**Quantized Model Inference Processing Unit**

1. **개요 및 요약**

**MNIST 데이터 셋 기반 CNN 모델을 작성하고 학습시킨 후 양자화(Post-training Quantization)를 진행한다.**

**양자화된 모델의 연산 순서에 따라 Hardware Description Language인 Verilog를 통해 해당 모델을 위한 Inference Hardware를 설계한다.**

**Vivado내 RTL-level 시뮬레이션을 통해 양자화된 모델의 Inference를 진행해보며 정확히 모델 연산이 이뤄지는 지 확인한다.**

텍스트, 스크린샷, 직사각형, 도표이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**Fig 1. Inference Model Custom RTL Design block diagram**

**목차**

1. **CNN 모델 작성**
2. **모델 학습 진행**
3. **INT8 양자화**
4. **모델의 구조에 따른 하드웨어 작성**
5. **RTL Design Simulation**
6. **CNN 모델 학습**

**모델에 대한 학습을 진행시키고 파라미터를 뽑아내기 위해 CNN 모델을 작성한다.**

텍스트, 폰트, 번호, 타이포그래피이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**Fig 2. MNIST DATA & Model Structure**

**MNIST 데이터셋은 일반적으로 1x28x28(C, H, W)의 크기를 가지는 데이터이다.**

**일반적 이미지 데이터인 RGB는 3개의 채널을 가지고 있지만 MNIST 이미지 데이터는 gray scale로 이뤄진 1개의 채널만 가지고 있다.**

**이를 토대로 CNN 모델의 INPUT DATA는 1개의 채널을 가지고 있으며 28x28의 resolution을 가지고 있다고 할 수 있다.**

**전체 구조는 2개의 Convolution layer와 2개의 FC layer로 이뤄진다.**

**Convolution layer 내 filter의 크기는 5x5이며 개수는 3개로 설정되어 있다.**

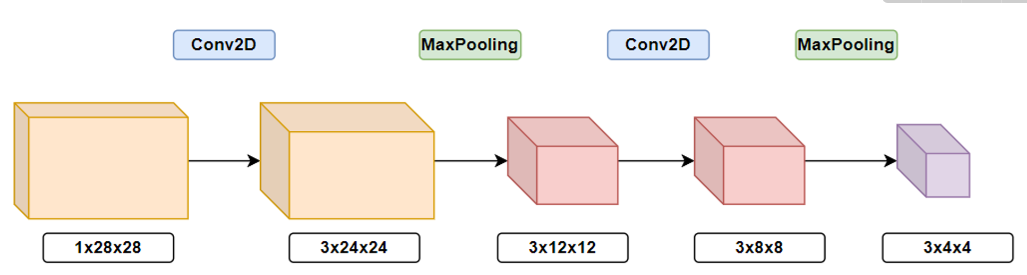
텍스트, 영수증, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

**Fig 3. Resolution 연산 과정 & 과정별 data shape, parameters**

**(해당 프로젝트에서 학습 시 convolution layer의 bias는 없이 진행하였다.)**



**Fig 4. Convolution & MaxPooling process**

**Input data는 1x28x28이며 해당 데이터에 3x5x5 filter를 Convolution 연산을 하게 되면 3x24x24의 데이터가 나오게 된다.**

**해당 Fig 3내 연산 과정 식을 통하면 resolution을 연산 가능하며, 모든 Convolution layer는 stride: 1, padding: 0으로 설정하여 진행하였다.**

스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

**Fig 5. Flatten -> output 과정**

**해당 모델의 Convolution layers를 통과하고 나면 3x4x4의 Tensor가 나오게 된다.**

**해당 Tensor를 flatten하는 과정, 즉 1차원의 배열로 만드는 과정을 통해준다.**

**이 과정은 Convolution layer 과정 이후 분류 과정을 위해 필수적으로 필요한 과정이다.**

**Flatten 과정을 통해 데이터는 48개의 1차원 배열 형태가 되게 되며, 해당 과정이후 FC layers를 통과하고 분류 과정이 일어나게 된다.**

1. **모델 학습 진행**

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

**Fig 6. 학습 파라미터 설정**

**MNIST학습 데이터의 특징은 10개의 분류를 가지고 있고 1x28x28의 데이터 크기를 가지는 데이터 셋이라 할 수 있다.**

**파라미터 학습을 위한 Optimizer는 Adam을 선택하였고 학습 시 데이터 배치 사이즈는 64로 진행하였다.**

**학습에 중요한 파라미터인 Learning Rate는 0.01로 설정해주었다.**

**전체 모델에서의 Activation 단계에서는 RELU 함수를 설정해주었다.**

**Training Epochs는 15로 설정하여 적당히 학습되도록 진행해주었다.**



**Fig 7. 모델의 테스트 데이터셋에 대한 정확도**

**모델의 Training Dataset에 대한 정확도는 약 96퍼에 육박했지만 Test Dataset에 대해서는 약간 떨어진 약 90퍼 정도였다.**

**해당 모델의 학습 Epochs를 높여주면 더 높은 정확도가 나올 것으로 판단하였고 이 프로젝트의 메인 주제는 CNN 모델에 따른 추론 연산 하드웨어 설계이기에 정확도는 이정도로 설정하고 진행하였다.**

1. **INT8 양자화**

**인공지능 모델에 대한 양자화 방법은 여러가지 방안이 존재한다.**

**일반적으로 진행하는 것은 Training-aware Quantization으로, 주로 모델 학습 과정내에서 양자화된 파라미터를 결정시키는 과정이다.**

**하지만 본 프로젝트에서는 Post-Training Quantization을 이용해주었다.**

**간단히 말하자면 학습이 완료된 모델의 파라미터들에 대해 양자화해주는 과정이다.**

**양자화 과정은 어떤 Layer를 어떻게 양자화 할 것인가로 이룬다.**

**먼저 FP32로 설정된 데이터를 INT8로 바꾸는 양자화 방식에 대해 살펴보자.**

1. **Affine Quantization**

**Affine Quantization는 양자화 하고자 하는 숫자들의 집합에서 가장 큰 수 (Max\_value) 와 가장 작은 수 (Min\_value)를 찾고, 해당 수와 원하는 양자화 범위내 최대 값 (Max\_limit)과 최소값 (Min\_limit)를 이용하여 양자화를 진행한다.**

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

**Fig 8. Affine Scale, shift and clip.**

**Fig 8을 보면 양자화 범위내 최대값 최소값 내에 값이 위치하도록 clip이라는 함수가 존재하는 것을 확인할 수 있다.**

**해당 과정에서 x는 shift value인 z와 scale value인 s에 의해 계산된다고 할 수 있다.**

텍스트, 폰트, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명

**Fig 9. Affine 양자화의 예시**

**위 예시와 같이 clip안에 들어갈 값은 q와 동일하다고 할 수 있고 기존의 값이 x에 해당한다고 할 수 있다.**

**z는 위에서 말한 shift value에 해당하며 기존 값 x를 나누는 s는 scale value에 해당한다.**

**여기서 q는 양자화된 수치를 의미한다.**

**위 Affine Quantization을 이용하여 파라미터들을 양자화 하였다.**

**하지만 여기서 중요한 것은 파라미터들의 최대값(Max\_value)와 최소값(Min\_value)를 어떻게 구할 것인가이다.**

**Convolution layer에서의 Max\_value와 Min\_value는 FC layer 내 Max\_value, Min\_value와 다르다.**

**Layer의 종류뿐만 아니라 필터내 채널내 에서도 Max\_value와 Min\_value는 달라진다.**

**이러한 Max\_value, Min\_value는 양자화에 가장 큰 영향을 미치는 변수이고 양자화 과정에서는 필수적으로 기존 데이터의 손실이 발생하기에 최대한 기존 파라미터들의 정보를 유지하면서 양자화 시키는 것이 중요하다.**

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

**Fig 10. Affine 양자화 예시**

**위 표를 살펴보면 Per-Channel과 Per-Tensor의 예시로 나눌 수 있다. Per-Channel은 본 프로젝트의 모델에서의 두 번째 Convolution layer를 살펴보면 3x3x5x5의 필터 구조를 가진다.**

**즉 하나의 필터에 3개의 채널이 존재하는데, 여기서 Max\_value, Min\_value를 구할 때, 3개의 채널에서 따로따로 구하여 채널을 각각 양자화 해주는 방식이 Per-Channel 양자화에 해당하게 된다.**

**위 표에서 나온 논문의 내용을 착안하여 본 프로젝트내 모델에서 또한 Convolution layer에 사용되는 filter의 각 채널을 기준으로 양자화를 진행해주었다.**

**Convolution layer에서의 양자화는 Per-channel 방식으로 진행하였지만 FC Layer내에서 또한 양자화를 진행해야 한다.**

**FC Layer은 Per-Layer 양자화를 진행해주었다. 즉, FC layer는 2개가 존재하는데, 해당 Layer에서 각각 Max\_value, Min\_value를 뽑아내 각 LAYER에 따로따로 양자화를 진행해주었다.**

**해당 양자화 과정들을 통해 기존 Weight와 bias가 가지던 FP32의 Precision은 INT8형태의 Precision으로 바뀌었다.**

**해당 Weight와 Bias를 hexadecimal로 변환시켜 Verilog를 통해 사용할 수 있도록 만들어주었다.**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**Fig 11. 파라미터들의 16진수 변환 & fc1\_weight 파일 내 data**

1. **모델 구조에 따른 하드웨어 작성**

**위 작성한 양자화된 모델에 대한 추론 과정을 Verilog을 통해 RTL Design을 작성한다.**

**RTL Design을 하기위해선 먼저 전체 구조를 먼저 작성해야 하며 전체 구조를 이해한 뒤 모듈화 하여 진행한다.**

텍스트, 스크린샷, 직사각형, 도표이(가) 표시된 사진

자동 생성된 설명

**Fig 12. 전체 하드웨어 구조**

1. **Convolution layer**

**Convolution layer는 input data를 임시적으로 저장할 Buffer와 weight에 따라 연산이 이뤄지는 Calculator로 구성되어 있다.**

**Convolution layer내 filter의 연산은 5x5, 즉 25개의 데이터가 한 번에 연산을 진행하는데 input data는 하나의 이미지 데이터내 하나의 픽셀 값으로 들어오게 된다.**

**따라서 연산의 타이밍을 조정하기 위해 Buffer라는 모듈을 두어 Calculation이 원활하게 이뤄지기 위해 Convolution Layer에 2개의 모듈을 두었다.**

**Verilog 코드내 설명은 해당 input, output, wire, reg 선언된 변수들의 역할이나 구조에 따라 설명을 진행한다.**

텍스트, 스크린샷, 도표, 직사각형이(가) 표시된 사진

자동 생성된 설명

**Fig 13. Convolution Layer내 Conv2D 연산 과정**

텍스트, 스크린샷, 문서이(가) 표시된 사진

자동 생성된 설명

**Fig 14. Convolution layer의 구조**

**Clk: clk는 clock의 신호를 받아오게 된다. Convolution layer 모듈의 모든 연산은 클럭 신호에 동기화 되어 진행된다.**

**Rst\_n: rst\_n은 reset signal을 나타내며 output 연산 결과 및 모듈 내부 변수 값들을 초기화 하는 역할을 한다.**

**Data\_in: data\_in은 이미지 데이터의 픽셀 값을 받아오는 데이터이다. 일반적으로 1x28x28로 Gray scale을 가지는 이미지 데이터인 MNIST 데이터들은 8비트 정수형 형태로 되어 있다. 이에 따라 8비트를 할당시켜 데이터를 받아온다.**

**Conv\_out\_1, conv\_out\_2, conv\_out\_3: 이 세 가지 변수는 convolution layer 1의 weight 및 bias들의 연산 결과에 해당하게 된다. Filter의 수가 3개이기에 output 데이터 또한 채널의 개수와 같이 3개가 된다.**

**Valid\_out\_conv: 이 신호는 연산 결과의 타당성을 나타내 준다.**

**하드웨어 특성상 클럭에 따라 연산의 연속성이 보장되기에 이러한 연속적 연산 처리에서 타당한 데이터만 연산하기위해 해당 신호를 다음 모듈에 연결하여 데이터의 타당성을 보장한다.**

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**Fig 15. convolution layer 내 buffer Reg 변수들**

**Buffer: 해당 변수는 8비트 데이터 140개를 저장하는 배열이라 할 수 있다. 이러한 변수를 선언하는 이유는 앞서 말했듯이 1개씩 들어오는 이미지 데이터에 convolution 연산을 수행하기 위해선 25개의 데이터를 받아야 하기 때문이다.**

**140은 5x28, 즉, 전체 이미지의 column 개수인 28개와 5x5 연산을 맞춰주기 위한 5를 곱해줘 140개의 데이터를 받는 역할을 한다.**

**Buf\_idx : buf\_idx는 buffer index를 의미하며 클럭에 동기화되어 하나씩 상승한다.**

**해당 buf\_idx가 24까지 상승하면 buffer내 valid 신호가 꺼지게 되는데, 이는 한 번에 5x5씩 나가는 데이터가 index가 24일땐, 이미 해당 데이터를 다 보냈기 때문이다.**

**즉, 해당 변수는 데이터 타당성 및 데이터 전송을 위해 존재한다.**

**Buf\_flag, w\_idx, h\_idx: buffer flag, width index, height index는 버퍼내 데이터들의 출력을 맞춰주기 위해 존재한다.**

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

**Fig 16. Convolution layer 내 Calculator**

**Valid\_out\_buf: 해당 신호는 buffer내 데이터 출력이 유효성에 대한 신호를 받아 해당 신호가 1일 때만 데이터 연산을 진행하게 된다.**

**Data\_out[0~24]: 해당 신호들은 buffer의 출력 값인 이미지 데이터 들을 의미한다. 총 25개씩 클럭당 한 번에 들어오게 된다.**

**Conv\_out\_1, conv\_out\_2, conv\_out\_3: convolution layer 연산 결과값을 의미하며 12비트의 연산결과를 출력하게 된다.**

**Valid\_out\_calc: 연산 결과의 타당성을 나타내는 변수라 할 수 있다. Valid\_out\_buf와 동일한 값을 가지도록 설정된다.**

**Calc\_out\_1, calc\_out\_2, calc\_out\_3: 해당 값들은 연산 결과 값을 가지게 되는데 해당 값은 20개의 비트로 나타낸다. 이는 연산 후 오버플로우를 방지하기 위해 높은 비트 값을 할당하였다.**

**실제 Conv\_out\_[1~3]의 비트 수는 12비트를 가지는데 이는 위 Calc\_out\_[1~3]의 비트 중 [19:8]의 비트만 취해서 할당하기 때문이다.**

**부분적 비트를 취하는 이유는 전체 비트 수를 줄이고 인공지능 모델 특성상 정확한 값을 알 필요는 없기에 연산 과정에 낮은 영향을 끼칠 아래 비트 수들을 제거해주고 할당시켰다.**

1. **MaxPooling Layer**

**해당 프로젝트 내에서 Maxpooling Layer은 2x2 Matrix에 대한 Maxpooling을 진행한다. 해당 과정을 통해 불필요한 피연산자를 줄여 연산량을 줄일 수 있으며 성능 향상까지 야기할 수 있는 Layer라 할 수 있다.**

텍스트, 도표, 사각형, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**Fig 17. Maxpooling 연산 과정**

**해당 과정에서는 2x2 Matrix내 가장 큰 값을 출력 값으로 보낸다. 해당 과정을 통해 Input Data의 resolution을 줄 일 수 있다.**

**Maxpooling Layer은 Convolution Layer이 종료될 때마다 실행된다.**

**해당 작업을 통해 전체 Resolution을 줄여 FC Layer의 input data를 줄이게 된다.**

**해당 과정에서 Relu 연산도 진행된다.**

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

**Fig 18. Relu 연산 정의**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Fig 19. Maxpooling Layer Structure**

**해당 구조를 살펴보면 Convolution layer로부터 나오는 3개의 input을 한 번에 받게 된다.**

**Conv1 layer 다음에 있는 Maxpooling Layer의 Input은 3개의 채널을 가진 하나의 Input 데이터 값이다. (Output Shape: 24 x 24 x 3)**

**각 채널을 구분하여 Maxpooling 2D 작업을 진행하게 된다. (Output Shape: 12 x 12 x 3)**

**Conv2 layer 다음에 있는 Maxpooling Layer의 Input은 3개의 채널을 가진 하나의 Input 데이터 값일 것이다. (Output Shape: 8 x 8 x 3)**

**각 채널을 구분하여 Maxpooling 2D 작업을 진행하게 되며 해당 결과는 4 X 4 X 3의 Tensor이다.**

**Buffer 1, buffer2, buffer3 : 해당 buffer들은 Convolution Layer에서 나오는 Output, 3개의 채널에 대해 데이터를 임시적으로 담아두는 곳이다.**

**데이터들이 담아지는 과정에서 특정 개수의 데이터들이 들어오면 Maxpooling 과정 및 Relu 과정이 동시에 진행된다.**

**Pcount: pcount는 Output Data의 resolution에 따라 결정된다. 만약 output data의 Resolution이 12 x 12라면 pcount의 비트값은 [3:0]으로 4개의 비트값을 가지게 된다.**

**만약 Output Data의 resolution이 4x4이라면 pcount의 비트값은 [2:0]으로 3개의 비트값을 가지게 된다.**

1. **FC\_1 LAYER**

**해당 FC\_1 LAYER은 MaxPooling Layer의 output을 Input으로 받게 된다.**

**각 채널에 따라 하나씩 받게 되는데, Flatten 과정을 고려하여 Tensor내 배열에서 앞선 채널들을 먼저 FC\_1 LAYER로 줄 세우게 된다.**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Fig 20. FC\_1 LAYER MODULE RTL DESIGN**

**Clk: clock에 따라 FC\_1 LAYER가 동작하기에 clock 신호를 Input으로 받게 된다.**

**Rst\_n: Layer 내 버퍼들을 초기화해주기 위해 Reset 신호를 설정해주었다.**

**Valid\_in: 해당 신호는 Maxpooling의 output 값이 Valid한지 확인하기 위한 신호이다. 해당 신호가 있을 때만 버퍼 내로 Input data들을 넣게 된다.**

**Data\_in\_1, data\_in\_2, data\_in\_3: 해당 신호들은 Maxpooling layer의 Output에 해당하는 신호들이다. 해당 신호들은 Maxpooling 내 Input data의 각 채널에서 나오는 연산 이후의 출력 값에 해당하게 된다.**

**Data\_out: 해당 데이터는 FC layer 내 연산 과정이 마친 output 값이다.**

**FC\_1\_Layer은 48개의 데이터를 받아 16개의 데이터를 출력한다. 해당 Layer에서는 16개의 output값이 나오게 되는 것이다.**

**연산은 48의 데이터가 각각에 해당하는 weigh값과 연산을 수행하고 16개의 bias내 해당하는 하나의 Bias와 연산이 진행된다. 연산의 결과는 data\_out 신호로 출력된다.**

**FC layer에서는 많은 연산용을 요구하며 해당 연산량에 의해 연산 결과가 많은 bit를 차지할 수 있다.**

**그렇기에 이러한 것을 줄여주기 위해 20개 비트의 연산 결과 중, LSB를 기준으로 8개의 비트를 생략하여 진행하였다. 이렇게 해도 정확도의 손실이 크지 않을 것이라 생각하였다.**

**그 이유는 실제로 인공지능 모델에서 중요한 부분이 정확한 연산의 결과는 중요치 않다는 것이다.**

**어떤 것이 더 크고 작은 지가 모델 정확도에 많은 영향을 미치지, 얼마나 크고 작은 지는 그렇게 중요치 않기에 비트 수를 줄이기 위해 LSB 기준으로 8개의 비트를 생략하였다.**

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명

**Fig 21. FC\_1 LAYER내 연산 과정**

**앞서 말했듯이 FC LAYER에서 또한 Relu 작업을 진행해주었다. 실제로 Relu 작업을 Layer 마다 진행할 필요는 없지만 Relu 작업을 넣어 진행하였다.**

**연산 결과의 음수 여부를 확인하기 위해 연산 결과의 MSB를 이용하였다.**

**Sign-extension 된 연산 결과의 MSB의 값을 통하면 음수 여부를 확인할 수 있다.**

**해당 연산의 결과는 다시 FC\_2 LAYER로 들어가게 된다.**

1. **FC\_2 Layer**

**해당 FC2\_LAYER은 FC1\_LAYER의 output을 Input으로 받게 된다.**

**16개의 input data 값을 받고 10개의 output data를 출력한다.**

**해당 출력된 데이터는 Comparator 모듈로 이동하여 분류를 하게 된다.**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Fig 22. FC\_2 LAYER MODULE RTL DESIGN**

**Clk: Clock 신호에 동기화되어 Input data를 받게 된다.**

**Rst\_n: Reset 신호를 통해 Layer 내 버퍼를 초기화 시킨다.**

**Valid\_in: FC\_1 Layer의 valid\_out 값을 Input으로 받게 된다. 해당 신호가 있을 때만 data\_in 신호가 valid 하여 Buffer에 저장한다.**

**Data\_in: FC1\_Layer의 연산 결과 출력 값을 Input data로 받게 된다. 해당 data는 16개가 존재하며 해당 데이터들에 대해 FC2\_BIAS, FC2\_WEIGHT를 연산하여 출력한다.**

**Data\_out: FC2\_Layer 내 weight와 bias 값을 토대로 연산을 수행하고 해당 결과값을 출력한다.**

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

**Fig 23. FC2\_LAYER내 연산 과정**

**FC2\_LAYER 연산 과정 또한 FC1\_LAYER 연산 과정과 동일하다고 할 수 있다.**

**Layer 내 weight 와 bias를 통해 연산을 진행하고 해당 연산의 MSB를 통해 RELU 연산을 수행한다.**

1. **COMPARATOR**

**해당 과정은 FC2\_LAYER의 출력 값 10개 중 가장 값이 높은 수를 선택하는 과정이다.**

**실제 FC2\_LAYER 출력 값에 SoftMax 연산을 진행하게 되면 0~1 사이에서 10개 분류에 대해 각각의 값이 나오게 된다.**

**이 각각의 값은 원래 값의 크기에 비례하며, 원래 값이 크면 변환 값 또한 크다. 또한 분류의 결과는 변환 값들 중 가장 큰 값의 인덱스에 해당하게 된다.**

**이를 이용하여 해당 모듈에서는 출력 값의 대소를 비교하여 분류 결과를 내놓게 된다.**

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Fig 24. Softmax 연산 예시**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Fig 25. COMPARATOR MODULE RTL DESIGN**

**해당 모듈에서는 clock 및 valid\_in 신호에 따라 FC2\_LAYER의 출력 값을 받게 된다.**

**해당 출력 값을 버퍼에 저장하는데 10개의 신호 모두 저장한 뒤 해당 값들에 대한 비교가 진행된다.**

**Clk: clock에 동기화되어 작동하기에 clock 신호를 입력으로 받아 진행한다.**

**Rst\_n: Reset 신호를 통해 모듈내 존재하는 버퍼를 초기화 시킨다.**

**Valid\_in : 해당 신호는 이전 모듈(FC2\_LAYER)의 출력 값이 Valid한 지에 대해 알려주는 신호이다. 해당 신호가 존재할 때, 버퍼에 data\_in 신호를 채우게 된다.**

**Data\_in: 해당 신호는 FC\_LAYER의 출력 값에 해당하게 된다.**

**Decision: 해당 신호는 10개의 버퍼내 데이터 중 큰 수에 해당하는 Index 값을 출력하는 신호이다.**

**Valid\_Out: 해당 신호가 존재할 때만, decision 신호가 유효함을 나타낸다.**

1. **RTL Design Simulation (Inference Test)**

패턴, 대칭, 직사각형, 패브릭이(가) 표시된 사진

자동 생성된 설명

**Fig 26. MNIST DATA (Number 1)**

**Fig 26은 MNIST 내 존재하는 DATA를 hexadecimal로 나타낸 것이다.**

**28x28 resolution을 가지며 해당 이미지가 나타내는 값은 1을 나타내게 된다.**

**이 데이터를 Custom RTL Design의 입력으로 넣어 시뮬레이션을 실행해보겠다.**

스크린샷, 다채로움, 멀티미디어 소프트웨어이(가) 표시된 사진

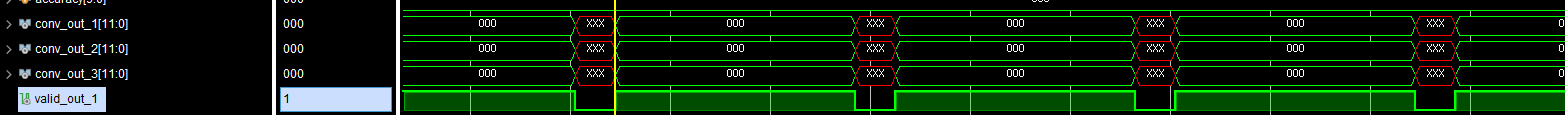
자동 생성된 설명스크린샷, 라인, 다채로움이(가) 표시된 사진

자동 생성된 설명

**Fig 27. Simulation Result**

**해당 이미지를 시뮬레이션 상에 넣게 되면 Decision 단계에서 1이 나옴을 확인할 수 있다. 연산의 입력이 대부분 0 값을 가지기에 연산 과정 자체내 변화가 적은 것을 확인할 수 있었다. 또한 Valid\_out[1~7]의 Signal내 assert 수를 확인하면 전체 과정 내에서 얼마나 데이터가 들어오는 지 확인 가능하다.**

1. **Convolution Layer 1의 시뮬레이션 확인**

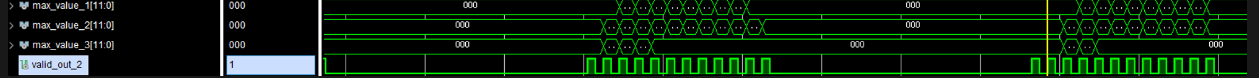


**Fig 28. Convolution layer 1의 연산 시뮬레이션 과정**

**앞서 말했듯이 Fig 26의 데이터내 대부분의 값은 0에 해당한다. 0에 해당하면 어떠한 곱셈 연산을 진행해도 0이 나오기에 해당 결과를 얻을 수 있음을 확인 가능하다.**

**(Conv1, 2 Parameters 내 bias 없이 학습을 진행하였기에 0이 나오게 된다. Bias가 존재하면 데이터의 값이 0이라고 출력 값이 다르게 나올 것이다.)**

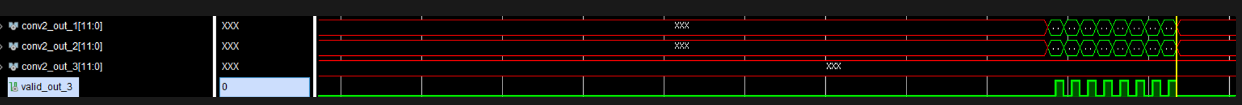
1. **Maxpooling Layer 1의 시뮬레이션 확인**



**Fig 29. Maxpooling layer 1 의 연산 시뮬레이션 과정**

**Maxpooling Layer 1의 output 들이 Valid\_out 신호와 함께 출력된다. 해당 과정에서는 Maxpooling 2x2 뿐만 아니라 Relu 연산까지 수행된다. 고로 결과 값에서는 음수가 없음을 확인할 수 있다. Maxpooling Layer 1의 Output은 하나의 이미지 추론에 대해 432개가 존재함을 알 수 있다.**

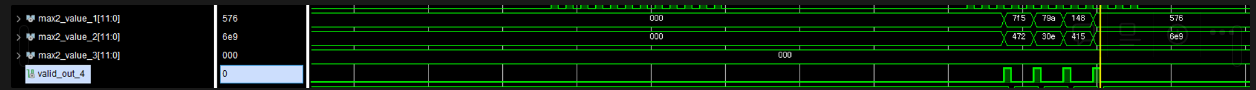
1. **Convolution Layer 2의 시뮬레이션 확인**



**Fig 30. Convolution Layer 2의 연산 시뮬레이션 과정**

**Convolution Layer 2의 output들이 Valid\_out 신호와 함께 출력됨을 확인할 수 있다. Convolution Layer 2의 output은 하나의 이미지 추론에 대해 192개가 존재함을 알 수 있다.**

1. **Maxpooling Layer 2의 시뮬레이션 확인**

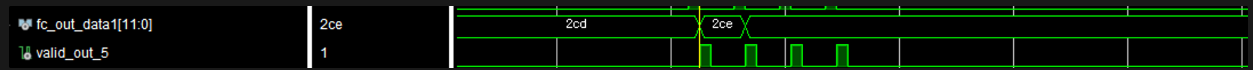


**Fig 31. Maxpooling Layer 2의 연산 시뮬레이션 과정**

**Maxpooling Layer 2의 output들이 Valid\_out 신호와 함께 출력된다. 해당 과정에서는 Maxpooling 2x2 뿐만 아니라 Relu 연산까지 수행된다. 고로 결과 값에서는 음수가 없음을 확인할 수 있다.**

**Maxpooling Layer 2의 Output은 하나의 이미지 추론에 대해 48개가 존재함을 알 수 있다.**

1. **FC LAYER 1의 시뮬레이션 확인**



**Fig 32. FC Layer 1 & FC Layer 2의 연산 시뮬레이션 과정**

**FC Layer 1의 output들이 Valid\_out 신호와 함께 출력된다.**

**FC Layer 1의 output은 하나의 이미지 추론에 대해 16개가 존재함을 알 수 있다.**

**FC Layer 2의 Output들이 Valid\_out 신호와 함께 출력된다.**

**FC Layer 2의 output은 하나의 이미지 추론에 대해 10개가 존재함을 알 수 있다.**

1. **Comparator의 시뮬레이션 확인**



**Fig 33. Comparator의 연산 시뮬레이션 과정**

**해당 모듈에서는 10개의 input을 받아 특정 인덱스를 출력하게 된다. 해당 추론에서 사용된 이미지는 1의 값에 해당되며 인덱스로 1를 출력함을 알 수 있다. 정확도를 측정해보지 않았지만 제대로 추론됨을 확인할 수 있었다.**

**시뮬레이션을 통해 전체 과정이 제대로 연산됨을 확인하였고 Valid 신호들을 통해 각 단계의 연산 결과확인을 쉽게 할 수 있었다.**

**양자화된 가중치에 대해 연산을 진행하였지만 실제 FC Layer의 Weight, bias에 대해선 양자화 방식에 대해 정확히 파악하지 못하여 정확도에 많은 영향을 줄 수 있다고 생각한다.**

**참고 문헌**

[**https://www.ibm.com/topics/convolutional-neural-networks**](https://www.ibm.com/topics/convolutional-neural-networks)

[**https://velog.io/@robert-lee/**](https://velog.io/@robert-lee/Tensorflow-MaxPooling-%EA%B3%BC-Average-Pooling-%EC%9D%98-%EC%B0%A8%EC%9D%B4%EB%8A%94)

[**https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=handuelly&logNo=221824080339**](https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=handuelly&logNo=221824080339)

[**https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60**](https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60)

[**https://arxiv.org/pdf/2106.08295.pdf**](https://arxiv.org/pdf/2106.08295.pdf)