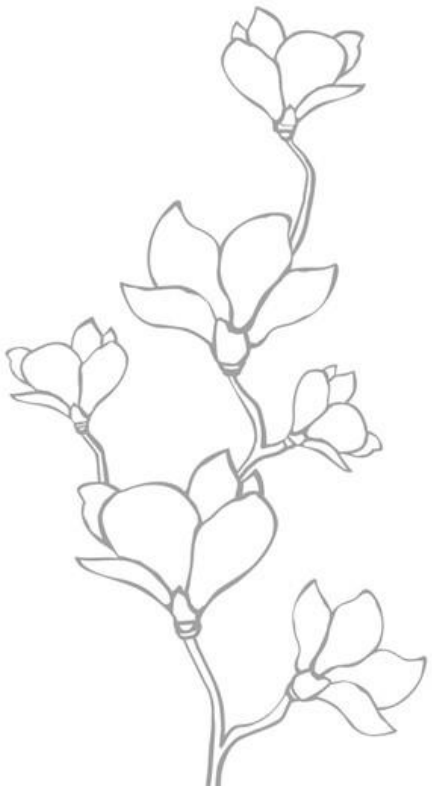
인공지능반도체

프로젝트 2

Pattern Classification DNN hardware Design

KYUNG HEE UNIVERSITY



과목명	인공지능반도체
담당교수	홍상훈 교수님
학과	전자공학과
팀명	Team3
학번	
이름	

- **팀원 평가**

팀원	권용환	최지훈	이규린
점수	100	100	100

- **Index**

PROJECT2: Pattern Classification DNN HW design

0. Project Goal

본 프로젝트는 학습에 최적화된 신경망 Hardware 설계를 목표한다. 이를 위해 RRAM weight multiply model을 사용하고, Inferencing과 Back-Propagation을 검증한다. 나아가, 설계한 모델을 통해 DNN을 구성하고, 패턴 분류를 진행하며 그 응용을 확장시킨다.

1. Modified Cell, Sigconv, Unit for Inference

우리가 만든 cell은 몇 가지 문제를 가지고 있었다. 우선 bl라인을 따라 voltage를 위의 cell로 올려주기 위해 만든 패스와 아래 cell로 내려 줄 패스를 분리해야 했다. 분리하지 않으면, 계속해서 가장 아래에 위치한 sigconv에서 공급하는 voltage로 고정된 출력이 있었다. 또한, 아래의 cell로 내려주는 current를 실제로 전류로 취급하기 위해서는 branch를 이용하여 다소 복잡한 작업이 필요했다. 실제로, wbp에 흐르는 전류가 Icp에 따라서 변하지 않고, 고정되는 문제가 있었다. Debugging을 편하게 하기 위해 Current를 모두 Votlage로 바꾸었다. 또한, cell의 동작 조건을 다음과 같이 수정하였다.

Table 1. Operation Mode

	WL	BL_D	Dback
Weight Setting	>0.5V	>1.5V	X
Weight Update	>0.5V	>0.5V	High
Reset	<0.5V	X	X

cell을 수정하기 전에는 SL라인에 의해서 Reset조건이 걸려 있었는데, 우리는 DNN의 첫번째 Layer의 cell만 sigconv로부터 SL신호를 받게 만들고, 다음 Layer부터는 sigconv의 SUM신호를 받도록 할 것이다. 따라서 SL에 제어되는 조건을 conflict가 나지 않도록 수정해야 했다. 그래서 Reset은 WL

라인이 0.5V아래로 떨어져 OFF가 되면 Reset이 되도록 수정하였다. 또한, 수정 전에는 BL_P라인 BL_N라인 GP, GN이 모두 존재했었다. 이는 다음의 분석을 거쳐 패스를 하나로 통합하였다.

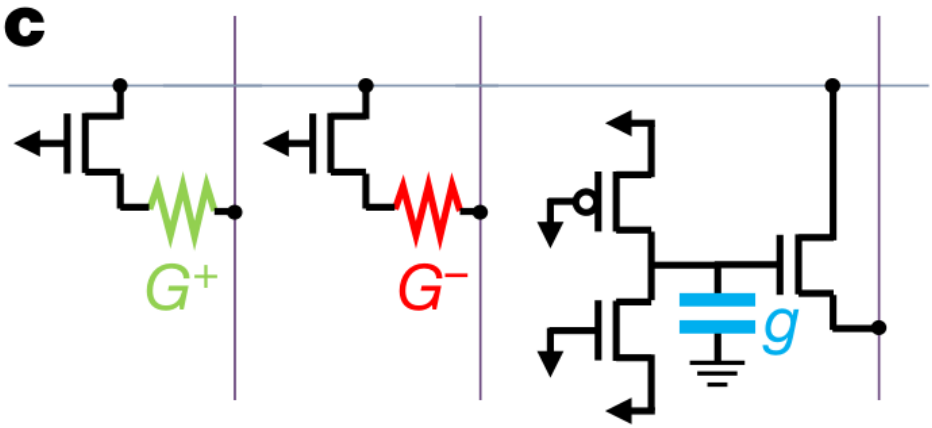


Fig 1. Actual logic for Backpropagation

*"Equivalent-accuracy accelerated neural-network training using analogue memory"*에 따르면 실제, RRAM은 2개의 저항을 사용한다. 이는 Weight가 negative값을 가질 수 있기 때문이다. 그런데, 물리적으로 고정된 저항인데, Training을 어떻게 할까? 해당 Article에서는 Capacitor를 가진 위의 Logic을 이용하여 cap의 charge를 control하여 합해진 current를 출력하도록 하고 있다. 우리가 설계한 cell의 경우에는 최초에 Weight를 만들 때, BL_D가 1.5V를 넘었을 때, 얼마나 많은 시간 동안 그 상태를 유지하고 있었는지에 따라서 최초의 Weight가 결정된다. 즉, 이미, 회로 관점에서는 고정되지 않고, Capacitor를 포함하고 있다고 생각할 수 있다. 우리가 Weight를 update하기 위해서는 $W=G^{+} - G^{-}$에서 W를 기준으로 Gradient Descent를 해야 한다. 그럼 G^{+}와 G^{-}중 무엇을 보정할 것인가? 우리의 코드를 기준으로 하면 Icp와 Icn중 어떤 것을 보정할 것인가에 대한 고민이 생긴다. 여기서 우리는 이미 Capacitor가 포함된 상태로 코드를 구성한 것이기 때문에 Icn도 Icp에 포함시켜도 문제가 없다는 판단이었다. 따라서 G^{+}, G^{-}와 g는 모두 Icp에 포함시킨다는 가정이다.

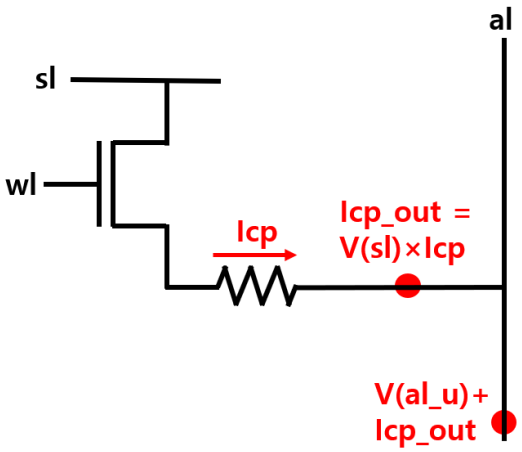


Fig 2

이에 따라서 sigconv도 수정이 있었다. sigconv의 경우 기존에 set신호, reset신호가 있었고, set신호가 들어오면 BL라인에 2V를 인가해 주었다. reset신호가 들어오면, SL라인에 2V를 인가했는데, 우리는 cell의 reset조건을 WL이 OFF될 때로 바꾸었다. 따라서 reset신호를 제거하였고, Digital signal과 Analog signal을 분리하기 위해 set은 Dset으로 naming을 하였다. 또한, Digital신호들의 경우 event trigger에서 cross조건에 따라 단 한 번만 수행해야 하는 동작들이 아니었기 때문에 다소 복잡해 보이는 cross조건들을 간단한 conditional statement로 수정하였다. 이러한 수정은 cell에서도 있었는데, Section 2에서 Backpropagation을 논의한 후 Code를 볼 수 있다. 또한, 우리는 activation을 모두 sigmoid로 사용하는 것으로 가정하여 al로 받아온 신호들을 sigmoid function에 맞추어 sum으로 출력하도록 수정하였다.

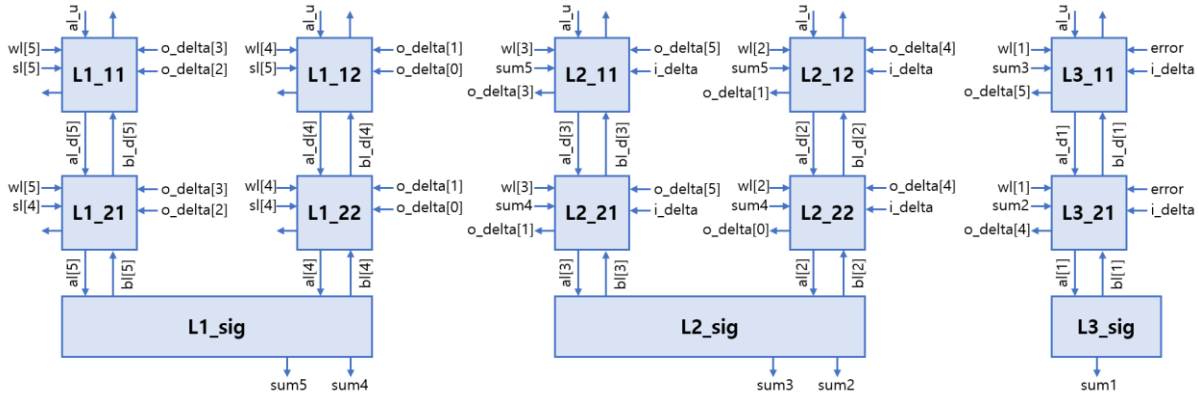


Fig 3

Top module에 해당하는 Unit의 경우 Section 2의 Simple Backpropagation Test를 기준으로 작성하도록 할 것이다. 이후 Section 3에 해당하는 Application인 Pattern Analysis Model에도 같은 방식으로 Unit을 수정할 것이다. Unit은 top module로 connection만 하고 있기 때문에 많은 수정은 없었다. cell들은 al라인을 따라 위에서 아래로 voltage를 더해주는 방식으로 설계되었는데, 가장 위의 단에 해당하는 cell은 위에서 내려올 voltage가 없기 때문에 처음에는 가장 위의 cell의 al_u포트를 open상태로 두었다. 그런데, 이 경우 simulator에서 이따금 voltage가 인가되는 문제가 있었다. 그래서 이 부분에 항상 0V를 인가하도록 접지에 해당하는 전극을 달았다. 이는 Backpropagation을 위해서도 필요한 작업인데, Backpropagation을 위해서 수정된 cell과 추가된 module은 Section 2의 Simple Backpropagation Test에서 다루도록 한다.

2. Simple Backpropagation Test

Backpropagation을 구현하기 위해 다음과 같은 간단한 네트워크를 하나 만들었다.

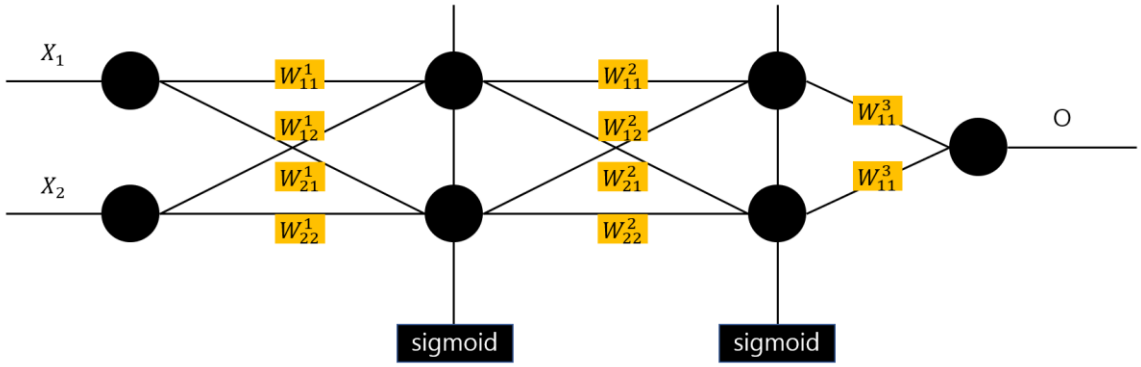


Fig 4. Simple Backpropagation Test Network

Python code

```
import numpy as np

X = np.array(([[1,1]]), dtype=float)
y = np.array(([[1]]), dtype=float)

class Neural_Network(object):
    def __init__(self):
        self.inputSize=2
        self.outputSize=1
        self.hidden1Size=2
        self.hidden2Size=2
        self.lr=0.1

        #Weights
        self.W1=np.array(([[0.1,0.1],[0.1,0.1]]), dtype=float)
        self.W2=np.array(([[0.1,0.1],[0.1,0.1]]), dtype=float)
        self.W3=np.array(([[0.1],[0.1]]), dtype=float)


    def forward(self,X):
        self.z=np.dot(X,self.W1)
        self.z2=self.sigmoid(self.z)
        self.z3=np.dot(self.z2,self.W2)
        self.z4=self.sigmoid(self.z3)
        o=np.dot(self.z4,self.W3)
        return o

    def sigmoid(self,s):
        return 1/(1+np.exp(-s))

    def sigmoidPrime(self,s):
        return s*(1-s)
```

```
    def backward(self,X,y,o):
        # Loss
        self.o_delta=o-y
        self.z4_delta=self.o_delta.dot(self.W3.T)*self.sigmoidPrime(self.z4)
        self.z2_delta=self.z4_delta.dot(self.W2.T)*self.sigmoidPrime(self.z2)
        # Update
        self.W3 -= self.lr*self.z4.T.dot(self.o_delta)
        self.W2 -= self.lr*self.z2.T.dot(self.z4_delta)
        self.W1 -= self.lr*X.T.dot(self.z2_delta)

    def train(self,X,y):
        o=self.forward(X)
        print('-----------Initial Weight-----------')
        print("W1: \n" + str(NN.W1))
        print("W2: \n" + str(NN.W2))
        print("W3: \n" + str(NN.W3))
        self.backward(X,y,o)
        o2=self.forward(X)
        print('-----------Updated Weight-----------')
        print("W1: \n" + str(NN.W1))
        print("W2: \n" + str(NN.W2))
        print("W3: \n" + str(NN.W3))
        print('-----------Prediction-----------')
        print("Before Train:\t" + str(o))
        print("After Train:\t" + str(o2))

NN = Neural_Network()
NN.train(X,y)
```

현재 모델은 모든 Weight를 0.1로 Initialization하고, 한 번의 Inference를 수행하고, 그 다음 Backpropagation을 하면서 Weight를 Update한 후 다시 Inference를 수행하여 Weight와 Output을 출력하도록 하였다. 코드를 짤 때, Assignment1을 참고하였는데, 본 코드의 경우 Loss를 모두 계산한 후 Weight를 Update했는데, 기존의 코드는 하나의 Weight가 Update된 후 Loss를 계산하는데 다시 사용되면서 값이 변해버리는 문제가 있었기 때문이다.

Backpropagation을 고려하여 모듈들을 수정하기 전에 top module의 구조를 보면 다음과 같다.

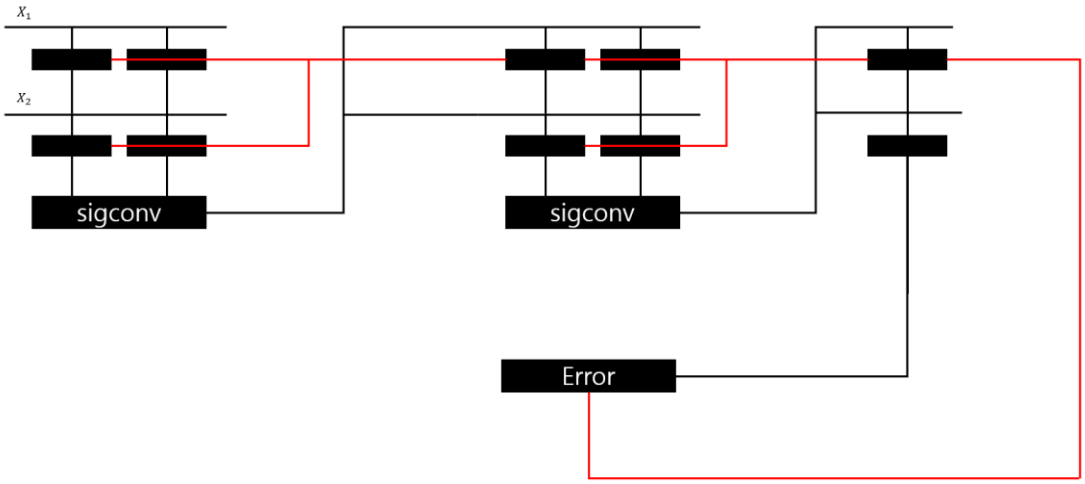


Fig 5. Block Diagram of Top module

빨간색 선은 Back-ward pass이다. 첫번째 Layer의 1열에 해당하는 cell은 두번째 Layer의 1행에 해당하는 cell로부터 Loss를 받아야 한다. 마찬가지로, 두번째 Layer의 1열에 해당하는 cell은 세번째 Layer의 1행에 해당하는 cell로부터 Loss를 받아야 한다. Loss는 Gradient Descent로부터 구해진다. Gradient Descent를 바탕으로 W_{11}^3 의 Loss를 구하면 다음과 같다.

우선, Mean-Square-Error를 사용할 것이고, 이때, Cost function은 다음과 같이 정의한다.

$$\textbf{Cost function} = \frac{1}{2N}\sum(t-o)^2 \qquad \textbf{Eq 1}$$

현재 출력은 하나밖에 없기 때문에 항상 $\frac{\partial E}{\partial O}$ 의 값은 $(o-t)$ 가 된다. 이때, t는 Label을 의미한다.

W_{11}^3 의 경우 $\frac{\partial E}{\partial W_{11}^3} = \frac{\partial E}{\partial O} \times \frac{\partial O}{\partial W_{11}^3}$ 인데, $\frac{\partial E}{\partial O}$ 는 앞에서 전달받은 Loss이다. 즉, 전달받은 Loss와 입력 데이터를 곱하면 W_{11}^3의 Loss가 구해진다. 이는 모든 Weight에 대하여 동일하게 적용된다. 단, 주의할 것은 첫번째 레이어에서 두번째 레이어로 넘어가면 패스가 2개가 되므로 입력으로 받는 Loss는 두 개가 되어야 한다. 우리는 모든 activation을 sigmoid로 사용하고 있다는 것을 전제로 하는데, 출력으로 내보내 주어야 할 Loss는 다음과 같이 정리된다.

$$\delta_{out} = Weight * input * (1 - input) * \sum \delta_{in} \textbf{Eq 2}$$

먼저, 출력으로 내보내는 Loss를 먼저 계산해야 한다. sigmoid activation의 영향으로 input * (1 – input)이 포함되었다. 또한, Update전에 Weight를 곱하여 내보내 주어야 한다.

$$Weight_{new} = Weight_{old} - learning\ rate * input * \sum \delta_{in} \qquad \textbf{Eq 3}$$

출력으로 내보낼 Loss를 계산한 후에는 Weight를 Update한다. 만약, Weight를 먼저 Update해버리면 앞의 Layer는 Update가 된 Weight를 기준으로 구한 Loss를 입력으로 받게 된다. 기본적인 논의가 끝났으니 이제 수정된 코드들을 하나씩 살펴보도록 하자.

RRAM_cell

```
`include "disciplines.h"
module RRAM_cell(Dback,i_delta1,i_delta2,o_delta,wl,sl,al_u,al_d,bl_u,bl_d);
//weight_out);
// input
input Dback;
// real
real Icp = 0;
real Icp_out = 0;
real vdd=1.0;
real Gp_time, Gp_time_start;
real lr = 0.1;
real delta = 0;
// electrical
electrical i_delta1, i_delta2, o_delta, wl, sl, al_u, al_d, bl_u, bl_d;
// analog
analog begin
        @(cross (V(bl_d)-1.5*vdd)) begin // Set
                if(V(wl)>0.5*vdd && V(bl_d)>1.5*vdd) // wl : 1V, bl : 1.5V+
                        Gp_time_start=$abstime;
                else if(V(wl)>0.5*vdd && V(bl_d)<=1.5*vdd) begin // wl : 1V,
bl : 1.5V-
                        Gp_time = $abstime - Gp_time_start;
                        Icp =  0.1 * Gp_time/10e-9;
                end
        end

        @(cross (V(bl_d)-0.5*vdd)) begin // back operation
                if(Dback && V(bl_d) > 0.5*vdd && V(wl) > 0.5*vdd) begin
                        delta = (V(i_delta1) + V(i_delta2)) * V(sl) * (1.0 -
V(sl)) * Icp;
                        Icp = Icp - lr * (V(i_delta1) + V(i_delta2)) * V(sl);
                end
        end

        if (V(wl)>0.5*vdd) begin // on mode
                Icp_out= Icp*V(sl);
        end
        else if (V(wl)<0.5*vdd) begin // Reset
                Icp_out = 0;
                Icp = 0;
                delta = 0;
        end

        V(al_d) <+ Icp_out + V(al_u);
        V(bl_u) <+ V(bl_d);
        V(o_delta) <+ delta;
end
```

```
endmodule
```

앞서 논의한 것처럼 2개의 Resistor와 1개의 Capacitor는 모두 Icp에 포함되었다고 가정하고, voltage를 올려주는 bl라인과 voltage를 내려 받는(실제로는 current) al라인이 존재한다. Weight를 Update하는 조건을 보게 되면 @(cross (V(bl_d) – 0.5 *vdd))에 의해 bl_d가 일단 0.5V를 가로지를 때 한 번 이하의 문장이 수행된다. 이때, 0.5V를 넘고 있어야 하며, wl은 ON상태를 유지하고 있어야 하며, Dback신호가 들어와 있어야 한다. 현재는 입력 받는 Loss는 최대 2개이므로 delta_in은 2개인 것을 알 수 있다. Update알고리즘은 위에서 정의한 것과 동일하다. 초기값을 정하는 경우 10ns강 0.1씩 증가되게 하였다. 이는 파이썬 모델과 비교를 쉽게 하기 위함으로 실제로 사용할 경우 훨씬 좋은 resolution을 갖도록 설계할 수 있다는 것이 장점이라고 생각된다.

RRAM_cell_sigconv

```
`include "disciplines.h"
module RRAM_cell_sigconv(Dwl,Dsl,Dbl,Dset,wl,sl,al,bl,sum);
// input
input[1:0] Dwl,Dsl,Dbl;
input Dset;
// electrical
electrical[1:0] wl,sl,al,bl,sum;
// real
real vdd=1.0;
// analog
analog begin
   // wl
   if(Dwl[1]) begin
      V(wl[1]) <+ vdd;
   end
   else begin
      V(wl[1]) <+ 0;
   end
   if(Dwl[0]) begin
      V(wl[0]) <+ vdd;
   end
   else begin
      V(wl[0]) <+ 0;
   end
   // sl
   if(Dsl[1]) begin
      V(sl[1]) <+ vdd;
   end
   else begin
      V(sl[1]) <+ 0;
   end
   if(Dsl[0]) begin
      V(sl[0]) <+ vdd;
   end
   else begin
      V(sl[0]) <+ 0;
   end
```

```
    // bl
    if (Dset) begin
       if(Dbl[1]) begin
          V(bl[1]) <+ 2*vdd;
       end
       else begin
          V(bl[1]) <+ 0;
       end
       if(Dbl[0]) begin
          V(bl[0]) <+ 2*vdd;
       end
       else begin
          V(bl[0]) <+ 0;
       end
    end
    else begin
       if(Dbl[1]) begin
          V(bl[1]) <+ vdd;
       end
       else begin
          V(bl[1]) <+ 0;
       end
       if(Dbl[0]) begin
          V(bl[0]) <+ vdd;
       end
       else begin
          V(bl[0]) <+ 0;
       end
    end
    // al
    V(sum[1])<+ 1 / (1 + exp(-1 * V(al[1])));
    V(sum[0])<+ 1 / (1 + exp(-1 * V(al[0])));

end
endmodule
```

우선, Digital입력에 대하여 cross조건을 모든 제거하였다. cross될 때, 한 번만 수행될 필요가 없기 때문이다. al라인으로 받은 signal을 Sigmoid Activation 함수에 넣어 sum으로 출력한다. 그 외에 특별한 수정 사항은 없다. 단, reset은 앞서 언급한 것과 마찬가지로 제거하였고, Dset이 들어왔을 때, Dbl이 high라면 bl라인에 2V를 인가하도록 수정하였다.

RRAM_error

```
`include "disciplines.h"
module RRAM_error(Dlabel,sum,error);
// input
input Dlabel;
// electrical
electrical sum, error;
// analog
analog begin
    V(error) <+ V(sum) - Dlabel;
```

```
end
endmodule
```

위의 모듈의 경우 $\frac{\partial E}{\partial O}$ 를 계산해 주는 역할을 한다. 현재는 출력단에 sigmoid activation을 사용하지 않았기 때문에 (o – t) 를 출력하고 있는데, 만약 sigmoid activation을 사용한다면 (o – t) * o * (1 – o) 를 출력하면 되고, 이는 앞서 언급한 input * (1 – input) 과 등가인 것을 알 수 있다.

RRAM_unit

```
`include "disciplines.h"
module
RRAM_unit(Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0,Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0,Dbl5,Dbl4
,Dbl3,Dbl2,Dbl1,Dbl0,Dset,Dback,Dlabel);
// input
input Dwl5, Dwl4, Dwl3, Dwl2, Dwl1, Dwl0;
input Dsl5, Dsl4, Dsl3, Dsl2, Dsl1, Dsl0;
input Dbl5, Dbl4, Dbl3, Dbl2, Dbl1, Dbl0;
input Dset, Dback, Dlabel;
// electrical
electrical[5:0] wl,al,bl,al_d,bl_d;
electrical[5:0] sl;
electrical[5:0] o_delta;
electrical sum5, sum4, sum3, sum2, sum1, sum0;
electrical error;
electrical al_u, i_delta;
// RRAM cell instantiation
RRAM_cell L1_11
(.Dback(Dback),.i_delta1(o_delta[3]),.i_delta2(o_delta[2]),.o_delta(),.wl(wl[5]
),.sl(sl[5]),.al_u(al_u),.al_d(al_d[5]),.bl_u(),.bl_d(bl_d[5]));
RRAM_cell L1_12
(.Dback(Dback),.i_delta1(o_delta[1]),.i_delta2(o_delta[0]),.o_delta(),.wl(wl[4]
),.sl(sl[5]),.al_u(al_u),.al_d(al_d[4]),.bl_u(),.bl_d(bl_d[4]));
RRAM_cell L1_21
(.Dback(Dback),.i_delta1(o_delta[3]),.i_delta2(o_delta[2]),.o_delta(),.wl(wl[5]
),.sl(sl[4]),.al_u(al_d[5]),.al_d(al[5]),.bl_u(bl_d[5]),.bl_d(bl[5]));
RRAM_cell L1_22
(.Dback(Dback),.i_delta1(o_delta[1]),.i_delta2(o_delta[0]),.o_delta(),.wl(wl[4]
),.sl(sl[4]),.al_u(al_d[4]),.al_d(al[4]),.bl_u(bl_d[4]),.bl_d(bl[4]));

RRAM_cell L2_11
(.Dback(Dback),.i_delta1(o_delta[5]),.i_delta2(i_delta),.o_delta(o_delta[3]),.w
l(wl[3]),.sl(sum5),.al_u(al_u),.al_d(al_d[3]),.bl_u(),.bl_d(bl_d[3]));
RRAM_cell L2_12
(.Dback(Dback),.i_delta1(o_delta[4]),.i_delta2(i_delta),.o_delta(o_delta[2]),.w
l(wl[2]),.sl(sum5),.al_u(al_u),.al_d(al_d[2]),.bl_u(),.bl_d(bl_d[2]));
RRAM_cell L2_21
(.Dback(Dback),.i_delta1(o_delta[5]),.i_delta2(i_delta),.o_delta(o_delta[1]),.w
l(wl[3]),.sl(sum4),.al_u(al_d[3]),.al_d(al[3]),.bl_u(bl_d[3]),.bl_d(bl[3]));
RRAM_cell L2_22
(.Dback(Dback),.i_delta1(o_delta[4]),.i_delta2(i_delta),.o_delta(o_delta[0]),.w
l(wl[2]),.sl(sum4),.al_u(al_d[2]),.al_d(al[2]),.bl_u(bl_d[2]),.bl_d(bl[2]));
```

```
RRAM_cell L3_11
(.Dback(Dback),.i_delta1(error),.i_delta2(i_delta),.o_delta(o_delta[5]),.wl(wl[
1]),.sl(sum3),.al_u(al_u),.al_d(al_d[1]),.bl_u(),.bl_d(bl_d[1]));
RRAM_cell L3_21
(.Dback(Dback),.i_delta1(error),.i_delta2(i_delta),.o_delta(o_delta[4]),.wl(wl[
1]),.sl(sum2),.al_u(al_d[1]),.al_d(al[1]),.bl_u(bl_d[1]),.bl_d(bl[1]));

// Error Calculator
RRAM_error i_error(.Dlabel(Dlabel),.sum(al[1]),.error(error));

// RRAM sigconv instantiation
RRAM_cell_sigconv L1_sig (
.Dwl({Dwl5,Dwl4}),
.Dsl({Dsl5,Dsl4}),
.Dbl({Dbl5,Dbl4}),
.Dset(Dset),
.wl(wl[5:4]),
.sl(sl[5:4]),
.al(al[5:4]),
.bl(bl[5:4]),
.sum({sum5,sum4})
);
RRAM_cell_sigconv L2_sig (
.Dwl({Dwl3,Dwl2}),
.Dsl({Dsl3,Dsl2}),
.Dbl({Dbl3,Dbl2}),
.Dset(Dset),
.wl(wl[3:2]),
.sl(sl[3:2]),
.al(al[3:2]),
.bl(bl[3:2]),
.sum({sum3,sum2})
);
RRAM_cell_sigconv L3_sig (
.Dwl({Dwl1,Dwl0}),
.Dsl({Dsl1,Dsl0}),
.Dbl({Dbl1,Dbl0}),
.Dset(Dset),
.wl(wl[1:0]),
.sl(sl[1:0]),
.al(al[1:0]),
.bl(bl[1:0]),
.sum({sum1,sum0})
);

analog begin
    V(al_u) <+ 0;
    V(i_delta) <+ 0;
end

endmodule
```

Top module에 해당한다. Connection만을 하고 있기 때문에 특별한 점은 없으나 Questa로 Simulation 할 때, 각 Layer의 1행에 해당하는 cell의 al_u을 비워 두면 어떤 cell은 0V가 아닌 voltage가 인가

되는 형태로 출력되는 경우가 종종 있었다. 그래서 입력으로 0V를 받아야 하는 경우 외부에서 0V에 해당하는 voltage 패스를 따로 만들어 넣어주고 있다.

RRAM_unit_tb

```
`timescale 1ns/1ns
module
RRAM_unit_tb(Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0,Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0,Dbl5,D
bl4,Dbl3,Dbl2,Dbl1,Dbl0,Dset,Dback,Dlabel);
output reg Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0;
output reg Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0;
output reg Dbl5,Dbl4,Dbl3,Dbl2,Dbl1,Dbl0;
output reg Dset,Dback,Dlabel;
initial begin
   Dwl5 = 0;   Dwl4 = 0;   Dwl3 = 0;   Dwl2 = 0;   Dwl1 = 0;   Dwl0 = 0;
   Dset = 0;   Dback = 0;  Dlabel = 0;
   Dbl5 = 0;   Dbl4 = 0;   Dbl3 = 0;   Dbl2 = 0;   Dbl1 = 0;   Dbl0 = 0;
   Dsl5 = 0;   Dsl4 = 0;   Dsl3 = 0;   Dsl2 = 0;   Dsl1 = 0;   Dsl0 = 0;
   #10;
   // Transistor ON
   Dwl5 = 1;   Dwl4 = 1;   Dwl3 = 1;   Dwl2 = 1;   Dwl1 = 1;   Dwl0 = 1;
   #10;
   // Initialization
   Dset = 1;
   Dbl5 = 1;   Dbl4 = 1;   Dbl3 = 1;   Dbl2 = 1;   Dbl1 = 1;   Dbl0 = 1;
   #10;
   // Feed-foward
   Dset = 0;
   Dbl5 = 0;   Dbl4 = 0;   Dbl3 = 0;   Dbl2 = 0;   Dbl1 = 0;   Dbl0 = 0;
   Dsl5 = 1;   Dsl4 = 1;
   #10;
   // Error Calculation
   Dlabel = 1;
   #10;
   // Weight Update
   Dback = 1;
   Dbl5 = 0;   Dbl4 = 0;   Dbl3 = 0;   Dbl2 = 0;   Dbl1 = 1;   Dbl0 = 1;
   #10;
   Dbl5 = 0;   Dbl4 = 0;   Dbl3 = 1;   Dbl2 = 1;   Dbl1 = 1;   Dbl0 = 1;
   #10;
   Dbl5 = 1;   Dbl4 = 1;   Dbl3 = 1;   Dbl2 = 1;   Dbl1 = 1;   Dbl0 = 1;
   #10;
   Dbl5 = 0;   Dbl4 = 0;   Dbl3 = 0;   Dbl2 = 0;   Dbl1 = 0;   Dbl0 = 0;
   #10;
   Dback = 0;
   #10;


end
endmodule
```

Python과 동일한 상황으로 테스트하기 위해 모든 Weight를 0.1로 Setting하도록 입력을 주었다. 그 후, Label값을 주고, Dsl5, Dsl4에 1을 주어 입력과 라벨 값을 할당했다. 그 다음, Dback을 high로 두고 Dbl1, Dbl0 부터 순차적으로 high로 바꾸면서 Back propagation이 되도록 입력을 주었다. 이는 3.에서 Signal Controller를 만들어 FSM을 돌면서 Label, Dsl5, Dsl4를 바꾸면서 일련의 update 과정을 반복할 수 있도록 수정될 것이다.

Python Result

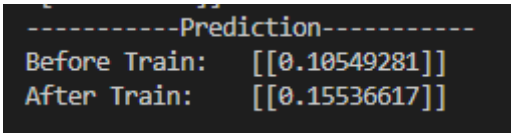


Fig 6

One feedforward 후엔 predict value가 0.10549281로, one backpropagation 후엔 predict value가 0.15536617로 나왔다. 이는 label값을 1로 설정해 두었기 때문에 1에 가까워지려는 train이 잘 되었다고 해석할 수 있다.

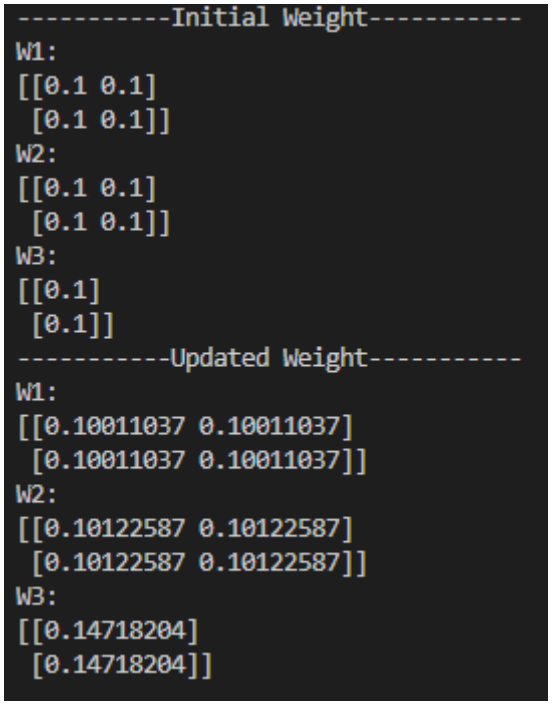


Fig 7

1회의 Backpropagation으로 Python을 통해 구한 Weight는 Fig 7과 같다. 모든 Weight를 동일하게 설정했기 때문에 Penalty를 동일하게 받는 것을 알 수 있다.

Questa Result

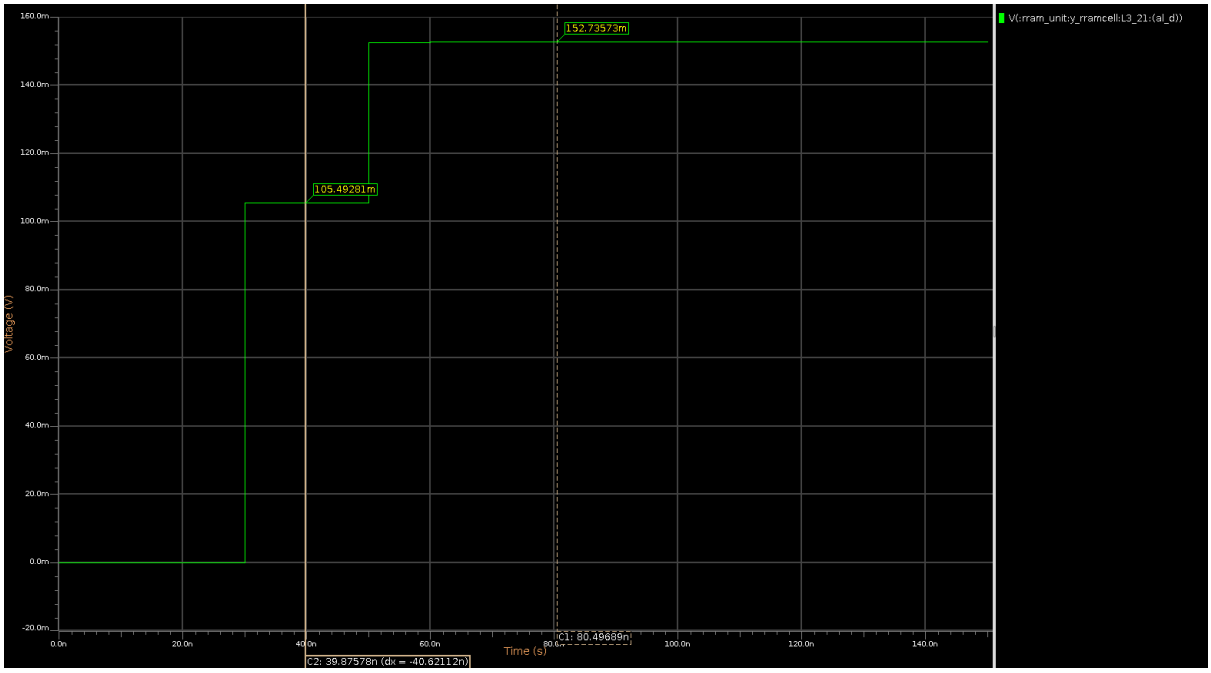


Fig 8

One feedforward 후엔 0.10549281 V가, One backpropagation 후엔 0.15273573 V로 나왔다. Python과 완벽하게 동일한 값이 나온 것은 아니나, 이는 python과 veriloga가 사용하는 근사 방식 등이 달라서 생긴 문제라고 해석하였고, 전체적인 경향성이 같기 때문에 옳게 설계되었다고 말할 수 있다.

Fig 9

Fig 9 가 잘 보이지 않기 때문에 표로 정리하면 다음과 같다.

Table 2. Updated Weight

	Python	Questa
W_{11}^{1}	0.10011037	0.10010455
W_{21}^{1}	0.10011037	0.10010455
W_{12}^{1}	0.10011037	0.10010455
W_{22}^{1}	0.10011037	0.10010455
W_{11}^{2}	0.10122587	0.10116125
W_{21}^{2}	0.10122587	0.10116125
W_{12}^{2}	0.10122587	0.10116125
W_{22}^{2}	0.10122587	0.10116125
W_{11}^{3}	0.14718204	0.14469504
W_{21}^{3}	0.14718204	0.14469504

앞서 Prediction Result 를 본 것처럼 system 상의 약간의 오차는 보이지만 매우 비슷하게 Weight Update 가 일어난 것을 알 수 있다. 이를 바탕으로 Backpropagation 은 잘 수행된다는 것을 확인했기 때문에 다음 Test 는 Depth, Width 가 더 큰 Layer 를 만들고, Training 횟수를 늘릴 수 있도록 Test Sample Generator(Controller)를 만들어 Section 3 를 진행하도록 할 것이다.

3. Pattern Analysis Model

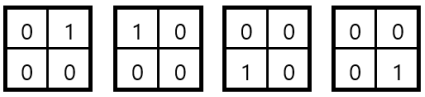
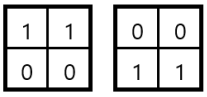
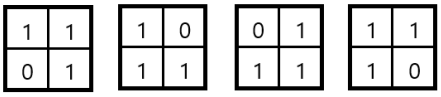
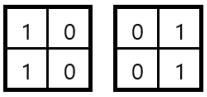
Label		Label	
000		100	
001		101	
010		110	
011		111	

Fig 10

위와 같은 2by2의 binary 정보 16가지를 8가지 종류의 direction(label)로 구분하는 모델을 만들고자 하였다.

이를 위해 input은 4bit, output은 3bit로 결정되는 모델, 즉 input layer의 neuron 개수는 4개이고 output layer의 neuron은 3개인 모델이 필요하고, python 시뮬레이션 결과 hidden layer1의 neuron 수 4개, hidden layer2의 neuron 수 4개인 모델이 적절하다는 것을 알게 되었다.

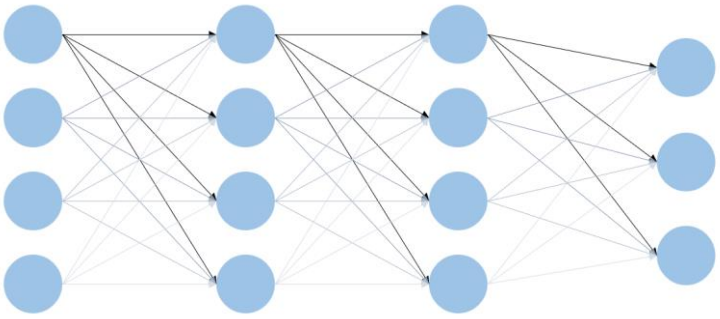


Fig 11

Fig 12

Python code

```
import random as r
import numpy as np


practice_number = 10


A = [[0,0,0,0, 0,0,0],
          [0,0,0,1, 0,0,1],[0,0,1,0, 0,0,1],[0,1,0,0, 0,0,1],[1,0,0,0, 0,0,1
],
          [0,0,1,1, 0,1,0],[1,1,0,0, 0,1,0],
          [1,0,1,0, 0,1,1],[0,1,0,1, 0,1,1],
          [0,1,1,0, 1,0,0],[1,0,0,1, 1,0,1],
          [1,1,0,1, 1,1,0],[1,1,1,0, 1,1,0],[1,0,1,1, 1,1,0],[0,1,1,1, 1,1,0
],
          [1,1,1,1, 1,1,1]]

B=[]
X=[]
Y=[]

for i in range(practice_number) :
  j = r.randint(0,15)
  B.append(A[j])
  X.append(A[j][0:4])
  Y.append(A[j][4:7])

B = np.array(B)
X = np.array(X, dtype=float)
y = np.array(Y, dtype=float)

listB = B.tolist()
f = open('traindata.txt', 'w')
for i in listB :
  for j in range(7) :
    f.write(str(i[j]))
  f.write(str("\n"))

class Neural_Network(object):
    def __init__(self):
        #self.inputSize=4
        #self.outputSize=3
        #self.hidden1Size=4
        #self.hidden2Size=4
        self.lr=0.1

        #Weights
```

```
        self.W1=np.array(([[0.1,0.1,0.1,0.1],[0.1,0.1,0.1,0.1],[0.1,0.1,0.1
,0.1],[0.1,0.1,0.1,0.1]]), dtype=float)
        self.W2=np.array(([[0.1,0.1,0.1,0.1],[0.1,0.1,0.1,0.1],[0.1,0.1,0.1
,0.1],[0.1,0.1,0.1,0.1]]), dtype=float)
        self.W3=np.array(([[0.1,0.1,0.1],[0.1,0.1,0.1],[0.1,0.1,0.1],[0.1,0
.1,0.1]]), dtype=float)


    def forward(self,X):
        self.z=np.dot(X,self.W1)
        self.z2=self.sigmoid(self.z)
        self.z3=np.dot(self.z2,self.W2)
        self.z4=self.sigmoid(self.z3)
        o=np.dot(self.z4,self.W3)
        o=self.sigmoid(o)
        return o

    def sigmoid(self,s):
        return 1/(1+np.exp(-s))

    def sigmoidPrime(self,s):
        return s*(1-s)


    def backward(self,X,y,o):
        # Loss
        self.o_delta=(o-y) * o * (1-o)
        self.z4_delta = self.o_delta.dot(self.W3.T)*self.sigmoidPrime(self.
z4)
        self.z2_delta=self.z4_delta.dot(self.W2.T)*self.sigmoidPrime(self.z
2)
        # Update
        self.W3 -= self.lr  *  self.z4.T.dot(self.o_delta)
        self.W2 -= self.lr  *  self.z2.T.dot(self.z4_delta)
        self.W1 -= self.lr  *  X.T.dot(self.z2_delta)


    def train(self,X,y):
        o=self.forward(X)

        self.backward(X,y,o)
        o2=self.forward(X)

        print('-----------Prediction-----------')
        print("Before Train:\n" + str(o))
        print("After Train:\n" + str(o2))
        print("Real Label:\n" + str(y))
```

```
NN = Neural_Network()
for i in range(practice_number) :
  X_2 = []
  X_2.append(X[i].tolist())
  X_2 = np.array(X_2)

  y_2 = []
  y_2.append(y[i].tolist())
  y_2 = np.array(y_2)

  NN.train(X_2, y_2)
```

Verilоga를 통해 만든 구조를 활용하기 위해 파이썬에서 practice data set 16개 중 random한 순서로 1000개를 txt 파일로 저장하였다.

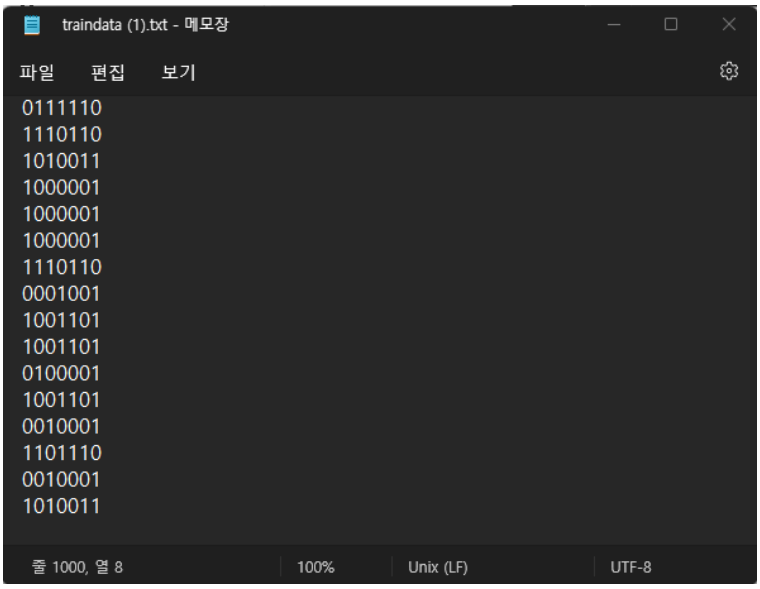


Fig 13

Python result

```
----------Prediction----------
Before Train:
[[0.42908674 0.58476576 0.51494344]]
After Train:
[[0.43323358 0.58771052 0.51106323]]
Real Label:
[[1. 1. 0.]]
----------Prediction----------
Before Train:
[[0.43373883 0.58705255 0.51097854]]
After Train:
[[0.43063795 0.58293832 0.51460583]]
Real Label:
[[0. 0. 1.]]
```

```
-----------Prediction-----------
Before Train:
[[0.43010911 0.58356706 0.5147186 ]]
After Train:
[[0.43425161 0.58652457 0.5108413 ]]
Real Label:
[[1. 1. 0.]]
-----------Prediction-----------
Before Train:
[[0.43449792 0.58620321 0.51080022]]
After Train:
[[0.43862144 0.58202601 0.51445008]]
Real Label:
[[1. 0. 1.]]
-----------Prediction-----------
Before Train:
[[0.43862303 0.5820239  0.5144497 ]]
After Train:
[[0.43542325 0.58500753 0.51061275]]
Real Label:
[[0. 1. 0.]]
-----------Prediction-----------
Before Train:
[[0.43567169 0.58468319 0.51057148]]
After Train:
[[0.43255245 0.58057559 0.51419938]]
Real Label:
[[0. 0. 1.]]
-----------Prediction-----------
Before Train:
[[0.43254964 0.58057894 0.51419998]]
After Train:
[[0.42946167 0.57648074 0.51779809]]
Real Label:
[[0. 0. 1.]]
```

위와 같이 one forward와 one backpropagation 후 output layer의 3개의 label이 모두 real label 값에 가까워지고 있음을 알 수 있다. 즉, weight가 잘 업데이트 되고 있음을 알 수 있다.

Verilog & Verilog-A Result

Neural Network의 학습 과정은 Feed-Forward와 Backpropagation의 반복이다. 따라서, 이 반복되는 과정을 효과적으로 제어하기 위해 State Machine을 이용하여 Input, Control 모듈을 Digital 설계하였다.

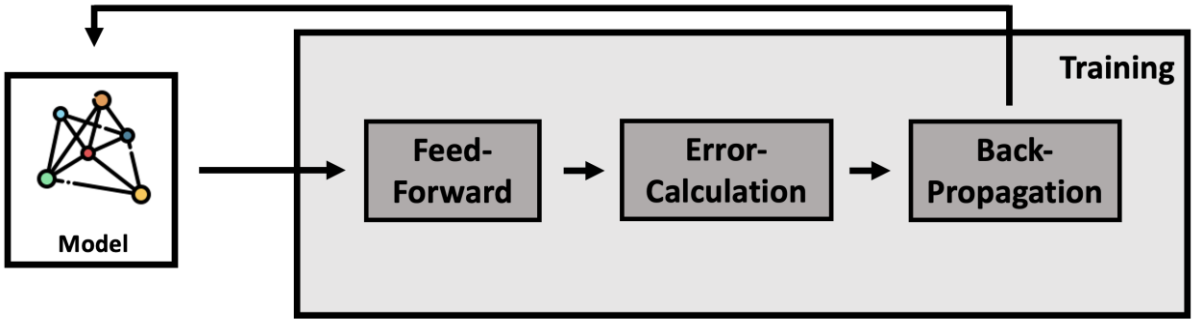


Fig 14

본 프로젝트에서는 Control 모듈에서 아래와 같은 상태를 정의하며 DNN의 학습 과정을 효과적으로 수행한다. 총 4가지의 상태 Initialization, Feed-Forward, Error-Calculation, Back-Propagation을 가진다. Learn 신호를 통해 학습 과정이 시작되며 학습 연산이 수행되는 이벤트를 통해 상태가 천이 된다.

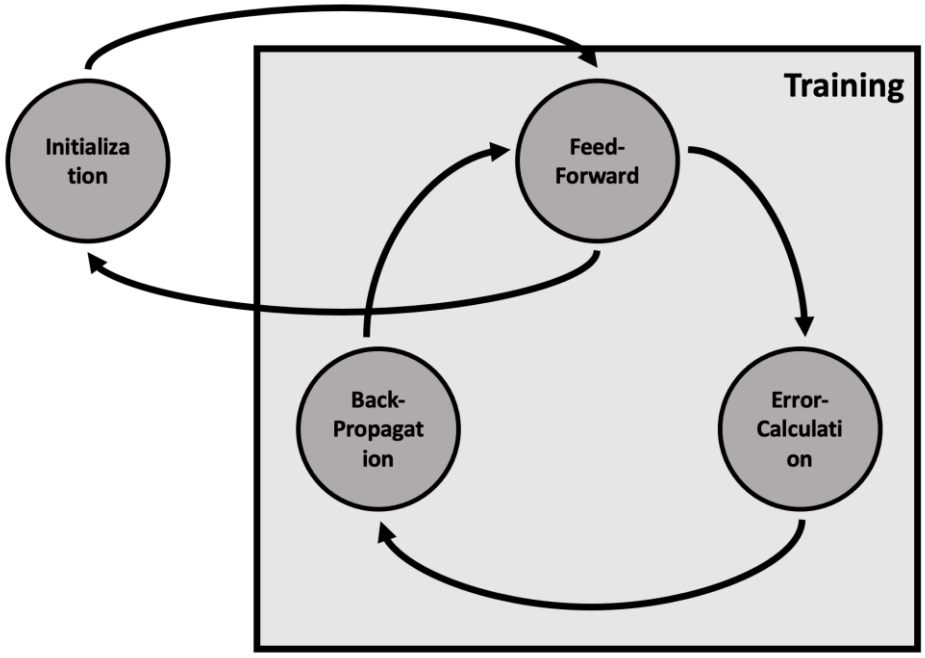


Fig 15

또한, 학습 및 추론을 위한 패턴 입력을 위해 Input allocation 모듈을 설계하였다. $readmemb를 이용해 txt file의 패턴과 label 값을 불러와 enable 신호를 통해 입력하였다. 하단 그림은 최종 설계한 Input과 Control 모듈 구조이다.

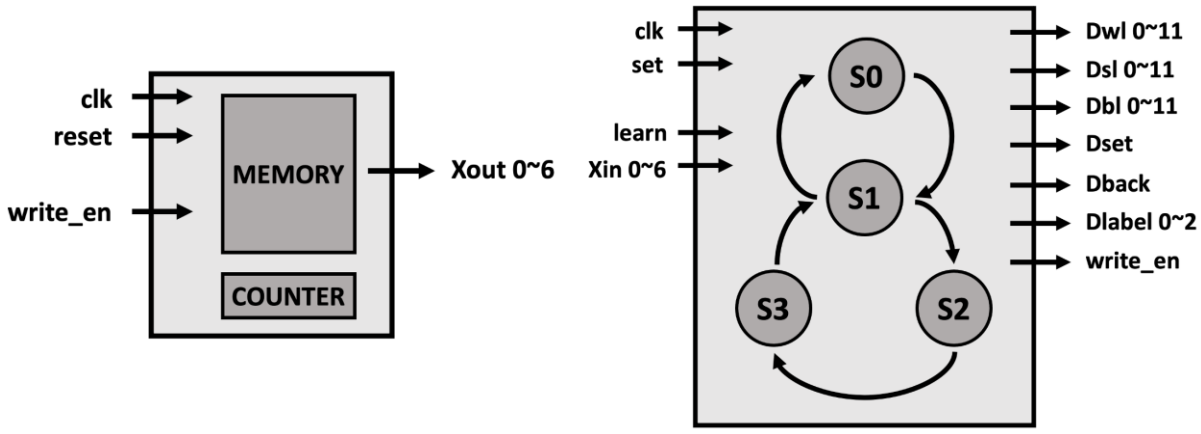


Fig 16

앞서 만든 veriloga 코드를 4, 4, 4, 3의 모델 형식에 맞추어 수정하였고, 다음과 같다.

RRAM_cell

```
`include "disciplines.h"
module RRAM_cell(Dback,i_delta1,i_delta2,i_delta3,i_delta4,
        o_delta,wl,sl,al_u,al_d,bl_u,bl_d);   //weight_out);
// input
input Dback;
// real
real Icp = 0;
real Icp_out = 0;
real vdd=1.0;
real Gp_time, Gp_time_start;
real lr = 0.1;
real delta = 0;
// electrical
electrical i_delta1, i_delta2, i_delta3, i_delta4,
            o_delta, wl, sl, al_u, al_d, bl_u, bl_d;
// analog
analog begin
        @(cross (V(bl_d)-1.5*vdd)) begin // Set
                if(V(wl)>0.5*vdd && V(bl_d)>1.5*vdd) // wl : 1V, bl : 1.5V+
                        Gp_time_start=$abstime;
                else if(V(wl)>0.5*vdd && V(bl_d)<=1.5*vdd) begin // wl : 1V, bl : 1.5V-
                        Gp_time = $abstime - Gp_time_start;
```

```
                    Icp =   0.1 * Gp_time/10e-9;
                end
        end

        @(cross (V(bl_d)-0.5*vdd)) begin // back operation
                if(Dback && V(bl_d) > 0.5*vdd && V(wl) > 0.5*vdd) begin
                        delta = (V(i_delta1) + V(i_delta2)+V(i_delta3)+V(i_delta4))* V(sl) *
(1.0 - V(sl)) * Icp;
                        Icp = Icp - lr * (V(i_delta1) + V(i_delta2)+V(i_delta3)+V(i_delta4)) *
V(sl);
                end
        end

        if (V(wl)>0.5*vdd) begin // on mode
                Icp_out= Icp*V(sl);
        end
        else if (V(wl)<0.5*vdd) begin // Reset
                Icp_out = 0;
                Icp = 0;
                delta = 0;
        end

        V(al_d) <+ Icp_out + V(al_u);
        V(bl_u) <+ V(bl_d);
        V(o_delta) <+ delta;
end
endmodule
```

RRAM_cell_sigconv

```
`include "disciplines.h"
module RRAM_cell_sigconv(Dwl,Dsl,Dbl,Dset,wl,sl,al,bl,sum);
// input
input[3:0] Dwl,Dsl,Dbl;
input Dset;
// electrical
electrical[3:0] wl,sl,al,bl,sum;
```

```
// real
real vdd=1.0;
// analog
analog begin
    // wl
    if(Dwl[3]) begin V(wl[3]) <+ vdd; end
    else begin V(wl[3]) <+ 0; end
    if(Dwl[2]) begin V(wl[2]) <+ vdd; end
    else begin V(wl[2]) <+ 0; end
    if(Dwl[1]) begin V(wl[1]) <+ vdd; end
    else begin V(wl[1]) <+ 0; end
    if(Dwl[0]) begin V(wl[0]) <+ vdd; end
    else begin V(wl[0]) <+ 0; end
    // sl
    if(Dsl[3]) begin   V(sl[3]) <+ vdd; end
    else begin V(sl[3]) <+ 0; end
    if(Dsl[2]) begin   V(sl[2]) <+ vdd; end
    else begin V(sl[2]) <+ 0; end
    if(Dsl[1]) begin   V(sl[1]) <+ vdd; end
    else begin V(sl[1]) <+ 0; end
    if(Dsl[0]) begin   V(sl[0]) <+ vdd; end
    else begin V(sl[0]) <+ 0; end
    // bl
    if (Dset) begin
        if(Dbl[3]) begin V(bl[3]) <+ 2*vdd; end
        else begin V(bl[3]) <+ 0; end
        if(Dbl[2]) begin V(bl[2]) <+ 2*vdd; end
        else begin V(bl[2]) <+ 0; end
        if(Dbl[1]) begin V(bl[1]) <+ 2*vdd; end
        else begin V(bl[1]) <+ 0; end
        if(Dbl[0]) begin V(bl[0]) <+ 2*vdd; end
        else begin V(bl[0]) <+ 0; end
    end
    else begin
        if(Dbl[3]) begin V(bl[3]) <+ vdd; end
        else begin V(bl[3]) <+ 0; end
        if(Dbl[2]) begin V(bl[2]) <+ vdd; end
```

```
        else begin V(bl[2]) <+ 0; end
        if(Dbl[1]) begin V(bl[1]) <+ vdd; end
        else begin V(bl[1]) <+ 0; end
        if(Dbl[0]) begin V(bl[0]) <+ vdd; end
        else begin V(bl[0]) <+ 0; end
    end
    // al
    V(sum[3])<+ 1 / (1 + exp(-1 * V(al[3])));
    V(sum[2])<+ 1 / (1 + exp(-1 * V(al[2])));
    V(sum[1])<+ 1 / (1 + exp(-1 * V(al[1])));
    V(sum[0])<+ 1 / (1 + exp(-1 * V(al[0])));

end
endmodule
```

RRAM_unit

```
`include "disciplines.h"
module RRAM_unit(
            Dwl11,Dwl10,Dwl9,Dwl8,Dwl7,Dwl6,Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0,
            Dsl11,Dsl10,Dsl9,Dsl8,Dsl7,Dsl6,Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0,
            Dbl11,Dbl10,Dbl9,Dbl8,Dbl7,Dbl6,Dbl5,Dbl4,Dbl3,Dbl2,Dbl1,Dbl0,
            Dset,Dback,Dlabel1,Dlabel2,Dlabel3);
// input
input Dwl11,Dwl10,Dwl9, Dwl8, Dwl7, Dwl6, Dwl5, Dwl4, Dwl3, Dwl2, Dwl1, Dwl0;
input Dsl11,Dsl10,Dsl9, Dsl8, Dsl7, Dsl6, Dsl5, Dsl4, Dsl3, Dsl2, Dsl1, Dsl0;
input Dbl11,Dbl10,Dbl9, Dbl8, Dbl7, Dbl6, Dbl5, Dbl4, Dbl3, Dbl2, Dbl1, Dbl0;
input Dset, Dback;
input Dlabel1,Dlabel2,Dlabel3;
// electrical
electrical[11:0] wl,al,bl,al_d_1,al_d_2,al_d_3,bl_d_1,bl_d_2,bl_d_3;
electrical[11:0] sl;
electrical[27:0] o_delta;
electrical sum11, sum10, sum9, sum8, sum7, sum6, sum5, sum4, sum3, sum2, sum1, sum0;
electrical[3:1] error;
electrical al_u;
electrical i_delta;
```

```
////////////////////////////////////////////////////////////////////////////////////
// RRAM cell instantiation
// layer1
RRAM_cell L1_11 (.Dback(Dback),
        .i_delta1(o_delta[15]),.i_delta2(o_delta[14]),.i_delta3(o_delta[13]),.i_delta4(o_delta[12]),
        .o_delta(),.wl(wl[11]),.sl(sl[11]),
        .al_u(al_u),.al_d(al_d_1[11]),.bl_u(),.bl_d(bl_d_1[11]));
RRAM_cell L1_21 (.Dback(Dback),
        .i_delta1(o_delta[15]),.i_delta2(o_delta[14]),.i_delta3(o_delta[13]),.i_delta4(o_delta[12]),
        .o_delta(),.wl(wl[11]),.sl(sl[10]),
        .al_u(al_d_1[11]),.al_d(al_d_2[11]),.bl_u(bl_d_1[11]),.bl_d(bl_d_2[11]));
RRAM_cell L1_31 (.Dback(Dback),
        .i_delta1(o_delta[15]),.i_delta2(o_delta[14]),.i_delta3(o_delta[13]),.i_delta4(o_delta[12]),
        .o_delta(),.wl(wl[11]),.sl(sl[9]),
        .al_u(al_d_2[11]),.al_d(al_d_3[11]),.bl_u(bl_d_2[11]),.bl_d(bl_d_3[11]));
RRAM_cell L1_41 (.Dback(Dback),
        .i_delta1(o_delta[15]),.i_delta2(o_delta[14]),.i_delta3(o_delta[13]),.i_delta4(o_delta[12]),
        .o_delta(),.wl(wl[11]),.sl(sl[8]),
        .al_u(al_d_3[11]),.al_d(al[11]),.bl_u(bl_d_3[11]),.bl_d(bl[11]));

RRAM_cell L1_12 (.Dback(Dback),
        .i_delta1(o_delta[11]),.i_delta2(o_delta[10]),.i_delta3(o_delta[9]),.i_delta4(o_delta[8]),
        .o_delta(),.wl(wl[10]),.sl(sl[11]),
        .al_u(al_u),.al_d(al_d_1[10]),.bl_u(),.bl_d(bl_d_1[10]));
RRAM_cell L1_22 (.Dback(Dback),
        .i_delta1(o_delta[11]),.i_delta2(o_delta[10]),.i_delta3(o_delta[9]),.i_delta4(o_delta[8]),
        .o_delta(),.wl(wl[10]),.sl(sl[10]),
        .al_u(al_d_1[10]),.al_d(al_d_2[10]),.bl_u(bl_d_1[10]),.bl_d(bl_d_2[10]));
RRAM_cell L1_32 (.Dback(Dback),
        .i_delta1(o_delta[11]),.i_delta2(o_delta[10]),.i_delta3(o_delta[9]),.i_delta4(o_delta[8]),
        .o_delta(),.wl(wl[10]),.sl(sl[9]),
        .al_u(al_d_2[10]),.al_d(al_d_3[10]),.bl_u(bl_d_2[10]),.bl_d(bl_d_3[10]));
RRAM_cell L1_42 (.Dback(Dback),
        .i_delta1(o_delta[11]),.i_delta2(o_delta[10]),.i_delta3(o_delta[9]),.i_delta4(o_delta[8]),
        .o_delta(),.wl(wl[10]),.sl(sl[8]),
        .al_u(al_d_3[10]),.al_d(al[10]),.bl_u(bl_d_3[10]),.bl_d(bl[10]));
```

```
RRAM_cell L1_13 (.Dback(Dback),
        .i_delta1(o_delta[7]),.i_delta2(o_delta[6]),.i_delta3(o_delta[5]),.i_delta4(o_delta[4]),
        .o_delta(),.wl(wl[9]),.sl(sl[11]),
        .al_u(al_u),.al_d(al_d_1[9]),.bl_u(),.bl_d(bl_d_1[9]));
RRAM_cell L1_23 (.Dback(Dback),
        .i_delta1(o_delta[7]),.i_delta2(o_delta[6]),.i_delta3(o_delta[5]),.i_delta4(o_delta[4]),
        .o_delta(),.wl(wl[9]),.sl(sl[10]),
        .al_u(al_d_1[9]),.al_d(al_d_2[9]),.bl_u(bl_d_1[9]),.bl_d(bl_d_2[9]));
RRAM_cell L1_33 (.Dback(Dback),
        .i_delta1(o_delta[7]),.i_delta2(o_delta[6]),.i_delta3(o_delta[5]),.i_delta4(o_delta[4]),
        .o_delta(),.wl(wl[9]),.sl(sl[9]),
        .al_u(al_d_2[9]),.al_d(al_d_3[9]),.bl_u(bl_d_2[9]),.bl_d(bl_d_3[9]));
RRAM_cell L1_43 (.Dback(Dback),
        .i_delta1(o_delta[7]),.i_delta2(o_delta[6]),.i_delta3(o_delta[5]),.i_delta4(o_delta[4]),
        .o_delta(),.wl(wl[9]),.sl(sl[8]),
        .al_u(al_d_3[9]),.al_d(al[9]),.bl_u(bl_d_3[9]),.bl_d(bl[9]));

RRAM_cell L1_14 (.Dback(Dback),
        .i_delta1(o_delta[3]),.i_delta2(o_delta[2]),.i_delta3(o_delta[1]),.i_delta4(o_delta[0]),
        .o_delta(),.wl(wl[8]),.sl(sl[11]),
        .al_u(al_u),.al_d(al_d_1[8]),.bl_u(),.bl_d(bl_d_1[8]));
RRAM_cell L1_24 (.Dback(Dback),
        .i_delta1(o_delta[3]),.i_delta2(o_delta[2]),.i_delta3(o_delta[1]),.i_delta4(o_delta[0]),
        .o_delta(),.wl(wl[8]),.sl(sl[10]),
        .al_u(al_d_1[8]),.al_d(al_d_2[8]),.bl_u(bl_d_1[8]),.bl_d(bl_d_2[8]));
RRAM_cell L1_34 (.Dback(Dback),
        .i_delta1(o_delta[3]),.i_delta2(o_delta[2]),.i_delta3(o_delta[1]),.i_delta4(o_delta[0]),
        .o_delta(),.wl(wl[8]),.sl(sl[9]),
        .al_u(al_d_2[8]),.al_d(al_d_3[8]),.bl_u(bl_d_2[8]),.bl_d(bl_d_3[8]));
RRAM_cell L1_44 (.Dback(Dback),
        .i_delta1(o_delta[3]),.i_delta2(o_delta[2]),.i_delta3(o_delta[1]),.i_delta4(o_delta[0]),
        .o_delta(),.wl(wl[8]),.sl(sl[8]),
        .al_u(al_d_3[8]),.al_d(al[8]),.bl_u(bl_d_3[8]),.bl_d(bl[8]));

//layer2
RRAM_cell L2_11 (.Dback(Dback),
```

```
        .i_delta1(o_delta[27]),.i_delta2(o_delta[26]),.i_delta3(o_delta[26]),.i_delta4(i_delta),
        .o_delta(o_delta[15]),.wl(wl[7]),.sl(sum11),
        .al_u(al_u),.al_d(al_d_1[7]),.bl_u(),.bl_d(bl_d_1[7]));
RRAM_cell L2_21 (.Dback(Dback),
        .i_delta1(o_delta[27]),.i_delta2(o_delta[26]),.i_delta3(o_delta[26]),.i_delta4(i_delta),
        .o_delta(o_delta[11]),.wl(wl[7]),.sl(sum10),
        .al_u(al_d_1[7]),.al_d(al_d_2[7]),.bl_u(bl_d_1[7]),.bl_d(bl_d_2[7]));
RRAM_cell L2_31 (.Dback(Dback),
        .i_delta1(o_delta[27]),.i_delta2(o_delta[26]),.i_delta3(o_delta[26]),.i_delta4(i_delta),
        .o_delta(o_delta[7]),.wl(wl[7]),.sl(sum9),
        .al_u(al_d_2[7]),.al_d(al_d_3[7]),.bl_u(bl_d_2[7]),.bl_d(bl_d_3[7]));
RRAM_cell L2_41 (.Dback(Dback),
        .i_delta1(o_delta[27]),.i_delta2(o_delta[26]),.i_delta3(o_delta[26]),.i_delta4(i_delta),
        .o_delta(o_delta[3]),.wl(wl[7]),.sl(sum8),
        .al_u(al_d_3[7]),.al_d(al[7]),.bl_u(bl_d_3[7]),.bl_d(bl[7]));

RRAM_cell L2_12 (.Dback(Dback),
        .i_delta1(o_delta[24]),.i_delta2(o_delta[23]),.i_delta3(o_delta[22]),.i_delta4(i_delta),
        .o_delta(o_delta[14]),.wl(wl[6]),.sl(sum11),
        .al_u(al_u),.al_d(al_d_1[6]),.bl_u(),.bl_d(bl_d_1[6]));
RRAM_cell L2_22 (.Dback(Dback),
        .i_delta1(o_delta[24]),.i_delta2(o_delta[23]),.i_delta3(o_delta[22]),.i_delta4(i_delta),
        .o_delta(o_delta[10]),.wl(wl[6]),.sl(sum10),
        .al_u(al_d_1[6]),.al_d(al_d_2[6]),.bl_u(bl_d_1[6]),.bl_d(bl_d_2[6]));
RRAM_cell L2_32 (.Dback(Dback),
        .i_delta1(o_delta[24]),.i_delta2(o_delta[23]),.i_delta3(o_delta[22]),.i_delta4(i_delta),
        .o_delta(o_delta[6]),.wl(wl[6]),.sl(sum9),
        .al_u(al_d_2[6]),.al_d(al_d_3[6]),.bl_u(bl_d_2[6]),.bl_d(bl_d_3[6]));
RRAM_cell L2_42 (.Dback(Dback),
        .i_delta1(o_delta[24]),.i_delta2(o_delta[23]),.i_delta3(o_delta[22]),.i_delta4(i_delta),
        .o_delta(o_delta[2]),.wl(wl[6]),.sl(sum8),
        .al_u(al_d_3[6]),.al_d(al[6]),.bl_u(bl_d_3[6]),.bl_d(bl[6]));

RRAM_cell L2_13 (.Dback(Dback),
        .i_delta1(o_delta[21]),.i_delta2(o_delta[20]),.i_delta3(o_delta[19]),.i_delta4(i_delta),
        .o_delta(o_delta[13]),.wl(wl[5]),.sl(sum11),
        .al_u(al_u),.al_d(al_d_1[5]),.bl_u(),.bl_d(bl_d_1[5]));
```

```
RRAM_cell L2_23 (.Dback(Dback),
        .i_delta1(o_delta[21]),.i_delta2(o_delta[20]),.i_delta3(o_delta[19]),.i_delta4(i_delta),
        .o_delta(o_delta[9]),.wl(wl[5]),.sl(sum10),
        .al_u(al_d_1[5]),.al_d(al_d_2[5]),.bl_u(bl_d_1[5]),.bl_d(bl_d_2[5]));
RRAM_cell L2_33 (.Dback(Dback),
        .i_delta1(o_delta[21]),.i_delta2(o_delta[20]),.i_delta3(o_delta[19]),.i_delta4(i_delta),
        .o_delta(o_delta[5]),.wl(wl[5]),.sl(sum9),
        .al_u(al_d_2[5]),.al_d(al_d_3[5]),.bl_u(bl_d_2[5]),.bl_d(bl_d_3[5]));
RRAM_cell L2_43 (.Dback(Dback),
        .i_delta1(o_delta[21]),.i_delta2(o_delta[20]),.i_delta3(o_delta[19]),.i_delta4(i_delta),
        .o_delta(o_delta[1]),.wl(wl[5]),.sl(sum8),
        .al_u(al_d_3[5]),.al_d(al[5]),.bl_u(bl_d_3[5]),.bl_d(bl[5]));

RRAM_cell L2_14 (.Dback(Dback),
        .i_delta1(o_delta[18]),.i_delta2(o_delta[17]),.i_delta3(o_delta[16]),.i_delta4(i_delta),
        .o_delta(o_delta[12]),.wl(wl[4]),.sl(sum11),
        .al_u(al_u),.al_d(al_d_1[4]),.bl_u(),.bl_d(bl_d_1[4]));
RRAM_cell L2_24 (.Dback(Dback),
        .i_delta1(o_delta[18]),.i_delta2(o_delta[17]),.i_delta3(o_delta[16]),.i_delta4(i_delta),
        .o_delta(o_delta[8]),.wl(wl[4]),.sl(sum10),
        .al_u(al_d_1[4]),.al_d(al_d_2[4]),.bl_u(bl_d_1[4]),.bl_d(bl_d_2[4]));
RRAM_cell L2_34 (.Dback(Dback),
        .i_delta1(o_delta[18]),.i_delta2(o_delta[17]),.i_delta3(o_delta[16]),.i_delta4(i_delta),
        .o_delta(o_delta[4]),.wl(wl[4]),.sl(sum9),
        .al_u(al_d_2[4]),.al_d(al_d_3[4]),.bl_u(bl_d_2[4]),.bl_d(bl_d_3[4]));
RRAM_cell L2_44 (.Dback(Dback),
        .i_delta1(o_delta[18]),.i_delta2(o_delta[17]),.i_delta3(o_delta[16]),.i_delta4(i_delta),
        .o_delta(o_delta[0]),.wl(wl[4]),.sl(sum8),
        .al_u(al_d_3[4]),.al_d(al[4]),.bl_u(bl_d_3[4]),.bl_d(bl[4]));

//layer3
RRAM_cell L3_11 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(i_delta),.i_delta3(error[3]),.i_delta4(i_delta),
        .o_delta(o_delta[27]),.wl(wl[3]),.sl(sum7),
        .al_u(al_u),.al_d(al_d_1[3]),.bl_u(),.bl_d(bl_d_1[3]));
RRAM_cell L3_21 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(i_delta),.i_delta3(error[3]),.i_delta4(i_delta),
```

```
        .o_delta(o_delta[24]),.wl(wl[3]),.sl(sum6),
        .al_u(al_d_1[3]),.al_d(al_d_2[3]),.bl_u(bl_d_1[3]),.bl_d(bl_d_2[3]));
RRAM_cell L3_31 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(i_delta),.i_delta3(error[3]),.i_delta4(i_delta),
        .o_delta(o_delta[21]),.wl(wl[3]),.sl(sum5),
        .al_u(al_d_2[3]),.al_d(al_d_3[3]),.bl_u(bl_d_2[3]),.bl_d(bl_d_3[3]));
RRAM_cell L3_41 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(i_delta),.i_delta3(error[3]),.i_delta4(i_delta),
        .o_delta(o_delta[18]),.wl(wl[3]),.sl(sum4),
        .al_u(al_d_3[3]),.al_d(al[3]),.bl_u(bl_d_3[3]),.bl_d(bl[3]));

RRAM_cell L3_12 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(error[2]),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[26]),.wl(wl[2]),.sl(sum7),
        .al_u(al_u),.al_d(al_d_1[2]),.bl_u(),.bl_d(bl_d_1[2]));
RRAM_cell L3_22 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(error[2]),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[23]),.wl(wl[2]),.sl(sum6),
        .al_u(al_d_1[2]),.al_d(al_d_2[2]),.bl_u(bl_d_1[2]),.bl_d(bl_d_2[2]));
RRAM_cell L3_32 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(error[2]),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[20]),.wl(wl[2]),.sl(sum5),
        .al_u(al_d_2[2]),.al_d(al_d_3[2]),.bl_u(bl_d_2[2]),.bl_d(bl_d_3[2]));
RRAM_cell L3_42 (.Dback(Dback),
        .i_delta1(i_delta),.i_delta2(error[2]),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[17]),.wl(wl[2]),.sl(sum4),
        .al_u(al_d_3[2]),.al_d(al[2]),.bl_u(bl_d_3[2]),.bl_d(bl[2]));

RRAM_cell L3_13 (.Dback(Dback),
        .i_delta1(error[1]),.i_delta2(i_delta),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[25]),.wl(wl[1]),.sl(sum7),
        .al_u(al_u),.al_d(al_d_1[1]),.bl_u(),.bl_d(bl_d_1[1]));
RRAM_cell L3_23 (.Dback(Dback),
        .i_delta1(error[1]),.i_delta2(i_delta),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[22]),.wl(wl[1]),.sl(sum6),
        .al_u(al_d_1[1]),.al_d(al_d_2[1]),.bl_u(bl_d_1[1]),.bl_d(bl_d_2[1]));
RRAM_cell L3_33 (.Dback(Dback),
```

```
        .i_delta1(error[1]),.i_delta2(i_delta),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[19]),.wl(wl[1]),.sl(sum5),
        .al_u(al_d_2[1]),.al_d(al_d_3[1]),.bl_u(bl_d_2[1]),.bl_d(bl_d_3[1]));
RRAM_cell L3_43 (.Dback(Dback),
        .i_delta1(error[1]),.i_delta2(i_delta),.i_delta3(i_delta),.i_delta4(i_delta),
        .o_delta(o_delta[16]),.wl(wl[1]),.sl(sum4),
        .al_u(al_d_3[1]),.al_d(al[1]),.bl_u(bl_d_3[1]),.bl_d(bl[1]));

////////////////////////////////////////////////////////////////////////////////////////

// Error Calculator
RRAM_error i_error1(.Dlabel(Dlabel1),.sum(sum1),.error(error[1]));
RRAM_error i_error2(.Dlabel(Dlabel2),.sum(sum2),.error(error[2]));
RRAM_error i_error3(.Dlabel(Dlabel3),.sum(sum3),.error(error[3]));

// RRAM sigconv instantiation
RRAM_cell_sigconv L1_sig (
.Dwl({Dwl11,Dwl10,Dwl9,Dwl8}),
.Dsl({Dsl11,Dsl10,Dsl9,Dsl8}),
.Dbl({Dbl11,Dbl10,Dbl9,Dbl8}),
.Dset(Dset),
.wl(wl[11:8]),
.sl(sl[11:8]),
.al(al[11:8]),
.bl(bl[11:8]),
.sum({sum11,sum10,sum9,sum8})
);
RRAM_cell_sigconv L2_sig (
.Dwl({Dwl7,Dwl6,Dwl5,Dwl4}),
.Dsl({Dsl7,Dsl6,Dsl5,Dsl4}),
.Dbl({Dbl7,Dbl6,Dbl5,Dbl4}),
.Dset(Dset),
.wl(wl[7:4]),
.sl(sl[7:4]),
.al(al[7:4]),
.bl(bl[7:4]),
.sum({sum7,sum6,sum5,sum4})
```

```
);
RRAM_cell_sigconv L3_sig (
.Dwl({Dwl3,Dwl2,Dwl1,Dwl0}),
.Dsl({Dsl3,Dsl2,Dsl1,Dsl0}),
.Dbl({Dbl3,Dbl2,Dbl1,Dbl0}),
.Dset(Dset),
.wl(wl[3:0]),
.sl(sl[3:0]),
.al(al[3:0]),
.bl(bl[3:0]),
.sum({sum3,sum2,sum1,sum0})
);

analog begin
    V(al_u) <+ 0;
    V(i_delta) <+ 0;
end

endmodule
```

control

```
module control (clk, set, learn, Xin0, Xin1, Xin2, Xin3, Xin4, Xin5, Xin6,
    Dwl11,Dwl10,Dwl9,Dwl8,Dwl7,Dwl6,Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0,
    Dsl11,Dsl10,Dsl9,Dsl8,Dsl7,Dsl6,Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0,
    Dbl11,Dbl10,Dbl9,Dbl8,Dbl7,Dbl6,Dbl5,Dbl4,Dbl3,Dbl2,Dbl1,Dbl0,
    Dset,Dback,Dlabel0,Dlabel1,Dlabel2,write_en);

input clk, set, learn;
input Xin0, Xin1, Xin2, Xin3, Xin4, Xin5, Xin6;

output reg Dwl11,Dwl10,Dwl9, Dwl8, Dwl7, Dwl6, Dwl5, Dwl4, Dwl3, Dwl2, Dwl1, Dwl0;
output Dsl11, Dsl10, Dsl9, Dsl8;
output reg Dsl7, Dsl6, Dsl5, Dsl4, Dsl3, Dsl2, Dsl1, Dsl0;
output reg Dbl11,Dbl10,Dbl9, Dbl8, Dbl7, Dbl6, Dbl5, Dbl4, Dbl3, Dbl2, Dbl1, Dbl0;
output reg Dset, Dback;
output Dlabel0,Dlabel1,Dlabel2;
output reg write_en;
```

```
reg train_start;
reg update_start;
reg input_delay;
reg [1:0] pstate, nstate;
reg [2:0] update;
reg [15:0] trained_num;
parameter s0 = 2'b00;
parameter s1 = 2'b01;
parameter s2 = 2'b10;
parameter s3 = 2'b11;

parameter train_num = 1000; // set train repeat num

assign Dsl11 = Xin6;
assign Dsl10 = Xin5;
assign Dsl9 = Xin4;
assign Dsl8 = Xin3;
assign Dlabel2 = Xin2;
assign Dlabel1 = Xin1;
assign Dlabel0 = Xin0;

initial begin
    Dwl0 = 0;   Dwl1 = 0;   Dwl2 = 0;   Dwl3 = 0;   Dwl4 = 0;   Dwl5 = 0;
    Dwl6 = 0;   Dwl7 = 0;   Dwl8 = 0;   Dwl9 = 0;   Dwl10 = 0;   Dwl11 = 0;
    Dset = 0;   Dback = 0;
    // Dlabel0 = 0;   Dlabel1 = 0;   Dlabel2 = 0;
    Dbl0 = 0;   Dbl1 = 0;   Dbl2 = 0;   Dbl3 = 0;   Dbl4 = 0;   Dbl5 = 0;
    Dbl6 = 0;   Dbl7 = 0;   Dbl8 = 0;   Dbl9 = 0;   Dbl10 = 0;   Dbl11 = 0;
    Dsl0 = 0;   Dsl1 = 0;   Dsl2 = 0;   Dsl3 = 0;   Dsl4 = 0;   Dsl5 = 0;
    Dsl6 = 0;   Dsl7 = 0;
    //Dsl8 = 0;   Dsl9 = 0;   Dsl10 = 0;   Dsl11 = 0;
    // Transistor ON
    Dwl0 = 1;   Dwl1 = 1;   Dwl2 = 1;   Dwl3 = 1;   Dwl4 = 1;   Dwl5 = 1;
    Dwl6 = 1;   Dwl7 = 1;   Dwl8 = 1;   Dwl9 = 1;   Dwl10 =1;   Dwl11 = 1;
    pstate = s1;
    trained_num = 0;
```

```
        train_start = 0;
        update = 0;
        update_start = 0;
        write_en = 0;
        input_delay = 0;

    end

    always @(posedge clk) begin
        pstate <= nstate;
        input_delay <= write_en;

        if(update_start) update <= update + 1;


    end

    always @(pstate or set or learn or update or input_delay) begin
        case (pstate)
            // Initialization
            s0 : begin
                Dset <= 1;
                Dbl0 <= 1;    Dbl1 <= 1;    Dbl2 <= 1;    Dbl3 <= 1;    Dbl4 <= 1;    Dbl5 <=
1;
                Dbl6 <= 1;    Dbl7 <= 1;    Dbl8 <= 1;    Dbl9 <= 1;    Dbl10 <= 1;    Dbl11
<= 1;
                if(!set) nstate <= s1;
            end
            // Feed-foward
            s1 : begin
                Dset <= 0;
                Dbl0 <= 0;    Dbl1 <= 0;    Dbl2 <= 0;    Dbl3 <= 0;    Dbl4 <= 0;    Dbl5 <=
0;
                Dbl6 <= 0;    Dbl7 <= 0;    Dbl8 <= 0;    Dbl9 <= 0;    Dbl10 <= 0;    Dbl11
<= 0;
                //Dsl11 <= 1;    Dsl10 <= 1;    Dsl9 <= 1;    Dsl8 <= 1; // 4 input
                //Dsl11 <= Xin0;    Dsl10 <= Xin1;    Dsl9 <= Xin2;    Dsl8 <= Xin3; // 4 input
```

```
                update <= 0;
                if(learn) begin
                    train_start <= 1;
                end
                if(train_start)begin
                    write_en <= 1;
                end
                if(input_delay)begin
                    nstate <= s2;
                    write_en <= 0;
                end

                if(set)nstate <= s0;

            end
            // Error Calculation
            s2 : begin
                // Dlabel <= 1;      //input label
                nstate <= s3;
            end
            // Weight Update
            s3:
            begin
                case (update)
                    0 : begin
                        Dback <= 1;
                        Dbl0 <= 1;    Dbl1 <= 1;    Dbl2 <= 1;    Dbl3 <= 1;    Dbl4 <= 0;
Dbl5 <= 0;
                        Dbl6 <= 0;    Dbl7 <= 0;    Dbl8 <= 0;    Dbl9 <= 0;    Dbl10 <= 0;
Dbl11 <= 0;
                        update_start <= 1;
                    end
                    1 : begin
                        Dbl0 <= 1;    Dbl1 <= 1;    Dbl2 <= 1;    Dbl3 <= 1;    Dbl4 <= 1;
Dbl5 <= 1;
                        Dbl6 <= 1;    Dbl7 <= 1;    Dbl8 <= 0;    Dbl9 <= 0;    Dbl10 <= 0;
Dbl11 <= 0;
```

```
                end
                2 : begin
                    Dbl0 <= 1;    Dbl1 <= 1;    Dbl2 <= 1;    Dbl3 <= 1;    Dbl4 <= 1;
Dbl5 <= 1;
                    Dbl6 <= 1;    Dbl7 <= 1;    Dbl8 <= 1;    Dbl9 <= 1;    Dbl10 <= 1;
Dbl11 <= 1;
                end
                3 : begin
                    Dbl0 <= 0;    Dbl1 <= 0;    Dbl2 <= 0;    Dbl3 <= 0;    Dbl4 <= 0;
Dbl5 <= 0;
                    Dbl6 <= 0;    Dbl7 <= 0;    Dbl8 <= 0;    Dbl9 <= 0;    Dbl10 <= 0;
Dbl11 <= 0;

                end
                4 : begin
                    Dback <= 0;
                    nstate <= s1;
                    update_start <= 0;

                    if(train_num == (trained_num + 1)) begin
                    train_start <= 0;
                    trained_num <= 0;
                    end
                    else begin
                        trained_num <= trained_num + 1;
                    end
                end
            endcase

        end
        default : nstate <= s1;

    endcase


end

endmodule
```

RRAM_control

```
`include "disciplines.h"

module RRAM_control(clk, set, learn, reset);

electrical clk, set, learn;
electrical reset;
// electrical Xin0, Xin1, Xin2, Xin3, Xin4, Xin5, Xin6,
//    Dwl11,Dwl10,Dwl9,Dwl8,Dwl7,Dwl6,Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0,
//    Dsl11,Dsl10,Dsl9,Dsl8,Dsl7,Dsl6,Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0,
//    Dbl11,Dbl10,Dbl9,Dbl8,Dbl7,Dbl6,Dbl5,Dbl4,Dbl3,Dbl2,Dbl1,Dbl0,
//    Dset,Dback,Dlabel0,Dlabel1,Dlabel2,write_en;

control ctrl(clk, set, learn, Xin0, Xin1, Xin2, Xin3, Xin4, Xin5, Xin6,
      Dwl11,Dwl10,Dwl9,Dwl8,Dwl7,Dwl6,Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0,
      Dsl11,Dsl10,Dsl9,Dsl8,Dsl7,Dsl6,Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0,
      Dbl11,Dbl10,Dbl9,Dbl8,Dbl7,Dbl6,Dbl5,Dbl4,Dbl3,Dbl2,Dbl1,Dbl0,
      Dset,Dback,Dlabel0,Dlabel1,Dlabel2,write_en);


RRAM_unit Rut(
      Dwl11,Dwl10,Dwl9,Dwl8,Dwl7,Dwl6,Dwl5,Dwl4,Dwl3,Dwl2,Dwl1,Dwl0,
      Dsl11,Dsl10,Dsl9,Dsl8,Dsl7,Dsl6,Dsl5,Dsl4,Dsl3,Dsl2,Dsl1,Dsl0,
      Dbl11,Dbl10,Dbl9,Dbl8,Dbl7,Dbl6,Dbl5,Dbl4,Dbl3,Dbl2,Dbl1,Dbl0,
      Dset,Dback,Dlabel0,Dlabel1,Dlabel2);

input_allocation ia(clk, reset, write_en, Xin6, Xin5, Xin4, Xin3, Xin2, Xin1, Xin0);

endmodule
```

input_allocation

```
module input_allocation (clk, reset, write_en, Xout6, Xout5, Xout4, Xout3, Xout2, Xout1,
Xout0);

      input clk, reset, write_en;
      output Xout6, Xout5, Xout4, Xout3, Xout2, Xout1, Xout0; //4bit Input allocation
```

```
// reg first;
reg [6:0] x_memory[0:999]; //save 6bit * 1152 Input
integer i = 0;
reg [6:0] Xout;
// integer j = 0;

initial begin
    $readmemb("traindata.txt", x_memory);
    Xout <= 0;

end

assign Xout0 = Xout[0]; //label0
assign Xout1 = Xout[1]; //label1
assign Xout2 = Xout[2]; //label2
assign Xout3 = Xout[3]; //input0
assign Xout4 = Xout[4]; //input1
assign Xout5 = Xout[5]; //input2
assign Xout6 = Xout[6]; //input3

always @(posedge clk) begin
    if (reset)begin
        Xout   <= 0;
        // first <= 1;
    end
    else begin
        if (write_en == 1) begin

            // if (first) begin
            //     first <= 0;
            //     $readmemh("FC_1_i.txt",x_memory,0,1151);
            // end



                //output neuron input allocation end
                //neuron num : 1152
```

```
                    if (i == 999)begin
                        //i = 0;
                        Xout <= 0;
                    end
                    else begin
                        Xout <= x_memory[i];
                        i    <= i + 1;
                    end

                end
            end


    end
endmodule
```

tb_RRAM_control

```
//`include "control.v"
`timescale 1ns/1ns
//`default_nettype none

module tb_RRAM_control(clk, set, learn, reset);

output reg clk, set, learn, reset;
//input [7:0] train_num;
//output reg train_num;
// output reg Dwl5, Dwl4, Dwl3, Dwl2, Dwl1, Dwl0;
// output reg Dsl5, Dsl4, Dsl3, Dsl2, Dsl1, Dsl0;
// output reg Dbl5, Dbl4, Dbl3, Dbl2, Dbl1, Dbl0;
// output reg Dset, Dback, Dlabel;


localparam CLK_PERIOD = 10;
always #(CLK_PERIOD/2) clk=~clk;

// initial begin
//         $dumpfile("tb_control.vcd");
```

```
//        $dumpvars(0, tb_control);
// end

initial begin
      clk = 0;
      learn = 0;
      set = 0;
      reset = 1;
      //train_num = 0;

      #20
      set = 1;
      reset = 0;
      #10
      set = 0;
      #20
      learn = 1;
      //train_num = 5;
      //train_num = 1;
      #10
      learn = 0;
      #100000

      $finish(2);
end

endmodule
```

RRAM_control.cmd

```
*
.MODEL a2d_default A2D MODE=std_logic vth=0.5
.MODEL d2a_default D2A MODE=std_logic vlo=0.0 vhi=1.0 trise=1p tfall=1p
.DEFHOOK a2d_default d2a_default

.MODEL RRAM_unit_tb MACRO LANG=veriloga LIB=RRAM_lib
.MODEL RRAM_unit MACRO LANG=veriloga LIB=RRAM_lib
```

Y_RRAMstim tb_RRAM_control
+PORT: clk set learn reset

Y_RRAMcell RRAM_control
+PORT: clk set learn reset

.TRAN 0 100000ns
.PLOT TRAN V(clk) V(set) V(learn) V(reset)
.end

Python과 Verilog 결과 비교

Python의 경우 1000번의 forward와 back을 반복한 후 마지막 데이터에 대해

```
After Train:
[[0.42946167 0.57648074 0.51779809]]
Real Label:
[[0. 0. 1.]]
```

와 같은 결과를 도출했다.

Verilog의 경우 0.43016213 V, 0.51214874 V, 0.51748353 V의 결과를 도출했다. 약간의 값 차이가 있으나 이는 python과 verilogа가 사용하는 근사 방식 등이 달라서 생긴 문제라고 해석하였다. 이러한 작은 오차를 배제하고 모든 train data에 대하여 weight와, 그에 따른 최종 predict 결과의 전체적인 경향성은 동일하게 나왔다.

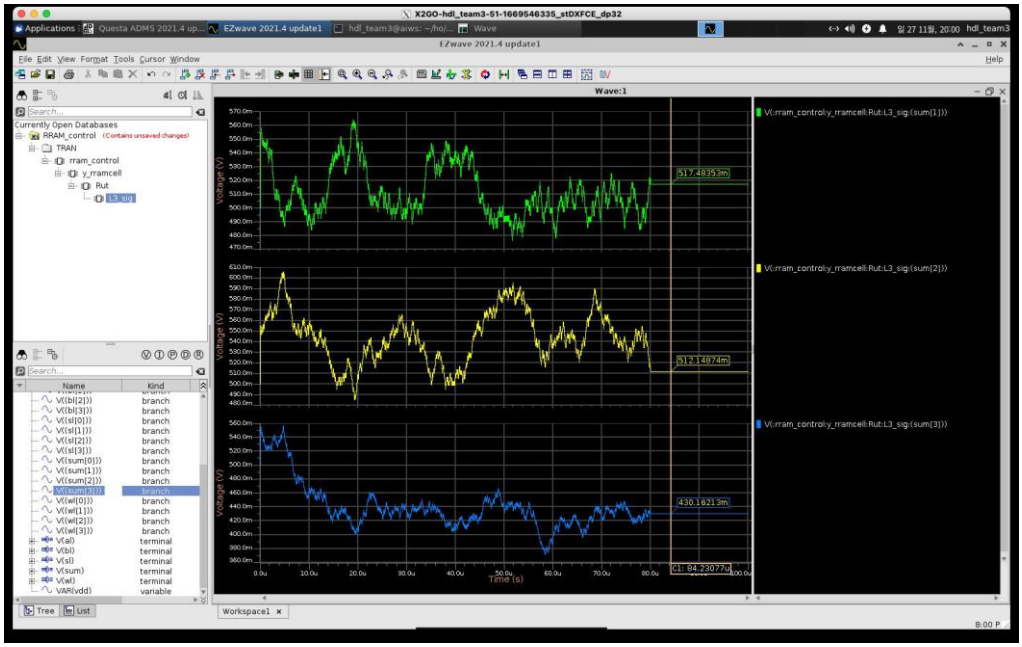


Fig 17