

HARDWARE DESIGN SYSTEM

Term Project V0

이다운 2016-13919

1.

Term project의 목표는 지금까지 LAB에서 구현한 pe_array Verilog코드를 ZED보드의 hardware accelerator에 올려 deep learning의 행렬 곱연산을 하는 것이다. 이를 위해서 Bram과 데이터 교환을 하는 인터페이스를 구현하고 Custom IP를 만들어야 한다.

2. CODE

중요 코드는 크게 3가지로 이루어져 있다. myip_v1_0_S00_AXI, my_pearray, pe_con 이 각각 그 코드다. myip_v1_0_S00_AXI은 PS, BRAM, IP가 서로 소통하는 인터페이스라고 생각하면 된다. 그래서 이 모듈 하위에 my_pearray를 구현하였다.

```
wire start;
wire done;

reg[C_S_AXI_DATA_WIDTH-1:0] slv_reg0_d;
always@( posedge S_AXI_ACLK )
    if(S_AXI_ARESETN ==1'b0)
        slv_reg0_d <= 32'd0;
    else
        slv_reg0_d <= slv_reg0;
assign start = (slv_reg0 == 32'h5555 && slv_reg0_d == 32'd0);

reg done_d;
always@( posedge S_AXI_ACLK )
    if(S_AXI_ARESETN ==1'b0)
        done_d <= 1'b0;
    else
        done_d <= done;

assign run_complete = (done && ~done_d);

my_pearray #(
    .MSIZE(MSIZE),
    .VSIZE(VSIZE)
)(
    .start(start),
    .done(done),
    .S_AXI_ACLK(S_AXI_ACLK),
    .S_AXI_ARESETN(S_AXI_ARESETN),

    .BRAM_ADDR(BRAM_ADDR),
    .BRAM_WRDATA(BRAM_WRDATA),
    .BRAM_WE(BRAM_WE),
    .BRAM_CLK(BRAM_CLK),
    .BRAM_RDDATA(BRAM_RDDATA)
);
```

코드는 위와 같다. slv_reg0에 0x5555라는 신호가 들어오면 start이 한 사이클동안 1이되고 이 1의 값이 my_pearray에 전달되어 연산을 시작시키는 것이다. 그리고 my_pearray에서 연산이 끝났다는 의미로 done이 1이 출력되면 run_complete가 1이되어 전체 연산이 종료되었음을 외부에 알린다.

my_pearray는 pe_array연산을 하는 pe_control와 bram을 이어주는 인터페이스 모듈이다.
 그래서 아래와 같이 input, output은 bram과 data 전송을 위한 포트이다.

```

module my_pearray#(
    parameter integer BRAM_ADDR_WIDTH = 32,
    parameter integer BRAM_DATA_WIDTH = 32,
    parameter integer BRAM_WE_WIDTH = 4,
    parameter MSIZE = 4,
    parameter VSIZE = 4
)
(
    input start,
    output done,
    input S_AXI_ACLK,
    input S_AXI_ARESETN,

    output [BRAM_ADDR_WIDTH-1:0] BRAM_ADDR,
    output [BRAM_DATA_WIDTH-1:0] BRAM_WRDATA,
    output [BRAM_WE_WIDTH-1:0] BRAM_WE,
    output BRAM_CLK,
    input [BRAM_DATA_WIDTH-1:0] BRAM_RDDATA,

```

그리고 start 신호가 들어오면 delay를 고려하여 pe_control에도 start 신호를 보내고 bram에서 부터 data 값을 read하기 시작한다.

```

always@(posedge S_AXI_ACLK)begin
    if(reset==1)begin
        counter <=0;
        size <= 0;
        pe_start2 <=0;
    end

always@(posedge S_AXI_ACLK)begin

    if(reset==1)begin
        startload <=0;
        pe_start1 <=0;
    end
    if(start==1)begin
        startload <=1;
        pe_start1<=1;
    end
    if(pe_start1 ==1) pe_start1 <= 0;

    if(size == sizeparam) startload <= 0;

end

    if(startload ==1)begin
        counter <= counter + 32'd4;
        size <= size + 32'd1;
        if(pe_start1==1) pe_start2 <= 1; else pe_start2 <= 0;
    end
    else begin
        counter <= 0;
        size <= 0;
    end
end
end

```

BRAM에 Address값을 보낼때 가장 하위 2bit는 무시되므로 counter를 1씩 더하는게 아니라 4씩더해 주었다. 그리고 read값은 바로 안오고 delay가 있으므로 이를 고려하여 buffer 변수를 활용하였다.

```

module myip_v1_0_S00_AXI #
(
    // Users to add parameters here
    parameter integer BRAM_ADDR_WIDTH = 32,
    parameter integer BRAM_DATA_WIDTH = 32,
    parameter integer BRAM_WE_WIDTH = 4,
    // User parameters ends
    // Do not modify the parameters beyond this line

    // Width of S_AXI data bus
    parameter integer C_S_AXI_DATA_WIDTH = 32,
    // Width of S_AXI address bus
    parameter integer C_S_AXI_ADDR_WIDTH = 4,
    parameter MSIZE = 6,
    parameter VSIZE = 6
)
(

```

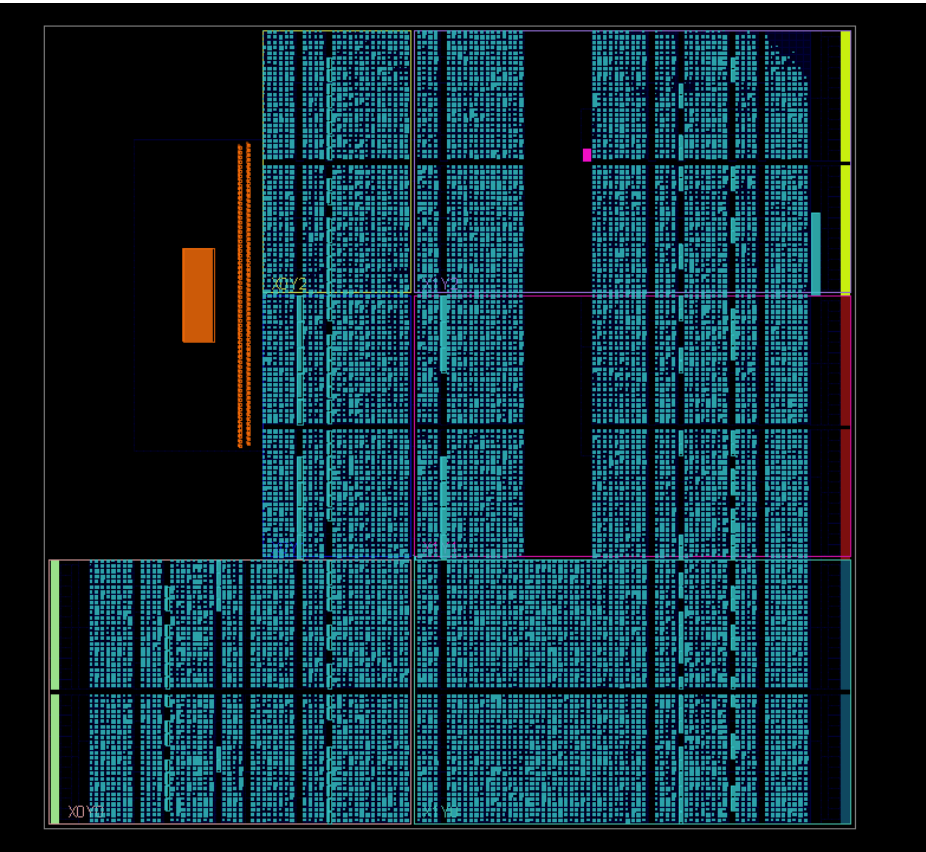
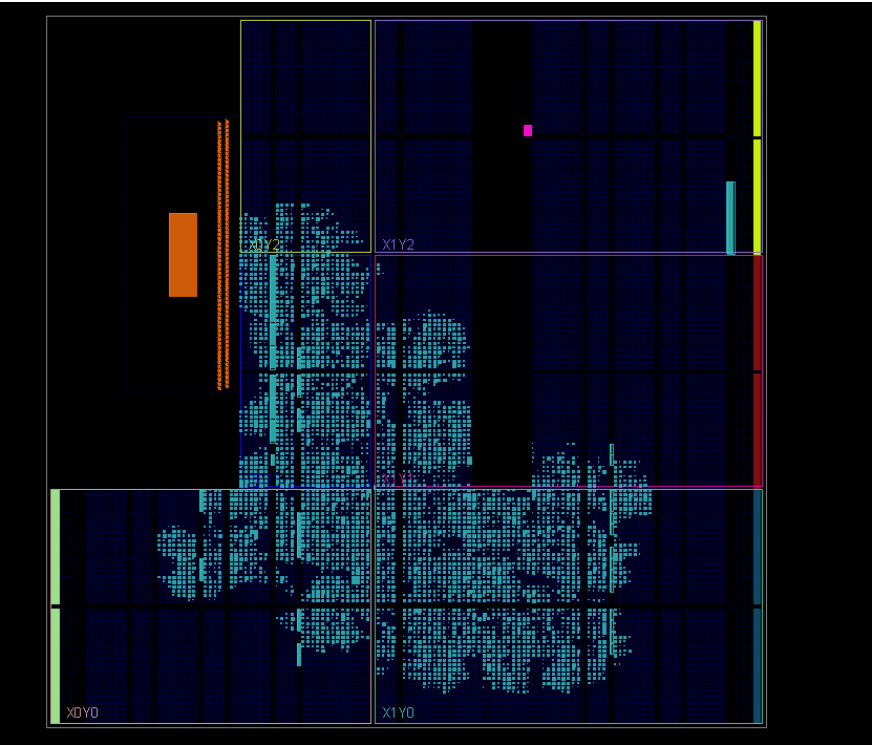
MxV의 크기가 달라질 수 있음을 고려하여 MSIZE와 VSIZE를 parameter로 만들어 이 크기에 따라 코드가 동작하도록 구현하였다. 즉 이 parameter가 4라면 2^4 크기라는 의미이다. 이런 식으로 크기를 가변적으로 구현하여 여러 벡터 사이즈로 모듈을 테스트를 할 수 있다.

3. 실행 방법

모듈 작동은 benchmark를 실행하여 확인하였다. 이미지 가중치를 다운로드 한 뒤 benchmark.sh 파일에서 M,V 사이즈를 원하는 대로 바꾼 뒤 `sudo bash benchmark.sh`로 실행하였다. 이를 위한 zynq.bit 파일이 3개 첨부되어 있는데 zynq.bit 는 벡터 사이즈가 16일때를 위한 bit파일이고 zynq-32.bit, zynq-64.bit은 각각 32, 64일때를 위한 파일이다. 자신이 원하는 사이즈에 맞춰 sd카드에서 파일을 교체해 주었다.

4. 결과

구현한 디자인을 Run implementation하여 보드에 올라간 것을 확인하면 아래와 같다. 첫번째는 벡터 사이즈가 16일 경우이고 두번째는 64일때다. 16x16일 때 대략 40%정도 올라가는데 64x64 사이즈일 경우에는 거의 보드에 꽉 차게 들어 감을 확인할 수 있다.



아래 사진은 차례대로 16x16, 32x32, 64x64 크기 일 때의 benchmark 결과이다.

```
[*] Arguments: Namespace(m_size=16, network='mlp', num_test_images=100, run_type='cpu', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 9375,
 'm_size': 16,
 'total_image': 100,
 'total_time': 6.515059000000008,
 'v_size': 16}
=> Accuracy should be 0.97

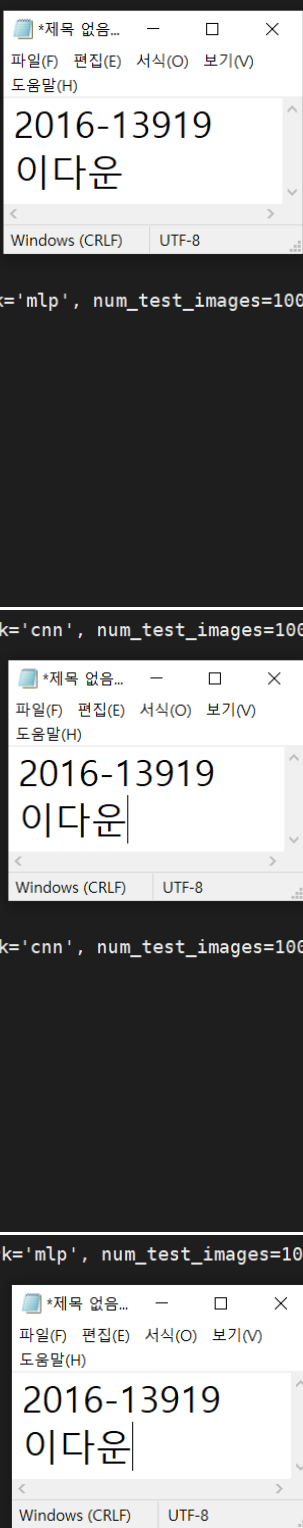
[*] Arguments: Namespace(m_size=16, network='mlp', num_test_images=100, run_type='fpga', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 9375,
 'm_size': 16,
 'total_image': 100,
 'total_time': 49.738203999999996,
 'v_size': 16}
=> Accuracy should be 0.97

[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=100, run_type='cpu', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 1186,
 'm_size': 16,
 'total_image': 100,
 'total_time': 0.74274100000000236,
 'v_size': 16}
=> Accuracy should be 1.0

[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=100, run_type='fpga', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 1186,
 'm_size': 16,
 'total_image': 100,
 'total_time': 5.0290200000000003,
 'v_size': 16}
=> Accuracy should be 1.0

[*] Arguments: Namespace(m_size=32, network='mlp', num_test_images=100, run_type='cpu', v_size=32)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 2432,
 'm_size': 32,
 'total_image': 100,
 'total_time': 6.3460010000000001,
 'v_size': 32}
=> Accuracy should be 0.97

[*] Arguments: Namespace(m_size=32, network='mlp', num_test_images=100, run_type='fpga', v_size=32)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 2432,
 'm_size': 32,
 'total_image': 100,
 'total_time': 45.407182000000006,
 'v_size': 32}
=> Accuracy should be 0.97
```



The image displays three screenshots of a terminal window, each showing the output of a benchmark script for different network configurations and image sizes. The terminal output includes arguments, statistics, and accuracy results. Overlaid on each screenshot is a small window showing the handwritten digit '2016-13919' and the Korean text '이다운' (Idoun).

First Screenshot (m_size=16, network='mlp', run_type='cpu', v_size=16):

```
[*] Arguments: Namespace(m_size=16, network='mlp', num_test_images=100, run_type='cpu', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 9375,
 'm_size': 16,
 'total_image': 100,
 'total_time': 6.515059000000008,
 'v_size': 16}
=> Accuracy should be 0.97
```

Second Screenshot (m_size=16, network='mlp', run_type='fpga', v_size=16):

```
[*] Arguments: Namespace(m_size=16, network='mlp', num_test_images=100, run_type='fpga', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 9375,
 'm_size': 16,
 'total_image': 100,
 'total_time': 49.738203999999996,
 'v_size': 16}
=> Accuracy should be 0.97
```

Third Screenshot (m_size=16, network='cnn', run_type='cpu', v_size=16):

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=100, run_type='cpu', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 1186,
 'm_size': 16,
 'total_image': 100,
 'total_time': 0.74274100000000236,
 'v_size': 16}
=> Accuracy should be 1.0
```

Fourth Screenshot (m_size=16, network='cnn', run_type='fpga', v_size=16):

```
[*] Arguments: Namespace(m_size=16, network='cnn', num_test_images=100, run_type='fpga', v_size=16)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 1186,
 'm_size': 16,
 'total_image': 100,
 'total_time': 5.0290200000000003,
 'v_size': 16}
=> Accuracy should be 1.0
```

Fifth Screenshot (m_size=32, network='mlp', run_type='cpu', v_size=32):

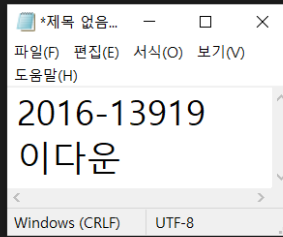
```
[*] Arguments: Namespace(m_size=32, network='mlp', num_test_images=100, run_type='cpu', v_size=32)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 2432,
 'm_size': 32,
 'total_image': 100,
 'total_time': 6.3460010000000001,
 'v_size': 32}
=> Accuracy should be 0.97
```

Sixth Screenshot (m_size=32, network='mlp', run_type='fpga', v_size=32):

```
[*] Arguments: Namespace(m_size=32, network='mlp', num_test_images=100, run_type='fpga', v_size=32)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 2432,
 'm_size': 32,
 'total_image': 100,
 'total_time': 45.407182000000006,
 'v_size': 32}
=> Accuracy should be 0.97
```

```
[*] Arguments: Namespace(m_size=32, network='cnn', num_test_images=100, run_type='cpu', v_size=32)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 804,
 'm_size': 32,
 'total_image': 100,
 'total_time': 1.4287990000000264,
 'v_size': 32}

=> Accuracy should be 1.0
```

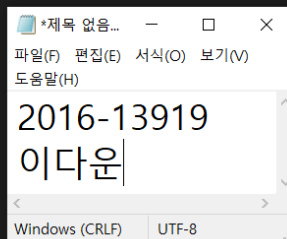


```
[*] Arguments: Namespace(m_size=32, network='cnn', num_test_images=100, run_type='fpga', v_size=32)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 804,
 'm_size': 32,
 'total_image': 100,
 'total_time': 9.314254000000005,
 'v_size': 32}

=> Accuracy should be 1.0
zed@debian-zynq:~/hsd_v0$
```

```
[sudo] password for zed:
[*] Arguments: Namespace(m_size=64, network='mlp', num_test_images=100, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.97,
 'avg_num_call': 627,
 'm_size': 64,
 'total_image': 100,
 'total_time': 6.7929659999999785,
 'v_size': 64}

=> Accuracy should be 0.97
```

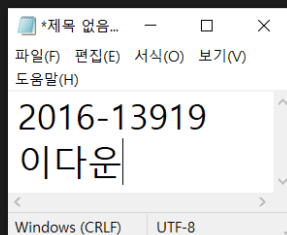


```
[*] Arguments: Namespace(m_size=64, network='mlp', num_test_images=100, run_type='fpga', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 0.96,
 'avg_num_call': 627,
 'm_size': 64,
 'total_image': 100,
 'total_time': 44.307700000000001,
 'v_size': 64}

=> Accuracy should be 0.97
```

```
[*] Arguments: Namespace(m_size=64, network='cnn', num_test_images=100, run_type='cpu', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 741,
 'm_size': 64,
 'total_image': 100,
 'total_time': 4.3765030000000014,
 'v_size': 64}

=> Accuracy should be 1.0
```



```
[*] Arguments: Namespace(m_size=64, network='cnn', num_test_images=100, run_type='fpga', v_size=64)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 741,
 'm_size': 64,
 'total_image': 100,
 'total_time': 28.601467999999983,
 'v_size': 64}

=> Accuracy should be 1.0
```

일단 cpu, fpga type에 관계없이 단순 MLP연산보다 convolution lowering을 거치는 cnn 연산이 훨씬 빠르다는 것을 알 수 있다. fpga도 잘 작동은 하지만 전체적으로 cpu보다는 확연히 느리다. 그리고 벡터 사이즈가 커질 수록 연산 시간도 길어 짐을 확인할 수 있다. 그래서 16x16 연산이 가장 빨랐다.

5. Discussion

보드에 코드를 올리는데 많은 에로점이 있었다. 첫째로 bram과 clk 사이클 타이밍을 맞추는데 있었고 data transfer를 위한 포트를 맞추는 어려움도 있었다. 이를 해결하기 위해 lab5에서 구현한 my_bram을 zed보드에 있는 board라고 생각하고 구현하여 simulation을 돌려보아 하나씩 고쳐 나갔다.

fpga, cpu 타입 모두 벡터사이즈가 16일때가 연산속도가 가장 빨랐다. 그리고 convolution lowering을 하면 시간을 대폭 줄일 수 있음을 확인하였다. 여기서 더 속도를 줄이기 위해 quantaization을 추가로 구현하고 있다.

이번 term project를 구현하기 위해서 하드웨어시스템설계 수업에서 배운 것을 모두 쏟아 부어야 완성시킬 수 있었다. 단순히 한 분야에 특정되지 않고 하나의 거대한 프로젝트를 완성시킨 것만 같아 지금까지 컴퓨터공학 수업에서 한 project 중에 가장 성취감이 컸다.