

결 과 보 고 서

FPGA 기반 경량형 MLP 추론 가속기 설계 및 임베디드 분류 시스템 구현

팀명	EESOFT
팀원	김준호
	이준기

전남대학교 IDEC 2025 스마트 전자회로설계 챌린지

목차

1. 설계 개요.....	3
가. 설계 배경.....	3
나. 설계 목표.....	3
2. 설계 과정.....	5
가. 알고리즘 구성.....	5
나. 설계 회로 구성	6
3. 설계 회로 검증	12
4. 결론	19
가. 최종 결론.....	19
나. 한계	20

1. 설계 개요

가. 설계 배경

- 산업 전반에서 다양한 발전을 통해 인공지능 기술이 급속히 확산되면서 Edge device 나 임베디드 환경에서는 효율적으로 AI 추론을 수행 가능한 경량형 하드웨어 플랫폼(TinyML)의 중요성이 커지고 있다. 기존의 대형 GPU 기반 AI 시스템은 높은 연산 성능을 제공하지만 전력 소모가 크고 하드웨어 자원이 제한된 환경에서는 적용이 어렵다는 문제가 있다. 이러한 한계를 극복하기 위해서, 프로그래밍의 유연성과 병렬 처리 능력을 동시에 갖춘 FPGA가 새로운 대안으로 주목받고 있다.

FPGA는 병렬 연산에서 구조적으로 효율성이 뛰어난 하드웨어로서, 행렬 곱셈이나 다양한 연산이 반복되는 딥러닝에 특화된 하드웨어다.

특히 소형화된 다층 퍼셉트론(multi layer perceptron) 구조는 CNN 등 복잡한 모델보다 단순하면서도 학습된 가중치를 하드웨어로 적용하여, 실시간 분류 시스템의 구조를 구현하기에 적합하다.

본 프로젝트에서는 이러한 FPGA 환경에 TinyML 개념을 적용하여, 학습이 완료된 MLP 모델의 가중치를 기반으로 실시간 추론을 수행하는 FPGA 기반 MLP 분류기를 구현하였다.

또한, 단순히 연산만 수행하는 수준을 넘어서, UART 통신을 통해 외부 입력 데이터를 주고받으며 실시간으로 LED 및 시리얼 포트에 분류 결과를 표시해 보았다.

이를 통해서 학습을 제외하여 동작하는 추론 하드웨어를 구현하고, 향후 다양한 타겟 데이터를 바탕으로 학습까지 가능한 ML의 구조적 기반을 마련하는 것을 목표로 하였다.

나. 설계 목표

- 본 프로젝트의 목표는 FPGA 단일 칩에서 동작하는 MLP 추론 시스템을 구현하는 것이다. 이를 목적으로 설정하여, 입력 데이터를 받아 연산을 수행하고, 분류 결과를 실시간으로 출력할 수 있는 하드웨어 기반 추론 구조를 설계하였다.

첫 번째 목표로는 이중 동작 모드를 구현하는 것이다. SW0=0일 때, 기능 검증 및 통신 오류를 확인하기 위해 FPGA 내부에 LFSR(Random Generator)모듈을 만들고 8 바이트 크기의 입력 데이터를 자동으로 생성한다. SW0=1일 때는 UART 통신을 통해 PC에서 전송되는 외부 입력 데이터를 받아 연산을 수행한다. 두 입력이 mux와 SW0를 통해서 구분되고 각 layer에서 rom의 데이터를 읽을 수 있는 주소 배열을 생성하여 연산을 진행하게 한다. 이를 통해 스위치에 따라 독립적인 내부 테스트와 외부 데이터 기반 추론을 모두 수행할 수 있는 구조를 만드는 것을 목표로 삼았다.

두 번째 목표는 MLP 기반 하드웨어 추론 구조의 효율적 구현이다. 은닉층은 8개의 노드, 출력층은 2개의 노드로 구성된 2-Layer MLP를 verilog로 직접 설계하고 각 노드는 입력과 가중치의 곱셈 누적 연산을 고정소수점 기반으로 수행하며, 활성화 함수는 ReLU 함수를 논리회로 구현하였다. 이를 통해 불필요한 부동소수점 연산을 제거하고 자원 사용을 최소화하였다. 모든 가중치(weight)와 바이어스(bias)는 ROM에 저장되어 이를 통해 ROM 데이터만 교체하면 다른 MLP 구조에도 적용이 가능하도록 설계하였다.

세 번째 목표는 실시간 시각화 및 결과 확인 기능을 구현하는 것이다. 연산이 완료되면 Done 신호가 활성화되며, Class 결과에 따라 LED 가 점등된다. Class0 은 적색, Class1 은 녹색으로 표시되어 분류 결과를 직관적으로 확인 가능하다. 동시에 UART 를 통해 입력 벡터($x_0 \sim x_7$), 연산 결과(y_0, y_1), 분류 결과(class), 스위치 상태(sw)를 포함한 프레임이 PC로 전송되어 python을 통해서 실시간으로 확인할 수 있다. 이를 통해 사용자는 FPGA 내부 연산의 과정을 시각적으로 확인 가능하다..

네 번째 목표는 모듈화 및 확장성 확보이다. 전체적인 모듈은 Weight/Bias ROM, Hidden Layer, Output Layer, UART 송수신기, Controller FSM 등으로 구성되어 있으며, 각 블록을 독립적인 모듈로 분리하였다. 이러한 구조를 통해서 향후에 구조와 가중치 변경, 클럭 변경 등 다양한 조건에 대응할 수 있는 유연성을 제공한다. 향후 CNN-RNN 구조로 확장 시에도 모듈 인터페이스 수정을 통해서 쉽게 적용이 가능하다.

마지막 목표는 실시간 반응성과 안정성 확보다. 본 프로젝트에서는 클럭 주파수를 125 MHz로 설정하였고, 입력에서 출력까지의 전체 latency 은 약 $5 \mu s$ 미만으로 설계하고, 모든 신호는 파이프라인 구조로 연결되게 설계하여 연속적인 입력에도 병목 현상 없이 안정적으로 동작하게 하는 것이 목표다.

결론적으로, 본 프로젝트의는 단순한 신경망 연산이 아니라, FPGA 기반의 독립형 MLP 추론 하드웨어를 구축하여 소형화된 임베디드 시스템의 효율성을 보여주고자 한다. 이번 프로젝트를 통해서 학습 기능이 없는 환경에서도 효율적인 추론을 수행이 가능하고, 저전력, 고속 모듈화된 구조를 바탕으로 향후 다양한 TinyML 응용 분야에 적용할 수 있는 확장 가능한 하드웨어를 완성하고자 한다.

2. 설계 과정

가. 알고리즘 구성

- 본 프로젝트에서는 $8 \times 8 \times 2$ 구조의 MLP 를 FPGA 상에서 고정소수점 방식으로 구현하여 실시간으로 입력 데이터를 분류하도록 설계했다. 연산은 8 비트 signed 정수 연산 기반으로 수행되며, 누산기의 비트 폭은 20 비트, 스케일 정규화를 위해 SHIFT 값은 6 으로 설정하였다. 전체 구조는 입력 획득부, 은닉층 연산부, 출력층 연산부, 제어 상태기계(FSM), 그리고 결과 표시 및 통신부로 구성된다.

시스템의 동작은 하나의 추론 사이클(입력 준비→은닉층 계산→출력층 계산→결과 출력→대기)을 기준으로 수행한다. 우선 입력 데이터를 획득한 뒤, 은닉층에서 가중치와 바이어스를 이용한 곱셈-누산(MAC) 연산을 수행하고, 비선형 활성화 함수인 ReLU 를 적용한다. 이후 출력층에서 다시 한 번 MAC 연산을 수행하여 두 개의 클래스 점수를 계산하고, 이 결과를 비교하여 분류를 결정한다. 분류 결과는 LED 로 시각적으로 표시되며 동시에 UART 를 통해 PC 로 전송된다. 마지막으로 FSM 은 모든 처리를 마친 뒤 자동으로 대기 상태로 복귀한다.

입력 데이터는 두 가지 모드에서 제공된다. 첫 번째는 내부 LFSR 기반의 랜덤 생성 모드이며, 두 번째는 UART 통신을 통한 외부 입력 모드이다. 사용자는 스위치(SW0)를 통해 동작 모드를 선택할 수 있다. SW0 가 0 일 때는 LFSR 이 버튼 입력(btn0)을 트리거로 8 바이트 크기의 입력 벡터를 생성하고, SW0 가 1 일 때는 UART 수신부가 PC 에서 전송된 데이터를 수신하여 입력으로 전달한다. 두 경우 모두 입력은 64 비트 벡터 형태로 은닉층으로 전달된다.

연산 제어는 controller_fsm 모듈에 의해 수행된다. FSM 은 start_infer 신호가 활성화되면 hidden_layer 의 연산을 시작하도록 hid_start 펄스를 출력하고, hidden_layer 가 완료 신호인 hid_done 을 발생시키면 output_layer 를 구동하기 위해 out_start 를 출력한다. output_layer 가 완료되면 FSM 은 done_all 신호를 발생시켜 전체 연산이 종료되었음을 알린다. 시스템은 Done 상태를 한 클럭 유지한 뒤 자동으로 IDLE 상태로 복귀하며, reset 신호가 입력될 경우 어느 상태에서든 즉시 IDLE 상태로 복귀한다.

가중치와 바이어스는 모두 ROM 에 저장되어 있으며, 은닉층은 weight_rom_hidden 과 bias_rom_hidden 에서, 출력층은 weight_rom_out 과 bias_rom_out 에서 데이터를 읽어온다. 각 ROM 은 주소 입력에 따라 병렬적으로 연산을 위해 입력 되며, 은닉층의 주소는 0 부터 7 까지, 출력층의 주소는 0 과 1 두 개로 구성된다. 이러한 구조를 통해 동일한 입력에 대해 항상 동일한 출력이 보장되는 결정적 연산 환경을 유지한다.

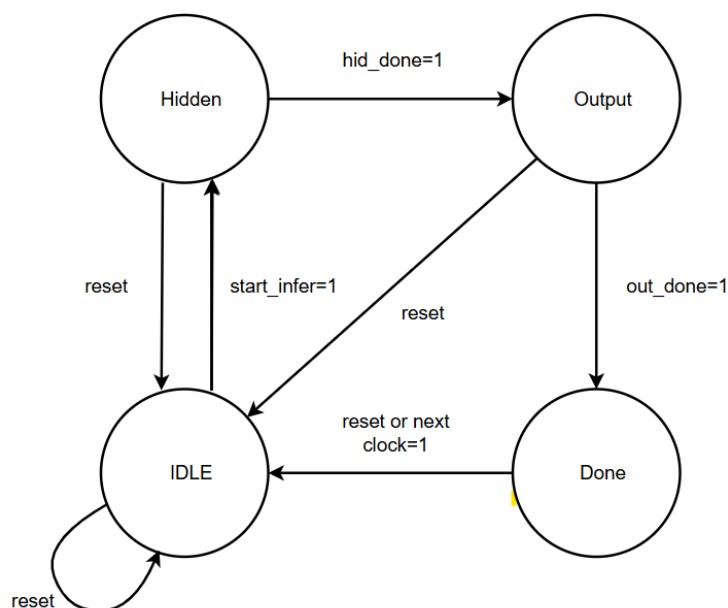
모든 데이터는 8 비트 signed 정수로 표현되며, MAC 연산 결과는 20 비트 누산기로 처리된다. 각 연산 결과는 SHIFT(=6) 연산을 통해 비트 스케일을 맞춘 뒤 8 비트로 다시 변환된다. 은닉층에는 ReLU 함수가 적용되어 음수 값을 0 으로 처리하며, 출력층은 ReLU 를 사용하지 않고 선형 연산 결과를 그대로 전달한다.

연산이 완료되면 결과는 즉시 LED 를 통해 출력된다. class0 또는 class1 중 높은 값을 갖는 쪽에 따라 LED0 또는 LED1 이 점등되며, 동시에 done LED 가 켜져 추론 완료를 표시한다. 또한 UART 송신부는 A7 프레임 형태로 결과를 PC 로 전송한다. 프레임 구조는 [A7][입력 8 바이트][y0][y1][class][sw][5A] 순으로 구성되어 있으며, PC 에서 이를 수신하여 결과를 시각화하거나 기록할 수 있다.

시스템은 예외적인 상황에서도 안정적으로 동작하도록 설계되었다. 잘못된 ROM 주소 접근이나 비정상적인 시작 신호가 입력될 경우, 모든 출력은 0 으로 초기화되고 FSM 은 자동으로 IDLE 상태로 복귀한다. 이러한 예외 처리 구조를 통해 FPGA 내부에서의 연산 오류나 타이밍 이상에 의한 비정상 출력을 방지하였다.

나. 설계 회로 구성

1) FSM controller 구조



- FSM 의 구조는 네 개의 상태(IDLE, Hidden, Output, Done)로 구성되며, 한 번의 추론 과정이 이 순서에 따라 진행된다.

IDLE 상태는 초기화 이후 또는 연산이 완료된 후 대기하는 기본 상태로, reset=1 신호가 들어오면 이 상태로 복귀한다. 입력 데이터가 준비되어 start_infer=1 신호가 발생하면 Hidden 상태로 전환된다.

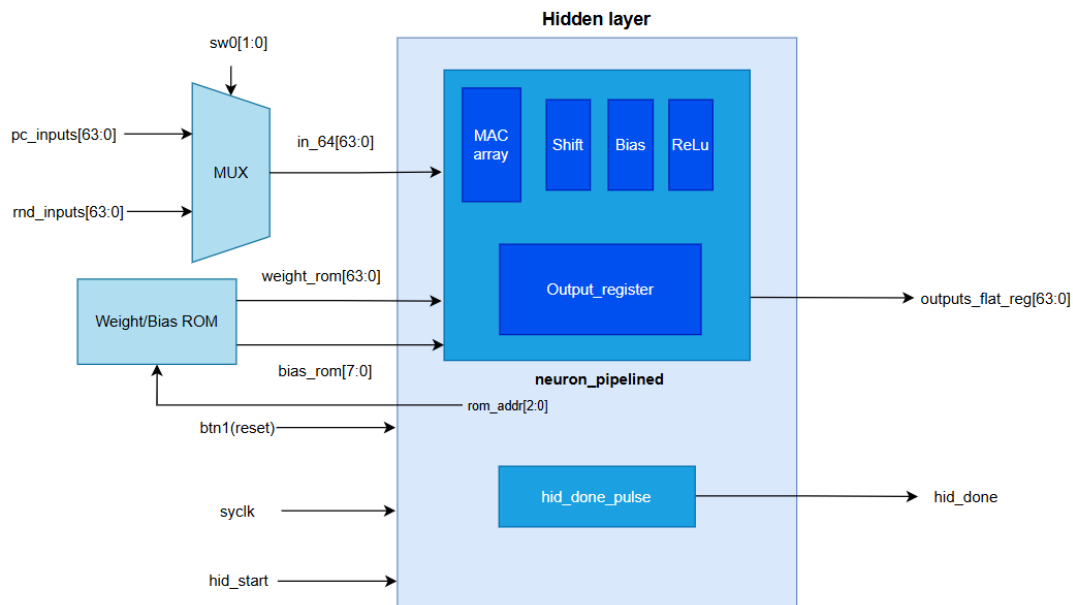
Hidden 상태에서는 입력 데이터(in64)와 ROM 에 저장된 가중치, 바이어스를 이용해 8 개의 뉴런이 병렬로 곱셈-누산(MAC) 및 ReLU 연산을 수행한다. 각 뉴런의 누산 결과는 SHIFT(6)만큼 비트 시프트되어 스케일이 조정된다. 모든 연산이 완료되면 hid_done=1 신호가 발생하며, FSM 은 Output 상태로 이동한다.

Output 상태에서는 은닉층의 결과(hid_out)를 입력으로 받아 2 개의 출력 뉴런 연산을 수행한다. 가중치(weight_rom_out)와 바이어스(bias_rom_out)를 사용하여 MAC 연산을 수행하고, 두 출력결과(y0, y1)를 계산한다. 연산이 완료되면 out_done=1 신호가 발생하며 Done 상태로 전환된다.

Done 상태에서는 연산 결과가 LED 와 UART 송신 블록으로 전달된다. comparator 가 y0, y1 을 비교해 class 결과(0 또는 1)를 결정하며, LED[0]에는 class, LED[1]에는 완료 신호가 점등된다. UART 송신 블록은

[A7][입력 8B][y0][y1][class][sw][5A] 형식으로 결과를 전송한다. 출력이 완료되면 FSM 은 다음 클록에서 다시 IDLE 상태로 돌아와서 새로운 입력을 받을 준비를 한다.

2) Hidden layer 구조



- Hidden Layer 는 MLP 구조에서 입력 데이터를 받아 비선형 변환을 수행하는 첫 번째 연산 계층으로, 입력 데이터와 ROM 에 저장된 가중치 및 바이어스를 이용하여 파이프라인을 통해 뉴런 단위의 병렬 연산을 수행한다. 본 설계의 Hidden Layer 는 입력 선택부, 연산부, 제어부의 세 블록으로 구성되어 있으며, 각 블록은 독립적으로 동작하면서 공통 클록과 제어 신호를 통해 순차적으로 연산을 수행한다.

입력 선택부는 외부 입력과 내부 테스트 입력 중, MUX 를 통해서 하나를 선택하는 역할을 담당한다. 외부 입력 모드에서는 PC 로부터 UART 통신을 통해 전달된 입력 데이터(pc_inputs[63:0])가 사용되며, 내부 테스트 모드에서는 LFSR 을 통해 생성된 8 바이트 크기의 랜덤 데이터(rnd_inputs[63:0])가 사용된다. 이 두 입력은 스위치 SW0[1:0]에 의해 제어되는 MUX 를 통해 선택되며, 선택된 신호는 in_64[63:0] 형태로 Hidden Layer 내부로 전달된다.

가중치 및 바이어스 데이터는 독립된 Weight/Bias ROM 에 저장되어 있으며 필요한 데이터를 읽는다. ROM 내부에는 각 뉴런이 사용할 8 개의 Weight(w_flat_rom[63:0])와 1 개의 Bias(b_data_rom[7:0])가 저장되어 있으며 ROM 은 설계 시점에 고정된 가중치와 바이어스로 초기화되고, 합성 후에는 값이 변경되지 않는다. 따라서 동작 중 동일 입력에 대해서는 항상 일관된 연산 결과가 나온다.

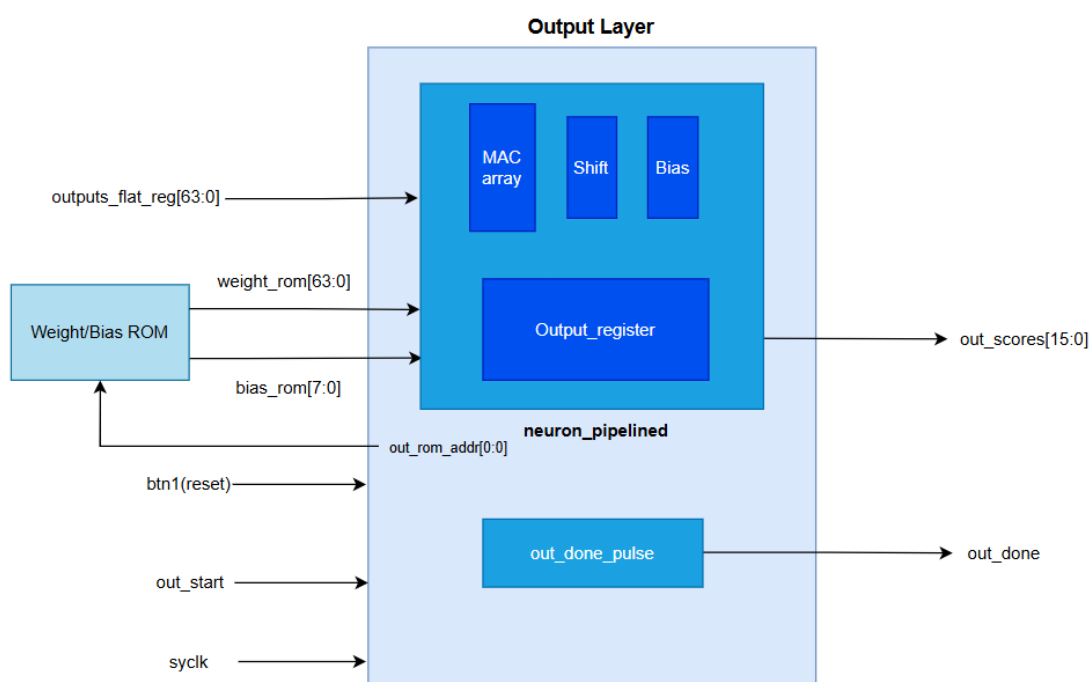
연산부에서는 입력 데이터와 ROM 으로부터 읽어온 가중치를 이용해 MAC 연산을 수행한다. MAC Array 는 8 개의 입력(x0~x7)과 8 개의 가중치(w0~w7)를 동시에 곱한 후 누산 하여 하나의 뉴런 출력을 계산한다. 누산 결과 최대 17 비트로 나오고 약간의 여유를 위해서 20 비트 폭으로 표현하였으며, 고정소수점 스케일

조정을 위해 6 비트 시프트연산이 적용된다. 이후 ROM 에서 읽은 바이어스가 더해지고, $\text{ReLU}(\max(0, x))$ 가 적용되어 음수 결과는 0 으로, 양수는 그대로 통과한다. 이 결과는 ± 127 범위 내에서 포화 연산(saturation) 처리된 후 8 비트 단위로 정렬되어 출력 레지스터(Output_register)에 저장된다. 모든 뉴런 연산이 완료되면 결과는 `outputs_flat_reg[63:0]` 형태로 합쳐져 출력된다.

제어부는 Hidden Layer 의 전체 연산 흐름을 제어하는 모듈로, 연산의 시작, 완료 및 리셋 동작을 담당한다. FSM 컨트롤러에서 연산 시작 신호인 `hid_start` 가 hidden layer 로 전달되면, 내부 주소 카운터가 증가하면서 ROM 의 주소(`rom_addr[2:0]`)를 순차적으로 갱신하고, 각 뉴런의 연산이 순차적으로 수행된다. 모든 연산이 종료되면 `hid_done_pulse` 블록에서 `hid_done` 신호를 발생하고, 이 신호는 상위 FSM 컨트롤러에서 다음 단계인 Output Layer 연산을 시작하는 트리거로 사용된다. 또한 `btn1(reset)`가 입력 될 경우, 내부 레지스터 및 제어 신호는 즉시 초기화되어 IDLE 상태로 복귀한다.

이와 같은 구조를 통해서 Hidden Layer 는 입력부터 결과 출력까지의 모든 연산 단계는 클록 신호에 맞춰 순차적으로 진행되며, 파이프라인 구조를 통해 각 단계가 연속적으로 실행되어 전체 처리 지연을 최소화하였다. 결과적으로 Hidden Layer 는 MLP 하드웨어 구조 내에서 비선형 변환을 수행하는 중요한 연산 블록으로서 연산 결과와 반복 가능한 출력 특성을 보여준다.

3) output layer 구조



- Output Layer 는 MLP 구조에서 최종적인 분류 결과를 생성하는 단계로, Hidden Layer 에서 출력을 기반으로 두 개의 클래스(class0, class1)에 대한 분류 결과값을 계산하는 역할을 한다. Output Layer 는 연산부, 제어부로 구성되어 있으며, 각 구성요소는 클록 및 제어 신호를 통해 순차적으로 동작하여 최종 출력을 생성한다.

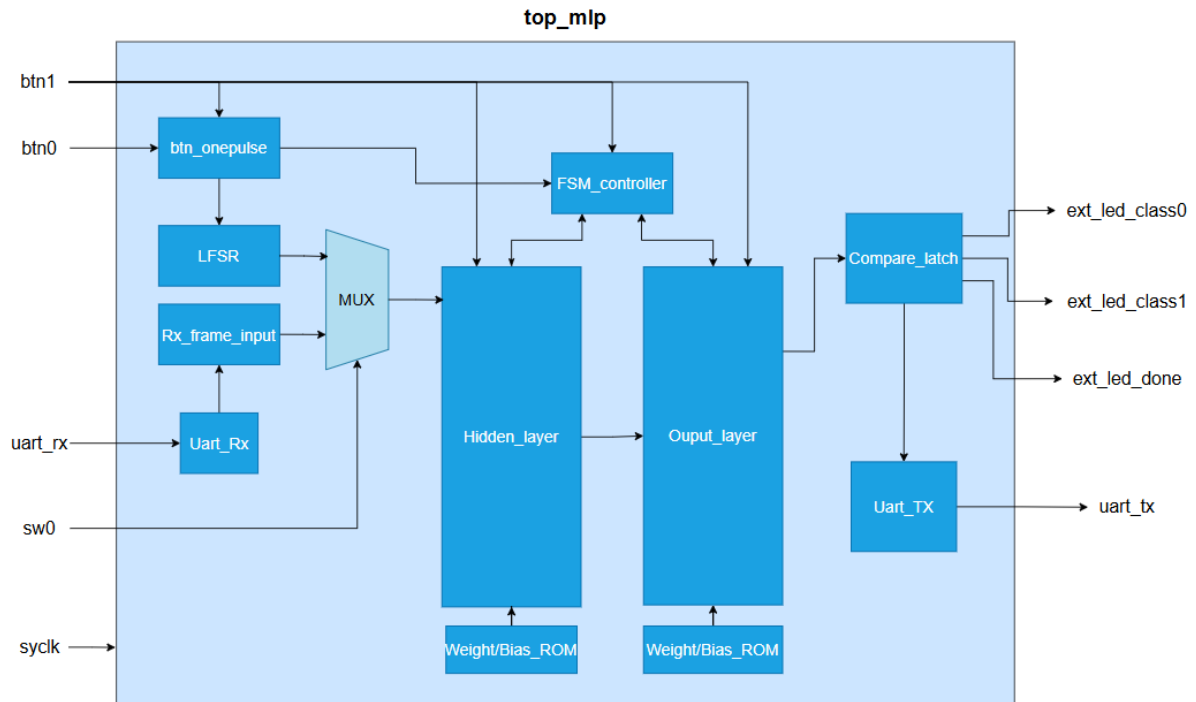
Output Layer 의 입력은 Hidden Layer 의 출력 신호인 outputs_flat_reg[63:0]가 입력으로 들어온다. 이 64 비트 데이터는 8 개의 뉴런 출력을 8 비트 단위로 묶은 값이며, Output Layer 내부의 neuron_pipelined 모듈로 전달된다. 연산에 필요한 가중치와 바이어스는 외부의 Weight/Bias ROM 에서 제공되고 ROM 내부 데이터는 설계 시 고정되어 있으며, 연산 중 수정되지 않기 때문에 동일한 입력에 대해 항상 동일한 결과가 나온다.

연산부는 neuron_pipelined 내부에 구현되어 있으며, 곱셈-누산(MAC) 구조를 기반으로 한다. 입력 데이터와 가중치가 곱해진 후 8 개의 항이 모두 더해져 누산되며, 누산 결과는 은닉층과 마찬가지로 최대 17 비트에 여유 3 비트를 두어 20 비트로 표현된다. 이 결과는 고정소수점 스케일 보정을 위해 6 비트 시프트 연산을 거친 뒤 바이어스가 더해진다. Output Layer 는 Hidden Layer 와 달리 ReLU 비활성화 함수를 적용하지 않으며, 음수값도 그대로 유지하여 두 클래스 간의 상대적인 크기를 비교할 수 있도록 한다. 결과값은 8 비트 단위로 포화연산(saturation) 거쳐 Output_register 에 저장된다. 두 뉴런의 출력(y0, y1)은 순차적으로 out_scores[15:0]의 배열로 저장되며, 이는 상위 FSM 제어기로 전달된다.

제어부는 출력층 연산의 시작, 진행, 종료를 제어하는 블록으로, out_start 신호가 활성화되면 내부 FSM 이 동작을 시작하며, 이 신호에 따라 ROM 에서 대응하는 가중치와 바이어스 데이터를 병렬적으로 동시에 읽어 연산 모듈(MAC Array)에 전달한다. 이후 모든 뉴런의 연산이 완료되면 out_done_pulse 블록에서 out_done 신호가 1 클록 폭으로 발생하여 FSM 에 전달된다. 또한 btn1(reset) 신호가 입력되면 모든 내부 레지스터와 제어 신호가 즉시 초기화되어, 출력층은 연산 중이더라도 바로 IDLE 상태로 돌아오도록 동작한다.

결과적으로 Output Layer 는 입력 데이터를 기반으로 두 클래스에 대한 분류 값을 계산하고, 연산 완료 시 out_done 신호를 발생시켜 다음 단계로 제어 신호를 전달한다. 파이프라인 구조를 적용하여 연산 단계를 분리함으로써, 연산 지연을 최소화하고 125 MHz 의 환경에서도 안정적으로 실시간 추론이 가능하도록 설계되었다.

4) top mlp 구조



- Top MLP 모듈은 전체 MLP 하드웨어의 최상위 블록으로서, 입력 데이터의 선택부터 연산 제어, 결과 비교 및 출력까지의 전체 추론 과정을 일괄적으로 관리한다. 입력부에서는 FPGA 내부 또는 외부로부터 데이터를 받아서 Hidden Layer로 전달 하는데, 외부 입력은 PC로부터 UART 통신을 통해 전달되고, Uart_Rx 모듈에서 수신된 데이터는 Rx_frame_input 블록을 통해 64 비트 단위의 입력 데이터로 변환하고, 내부 입력은 LFSR(Random Generator)에 의해 자동 생성된 64 비트 랜덤 데이터로 구성된다. 이 두 입력은 스위치(SW0)에 의해 제어되며, SW0 이 0 일 때는 내부 랜덤 입력, 1 일 때는 외부 UART 입력이 선택된다. 선택된 데이터는 MUX 블록을 통해 Hidden Layer로 전달된다.

제어부는 FSM_controller를 중심으로 구성되어 있으며, 전체 연산 순서를 관리한다. BTN0은 btn_onepulse 모듈을 통해 디바운싱을 하여서, 단일 클록 폭의 시작 신호(start_pulse_btn)로 변환되고, 이 신호가 FSM_controller에 입력되어 연산이 시작된다. FSM_controller는 hid_start 신호를 통해 Hidden Layer 연산을 시작시키고, Hidden Layer에서 hid_done 신호가 발생되면 out_start 신호를 발생시켜 Output Layer에서 연산을 시작하게 한다. Output Layer의 연산이 완료되면 out_done 신호가 FSM_controller로 전달되어 전체 추론이 종료된다. 그리고 모든 신호가 클록(sysclk)에 동기화 되어 동작하며, BTN1(reset) 신호가 입력되면 모든 내부 상태가 초기화되어 시스템은 IDLE 상태로 돌아온다.

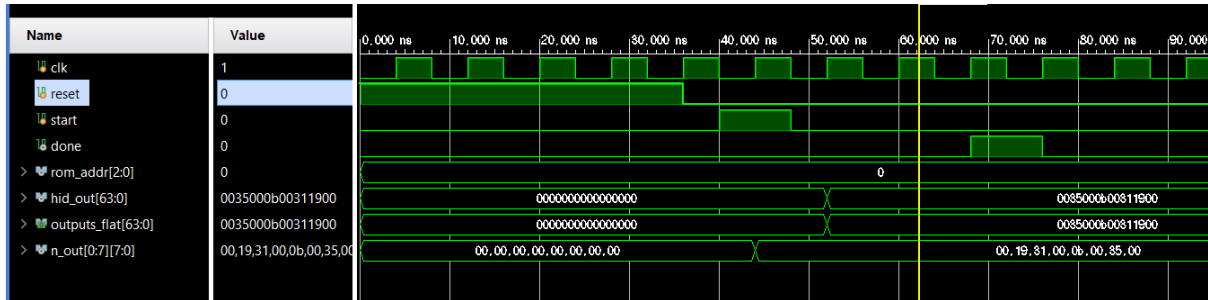
연산부는 Hidden Layer와 Output Layer로 구성되어 있고, 각 Layer는 독립적인 Weight/Bias ROM을 사용한다. Hidden Layer는 mux와 스위치를 통해서 정해진 입력 데이터(in_64[63:0])와 ROM에서 읽어온 가중치 및 바이어스를 이용해 8개의 은닉 뉴런 출력을 계산하고, 이 결과(Q[63:0])를 Output Layer로 전달한다. Output Layer는 Hidden Layer의 출력을 입력으로 받아 Weight/Bias ROM에서 해당 가중치와 바이어스를 읽어 연산을 시작하고 2개의 출력 뉴런을 계산한다. 두 계층 모두 파이프라인 구조(neuron_pipelined)를 적용하여 MAC, Shift, Bias, ReLU 연산이 순차적으로 진행되며, 각 단계의 결과는 Output_register에 저장된다.

Output Layer 의 출력(out_scores[15:0])은 Compare Latch 블록으로 전달되어 두 클래스의 뉴런 출력(y0, y1)을 비교한다. Compare Latch 는 더 큰 값을 갖는 출력을 최종 분류 결과로 선택하며, 그에 따라서 bread board 연결된 외부 LED 가 점등된다. class 0 이 선택될 경우 ext_led_class0 이 점등되고, class 1 이 선택될 경우 ext_led_class1 이 점등된다. 그리고 모든 연산이 완료되었음을 나타내는 ext_led_done 신호가 함께 출력되는 것을 실험을 통해 알 수 있었다. 동시에 비교 결과는 Uart_TX 모듈을 통해 PC 로 전송되며, 이를 통해 외부에서도 FPGA 의 추론 결과를 실시간으로 확인할 수 있는 것을 알 수 있다.

Top MLP는 전체 추론 과정을 하나로 통합한 상위 제어 구조로, mux와 스위치를 통한 입력 선택부터 연산 제어, 결과 비교와 출력까지의 모든 과정을 하나의 체계적인 경로로 연결하였다. FSM 기반 제어 구조를 통해 Hidden Layer 와 Output Layer 를 순차적으로 동작 시키며, 각 단계의 연산이 완료될 때마다 제어 신호를 생성하여서 다음 단계로 진행하게 한다. 입력은 스위치(SW0)에 따라 내부 랜덤 입력(LFSR)과 외부 UART 입력 중 하나가 선택되며, 이를 통해서 송신 검증 모드와 실시간 데이터 기반 추론 모드로 제어, 전환이 가능하다.

3. 설계 회로 검증

1) Hidden layer



- 위 그림은 병렬 구조로 동작하는 은닉층(hidden_layer_par) 모듈의 실제 연산 파형이다. start 신호가 1 클럭 pulse 하면 8 개의 뉴런이 동시에 연산을 시작하며 모든 입력 벡터 요소와 각 뉴런의 고정 가중치 바이어스가 병렬로 곱셈 누산 파이프라인을 거친다

순차형 구조와 달리 rom_addr 신호는 고정된 상태(0 또는 미사용)로 유지되며 모든 뉴런의 가중치가 내부 상수로 연결되어 있기 때문에 주소를 순차적으로 증가시키는 동작이 존재하지 않는다

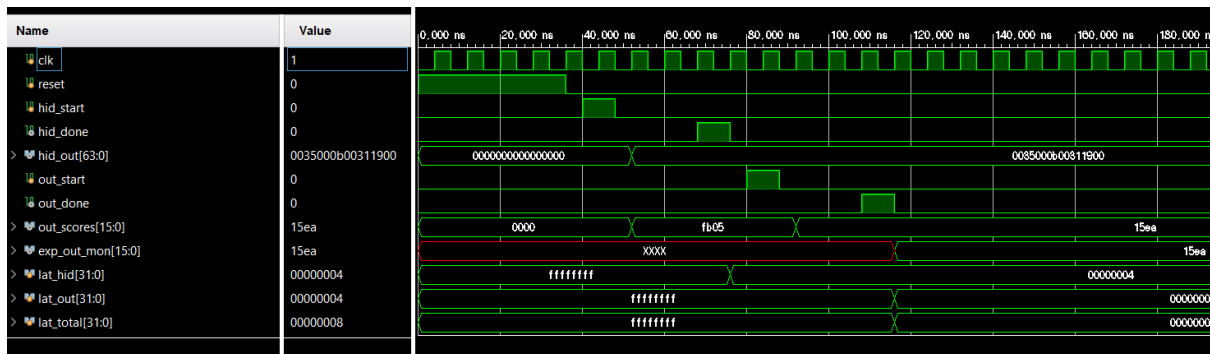
파형에서 start 가 발생한 후 약 28ns 만에 done 신호가 1 로 상승하였다 이는 125MHz 기준으로 약 3.5 클럭에 해당하며 각 뉴런이 동일한 클럭 경로 내에서 파이프라인 단계를 공유하기 때문인것으로 예상할 수 있다. 모든 뉴런이 3 단계(MAC 파이프라인 2 단계와 출력 레지스터 1 단계)를 거친 뒤 동시에 출력을 완료하므로 지연은 뉴런 수와 무관하게 일정하게 유지된다는 것을 알 수 있다.

Hidden layer 의 출력 신호인 hid_out 은 [n7 n6 n5 n4 n3 n2 n1 n0] 형태로 묶여 있어 waveform 에서는 MSB 가 n7 LSB 가 n0 으로 표시된다. 결과적으로 waveform 을 보았을 때는 실제 hid_out 에 대해서 출력 결과가 뒤집혀 보이지만 실제 각 뉴런의 결과는 논리적으로 올바른 순서(n0~n7)에 대응된다.

테스트벤치에서는 expect_neuron64() 함수를 통해 동일한 입력, 가중치, 바이어스, SHIFT 조건에서 이론적 출력을 계산하고 hidden layer 의 최종 출력 값인 hid_out 과 비교하였다. 그 결과 테스트 벤치에서 생성한 이론값과 실제 회로를 통해 연산된 결과가 같다는 메시지가 tcl console 출력되게 하였으며 이를 통해서 병렬 구조에서도 곱셈 누산 SHIFT ReLU 연산이 정확히 수행되고 각 뉴런의 연산 결과가 이론 값과 완전히 일치한다는 것을 알 수 있다.

결론적으로 병렬화된 hidden layer 는 주소 순차 접근 없이 모든 뉴런이 동시에 연산을 수행하며 약 28ns 의 짧은 지연 내에 정확한 결과를 출력한다. 그리고 Vivado 파형에서 보이는 출력 순서의 반전은 비트 결합 순서의 차이에 의한 시각적 현상이며 실제 하드웨어를 통한 결과는 이론 값과 일치함을 확인하였다

2) Output layer



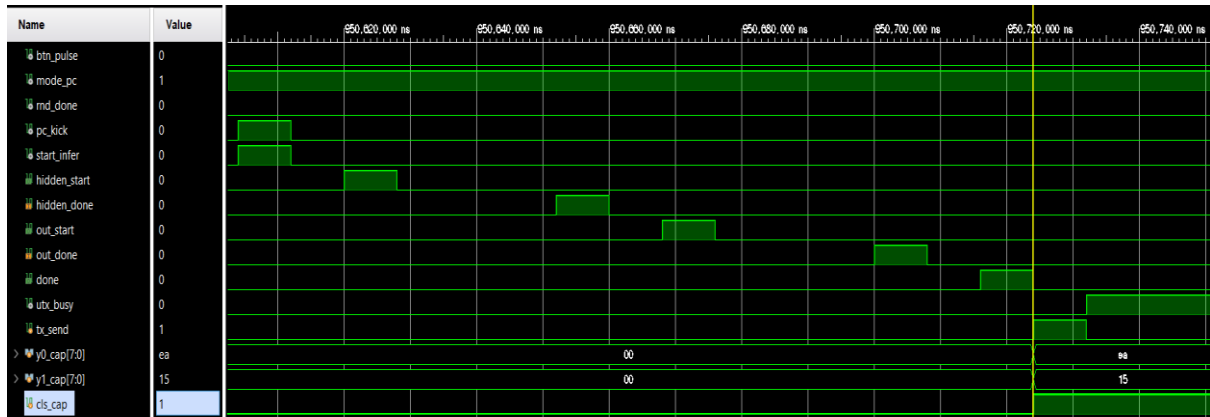
- 위 그림은 은닉층(hidden layer)과 출력층(output layer)이 병렬 구조로 연결된 MLP 최종 연산 블록의 동작 파형이다. 테스트벤치는 hid_start 신호를 1 클럭 pulse 하면 hidden layer 에서 연산을 시작하고, hid_done 신호가 1 이 되는 시점에서 4ns 뒤에 출력층의 out_start 신호를 발생시켜 두 모듈의 순차적 연계를 검증하였다.

파형 상에서는 hid_start-hid_done 사이가 약 28 ns, out_start-out_done 사이가 약 28 ns 로 나타나지만, 이는 start 신호가 하강 엣지에서 전달되고 실제 연산이 다음 상승 엣지부터 시작되기 때문이다. 따라서 실제 연산 지연은 파형 상 보이는 시간보다 반 클럭(4 ns) 더 길며, 내부 클럭 기준으로는 정확히 4 클럭임을 확인할 수 있다. 이를 통해 결과적으로 hidden layer 연산(hid_start → hid_done) 구간은 약 4 클럭(≈32 ns), output layer 연산(out_start → out_done) 구간은 4 클럭(≈32 ns), 전체 MLP 추론(hid_start → out_done)은 총 8 클럭(≈64 ns)으로 측정되었다. Hidden layer 와 Output layer 각각이 파이프라인 구조로 구성되어 있으며, 한번의 연산이 3~4 클럭 이내에 완료된다는 설계 목표와 거의 일치한다는 것을 알 수 있다.

출력 결과 비교를 위해 테스트 벤치 내에서 pipeline 에 있는 연산과 동일한 과정을 거치는 함수를 생성하여서 이론적 출력(exp_out)을 계산하였고, 하드웨어 출력(out_scores)과 직접 비교하였다. 두 결과는 동일한 입력에 대해서 각각 $y_0 = -22$, $y_1 = 21$ 로 완전히 동일한 출력을 나타내었으며, Vivado 안에 있는 Tcl Console 을 이용하여서 " output layer match" 메시지가 출력되어 병렬 구조에서도 연산 정확성이 유지됨을 확인하였다.

결과적으로, 본 파형은 병렬 구조의 Output Layer 가 정상적으로 은닉층 출력(hid_out)을 입력받아 2 개의 뉴런 연산을 동시에 수행하며, 파이프라인 단계에서 오차 없이 done 신호와 함께 최종 결과를 출력함을 보여준다. 또한, 총 8 클럭(≈64 ns)의 지연은 단일 CPU 연산 대비 수십 배 이상 빠른 하드웨어 연산 속도를 달성했음을 의미한다.

3) top_mlp



- 위 그림은 top_mlp_simple_dual 모듈의 전체 FSM 동작을 보여주는 파형으로, 은닉층과 출력층의 병렬 파이프라인 연산 및 최종 UART 송신까지의 절차가 올바르게 수행됨을 확인할 수 있었다. 본 실험에서는 입력 데이터를 $x = [0, 0, 0, 0, 0, 0, 0, 0]$ 으로 고정하여, 동일 입력에 대해 FPGA 내부 추론 결과가 이론적으로 예측된 값과 일치하는지를 검증하였다.

waveform을 보았을 때 950.604 μ s 시점에 pc로부터 데이터 수신 완료 신호인 pc_kick 신호가 1 클럭 동안 발생하여 입력 데이터 수신이 완료되고, 바로 start_infer 신호가 상승하면서 전체 연산이 시작 된다. 그 다음 단계로 950.620 μ s 에서 hidden_start 신호가 펄스 형태로 1 클럭 발생하여 hidden layer 연산이 시작된다. Hidden layer 는 8 개의 뉴런을 병렬로 동작 하도록 설계되어 있으며, 950.652 μ s 에서 hidden_done 이 상승함에 따라 hidden layer 의 연산이 완료된다. 그리고 1 클럭 뒤에 out_start 상승하며 output layer 연산이 시작되고, 약 32 ns 후인 950.700 μ s 에서 out_done 이 상승하면서 output layer 연산이 종료된다. 마지막으로 상위 FSM 이 이를 인식하고 2 클럭 뒤인 950.716 μ s 에서 done 신호를 1 클럭 동안 출력함으로써 전체 추론 과정이 종료된다

구간	신호 변화 시점(μ s)	시간차이(ns)	소요 클럭 수
start_infer → hidden_start	950.604→950.620	16ns	2 clk
hidden_start→ hidden_done	950.620→950.652	32 ns	4 clk
hidden_done → out_start	950.652→950.668	16 ns	2 clk
out_start → out_done	950.668→950.700	32 ns	4 clk
out_done → done	950.700→950.716	16 ns	2 clk
전체 소요시간	950.604→950.716	112 ns	14clk

이론적으로 은닉층 4 클럭, 출력층 4 클럭, 제어 및 데이터 동기화 과정 약 6 클럭이 요구되며, 실제 관측된 총 14 클럭의 지연은 실제 파이프라인 동작이 정확히 구현되었음을 보여준다.

done 신호가 상승하는 950.716 μ s 시점에서 결과 저장 레지스터(y0_cap, y1_cap, cls_cap)가 동시에 갱신되었으며, 관측된 결과는 다음과 같다.

신호명	HEX 값	10 진수 값	의미
y0_cap	EA	-22	Class0 의 뉴런 출력
Y1_cap	15	21	Class1 의 뉴런 출력
cls_cap	1	Class 1 선택	

이는 출력층의 두 결과 중 y1 이 더 큰 값을 가지므로 Class 1 을 선택한 상황이며, 연산 결과와 논리적인 분류 결과가 정상적으로 작동하는 것을 알 수 있다.

마지막으로 done 이 비활성화된 직후 950.724 μ s 에서 tx_send 가 1 클럭 동안 펄스로 발생하였고, 950.732 μ s 에서 utx_busy 가 1 로 상승하며 UART 송신이 개시되었다. 이는 FSM 이 추론 완료 직후 정상적으로 UART 전송을 수행하고, 작동하고 있음을 확인할 수 있는 부분이다.

요약하면 본 파형은 입력 [0 0 0 0 0 0 0]에 대해 hidden layer 와 output layer 가 각각 뉴런에 대해서 정상적으로 병렬 계산을 수행하고, FSM 제어 흐름과 결과 저장, 클래스 판정, 그리고 UART 송신까지 모든 단계가 설계된 타이밍에 따라 정확히 진행됨을 보여준다. 총 지연시간이 이론값(112 ns)과 일치하며 출력과 클래스 결과가 논리적으로 알맞게 나오므로 정상 수행을 하고 있는 것을 알 수 있다.

4) 실제 동작 구현

```
C:\python_prj>python a.py
Ports: ['COM3', 'COM7', 'COM5', 'COM4']
Connected: COM3 @ 115200bps

SW0=0: 버튼 내부 난수 | SW0=1: Enter로 PC 난수 전송 | q=종료
```

Inputs (x0..x7)	y0	y1	Cls	SW
[-108, 40, 80, -95, 66, -124, 9, 19]	21	-22	0	0
[39, 79, -97, 63, 127, -1, -2, -4]	-58	57	1	0
[-8, -16, -31, -62, -123, 11, 23, 47]	-43	42	1	0
[94, -68, 120, -15, -29, -58, -115, 26]	-50	49	1	0
[52, 104, -48, -96, 64, -128, 1, 2]	-87	86	1	0
[4, 8, 17, 35, 71, -114, 28, 56]	-20	19	1	0
[113, -30, -60, -119, 18, 37, 75, -105]	-107	106	1	0
[46, 92, -72, 112, -32, -64, -127, 3]	-16	15	1	0
[6, 12, 25, 50, 100, -55, -110, 36]	-81	80	1	0
[73, -109, 38, 77, -101, 55, 110, -36]	40	-41	0	0
[-71, 114, -28, -56, -112, 32, 65, -126]	50	-51	0	0
[5, 10, 21, 43, 86, -83, 91, -74]	-78	77	1	0
[-99, 66, 54, 48, -13, 17, -84, -33]	-96	95	1	1
[-18, -51, 45, -43, -44, -115, 34, -39]	-50	49	1	1
[-80, -36, 92, 84, 109, 110, -13, 103]	-116	115	1	1
[80, -128, 41, 98, -68, 113, 51, 78]	-111	110	1	1
[46, 91, 87, -73, 35, 62, -91, -26]	-65	64	1	1
[85, 82, -66, -57, -28, -25, 40, -83]	-37	36	1	1
[25, 97, 113, -33, 21, 70, 32, -74]	-70	69	1	1
[25, -122, 84, 93, 73, 65, -57, -93]	58	-59	0	1
[-23, -76, 114, 23, -71, 13, -97, -109]	-36	35	1	1

- 위 그림은 cmd 에서 PC 와 FPGA 간의 실제 동작을 Python 시리얼 통신 프로그램을 통해 구현 한 것이다. 프로그램은 115200 bps 의 속도로 COM3 포트에 연결되어 있으며, 모드 선택 스위치 SW0 값에 따라 내부 난수(SW0=0) 또는 PC 입력 난수(SW0=1)를 기반으로 추론을 수행한다.

입력 데이터는 8 차원 벡터(x0~x7, signed 8-bit)로 구성되어 있으며, 각 행은 FPGA 로 전송된 입력 값과 그에 대한 추론 결과(y0, y1), 분류 결과(Cls), 현재 모드(SW)를 포함한다. SW=0 구간에서는 FPGA 내부의 LFSR(Random Generator)에 의해 자동으로 생성된 입력 값이 사용되고, SW=1 구간에서는 Python 프로그램이 생성한 난수를 UART 통신 프레임의 구분용 헤더와 테일 바이트를 포함해 (A6 | x[0]..x[7] | 5A) 형태로 FPGA 로 송신하였다.

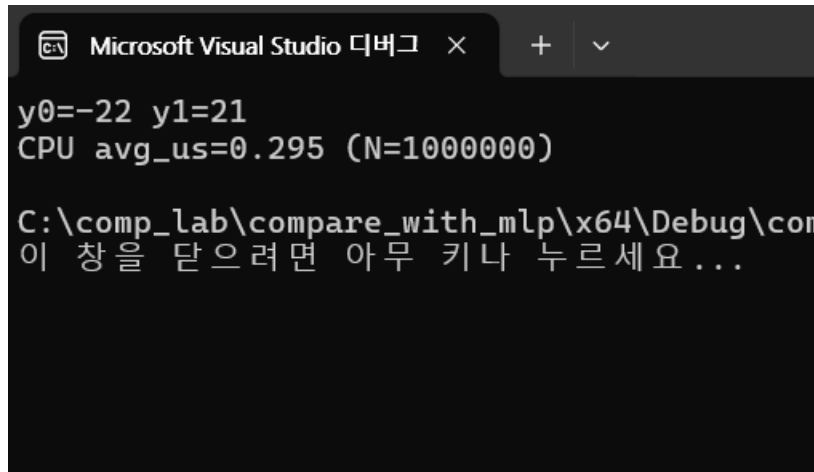
FPGA 내부에서는 앞서 확인했듯 컨트롤러 FSM 이 순차적으로 hidden_start → hidden_done → out_start → out_done → done 흐름으로 동작하며, done 신호가 활성화될 때 출력값이 송신 프레임(A7 | x[0]..x[7] | y0 | y1 | cls | sw | 5A)으로 PC 로 전송된다. Python 측에서는 해당 프레임을 디코딩하여 입력 및 출력 결과를 표 형태로 시각화한다.

콘솔에 출력된 결과를 보면, 각 입력 벡터에 대해 y0, y1 의 부호와 크기가 다르게 나타나며, Cls 는 y1 > y0 일 때 1, 그렇지 않을 때 0 으로 표시된다. 각 다른 입력에 따라 출력 결과가 항상 다르게 재현되어 하드웨어의 동작이 확인되었다. 또한 SW=0 에서 SW=1 로 전환 시에도 오류 없이 UART 프레임이 정상적으로 송수신된다는 것을 알 수 있다.

5) FPGA 와 CPU 의 평균 지연과 처리율 차이

```
C:\python_prj>python b.py
x=[0, 0, 0, 0, 0, 0, 0, 0]
y0=-22 y1=21 cls=1 sw=1
cyc_inf=14 latency=0.112 us throughput=8928571.4 inf/s
```

(a) 실제 실험 회로의 평균 지연과 처리율



```
Microsoft Visual Studio 디버그 × + v
y0=-22 y1=21
CPU avg_us=0.295 (N=1000000)

C:\comp_lab\compare_with_mlp\x64\Debug\com
이 창을 닫으려면 아무 키나 누르세요 ...
```

(b) CPU 의 평균 지연과 처리율

- 각각 동일한 입력 데이터에 대해 CPU 와 FPGA 에서 추론 결과와 지연율 그리고 처리율에 대해서 비교하였다. 두 플랫폼 모두 동일한 가중치 및 활성화 함수(ReLU)를 사용하였다.

플랫폼	y_0	y_1	Class	평균지연(μ s)	처리율(추론/s)
CPU	-22	21	1	0.311	3.21M
FPGA	-22	21	1	0.112	8.93M

(C) 두 결과를 비교한 표

동일한 입력에 대해서 두 결과 차이를 표를 통하여 확인해 보았을 때, FPGA 와 CPU 의 출력 값 및 분류 결과(Class)가 동일하게 나타나는 것을 확인 가능하였다. 이는 하드웨어로 구현된 고정 소수점 MLP 연산이 소프트웨어 모델과 논리적으로 동일한 결과를 낸다는 것을 알 수 있었다.

CPU 는 소프트웨어적인 계산을 통해 8×8 의 은닉층 MAC 연산과 8×2 의 출력층 연산을 순차적으로 수행한다. 각 뉴런의 계산은 메모리 접근, 곱셈, 덧셈, 활성화 연산이 모두 CPU 의 명령 파이프라인을 거치며 진행되기 때문에, 연산량이 늘어날수록 지연이 점점 누적되는 구조다. 반면 FPGA 에서는 각 뉴런의 연산을 병렬 회로로 독립적으로 배치하였다. 그 결과 hidden layer 의 8 개 뉴런과 output layer 의 2 개 뉴런이 각각 동일 클럭 사이클 내에서 동시에 계산되므로, 전체 MLP 추론이 14 클럭(0.112μ s) 만에 완료된다.

FPGA 구조는 연산 순서가 아니라 연산 단위별 하드웨어 병렬성을 통해 전체적인 지연시간을 줄인다. 각 뉴런 내부의 MAC 연산은 파이프라인 레지스터를 포함한 전용 연산 회로로 구성되어 있어, 소프트웨어에서 반복적으로 실행되는 곱셈과 덧셈을 하드웨어 레벨에서는 동시에 수행한다. 또한 데이터 이동이 메모리 버스를 거치지 않고 로컬 신호선으로 연결되므로, CPU 에서 발생하는 명령어 디코딩과 메모리 접근 오버헤드가 존재하지 않는다. 이러한 구조적 차이로 인해 FPGA 구현은 CPU 대비 약 2.78 배 빠른 처리 속도($0.311 \mu s \rightarrow 0.112 \mu s$) 를 보였으며, 처리율(frequency/clock) 기준으로는 $3.21 \text{ M inf/s(cpu)} \rightarrow 8.93 \text{ M inf/s(fpga)}$ 로 향상되었다.

결과적으로, FPGA 기반 추론 구조는 동일한 연산 정확도를 유지하면서도 뉴런 연산의 병렬화, 파이프라인 구조를 통해 연산 효율을 극대화하였다. 이는 CPU 의 순차적 명령 실행 구조와 대비되는 FPGA 의 하드웨어 병렬 처리 특성을 알 수 있는 결과이며, 실시간 추론을 요구하는 임베디드 환경에서 FPGA 가속이 유리하다는 것을 보여준다.

4. 결론

가. 최종 결론

- 본 프로젝트에서는 FPGA 상에서 $8 \times 8-2$ 구조의 MLP 추론 하드웨어를 구현하여, 입력 데이터를 받고 이를 바탕으로 연산 제어, 결과 출력 및 UART 통신까지 전 과정을 수행하였다. 전체 동작은 FSM 기반 순차 제어 흐름으로 진행되며, 각 단계의 지연과 신호 타이밍을 파형 분석을 통해 검증하였다.

실험은 두 가지 모드로 진행되었다. SW0=0 에서는 FPGA 내부 LFSR 모듈이 자동으로 8 바이트 입력 데이터를 생성하였고, SW0=1 에서는 Python 프로그램을 통해 PC 에서 입력 데이터를 UART 프레임(A6 | x0~x7 | 5A) 형태로 전송하였다. FPGA 내부의 rx_frame_to_inputs 블록은 PC 에서 전송된 데이터를 한 바이트씩 받아 정해진 형식에 맞게 저장한 뒤, 8 개의 입력 값을 하나의 64 비트 데이터로 변환한다. 모든 데이터가 정상적으로 수신되면 입력이 완성되었다는 신호로 pc_kick 펄스를 1 클럭 동안 발생시켜, 이후 연산이 자동으로 시작되도록 한다.

시뮬레이션 파형에 보았을 때는 , 950.604 μ s 시점에 pc_kick 이 발생하면서 전체 연산이 개시되고, 950.620 μ s 에 hidden_start, 950.652 μ s 에 hidden_done, 950.668 μ s 에 out_start, 950.700 μ s 에 out_done, 950.716 μ s 에 done 이 순차적으로 활성화되었다. 각 단계별 평균 지연은 2~4 클럭 수준으로, 전체 추론이 14 클럭(112 ns) 내에 완료됨이 파형적으로도, 이론적으로 일치하다는 것을 확인 가능하였다.

Python 통신 프로그램에서 수신된 프레임(A7 | x0~x7 | y0 | y1 | cls | sw | 5A)을 디코딩한 결과, FPGA 내부에서 계산된 출력과 동일한 값이 확인되었으며, 입력에 따라 분류 결과가 즉시 갱신되는 것을 볼 수 있었다. SW0 를 0→1 로 전환하더라도 송수신 에러 없이 연속적인 추론이 가능한 것을 pc 를 통해 확인했다.

또한 동일한 가중치와 입력 데이터로 Python(CPU) 기반 소프트웨어 모델을 실행한 결과, 연산 결과(y0=-22, y1=21, cls=1)가 FPGA 결과와 정확히 일치하였다. 평균 지연은 CPU 에서 0.311 μ s, FPGA 에서 0.112 μ s 로 측정되어 FPGA 구현이 약 2.78 배 빠른 처리 속도를 보였다. 처리율 기준으로는 CPU 3.21 M inf/s → FPGA 8.93 M inf/s 로 향상되었다. 이러한 차이는 FPGA 가 CPU 와 달리 각 뉴런의 MAC 연산을 병렬 파이프라인 구조로 수행하기 때문이고, 명령어를 해석하고 메모리 접근 오버헤드가 없는 구조적 또한, 처리속도가 효율적 임에 근거가 된다.

결과적으로 고정소수점 기반 MLP 연산의 정확성과 CPU 대비 높은 연산 효율과 짧은 지연성이 있는 것을 확인하였다.

이 설계는 추론만 수행하는 저전력 임베디드 AI 가속기 구조로서, 외부 DRAM 접근이 필요 없는 구조를 통해 전력 소모를 최소화할 수 있다. 또한 Weight/Bias ROM 교체만으로 다른 모델로의 이식이 가능하므로, 향후 CNN 또는 TinyML 모델 가속기로 확장할 수 있는 발전 가능성이 있다

나. 한계

- 본 프로젝트는 FPGA 기반 MLP 추론 시스템을 구현하여 하드웨어 수준에서 실시간 분류가 가능함을 확인하였지만, 몇 가지 기술적, 구조적 한계가 존재한다.

첫째, 학습(Training) 과정이 포함되지 않았다는 점이다. 현재 구조는 이미 고정된 weight와 bias를 ROM에 저장해 사용하는 추론 전용 형태로 되어 있어, 새로운 데이터에 대해 스스로 가중치를 갱신 해야 한다는 단점이 있다.

둘째, 가중치 저장 방식의 유연성이 부족하다. ROM 구조로 고정되어 있어, 다른 모델이나 입력에 이 바뀔 때마다 비트스트림을 새로 생성해야 한다.

셋째, 정밀도 및 연산 확장성의 한계가 있다. 현재 시스템은 8 비트 고정 소수점 기반이기 때문에 연산 오차가 누적될 수 있다. 그래서 향후에는 16 비트 또는 가변 고정소수점 연산으로 확장해 정확도를 높일 필요가 있다고 생각한다.

마지막으로, 전력 측정 및 효율 검증 단계가 충분히 이루어지지 않았다. 이론상 CPU 보다 저전력일 것으로 예상하고 있지만, 실제 보드에서의 소비전력 측정 및 에너지당 추론량 분석은 이루어지지 않았다. 향후에는 전력 분석 도구나 외부 장비와 연동하여 전력을 비교하고, 효율 개선 방향을 수치적으로 제시할 필요가 있다고 생각한다.