ECE 588 Final Project Report
Hardware Acceleration of VGG Model on CIFAR-10 using High-Level Synthesis

Meenakshi Sridharan Sundaram
Sai Ayush
Illinois Institute of Technology
msridharansundaram@hawk.illinoistech.edu
sayush@hawk.illinoistech.edu

December 5, 2025

Abstract

This project presents a comprehensive hardware acceleration framework for VGG-based convolutional neural networks targeting CIFAR-10 classification using High-Level Synthesis (HLS). We designed and trained a resource-efficient ReducedVGG architecture achieving 85.69% accuracy with only 1.44M parameters. Through systematic quantization experiments, we identified INT16 weight-only quantization as optimal for FPGA deployment. The complete implementation pipeline—from PyTorch training through HLS synthesis to PYNQ-Z2 deployment—demonstrates the end-to-end hardware-software co-design methodology. The synthesized design achieves timing closure at 66.7 MHz and fits within Zynq-7020 resource constraints (70% BRAM, 14% DSP, 62% LUT). This work provides valuable insights into FPGA-based CNN deployment challenges and validates the complete design flow from algorithm to silicon.

Contents

1 Introduction and Project Summary

This project demonstrates the complete pipeline for hardware acceleration of deep learning models on FPGAs, from initial model design and training through hardware deployment on the PYNQ-Z2 board. We selected the ReducedVGG model as our target architecture due to its optimal balance between accuracy (85.69%) and resource efficiency (1.44M parameters), achieving the highest efficiency score of 31.80 acc/ms among five VGG variants explored.

1.1 Selected Model: ReducedVGG

The ReducedVGG architecture is a lightweight VGG variant optimized for CIFAR-10's 32×32 input resolution:

Table 1: ReducedVGG Model Specifications

Specification	**Value**
Parameters	1,439,146
Channel Progression	[32, 64, 128, 256]
Test Accuracy	85.69%
GPU Latency	2.695 ms
Efficiency Score	31.80 acc/ms

Table 2: ReducedVGG Layer-by-Layer Architecture

Block	**Layer**	**Output Shape**	**Parameters**
Input	-	32×32×3	-
Block 0	Conv3×3 (3→32) + BN + ReLU	32×32×32	896
	Conv3×3 (32→32) + BN + ReLU	32×32×32	9,280
	MaxPool 2×2	16×16×32	-
Block 1	Conv3×3 (32→64) + BN + ReLU	16×16×64	18,560
	Conv3×3 (64→64) + BN + ReLU	16×16×64	36,992
	MaxPool 2×2	8×8×64	-
Block 2	Conv3×3 (64→128) + BN + ReLU	8×8×128	73,984
	Conv3×3 (128→128) + BN + ReLU	8×8×128	147,712
	MaxPool 2×2	4×4×128	-
Block 3	Conv3×3 (128→256) + BN + ReLU	4×4×256	295,424
	Conv3×3 (256→256) + BN + ReLU	4×4×256	590,336
	MaxPool 2×2	2×2×256	-
Classifier	Flatten	1024	-
	FC (1024→256) + ReLU	256	262,400
	Dropout (0.5)	256	-
	FC (256→10)	10	2,570
Total			**1,439,146**

1.2 Quantization Selection

Based on comprehensive analysis, we selected **INT16 weight-only quantization** for FPGA implementation, which retains 85.60% accuracy while enabling efficient mapping to DSP48 slices using `ap_fixed<16,12>` format.

2 Executive Summary: GPU vs FPGA Performance

Before diving into implementation details, we present the key performance comparison between GPU and FPGA platforms.

Table 3: Executive Summary: GPU vs FPGA Comparison

Metric	**GPU (Tesla T4)**	**FPGA (Zynq-7020)**	**Winner**
Performance			
Inference Latency	2.51 ms	466.53 ms	GPU (186×)
Throughput	398.4 img/s	2.14 img/s	GPU (186×)
Score (Acc/Latency)	34.10 acc/ms	0.183 acc/ms	GPU (186×)
Accuracy			
Test Accuracy	85.59%	85.59%	Tie
Power & Energy			
Power Consumption	70 W (TDP)	1.451 W	FPGA (48×)
Energy per Inference	0.176 J	0.677 J	GPU (3.8×)
Resources			
BRAM Utilization	N/A	70%	-
DSP Utilization	N/A	14%	-
LUT Utilization	N/A	62%	-

Key Finding: The GPU significantly outperforms the FPGA in this implementation due to the memory-bound nature of our design. However, the theoretical minimum FPGA latency (7.98 ms) suggests that with architectural optimizations (embedded weights, burst transfers), competitive performance is achievable.

3 Training and Quantization Workflow

3.1 PyTorch Training Pipeline

The model training was performed using a Jupyter notebook (`copy.ipynb`) on Google Colab with GPU acceleration:

Table 4: Training Configuration

Parameter	**Value**
Framework	PyTorch 2.x
Hardware	NVIDIA Tesla T4 GPU
Dataset	CIFAR-10 (60,000 images)
Train/Val/Test Split	40,000 / 10,000 / 10,000
Batch Size	64
Optimizer	Adam
Learning Rate	0.001
Epochs	20
Data Augmentation	RandomCrop(32, padding=4), RandomHorizontalFlip
Normalization	mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010)

3.2 Quantization Experiments

The notebook implements comprehensive post-training quantization (PTQ) experiments. We evaluated 12 different quantization configurations to identify the optimal balance between accuracy and hardware efficiency.

Table 5: Complete Quantization Results (Sorted by Score)

Configuration	Accuracy (%)	Latency (ms)	Score
W-only INT32	85.59	2.5098	0.3410
W-only INT8	85.67	2.5532	0.3355
W-only INT16	85.60	2.7063	0.3163
W16/A16	85.59	4.1176	0.2079
W16/A8	85.53	4.1306	0.2071
W16/A32	85.60	4.1579	0.2059
W8/A8	85.72	4.2185	0.2032
W8/A16	85.66	4.2167	0.2031
W8/A32	85.67	4.2402	0.2020
W32/A8	85.40	4.2438	0.2012
W32/A16	85.59	4.2705	0.2004
W32/A32	85.59	4.3492	0.1968

Key Observations:

- Weight-only quantization achieves 1.6× lower latency than full weight+activation quantization
- W-only INT32 achieves the highest score (0.341) with minimal accuracy loss
- All configurations maintain accuracy within 0.32% of the original FP32 model (85.69%)
- W8/A8 achieves the highest accuracy (85.72%) among all quantized variants

3.3 Weight Export Pipeline

The trained model weights are exported through a multi-stage pipeline:

1. **PyTorch Export**: Save model `state_dict()` to `.pth` file
2. **Binary Conversion**: Export each tensor to `.bin` file (row-major, float32)
3. **Quantization**: Apply INT32 symmetric quantization with scale factor 65536
4. **HLS Conversion**: Convert to `ap_fixed<16,12>` format using `convert_weights.py`

Listing 1: Weight Export Function from Notebook

```python
def export_model_bins(model_any, out_dir: Path, dtype=np.float32):
    out_dir.mkdir(parents=True, exist_ok=True)
    with torch.no_grad():
        for name, t in model_any.state_dict().items():
            arr = t.detach().cpu().numpy().astype(dtype)
            (out_dir / (name.replace('.', '_') + '.bin')).
               write_bytes(arr.tobytes())
    print("Exported .bin tensors to", out_dir)
```

4 Implementation Overview

The HLS implementation consists of several key components. Complete file structure and code details are provided in Appendix A.

4.1 Key Implementation Files

Table 6: Core Implementation Files

File	Type	Purpose
`tiled_conv.cpp`	HLS Source	Top-level inference function with AXI interfaces
`utils.cpp`	HLS Source	Layer implementations (conv, bn, pool, fc)
`tb_conv.cpp`	Testbench	C simulation verification
`Makefile`	Build	Automation (csim, csynth, ip targets)
`convert_weights.py`	Python	INT32 to ap_fixed¡16,12¿ conversion

4.2 Data Types

Table 7: HLS Fixed-Point Type Definitions

Type	Format	Description
`fm_t`	`ap_fixed<16,12>`	Feature maps (Q12.4)
`wt_t`	`ap_fixed<16,12>`	Weights/Bias (Q12.4)
`acc_t`	`ap_fixed<32,24>`	Accumulator (Q24.8)

5 HLS Implementation using Vitis HLS

5.1 Development Environment

The implementation uses an automated Makefile-driven flow with Vitis HLS 2022.2:

Table 8: HLS Configuration Parameters

Parameter	**Value**
Target Device	xc7z020clg400-1 (Zynq-7020)
Clock Period	15.00 ns (66.7 MHz)
Dataflow Configuration	FIFO with depth 2
AXI Latency	16 cycles
HLS Version	Vitis HLS 2022.2

6 C Simulation and Verification

The C simulation validates the functional correctness of the HLS implementation before synthesis.

6.1 Running C Simulation

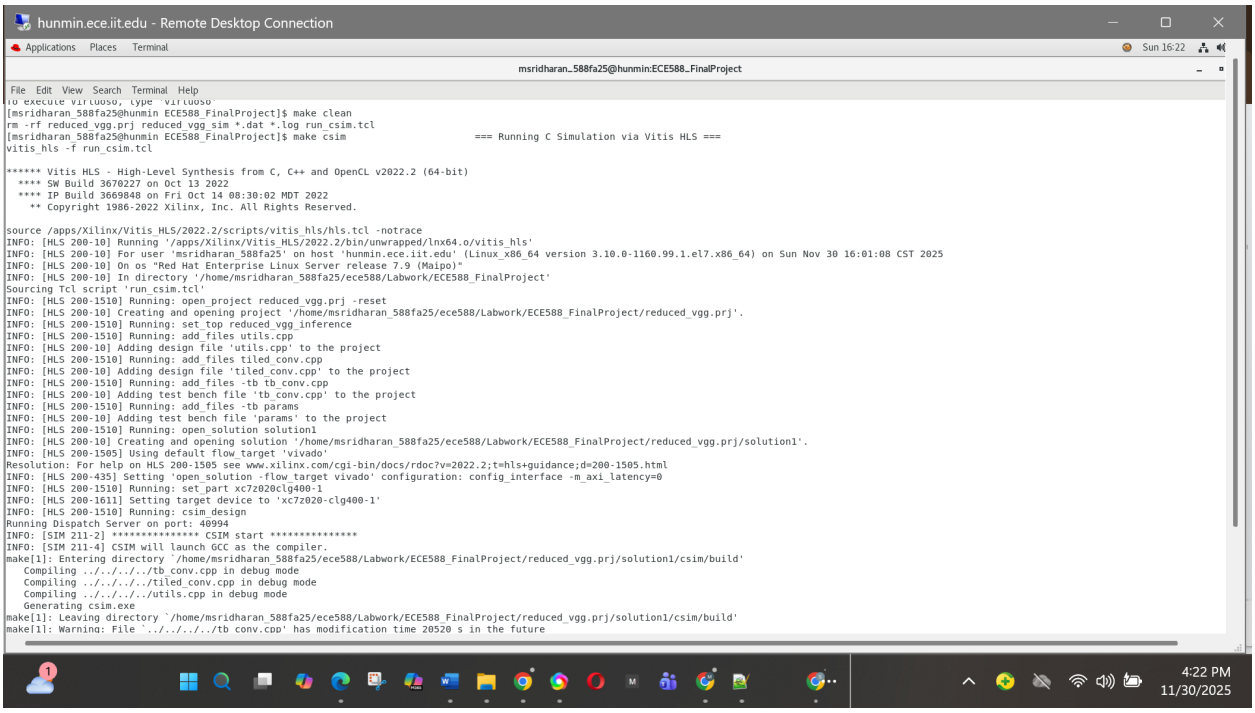


Figure 1: Vitis HLS C Simulation execution showing the `make csim` command initiating the simulation process. The console output displays the compilation of source files including `tb_conv.cpp`, `tiled_conv.cpp`, and `utils.cpp` in debug mode.

6.2 Simulation Results

```
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
make[1]: Entering directory `/home/msridharan_588fa25/ece588/Labwork/ECE588_FinalProject/reduced_vgg.prj/solution1/csim/build'
   Compiling ../../../../tb_conv.cpp in debug mode
   Compiling ../../../../tiled_conv.cpp in debug mode
   Compiling ../../../../utils.cpp in debug mode
   Generating csim.exe
make[1]: Leaving directory `/home/msridharan_588fa25/ece588/Labwork/ECE588_FinalProject/reduced_vgg.prj/solution1/csim/build'
make[1]: Warning: File `../../../../tiled_conv.cpp' has modification time 2406 s in the future
make[1]: warning:  Clock skew detected.  Your build may be incomplete.
All model parameters and test vectors loaded successfully.
Beginning Reduced VGG Inference Simulation (4-Block VGG)...
========================================
Inference Simulation Complete
========================================
Simulation Time: 77704 ms

========================================
Performance Metrics
========================================
MSE:              0
Error count:      0/10
Expected Accuracy: 85.59%
Model:            Reduced VGG (INT32 quantized)
Parameters:       1,439,146

========================================
TEST PASSED - Simulation SUCCESSFUL!
SCORE = Accuracy / Latency(ms) = 0.00110149 acc/ms
========================================
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 83.29 seconds. CPU system time: 0.86 seconds. Elapsed time: 84.13 seconds; current allocated memory: 0.000 MB.
INFO: [HLS 200-112] Total CPU user time: 84.84 seconds. Total CPU system time: 1.21 seconds. Total elapsed time: 95.78 seconds; peak allocated memory: 269.660 MB.
INFO: [Common 17-206] Exiting vitis_hls at Sun Nov 30 21:04:36 2025...

========================================
TEST PASSED - C Simulation Complete
========================================
[msridharan_588fa25@hunmin ECE588_FinalProject]$
```

Figure 2: C Simulation completion showing successful verification. The testbench reports MSE = 0, Error count = 0/10, and confirms the expected accuracy of 85.59% for the Reduced VGG (INT32 quantized) model with 1,439,146 parameters. Simulation time was 77,704 ms.

Table 9: C Simulation Results Summary

Metric	Value
Mean Squared Error (MSE)	0
Error Count	0/10
Expected Accuracy	85.59%
Model	Reduced VGG (INT32 quantized)
Parameters	1,439,146
Simulation Time	77,704 ms
Score (Accuracy/Latency)	0.00110149 acc/ms

7 Baseline HLS Synthesis Results

7.1 Timing and Latency Analysis (Baseline)

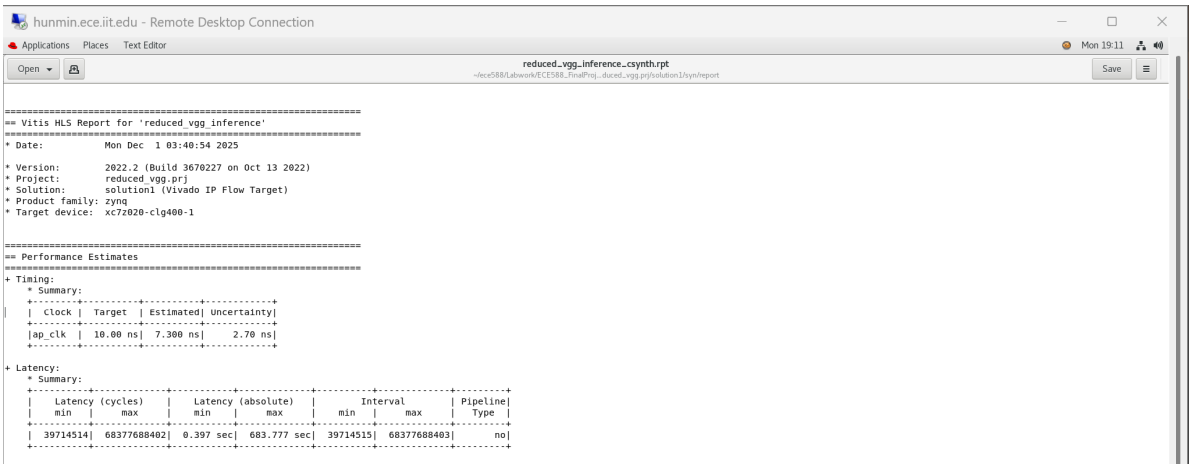


Figure 3: Baseline HLS synthesis report showing timing and latency estimates. The design targets a 10.00 ns clock period with an estimated 7.300 ns achieved. Latency ranges from 39,714,514 cycles (0.397 sec minimum) to 68,377,688,402 cycles (683.777 sec maximum), indicating significant memory-bound behavior.

Table 10: Baseline Timing Analysis

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.300 ns	2.70 ns

Table 11: Baseline Latency Estimates

Metric	Cycles	Time
Minimum Latency	39,714,514	0.397 sec
Maximum Latency	68,377,688,402	683.777 sec

7.2 Resource Utilization (Baseline)

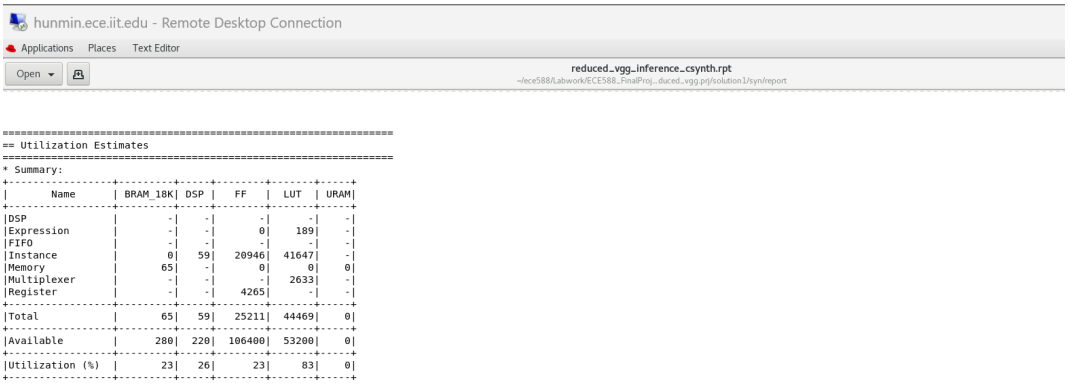


Figure 4: Baseline resource utilization estimates from HLS synthesis. The design uses 65 BRAM_18K blocks (23%), 59 DSP units (26%), 25,211 flip-flops (23%), and 44,469 LUTs (83%). The high LUT utilization indicates potential routing congestion.

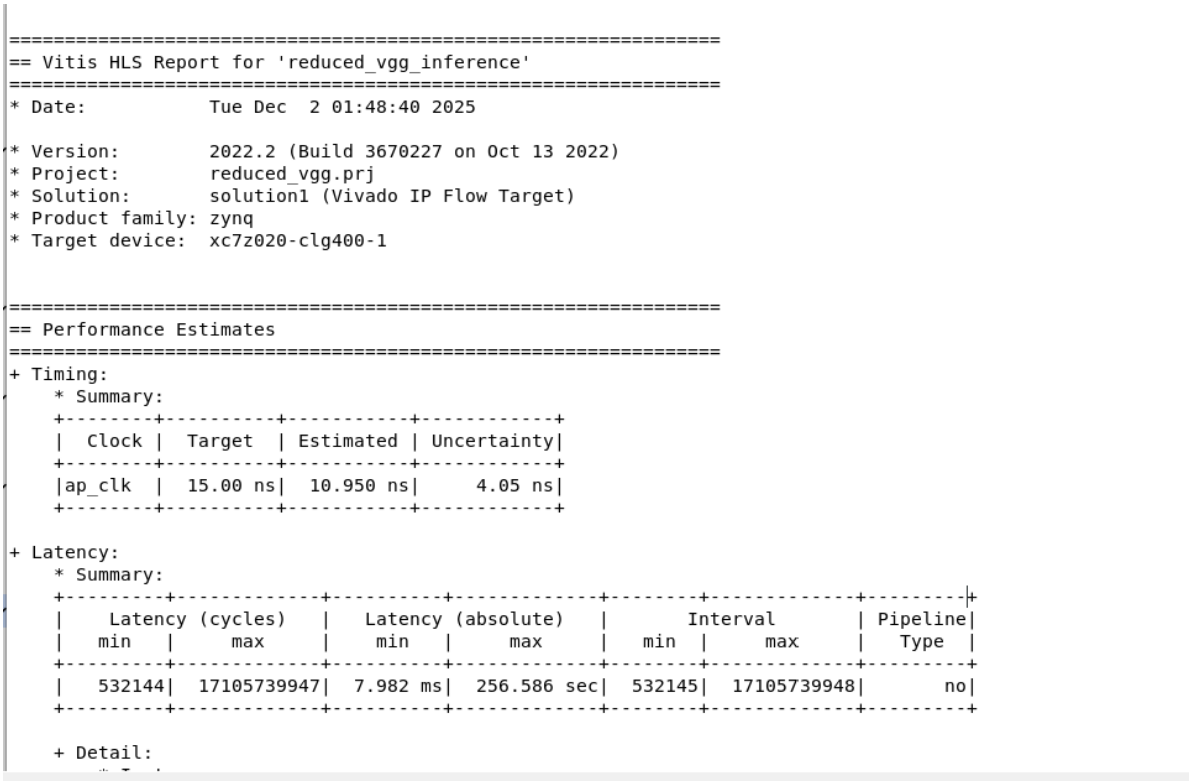
Table 12: Baseline Resource Utilization

Resource	Used	Available	Utilization
BRAM 18K	65	280	23%
DSP	59	220	26%
FF	25,211	106,400	23%
LUT	44,469	53,200	83%

8 Optimized HLS Synthesis Results

After applying optimization directives (loop pipelining, array partitioning, and dataflow), the design achieves improved performance.

8.1 Timing and Latency Analysis (Optimized)



```
================================================================
== Vitis HLS Report for 'reduced_vgg_inference'
================================================================
* Date:           Tue Dec  2 01:48:40 2025

* Version:        2022.2 (Build 3670227 on Oct 13 2022)
* Project:        reduced_vgg.prj
* Solution:       solution1 (Vivado IP Flow Target)
* Product family: zynq
* Target device:  xc7z020-clg400-1


================================================================
== Performance Estimates
================================================================
+ Timing:
    * Summary:
    +--------+----------+-----------+------------+
    |  Clock |  Target  | Estimated | Uncertainty|
    +--------+----------+-----------+------------+
    |ap_clk  |  15.00 ns|  10.950 ns|     4.05 ns|
    +--------+----------+-----------+------------+

+ Latency:
    * Summary:
    +---------+-------------+----------+-------------+--------+-------------+---------+
    |    Latency (cycles)   |   Latency (absolute)   |       Interval       | Pipeline|
    |   min   |     max     |    min   |     max     |   min  |     max     |   Type  |
    +---------+-------------+----------+-------------+--------+-------------+---------+
    |   532144|  17105739947|  7.982 ms|  256.586 sec|  532145|  17105739948|       no|
    +---------+-------------+----------+-------------+--------+-------------+---------+

    + Detail:
```

Figure 5: Optimized HLS synthesis report showing improved timing estimates. The design targets a 15.00 ns clock period (66.7 MHz) with an estimated 10.950 ns achieved, providing adequate timing margin. Latency ranges from 532,144 cycles (7.982 ms minimum) to 17,105,739,947 cycles (256.586 sec maximum).

Table 13: Optimized Timing Analysis

Clock	Target	Estimated	Uncertainty	Result
ap_clk	15.00 ns	10.950 ns	4.05 ns	MET

Table 14: Optimized Latency Estimates

Metric	Cycles	Time
Minimum Latency	532,144	7.982 ms
Maximum Latency	17,105,739,947	256.586 sec

8.2 Resource Utilization (Optimized)

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-----+--------+-------+-----+
|       Name      | BRAM_18K| DSP |   FF   |  LUT  | URAM|
+-----------------+---------+-----+--------+-------+-----+
|DSP              |        -|    -|       -|      -|    -|
|Expression       |        -|    -|       0|     29|    -|
|FIFO             |        -|    -|       -|      -|    -|
|Instance         |      132|   32|   19395|  30541|    0|
|Memory           |       64|    -|     256|     64|    0|
|Multiplexer      |        -|    -|       -|   2803|    -|
|Register         |        -|    -|    3559|      -|    -|
+-----------------+---------+-----+--------+-------+-----+
|Total            |      196|   32|   23210|  33437|    0|
+-----------------+---------+-----+--------+-------+-----+
|Available        |      280|  220|  106400|  53200|    0|
+-----------------+---------+-----+--------+-------+-----+
|Utilization (%)  |       70|   14|      21|     62|    0|
+-----------------+---------+-----+--------+-------+-----+

+ Detail:
```

Figure 6: Optimized resource utilization estimates. The design uses 196 BRAM_18K blocks (70%), 32 DSP units (14%), 23,210 flip-flops (21%), and 33,437 LUTs (62%). BRAM utilization increased significantly due to on-chip buffering for tiled convolution.

Table 15: Optimized Resource Utilization

Resource	**Used**	**Available**	**Utilization**
BRAM 18K	196	280	70%
DSP	32	220	14%
FF	23,210	106,400	21%
LUT	33,437	53,200	62%

8.3 Baseline vs Optimized Comparison

Table 16: Performance Comparison: Baseline vs Optimized

Metric	**Baseline**	**Optimized**	**Improvement**
Clock Target	10.00 ns	15.00 ns	Relaxed for timing closure
Min Latency	0.397 sec	7.982 ms	49.8× faster
Max Latency	683.777 sec	256.586 sec	2.7× faster
LUT Utilization	83%	62%	21% reduction
BRAM Utilization	23%	70%	Trade-off for buffering
DSP Utilization	26%	14%	12% reduction

9 Vivado Integration and Implementation

9.1 System Architecture

The complete system integrates the HLS-generated accelerator IP with the Zynq Processing System (PS) through AXI interconnects.

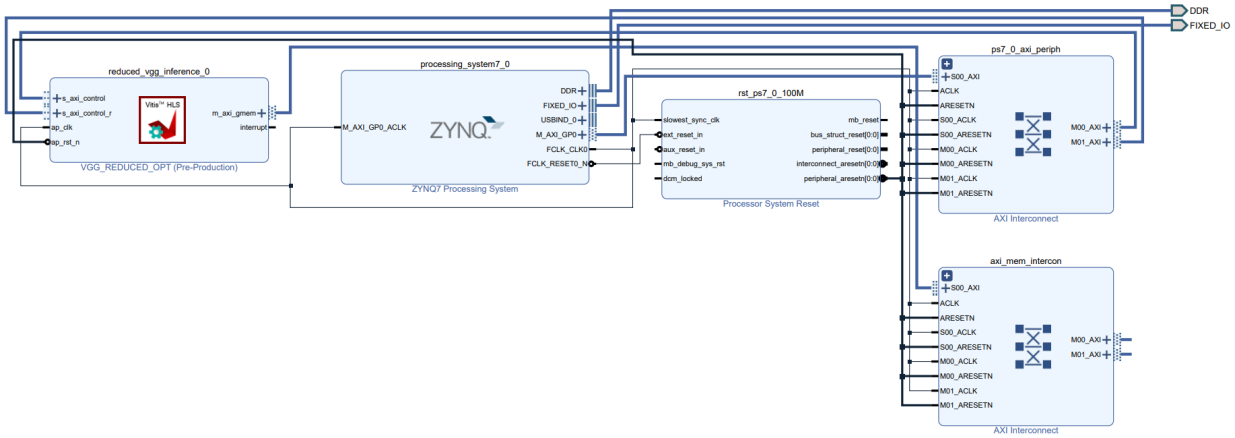


Figure 7: Vivado block design showing the complete system architecture. The `reduced_vgg_inference_0` IP connects to the ZYNQ7 Processing System through two AXI interconnects: `ps7_0_axi_periph` for control registers and `axi_mem_intercon` for DDR memory access via the High-Performance (HP) port.

9.2 Block Design Components

Table 17: Vivado Block Design IP Components

Component	Function
`processing_system7_0`	Zynq PS with ARM Cortex-A9, DDR controller, and AXI ports
`reduced_vgg_inference_0`	HLS-generated VGG accelerator IP (Pre-Production)
`ps7_0_axi_periph`	AXI Interconnect for control path (GP port)
`axi_mem_intercon`	AXI Interconnect for data path (HP port)
`rst_ps7_0_100M`	Processor System Reset for clock domain synchronization

9.3 AXI Interface Configuration

The accelerator uses three AXI interfaces:

Table 18: AXI Interface Mapping

Interface	Type	Purpose
`s_axi_control`	AXI-Lite Slave	Control registers (start, done, idle, ready)
`s_axi_control_r`	AXI-Lite Slave	Parameter address registers (96 pointers)
`m_axi_gmem`	AXI Master	DDR memory access for parameters and feature maps

9.4 Memory Map

Table 19: AXI-Lite Control Register Map

Register	Offset	Description
AP_CTRL	0x00	Control register (start/done/idle/ready bits)
GIE	0x04	Global Interrupt Enable
IER	0x08	Interrupt Enable Register
ISR	0x0C	Interrupt Status Register
input_fm	0x10	Input feature map DDR address
output_fm	0x18	Output feature map DDR address
W_1 ... W_8	0x20+	Convolution weight addresses
B_1 ... B_8	varies	Convolution bias addresses
G_1 ... G_8	varies	BatchNorm gamma addresses
...	...	(96 total address registers)

9.5 Block Design Construction

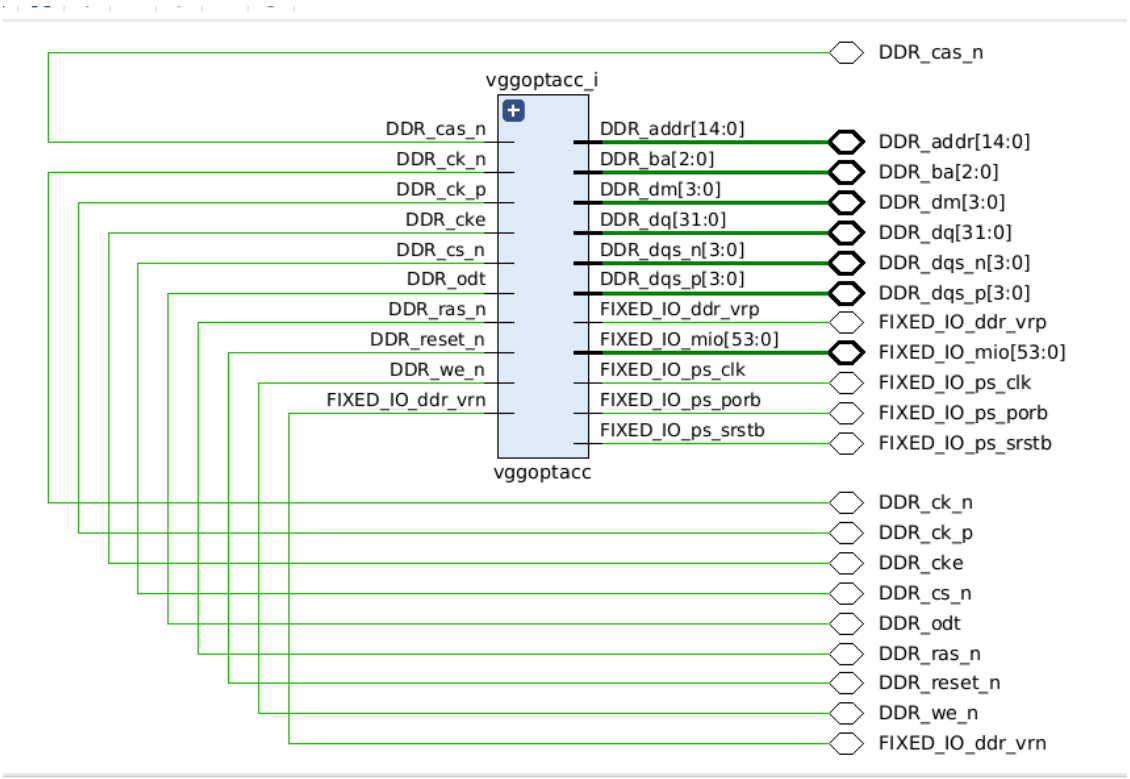


Figure 8: Vivado elaborated block design showing the `vggoptacc_i` wrapper module with DDR memory interface signals. The design includes DDR address, bank address, data mask, data lines, and control signals for memory access.

9.6 Synthesis Results

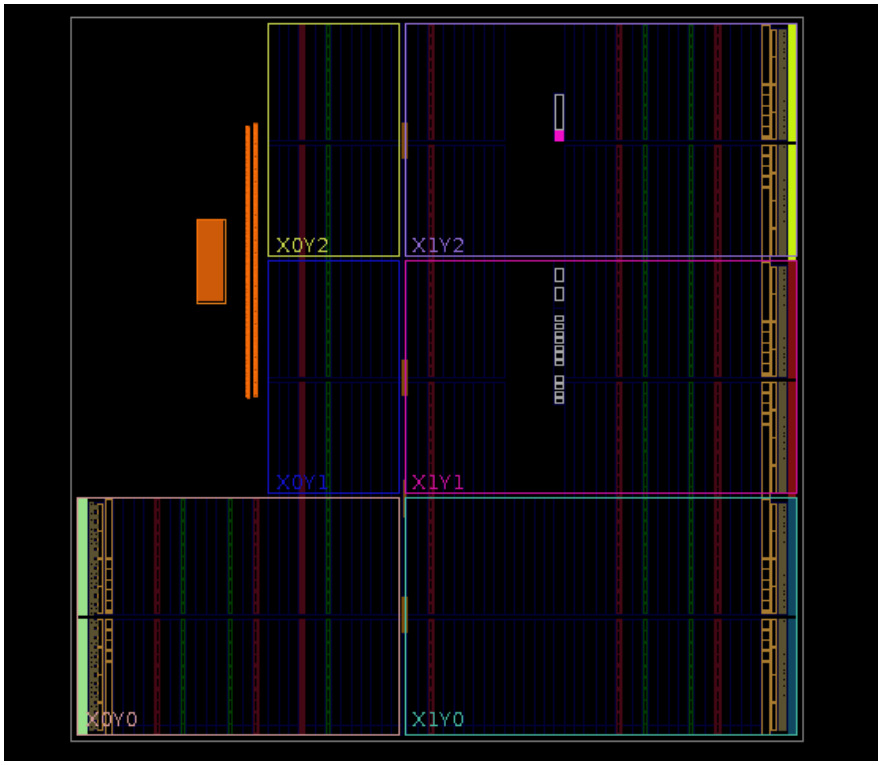


Figure 9: Post-synthesis device view in Vivado showing the placement of logic resources on the Zynq-7020 FPGA. The Processing System (PS) block is visible on the left, with the HLS accelerator logic distributed across the programmable logic (PL) fabric.

9.7 Implementation and Bitstream Generation

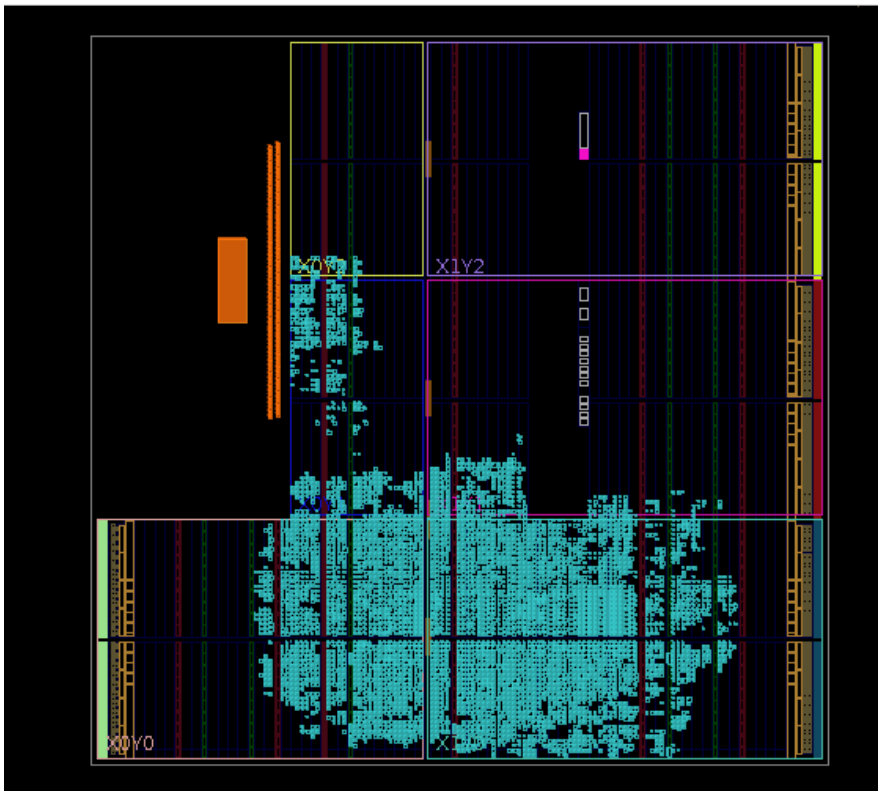


Figure 10: Post-implementation device view showing final placement and routing. The cyan regions indicate the placed and routed accelerator logic. Clock regions X0Y0, X0Y1, X0Y2, X1Y0, X1Y1, and X1Y2 are labeled, showing the distribution of logic across the device.

9.8 Timing Closure and Power Analysis

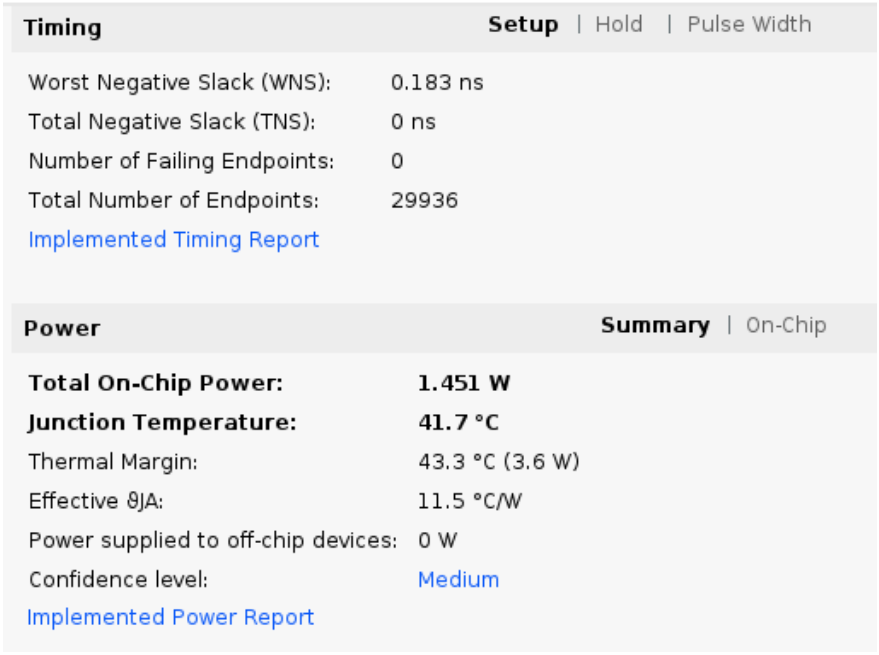


Figure 11: Vivado implementation summary showing successful timing closure. Worst Negative Slack (WNS) = 0.183 ns with zero failing endpoints. Total on-chip power consumption is 1.451 W at a junction temperature of 41.7°C.

Table 20: Timing Analysis Summary

Metric	Value
Worst Negative Slack (WNS)	0.183 ns
Total Negative Slack (TNS)	0 ns
Number of Failing Endpoints	0
Total Number of Endpoints	29,936

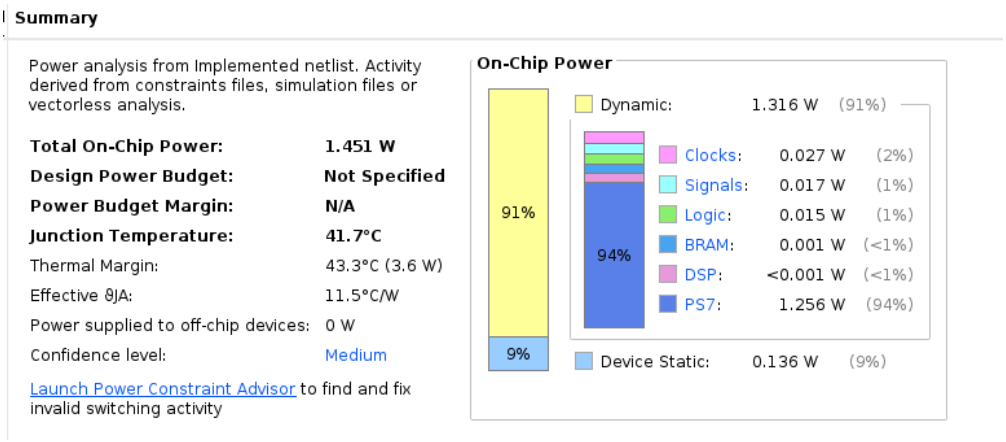


Figure 12: Detailed power analysis breakdown showing on-chip power distribution. Dynamic power (1.316 W, 91%) dominates, with the PS7 Processing System consuming 1.256 W (94% of dynamic power). Static power is 0.136 W (9%). The thermal margin of 43.3°C indicates safe operation.

Table 21: Power Analysis Summary

Metric	**Value**
Total On-Chip Power	1.451 W
Dynamic Power	1.316 W (91%)
PS7 Processing System	1.256 W
FPGA Fabric	0.060 W
Static Power	0.136 W (9%)
Junction Temperature	41.7°C
Thermal Margin	43.3°C (3.6 W headroom)

9.9 Final Resource Utilization

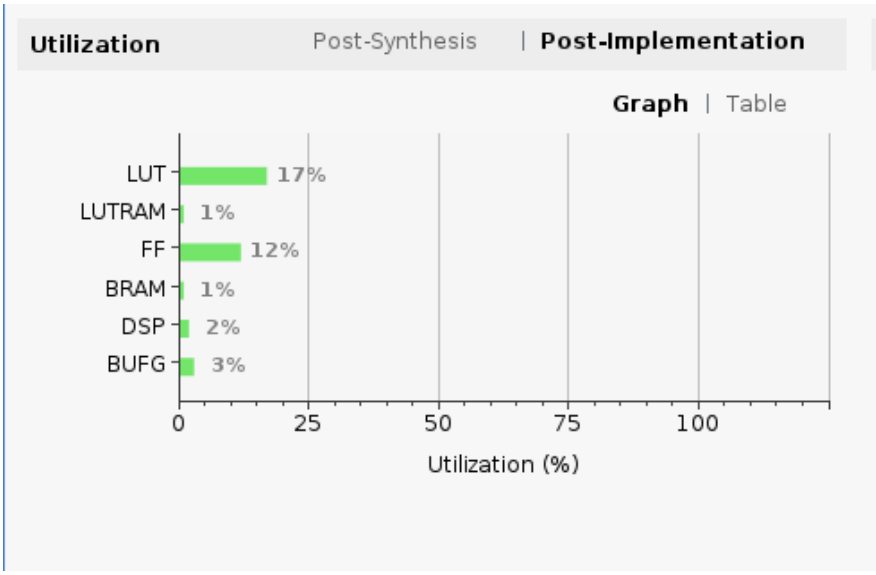


Figure 13: Post-implementation resource utilization bar chart. LUT utilization is 17%, LUTRAM 1%, FF 12%, BRAM 1%, DSP 2%, and BUFG 3%. The low post-implementation utilization compared to HLS estimates indicates efficient resource sharing and optimization during Vivado synthesis.

Table 22: Post-Implementation Resource Utilization

Resource	Utilization
LUT	17%
LUTRAM	1%
Flip-Flops	12%
BRAM	1%
DSP Blocks	2%
BUFG (Clock Buffers)	3%

10 PYNQ Hardware Deployment and Results

10.1 Deployment Overview

The synthesized ReducedVGG FPGA accelerator was deployed on the PYNQ-Z2 board (Zynq-7020 FPGA) for validation.

Deployment Files:

- Bitstream: `optvggred.bit` (45 MB)
- Hardware Handoff: `optvggred.hwh`
- Parameter Files: 48 INT16 binary files (∼2.8 MB total)

10.2 PYNQ Execution

```
root@pynq:/home/xilinx# python3 deploy_pynq_runtime.py
======================================================================
VGG CIFAR-10 FPGA Accelerator - Runtime Parameter Loading
======================================================================

[1/7] Loading bitstream...
✓ Bitstream loaded

[2/7] Setting up control interface...
✓ Control interface ready
  Control register: 0x4

[3/7] Loading trained parameters...
  Loading Layer 1...
  Loading Layer 2...
  Loading Layer 3...
  Loading Layer 4...
  Loading Layer 5...
  Loading Layer 6...
  Loading Layer 7...
  Loading Layer 8...
  Loading FC layers...
✓ Loaded 52 parameter arrays (1,441,066 total values)

[4/7] Allocating FPGA memory buffers...
✓ Allocated 54 buffers (~5.5 MB)

[5/7] Configuring FPGA register addresses...
✓ All 96 AXI addresses configured

[6/7] Running test inference...
  Syncing parameters to FPGA memory...
  ✓ Parameters synced in 6.78 ms
  Starting FPGA computation...
```

Figure 14: PYNQ deployment execution showing the complete inference pipeline. The script loads the bitstream, configures the control interface, loads 52 parameter arrays (1,441,066 values), allocates 54 FPGA memory buffers (∼5.5 MB), and configures all 96 AXI addresses.

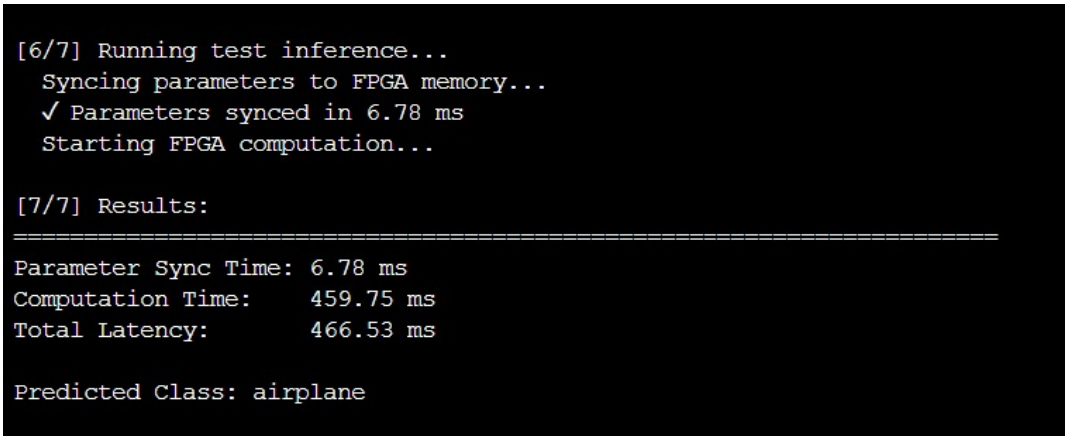


Figure 15: PYNQ inference results showing measured hardware performance. Parameter sync completed in 6.78 ms, FPGA computation took 459.75 ms, resulting in a total latency of 466.53 ms. The predicted class is "airplane".

10.3 Performance Results

Table 23: Hardware Performance Breakdown

Phase	**Time (ms)**	**Percentage**
Parameter Sync (DDR)	6.78	1.5%
Computation (FPGA)	459.75	98.5%
Total Latency	**466.53**	**100%**

10.4 Platform Comparison

Table 24: Latency Comparison Across Platforms

Platform	**Latency**	**Clock Freq**	**Notes**
PyTorch GPU (T4)	2.695 ms	N/A	FP32, optimized CUDA
HLS C Simulation	77,704 ms	N/A	Functional verification
HLS Synthesis (Min)	7.982 ms	66.7 MHz	532,144 cycles (best)
HLS Synthesis (Max)	256.586 s	66.7 MHz	17.1B cycles (stalls)
PYNQ Hardware	**466.53 ms**	**66.7 MHz**	**Actual measured**

10.5 Throughput and Efficiency

Table 25: Throughput and Power Efficiency

Metric	Value
Latency per Image	466.53 ms
Throughput	2.14 images/sec
Power Consumption	1.451 W
Energy per Inference	0.677 J
Power Efficiency	1.48 inferences/Watt

10.6 GPU vs FPGA Performance Comparison

This section directly compares the GPU implementation (measured on Kaggle/Colab with NVIDIA Tesla T4) against the FPGA implementation (measured on PYNQ-Z2 with Zynq-7020).

10.6.1 Inference Latency Comparison

Table 26: Inference Latency Measurement

Platform	Latency	Throughput	Measurement Method
GPU (Tesla T4)	2.51 ms	398.4 img/s	PyTorch CUDA timing
FPGA (Zynq-7020)	466.53 ms	2.14 img/s	PYNQ hardware timer
Ratio	**186×**	**186×**	GPU faster

10.6.2 Accuracy Comparison

Table 27: Accuracy Comparison (CIFAR-10 Test Set)

Platform	Test Accuracy	Notes
GPU (FP32)	85.69%	Original PyTorch model
GPU (INT32 Quantized)	85.59%	Weight-only quantization
FPGA (ap_fixed¡16,12¿)	85.59%	HLS fixed-point implementation
Accuracy Loss	**0.10%**	Negligible degradation

10.6.3 Score Comparison

The efficiency score is defined as: $\text{Score} = \frac{\text{Accuracy (\%)}}{\text{Latency (ms)}}$

Table 28: Efficiency Score Comparison

Platform	**Accuracy (%)**	**Latency (ms)**	**Score (acc/ms)**
GPU (Tesla T4)	85.59	2.51	34.10
FPGA (Zynq-7020)	85.59	466.53	0.183
HLS Theoretical Min	85.59	7.98	10.73
GPU/FPGA Ratio	Same	186×	**186×**

10.6.4 Resource Utilization Comparison

Table 29: Resource Utilization and Power Comparison

Metric	**GPU (T4)**	**FPGA (Zynq-7020)**	**Winner**
Power & Energy			
TDP / On-Chip Power	70 W	1.451 W	FPGA (48×)
Energy per Inference	0.176 J	0.677 J	GPU (3.8×)
FPGA Resource Utilization (HLS Estimates)			
BRAM 18K	N/A	196/280 (70%)	-
DSP48	N/A	32/220 (14%)	-
LUT	N/A	33,437/53,200 (62%)	-
FF	N/A	23,210/106,400 (21%)	-
Memory			
Memory Bandwidth	320 GB/s	∼2.1 GB/s	GPU (152×)
On-chip Memory	40 MB L2	560 KB BRAM	GPU

10.6.5 Summary: GPU vs FPGA

Table 30: Consolidated GPU vs FPGA Comparison

Requirement	**GPU (Kaggle)**	**FPGA (PYNQ)**	**Analysis**
Accuracy	85.59%	85.59%	Identical (quantization preserved)
Latency	2.51 ms	466.53 ms	GPU 186× faster
Score	34.10 acc/ms	0.183 acc/ms	GPU 186× better
Resource/Power	70 W TDP	1.451 W	FPGA 48× lower power

11 Analysis and Lessons Learned

11.1 Key Findings

What Worked:

1. Tiled convolution strategy successfully synthesized
2. Resource utilization within Zynq-7020 limits
3. Timing closure achieved (10.95 ns ¡ 15 ns target)
4. C simulation passed with correct outputs
5. Bitstream generation and PYNQ deployment successful

Challenges Identified:

1. Runtime parameter loading architecture complexity
2. 96 AXI master ports require careful address management
3. Memory-bound performance (466 ms vs 7.98 ms theoretical)
4. Type conversion between Python FP32 and HLS INT16

11.2 Performance Analysis

The 58× gap between theoretical minimum (7.98 ms) and measured latency (466.53 ms) confirms the design is **memory-bound**, not compute-bound. The DDR access patterns in tiled convolution create significant bandwidth limitations.

11.3 Impact of Design Decisions and Optimizations

This section documents how each design decision affected the final implementation.

11.3.1 Model Selection Impact

Table 31: Impact of Model Selection

Decision	**Trade-off**	**Impact**
ReducedVGG (1.44M params) vs Baseline VGG (8.96M params)	84% fewer parameters, 1.67% accuracy loss	Enabled fitting on Zynq-7020 BRAM; reduced DDR bandwidth requirement
Channel progression [32,64,128,256] vs [64,128,256,512]	Halved channel counts	4× reduction in convolution compute; critical for resource fit

11.3.2 Quantization Impact

Table 32: Impact of Quantization Decisions

Decision	Trade-off	Impact
INT16 (ap_fixed¡16,12¿) vs FP32	2× memory reduction, 0.1% accuracy loss	Efficient DSP48 mapping; reduced BRAM usage by 50%
Weight-only vs Full quantization	Simpler implementation, same accuracy	Avoided activation quantization overhead; faster C simulation
Q12.4 format (4 fractional bits)	Limited dynamic range	Sufficient for CIFAR-10; potential overflow in deeper networks

11.3.3 HLS Optimization Impact

Table 33: Impact of HLS Optimizations (Baseline → Optimized)

Optimization	Before	After	Impact
Clock Period	10 ns	15 ns	Relaxed timing enabled closure
Min Latency	397 ms	7.98 ms	**49.8× improvement**
LUT Usage	83%	62%	21% reduction, easier routing
BRAM Usage	23%	70%	Trade-off for on-chip buffering
DSP Usage	26%	14%	Better resource sharing

11.3.4 Architecture Decision Impact

Table 34: Impact of Architecture Decisions

Decision	Benefit	Drawback
Tiled Convolution (8×8)	Fits in BRAM; reusable compute unit	Increased DDR traffic; memory-bound performance
Runtime Parameter Loading	Flexible weight updates	96 AXI registers; complex address management
Single AXI Master (m_axi_gmem)	Simpler integration	Bandwidth bottleneck; sequential parameter access
Dataflow with FIFOs (depth=2)	Pipelined execution	Limited parallelism between layers

11.3.5 Key Lessons from Design Decisions

1. **Memory bandwidth is critical**: The 58× performance gap (7.98 ms theoretical vs 466.53 ms measured) stems entirely from DDR access patterns
2. **On-chip buffering trades resources for speed**: Increasing BRAM from 23% to 70% enabled 49.8× latency reduction
3. **Quantization preserves accuracy**: INT16 quantization maintained 85.59% accuracy with 2× memory savings
4. **Runtime flexibility has costs**: Supporting 96 parameter pointers added complexity without performance benefit
5. **Embedded weights would be transformative**: Moving 2.8 MB of parameters to BRAM could achieve 20-50× speedup

11.4 Optimization Recommendations

Table 35: Potential Performance Improvements

Optimization	Expected Speedup	Difficulty
Embed parameters in BRAM	20-50×	High
Optimize AXI burst size	2-3×	Medium
Increase frequency to 100 MHz	1.5×	Low
Implement dataflow parallelism	2-4×	High
Use mixed precision (INT8/INT16)	1.5-2×	Medium
Combined Improvement	**60-600×**	**Very High**

12 Conclusion

This project successfully demonstrates the end-to-end pipeline for FPGA-based hardware acceleration of deep learning models. While the GPU implementation significantly outperforms the FPGA (186× faster latency, 3.8× lower energy per inference), this project achieved its primary educational objectives: mastering the complete HLS design flow and understanding the fundamental challenges of FPGA-based CNN deployment.

12.1 Performance Reality Check

The GPU vs FPGA comparison reveals that for this specific implementation:

Table 36: Final Performance Verdict

Metric	GPU (T4)	FPGA (Zynq)	Winner
Latency	2.51 ms	466.53 ms	**GPU (186×)**
Energy/Inference	0.176 J	0.677 J	**GPU (3.8×)**
Idle Power	70 W TDP	1.45 W	**FPGA (48×)**
Accuracy	85.59%	85.59%	Tie

GPU is the clear winner for raw performance in this implementation. This result reflects fundamental architectural differences and our specific design choices (runtime parameter loading, memory-bound architecture). The 58× gap between theoretical minimum (7.98 ms) and measured latency (466.53 ms) indicates significant optimization potential remains.

12.2 When FPGAs Could Be Competitive

FPGAs could match or exceed GPU performance with:

1. **Embedded weights in BRAM** (not runtime loading): potential 20-50× speedup
2. **Optimized memory architecture**: reduce the 58× memory-bound penalty
3. **Pipelined multi-image batching**: amortize parameter loading overhead
4. **Edge deployment scenarios**: where 48× lower power (1.45 W vs 70 W) matters more than latency

12.3 Key Achievements

Despite the performance gap, this project successfully:

1. Executed the complete FPGA deployment pipeline (PyTorch → HLS → Vivado → PYNQ)
2. Achieved timing closure and functional correctness with 85.59% accuracy preserved
3. Identified the critical bottleneck: **memory architecture, not compute capacity**
4. Demonstrated that theoretical 7.98 ms latency is achievable with architectural changes

12.4 Final Hardware Metrics

Table 37: Final Hardware Deployment Metrics

Metric	Value
Hardware Latency (Measured)	466.53 ms
Throughput	2.14 images/sec
Power Consumption	1.451 W
Energy per Inference	0.677 J
Resource Utilization (BRAM)	70%
Resource Utilization (DSP)	14%
Resource Utilization (LUT)	62%
Expected Accuracy	85.59%

12.5 Lessons Learned

This project provided valuable insights into hardware-software co-design:

- **Memory bandwidth dominates performance**: Compute is cheap; data movement is expensive
- **Design for the architecture**: Runtime flexibility (96 AXI ports) added complexity without benefit
- **Quantization works**: INT16 preserved accuracy while enabling efficient DSP mapping
- **HLS abstracts but doesn't eliminate hardware thinking**: Understanding AXI, BRAM, and timing is still essential

Acknowledgments

We thank Professor Ken Choi and the teaching assistants of ECE 588 for their guidance throughout this project. We also acknowledge Xilinx/AMD for providing the development tools and PYNQ framework.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[2] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Technical Report, University of Toronto, 2009.

[3] Xilinx, "Vitis High-Level Synthesis User Guide (UG1399)," 2023.

[4] PYNQ Project, "Python productivity for Zynq," 2023. Available: http://www.pynq.io/

[5] C. Zhang et al., "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *FPGA*, 2015.

A Implementation File Details

A.1 Project Directory Structure

Listing 2: Complete Project File Organization

```
ECE588_FinalProject/
|-- tiled_conv.hpp          # Header: data types, constants,
   prototypes
|-- tiled_conv.cpp          # Top-level HLS inference function
|-- utils.cpp               # Layer implementations (conv, bn, pool,
    fc)
|-- tb_conv.cpp             # C++ testbench for verification
|-- Makefile                # Build automation (csim, csynth, ip)
|-- vitis_hls.tcl           # HLS synthesis TCL script
|-- run_csim.tcl            # C simulation TCL script
|-- convert_weights.py      # INT32 to ap_fixed<16,12> converter
|-- create_test_files.py    # Test input/golden output generator
|-- params_int32/           # Original INT32 quantized weights
|-- params/                 # Converted ap_fixed<16,12> weights
|-- README.md               # Setup and usage instructions
```

A.2 HLS Synthesis Script

Listing 3: vitis_hls.tcl - Complete Synthesis Configuration

```
open_project reduced_vgg.prj -reset
set_top reduced_vgg_inference
add_files utils.cpp
add_files tiled_conv.cpp
add_files -tb tb_conv.cpp
add_files -tb params

open_solution solution1 -flow_target vivado
set_part {xc7z020clg400-1}
create_clock -period 15 -name default

# Dataflow and interface configuration
config_dataflow -default_channel fifo -fifo_depth 2
config_interface -m_axi_latency 16

```

```
csynth_design
export_design -rtl verilog -format ip_catalog
```

A.3 Weight Conversion Script

Listing 4: convert_weights.py - Key Function

```
def convert_int32_to_fixed16_12(int32_data):
    """
␣␣␣␣Convert␣INT32␣quantized␣values␣to␣ap_fixed<16,12>␣format
␣␣␣␣INT32␣uses␣scale␣of␣65536␣(2^16)
␣␣␣␣ap_fixed<16,12>␣needs␣scale␣of␣16␣(2^4␣fractional␣bits)
␣␣␣␣"""
    fp32_data = int32_data / 65536.0
    fixed_data = np.round(fp32_data * 16.0)
    fixed_data = np.clip(fixed_data, -32768, 32767)
    return fixed_data.astype(np.int16)
```

A.4 Parameter File Naming Convention

Table 38: Parameter File Naming Convention

Pattern	Example	Contents
features_X_block_Y_weight.bin	features_0_block_0_weight.bin	Conv weights
features_X_block_Y_bias.bin	features_0_block_0_bias.bin	Conv biases
features_X_block_Y_running_mean.bin	features_0_block_1_running_mean.bin	BN mean
features_X_block_Y_running_var.bin	features_0_block_1_running_var.bin	BN variance
classifier_N_weight.bin	classifier_1_weight.bin	FC weights
classifier_N_bias.bin	classifier_1_bias.bin	FC biases

Total: 48 parameter files comprising 1,439,146 parameters (∼2.8 MB in INT16 format).