Convolutional Neural Network(CNN)

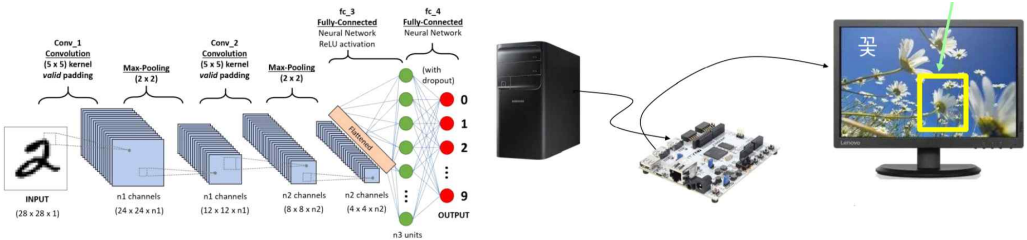
AI RTL

이재성

목차

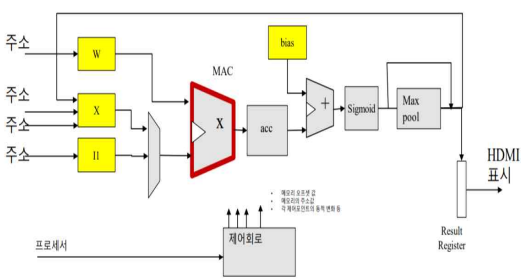
Project 목표

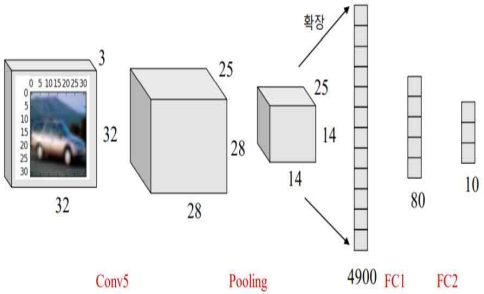
- Verilog HDL로 Convolutional Neural Network 구현
- HDMI로 모니터와 연결후 인식 결과 확인



Project 목표

CNN 연산 구조





MAC 하나만 사용
연산 중 MAC은 무휴로 운용

4개의 32x32x3 이미지, 25개의 5x5x3 weight,
maxpooling, bias, sigmoid, fully-connected 연산

설계 내용

1. 입력 이미지 i1 , weight address value

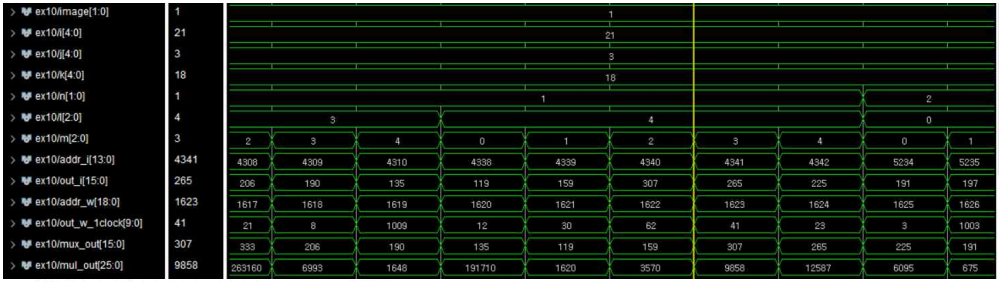
```
always @(posedge clk) begin
    case(s_fsm)
        2'b00: begin s_mux<=1; last<=0; wea<=1; max_pool_on_off<=1; en<=1;//convolution operation

                if(addr_i>12287) addr_i<=0;
                if(m==4) begin
                    m<=0;
                    if(i==4) begin
                        i<=0;
                        if(n==2) begin
                            n<=0;
                            last<=1;
                            if(k==27) begin
                                k<=0;
                                if(j==27) begin
                                    j<=0;
                                    if(i==24) begin
                                        i<=0;
                                        s_fsm<=2'b01;
                                        addr_w<=addr_w+1;
                                        addr_bias<=addr_bias+1;
                                        if(image==3) begin
                                            image<=0;
                                            addr_i<=0;
                                        end else begin
                                        image<=image+1;
                                        addr_i<=addr_i+1;
                                        i<=i+1;
                                        addr_w<=75*i;
                                        addr_i<=addr_i-3071;
                                        addr_bias<=addr_bias+1;
                                        addr_x_Write<=addr_x_Write+1;
                                        wea<=1;
                                        end
                                    end else begin
                                    j<=j+1;
                                    addr_i<=addr_i-21775; addr_w<=75*i; //addr_i<=image*3072+32*j;
                                    end
                                end else begin
                                k<=k+1;
                                addr_i<=addr_i-2179; addr_w<=75*i; //addr_i<=image*3072+32*j+k;
                                end
                            end else begin
                            n<=n+1;
                            addr_i<=addr_i+892; addr_w<=addr_w+1; //addr_i<=image*3072+32*j+k+1024*n;
                            end
                        end else begin
                        i<=i+1;
                        addr_i<=addr_i+28; addr_w<=addr_w+1; //image*3072+32*j+k+1024*n+32*i;
                        end
    end else begin
    m<=m+1;
    addr_i<=addr_i+1; addr_w<=addr_w+1;
    end
```

3개의 state(convolution, fully-connect 1,2)
Image와 weight의 행과 열 값에 따라 address 계산

주소 값 직접 연산
Bias, x memory write 주소 설정

설계 내용

1. 입력 이미지 i1 , weight address value



i1 address = addr_i
첫번째 이미지에서 i=21, j=3, k=18,
n=1, l=4, m=0일때 addr_i=4338

weight address = addr_w
첫번째 이미지에서 i=21, n=1, l=4,
m=0일때 addr_w=1620

설계 내용

2. MAC operation

multiple

```
always @(posedge clk) begin
    out_w_1clock<=out_w;
    mul_out<=mux_out*out_w_1clock;
end
```

mux의 output과 클락을 맞추기 위해 read한 weight값 한클락 지연시킨 후 multiple

accumulate

```
always @(posedge clk) begin
    acc_out<=mul_out+acc_out;
    if(last) acc_out<=0;
end
```

마지막 데이터일때 last=1로 만들어 데이터 초기화

설계 내용

3. bias

```
if(i==24) begin
    i<=0;
    s_fsm<=2'b01;
    addr_w<=addr_w+1;
    addr_bias<=addr_bias+1;
    if(image==3) begin
        image<=0;
        addr_i<=0;
    end else begin
    image<=image+1;
    addr_i<=addr_i+1;
    addr_x_Read<=0;
    end
end else begin
i<=i+1;
addr_w<=75*i;
addr_i<=addr_i-3071;
addr_bias<=addr_bias+1;
```

bias address 값 연산
i값이 바뀔때마다 1씩 증가

```
mem_bias Bias (
  .clka(clk),    // input wire clka
  .addra(addr_bias),  // input wire [8 : 0] addra
  .douta(out_bias)  // output wire [15 : 0] douta
);
```

bias memory read (16bits)

```
(* mark_debug = "true" *) reg [31:0] add_bias;

always @(posedge clk) begin
    add_bias<= out_bias_conv + acc_out;
end
```

Overflow 방지와 sigmoid 연산을 위해
출력인 add_bias는 32비트로 설정

설계 내용

4. sigmoid

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

```
fixed_to_float biasfloat (
  .aclk(clk),                                   // input wire aclk
  .s_axis_a_tvalid(1),                // input wire s_axis_a_tvalid
  .s_axis_a_tdata(add_bias),                  // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(),   // output wire m_axis_result_tvalid
  .m_axis_result_tdata(add_bias_float)     // output wire [31 : 0] m_axis_result_tdata
);
```

bias output floating-point convert

```
floating_exp exponential (
  .aclk(clk),                                   // input wire aclk
  .s_axis_a_tvalid(1),                // input wire s_axis_a_tvalid
  .s_axis_a_tdata(add_bias_float),                  // input wire [31 : 0] s_axis_a_tdata
  .m_axis_result_tvalid(),   // output wire m_axis_result_tvalid
  .m_axis_result_tdata(sigmoid_exp)     // output wire [31 : 0] m_axis_result_tdata
);
```

floating-point exponential

```
floating_div division (
  .aclk(clk),                                   // input wire aclk
  .s_axis_a_tvalid(1),                // input wire s_axis_a_tvalid
  .s_axis_a_tdata(1),                  // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(1),                // input wire s_axis_b_tvalid
  .s_axis_b_tdata(sigmoid_add),                  // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(),   // output wire m_axis_result_tvalid
  .m_axis_result_tdata(sigmoid_out)     // output wire [31 : 0] m_axis_result_tdata
);
```

floating-point division

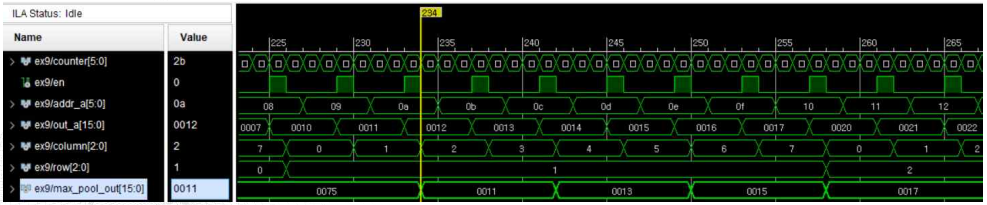
floating-point
ip(exp, add, divide)
사용하여 연산

설계 내용

5. maxpooling

```
always @(posedge clk) begin
    if(max_pool_on_off) begin
        if(en) begin
            column <= column +1;
            reg_array[0]<=sigmoid_out_fixed;
            reg_array[1]<=reg_array[0]; reg_array[2]<=reg_array[1]; reg_array[3]<=reg_array[2];
            reg_array[4]<=reg_array[3]; reg_array[5]<=reg_array[4]; reg_array[6]<=reg_array[5];
            reg_array[7]<=reg_array[6]; reg_array[8]<=reg_array[7]; reg_array[9]<=reg_array[8];
            reg_array[10]<=reg_array[9]; reg_array[11]<=reg_array[10]; reg_array[12]<=reg_array[11];
            reg_array[13]<=reg_array[12]; reg_array[14]<=reg_array[13]; reg_array[15]<=reg_array[14];
            if(column==5'b11100) begin row<=row+1; column <=5'b00000; end
            if(row==5'b11100) row<=5'b00000;
            if(column[0]==1&&row[0]==1) begin
                max1=max(reg_array[14], reg_array[15]);
                max2=max(reg_array[1], max1);
                max_pool_out=max(reg_array[0],max2);
            end
        end
    end
    else max_pool_out<=sigmoid_out_fixed;
```

-en 값 설정하여 신호제어
-16개의 register array 사용하여 row, column 에 따라 연산
-max_pool_on_off 값에 따라 maxpool 연산 수행여부 결정



ex9 maxpooling 과제참고

설계 내용

6. fully-connect operation - 1

```
if(fc1==79) begin
    fc1<=0;
    addr_x_Read<=addr_x_Read+1;
    addr_w<= addr_w+1;
    addr_bias<=addr_bias+1;
    addr_x_Write<=0;
    s_fsm<=2'b10;
end
else begin
    fc1<=fc1+1;
    addr_x_Read <= 0;
    addr_w<= addr_w+1;
    addr_bias<=addr_bias+1;
    addr_x_Write<=addr_x_Write+1;
    wea<=1;
end
```

-input: 4900개의 maxpooling 출력 데이터
-output: 80개
-weight: 4900x80개의 데이터
-x memory write address: 출력값 나올때마다 1씩 증가
-x memory read address, weight address: 데이터 읽을때마다 1씩 증가

설계 내용

7. fully-connect operation - 2

```
if(r==79) begin
    r<=0;
    if(fc2==9) begin
        fc2<=0;
        addr_x_Read<=0;
        addr_bias<=0;
        addr_w<=0;
        s_fsm<=2'b00;
    end
    else begin
        fc2<=fc2+1;
        addr_bias<=addr_bias+1;
        addr_x_Read<=4900;
        addr_w<=addr_w+1;
        en2<=1;
    end
end else begin
    r<=r+1;
    addr_x_Read<=addr_x_Read+1;
    addr_w<=addr_w+1;
end
end
```

-input: 80개의 첫번째 fully-connect 출력 데이터
-output: 10개
-weight: 80x10개의 데이터
-x memory write address: 0으로 설정
-x memory read address, weight address: 데이터 읽을때마다 1씩 증가

설계 내용

8. output

```
always @(posedge clk) begin
    if(en2) begin
        resultReg[0]<=max_pool_out;
        resultReg[1]<=resultReg[0];
        resultReg[2]<=resultReg[1];
        resultReg[3]<=resultReg[2];
        resultReg[4]<=resultReg[3];
        resultReg[5]<=resultReg[4];
        resultReg[6]<=resultReg[5];
        resultReg[7]<=resultReg[6];
        resultReg[8]<=resultReg[7];
        resultReg[9]<=resultReg[8];

        resultcount<=resultcount+1;
        if(resultcount==4'b1010) begin
            max1a<=max(resultReg[0],resultReg[1]); max2a<=max(max1a,resultReg[2]);
            max3a<=max(max2a,resultReg[3]); max4a<=max(max3a,resultReg[4]);
            max5a<=max(max4a,resultReg[5]); max6a<=max(max5a,resultReg[6]);
            max7a<=max(max6a,resultReg[7]); max8a<=max(max7a,resultReg[8]);
            max_result<=max(max8a,resultReg[9]);
        end
    end
end
```

-2번째 fully-connect 연산의 결과를 저장하기 위해 10개의 register array 사용
-register array에 데이터가 모두 들어오면 최대값 구한 후 출력

결과

- 최종 미완성
- 완성하지 못한 것
- 1. sigmoid에서 exponential 값 에러

- 2. multiple 연산 이후의 delay 등을 고려한 타이밍 조절
- 3. max pooling 후 x memory write address 할당
- 4. accumulator overflow 조절
- 5. 연산 결과 및 HDMI 케이블로 연결하여 화면에서 인식 결과 확인