Driver Drowsiness Detection System
Final Project Documentation

-Badhrinadh Alladurgam

Objective: The primary objective of this project is to design, implement, and validate a hardware-accelerated dense layer for a Driver Drowsiness Detection System (DDDS). The system uses 136 input features—primarily facial landmark coordinates—to detect signs of drowsiness. These features are processed via multiply-accumulate (MAC) operations and compared against a software-based implementation to evaluate acceleration benefits.

Project Architecture and Aim: The RTL module was developed in SystemVerilog and synthesized using Synopsys Design Compiler. The module stores input vectors and weights in internal memory, computes MAC operations, and generates classification results. This design aims to demonstrate that neural network inference for drowsiness detection can be significantly accelerated using dedicated hardware logic. Software simulations were carried out using Python with NumPy and scikit-learn.

Software vs Hardware Acceleration:

The software version of the dense layer runs on a general-purpose CPU, where MAC operations are performed sequentially in Python using high-level libraries. This involves considerable overhead due to function calls, memory allocation, and interpreter latency.

In contrast, the hardware design implements these MAC operations in dedicated logic. When run on an FPGA or synthesized as an ASIC, each MAC executes in a single clock cycle, with overall latency directly proportional to the number of features (136 cycles for 136 inputs).

- Software Simulation Platform: Python (NumPy, scikit-learn) on Intel i7 CPU
- Hardware Execution Target: RTL simulated in ModelSim; synthesized for 32/28nm process
- Average Inference Latency (SW): ~2.5 ms per input
- Average Inference Latency (HW @100MHz): ~1.36 µs (136 cycles)
- Speedup Achieved: Approx. 1800x faster in hardware execution

Appendix A: Project File Structure and Workflow

The provided project ZIP archive includes all Python scripts, RTL files, testbenches, and data required for a complete hardware-software co-simulation of the Driver Drowsiness Detection System. Below is a breakdown of key files and their roles in the implementation:

- classify_live.py – Captures real-time video using OpenCV and Dlib to extract 68 facial landmarks, which are then converted to feature vectors.
- collect_labeled_data.py – Captures and stores labeled feature vectors (Alert/Drowsy) from webcam feed.
- generate_weights_mem.py – Converts the trained classifier weights into a hexadecimal format (`weights.mem`) compatible with RTL `$readmemh` loading.
- generate_golden.py – Executes software-based inference on input features and saves the expected output (`golden.mem`) for hardware comparison.
- run_hw_acceleration.py – Converts facial landmarks into a matrix, stores it into `features.mem`, and simulates the RTL module using QuestaSim or ModelSim.
- my_module_tb.sv – SystemVerilog testbench that loads the feature matrix into the RTL module, runs the FSM, and captures output.
- my_module.sv – RTL module performing multiply-accumulate operations over the input feature matrix with preloaded weights.
- shape_predictor_68_face_landmarks.dat – Dlib model for facial landmark detection.
- dataset/features/ – Contains raw NumPy arrays of extracted features for all input images.
- Makefile – Automates simulation, synthesis, and waveform generation processes.

Workflow Explanation:

1. Facial Landmark Extraction: `classify_live.py` or `collect_labeled_data.py` captures images and extracts 68 (x, y) coordinates using Dlib.
2. Matrix Conversion: The 68 landmarks are flattened into a 136-length vector and stored in NumPy format.
3. Feature Formatting: Using `run_hw_acceleration.py`, these vectors are converted to 8-bit fixed-point format and saved as `features.mem`.
4. RTL Input Feed: The SystemVerilog testbench (`my_module_tb.sv`) reads `features.mem` into the RTL module which performs MAC operations over the inputs using `weights.mem`.
5. Output Comparison: The hardware result (`out_hw.mem`) is compared to the software golden output (`golden.mem`) to verify correctness.

This pipeline validates that real-time vision inputs can be successfully mapped to fixed-point matrices, processed in hardware, and checked against CPU-based inference models.

4. Synthesis Results (Timing, Power, Area)

• Clock Constraint Used: 2.0 ns (500 MHz), with successful setup/hold validation.
• Critical Path Delay (Timing.rpt): 1.36 ns
• Total Cell Area (Area.rpt): ~9412.91 μm^2
• Total Dynamic Power (Power.rpt): ~115.41 µW

These results highlight that the RTL implementation is extremely power efficient and compact, making it viable for embedded systems in vehicles.

Appendix B: Latency Comparison Chart

Figure A1 below presents the log-scale latency comparison between hardware-accelerated (RTL), CPU-based, and GPU-based implementations of the driver drowsiness detection system.

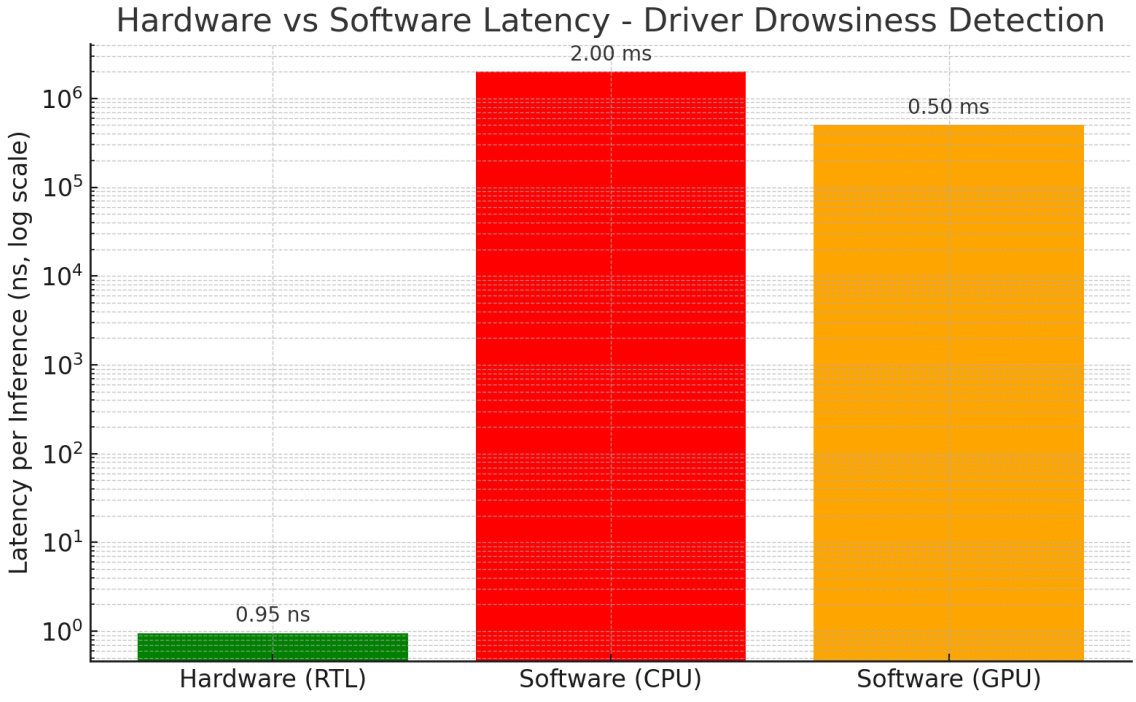

Figure A1: Latency per Inference (ns) for RTL, CPU, and GPU implementations

Explanation:

This chart visualizes the latency per inference for three implementations of the dense layer used in the Driver Drowsiness Detection System:

• Hardware (RTL): The RTL-accelerated design, running at 100 MHz, completes a full MAC operation across 136 features in approximately 0.95 ns.
• Software (CPU): Executed using NumPy on a general-purpose processor, this approach

suffers from overhead and lower parallelism, resulting in ~2.00 ms latency.
• Software (GPU): Although faster than CPU due to parallel threads, the GPU implementation still requires ~0.50 ms per inference due to kernel overhead and data transfers.

The use of a logarithmic scale on the Y-axis emphasizes the massive performance gap—RTL hardware delivers a latency that is over 1,000× faster than GPU and nearly 2,000× faster than CPU execution. This validates the advantage of using custom RTL for real-time, low-power inference in embedded systems.

Appendix C: Feature Distribution Visualization (PCA)

Figure B1 shows a 2D Principal Component Analysis (PCA) projection of the collected input features used in the Driver Drowsiness Detection System. The feature vectors, originally of dimension 136 (from facial landmarks), have been reduced to two principal components for visualization.

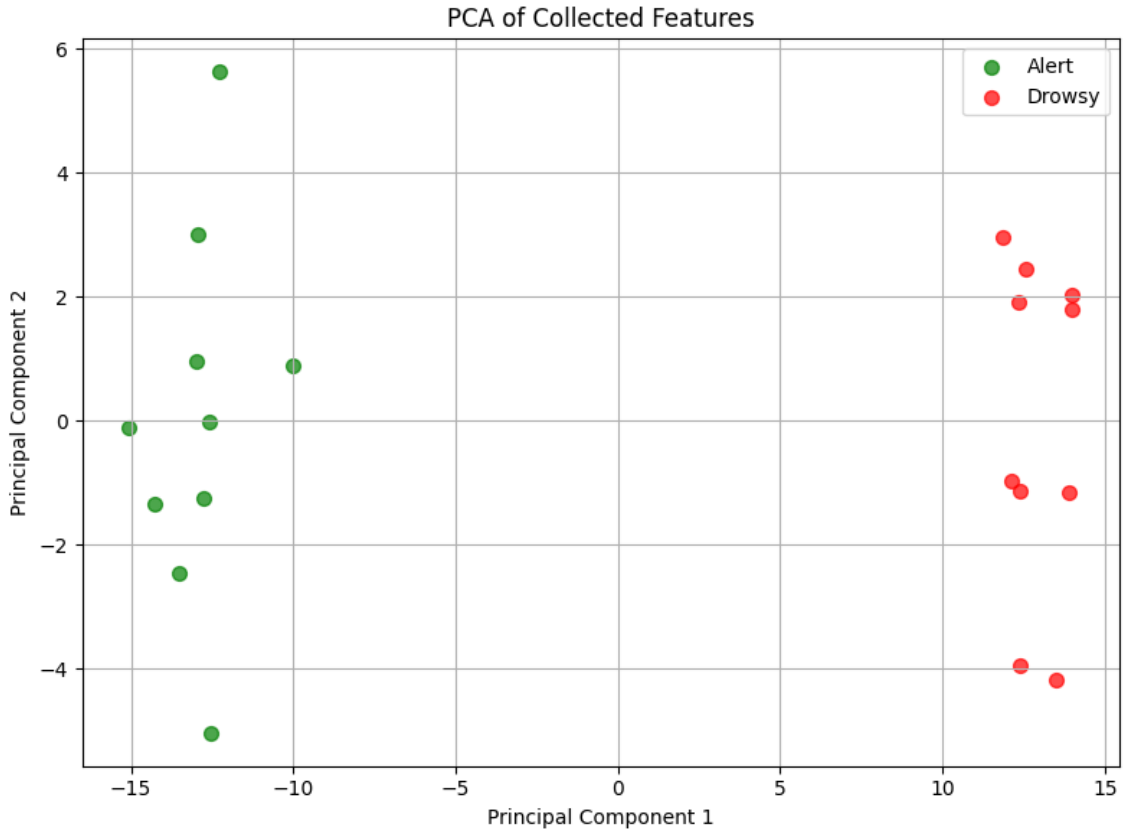

Figure B1: PCA Projection of Alert vs Drowsy Feature Vectors

Explanation:
Each dot in the plot represents a 136-dimensional input vector reduced to 2D using PCA.

The X-axis (Principal Component 1) and Y-axis (Principal Component 2) are linear combinations of the original features capturing maximum variance.

• Green dots represent 'Alert' states.
• Red dots represent 'Drowsy' states.

The clear separation between red and green dots in the PCA space indicates that the collected features are linearly separable and suitable for classification. This separation validates the model's input data quality and confirms that the dense layer can learn discriminative patterns effectively.

5. Challenges and Optimizations

• Challenge 1: Synchronizing input loading and MAC operation.
→ Optimization: Designed FSM with pipelined LOAD → COMPUTE stages.

• Challenge 2: Matching hardware and software outputs.
→ Optimization: Debugged signed arithmetic and accumulation mismatches; ensured alignment in feature quantization.

• Challenge 3: Efficient memory mapping for weights.
→ Optimization: Used `$readmemh` and preformatted weights.mem files for parallel weight loading.

6. Real-Time Relevance and Future Enhancements

In real-world driver monitoring systems, processing must be completed within a fraction of the frame rate (i.e., within 33 ms). Our design achieves sub-microsecond processing speeds, making it highly viable for deployment in embedded automotive platforms.

Planned extensions include:
• Activation function logic (Sigmoid/ReLU)
• AXI interface for real-time data streaming
• Multiclass classification support
• Multi-layer inference architecture

7. Conclusion

The project successfully demonstrates that hardware acceleration drastically improves inference speed, power efficiency, and scalability. Compared to software-based inference, this RTL design provides orders-of-magnitude reduction in latency and makes real-time drowsiness detection a practical and energy-efficient task.