Practice 5

PE implementation & BRAM modeling

2017-19428 서준원

1. Introduction

이번 랩은 RAM을 모델링하고, Final Project인 Matrix Multiplication IP를 만들기 위해 필수적인 Processing Element를 만드는 실습이었다. Matrix Multiplication은 Floating Point의 Fused Multiplier를 사용하게 되며, 메모리로부터 데이터를 저장한 후, 저장된 값들을 하나씩 읽어 곱하고 누적하여 더해주는 과정으로 Matrix Multiplication을 작동하게 된다.

구현한 모듈을을 테스트하기 위해 시뮬레이션을 했다. 값을 정확한 타이밍에 넣어주고, Latency동안 다른 노이즈 값들이 들어가지 않도록 신경써주는 작업이 아주 중요했다.

Bram 인터페이스는 데이터를 베릴로그의 레지스터에 메모리 형태로 저장해놓은 후 주소를 입력으로 받아 값을 저장하거나 저장된 값을 읽는 동작을 하도록 구현했다. 메모리는 본질적으로 Sequential circuit으로 구현되기 때문에 Flip-flop을 사용해야 한다. 클락 사이클을 기준으로 엣지 컨트롤을 통해 원하는 작업을 했다.

이번 랩에서는 메모리를 시뮬레이션하기 위해 텍스트파일로부터 데이터를 읽고 그 데이터를 다른 메모리 모듈에다가 그대로 저장한 후 텍스트파일로 저장하는 작업을 진행하였다.

PE(Processing Element)를 위한 시뮬레이션은 다음과 같은 과정을 거쳐 진행된다. 첫째로 데이터를 읽은 후 이를 로컬 레지스터에 저장한다. 이는 Matrix Multiplication에서 input 벡터에 해당한다. 그리고 Matrix에 해당하는 데이터들이 들어오면 이를 MAC IP를 사용하여 계산을 한다. 중간 계산 결과는 레지스터에 저장했다가, 모든 계산이 끝나면 값을 반환한다. 여기서 데이터의 길이는 16으로 지정했다.

Fused Multiplier를 Non-blocking으로 사용했기 때문에 중간에 이상한 값이 들어가지 않도록 하고, 끝날 때까지 16사이클을 기다려주는 것이 아주 중요했다.

2. Implementation

(1) BRAM

BRAM의 기본적인 구현은 템플릿 코드를 따랐으며, 핵심 부는 아래와 같다.

```
always @(posedge BRAM_CLK or posedge BRAM_RST) begin
 if(BRAM_RST) /*asyncronous resest*/ BRAM_RDDATA <= 0;
 else if(BRAM_EN) begin
     if(BRAM_WE == 4'b0000) BRAM_RDDATA <= mem[addr];
     else begin
         if(BRAM_WE[0] == 1) mem[addr][8*(0+1)-1:8*0] <= BRAM_WRDATA[8*(0+1)-1:8*0];
         if(BRAM_WE[1] == 1) mem[addr][8*(1+1)-1:8*1] <= BRAM_WRDATA[8*(1+1)-1:8*1];
         if(BRAM_WE[2] == 1) mem[addr][8*(2+1)-1:8*2] <= BRAM_WRDATA[8*(2+1)-1:8*2];
         if(BRAM_WE[3] == 1) mem[addr][8*(3+1)-1:8*3] <= BRAM_WRDATA[8*(3+1)-1:8*3];
         end
     end
```

Edge Control을 위해 clk과 reset의 변화를 확인했고, 그 때마다 옵션인 BRAM_WE의 값에 따라 적당한 값을 읽어주거나 저장해주는 작업을 했다.

(2) PE

```
always @(posedge aclk or negedge aresetn) begin
    if(!aresetn) begin
        psum = 0;
        weight = 0;
    end
    if(we) peram[addr] = din;
    else weight = peram[addr];
end

always @(cout) begin
    psum = dvalid ? cout : 0;
end

assign dout = psum; // psum(output value register -> wire)

// fused multiplier for MAC :: res = a * b + c

floating_point_0 mul(
    .aclk(aclk),
    .aresetn(aresetn),
    .s_axis_a_tvalid(valid),
    .s_axis_a_tdata(ain),
    .s_axis_b_tvalid(valid),
    .s_axis_b_tdata(weight),
    .s_axis_c_tvalid(valid),
    .s_axis_c_tdata(psum),
    .m_axis_result_tvalid(dvalid),
    .m_axis_result_tdata(cout)
);
```

PE는 IP를 사용하여 아래와 같이 간단하게 구현하였다.

Dvalid 여부에 따라 값을 누적해서 더한 값을 결과값으로 리턴해줄지, 아니면 0으로 세팅해줄지를 결정하였다. 또한 엣지 컨트롤을 통해 write enable signal이 있을 때와 없을 때를 구분해서 작업을 진행했다.

We = 1일 때는 값을 내부 레지스터에 저장했고, 0일 때는 MAC 작업을 수행할 수 있도록, 내부 레지스터에 저장된 값을 weight란 변수에 각 사이클마다 넣어주는 작업을 했다.

3. Result

시뮬레이션을 위해 다음과 같이 테스트 벤치를 작성하였다.

(1) BRAM

```
initial
begin
        BRAM_CLK <= 0;
        BRAM_RST <= 0;
        BRAM_EN <= 1;
        done <= 0;
        #4

        for(i=0; i<8192; i=i+1) begin
            BRAM_ADDR = i * 4;
            #4; // wait 5 clock cycle for visualizing
        end
        done = 1;
    end

    always #1 BRAM_CLK = ~BRAM_CLK;
```

my_bram #(.BRAM_ADDR_WIDTH(15), .INIT_FILE(INIT_FILE), .OUT_FILE("")) READ (BRAM_ADDR, BRAM_CLK, BRAM_WRDATA, BRAM_RDDATA, BRAM_EN, BRAM_RST, 4'b0000, done);

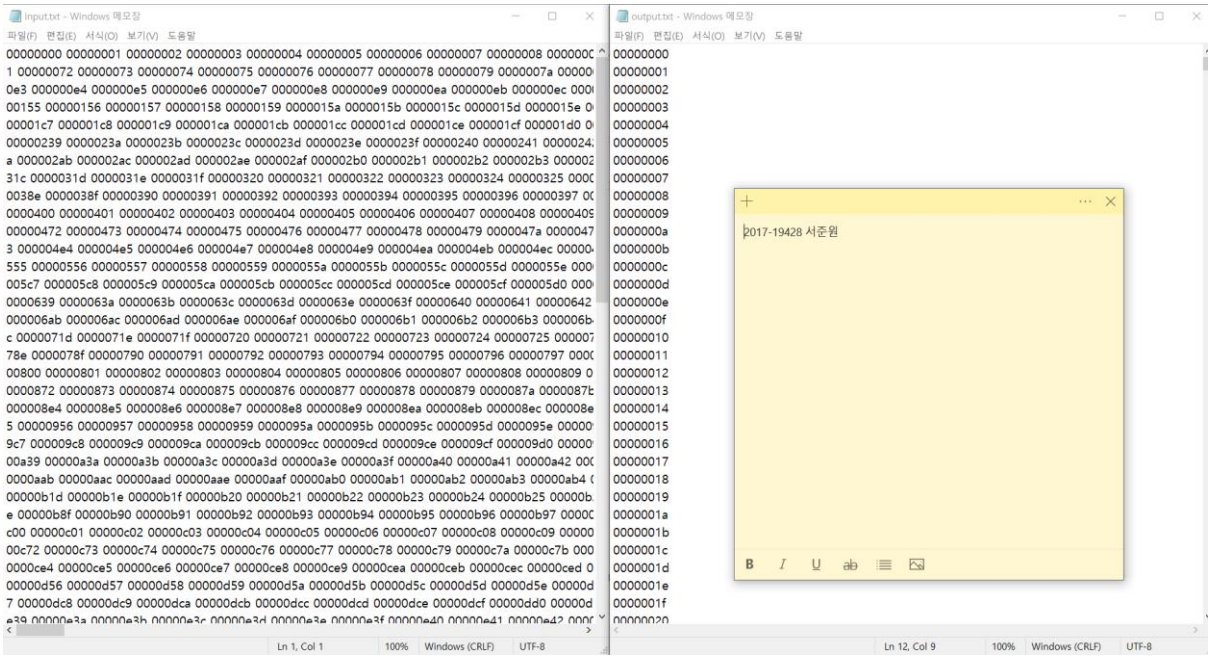
my_bram #(.BRAM_ADDR_WIDTH(15), .INIT_FILE(""), .OUT_FILE("output.txt")) WRITE (BRAM_ADDR, BRAM_CLK, BRAM_RDDATA, read, BRAM_EN, BRAM_RST, 4'b1111, done);

두 개의 BRAM을 설정하여 하나는 input.txt를 읽어 와이어에 연결해주었고, 또다른 모듈의 input에 그 와이어를 연결하여 write하도록 코드를 작성하였다. 결과를 output.txt에 저장하였다.

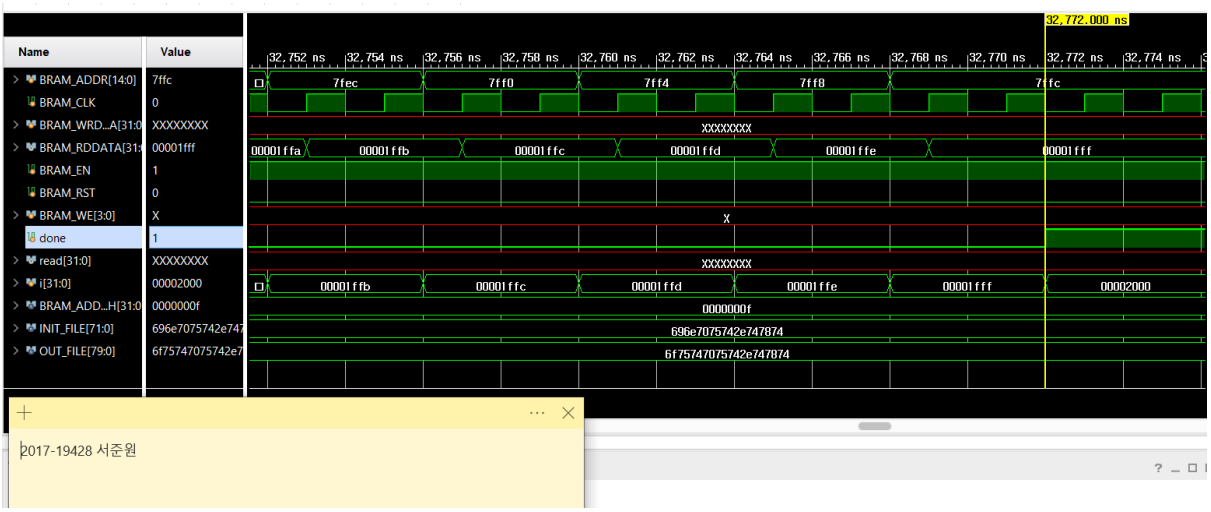
Input.txt는 다음과 같은 c 코드로 작성하였다. 0부터 8191까지를 작성하였다.

```c
#include<stdio.h>

int main(){
        FILE *fp = fopen("input.txt", "wb");

        for(int i=0; i<8192; i++){
                fwrite(&i, sizeof(int), 1, fp);
        }

        fclose(fp);

        FILE *fc = fopen("input.txt", "rb");

        int num;
        for(int i=0; i<8192; i++){
                fread(&num, sizeof(int), 1, fc);
                printf("%d ",num);
        }
        return 0;
}
```

결과로 생성된 output.txt 파일은 다음과 같이 생겼다. 값이 제대로 옮겨진 것을 확인할 수 있다. (0부터 8192까지)



시뮬레이션의 waveform은 다음과 같다.



주소를 할당한 다음 사이클에 값이 읽어지며, 8191까지 값을 작성한 후에 done 시그널이 켜지게 된다.

(2) PE

PE는 non-blocking fused multiplier를 사용했기 때문에 중간에 쓰레기 값이 들어가지 않도록 테스트벤치를 설계할 때 버퍼를 잘 설정해주었다. 다음과 같이 설계했다.

```
    initial begin
        aclk <= 1;
        aresetn = 0;
        valid = 0;
        #10;

// Load matrix to local register
        aresetn = 1; // negative reset
        we = 1;

        for(i=0; i<16; i=i+1) begin
            din = $urandom%(2**31);
            addr = i;
            #10; // TODO :: delay control !
        end
// MAC
        valid = 1;
        we = 0;
        for(i=0; i<16; i=i+1) begin
            ain = $urandom%(2**31);
            addr = i;
            #160; // TODO :: delay control for MAC -> use valid signal?
        end
    end
```

Clock Cycle을 10으로 맞추고, 16 Cycle동안 계산이 실행되므로 랜덤 인풋을 넣어준 후 16사이클을 기다렸다. 값을 메모리에 넣어주는 것은 한 데이터 당 10의 사이클을 기다렸다.

그 후로는 연속해서 dvalid이 1로 켜지게 된다. Waveform은 다음과 같다. 새로운 인풋이 들어가는 주기와 값이 계산되는 사이클이 일치하며, 그 때 dvalid가 1로 세팅된다.