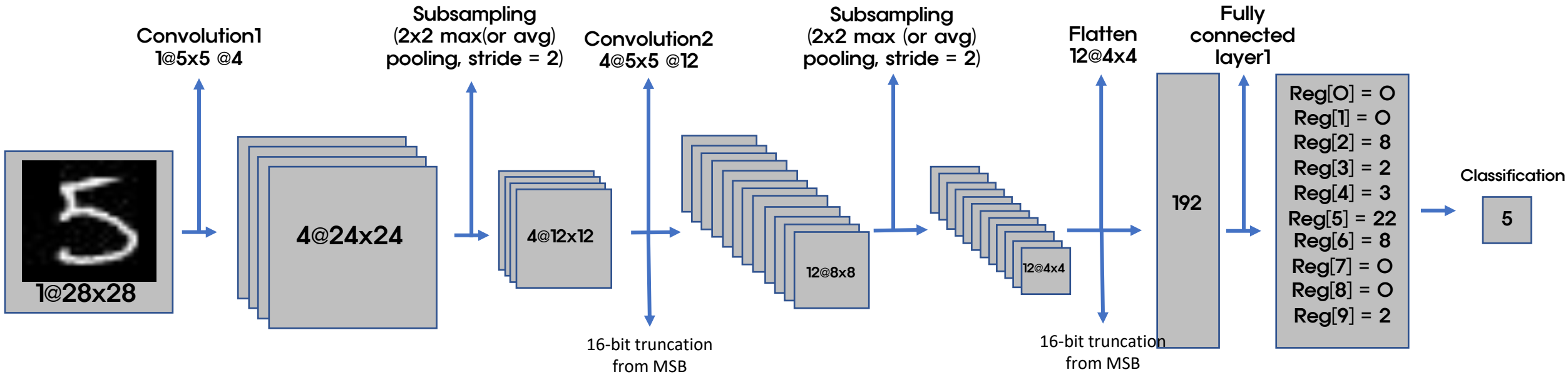
This project is a private project, suitable for beginners like me who study CNN Verilog design

Lenet5 CNN Structure

It has a lighter structure than the original version.



1. 논문들과 같이 병렬 연산을 줄여 리소스를 줄인다.
2. 하지만 CNN 구조 자체를 상당한 경량화를 함으로써 다른곳에 쓰일 리소스를 병렬연산에 좀 더 쓸 수 있다.
3. 5x5 커널은 병렬연산(25개의 Mac 사용)으로 진행되며 Convolution 2 에서 연산은 out channel1 -> out channel2 -> out channel3 -> out channel4 -> … out channel12 로 순차적 진행된다. 다만 각 채널의 연산(in channel1, in channel2, in channel3, in channel4 는 병렬(동시)연산된다).
4. 실시간으로 maxpooling, relu function 을 지나서 FC layer 로 입력되며 이 layer 에서 즉각 계산하여 버퍼에 저장하고 최종적으로 classification 하기 위한 (0 부터 9) 내부 maxpooler 로부터 최종 출력이 나오게 된다.

----------구조를 변경하지 않고 즉시 시뮬하실분----------

제가 설계한 파일들 그대로만 사용해서 한번 시뮬레이션 결과를 보고싶다 하시는 분들은,
Test_image.zip 압축해제 하신 뒤 본인이 원하는 숫자의 mem 파일을 넣어주신 뒤,
lenet_tb_fix_weight.v 파일에서 $readmemh("HW_test_num3_1.mem", in_fmap); 부분에서 파일이름만 변경해주시면 됩니다.
예) 4 입력시, $readmemh("HW_test_num4_1.mem", in_fmap); 혹은 $readmemh("HW_test_num4_2.mem", in_fmap); 이런식

Test_image.zip 파일은 숫자0의 이미지 10개를 각각 데이터화 한 mem 파일 10개, txt 파일 10개,
숫자1의 이미지 10개를 각각 데이터화 한 mem 파일 10개, txt 파일 10개,
.....
숫자9의 이미지 10개를 각각 데이터화 한 mem 파일 10개, txt 파일 10개 로 구성되어 있습니다.

----------구조를 변경할 경우----------

모든 코드는 자동화? 로 되어있다. 따라서 각 레이어의 채널 수를 변경하고 싶거나, 각 레이어 마다 출력 bit 수를 truncate 하는데 이를 조정하고 싶다면 param_clog2.vh 파일에서 해당 부분을 수정하면 된다.
이 후 이에 맞는 가중치와 바이어스 값을 구해야 하는데 이는 MNIST.ipynb 파일에서 코드를 수정하면 된다.
이 코드는 크게 5장의 코드로 되어있다.(구글에서 스윽 할 수 있는 코드들이고 입맛에 맞게 좀 수정하면 된다)

1번째장 : MNIST training 및 testing 하는 코드이다. 본인이 원하는 Lenet CNN 구조(채널의 수)에 맞게 코드를 수정하면 된다.

2번째 장 : test image jpg 이미지파일들을(구글에서 받으면 된다) 0~255 범위의 28x28 크기인 txt 파일(SW를 위한)과 동일한 범위와 크기이지만 표현법인 hex인 mem 파일(HW를 위한)을 저장하기 위한 코드이다.

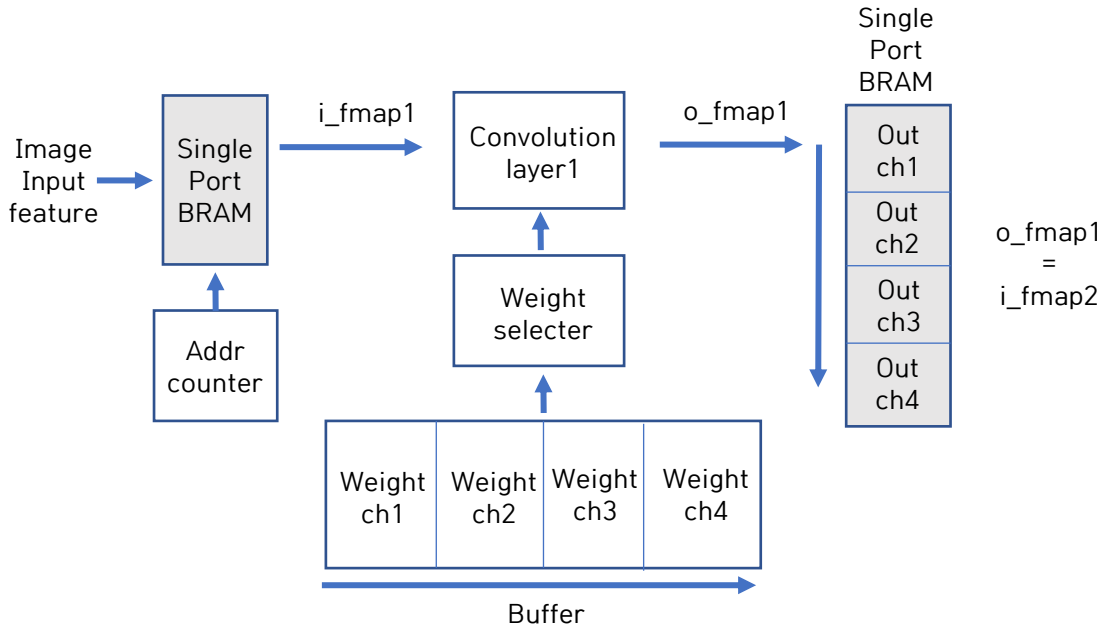
3번째 장 : Convolution layer 1 의 가중치들을 출력하고 저장하기 위한 코드이다.
기존 가중치에 x128 을 해주어 int 형으로 변경해준다. 이 때 127 보다 큰 값을 8bit 로 받아들이면 HW 에서는 이를 음수로 처리한다(127 = 0111 1111 이고 128부터 MSB 가 1이 되므로). 따라서, 10번정도의 트레이닝을 통한 각 가중치를 살펴본다. 그 결과 127을 넘는 양수 가중치는 10개 미만이었고, 그 값들 각각의 차이마저 미비한 수준이었으므로, 127을 넘을 경우 단순히 127로 고정해버린다.(1~10정도의 오차는 있지만 오버플로우는 막는다)
이 후 가로로 쌓아서 한번에 배열을 합치는 np.vstack 을 사용해서 합친 후 저장하면 매우 편하다.

4번째 장 : Convolution layer 2 의 가중치들을 출력하고 저장하기 위한 코드이다. Layer 1 의 경우, 입력 이미지가 1 채널이기에 가중치의 순서가 헷갈리지 않지만, Layer 2 의 경우, 입력 feature map이 4채널이다. 이를 유의하여 본인의 SW 코드에 맞도록(본인만의 코드가 있다면) stack 을 한 뒤 저장한다. 본인의 경우 conv.cpp 가 작성한 SW 코드이다.

5번째 장 : fc layer 의 가중치들과 바이어스 값을 출력하고 저장하기 위한 코드이다. 이 때 바이어스는 16bit 이므로 배수가 다른 점을 유의한다.

MNIST.ipynb 의 경우 반드시 첫 장 코드부터 실행 한 뒤, 결과가 나오는 것을 본 후 2 장부터 실행하여 가중치,바이어스 값을 저장하면 된다.

간단한 구조도
Convolution layer1 의 경우



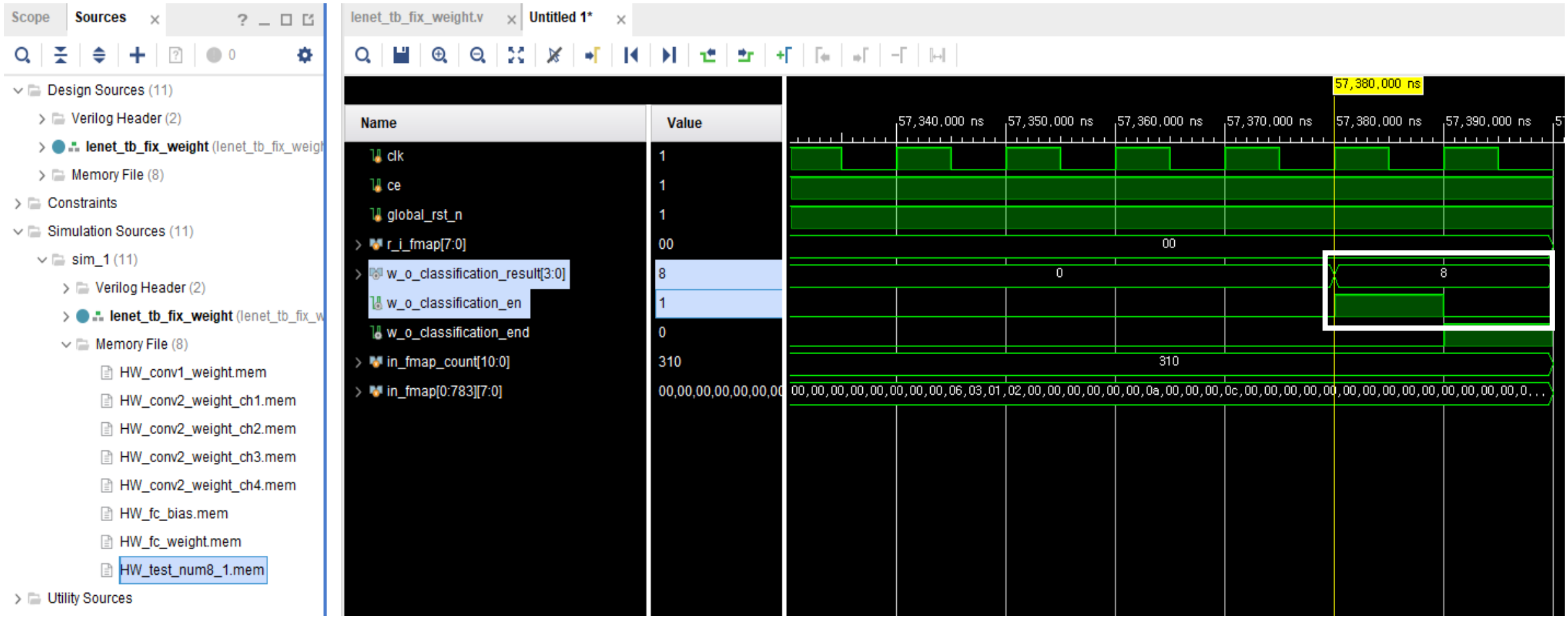
Convolution layer2 는 layer1 의 경우에서 2가지가 달라진다.

1. kernel(weight)의 채널수
2. Output kernel의 수

당연히 설계가 조금은 다르지만 동작 메커니즘은 거의 같다.

이 후 FC layer는 convlayer 2 의 아웃풋을 실시간 입력받아 mac -> 버퍼에 저장을 반복하다 convlayer2 가 종료되면 그 때부터 classification 하게 된다.

예시로 tb file 에서 HW_test_num8_1.mem 파일을 인풋으로 넣었을 경우 출력 결과 8 로 정상적으로 나옴
현재까지 100개 정도 실행했는데 2개? 3개? 제외하고 정확한 답을 내놓는 정확성을 보이지만,
더 많은 결과를 확인하거나, 트레이닝을 다시 한 뒤, 새로 얻은 가중치로 하면 또 다른 정확성을 보일 수 있음. 제대로 된 가중치를 얻어내는 것도 중요



Synthesis 결과

1개의 warning 이 존재하지만, 이는 FC layer 에서 bias bit를 30bit 로 만들어 주기 위함에서 14bit 증가됐다고 뜨는 일반적인 메시지 무시가능

Project Summary

Overview | Dashboard

Display name:	Zybo Z7-20
Board part name:	digilentinc.com:zybo-z7-20:part0:1.1
Board revision:	B.2
Connectors:	No connections
Repository path:	D:/tools/Vivado/2022.1/data/boards/board_files
URL:	https://digilent.com/reference/programmable-logic/zybo-z7/start
Board overview:	Zybo Z7-20

Synthesis

Status:	✓ Complete
Messages:	1 warning
Part:	xc7z020clg400-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	Automatically selected checkpoint

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z020clg400-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Run Implementation to see DRC results

Timing

Run Implementation to see timing results

Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	24876	53200	46.76
FF	35428	106400	33.30
BRAM	5.50	140	3.93
DSP	110	220	50.00
IO	18	125	14.40
BUFG	1	32	3.13

Power

Run Implementation to see power results