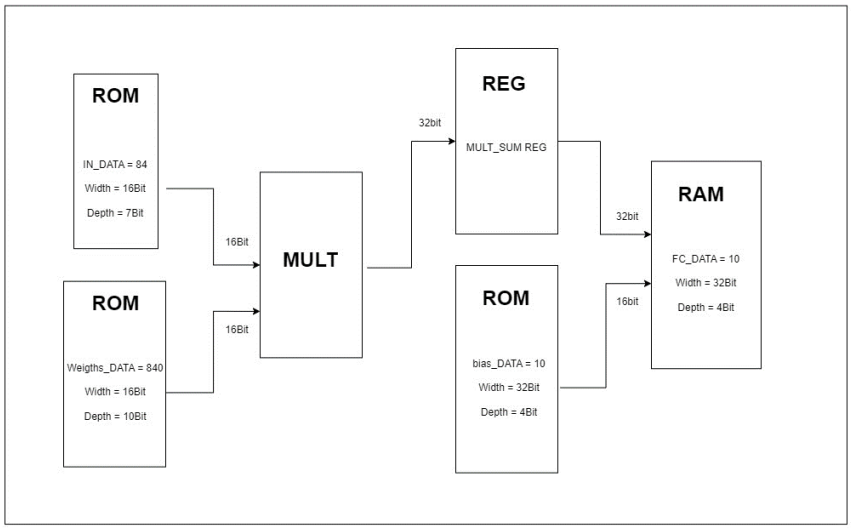
5 주차 Technical brief_ 윤중철

개요 – Verilog 하드웨어 언어를 통한 84 to 10 Fully Connected layer 구현

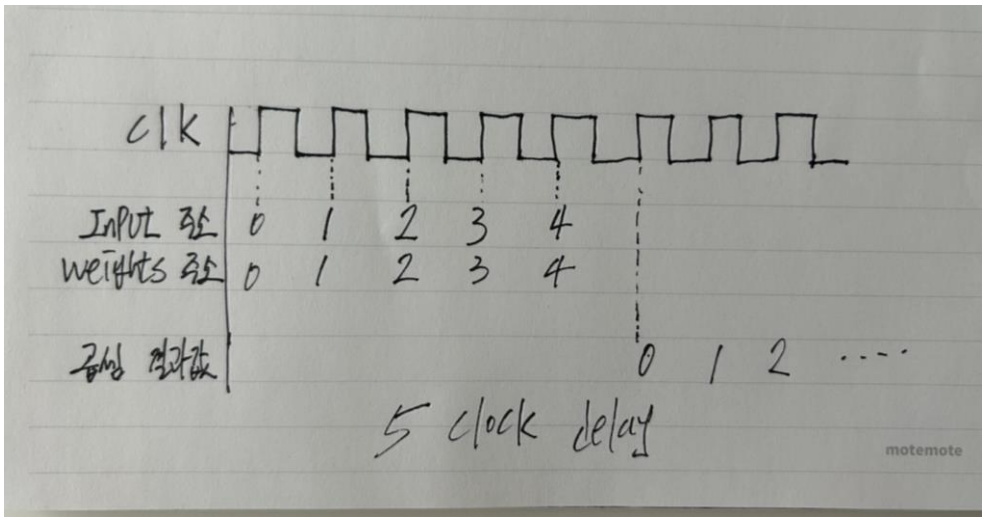
설계 및 구현 내용

1. TOP 설계



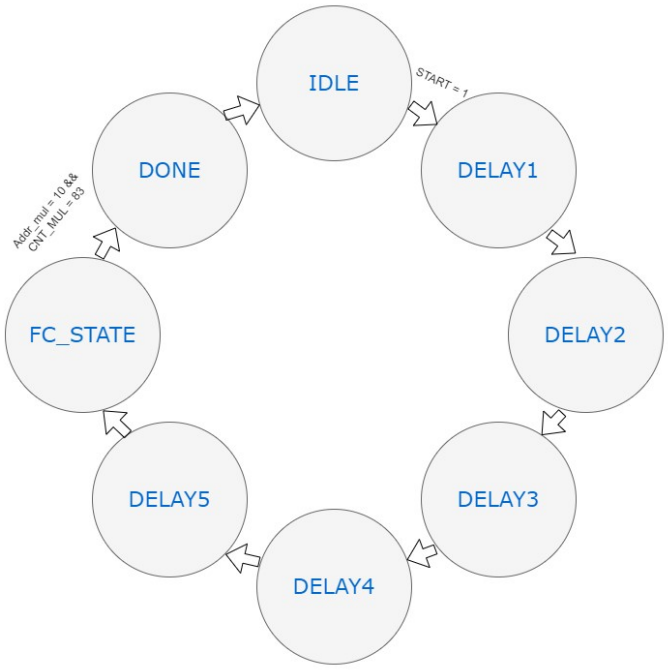
Comment – INPUT, WEIGHTS, BIAS 데이터를 저장한 3개의 ROM과 MULT IP, 결과를 저장할 RAM 으로 구성 FC 연산을 위해 INPUT의 주소는 계속 증가하다 83 번지수가 되면 초기화를 진행 후 다시 증가하도록 구성, WEIGHTS 주소 역시 계속 증가하다 839번지수가 되면 초기화를 진행 후 읽기 종료, MULT는 결과 값은 REG에 저장되고 84개의 누적합이 이루어지면 BIAS와 더해져서 RAM에 INPUT DATA로 들어가고 초기화 진행. 총 RAM 10개의 주소에 누적합과 BIAS값을 쓰도록 구성.

2. DELAY



Comment – 곱셈 결과 값 까지 ROM에서의 READ_DELAY 2CLK, MULT IP에서의 MULT_DELAY 3CLK, 총 5CLK의 딜레이가 발생.

3. FSM



Comment – 총 8개의 state로 구성 IDLE state에서 start 신호가 rising 되면 다음 state로 이동 5clk delay를 위해 delay state 5개를 지나고 FC_State에서 fully connected 연산 진행 ram의 주소가 10번지이고 Control를 위한, cnt_mul이 83이 되면 Done state로 이동 하고 state가 다시 IDLE로 이동하도록 구성.

3. TOP module

```
module FC_84to10(clk, rst, start,dout);
 input clk, rst, start;
 output[31:0] dout;
 (*keep = "true" *)reg  [6:0] addr_in;
 (*keep = "true" *)reg [9:0] addr_w;
 (*keep = "true" *)reg [3:0] addr_mul;
 (*keep = "true" *)reg [6:0] cnt_mul;
 (*keep = "true" *)reg wea;
 (*keep = "true" *)reg signed [31:0] din_mul;
 (*keep = "true" *)reg signed [31:0] dout_mul;
 (*keep = "true" *)reg [3:0] state;
 wire signed [15:0] out_in, out_w, out_b;
 wire signed [31:0] data_out;
```

Comment – Input, output, reg, wire 선언

```
 in_84 u0(.clka(clk), .addra(addr_in), .douta(out_in));
 weights_840 u1(.clka(clk), .addra(addr_w), .douta(out_w));
 bias_10 u2(.clka(clk), .addra(addr_mul), .douta(out_b));
 mult_gen_0 u3(.CLK(clk), .A(out_in), .B(out_w), .P(data_out));
 mul_reg10 u4(.clka(clk), .wea(wea), .addra(addr_mul), .dina(din_mul), .douta(dout));
```

Comment – 모듈 인스턴트

```
//state
always@(posedge clk or posedge rst)
begin
    if(rst)
        state <= IDLE;
    else
        case(state)
            IDLE : if(start) state <= DELAY_1; else state <= IDLE;
            DELAY_1 : state <= DELAY_2;
            DELAY_2 : state <= DELAY_3;
            DELAY_3 : state <= DELAY_4;
            DELAY_4 : state <= DELAY_5;
            DELAY_5 : state <= FC_1;
            FC_1 :if(addr_mul == 4'd10 && cnt_mul == 7'd83) state <= DONE; else state <= FC_1;
            DONE : state <= IDLE;
            default : state <= IDLE;
            endcase
end
```

Comment – State 구성

```
// weights
always@(posedge clk or posedge rst)
begin
    if(rst)
        addr_w <= 10'd0;
    else
        case(state)
            DELAY_1 : addr_w <= addr_w + 1'd1;
            DELAY_2 : addr_w <= addr_w + 1'd1;
            DELAY_3 : addr_w <= addr_w + 1'd1;
            DELAY_4 : addr_w <= addr_w + 1'd1;
            DELAY_5 : addr_w <= addr_w + 1'd1;
            FC_1 :if(addr_w == 10'd839) addr_w <= 0; else addr_w <= addr_w + 1'd1;
            default : addr_w <= 10'd0;
            endcase
end

// input
always@(posedge clk or posedge rst)
begin
    if(rst)
        addr_in <= 7'd0;
    else
        case(state)
            DELAY_1 : addr_in <= addr_in + 1'd1;
            DELAY_2 : addr_in <= addr_in + 1'd1;
            DELAY_3 : addr_in <= addr_in + 1'd1;
            DELAY_4 : addr_in <= addr_in + 1'd1;
            DELAY_5 : addr_in <= addr_in + 1'd1;
            FC_1 :if(addr_in == 7'd83) addr_in <= 0; else addr_in <= addr_in + 1'd1;
            default : addr_in <= 7'd0;
            endcase
end
```

Comment – Input 주소, Weights 주소 제어 부분, IDLE state를 이동하면 주소 값이 증가하도록 설정 Input 주소는 83번지가 되면 초기화 후 다시 증가하고 Weights 주소는 839번지가 되면 초기화 되고 읽기를 종료하도록 구성

```
always@(posedge clk or posedge rst)
begin
    if(rst)
        cnt_mul <= 7'd0;
    else
        case(state)
            FC_1 :if(cnt_mul == 7'd83) cnt_mul <= 0; else cnt_mul <= cnt_mul + 1'd1;
            default : cnt_mul <= 7'd0;
            endcase
end
```

Comment – Control을 위한 cnt, FC_1 state로 이동하면 cnt는 순차적으로 증가 83이되면 초기화 되고 다시 증가하도록 구성

```
 // Multiplier sum
always@(posedge clk or posedge rst)
begin
    if(rst)
        dout_mul <= 32'd0;
    else
        case(state)
            FC_1 :if(cnt_mul == 7'd83) dout_mul <= 32'd0; else dout_mul <= dout_mul + data_out;
            default : dout_mul <= 32'd0;
            endcase
end
```

Comment – 누적합이 이루어지는 부분, MULT 결과 값인 data_out 값이 dout_mul 레지스터에 누적되어 저장되고 제어 카운트인 cnt_mul이 83이되면 초기화 되도록 구성

```
 // din_ram = dout 10 + bias
always@(posedge clk or posedge rst)
begin
    if(rst)
        din_ram <= 32'd0;
    else
        case(state)
            FC_1 :if(cnt_mul == 7'd83) din_ram <= dout_mul + out_b; else din_ram <= din_ram;
            default : din_ram <= 32'd0;
            endcase
end
```

Comment – RAM의 input data 부분, FC_1 State에서 제어 카운트인 cnt_mul이 83이되면 누적합인 dout_mul 과 bias를 더하여 쓰여지도록 구성, 83이 아닐때는 값을 유지하도록 구성

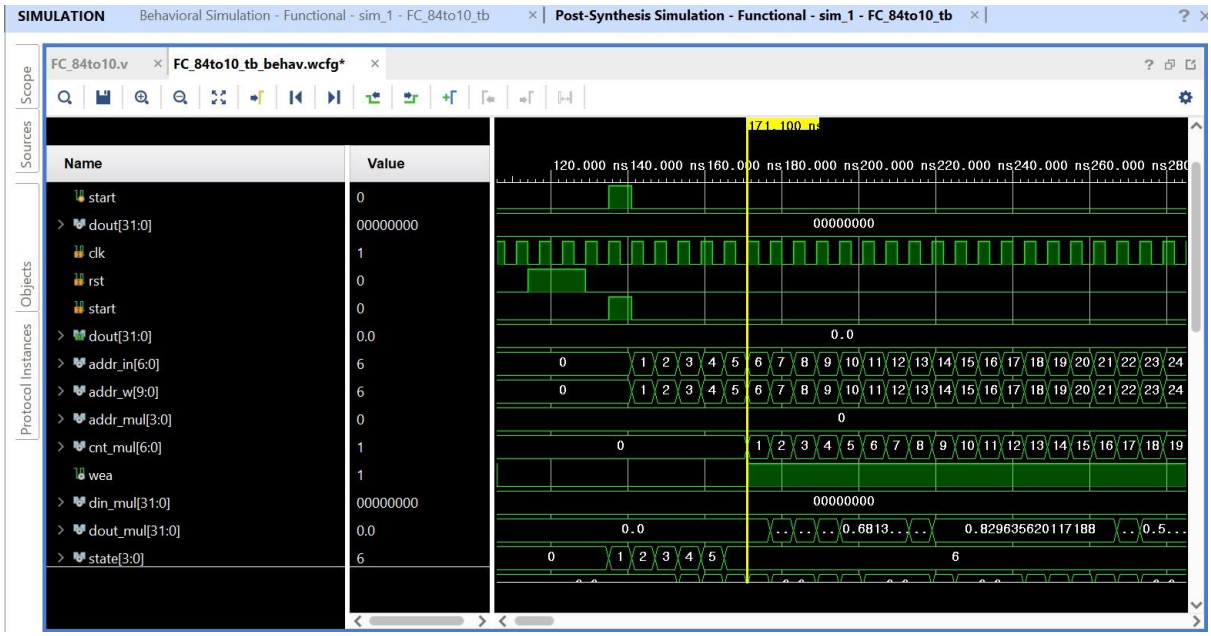
```
// ram_addr
always@(posedge clk or posedge rst)
begin
    if(rst)
        addr_ram <= 4'd0;
    else
        case(state)
            FC_1 :if(cnt_mul == 7'd83 && addr_ram != 4'd10) addr_ram <= addr_ram + 1'd1;
                  else if(cnt_mul == 7'd83 && addr_ram == 4'd10) addr_ram <= 4'd0;
                  else addr_ram <= addr_ram;
        default : addr_ram <= 4'd0;
        endcase
end
```

Comment – RAM의 주소 이동을 위한 부분, cnt_mul이 83이되고 주소가 10이 아니라면 주소 값이 증가하도록 구성 이후, 주소가 10번지가되고 카운트가 83이되면 초기화 하도록 구성

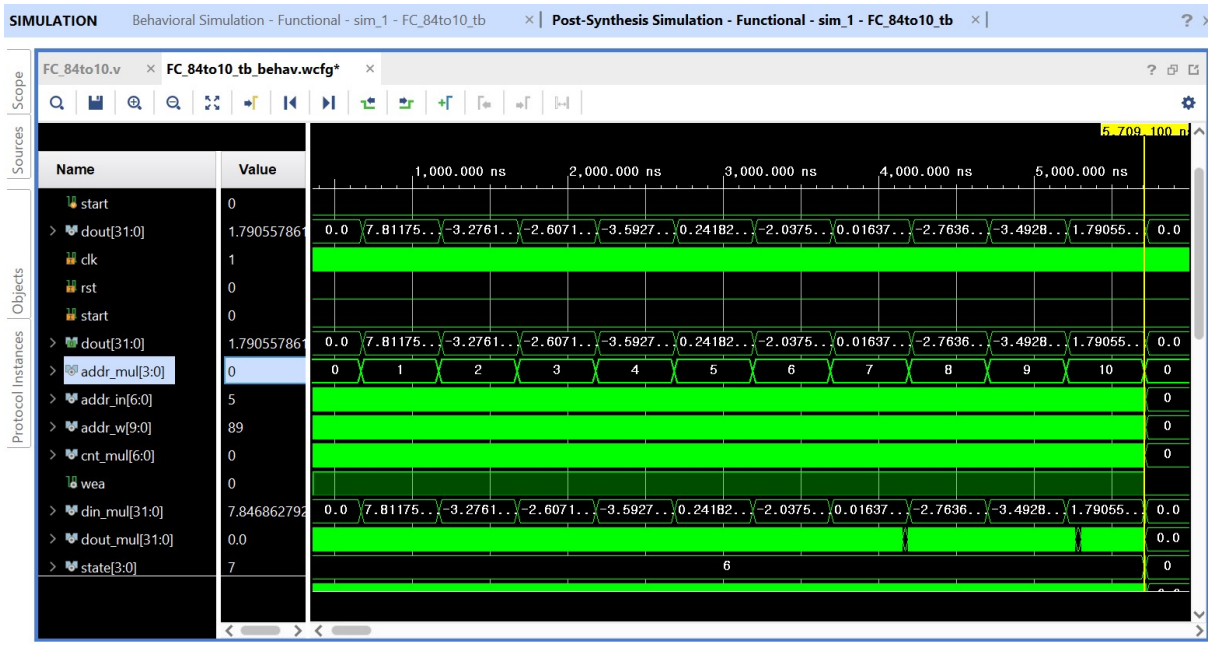
```
// wea
always@(posedge clk or posedge rst)
begin
    if(rst)
        wea <= 1'd0;
    else
        case(state)
            FC_1 : if(addr_mul == 4'd10&& cnt_mul == 7'd83) wea <= 1'd0; else wea <= 1'd1;
            default : wea <= 1'd0;
            endcase
end
```

Comment – RAM의 Write enable 부분, FC_1 State 에서 다음 조건을 만족하면 0이되고 아닐시 1을 유지하도록 구성

결과



Comment – IDLE 신호가 이동하면 INPUT, WEIGHTS 주소가 증가하는 것을 확인, FC_1 State에서 Control Count인 cnt_mul이 증가하고 wea 신호가 1이 되는 것을 확인.



Comment – 누적합과 bias 가 더해진 결과 값이 RAM 주소에 정상적으로 저장되는 것을 확인.

```c
#include <stdio.h>
#include "weights.h"
#include "biases.h"

int main()
{

    FILE* fp_result = fopen("layer6_out.dat", "rb");
     float buf[84];
    fread(buf, sizeof(float), 84, fp_result);
    fclose(fp_result);
    int i, j;
    float sum[10];

    for(i = 0; i < 10; i++)
    {
        sum[i] = (int)(biases_F3[i]* 256*256);
        for(j = 0; j < 84; j++)
         {
            sum[i] = sum[i] + (int)(weights_F3[i][j] * 256)*(int)(buf[j] * 256);
         }
        printf("%f\n", (float)(sum[i]/(256*256)));
    }
}
```

```
PS C:\Users\user\Desktop\python> gcc verification.c
PS C:\Users\user\Desktop\python> .\a.exe
7.811752
-3.276199
-2.607178
-3.592789
0.241821
-2.037506
0.016373
-2.763657
-3.492828
1.790558
```

Comment – simulation 결과 값 검증을 위해 c언어를 통해 연산 값 출력 일치함을 확인함.