붙임3

연구논문/작품 최종보고서

2021학년도 제2학기

제목 : Verilog를 이용한 Convolution Neural Network 구현

강보영(2017312548)

2021년 10월 22 일

지도교수: 서명

계획(10)	주제(20)	개념(20)	상세(30)	보고서(20)	총점(100)

* 지도교수가 평가결과 기재

■ 요약

최근 인간의 뇌를 모사한 뉴로모픽(Neuromorphic) 기술이 차세대 반도체 기술로 주목되고 있다. 뉴로모픽 칩은 기존 반도체 칩의 데이터 처리 과정을 통합하여 기억과 병렬연산을 대량으로 진행하여 기존 폰 노이만(Von-Neumann) 구조의 병목현상(Bottleneck)을 해결한다. 본 연구작품에서는 신경망을 통한 학습 방법 중 하나인 합성곱 신경망(Convolutional Neural Network)을 verilog 언어를 사용하여 하드웨어로 설계한다. 설계한 CNN을 통해 MNIST 손글씨 데이터셋 [11] 입력에 대한 학습을 진행하고, 소요되는 시간과 인식률을 확인해 볼 것이다. 설계 검증을 위해 파이토치(PyTorch) 프레임워크 [12]에서 MNIST 데이터셋 [11]에 대한 학습을 진행하여 추출한 가중치와 편향값을 모듈 내에서 사용한다.

Keyword : CNN, Verilog, PyTorch, MNIST, Deep Learning

■ 서론

1) 제안배경 및 필요성

오늘날 인공지능이 사람의 인식에 매우 근접하다고 평가되며, 이미 다양한 분야에서 상용화되어 있다. 특히, 코로나19 팬데믹으로 AI 기술의 개발과 도입이 매우 활발한 상황이다. 인공지능이란 뇌과학에서의 신경망(Neural Network)을 프로그램으로 구현한 것이다. 뉴로모픽 기술은 초저전력으로 동작하는 인간 뇌의 주요 기능을 모방하여 기존 폰 노이만 구조의 한계를 극복하기 위한 기술이다. 폰 노이만 구조는 CPU와 메모리가 분리되어 있어 데이터의 저장과 연산이 다른 공간에서 수행되기 때문에 병목현상이 발생하고, 데이터의 입/출력을 한 번에 하나씩 직렬로 진행해야 한다 [1]. 하지만 뉴로모픽 구조는 인간의 뇌와 같이 데이터 저장과 연산을 함께 처리하기 때문에 다양한 데이터 입/출력을 동시에 수행할 수 있다.

이러한 뉴로모픽 기술은 방대한 양의 컴퓨팅 자원 및 전력 소모를 요구하는 기존의 딥러닝 모델을 경량화할 수 있는 해결책으로 생각할 수 있다. 현재까지는 뉴로모픽 반도체 칩은 회로로 설계한 뉴런과 시냅스의 집적도가 낮아 칩의 면적이 크고, 가격이 비싸 상용화되지 않았다. 하지만, AI를 빼고는 정의 내릴 수 없는 오늘날 반도체 시장에서 뉴로모픽 기술에 대한 연구 개발이 활발하게 이루어지고 있으며, 향후 IoT 디바이스, 웨어러블 디바이스, 자율주행 자동차, 인지로봇 등 다양한 응용분야에서 적용될 수 있을 것으로 기대된다.

인간의 신경 구조를 CMOS 집적회로 기술 기반의 하드웨어로 구현한 것을 뉴로모픽 기술이라고 한다. 인간의 뇌는 천억 개가 넘는 신경 세포, 즉 뉴런이 시냅스

라는 연결고리를 통해 다른 뉴런과 상호 간에 신호를 주고받으며 데이터를 처리하고 저장한다. 즉, 외부에서 입력된 자극을 전기적 신호로 전환하는 뉴런(neuron)과 전기적 신호를 인접한 뉴런으로 전달하면서 신호 전달 능력을 의미하는 시냅스 가중치(weight)를 변화시켜 적응 학습 및 기억 기능을 수행한다. 따라서 데이터의 연산과 저장을 같은 장소에서 진행하기 때문에 저전력으로 고속 동작이 가능하다. 시냅스와 뉴런의 다양한 기능들을 전기적인 소자로 모방하기 위해서는 스파이크 신호 기반 인공신경망(Spiking Neural Network)을 구현하는 것이 주가 된다.

2) 연구작품의 목표

본 연구작품에서는 소프트웨어 기반 신경망 학습 기법 중 하나인 합성곱 신경망(Convolutional Neural Network, CNN)을 verilog 언어를 사용하여 RTL level로 설계하고 검증하는 것을 목표로 한다. 최종 목표는 94% 이상의 적중률을 가진 Convolutional Neural Network를 설계하는 것이다. 설계를 검증하기 위해 단일 입력이미지와 연속 입력 이미지에 대한 테스트벤치를 설계하여 인식률 및 소요 시간을 측정할 것이다.

3) 연구작품 전체 개요

본 연구작품은 크게 두 단계로 나누어 진행된다.

1. 파이토치(PyTorch) 기반 딥러닝 데이터셋 학습

PyTorch란 파이썬 기반의 딥러닝 프레임워크로 데이터에 대한 딥러닝 분석을 쉽게 할 수 있도록 제공하는 오픈소스다 [12]. 본 연구작품의 학습 대상인 MNIST 데이터셋 [11]을 이용하여 학습을 진행한다. MNIST란 손으로 쓴 글자 데이터셋에서 숫자만 따로 뽑아낸 데이터셋으로 0부터 255까지의 값을 갖는 흑백 이미지로 구성되어 있다. 이미지 크기는 28*28로 고정되어 있고, 총 60,000개의 training set로 구성되어 있다. MNIST 데이터셋을 로드하여 학습을 진행하고, 94% 이상의 적중률을 갖는 CNN 모델을 결정한다. 이후, verilog 설계를 위한 가중치와 편향값을 텍스트 파일로 추출하고 값을 보정하여 저장하도록 한다. 또한, 설계 검증을 위해 각 layer의 output data도 마찬가지로 저장한다.

2. Convolution Neural Network의 RTL level 설계

설계는 Mentor사의 Modelsim 툴 [10]을 사용하여 verilog 언어로 설계된다. CNN의 구조에 따라 convolution layer, max pooling layer, activation function, fully connected layer 등의 sub-block으로 나누어 RTL 설계를 진행한다. 더하여, 해당 RTL을 검증하기 위해 testbench를 작성하고 동작을 확인한다. 최종적으로 시뮬레이

션 waveform을 통해 MNIST 이미지 데이터 입력에 대해 어느 정도의 적중률을 가지며, 소요되는 클럭 수는 얼마인지 측정하도록 한다.

이어지는 관련 연구 파트에서는 현재까지 진행된 다양한 선행연구 분석을 통해 인공지능기술과 CNN 개발 및 연구 현황에 대해 알아보고, 더 나아가 본 연구작품의 진행 방향을 살펴보도록 한다. 제안 작품 소개 파트에서는 합성곱 신경망의 개념과 원리에 대해 알아보고, 프레임워크 및 툴 환경 설정에 관해 설명한다. PyTorch에서 MNIST 데이터셋에 대한 학습을 진행한 결과 및 데이터 추출, 입력데이터 생성에 대해 설명한다. 또한, verilog 설계 block diagram과 각 sub module에 대해 간략히 설명한다. 구현 및 결과분석 파트에서는 각 sub module의 input/output port와 파라미터 명세에 대해 설명한다. 작성된 소스 코드의 핵심이 되는 부분을 설명/분석한다. Modelsim 툴을 사용하여 시뮬레이션한 결과 waverform을 보이고, 그에 대한 분석을 한다. 결론 및 소감에서는 연구작품 결과에 대한 고찰 및 분석을 하고 본 연구의 한계점과 개선사항에 대해 설명하고, 간략한 후기와 참고문헌으로 본 연구작품의 최종보고서를 마친다.

■ 관련연구

딥러닝(Deep Learning)은 오늘날 산업 전반에 걸쳐 큰 혁명을 일으켰다. AI(Artificial Intelligence)와 DL는 비슷하게 이해될 수 있지만, 실질적인 차이가 존재한다. AI, ML(Machine Learning), DL의 관계를 이해하기 위해서는 아래의 그림과 같은 포함관계로 이해함이 바람직하다 [1].

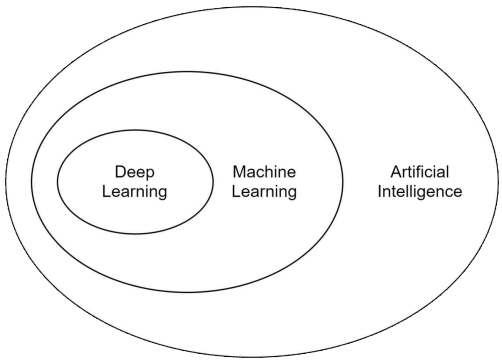

Fig. 1. AI＆ML & DL

AI는 인간의 학습 능력, 추론 능력, 지각 능력, 언어 이해 능력 등을 컴퓨터에 구현한 기술이다 [1]. ML은 과거의 경험을 바탕으로 추론 또는 결정을 하는 방법을 머신에게 알려준다. 즉, 특정 패턴을 파악하고 과거의 데이터를 분석함으로써 그 의미

를 추측하고 결론을 도출한다 [1]. 빅데이터를 분석하고 가공하여 새로운 정보를 얻어낸다. DL은 ML의 한 종류로 생각할 수 있다. 패턴으로부터 결과를 분류하고 추측하기 위해, Layer를 통해 입력을 처리하는 것을 머신에게 알려준다. DL은 축적된 데이터를 분석뿐만 아니라 학습까지 하는 능력을 활용하여 결론을 도출한다 [1]. 딥러닝은 컴퓨터 비전, 음성 인식, 자연어 통역 및 이미지 인식 등 다양한 비형식 데이터 처리 분야에서 다른 ML 알고리즘보다 뛰어난 성능 및 범용성을 보여준다 [1]. 이는 인간의 신경 세포와 유사한 원리에 따라 기능을 한다. 딥러닝에서는 분야에 따라 CNN(Convolutional Neural Network), RNN(Recurrent Neural Network), RBM(Restricted Boltzmann Machine), DBN(Deep Belief Network) 등 다양한 학습 모델이 있다.

인공 신경망(ANN)은 인간의 신경 세포 즉 뉴런과 유사한 인공 뉴런을 이용하여 복잡한 프로세서로 이루어진다. 시냅스에 해당되는 가중치는 인공 뉴런들 사이에 신호가 지나가는 링크들로 연결되어 있다. 가중치는 인공 신경망 모델에서 기억과 수행한 학습을 바탕으로 뉴런 입출력 강도, 즉 특성 문제에 대한 신경망의 패턴을 표현한다 [2-6]. 이 가중치를 반복적으로 조정함으로써 학습이 이루어진다.

최근 많은 컴퓨터 비전 분야에서 딥러닝을 이용하여 최고의 성능을 도출하고 있다 [7]. 하지만 딥러닝은 연산량과 메모리 사용량이 매우 커서 실시간 구현을 위해서는 하드웨어 가속기가 필수적이다 [7]. 특히, 영상 콘텐츠에 대한 수요가 증가하는 요즈음 촬영과 동시에 유통을 하는 생방송 영상의 지적재산권을 보호할 수 있는 워터마킹 삽입 방법이 요구된다 [7]. 디지털 워터마킹 프로세서의 구조는 주동작에 따라 크게 데이터 패스부와 제어부로 구분된다 [7]. 데이터 패스부에서는 CNN의 핵심인 Convolution 연산을 하며, 제어부는 메모리 제어부와 메인 제어부로 나뉘게 된다 [7]. CNN은 컴퓨터 비전 분야 중에서 이미지 인식 문제를 해결하기 위해 특화된 딥러닝 학습 모델 중 하나이다 [8]. Convolution block에서는 Filter Register나 Input Feature Map Register를 입력으로 받아 Convolution 연산을 수행한다 [7]. 그 결과값을 Partial Sum Register에 넘겨준다 [7]. 이는 SRAM 메모리에 저장되어 다음 동작에 사용되거나 다음 block으로 넘어가 활성화 함수를 통해 Output Register에 저장된다 [7].

CNN은 대표적인 feedforward neural network로 보통 여러 층의 convolutional layer와 pooling layer, 그리고 fully connected layer를 포함한다 [9]. 이 층들이 input부터 output까지 연결된다 [9]. RTL level로 CNN 연간 가속기뿐만 아니라 CNN application system의 빌드 속도를 향상시키는 CNN IP core 설계를 위한 sub-block partitioning은 다음과 같다.

i) Convolution module : image feature 추출

Input Feature map에 convolution kernal과 weight로 합성곱연산을 한다. 그 후, 활성화 함수(activation function)를 거쳐 해당 layer의 feature map을 얻는다. 이를 표현한 식은 아래 [9]와 같다.

$$y_j^l = f(\sum_{i \in M_j}^{i^{l-1}} x_i^{l-1} * k_{ij}^l + b_j^l)$$

$y_j^l = feature\,map\,of\,the\,j-th\,convolution\,kernal\,convolution\,result\,of\,l-th\,layer$
$M_j = selection\,of\,the\,previous\,input\,feature\,map\,by\,the\,current\,convolution$
$x_i^{l-1} = previous\ input\,feature\,map$
$k_{ij}^l = i-th\,weighting\,coefficient\,of\,the\,j-th\,convolution\,\mathrm{kernel}\,of\,the\,l-th\,layer$
$b_j^l = bias\,parameter\,of\,the\,j-th\,convolution\,kernel\,of\,the\,l-th\,layer$
$f = nonlinear\,activation\,function$

ii) Pooling module

주로 Convolution module 뒤에 위치하여, input feature map의 사이즈를 줄이는 기능을 한다. Maximum pooling과 average pooling 방법이 있다 [9]. 각각 최댓값과 평균값이 해당 영역의 output data로 쓰이며, 2x2 size를 가장 많이 쓴다.

iii) Fully connected module

Fully Connected Layer에서는 이전 layer의 feature map output을 one-dimensional vector로 변환한다. Multiply-accumulate operation으로 적은 양의 연산이지만 많은 양의 가중치 파라미터와 메모리 접근이 요구된다 [9]. 따라서 FC layer parallel computing을 확장시키는 것은 비현실적이므로, 본 선행연구에서는 locally parallel acceleration 전략을 사용한다. 이는 $P_x \times P_y$가 previous layer의 FC layer의 output feature map 사이즈, P_n이 채널의 수, M_n이 FC layer의 뉴런 수라고 가정한다 [9]. 이때, 변환된 벡터의 총 길이는 $L_n = P_x P_y P_n$이고, 각 채널의 길이는 $P_x P_y$이다 [9].

■ 제안 작품 소개

1) CNN이란?

Convolutional Neural Network(CNN)는 이미지를 분석하기 위해 패턴을 찾는 데에 유용한 알고리즘으로 입력데이터로부터 직접 학습하고 패턴을 사용하여 이미지를 분류한다. Fully Connected Layer(FC)로 구성된 신경망은 입력데이터가 1차원 배열 형태로 한정되어있다는 한계가 있다. 하지만 한 장의 컬러 사진은 RGB로 이루어져 있기 때문에 3차원 데이터이다. 또한, 배치 모드에 사용되는 여러 장의 사진은 4차원 데이터이다. 따라서, 사진 데이터로 Fully Connected 신경망을 학습 학습시킬

때, 3차원의 데이터를 1차원으로 평면화해야 한다. 그 과정에서 공간 정보의 손실이 발생한다. 이러한 한계를 극복하고 이미지의 공간 정보를 유지한 상태로 학습이 가능한 모델이 CNN이다. CNN은 각 Layer의 입출력 데이터의 형상을 유지하고, 이미지의 공간 정보를 유지하며 인접 이미지와의 특징을 인식한다는 차별점이 있다. 또한, 필터를 공유 파라미터로 사용하기 때문에 일반 인공 신경망과 비교하였을 때 학습 파라미터가 적다는 특징이 있다.

CNN은 크게 두 가지 파트로 나눌 수 있다. 첫 번째는 이미지의 특징을 추출하는 부분이고 두 번째는 클래쓰를 분류하는 부분이다. 이미지 특징을 추출하는 부분은 Convolution Layer와 Pooling Layer를 여러 겹 싸는 형태이다. Convolution Layer는 입력데이터에 필터를 적용 후 activation function(활성화 함수)을 반영하는 필수 요소로 필터를 사용하여 공유 파라미터 수를 최소화하면서 이미지의 특성을 찾는다. Pooling Layer는 선택적 레이러로 특징을 강화하고 모은다. Max Pooling, Average Pooling, Min Pooling 등의 방법이 있는데 CNN에서는 주로 Max Pooling을 사용한다. 필터의 크기, stride, padding 적용 여부, pooling 크게 등에 따라 출력 데이터의 shape이 변경되므로 적절히 조절하여 입/출력 데이터 shape을 맞추는 작업이 중요하다. 클래쓰를 분류하는 Fully Connected Layer는 CNN의 마지막에 추가되며, 그 사이에 이미지 형태의 데이터를 배열 형태로 만드는 Flatten Layer가 존재한다.

Channel(채널)은 데이터의 차원을 의미한다. 즉, 컬러 사진은 천연색을 표현하기 위해 RGB 3개의 실수로 표현한 3-channel의 3차원 데이터이고, 흑백사진은 2차원 데이터로 1-channel로 구성된다. Convolution Layer에 유입되는 입력데이터에는 보통 한 개 이상의 필터가 적용된다. 즉, n개의 필터가 적용되면 출력 데이터는 n개의 channel을 갖게 된다. Filter(필터)는 이미지의 특징을 찾아내기 위한 공용 파라미터로 정사각 행렬이다. 즉, CNN에서 학습의 대상이 필터 파라미터이고, 필터는 입력데이터를 유입하고 학습시키면 자동으로 특징을 인식하여 필터를 만들어 낸다. 입력데이터에 대해 지정된 간격(stride)으로 순회하며 채널별로 합성곱 연산을 하고 모든 채널의 합성곱의 합을 Feature Map으로 만든다. 필터는 입력받은 데이터에서 그 특성을 가지고 있으면 큰 값이 나오고 그렇지 않다면 0으로 수렴하는 양상이 있다. 즉, 데이터가 그 특성을 가지고 있는지 없는지의 여부를 합성곱 연산의 결과로부터 예상할 수 있다. 입력데이터가 여러 채널을 가질 경우 필터는 각 채널별로 순회하며 합성곱을 계산한 후 각 채널별로 Feature Map을 만든다. 그 후, 각 채널의 Feature Map을 합산하여 최종적으로 한 개의 Feature Map을 반환한다. 합성곱 연산을 통해 얻은 Feature Map은 비선형성 추가를 위해 활성화 함수를 지나게 되는데, CNN에서는 주로 ReLU 함수를 사용한다. Padding(패딩)이란 Convolution Layer의 출력 데이터 크기가 줄어드는 것을 방지하기 위한 방법으로 입력데이터의 외각

에 지정된 픽셀만큼 특정값으로 채워 넣게 된다. 보통 0으로 채워 넣으며 이를 zero padding이라고 한다. Padding을 통해 출력 데이터의 사이즈 조절뿐만 아니라 이미지 외각을 인식하는 학습의 효과도 얻을 수 있다. Bias(편향)은 필터를 적용한 후에 더해지는 파라미터이다.

Convolution Layer의 Output Data 크기는 아래의 식을 통해 계산할 수 있다.

$$(Output\ Height) = \frac{H + 2P - FH}{S} + 1$$
$$(Output\ Width) = \frac{W + 2P - FW}{S} + 1$$

H: Input Data Height
W: Input Data Width
FH: Filter Height
FW: Filter Width
S: Stride
P: Padding

Max Pooling Layer의 출력 데이터 크기는 아래의 식을 통해 계산할 수 있다.

$$(Output\ Height) = \frac{H}{Pooling\ Size}$$
$$(Output\ Width) = \frac{W}{Pooling\ Size}$$

2) 환경 설정

- Python 설치

https://www.python.org/

- Anaconda 설치

https://www.anaconda.com/

- PyTorch 설치

https://pytorch.org/
파이토치는 앞서 설치한 아나콘다의 버전, OS, 플랫폼에 따라 설치 명령어가 다르므로 위 링크에서 확인 후 설치하도록 한다. 설치 후 아래 명령을 통해 아나콘다 프롬프트에서 버전 확인이 가능하다.

```
import torch
torch.__version__
```

- Modelsim 설치

https://www.intel.co.kr/

해당 연구작품은 21.2버전 환경에서 설계되었다.

3) PyTorch를 이용한 MNIST 데이터셋 분석

1. CNN Layer Optimization

PyTorch에서 데이터로더를 사용하여 MNIST 손글씨 데이터셋에 대한 학습을 진행한 결과이다. 파라미터는 아래 표와 같이 설정하였고, 각각 2개, 4개 층의 Layer를 이용해 학습을 진행하고 적중률을 확인하였다.

Table. 1. Parameter Setting for CNN

Batch Size	100
Training Epoch	15
Learning Rate	0.001
Optimizer	Adam
Activation Function	ReLU

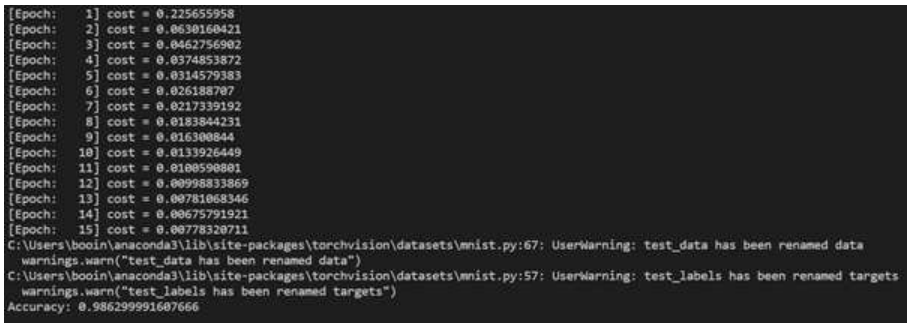

Fig. 2. Test Result of 2-Layer CNN

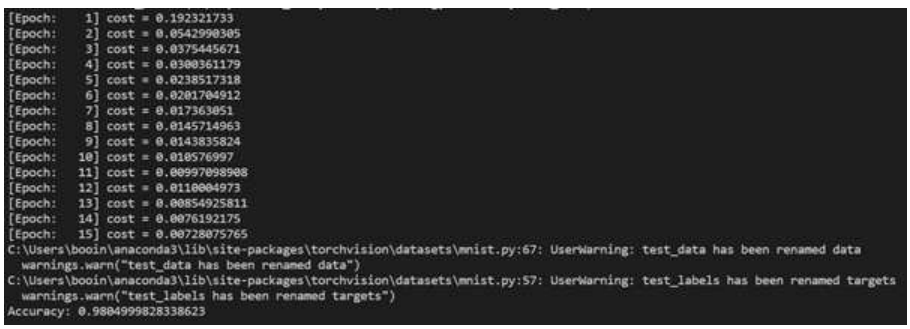

Fig. 3. Test Result of 4-Layer CNN

2개의 Layer로 학습하였을 때는 98.6%, 4개의 Layer로 학습하였을 때는 98%의 적

중률을 가졌다. 더 많은 Convolution Layer를 통과시켰지만 오히려 적중률을 0.6% 떨어졌다. 결과적으로 Layer를 여러번 통과시킨다고 적중률이 항상 오르는 것을 아니기 때문에 효율적으로 Layer를 쌓는 것이 중요하다. 즉, 환경에 따라 적절한 Learning rate, Batch Size, Training Epochs를 결정하여 안정적으로 학습이 가능하도록 하는 것이 바람직하다.

2. Data Extraction from CNN

최소 94% 이상의 적중률을 가지면서 Layer를 가능한 한 적게 쌓아 용량이 작은 CNN 모델을 채택하였다. 채택한 CNN의 모델은 2-Layer로 구조와 parameter set은 다음과 같다.

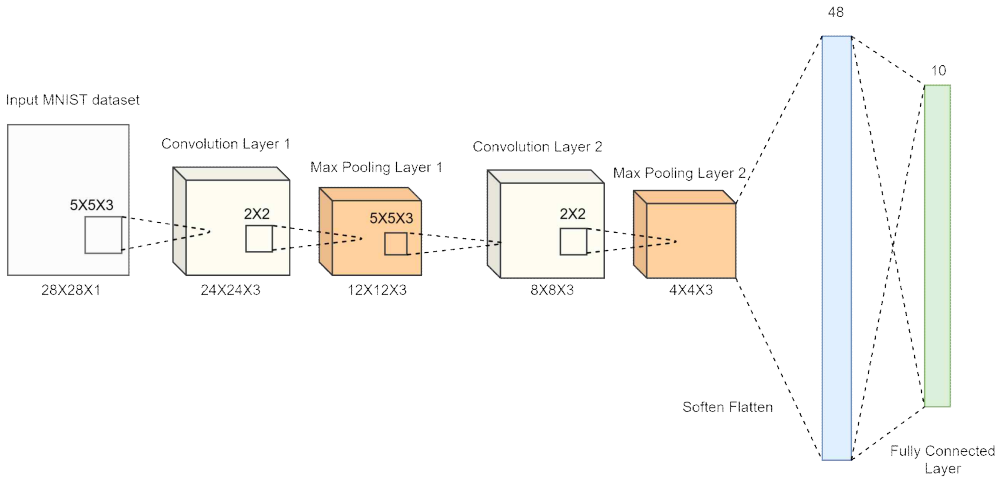

Fig. 4. 2-Layer CNN Structure for MNIST Dataset

Table. 2. Size Information

Layer	Filter	Stride	Pooling	Activation Function	Input Shape	Output Shape
Convolution Layer1	(5, 5, 3)	1	X	ReLU	(28, 28, 1)	(24, 24, 3)
Max Pooling Layer1	X	X	(2, 2)	X	(24, 24, 3)	(12, 12, 3)
Convolution Layer2	(5, 5, 3)	1	X	ReLU	(12, 12, 3)	(8, 8, 3)
Max Pooling Layer2	X	X	(2, 2)	X	(8, 8, 3)	(4, 4, 3)
Flatten	X	X	X	X	(4, 4, 3)	(48, 1)
Fully Connected Layer	X	X	X	X	(48, 1)	(10, 1)

파라미터는 아래의 표와 같이 설정하였다.

Table. 3. Parameter Setting for MNIST CNN

Batch Size	64
Training Epoch	10
Learning Rate	0.01
Optimizer	Stochastical Gradient Descent, Momentum = 0.5
Activation Function	ReLU

학습 결과 96.29%의 적중률을 보였다.

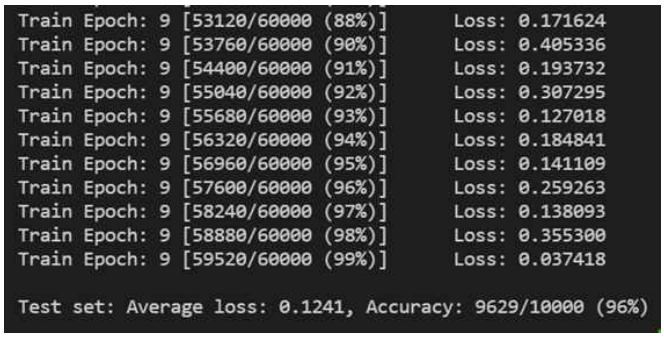

Fig. 5. Test Result of 2-Layer CNN for MNIST Dataset

해당 CNN model을 이용하여 RTL-level Verilog 설계를 위해 필요한 가중치(weight)와 편향(bias) 그리고 각 Layer의 Output Data를 추출하여 텍스트 파일로 저장한다. PyTorch에서 torchvision 패키지를 이용해 다운받은 MNIST dataset의 binary file을 big-endien 포맷으로 읽어 각 이미지파일을 bitmap file로 저장한다. bmp file을 읽어 28X28 크기로 reshape한 후 해당 이미지에 대한 학습을 진행한다. 각 Convolution Layer에서의 weight와 bias에 128(2^7)를 곱하고, 2의 보수를 취한 int 형으로 각각 텍스트 파일에 저장한다. 아래의 그림은 CNN결과 추출된 첫 번째 Convolution Layer에 대한 3개 채널의 weight와 bias이다.

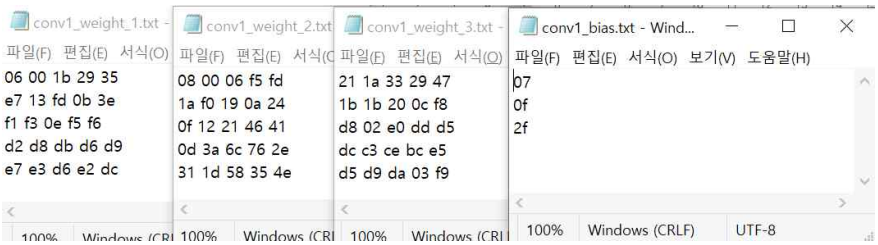

Fig. 6. Extracted weight & bias value from 1st Layer of CNN

또한, 후에 설계된 CNN을 검증하기 위해 각 Layer를 통과시킨 후 출력 데이터의

Feature Map 또한 위와 유사한 방법으로 calibration한 후 아래와 같이 텍스트 파일로 저장한다.

```
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 = " ")
```

Fig. 7. Save Output Data of 1st Convolution Layer

4) MNIST 입력데이터 생성

PyTorch 프레임워크에서 MNIST 입력데이터를 추출할 때, 배열 형태로 텍스트 파일 혹은 비트맵 파일로 아래와 같이 저장할 수 있다. 또한, 여러 개의 텍스트 파일을 병합하여 여러 개의 연속적인 입력데이터를 만들 수 있는데 파이썬 스크립트를 사용하여 소요되는 시간을 줄일 수 있다.

```
for image in range(0, 500):
    # the image is 28*28=784 unsigned chars
    im = struct.unpack_from('>784B', buf, idx)
    idx += struct.calcsize('>784B')

    # create a np array to save the image
    im = np.array(im, dtype='uint8')
    im = im.reshape(28, 28)

    np.savetxt("extracted/0_%s.txt" % image, im, fmt = '%02x', delimiter = ' ')

    im = Image.fromarray(im)
    im.save("extracted/0_%s.bmp" % image, "bmp")
```

Fig. 8. Save Image as txt file & bmp file

5) Verilog Block Diagram & Sub Module Description

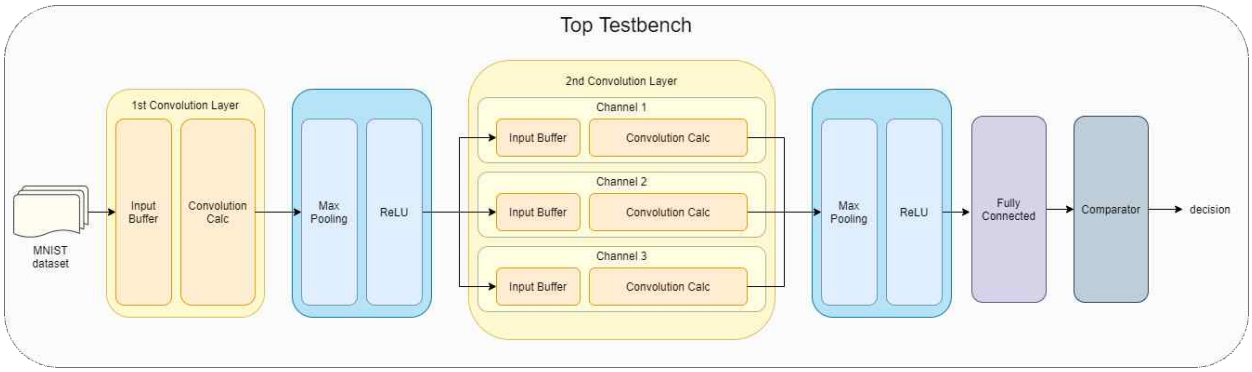

Fig. 9. CNN Verilog Block diagram

- Top Testbench

: 설계 검증 및 시뮬레이션을 위한 top testbench로 파일 입출력을 통해 MNIST 데

이터셋을 입력하고, sub module을 instantiation 한다. Test input에 대한 결과값과 기댓값을 비교하여 설계를 검증한다.

- Convolution Layer
: 합성곱 연산을 수행하는 모듈로, layer에 따라 파라미터가 달라지며 Input Buffer 모듈과 Convolution Calc 모듈로 구성된다.

- Maxpooling + ReLU
: Maxpooling과 활성화 함수인 ReLU 함수를 함께 수행하는 모듈이다.

- Fully Connected
: 마지막 maxpooling_relu 모듈의 결과값으로 최종적인 class를 분류하기 위해 10개의 output으로 합성곱연산을 수행하는 모듈이다.

- Comparator
: 최종적으로 CNN의 학습 결과 데이터값을 결정하는 모듈이다.

■ 구현 및 결과분석

1) Verilog Module Declaration & Specification

1. Top Testbench (top_tb.v, top_tb_1000.v)

- top_tb.v : 단일 입력데이터에 대한 테스트 벤치
- top_tb_1000.v : 1,000개의 랜덤 연속 입력데이터에 대한 테스트 벤치

<Input/Output>
Top Testbench는 input/output port가 없다.

<핵심 코드 및 기능>
- Clock Generator : always block을 사용하여 클럭을 생성한다.
- Sub module Instantiation : 두 개의 convolution layer, maxpooling layer, ReLU 모듈과 fully connected layer, comparator 모듈을 instantiation 한다. 각각의 input/output port를 wire로 연결한다.
- File Input : Verilog system task인 $readmemh를 사용하여 텍스트 파일을 읽고, 2차원 배열에 저장한다. 클럭의 rising edge에서 해당 배열의 값을 한 픽셀씩 첫 번째 convolution layer로 입력한다.
- 설계한 디자인에 test input을 차례로 인가하여 나오는 결과값과 기댓값을 비교하여 설계를 검증하게 된다. %display등의 구문을 사용하여 출력을 통해 확인할 수

있다.
- 1,000개의 랜덤 연속 입력데이터 테스트 벤치의 경우 매회 난수를 발생시켜 해당 인덱스에 해당하는 MNIST dataset에 대한 학습을 진행한다.

2. Convolution Layer Module (conv1_layer.v, conv2_layer.v)

각 convolution layer는 버퍼와 합성곱 연산을 하는 모듈로 구성되어 있다.

1) 1st Convolution Layer Module (conv1_layer.v)

<Input/Output>

```
module conv1_layer (
  input clk,
  input rst_n,
  input [7:0] data_in,
  output [11:0] conv_out_1, conv_out_2, conv_out_3,
  output valid_out_conv
);
```

Fig. 10. conv1_layer Module Declaration

- clk : clock input이다.
- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- data_in : top testbench에서 MNIST input text 파일을 읽어 각 픽셀별로 넘겨준 값으로, 각 픽셀값은 0~255 사이의 값을 갖기 때문에 8 bits이다.
- conv_out_# : 첫 번째 convolution layer에서 합성곱을 수행한 결과값이다.
- valid_out_conv : 첫 번째 convolution layer에서 현재 valid한 state인지 알려주는 signal이다.

<핵심 코드 및 기능>
- Sub module Instantiation : 입력데이터의 픽셀값을 입력으로 받아 버퍼로 저장하는 conv1_buf 모듈과 합성곱 연산을 수행하는 conv1_calc 모듈을 instantiation 한다. conv1_buf의 output과 conv1_calc의 input을 wire로 연결한다.

(1) 1st Layer Convolution Buffer Module (conv1_buf.v)

<Input/Output>

```
module conv1_buf #(parameter WIDTH = 28, HEIGHT = 28, DATA_BITS = 8)(
  input clk,
  input rst_n,
  input [DATA_BITS - 1:0] data_in,
  output reg [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
  data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
  data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
  data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
  data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
  output reg valid_out_buf
);
```

Fig. 11. conv1_buf Module Declaration

- clk : clock input이다.
- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- data_in : top testbench에서 MNIST input text 파일을 읽어 각 픽셀별로 넘겨준 값으로, 각 픽셀값은 0~255사이의 값을 갖기 때문에 8 bits이다.
- data_out_# : 필터 크기가 (5, 5, 3)이므로 총 5 * 5 = 25개의 output을 갖는다. 각 signal은 8 bits이며 verilog는 2차원 배열을 port로 넘길 수 없기 때문에 25개의 signal로 feature map을 output port로 넘긴다.
- valid_out_buf : 첫 번째 layer의 buffer 모듈에서 현재 valid한 state인지 알려주는 signal이다.

<핵심 코드 및 기능>

- Buffer : 사이즈가 28 * 5(filter size) = 140인 버퍼를 사용하여 합성곱 연산을 하기 위한 데이터를 담는다. 각 클럭마다 모듈의 input port로 들어오는 data_in을 버퍼에 담는다. 이때, 최초 140개의 데이터를 담아야 버퍼가 모두 차기 때문에 최초 140 클럭 동안은 unvalid한 state가 된다. 사이즈가 5 * 5인 필터를 버퍼의 왼쪽에서 오른쪽, 위에서 아래로 움직이며 출력값인 data_out_#을 결정하게 되는데, 필터의 오른쪽이 버퍼를 넘어가면 unvalid한 영역이므로 valid_out_buf를 0으로 할당한다. 오른쪽 끝까지 필터가 이동했다면 아래로 내려가며 버퍼를 다시 왼쪽에서 오른쪽으로 이동하며 위의 과정을 반복한다. 이때, 버퍼는 140으로 사이즈가 고정되어 있고, 141번째 데이터는 다시 인덱스 0으로 들어오기 때문에 필터의 위치에 따른 별도의 flag가 필요하다. 각 flag별 출력값은 아래와 같이 할당된다.

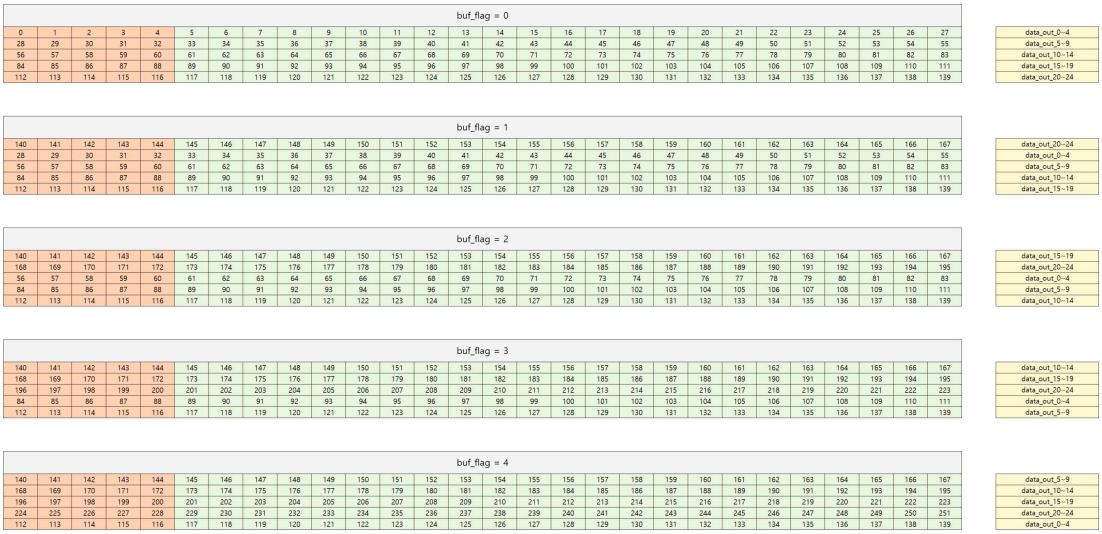

Fig. 12. Decision of Output Data

(2) 1st Layer Convolution Calculation Module (conv1_calc.v)

<Input/Output>

```
module conv1_calc #(parameter WIDTH = 28, HEIGHT = 28, DATA_BITS = 8)(
  input valid_out_buf,
  input [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
  data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
  data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
  data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
  data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
  output signed [11:0] conv_out_1, conv_out_2, conv_out_3,
  output reg valid_out_calc
);
```

Fig. 13. conv1_calc Module Declaration

- valid_out_buf : conv1_buf 모듈의 valid_out_buf가 연결되는 port로 valid state에서만 동작한다.
- data_out_# : conv1_buf 모듈의 output signal인 data_out_#이 연결되는 port로 가중치값과 곱하여 합성곱 연산을 수행한다.
- conv_out_# : 각 채널의 가중치와 input data를 사용하여 합성곱 연산을 수행한 결과값에 편향을 더하여 할당한다. 필터의 사이즈가 (5, 5, 3)이므로 채널이 3개이기 때문에 해당 port도 3개이다.
- valid_out_calc : 첫번째 layer에서 현재 valid한 state인지 알려주는 signal이며, input signal인 valid_out_buf가 그대로 할당된다.

<핵심 코드 및 기능>

- Weight & Bias File Input : Verilog system task인 $readmemh를 사용하여 가중치와 편향 텍스트 파일을 읽고, 2차원 배열에 저장한다. 편향의 경우 PyTorch 프

레임워크에서 값을 추출할 때 2의 보수를 취했기 때문에 MSB에 따라 calibration을 해준다. 이 가중치와 편향값, 모듈의 input signal값을 사용하여 합성곱 logic을 수행한다.

2) 2nd Convolution Layer Module (conv2_layer.v)

<Input/Output>

```
module conv2_layer (
  input clk,
  input rst_n,
  input valid_in,
  input [11:0] max_value_1, max_value_2, max_value_3,
  output [11:0] conv2_out_1, conv2_out_2, conv2_out_3,
  output reg valid_out_conv2
);
```

Fig. 14. conv2_layer Module Declaration

- clk : clock input이다.
- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- valid_in : 앞선 첫 번째 maxpooling_reLU 모듈의 valid output signal이 연결되는 port로 valid state에서만 동작한다.
- max_value_# : 앞선 첫 번째 maxpooling_relu 모듈의 output signal이 연결되는 port로 두 번째 convolution layer의 입력데이터가 된다.
- conv2_out_# : 두 번째 convolution layer에서 합성곱을 수행한 결과값이다.
- valid_out_conv2 : 두 번째 convolution layer에서 현재 valid한 state인지 알려주는 signal이다.

<핵심 코드 및 기능>

- Sub module Instantiation : Convolution Layer1의 maxpooling과 activation function을 거친 output signal을 입력으로 받아 버퍼로 저장하는 conv2_buf 모듈과 합성곱 연산을 수행하는 conv2_calc 모듈을 instantiation 한다. 이 layer는 channel이 총 3개이므로 각각 3개의 버퍼와 3개의 합성곱 모듈을 instantiation 한다. conv2_buf의 output과 conv2_calc의 input을 wire로 연결한다.
- Bias File Read : 편향(bias)값은 3개의 채널에서 동일한 값을 사용하므로 이 모듈에서 $readmemh를 사용해 값을 읽어 2차원 배열에 저장한다. PyTorch 프레임워크에서 값을 추출할 때 2의 보수를 취했기 때문에 MSB에 따라 calibration을 해준다. 보정한 편향값을 conv2_calc 모듈의 output signal에 더하여 conv2_layer의 최종 output signal로 할당한다.

(1) 2nd Layer Convolution Buffer Module (conv2_buf.v)

<Input/Output>

```
module conv2_buf #(parameter WIDTH = 12, HEIGHT = 12, DATA_BITS = 12) (
  input clk,
  input rst_n,
  input valid_in,
  input [DATA_BITS - 1:0] data_in,
  output reg [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
  data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
  data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
  data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
  data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
  output reg valid_out_buf
);
```

Fig. 15. conv2_buf Module Declaration

- clk : clock input이다.
- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- valid_in : 첫 번째 layer의 maxpooling_relu 모듈의 valid output signal이 연결되는 port로 valid state에서만 동작한다.
- data_in : 첫 번째 layer의 maxpooling_relu 모듈의 output signal이 연결되는 port로 두 번째 layer의 입력데이터가 된다.
- data_out_# : 필터 크기가 (5, 5, 3)이므로 총 5 * 5 = 25개의 output을 갖는다.
- valid_out_buf : 두 번째 layer의 buffer 모듈에서 현재 valid한 state인지 알려주는 signal이다.

<핵심 코드 및 기능>

- 1st Layer Convolution Buffer Module과 동일하다.

(2) 2nd Layer Convolution Calculation Module (conv2_calc_1.v, conv2_calc_2.v, conv2_calc_3.v)

<Input/Output>

```
module conv2_calc_1 #(parameter WIDTH = 28, HEIGHT = 28, DATA_BITS = 8)(
  input clk,
  input rst_n,
  input valid_out_buf,
  input [11:0] data_out1_0, data_out1_1, data_out1_2, data_out1_3, data_out1_4,
               data_out1_5, data_out1_6, data_out1_7, data_out1_8, data_out1_9,
               data_out1_10, data_out1_11, data_out1_12, data_out1_13, data_out1_14,
               data_out1_15, data_out1_16, data_out1_17, data_out1_18, data_out1_19,
               data_out1_20, data_out1_21, data_out1_22, data_out1_23, data_out1_24,

               data_out2_0, data_out2_1, data_out2_2, data_out2_3, data_out2_4,
               data_out2_5, data_out2_6, data_out2_7, data_out2_8, data_out2_9,
               data_out2_10, data_out2_11, data_out2_12, data_out2_13, data_out2_14,
               data_out2_15, data_out2_16, data_out2_17, data_out2_18, data_out2_19,
               data_out2_20, data_out2_21, data_out2_22, data_out2_23, data_out2_24,

               data_out3_0, data_out3_1, data_out3_2, data_out3_3, data_out3_4,
               data_out3_5, data_out3_6, data_out3_7, data_out3_8, data_out3_9,
               data_out3_10, data_out3_11, data_out3_12, data_out3_13, data_out3_14,
               data_out3_15, data_out3_16, data_out3_17, data_out3_18, data_out3_19,
               data_out3_20, data_out3_21, data_out3_22, data_out3_23, data_out3_24,

  output signed [13:0] conv_out_calc,
  output reg valid_out_calc
);
```

Fig. 16. conv2_calc_1 Module Declaration

- clk : clock input이다.
- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- valid_out_buf : conv2_buf 모듈의 valid_out_buf가 연결되는 port로 valid state에서만 동작한다.
- data_out_1_# : 첫 번째 채널의 conv2_buf 모듈의 output signal인 data_out_#이 연결되는 port로 가중치값과 곱하여 합성곱 연산을 수행한다.
- data_out_2_# : 두 번째 채널의 conv2_buf 모듈의 output signal인 data_out_#이 연결되는 port로 가중치값과 곱하여 합성곱 연산을 수행한다.
- data_out_3_# : 세 번째 채널의 conv2_buf 모듈의 output signal인 data_out_#이 연결되는 port로 가중치값과 곱하여 합성곱 연산을 수행한다.
- conv_out_calc : 두 번째 convolution layer는 출력 채널이 3이므로 3개의 conv2_calc 모듈에서 각각 3개의 입력 채널의 버퍼 데이터를 사용하여 가중치를 사용하여 3번씩 합성곱 연산을 하게 된다. 최종적으로 3개의 합성곱 결과값을 더하여 conv_out_calc로 할당한다. 편향값을 상위 모듈인 conv2_layer에서 일괄적으로 더해준다.
- valid_out_calc : 두 번째 convolution layer에서 현재 valid한 state인지 알려주는 signal이다.
- conv2_calc_2, conv2_calc_3도 동일하다.

<핵심 코드 및 기능>

- Weight File Input : Verilog system task인 $readmemh를 사용하여 가중치 텍스트 파일을 읽고, 2차원 배열에 저장한다. 이 가중치와 편향값, 모듈의 input signal값을 사용하여 합성곱 logic을 수행한다.
- conv2_calc_2, conv2_calc_3도 동일하다.

3. Max Pooling + ReLU Module (maxpool_relu.v)

<Input/Output>

```
module maxpool_relu #(parameter CONV_BIT = 12, HALF_WIDTH = 12, HALF_HEIGHT = 12, HALF_WIDTH_BIT = 4) (
  input clk,
  input rst_n,  // asynchronous reset, active low
  input valid_in,
  input signed [CONV_BIT - 1 : 0] conv_out_1, conv_out_2, conv_out_3,
  output reg [CONV_BIT - 1 : 0] max_value_1, max_value_2, max_value_3,
  output reg valid_out_relu
);
```

Fig. 17. maxpool_relu Module Declaration

- clk : clock input이다.

- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- valid_in : 직전 convolution layer의 valid output signal이 연결되는 port로, valid state에서만 동작한다.
- conv_out_# : 직전 convolution layer의 output signal이 연결되는 port로 해당값을 maxpooling_relu 모듈에서 입력데이터로 사용하여 동작한다.
- max_value_# : maxpool_relu 모듈의 output signal로 필터의 사이즈가 (2, 2)이므로 총 2 * 2 = 4개의 feature map 데이터에서 가장 큰 값을 결정하고, 활성화 함수인 ReLU를 거쳐 음수값은 제외하고 값이 할당된다.
- valid_out_relu : maxpool_relu 모듈에서 현재 valid한 state인지 알려주는 signal이다.

<핵심 코드 및 기능>
- Buffer : max pooling logic을 수행할 때 사이즈가 (2, 2)이므로 총 2 * 2 = 4개의 입력데이터값을 비교하게 된다. state와 flag를 이용하여 4개의 값을 크기를 비교하며 최댓값만 버퍼에 담는다. 이때, 마지막 4번째 입력데이터에 대해서는 reLU함수를 거쳐 양수일 때만 값을 넣고 음수일 때는 0을 할당하도록 한다.

4. Fully Connected Module (fully_connected.v)

<Input/Output>

```
module fully_connected #(parameter INPUT_NUM = 48, OUTPUT_NUM = 10, DATA_BITS = 8) (
  input clk,
  input rst_n,
  input valid_in,
  input signed [11:0] data_in_1, data_in_2, data_in_3,
  output reg [11:0] data_out,
  output reg valid_out_fc
);
```

Fig. 18. fully_connected Module Declaration

- clk : clock input이다.
- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- valid_in : 마지막 convolution layer의 maxpooling_relu 모듈의 valid output signal이 연결되는 port로, valid state에서만 동작한다.
- data_in_# : 마지막 convolution layer의 maxpooling_relu 모듈의 output signal이 연결되는 port로, MSB에 따라 signed로 calibration을 한 후 fully connected 모듈의 입력데이터로 사용된다.
- data_out : fully connected 모듈의 output signal로, 가중치와 편향값을 사용하여 합성곱 연산을 수행한 결과값이 할당된다.
- valid_out_fc : fully connected 모듈에서 현재 valid한 state인지 알려주는 signal이

다.

<핵심 코드 및 기능>

- Buffer : 앞선 두 번째 convolution layer에서 feature map의 크기가 (4, 4, 3)이므로 총 4 * 4 * 3 = 48 개의 입력데이터를 담을 버퍼가 필요하다. 각 클럭마다 모듈의 input port로 들어오는 3개 채널의 data_in을 MSB에 따라 signed로 calibration하여 버퍼에 담는다. 이때, 최초 48개의 데이터를 담아야 버퍼가 모두 차기 때문에 최초 16 클럭 동안은 unvalid한 state가 된다. 버퍼가 모두 차면 10 클럭 동안 각 버퍼 데이터값과 48개의 가중치값을 곱하고 더하여 합성곱 연산을 수행한다. 최종적으로 편향값을 더하여 output port로 할당한다.
- Weight & Bias File Input : Verilog system task인 $readmemh를 사용하여 가중치와 편향 텍스트 파일을 읽고, 2차원 배열에 저장한다. 이 가중치와 편향값, 버퍼 데이터값을 사용하여 합성곱 logic을 수행한다.

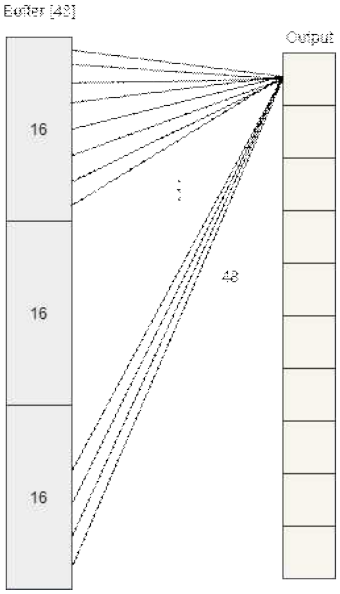

Fig. 19. fully_connected Module Operation

5. Comparator Module (comparator.v)

<Input/Output>

```
module comparator (
  input clk,
  input rst_n,
  input valid_in,
  input [11:0] data_in,
  output reg [3:0] decision,
  output reg valid_out
);
```

Fig. 20. comparator Module Declaration

- clk : clock input이다.
- rst_n : asynchronous reset으로 negative edge에서 동작한다.
- valid_in : 직전 fully conneted layer의 valid output signal이 연결되는 port로, valid state에서만 동작한다.
- data_in : 직전 fully connected layer의 output signal이 연결되는 port로, 해당 모듈에서 최종적으로 학습 결과값을 결정하는 입력데이터이다.
- decision : CNN에서 입력데이터에 대한 학습을 통해 인식한 값으로 0~9 사이의 숫자이다.
- valid_out : comparator 모듈에서 현재 valid한 state인지 알려주는 signal이다.

<핵심 코드 및 기능>

- 합성곱 연산 결과값이 클수록 해당 특징을 가질 확률이 높은 것이므로 버퍼 데이터의 최댓값이 최종 인식값이 된다.
- 앞선 fully_conneted 모듈에서 10개의 data값이 클럭마다 하나씩 순차적으로 들어오기 때문에 10개의 데이터값을 담고, 토너먼트 식으로 한 클럭마다 버퍼의 값들을 비교하며 최댓값을 결정하게 된다. 따라서, 이 과정에서 5 클럭이 소요되기 때문에 버퍼가 다 차고, 추가적으로 5 클럭이 지난 후 valid한 state가 되며 최종적으로 인식한 값이 decision 이라는 output port로 할당된다.

2) Verilog Simulation

아래는 Modelsim 프로그램 [10]에서 모든 디자인 파일 및 top testbench를 컴파일한 후, 시뮬레이션 한 결과 waveform이다. "3" MNIST 데이터셋 텍스트 파일을 입력으로 주었을 때, 13335ps에서 decision signal 값이 4'b0011 즉 3으로 할당됨을 확인할 수 있다. 동작클럭이 100MHz이므로 총 1335 클럭이 소요되었다.

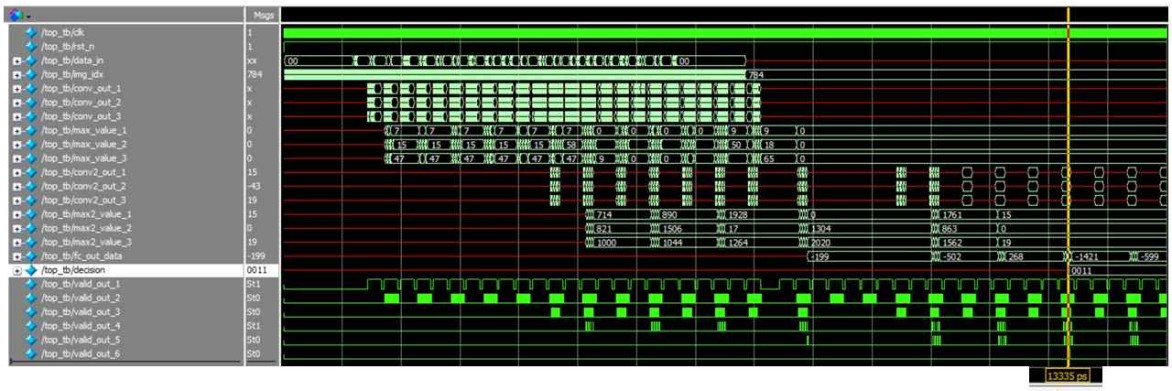

Fig. 21. Simulation Waveform for Single Input

아래는 1,000개의 MNIST 데이터셋 텍스트 파일을 랜덤으로 입력으로 주었을 때의

waveform과 transcript이다. 1000번 중 920번이 적중하여 92%의 적중률을 갖는 것을 transcript를 통해 확인할 수 있다.

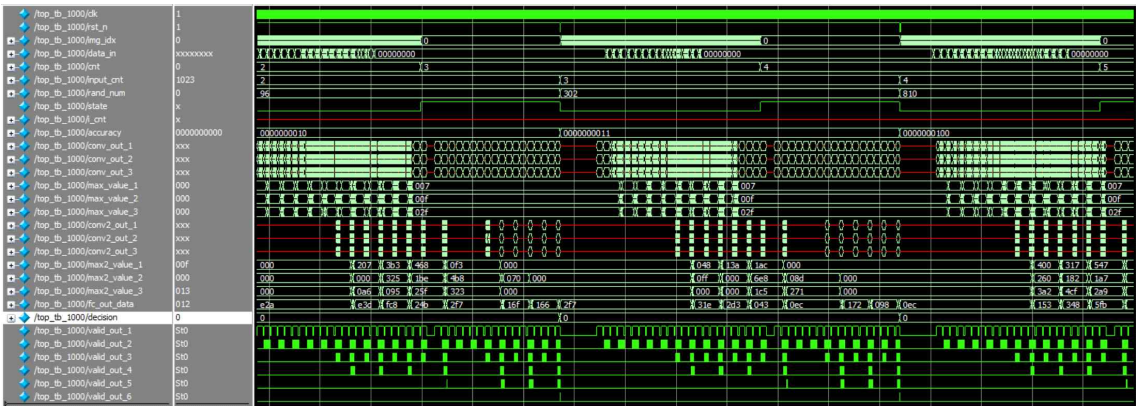

Fig. 22. Simulation Waveform for Multiple Random Input

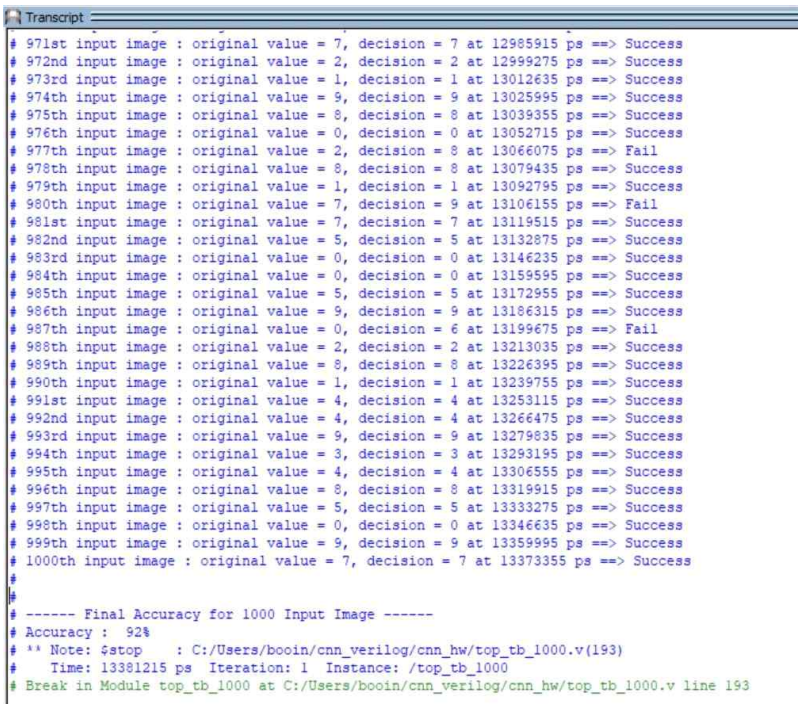

Fig. 23. Transcript for Multiple Random Input

■ 결론 및 소감

1) 결론

본 연구작품에서는 신경망을 통한 학습 과정 중 하나인 합성곱 신경망(Convolutional Neural Network)을 하드웨어로 구현하기 위한 RTL level 설계를 진행하였다. HDL은 verilog를 사용하였으며, Mentor사의 Modelsim EDA 툴 21.2 버전

[10]을 사용하여 설계하였다. 설계한 CNN은 0~9까지의 손글씨 숫자 데이터인 MNIST dataset [11]에 대한 학습을 진행하며, 2-Layer로 구성되어 있다. PyTorch 프레임워크 [12]에서 해당 CNN 모델을 사용하여 MNIST dataset에 대한 학습을 먼저 진행한 후 가중치와 편향값을 추출하여 파일 입출력을 통해 모듈 내 combinational logic에서 사용하였다.

시뮬레이션 결과 단일 MNIST 이미지 입력에 대해 100MHz에서 총 1335 클럭이 소요되어 13335ps에서 입력데이터값을 인식한 것을 확인하였다. 이후, 1,000개의 MNIST 이미지 입력을 랜덤으로 받아와 인식률을 확인했을 때 초기 목표였던 94%까지는 달성하지 못했지만 92%의 높은 적중률을 갖는 것이 확인되었다. 96.29%의 적중률을 갖는 CNN 모델을 사용하여 PyTorch 프레임워크에서 학습을 진행한 후 추출한 값을 설계에 사용했지만, 하드웨어 설계 특성상 많은 레지스터와 wire를 거치고 delay도 발생하기 때문에 적중률이 감소했다고 예상하였다.

2) 한계점 및 개선 방향

합성은 진행해보지 못하였지만, logic은 합성이 되어도 가중치와 편향값을 파일 입출력으로 가져오기 때문에 문제가 될 것이다. 따라서, 실제 하드웨어에서의 입출력 부분에 대한 고려를 하여 설계를 수정할 필요성이 있다.

또한, 일부 파라미터를 모듈에 포함시키긴 하였지만 보다 다양한 입력데이터와 필터 크기에 대한 CNN 동작을 위한 파라미터화가 부족하다. 입력데이터 크기나 필터 크기 등에 따라 설계에 사용되는 벡터나 signal의 크기(bit수)가 달라지는 부분이 존재하기 때문에 동적할당 방법을 고려할 수 있으나, 하드웨어 설계이기 때문에 어느 정도 감안해야 하는 한계점이라고 생각한다.

Verilog 언어의 문법 특성상 2차원 배열을 모듈의 port로 연결할 수 없기 때문에 크기가 28*28 인 MNIST dataset을 buffer로 입력하는 과정을 구현하는 것이 까다로웠다. Buffer를 이용하여 입력 이미지를 읽어오기 때문에 한 클럭 당 하나의 픽셀값이 입력된다. 이 과정에서 버퍼에 픽셀값이 모두 찰 때까지는 유효하지 않은 state이기 때문에 버퍼가 찰 때까지 기다려야 한다.

MNIST 입력데이터에 대해 어느 정도 수동으로 처리해 주어야 하는 한계가 존재한다. PyTorch에서 입력데이터를 텍스트 파일이나 비트맵 파일로 저장하고 병합하는 것은 파이썬 스크립트로 가능하지만, 입력데이터가 나타내는 숫자를 저장할 방법이 없다. Verilog testbench에서 test set에 대한 output과 예상되는 기댓값(실제 숫자값)을 비교하여 학습 결과가 옳은지 틀렸는지를 판단해야 하므로 PyTorch에서 추출한 비트맵 파일을 통해 직접 숫자를 판단하고 그 값을 별도로 저장해야 한

다. 본 프로젝트에서는 비트맵을 확인하여 판별한 숫자를 파일명으로 저장하여 이를 top testbench에서 파싱하여 기댓값으로 사용하였다.

3) 소감

해당 연구작품을 진행하며 CNN의 연산 방법에 대해 학습하고, RTL level 설계 능력을 기를 수 있었다. 또한, 학부 수준 수업에서 진행하는 RTL 설계 과제는 보통 각 모듈의 input/output에 대한 명세가 주어지는 경우가 많았는데, 본 프로젝트에서는 sub module specification 단계부터 직접 하다 보니 하나의 시스템을 설계하기 위해 개요를 작성하고, sub module을 명세화하는 경험을 할 수 있었다.

실질적인 설계도 설계였지만 해당 설계를 검증하는 것의 중요성을 깨달았다. 설계 규모가 커짐에 따라 전체 시스템에서의 디버깅은 쉽지 않은 일이다. 따라서, 큰 시스템을 설계할 때 더욱 꼼꼼한 검증이 요구되고, 전체 설계에 소요되는 시간을 줄이기 위해서는 각 sub module 별로 검증을 진행하는 것이 좋다. 실제로 이번 프로젝트에서도 5개 이상의 sub module이 존재하였는데, top testbench에서 원하는 동작이 나오지 않아 디버깅이 쉽지 않았다. 결국 어느 부분에서 에러가 발생했는지 확인하기 위해서는 각 sub module 별로 디버깅을 해야 했고, 해당 경험을 통해 검증의 중요성을 깨달을 수 있었고, 툴 사용 능력을 많이 기를 수 있었다.

■ 참고문헌

[1] Vishnu Subramanian, “Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch”, Birmingham : Packt Publishing, 2018.

[2] He X, Xu S. Artificial neural networks[J]. Process Neural Networks: Theory andApplications, 2010: 20-42.

[3] Artificial neural networks[M]. Humana Press, 2015.

[4] Pradhan B, Lee S. Landslide susceptibility assessment and factor effect analysis:backpropagation artificial neural networks and their comparison with frequency ratioand bivariate logistic regression modelling[J]. Environmental Modelling & Software,2010, 25(6): 747-759.

[5] Wang G, Hao J, Ma J, et al. A new approach to intrusion detection using ArtificialNeural Networks and fuzzy clustering[J]. Expert systems with applications, 2010,37(9): 6225-6232.

[6] Choi, Jae-Seung. "Voiced-Unvoiced-Silence Detection Algorithm usingPerceptron Neural Network." The Journal of the Korea institute of electroniccommunication sciences 6.2 (2011): 237-242.

[7] Lee, Jae-Eun, Seo, Young-Ho, Kim, Dong-Wook “Convolutional Neural Network 기반의 워터마킹 프로세서의 설계”, 2020년 한국방송, 미디어공학회 추계학술대회, November

(2020), pp. 106-107
[8] Convolutional Neural Networks. Stanford.edu, [Internet] Available : https://cs231n.github.io/convolutional-networks/
[9] R. Xiao, J. Shi and C. Zhang, "FPGA Implementation of CNN for Handwritten Digit Recognition," 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2020, pp. 1128-1133
[10] ModelSim Intel FPGA Starter Edition. (2021.2), Intel. [Online].
License : https://www.intel.com/content/www/us/en/programmable/documentation/esc1425946071433.html
Available : https://fpgasoftware.intel.com/20.1/?edition=lite
[11] Yann LeCun, CorinnaCortes, et al. "THE MNIST DATABASE of handwritten digits". [Online].
Available : http://yann.lecun.com/exdb/mnist/
[12] PyTorch. [Online]
License : https://pytorch.org/assets/tos-oss-privacy-policy/fb-tos-privacy-policy.pdf
Available : https://pytorch.org/