MNIST Dataset을 위한 Convolution Neural Net의 Verilog 구현

김정현

1. 설계 목표

2. Python Pytorch Coding

3. Verilog Coding

4. 결과 및 한계

1. 설계 목표

MNIST Dataset에 대해 95% 이상의 적중률을 보이는 CNN의 Verilog 설계

가정
1. 한 클럭 당 하나의 픽셀 값이 입력된다.
2. 여러 이미지를 연속으로 입력하지 않는다.

1. 설계 목표

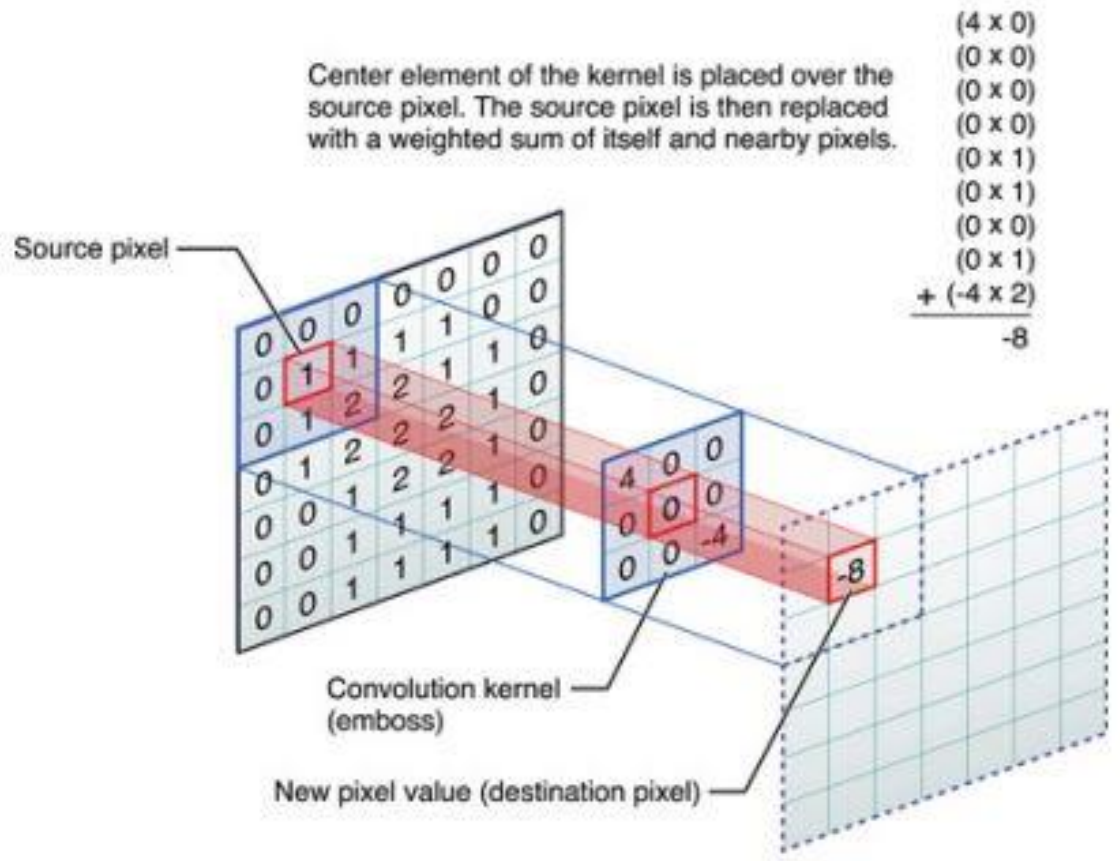
- 2D Convolution

- 입력 영상에 n*n (n은 홀수) Kernel (= filter, mask)을 좌측 상단부터 맞대어 겹쳐진 값을 모두 곱한 뒤 더해 출력으로 보낸다. 그 후 정한 stride 만큼 (보통 1) Kernel을 오른쪽으로 움직이며 값을 계산해 나간다.

- 출력 영상의 크기는 입력 영상에 비해 너비, 높이가 (n – 1) 씩 줄어든다.

- 필요시 출력의 크기를 입력의 크기와 같게 하기 위해 입력 주위를 0으로 덧댄 후 계산하는 zero padding을 해준다.

- CNN에선 Kernel 의 값이 Weight로써 학습된다. 또한, 그 값에 Bias를 더해 주어야 한다.



2. Python Pytorch Coding

1) CNN Structure

2) Training Optimization

3) Verilog 설계 시 필요한 값 추출

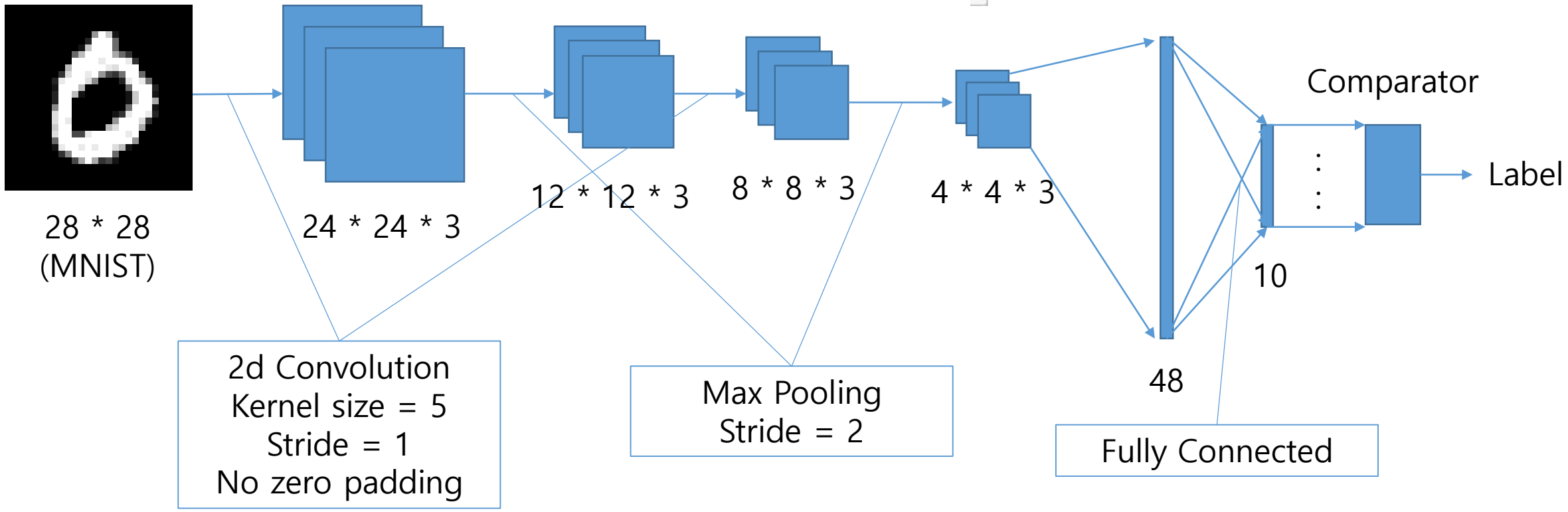
2. Python Pytorch Coding

- CNN Structure

적중률이 95% 이상인 선에서 가장 용량이
작은 구조 선택

```
Train Epoch: 9 [51200/60000 (85%)]      Loss: 0.107662
Train Epoch: 9 [51840/60000 (86%)]      Loss: 0.141438
Train Epoch: 9 [52480/60000 (87%)]      Loss: 0.133668
Train Epoch: 9 [53120/60000 (88%)]      Loss: 0.075324
Train Epoch: 9 [53760/60000 (90%)]      Loss: 0.144535
Train Epoch: 9 [54400/60000 (91%)]      Loss: 0.154840
Train Epoch: 9 [55040/60000 (92%)]      Loss: 0.059224
Train Epoch: 9 [55680/60000 (93%)]      Loss: 0.196860
Train Epoch: 9 [56320/60000 (94%)]      Loss: 0.247110
Train Epoch: 9 [56960/60000 (95%)]      Loss: 0.168821
Train Epoch: 9 [57600/60000 (96%)]      Loss: 0.072246
Train Epoch: 9 [58240/60000 (97%)]      Loss: 0.148895
Train Epoch: 9 [58880/60000 (98%)]      Loss: 0.068442
Train Epoch: 9 [59520/60000 (99%)]      Loss: 0.257442

Test set: Average loss: 0.1295, Accuracy: 9587/10000 (95%)
```



2. Python Pytorch Coding

- Training Optimization

- Optimizer : SGD(+Momentum) (learning rate = 0.01, momentum = 0.5)
- Loss Function : Negative Log Likelihood
- Weight Initialization : Xavier initialization (Pytorch 2dConv default)
- Activation Function : ReLU (Rectified Linear Unit)

Activation Function 이외의 요소는
HW 설계에 영향을 주지 않음

2. Python Pytorch Coding

- Verilog 설계 시 필요한 값 추출

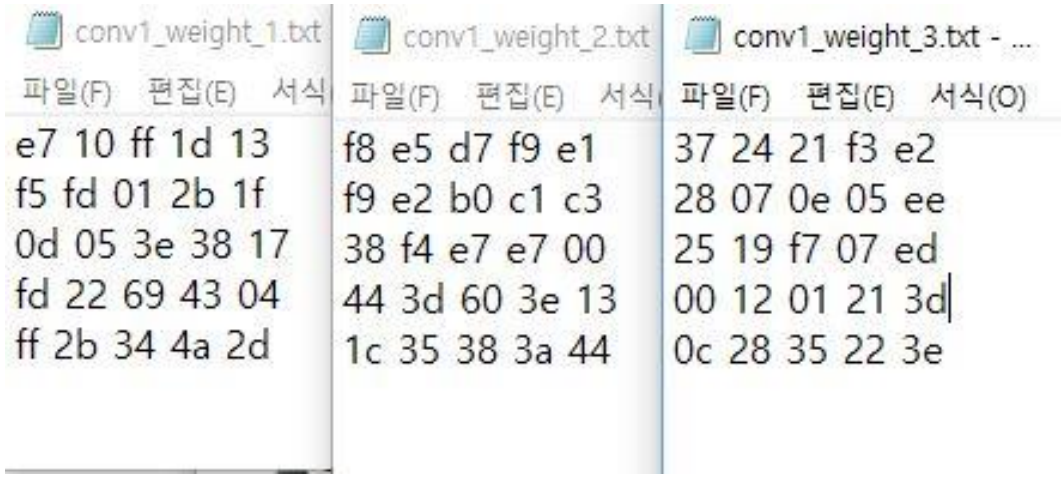
1. Verilog 설계를 위한 신경망의 각 Weight, Bias 값 추출

1) Weight, Bias에 2의 제곱수를 곱해준다. (할당 비트 수에 맞춤)
(-1,1) → (-a, a) (a = 2^n, n = 7~)

2) 소수점 밑을 절사한다. (float → int)

3) 2의 보수를 취한다.
→ 음수일 경우 2^(n+1)을 더해준다.

4) 16진수로 출력



2. Python Pytorch Coding

- Verilog 설계 시 필요한 값 추출

2. Verilog 설계 검증을 위한 각 Layer 통과 값 추출

1) MNIST Dataset 한 장을 신경망에 통과시켜
각 층의 출력 값을 따로 저장한다.

2) 각 출력 값에 2의 제곱수를 곱해준다.
(Weight, Bias 추출 시 곱한 값과 동일)

3) 본인이 보기 쉽게 출력
ex) 소수점 밑을 절사한다

```
######################## 신경망 통과 값 추출 ########################
print(np.shape(model.conv1_out_np))
np.savetxt('out_conv1_value_1.txt', model.conv1_out_np[0][0]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_conv1_value_2.txt', model.conv1_out_np[0][1]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_conv1_value_3.txt', model.conv1_out_np[0][2]*128, fmt='%1.5d',delimiter = " ")

print(np.shape(model.mp1_out_np))
np.savetxt('out_mp1_value_1.txt', model.mp1_out_np[0][0]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_mp1_value_2.txt', model.mp1_out_np[0][1]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_mp1_value_3.txt', model.mp1_out_np[0][2]*128, fmt='%1.5d',delimiter = " ")

print(np.shape(model.conv2_out_np))
np.savetxt('out_conv2_value_1.txt', model.conv2_out_np[0][0]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_conv2_value_2.txt', model.conv2_out_np[0][1]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_conv2_value_3.txt', model.conv2_out_np[0][2]*128, fmt='%1.5d',delimiter = " ")

print(np.shape(model.mp2_out_np))
np.savetxt('out_mp2_value_1.txt', model.mp2_out_np[0][0]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_mp2_value_2.txt', model.mp2_out_np[0][1]*128, fmt='%1.5d',delimiter = " ")
np.savetxt('out_mp2_value_3.txt', model.mp2_out_np[0][2]*128, fmt='%1.5d',delimiter = " ")

print(np.shape(model.fc_in_np))
np.savetxt('fc_in_value.txt', model.fc_in_np*128, fmt='%1.5d',delimiter = " ")

print(np.shape(model.fc_out_np))
np.savetxt('fc_out_value.txt', model.fc_out_np*128, fmt='%1.5d',delimiter = " ")
```

3. Verilog Coding

1) 전체 구조

2) Basic 2D Convolution Module

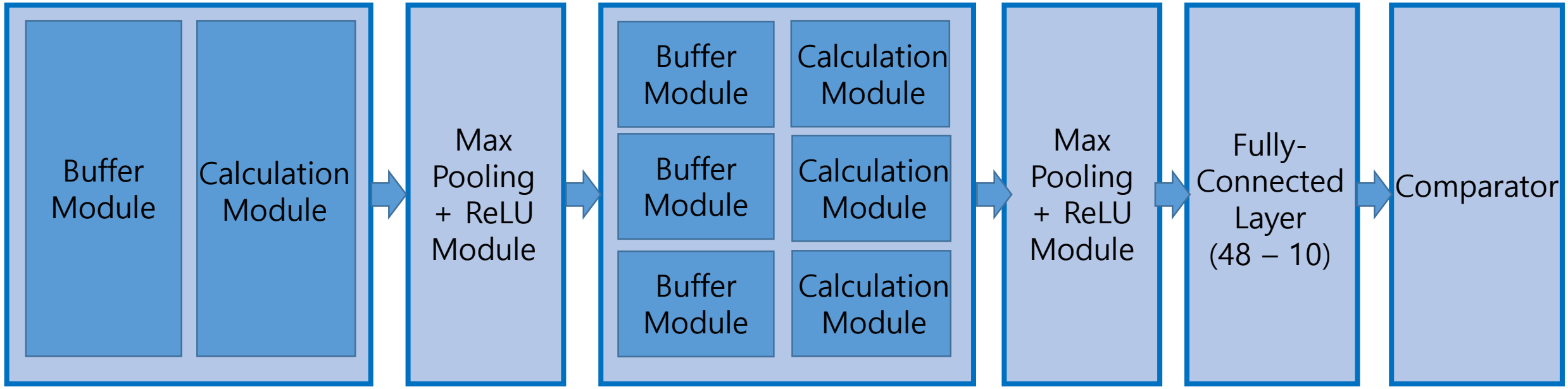
3) 2D Convolution with Multi-Channel

4) Max Pooling + ReLU (Rectified Linear Unit)

5) Fully-Connected Layer

6) Comparator

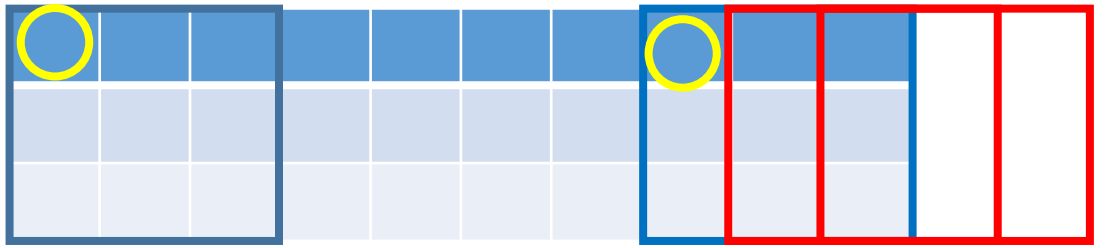
3. Verilog Coding



- 각 단계 공통으로 valid input, output signal 존재
  valid input signal 이 1일 때만 동작하여 유효하지 않은 값 출력 시 valid output signal 0 값 출력
  → 전 단계의 결과 값이 끊겨 들어와도 상관 없게 된다.

- 전 단계 계산 결과 값은 12-Bit로 동일
  계산 결과를 밑에서부터 잘라내어 Bit수 감소

3. Verilog Coding



- Basic 2D Convolution Module

2D Convolution Module은 Buffer Module, Calculation Module 로 구성

1) Buffer Module
Convolution 연산을 위해선 (영상 너비 * Kernel 높이) 크기의 버퍼 필요

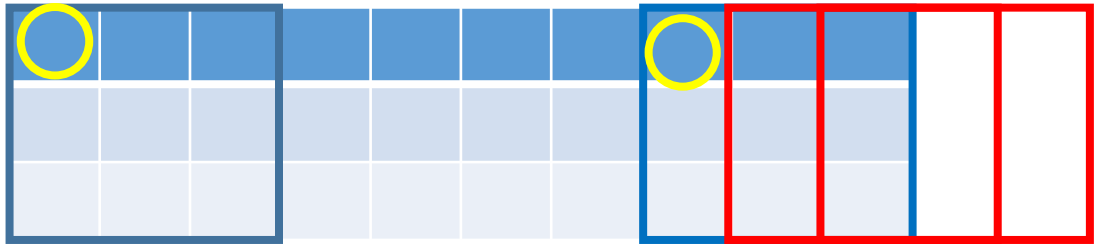
초기화 단계(Buffer를 pixel로 채움)와 출력 단계(pixel로 채움과 동시에 출력) 로 나뉜다.

단계 상관없이 pixel은 계속 왼쪽에서 오른쪽, 위에서 아래로 채우는 것을 반복

출력 단계에서 클럭 마다 Kernel과 같은 크기의 사각형을 왼쪽부터 오른쪽으로 반복해서 움직이며 범위에 포함된 값들을 출력으로 보낸다.

빨간 영역 도달 시 의미 없는 값 출력 (Pixel을 Buffer에 넣기 위해 대기 시간 필요)

3. Verilog Coding



- Basic 2D Convolution Module

출력 시 줄 단위로 출력 순서를 바꿔 주어야 한다.

단위시간	1	2	3	4	5	6	7	8	9
1st line	1	4	4	4	7	7	7	10	10
2nd line	2	2	5	5	5	8	8	8	11
3rd line	3	3	3	6	6	6	9	9	9
1st 출력 line	1	2	3	1	2	3	1	2	3
2nd 출력 line	2	3	1	2	3	1	2	3	1
3rd 출력 line	3	1	2	3	1	2	3	1	2

2) Calculation Module

Buffer Module에서 받은 입력 값을 Weight 값 각각과 곱한 뒤 더하여 (+ Bias) 출력한다.
입력 채널이 1, 출력 채널이 3일 경우 Kernel이 3개 필요, 각각 계산하여 3개 값을 출력한다.

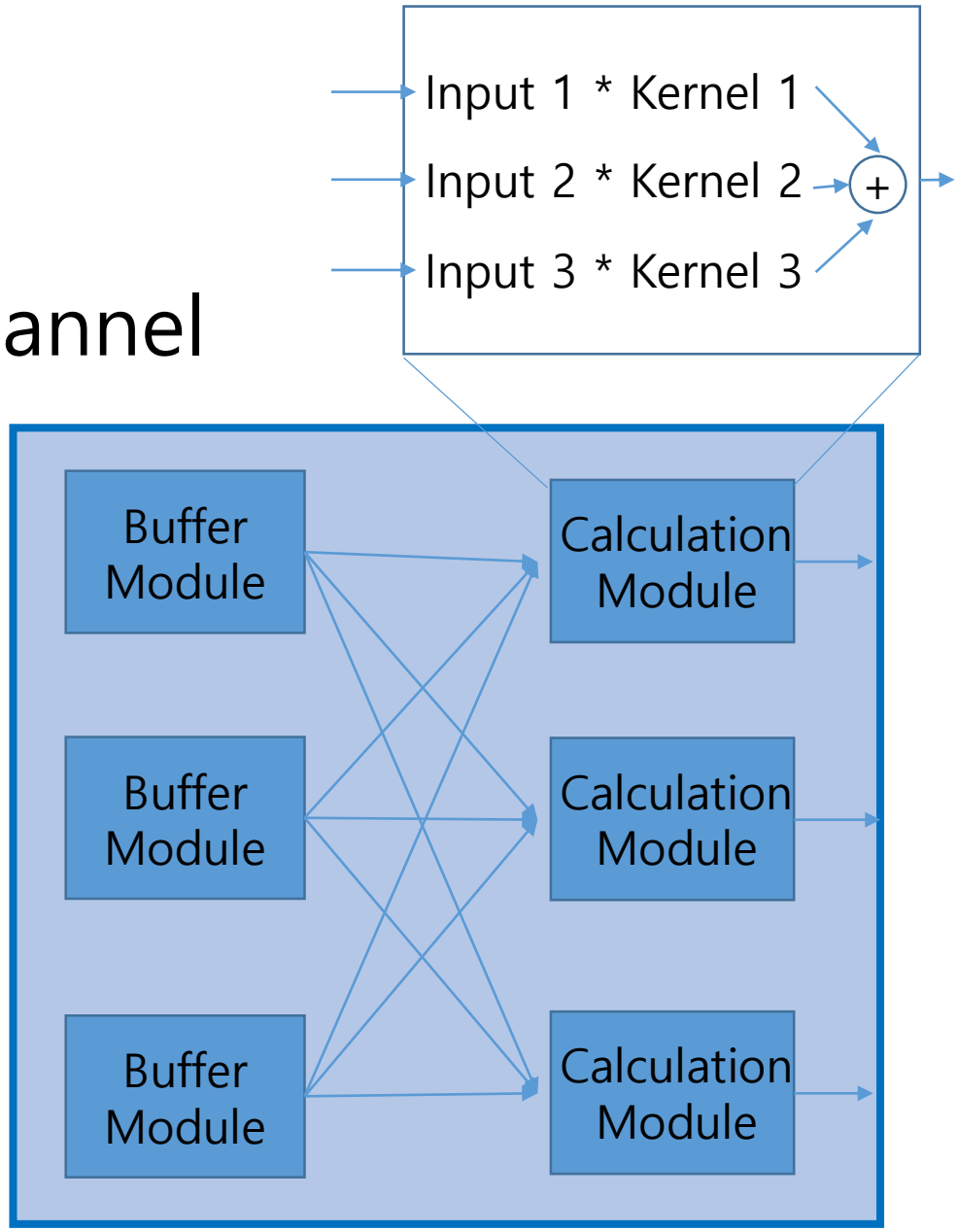
3. Verilog Coding

- 2D Convolution with Multiple Channel

입력 채널이 N, 출력 채널이 M 인 경우
N * M 개의 Kernel이 필요하다.

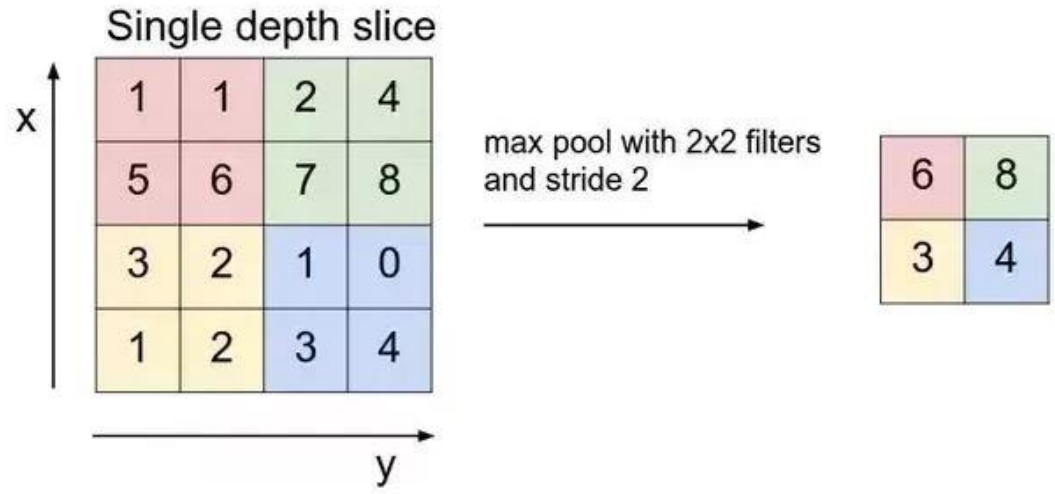
N 개의 Buffer Module,
M 개의 Calculation Module 필요

각 Calculation Module 은 N개의 Kernel을 가지고 있으며, N개의 Buffer Module 출력 전체를 받아 각각을 Kernel에 곱한 뒤 더하여 (+Bias) 출력한다.



3. Verilog Coding

- Max Pooling + ReLU Module

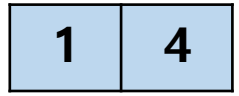


1 * (WIDTH / 2) 크기의 Buffer 요구
왼쪽 위부터 값이 순서대로 들어올 때

1) 첫번째 값을 Buffer에 넣는다.
state == 0, flag == 0

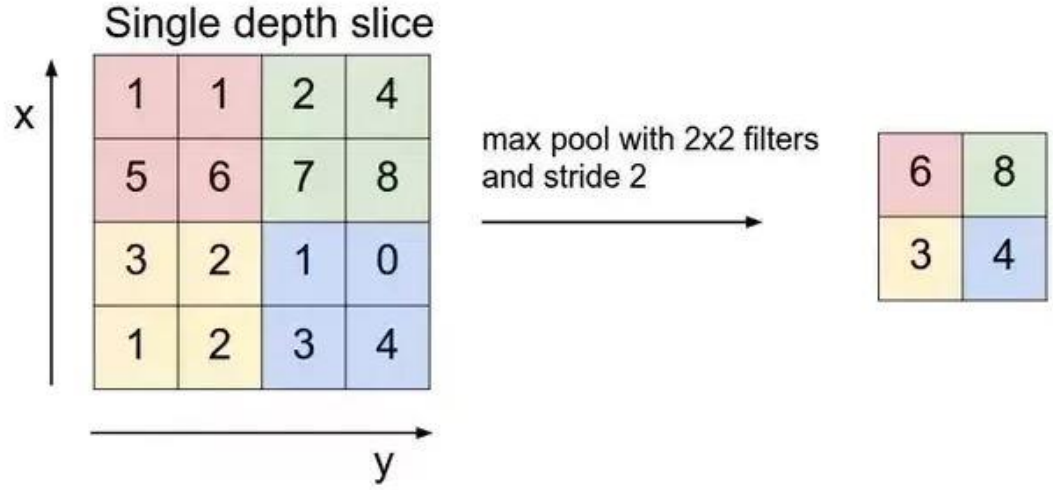
2) 두번째 값을 Buffer의 값과 대소 비교 후 크면 교체
state == 0, flag == 1, Index++

3) 한 줄에 대해 반복하면 (WIDTH / 2) 크기 Buffer가 모두 채워진다.
flag 0, 1 반복, Index = 0



3. Verilog Coding

- Max Pooling + ReLU Module



4) 그 다음 줄 첫번째 값을 Buffer의 값과 대소 비교 후 입력
state == 1, flag == 0

5	4

5) 두번째 값도 비교 후 입력
state == 1, flag == 1 => Valid Output == 1
Index++

6	4

→ 이때 해당 Index의 Buffer 출력 값 6이 유효 출력

6) 나머지 반복
state == 1, flag, Valid Output 0, 1 반복

6	8

→ 이때 해당 Index의 Buffer 출력 값 8이 유효 출력

7) 1)부터 반복 (state == 0, flag == 0, Index = 0)

3	4

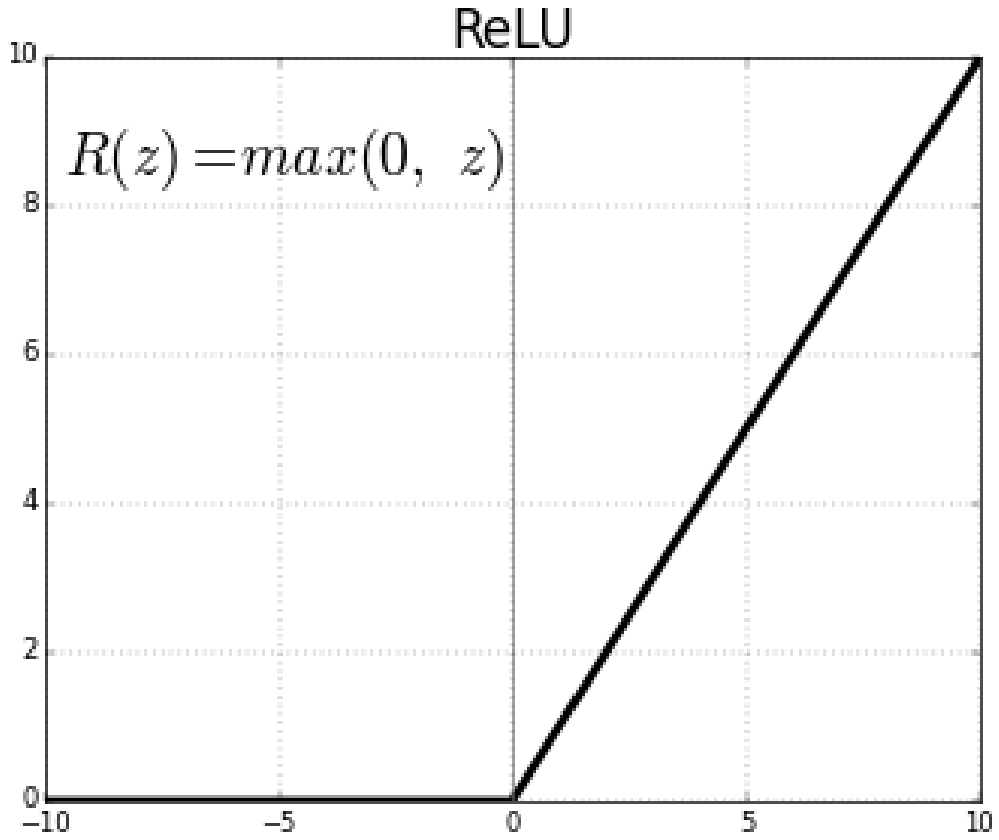
3. Verilog Coding

- Max Pooling + ReLU Module

계산 순서 상
Convolution – ReLU – MaxPooling 이지만

Convolution – MaxPooling – ReLU 로
바꾸어도 결과는 같다.

ReLU : Max Pooling 의 Buffer 값 출력 시
음수일 경우 0을 출력 한다.

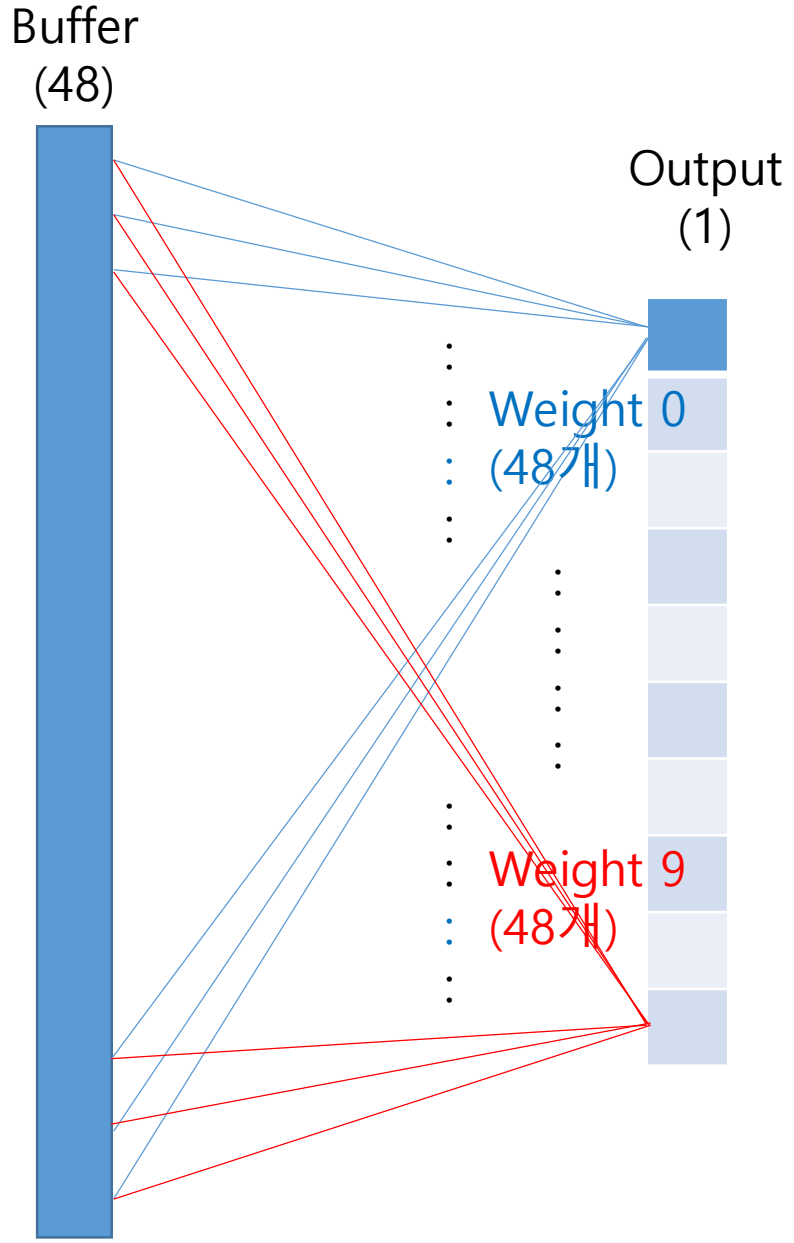


3. Verilog Coding

- Fully-Connected Layer Module

- Flattening
 앞의 Max Pooling Module에서 3개의 출력 값이 16 클럭 동안 나오기 때문에 한번에 계산하기 위해서 Buffer가 필요하다.

- Calculation
 Buffer를 채운 뒤 10 클럭 동안 480개의 Weight (48 * 10) 중 48개씩 곱해서 10개의 결과값을 하나씩 내보내게 된다.



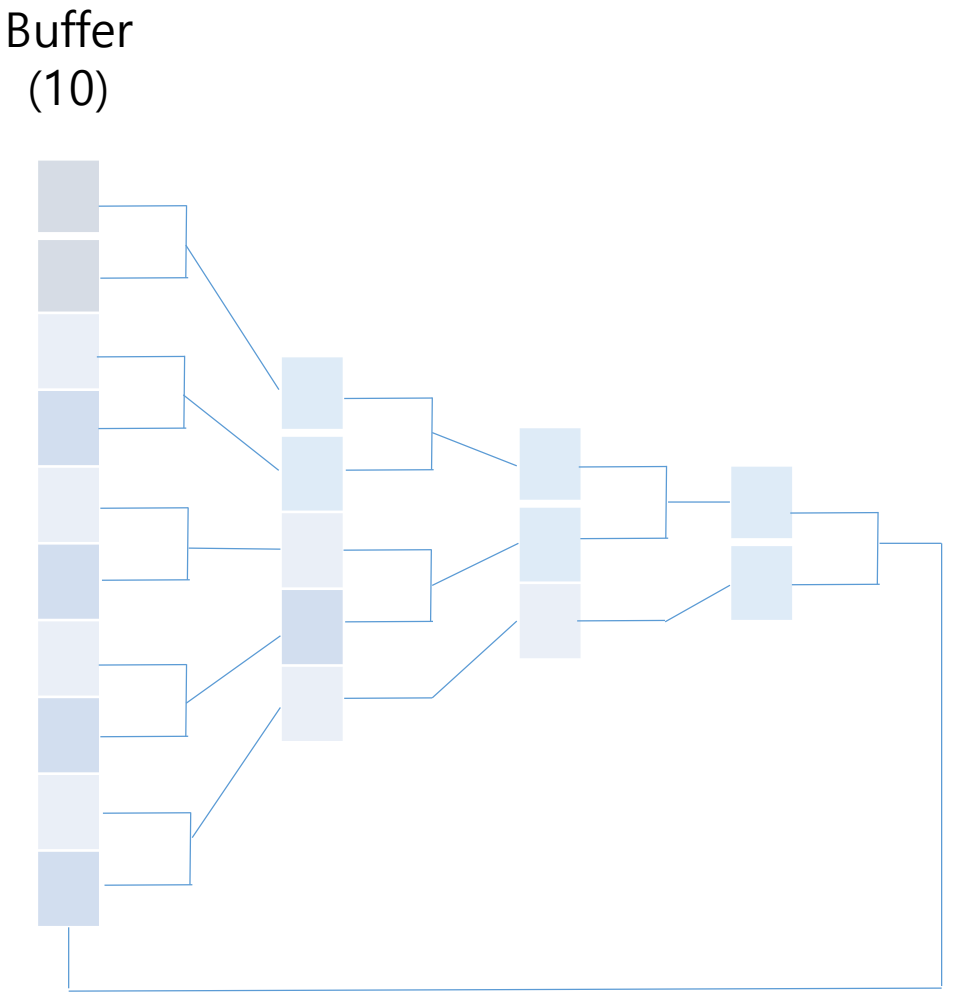
3. Verilog Coding

- Comparator Module

앞의 Module에서 값이 하나씩 들어오기 때문에
10개 값을 담는 Buffer가 필요하다.

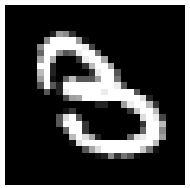
한 클럭 당 토너먼트 식으로
10 – 5 – 3 – 2 개의 값을 비교하게 되며

그 중 남은 최댓값을 Buffer내의 값과
비교하여 같은 값을 갖는 Buffer의 주소를
Decision으로 출력하게 된다.



4. 결과 및 한계

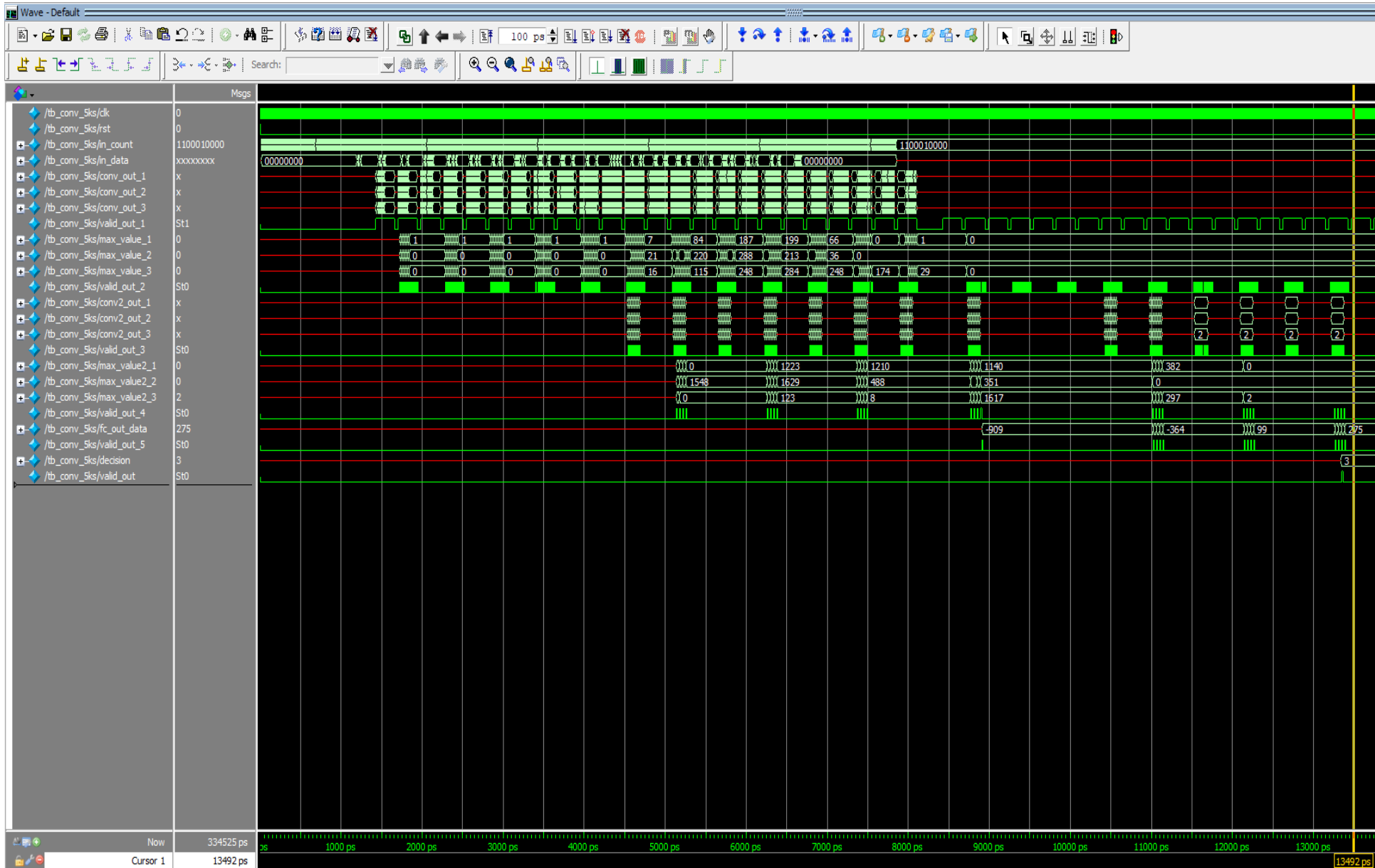
ModelSim Simulation Result



한 장 입력 결과

총 1335 클럭 소요

100MHz 동작 시
1.335μs 소요

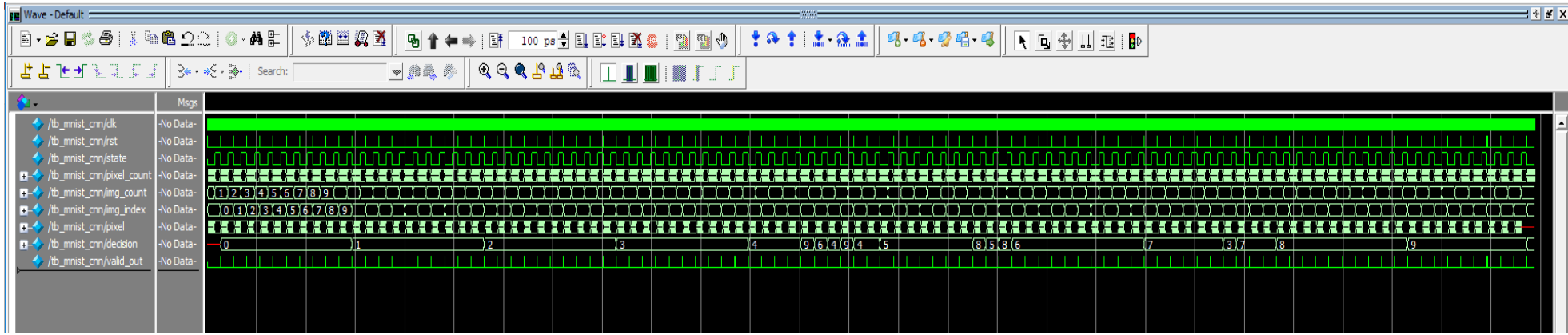


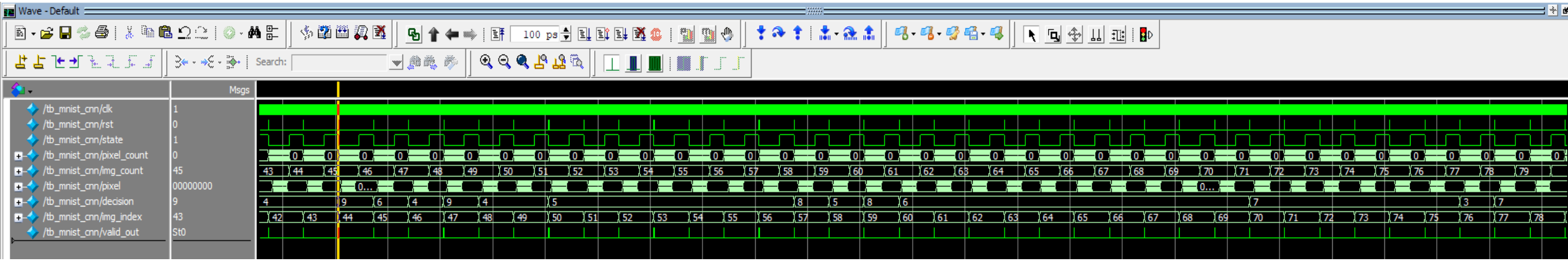
4. 결과 및 한계

ModelSim Simulation Result

100장 입력 결과

적중률 94 / 100 (94%)





오답 분류 값

4. 결과 및 한계

- 한계점 및 개선 방향

- 각 장에 대한 Pipelining 불가능
  각 단계별로 Buffer를 둔 Pipelining 고려

- 실제 입출력 고려 X
  CAM 입력, VGA(or HDMI) 출력 염두에 둔 설계로 FPGA Test 고려

- 많은 양의 Adder, Multiplier 요구
  Adder, Multiplier를 줄이고 클럭이나 버퍼를 늘리는 설계 및
  Multiplier를 줄이기 위한 Winograd Convolution 고려

- Weight 저장량 과다
  Weight 당 Bit 수를 낮추거나 더 작은 용량을 갖는 모델 설계 고려