Sam Belliveau SRB343
Ezra Riess EZ495

November 6, 2025

ECE6775 Lab 4: Accelerating Binarized Neural Networks

1 Overview

We accelerated a Binarized Neural Network for handwritten digit recognition using HLS on a Zynq FPGA. By systematically optimizing the convolutional layers via HLS directives, we achieved a **211.7x** speedup over the software baseline while maintaining **90% accuracy** and consuming minimal resources. To do this we utilized aggressive loop unrolling and pipelining to exploit parallelism inherent in the FPGA architecture as well as a linebuffer to reduce resource utilization and allow for the aggressive pipelining while still fitting on chip.

2 HLS Optimization

Design	Clock (ns)	Cycles	BRAM %	FF %	LUT %
`Baseline`	7.854	1 379 512	5.36	0.85	5.76
`Reshape + Unroll`	8.658	533 240	5.36	0.81	6.29
$\boxed{\texttt{FPGA Final}}$	8.631	5474	11.79	5.42	65.16
`Best Performance`	8.631	4127	11.79	5.33	66.26

Table 1: HLS synthesis results across design iterations.

We optimized the BNN accelerator iteratively by targeting the most time-consuming kernels with HLS directives, prioritizing latency first and then resource balance to meet FPGA constraints. The `Baseline` design applies no HLS directives and, lacking any FPGA-oriented optimizations, actually ran slower on the FPGA than the ZedBoard's ARM CPU.

Conv Reshape & Unroll. Because convolution dominated the $\sim$1.38M-cycle baseline, we first applied array *reshape* and loop *unroll* along the input-channel dimension. This reduced the innermost loop to a single cycle and cut total latency to 533K cycles, with only a small increase in LUT usage.

Dense Optimizations. For the dense layers, we reordered loops to enable effective array reshaping across features, exposing parallelism and easing BRAM pressure. This yielded a clear speedup while keeping memory use in check.

Line Buffer. To pipeline the convolution over input/output channels, filter taps, and spatial positions with an initiation interval (II) of 1, we fully partitioned the filter weights. A line buffer limited input reads to just two per loop iteration, avoiding aggressive multi-dimensional partitioning of the input array. With these changes we reached an estimated $\sim$4.2K cycles, but exceeded the LUT budget (peaking around 89% utilization).

Tiling. To reduce LUT usage below 85%, we introduced output tiling to shorten pipeline depth. Using a tile factor of 4 in the second convolution lowered LUT utilization to $\sim$65% at $\sim$5.5K cycles. This balance between throughput and resources produced the $\boxed{\texttt{FPGA Final}}$ configuration shown in Table 1.

Flatten Optimizations. We rewrote `flatten` to iterate over a single 1D index and decode 3D coordinates from its bit fields. The HLS tool recognized this as a pure layout transformation, allocating 0 LUTs and 0 cycles, while preserving $\sim$90% accuracy and removing the previous $\sim$1.2K-cycle overhead.

Future Work. The results suggest the BNN would benefit from aggressive kernel fusion: folding `flatten`, padding, and pooling directly into adjacent convolution/dense kernels so producers emit data in the required layout. This should further reduce memory traffic, control overhead, and pipeline depth while keeping resource usage within FPGA limits. I believe with perfect optimization we can get the total estimated cycles below 1K.

3 End-to-End Performance

Implementation	Time (ms)	Speedup (×)
Software Baseline (ARM)	11 887.2	2.32
FPGA Baseline	27 605.9	1.00
FPGA Final	130.4	211.70

Table 2: End-to-end performance over 2000 test images at 90% accuracy. Speedup is normalized to the FPGA Baseline.

On the deployed Zynq platform, the **FPGA Final** configuration completes inference on 2000 images in 130.4 ms, compared with 27.6 s for the unoptimized FPGA Baseline and 11.9 s for the `Software Baseline (ARM)`. This corresponds to a **211.7×** speedup over the FPGA Baseline and a **91.2×** gain over the ARM implementation, all while maintaining 90% classification accuracy. Resource utilization for the final design is modest—11.8% BRAM, 5.4% FF, and 65.2% LUT—indicating that exploiting parallelism via loop unrolling, deep pipelining (II=1), and line buffering delivers substantial performance headroom within on-chip constraints.

4 Conclusion

Through systematic HLS optimization, we achieved a 211.7x speedup in BNN inference while maintaining 90% accuracy and consuming minimal FPGA resources. This demonstrates that high-level synthesis effectively exploits FPGA parallelism for machine learning acceleration, enabling efficient deployment of neural networks on embedded platforms. With even further time spent optimizing, we believe we could have achieved up to a 500x speedup while still remaining in the resource range for the targeted FPGA.

5 Work Distribution

- **Ezra Reiss**: Worked on initial convolution optimizations, convolution tiling, dense optimizations, and overall LUT reduction
- **Sam Belliveau**: Worked on Optimizing the HLS functions, array partitioning, linebuffer creation, optimized flatten function, and data analysis in the report.