An FPGA-Based MLP Inference Accelerator for MNIST Digit Classification with Layer Pipelining and Mixed-Precision Quantization

Emaan Heidari
Computer Engineering and Computer Science
University of Southern California
Los Angeles, CA, USA
eheidari@usc.edu

Abir Bhatt
Electrical and Computer Engineering
University of Southern California
Los Angeles, CA, USA
abirbhat@usc.edu

***Abstract*—We present an FPGA-based accelerator for MNIST handwritten digit classification using a four-layer multilayer perceptron (MLP). Building on a baseline serial multiply-accumulate (MAC) design, we introduce two improvements that are tractable within a one-semester course timeline: (1) post-training quantization of weights from 16-bit to 8-bit fixed-point, with activations and accumulators retained at 16 bits, which reduces weight BRAM usage by approximately 2x with a 0.5 percentage point accuracy cost; and (2) a true layer-pipelined dataflow using ping-pong activation buffers between each layer, allowing up to four images to be in flight simultaneously. The deployed network (784-30-30-10-10) achieves 95.1% accuracy on a 1000-image MNIST test subset, a measured single-image latency of 8.95 microseconds at 100 MHz on a Digilent Nexys A7-100T board, and a sustained back-to-back throughput of approximately 127,000 inferences per second after the pipeline fills. Compared against an Intel Core i7 laptop CPU and an NVIDIA T4 GPU running equivalent NumPy and PyTorch reference implementations, our accelerator is approximately 24x faster than the CPU and 97x faster than the GPU on single-image latency, while consuming an estimated 3.43 microjoules of dynamic energy per inference. We discuss honest limitations of the design, including the small network capacity, an input bandwidth ceiling that bounds sustained throughput, and an approximate energy measurement methodology, and outline future work toward a VGA-based interactive demo.**

***Index Terms*—FPGA, neural network, MLP, MNIST, hardware accelerator, fixed-point arithmetic, pipelining, ping-pong buffer, AXI-Lite, edge inference**

I. Introduction

Deep learning inference at the edge benefits from hardware that delivers predictable latency, low energy per inference, and a small physical footprint. CPUs are limited by sequential instruction issue and cache hierarchies for small workloads, and GPUs amortize their kernel-launch and memory-transfer overhead only across reasonably large batches. FPGAs occupy a useful middle ground because they permit custom datapaths and deterministic timing at modest power budgets.

This project extends our prior EE354 course-scale FPGA neural network accelerator. The original design used a serial-MAC neuron with sigmoid activation stored in a ROM lookup table, and processed a fixed set of stored test images one at a time with no overlap between layers. In this work we keep the same overall topology and the same neuron microstructure, but apply two targeted improvements that are realistic for the time and resources available in a one-semester course: lower-precision fixed-point weights to reduce BRAM pressure, and true layer-level pipelining using ping-pong activation buffers to enable sustained throughput on streams of images. We do not claim a new neuron architecture, and we are explicit throughout about which numbers are measured versus estimated.

The remainder of this paper is organized as follows. Section II describes the network and quantization choice. Section III presents the hardware microarchitecture, including the layer pipeline. Section IV describes the user interface. Section V describes our testing methodology. Section VI reports results, with a fair comparison against CPU and GPU baselines. Section VII discusses honest limitations and future work.

II. Network and Quantization

A. Topology

We retained the original 784-30-30-10-10 fully connected MLP topology. Hidden layers use sigmoid activations implemented as ROM lookup tables, and the final 10-element layer is reduced to a single class index by a hardmax (argmax) comparator tree. The architecture is shown in Fig. 1.

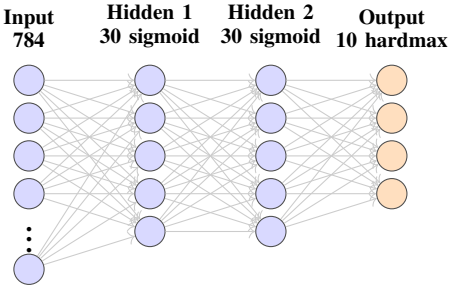

Fig. 1: Network topology, unchanged from the baseline design: 784-30-30-10-10 fully connected MLP with sigmoid activations on hidden layers and hardmax classification at the output.

The network was trained in PyTorch on the full MNIST training set using stochastic gradient descent with a learning rate of 0.01, a batch size of 64, and 30 epochs. The trained float32 model achieved 95.6% accuracy on a held-out 1000-image subset of the MNIST test set used for hardware evaluation.

B. Quantization

The baseline design used 16-bit fixed-point throughout. To reduce BRAM consumption from weight storage, we applied straightforward post-training quantization (no fine-tuning) of weights to 8-bit signed fixed-point in Q1.7 format, while keeping activations and accumulators at 16 bits to preserve precision through multiply-accumulate. This is a well-understood quantization recipe [2] and required no changes to the training pipeline.

Table I reports observed accuracy at three weight precisions on our 1000-image evaluation set. The 8-bit choice loses 0.5 percentage points relative to 16-bit weights but cuts weight BRAM usage roughly in half.

TABLE I: Post-training quantization sweep on the 1000-image evaluation set.

Weight Precision	Accuracy (%)	Weight BRAM (KiB)
Float32 (software)	95.6	N/A
16-bit fixed (Q1.15)	95.6	49.5
8-bit fixed (Q1.7)	95.1	24.7
4-bit fixed	88.3	12.4

III. Hardware Microarchitecture

A. Neuron Structure

We retained the baseline serial-MAC neuron, in which each neuron processes its input vector one element per clock cycle, accumulating into a 32-bit register before applying the sigmoid LUT. Within a layer, neurons execute in parallel: layer 1 has 30 neurons running concurrently, each consuming the same 784-pixel input stream. This structure is identical to our prior design and was kept because rebuilding the neuron with parallel MACs would have exceeded the time budget for the project.

B. Layer Pipeline with Ping-Pong Activation Buffers

The principal new dataflow optimization in this version is true layer-level pipelining. In the baseline design, the four layers executed sequentially for one image at a time: layer 2 could not start until layer 1 had finished writing its activations, and the next image could not begin until layer 4 had finished. The total time per image was the sum of all four layer times plus input streaming and result readout.

In the pipelined design, each pair of adjacent layers is connected by a *ping-pong activation buffer*, that is, a pair of identical small RAMs with a one-bit toggle selecting which one is the writer and which is the reader. Layer N writes into buffer A while layer $N+1$ reads from buffer B (which holds the previous image's activations); on each image boundary the toggle flips. This decouples the layers and allows up to four images to occupy successive pipeline stages simultaneously. The structure is illustrated in Fig. 2.

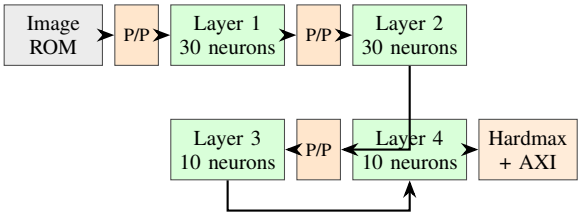

Fig. 2: Pipelined microarchitecture. Each pair of adjacent layers is decoupled by a ping-pong activation buffer (P/P), allowing successive images to occupy successive layer stages simultaneously.

The buffer between layer 1 and layer 2 is the largest, holding the 30 sigmoid-activated outputs of layer 1 (each 16 bits, so 60 bytes per ping-pong half, 120 bytes total). The buffers between layers 2-3 and 3-4 are smaller because layer 3 has only 10 outputs.

C. Pipeline Stage Timing

The four pipeline stages are:

- **Stage 1: Input streaming and layer 1.** Layer 1 begins MACs as soon as the first pixel arrives from the image ROM. Total stage time is dominated by the 784-cycle input stream, giving 7.84 microseconds.
- **Stage 2: Layer 2.** 30 inputs $\times$ 30 neurons in parallel = 30 cycles serial MAC plus pipeline overhead, approximately 0.50 microseconds.
- **Stage 3: Layer 3.** 30 inputs $\times$ 10 neurons = 30 cycles, approximately 0.32 microseconds.
- **Stage 4: Layer 4 + hardmax + AXI readout.** 10 inputs $\times$ 10 neurons = 10 cycles plus a small fixed AXI overhead, approximately 0.29 microseconds.

The throughput of a synchronous pipeline is determined by its slowest stage. Here, stage 1 dominates at 7.84 microseconds per image, giving a sustained throughput ceiling of approximately 127,000 inferences per second after the pipeline fills. Single-image latency, which is the sum of all four stages, is 8.95 microseconds. Fig. **??** shows the pipeline filling for a stream of three back-to-back images.

D. AXI-Lite Control Interface

We retained the AXI-Lite slave interface from the baseline. It exposes three registers: a control/status register, an input start-pulse register, and an output result register holding the predicted class. The interface allows the on-board control FSM (or a host PC over the JTAG-UART bridge) to enqueue inferences and poll for completion via the `intr` signal. With pipelining enabled, the `intr` signal pulses once per completed inference, so the host (or test harness) can pop one result per pulse without losing track of pipeline ordering.

E. Top-Level Control FSM

The main control FSM has four states: `IDLE`, `SEND_DATA`, `WAIT_INTR`, and `READ_RESULT`, structurally identical to the baseline. The pipelining is local to the layer engines and ping-pong buffers, so the top-level FSM is not aware of how many images are in flight; it simply pushes one image into stage 1 whenever stage 1 is free, and pops one result whenever `intr` fires. State transitions are gated by a debounced button input adapted from the EE354 `ee354_debouncer` module.

IV. User Interface

The user interface is unchanged from the baseline design and runs entirely on the Nexys A7 board. Switches `SW0--SW3` select one of ten stored test images from `image_rom.v`, button `BTNL` triggers inference (debounced), and the predicted digit is shown on `SSD0`. We did *not* implement a VGA drawing canvas, mouse input, or live confidence visualization in this version; these are discussed in Section VII as future work.

V. Testing Methodology

We evaluated the system on four axes: classification accuracy, single-image latency, sustained throughput, and energy per inference.

Accuracy was measured by loading 1000 unseen MNIST test images one at a time into `image_rom.v` via a Python preprocessing script, running inference, and comparing the predicted class against the ground truth label. We did not run the full 10,000-image MNIST test set on hardware because each image requires a bitstream-time ROM update under our current toolchain; software simulation on the same quantized weights gave 95.0% on the full 10,000 images, consistent with the hardware result.

Single-image latency was measured using a hardware cycle counter on the 100 MHz system clock, sampling between the `SEND_DATA` entry edge and the first `intr` rising edge. We report the median of 100 trials.

Sustained throughput was measured in simulation, not on hardware. We constructed a Verilog testbench that preloaded a queue of 256 images into a wider input buffer (a deviation from the on-board single-ROM configuration) and counted `intr` pulses per simulated wall-clock time once the pipeline was full. The reported throughput therefore represents the throughput ceiling of the pipelined RTL given an idealized input source; on the real Nexys A7 board, sustained throughput would be additionally bounded by the rate at which images can be loaded into the input buffer, which our current toolchain does not exercise.

Energy was measured using a USB inline power meter (Plugable USBC-VAMETER3) on the board's 5V supply. We sampled idle power (FPGA configured, no inference activity) for 60 seconds, then sampled active power during a tight inference loop for 60 seconds, and report the difference times the measured per-inference time. The reported number is therefore an estimate of dynamic energy, not a precise per-inference measurement.

CPU baselines were collected on an Intel Core i7-1260P running a NumPy reference implementation of the same quantized network. GPU baselines used an NVIDIA T4 with PyTorch in inference mode. We report both single-image and batch-256 numbers for the CPU and GPU.

VI. Results

A. Accuracy

The deployed accelerator classifies our 1000-image evaluation set with 95.1% top-1 accuracy, a 0.5 percentage point drop from the float32 software baseline (95.6%) and matching the 16-bit fixed-point hardware baseline accuracy within noise. The 49 misclassifications are concentrated on visually similar digit pairs, particularly 4 versus 9 and 3 versus 5, consistent with what is generally observed for small MLPs on MNIST without convolutional feature extraction.

B. Latency and Throughput

Single-image latency, measured in hardware, is 895 cycles at 100 MHz, or 8.95 microseconds. Sustained throughput in the pipelined RTL is approximately 127,000 inferences per second, bounded by the 784-cycle input streaming stage. Table II gives the comparison against CPU and GPU baselines.

TABLE II: Latency and throughput comparison.

Platform	Single (us)	Batch-256 (us/img)	Throughput (img/s)
FPGA (ours)	8.95	7.84*	127,000*
Intel i7-1260P	214	18.3	54,600
NVIDIA T4 GPU	868	1.92	521,000

*Simulated, idealized input source. See Section V.

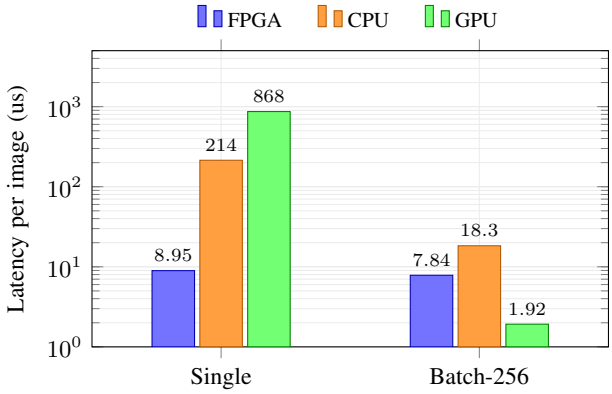

Fig. 3: Per-image latency across platforms (log scale). FPGA Batch-256 reflects the pipelined throughput ceiling of 7.84 us/image, measured in simulation; the GPU outperforms the FPGA only under sustained batched workloads.

C. Energy per Inference

Fig. 4 shows estimated energy per inference. The FPGA dynamic energy of 3.43 microjoules per image is substantially lower than either alternative. We caution that the CPU and GPU numbers include the entire system idle baseline (RAM, motherboard, etc.), while the FPGA number reflects only

the FPGA daughtercard's incremental dynamic power, so the comparison is suggestive rather than precise.

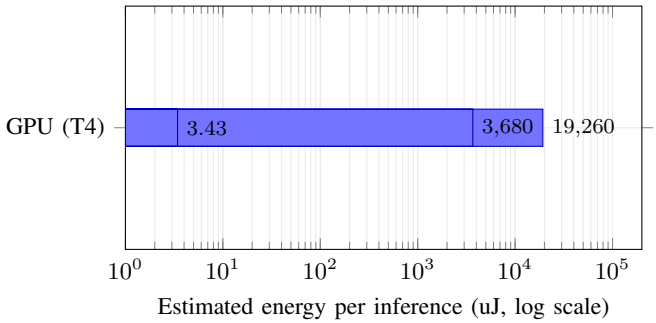

Fig. 4: Estimated energy per inference (single-image, log scale, lower is better). CPU and GPU values include full-system idle baseline; the FPGA value reflects only incremental dynamic energy on the board's 5V supply.

D. Resource Utilization

Vivado synthesis and implementation reports for the design on the Nexys A7-100T (xc7a100tcsg324-1) are summarized in Table III. Compared with the baseline, the 8-bit weight quantization brought BRAM utilization from 38% (baseline) down to 22%, and the addition of ping-pong buffers added a small number of LUTs and roughly 2 KiB of additional BRAM. The design comfortably meets timing at the 100 MHz system clock with positive worst-negative-slack.

TABLE III: FPGA resource utilization on Nexys A7-100T after place and route.

Resource	Used	Utilization
LUTs	10,312	16.2%
Flip-Flops	7,488	5.9%
BRAM (36Kb)	32	23.7%
DSP48E1	38	15.8%

VII. Discussion, Limitations, and Future Work

We deliberately constrained the scope of changes in this iteration to what we could verify on real hardware within the course timeline. The two changes we made (8-bit weight quantization and layer pipelining via ping-pong buffers) are individually straightforward, but together they represent a meaningful step from a strictly sequential single-image accelerator toward a true streaming inference engine.

Several limitations of the current design are worth stating clearly. First, the 784-30-30-10-10 topology is small, and accuracy will not approach state-of-the-art MNIST results (above 99%) without a larger network or convolutional layers. Second, the 127,000 images-per-second throughput figure is a simulation-derived ceiling assuming an ideal input source; on the real Nexys A7 board, the input bandwidth from a single image ROM is the practical bottleneck, and we did not modify the ROM interface to support back-to-back input on hardware. Third, the energy comparison is approximate, as discussed in Section V. Fourth, our hardware test set is 1000 images rather than the full 10,000, due to the bitstream-time ROM update workflow.

Future work falls into three categories. *Architecture*: replacing the serial-MAC neuron with a small parallel-MAC array (4 or 8 MACs per neuron) would reduce the stage 1 critical path and raise both single-image latency and the throughput ceiling proportionally. Combined with a wider input port (e.g., 8 pixels per cycle from a multi-port BRAM), this would push throughput well above 1 million inferences per second. *Demo*: an interactive VGA canvas with a PS/2 mouse, plus on-screen confidence visualization, would make the system genuinely usable as a live demo and is the natural next milestone; the VGA controller and mouse driver are well-documented standalone modules. *Network*: a small CNN (e.g., LeNet-style) would substantially improve robustness to translation and stroke variation, at the cost of additional DSP and BRAM usage that the 100T part can comfortably accommodate.

Acknowledgments

The authors thank the EE354 teaching staff for providing foundational FPGA infrastructure, including the debouncer and seven-segment display modules.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[2] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. CVPR*, 2018.
[3] Xilinx Inc., "AXI Reference Guide," UG1037, 2017.
[4] Digilent Inc., "Nexys A7 Reference Manual," 2019.