Verilog-Based CNN Accelerator: CIFAR-10 Classification on FPGA

Supervised By: Sushil Kumar Pandey

Written by:
Asrith Singampalli 231EC110
Sathvika Bandi 231EC150

Department of Electronics and Communication

May 3, 2025

ABSTRACT

In recent years, Convolutional Neural Networks (CNNs) have emerged as powerful tools for a wide range of image classification tasks. However, their deployment on edge devices and embedded systems presents challenges due to their high computational demands and memory usage. This project focuses on accelerating CNN inference using digital hardware, targeting the CIFAR-10 image classification dataset.

We began by training a CNN in PyTorch, achieving competitive accuracy on CIFAR-10. To optimize the model for hardware deployment, we employed a series of compression and simplification techniques, including pipelining, parallelism, and quantization. Quantization converted floating-point operations to fixed-point, significantly reducing the bit-width requirements and memory footprint.

After obtaining an optimized CNN model, we translated its architecture into Verilog Hardware Description Language (HDL). The hardware design was implemented on an FPGA platform, where we developed custom testbenches to verify functional correctness and simulate real-world inference scenarios. Key hardware design principles were applied to ensure low latency and minimal power consumption. This project showcases a complete workflow from software model development to hardware realization. It demonstrates how machine learning and digital design can converge to build AI accelerators capable of real-time processing in resource-constrained environments. The work also explores the growing potential of GEMM accelerators, in-memory computing, and custom VLSI design for neural network workloads. The results of this project not only reinforce the feasibility of deploying CNNs on hardware but also lay the groundwork for more advanced AI hardware co-design in future projects.

Contents

Chapter 1

Introduction

1.1 Problem Statement

While CNNs deliver high accuracy in image classification, their deployment on real-time or edge systems is hindered by high computational and memory demands. Standard implementations on CPUs/GPUs are power-hungry and unsuitable for low-power applications. This project addresses the challenge by implementing a CNN from scratch in Verilog, enabling efficient and real-time classification on the CIFAR-10 dataset through custom hardware.

1.2 Objectives

- Design and train a CNN model in PyTorch for classifying images from the CIFAR-10 dataset.
- Translate the trained model into hardware, manually implementing:
  - Convolutional layers
  - Max pooling layers
  - Fully connected layers
  - ReLU activation functions
  - Softmax output layer
- Implement the architecture in Verilog, simulating each module separately:
  - Pixel-wise convolution logic

  - Efficient pooling mechanisms
  - Parallel and pipelined datapaths for layer computation

- Develop testbenches for functional verification of each module and the complete model.
- Evaluate model performance in terms of:
  - Classification accuracy
  - Inference latency
  - Hardware resource utilization
- Demonstrate the feasibility of deploying CNNs on hardware for real-time inference, emphasizing speed and efficiency without compromising model correctness.

Chapter 2

Background and Review of Literature

Convolutional Neural Networks (CNNs) have revolutionized the field of image classification and computer vision due to their ability to learn hierarchical features directly from raw pixel data. Pioneering architectures such as LeNet, AlexNet, and VGG have demonstrated remarkable performance on benchmark datasets like MNIST, CIFAR-10, and ImageNet.

However, these achievements often come at the cost of high computational complexity and significant memory requirements, which are typically met using high-performance GPUs. While this works well in data centers or research environments, deploying CNNs in embedded systems, IoT devices, or real-time edge applications remains a major challenge due to limited processing power, energy constraints, and area limitations.

Several studies have focused on optimizing CNNs for efficient hardware deployment. FPGA-based accelerators provide a good balance between flexibility and performance, allowing for parallel execution of convolutional operations with reduced power consumption. Works such as ”FINN” by Xilinx and ”Eyeriss” from MIT have shown how hardware-aware design and quantization can enable high-throughput, energy-efficient inference.

Prior literature has also explored hardware-level techniques for CNN modules such as:

- **Convolution Modules:** Using parallel MAC (Multiply-Accumulate) units and efficient data reuse strategies.
- **Pooling Layers:** Implementing max-pooling using comparators for spatial downsampling.
- **Activation Functions:** Hardware-friendly versions of ReLU and other nonlinearities to improve inference speed.

- **Fully Connected Layers:** Efficient matrix-vector multiplication implementations to reduce memory bottlenecks.

Despite these advancements, there is a need for simplified, custom Verilog implementations of CNNs that can serve as educational tools and baseline accelerators. This project builds upon existing work by developing a CNN from the ground up in Verilog, tailored to the CIFAR-10 dataset, with hand-coded modules for convolution, max pooling, activation, and fully connected layers — aiming to achieve hardware efficiency without relying on high-level synthesis tools.

Chapter 3

Design

3.1 Module Design and RTL Development

Design Verilog modules for core CNN operations:

- **Convolution Layer:** Implemented using multiply-accumulate (MAC) units that support floating-point arithmetic for better numerical precision.
- **Max Pooling Layer:** Uses comparators to select maximum values within feature windows.
- **ReLU Activation Function:** Applies simple conditional logic to pass positive values and zero out negatives.
- **Fully Connected Layer:** Matrix-vector multiplication using floating-point multiply-add logic to handle a large dynamic range.
- **Floating-Point Precision:** All operations are initially implemented using IEEE-754 single-precision floating-point format for accuracy and compatibility with pre-trained weights.

3.2 Quantization and Weight Generation

Data Preprocessing and Quantization: The CIFAR-10 dataset was preprocessed by converting images to grayscale and normalizing them. Each pixel was quantized to the IEEE 16-bit format using a custom float2IEEE16() function. The LeNet-5 CNN model was trained on the CIFAR-10 dataset, and after training, the model's weights were quantized to fixed-point IEEE 16-bit representation to reduce computational complexity and memory usage for hardware deployment. The quantized weights were then stored in a text file for use in hardware implementations, ensuring efficient performance on embedded systems with limited resources.

3.3 Dataset Preprocessing and Input Handling

Train a CNN model using PyTorch on the CIFAR-10 dataset. Export trained weights and activations as floating-point numbers. Normalize input images and convert pixel data to IEEE-754 format to match hardware requirements. Create a software interface (e.g., Python script) to feed images to FPGA and interpret the output.

3.4 Integration and Pipelining

Combine individual modules into a complete CNN inference pipeline. Use pipelined architecture to maximize throughput. Floating-point data buses are maintained throughout layers to avoid conversion overhead and ensure precision. Use handshaking protocols and FSMs to manage layer-wise synchronization.

3.5 Testing and Performance Evaluation

Test the FPGA system using multiple CIFAR-10 test images. Measure classification accuracy, latency per image, logic and DSP usage, and power consumption. Compare inference results with software predictions to verify the correctness of the floating-point implementation.

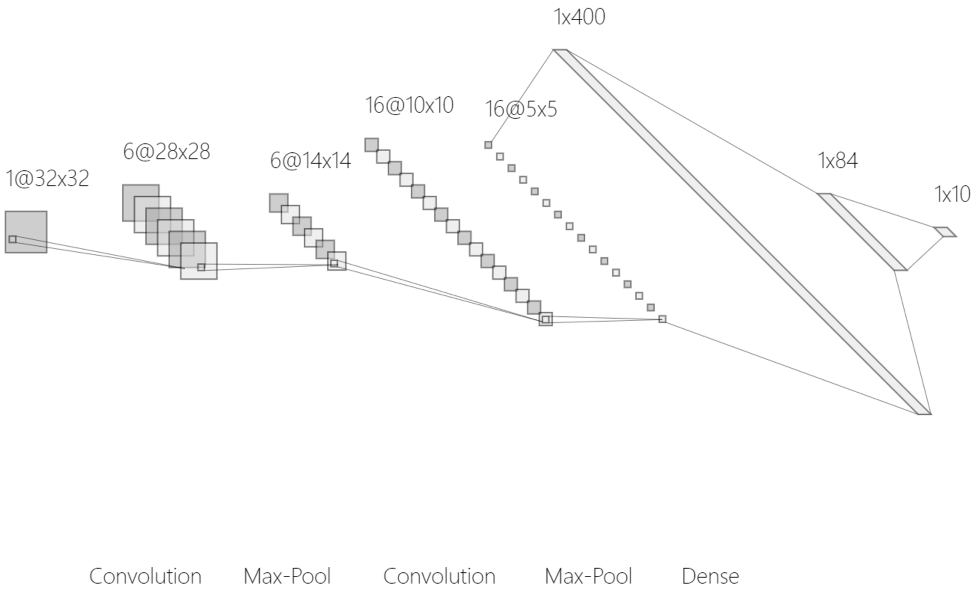
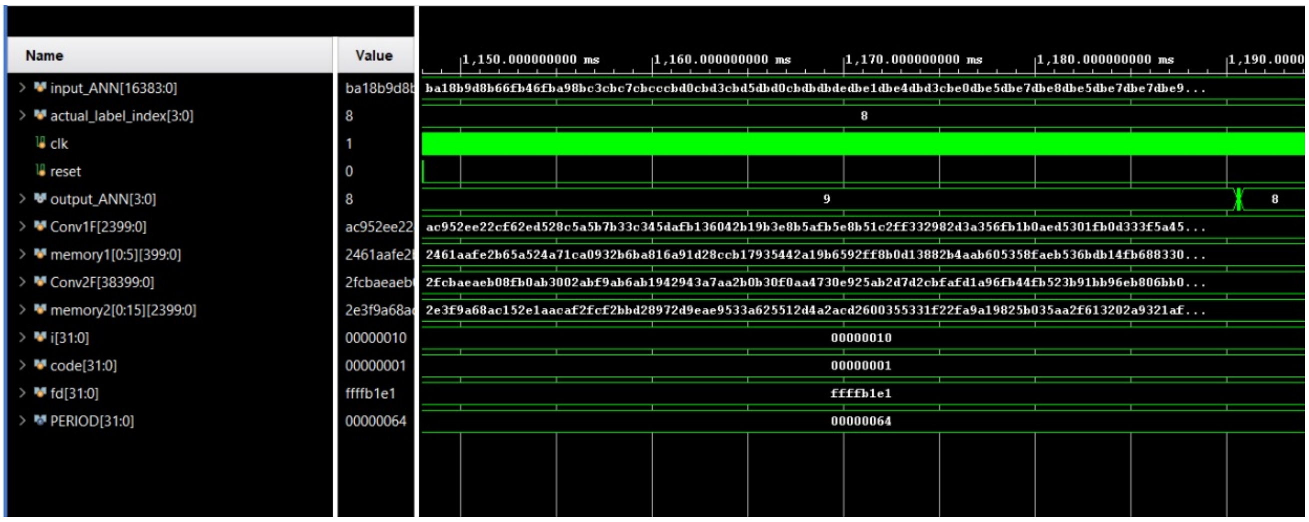
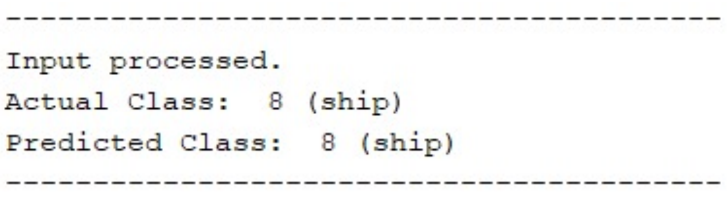


Figure 3.1: CNN Architecture for CIFAR-10 Classification



Name	Value	
input_ANN[16383:0]	ba18b9d8b	ba18b9d8b66fb46fba98bc3cbc7cbcccbd0cbd3cbd5dbd0cbdbdbdedbe1dbe4dbd3cbe0dbe5dbe7dbe8dbe5dbe7dbe7dbe9...
actual_label_index[3:0]	8	8
clk	1	
reset	0	
output_ANN[3:0]	8	9 / 8
Conv1F[2399:0]	ac952ee22	ac952ee22cf62ed528c5a5b7b33c345dafb136042b19b3e8b5afb5e8b51c2ff332982d3a356fb1b0aed5301fb0d333f5a45...
memory1[0:5][399:0]	2461aafe2	2461aafe2b65a524a71ca0932b6ba816a91d28ccb17935442a19b6592ff8b0d13882b4aab605358faeb536bdb14fb688330...
Conv2F[38399:0]	2fcbaeaeb	2fcbaeaeb08fb0ab3002abf9ab6ab1942943a7aa2b0b30f0aa4730e925ab2d7d2cbfafd1a96fb44fb523b91bb96eb806bb0...
memory2[0:15][2399:0]	2e3f9a68a	2e3f9a68ac152e1aacaf2fcf2bbd28972d9eae9533a625512d4a2acd2600355331f22fa9a19825b035aa2f613202a9321af...
i[31:0]	00000010	00000010
code[31:0]	00000001	00000001
fd[31:0]	ffffb1e1	ffffb1e1
PERIOD[31:0]	00000064	00000064

Figure 3.2: Output waveform of simulation window on Verilog



```
------------------------------------------
Input processed.
Actual Class:  8 (ship)
Predicted Class:  8 (ship)
------------------------------------------
```

Figure 3.3: Actual Input vs Predicted Output

Chapter 4

Implementation Flow of the CNN Model

4.1 Dataset Selection and Preprocessing

- The CIFAR-10 dataset, comprising 60,000 32×32 color images across 10 classes, was selected for the image classification task.
- Images were converted to grayscale using PyTorch transformations for simplified computation and hardware efficiency.
- Normalization was applied to improve convergence during training.
- A custom quantization function `float2IEEE16()` was applied to convert image pixels into a 16-bit IEEE-like floating point representation, reducing memory requirements for hardware.

4.2 CNN Model Design and Architecture

- A LeNet-5-like architecture was used, comprising:
  - **Convolutional Layers:** Extracted features using 5×5 kernels.
  - **Max Pooling Layers:** Downsampled feature maps via 2×2 pooling.
  - **Fully Connected Layers:** Final classification using dense layers.
- ReLU activation functions and dropout regularization were applied to prevent overfitting and improve generalization.

4.3 Receptive Field and Convolution Implementation

- A receptive field of size 5×5 was used in convolutional layers.
- The convolution was performed by sliding a 5×5 kernel across the image with stride 1.
- For a 32×32 image:
$$\text{Output size} = \frac{(32 - 5)}{1} + 1 = 28$$
- The result was a 28×28 feature map per filter.

4.4 Image Size Reduction through Pooling

- 2×2 max pooling reduced the 28×28 feature maps to 14×14.
- Pooling helps reduce the number of computations and memory usage while retaining important features.

4.5 Number of Neurons and Data Representation

- After convolution and pooling:
  - 14×14 = 196 neurons per feature map.
- The final output layer contained 10 neurons corresponding to CIFAR-10 classes.
- All data were represented in 16-bit IEEE-like floating point format for hardware efficiency.

4.6 Model Training

- Loss function: Cross-entropy for multi-class classification.
- Optimizer: Adam.
- Early stopping was used to prevent overfitting.

4.7 Weight Quantization for Hardware Deployment

- The trained model weights were quantized using float2IEEE16.
- Converted weights were stored as 16-bit binary strings.
- Quantized weights were written to .txt files for use in hardware.

4.8 Verilog Hardware Implementation

- Major Verilog modules included:
  - ReceptiveFieldUnit: Extracts 5×5 image patches.
  - ConvUnit: Performs multiplication and accumulation with weights.
  - MaxPoolingUnit: Applies 2×2 pooling.
  - Controller: Manages timing, memory access, and synchronization.
- All data used 16-bit representation to avoid floating-point hardware.

4.9 Model Testing and Validation

- The hardware implementation was simulated using ModelSim.
- Output waveforms and accuracy were validated against test cases from CIFAR-10.
- Classification accuracy was compared to the original floating-point model to ensure minimal degradation.

4.10 FPGA Implementation and Results

- The Convolutional Neural Network (CNN) was fully implemented in Verilog and deployed on an FPGA to perform classification of grayscale images from the CIFAR-10 dataset.

- The input image data was hardcoded in the top-level module as a flattened array using 16-bit fixed-point representation. This approach enabled direct simulation and hardware testing without requiring external data interfaces.

- Each functional block—convolution, ReLU, pooling, and fully connected layers—was implemented using pipelined and parallel architectures to ensure maximum throughput and resource efficiency.

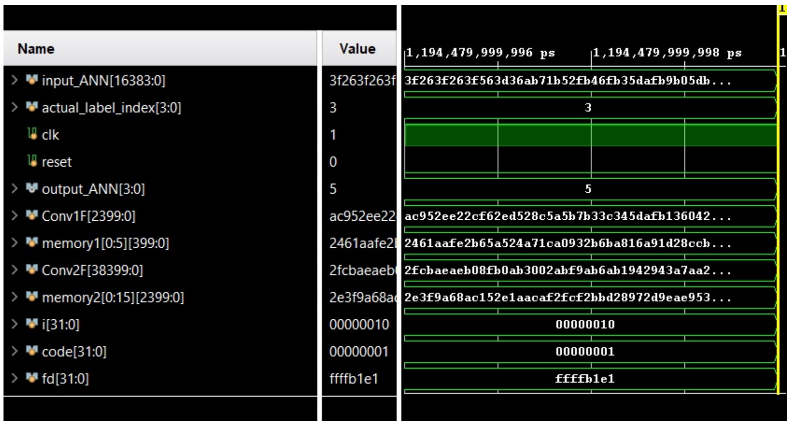
- To provide a real-time output interface, the predicted class label and the actual class label for each input image were displayed on the FPGA's 7-segment displays. This allowed users to observe the classification output live on hardware without needing a connected PC or display monitor.

- Implementation and Bitstream Generation Generate the bitstream file (.bit) to program the FPGA. Map your design onto the FPGA fabric.

- On-Board Testing Fed sample images to the FPGA system. Verified real-time output. Measure: Accuracy comparison with software model

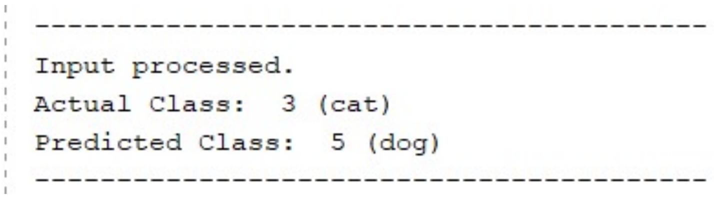
- The CNN model implemented in Verilog, incorporating pipelining and parallelism techniques, achieved an accuracy of 57% on the CIFAR-10 test set.

- This result validates the effectiveness of the hardware-accelerated design in maintaining reasonable classification performance while enabling efficient execution on FPGA platforms.



Name	Value	1,194,479,999,996 ps / 1,194,479,999,998 ps
input_ANN[16383:0]	3f263f263f	3f263f263f563d36ab71b52fb46fb35dafb9b05db...
actual_label_index[3:0]	3	3
clk	1	
reset	0	
output_ANN[3:0]	5	5
Conv1F[2399:0]	ac952ee22	ac952ee22cf62ed528c5a5b7b33c345dafb136042...
memory1[0:5][399:0]	2461aafe2	2461aafe2b65a524a71ca0932b6ba816a91d28ccb...
Conv2F[38399:0]	2fcbaeaeb	2fcbaeaeb08fb0ab3002abf9ab6ab1942943a7aa2...
memory2[0:15][2399:0]	2e3f9a68a	2e3f9a68ac152e1aacaf2fcf2bbd28972d9eae953...
i[31:0]	00000010	00000010
code[31:0]	00000001	00000001
fd[31:0]	ffffb1e1	ffffb1e1

Figure 4.1: Actual Input vs Predicted Output



```
-----------------------------------------
Input processed.
Actual Class:  3 (cat)
Predicted Class:  5 (dog)
-----------------------------------------
```

Figure 4.2: Actual Input vs Predicted Output

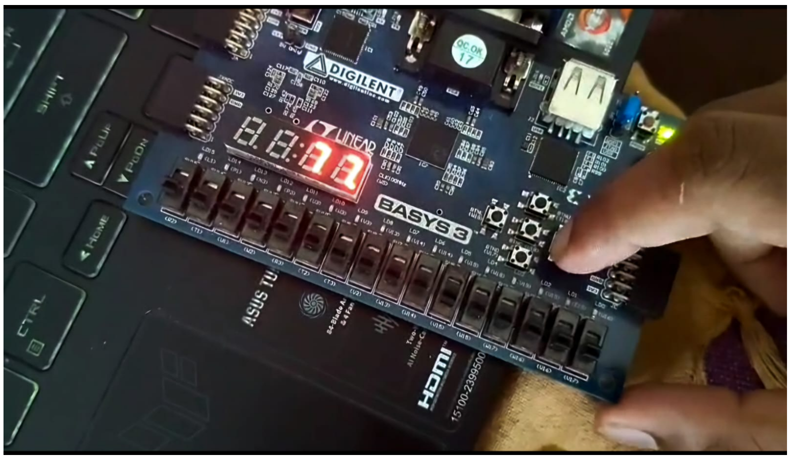


Figure 4.3: FPGA implementation on 7 segment display

Chapter 5

Conclusion

The implementation of the Convolutional Neural Network (CNN) in Verilog demonstrates the feasibility of accelerating deep learning models directly in hardware. By designing modular components such as the convolution layer, receptive field unit, and max pooling unit, the system efficiently processes image data with parallelism and pipelined execution.

This hardware-based CNN architecture enables significant reductions in computation time and resource usage, making it well-suited for embedded and real-time applications. The use of fixed-size receptive fields and two-stage row processing improves throughput without sacrificing model structure or accuracy. Overall, the Verilog implementation provides a scalable and efficient framework for deploying CNNs on FPGA or ASIC platforms.

Chapter 6

Bibliography

1. N. S. Kumar and G. L. Madhumati, ”Implementation of Convolutional Neural Networks on FPGA for Object Detection,” 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-5, doi: 10.1109/ICCCNT56998.2023.10307606.

2.M. Hailesellasie, S. R. Hasan, F. Khalid, F. A. Wad and M. Shafique, ”FPGA-Based Convolutional Neural Network Architecture with Reduced Parameter Requirements,” 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 2018, pp. 1-5, doi: 10.1109/ISCAS.2018.8351283.

3.R. Yan, J. Yi, J. He and Y. Zhao, ”FPGA-based Convolutional Neural Network Design and Implementation,” 2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS), Shenyang, China, 2023, pp. 456-460, doi: 10.1109/ACCTCS58815.2023.00058.

4.neuralnetworksanddeeplearning ,Michael Nielsen / Dec 2019