDTH-1:0] BRAM_ADDR;
    reg BRAM_CLK;
    reg BRAM_RST;
    reg [3:0] BRAM_WE;
    reg done;
    wire [31:0] BRAM_WRDATA1, BRAM_RDDATA1;
    wire [31:0] BRAM_WRDATA2, BRAM_RDDATA2;
    integer i;

    initial begin
        BRAM_ADDR <= 0;
        BRAM_CLK <= 1;
        BRAM_RST <= 0;
        BRAM_WE <= 0;
        done <= 0;
        for (i = 0; i < 8192; i = i + 1) begin
            BRAM_INIT[i][31:0] <= i;
        end
        #10 $writememh(INIT_FILE, BRAM_INIT);

        for (i = 0; i <= 8192; i = i + 1) begin
            BRAM_ADDR <= i << 2; #20;
            BRAM_WE <= 4'b1111; #10;
            BRAM_WE <= 0; #30;
        end
        done <= 1'b1; #30;
        BRAM_RST <= 1'b1;
    end

    always #5 BRAM_CLK = ~BRAM_CLK;
    assign BRAM_WRDATA2 = BRAM_RDDATA1;

    my_bram MY_BRAM1 (
        .BRAM_ADDR(BRAM_ADDR),
        .BRAM_CLK(BRAM_CLK),
        .BRAM_WRDATA(BRAM_WRDATA1),
        .BRAM_RDDATA(BRAM_RDDATA1),
        .BRAM_EN(1'b1),
        .BRAM_RST(BRAM_RST),
        .BRAM_WE(0),
        .done(0)
    );
    my_bram #(.INIT_FILE("")) MY_BRAM2 (
        .BRAM_ADDR(BRAM_ADDR),
        .BRAM_CLK(BRAM_CLK),
        .BRAM_WRDATA(BRAM_WRDATA2),
        .BRAM_RDDATA(BRAM_RDDATA2),
        .BRAM_EN(1'b1),
        .BRAM_RST(BRAM_RST),
        .BRAM_WE(BRAM_WE),
        .done(done)
    );
endmodule
```

위 Testbench 코드를 수행하면 아래와 같은 Waveform을 확인할 수 있다.

2.2 Processing Element, PE

3 Conclusion

입력되는 범위의 값들은 uniformly random하게 선택한 값들을 가지고 시뮬레이션 결과를 살펴보았을 때, 연산 결과 값이 실제 값과 일치하는 것을 확인할 수 있었다. Array를 이용하여 generate-for statement에서 간편하게 원하는 값을 얻어낸다는 점이 이번 프로젝트 구현 중 핵심 아이디어였다.

Practice #5. PE implementation & BRAM modeling

Jiwon Lee, Sangjun Son

cmd의 값에 따라 출력값을 변화시키기 위해 always @(cmd) 구문을 사용하지 않고, 삼항연산자를 사용할 때에도 원하는 형태로 waveform이 출력되었는데, 삼항연산자 자체도 always와 비슷한 역할로 event가 발생하면 출력값을 변화시키는 역할을 확인하였다. 향후 프로젝트에서 IP catalog를 이용해 기존의 모듈을 효율적으로 활용하여 코드를 작성하는 방법을 익힐 수 있었으며, custom modularization 과정을 익힐 수 있었다.

Hardware System Design 4190.309A
Seoul National University

Practice #5. PE implementation & BRAM modeling
Jiwon Lee, Sangjun Son