

Practice #6. BRAM to PE controller
Jiwon Lee, Sangjun Son

Goal

- Implement PE controller based on PE made in Practice #5.
 - PE controller consisted of PE and FSM
 - Inner product made with MAC operation of PE
- FSM controls PE to calculate inner product with several states.
(e.g., $\text{data1}[0] * \text{data2}[0] + \text{data1}[1] * \text{data2}[1] + \dots + \text{data1}[15] * \text{data2}[15]$)

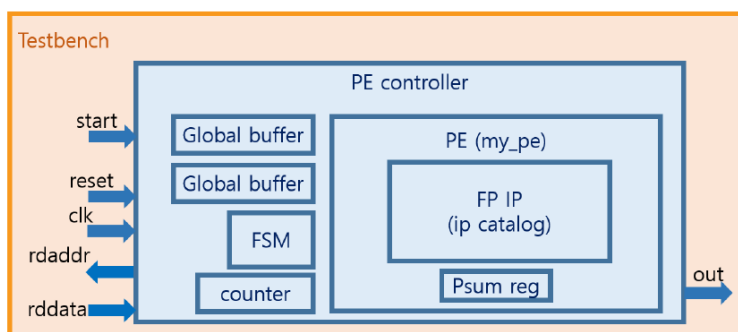


Figure 1: PE와 FSM이 결합된 2개의 벡터의 내적을 연산하는 PE Controller module이다.

1 Implementation

이번 프로젝트는 지난 프로젝트에서 구현한 Processing Element와 이를 실행을 시키기 위해 사용되었던 테스트 벤치 코드를 결합한 새로운 모듈을 구현하는 것을 목적으로 한다. PE Control 모듈은 추후 구현할 Matrix-Matrix Multiplication을 수행하기 위해 존재하는 하위 모듈이다. 아래는 코드 구현과 함께 아이디어 및 기능에 대한 설명이 진행된다.

1.1 PE Controller

Figure 1을 참고하면 Practice #5에서 구현한 벡터의 내적을 계산할 수 있는 my_pe 모듈과 그것을 구성하는 Floating-point fused multiplier IP catalog가 존재한다. PE controller에서는 내적을 연산할 벡터의 모음을 Global buffer에 저장한 뒤 PE에 전달, 연산을 수행하게 된다. 상기된 Sequential logic 모듈이 상황에 따라 어떤 기능을 수행할 지 다르게 명시해주어야 하기 때문에 State를 정의하여 FSM의 논리를 모두 구현해준다.

PE controller는 IDLE, LOAD, CALC, DONE이라는 총 4가지 state를 가진다. IDLE은 대기 상태를 의미하며 이 때 start 신호가 입력되면 LOAD state로 전이된다. 이 상태에서는 rdaddr 주소를 Testbench에 전달하여 rddata 값을 하나씩 입력받아 Global buffer에 저장한다.

모든 LOAD 과정이 마치면 CALC 단계에서 FP IP catalog를 수행하게 된다. Global buffer에 저장된 16개의 실수 벡터 2개를 추출하여 Multiply-Accumulator를 이용해 내적 값을 구하게 된다. 연산이 모두 끝난 후 state는 DONE으로 넘어가게 되고 5 사이클의 Latency를 가지고 done signal과 함께 결과 값을 반환한다. 5 사이클 이후에는 다시 state를 IDLE 상태로 돌려 놓고 start 신호를 기다린다.

Practice #6. BRAM to PE controller
Jiwon Lee, Sangjun Son

- 아래에 첨부된 코드 중 3-30 라인은 PE controller의 변수 선언에 관한 내용이다.
 - parameter에 포함된 (1) L_RAM_SIZE는 입력되는 벡터의 크기를 표현하는 상수이고 (2) BITWIDTH는 연산을 하기 위한 실수 자료형의 크기를 나타낸다. 나머지 입력과 출력에 관한 Wire는 모두 Figure 1에서 표기한 바와 같다.
 - FSM과 관련된 변수는 현재 상태와 앞으로 업데이트해야하는 상태 변수 present_state와 next_state가 있고 그 외에 Counter가 있다. 상태를 나타내는 상수는 총 네 가지로 0 ~ 3의 값을 S_IDLE, S_LOAD, S_CALC과 S_DONE에 해당하도록 설정하였다.
 - 내장 모듈인 MY_PE의 입출력 변수 ain, bin, valid, dout, dvalid를 함께 선언하고 지정시켜주어야 한다.
- 32-23 라인과 42-78 라인은 강의시간에 배운 FSM의 format을 그대로 차용하여 구현하였다. 초기 상태를 reset 신호와 함께 IDLE 상태로 초기화 한다. 현재 상태에서 어떤 조건이 충족되었을 때 다음 상태를 지정해주는 논리를 구현해준다.
 - IDLE 상태에서 start 신호가 들어올 경우 LOAD로 전환. (44 라인)
 - LOAD 상태에서 테스트 벤치에서 모든 데이터 (cnt_load 갯수 만큼의 데이터)를 전달받아 연산에 쓰일 값들이 Global buffer에 전부 저장이 되었을 경우 CALC로 전환. (45 라인)
 - CALC 상태에서 내장 모듈 MY_PE에서 2개의 벡터의 크기만큼의 연산을 모두 마쳤을 경우 (cnt_calc 갯수 만큼의 Fused multiplication 수행) DONE로 전환. (46 라인)
 - DONE 상태에서 Latency 만큼 딜레이가 진행된 후에 (Clock이 cnt_done 만큼 경과된 후) IDLE로 전환. (47 라인)
- 50-60라인은 각 상태에 위치했을 때 어떤 Counter를 활성화 시킬지 지정해주는 부분이다. 62-65라인에서는 현재 상태가 LOAD에 이르렀을 때 테스트벤치로 rdaddr을 전달해주고 해당하는 실수형 자료 rddata를 받아와 Global buffer에 저장한다.

MY_PE_CONTROLLER

```
1 `timescale 1ns / 1ps
2 module my_pe_controller #(
3     parameter L_RAM_SIZE = 6,
4     parameter BITWIDTH = 32
5 ) (
6     input start,
7     input reset,
8     input clk,
9     output [L_RAM_SIZE:0] rdaddr,
10    input [BITWIDTH-1:0] rddata,
11    output [BITWIDTH-1:0] out,
12    output done
13 );
14 parameter DONE_LATENCY = 5;
15 parameter S_IDLE = 2'd0, S_LOAD = 2'd1, S_CALC = 2'd2, S_DONE = 2'd3;
16 reg [1:0] present_state, next_state;
17
18 reg [L_RAM_SIZE:0] cnt_load, cnt_calc;
19 reg [2:0] cnt_done;
20 reg rst_cnt_load, rst_cnt_calc, rst_cnt_done;
21
22 reg [BITWIDTH-1:0] gb1[0:2**L_RAM_SIZE-1];
23 reg [BITWIDTH-1:0] gb2[0:2**L_RAM_SIZE-1];
24
25 reg [BITWIDTH-1:0] ain;
26 reg [BITWIDTH-1:0] bin;
27 reg valid = 0;
28
29 wire [BITWIDTH-1:0] dout;
30 wire dvalid;
31
32 always @(posedge clk or posedge reset)
```

Practice #6. BRAM to PE controller
Jiwon Lee, Sangjun Son

```
33     if (reset) present_state <= S_IDLE; else present_state <= next_state;
34
35     always @(posedge clk or posedge rst_cnt_load)
36         if (rst_cnt_load) cnt_load <= 0; else cnt_load <= cnt_load + 1;
37     always @(posedge clk or posedge rst_cnt_calc)
38         if (rst_cnt_calc) cnt_calc <= 0;
39     always @(posedge clk or posedge rst_cnt_done)
40         if (rst_cnt_done) cnt_done <= 0; else cnt_done <= cnt_done + 1;
41
42     always @(*)
43         case (present_state)
44             S_IDLE: if (start) next_state = S_LOAD; else next_state = present_state;
45             S_LOAD: if (cnt_load == 2**(L_RAM_SIZE+1)-1) next_state = S_CALC; else next_state = present_state;
46             S_CALC: if (cnt_calc == 2**L_RAM_SIZE) next_state = S_DONE; else next_state = present_state;
47             S_DONE: if (cnt_done == DONE_LATENCY-1) next_state = S_IDLE; else next_state = present_state;
48         endcase
49
50     always @(*)
51         case (present_state)
52             S_LOAD: rst_cnt_load <= 0;
53             S_CALC: rst_cnt_calc <= 0;
54             S_DONE: rst_cnt_done <= 0;
55             default: begin
56                 rst_cnt_load <= 1;
57                 rst_cnt_calc <= 1;
58                 rst_cnt_done <= 1;
59             end
60         endcase
61
62     always @(rddata or present_state)
63         if (present_state == S_LOAD) begin
64             if (cnt_load < 2**L_RAM_SIZE) gb1[cnt_load] = rddata; else gb2[cnt_load-2**L_RAM_SIZE] = rddata;
65         end
66
67     always @(dvalid or present_state)
68         if (present_state == S_CALC) begin
69             if (dvalid) begin
70                 cnt_calc <= cnt_calc + 1;
71                 valid <= 0;
72             end
73             else begin
74                 ain <= gb1[cnt_calc];
75                 bin <= gb2[cnt_calc];
76                 valid <= 1;
77             end
78         end
79
80     assign rdaddr = present_state == S_LOAD ? cnt_load : 0;
81     assign out = present_state == S_DONE ? dout : 0;
82     assign done = present_state == S_DONE ? 1 : 0;
83
84     my_pe #(L_RAM_SIZE, BITWIDTH) MY_PE(
85         .acclk(clk),
86         .aresetn(~reset),
87         .ain(ain),
88         .bin(bin),
89         .valid(valid),
90         .dvalid(dvalid),
91         .dout(dout)
92     );
93 endmodule
```

코드의 하단부에는 지난 실습에서 구현한 Floating point MAC모듈의 호출 부분이다. Parameter에 들어가는 값은 모두 변수명과 동일하게 매칭을 해주었다 [?].

Practice #6. BRAM to PE controller
Jiwon Lee, Sangjun Son

2 Result

2.1 Testbench Implementation

아래의 코드는 MY_PE_CONTROLLER의 구현의 Validity를 확인하기 위해 검증 시나리오를 설정하고 그에 맞게 구현한 것이다. MY_CONTROLLER 인스턴스를 입력 벡터의 크기, $2^{L_RAM_SIZE} = 2^4 = 16$ 와 저장되는 실수의 자료형 크기, $BITWIDTH = 32$ 를 사용하여 선언하였다.

- 모든 테스트를 시작하기 전에 input.txt를 초기화하기 위한 과정을 거친다. 주소에 해당하는 인덱스를 값으로 가질 수 있도록 for문을 통해 대입시킨 후 \$writememh를 통해 파일에 저장을 한다 (23-24 라인).
- 테스트벤치에서 BRAM.EN 신호는 항상 활성화 하고 (45, 55 라인) BRAM_RST와 done 신호는 모든 데이터 전송이 끝나고 입력과 출력이 완료되었을 때 True를 대입할 것이다 (33-34 라인).
- BRAM_ADDR의 경우 i번째 entry의 주소값은 BRAM에서 2개의 LSB를 사용하지 않으므로 주소값 또한 4의 배수로 증가시켜 대입해 주어야 한다. BRAM_WE 신호를 주소값이 유지되는 한 구간을 5 CLK 사이클과 1 CLK 사이클로 나누어 DISABLE과 ENABLE을 번갈아 대입해준다 (29-31 라인).

TB MY_PE_CONTROLLER

```
1  `timescale 1ns / 1ps
2  module tb_my_pe_controller #(
3      parameter L_RAM_SIZE = 4,
4      parameter BITWIDTH = 32
5  ) ();
6      reg [BITWIDTH-1:0] gb[0:2*(L_RAM_SIZE+1)-1];
7      reg [BITWIDTH-1:0] rddata;
8      wire [BITWIDTH-1:0] out;
9      wire [L_RAM_SIZE:0] rdaddr;
10     wire done;
11
12     reg start, clk, reset;
13     integer i;
14
15     initial begin
16         for(i = 0; i < 2*L_RAM_SIZE; i = i+1) begin
17             gb[i] = $urandom_range(2**30, 2**30+2**24);
18             gb[2*L_RAM_SIZE + i] = $urandom_range(2**30, 2**30+2**24);
19         end
20
21         clk <= 0;
22         start <= 0;
23         reset <= 1; #20;
24
25         start <= 1;
26         reset <= 0; #20;
27     end
28
29     always #5 clk = ~clk;
30     always @(rdaddr) begin
31         rddata = gb[rdaddr];
32     end
33
34     my_pe_controller #(L_RAM_SIZE, BITWIDTH) MY_PE_CONTROLLER(
35         .start(start),
36         .reset(reset),
37         .clk(clk),
38         .rdaddr(rdaddr),
39         .rddata(rddata),
40         .out(out),
41         .done(done)
```

Practice #6. BRAM to PE controller
Jiwon Lee, Sangjun Son

```
42     );  
43 endmodule
```

위 Testbench 코드를 수행하면 아래의 Figure 2와 같은 Waveform을 확인할 수 있다. 결과를 자세히 보면 mem에 해당하는 BRAM_ADDR이 4씩 증가하는 것을 확인할 수 있고 이에 따라 BRAM_RDDATA1 또한 BRAM_WE가 비활성화 되어 있을 때 2 cycle을 delay로 읽게 되는 것을 확인할 수 있다. BRAM_RDDATA1가 곧 BRAM2의 BRAM_WRDATA2이므로 같은 Waveform을 관찰하였다.

2.2 Simulation Results

Section 2.1에서 구현된 테스트 벤치를 실행하면 아래와 같은 Waveform을 확인할 수 있다.

2.3 Design Implementation

3 Conclusion

이후 프로젝트에서 어떤 모듈을 구현해야 하는지 Bottom-up으로 구현하다 보니 무슨 기능을 위한 구현인지는 아직 잘 모르겠지만 반대로 이전 lab 세션에서 구현을 진행한 모듈에 대해서는 연계성을 확인할 수 있었다. 지금 구현한 모듈이 앞으로 쓰일 수 있기 때문에 가독성을 높이면서 최대한 임의 구현 방식을 최대한 피하기 위해 노력하였다.

MY_BRAM 모듈을 구현하면서 WE signal에 따라 mem에 저장하는 statement를 for-generate로 구현해보려 했으나 이런 저런 오류가 나면서 나열형 방식으로 구현해 코드의 효율성이 떨어진다는 나름의 판단을 하였다. 추후 프로젝트를 진행하기 전에 always 구문 안에서 block assignment를 for-generate로 구현하는 방식을 익혀야겠다는 필요성을 제고하였다.

MY_PE 모듈 자체의 구현은 그리 어렵지 않았지만, IP catalog의 floating point MAC모듈의 내부 구조를 모르기 때문에, testbench를 작성하는데 어려움을 겪었는데, 이는 valid bit의 high 설정 후 dvalid bit가 high가 될 때까지의 정확한 clock cycle delay, valid bit와 input parameter가 동시에 할당되었을 때의 비정상적인 출력이 그리 직관적이지는 않았기 때문이라고 생각한다. MY_PE 모듈도 이후 구현에서 중요한 부분을 차지하기 때문에, 최대한 이번 과제에만 국한되지 않게 구현하려 하였다.

Hardware System Design 4190.309A Seoul National University

Practice #6. BRAM to PE controller Jiwon Lee, Sangjun Son



Figure 2: Waveform

Hardware System Design 4190.309A
Seoul National University

Practice #6. BRAM to PE controller
Jiwon Lee, Sangjun Son



Figure 3: Waveform

Practice #6. BRAM to PE controller
Jiwon Lee, Sangjun Son

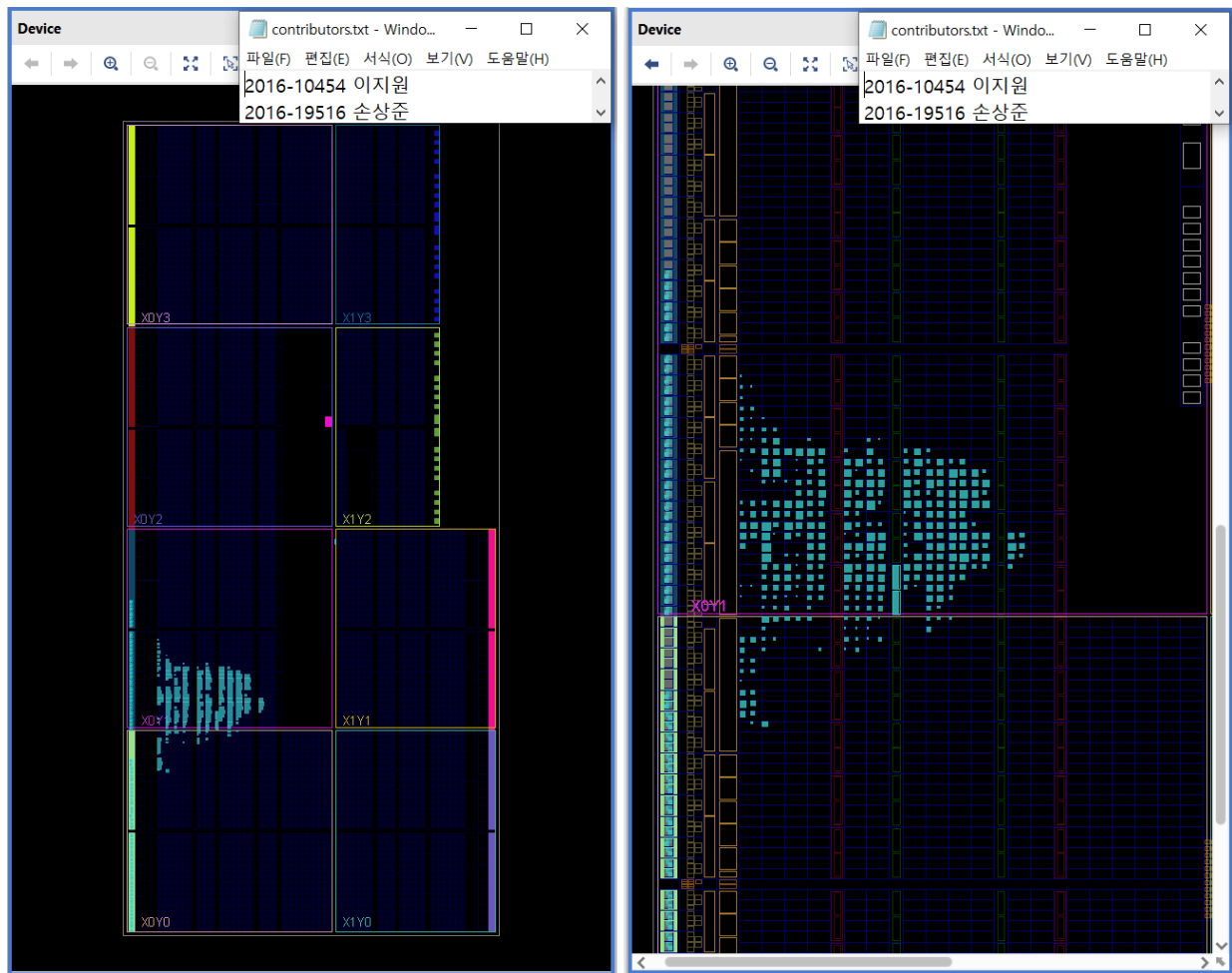


Figure 4: Implementation design