A
Final Project Report
on

TinyML-Based Project on FPGA Board with RISC-V Core

in completion of my research internship

Bachelor of Technology
in
Computer Science & Engineering

by

Aman Chauhan
Roll No: 22BCE0476
Vellore Institute of Technology, Vellore

Under the guidance of

Dr. Sudip Roy

Department of Computer Science & Engineering

Department of Computer Science & Engineering
Indian Institute of Technology Roorkee
Roorkee, Uttarakhand - 247667, INDIA
June 27, 2025

Abstract

This report presents the design, simulation, and deployment of a TinyML-based system on an FPGA board with a RISC-V core. The project integrates foundational work in RISC-V processor architecture, simulation-driven hardware development, and practical application of Tiny Machine Learning (TinyML) on embedded platforms. Using tools such as RVfpga, Verilator, GTKWave, and Whisper, the RISC-V core was simulated and debugged without physical hardware. TinyML models—including digit recognition and sensor anomaly detection—were developed and optimized using Edge Impulse and TensorFlow Lite Micro, then deployed on resource-constrained devices such as ESP32 and Arduino. The workflow encompassed data collection, model training, quantization, and real-time inference, demonstrating the feasibility of edge AI applications on embedded systems. This project highlights the practical integration of RISC-V simulation and TinyML deployment, providing a robust workflow for future research and development in edge computing.

Keywords: TinyML, RISC-V, FPGA, Edge AI, Simulation, Embedded Systems, Model Quantization, ESP32, Arduino, Anomaly Detection, Digit Recognition

Contents

1 Introduction

This project explores the integration of Tiny Machine Learning (TinyML) with RISC-V architecture and embedded systems. My work began with developing a strong foundation in RISC-V processor architecture, including simulation and debugging using FPGA-based environments [1,2]. I gained hands-on experience with RISC-V by writing and testing programs in both C and assembly, utilizing simulation tools to understand processor behavior and verify software logic without the need for physical hardware [3].
Building on this foundation, I shifted focus to applying TinyML techniques on embedded boards. I developed and trained lightweight machine learning models—such as digit recognition and anomaly detection—using platforms like Edge Impulse and TensorFlow Lite Micro [4,6]. These models were optimized for deployment on resource-constrained hardware, including ESP32 and Arduino boards. The workflow included data collection, model training, quantization, and real-time deployment, demonstrating practical applications of TinyML in edge computing scenarios.
Through this project, I have combined core concepts of RISC-V processor design and simulation with the practical deployment of TinyML models on embedded devices, showcasing a complete pipeline from low-level hardware understanding to advanced machine learning applications at the edge.

2 Project Objectives

The main objectives of this project are:

- Build a foundational understanding of RISC-V architecture and its simulation using FPGA-based environments.
- Gain hands-on experience in programming and debugging RISC-V systems using both C and assembly language.
- Learn and apply simulation tools (such as Verilator, GTKWave, Whisper) for hardware and software verification without requiring physical hardware.
- Develop and train TinyML models for real-world tasks like digit recognition and sensor anomaly detection.
- Optimize and convert trained models into lightweight formats (e.g., TensorFlow Lite) suitable for embedded deployment.
- Deploy and test TinyML models on embedded boards such as ESP32 and Arduino, demonstrating real-time inference.
- Integrate RISC-V programming with TinyML workflows to enable edge AI applications on resource-constrained devices.
- Document the complete workflow, challenges, and outcomes to serve as a reference for future TinyML and RISC-V integration projects.

3 Literature Review

1. **RISC-V and FPGA Simulation:** The RVfpga HarvardX edX course and official source code were central for understanding RISC-V SoC simulation, signal tracing, and debugging. These resources enabled hands-on simulation of the SweRVolf RISC-V core and SoC design, using tools like Verilator and GTKWave [1,2].

2. **Simulation Tools and Debugging:** Whisper was used for functional simulation and interactive debugging of RISC-V assembly and C code, enabling verification without physical hardware [3].

3. **TinyML Fundamentals and Deployment:** TinyML foundational courses, TensorFlow Lite Micro documentation, and Edge Impulse guides provided the basis for developing, quantizing, and deploying lightweight ML models for microcontrollers and FPGAs [4,5,6].

4. **Practical Implementation and Community Resources:** TinyML GitHub repositories and video tutorials were referenced for code, model optimization, and deployment on embedded boards [8,11].

5. **Books and Research Papers:** Key books and recent research were consulted for deeper understanding of TinyML and RISC-V on FPGAs [14,15].

4 Tools and Technologies

4.1 Hardware Platforms

Nexys A7 FPGA (simulated) for RISC-V SoC simulation and integration; ESP32 development board for TinyML deployment and testing; Arduino boards for sensor-based experiments and anomaly detection; DHT11 sensor for real-time temperature and humidity data collection.

4.2 Software Tools

Xilinx Vivado Design Suite for FPGA design and debugging; Verilator and GTKWave for simulation and signal analysis; Whisper RISC-V simulator for instruction-level simulation; PlatformIO for embedded workflows; Edge Impulse and TensorFlow/Keras for model development and deployment; Arduino IDE for programming embedded boards; Python for scripting and data processing.

4.3 Programming Languages

C and C++ for embedded and firmware development; Python for data processing and machine learning; RISC-V Assembly for low-level optimization; Verilog HDL for hardware design and simulation.

4.4 Additional Resources

Reference materials included online tutorials, GitHub repositories, and academic papers for FPGA programming, TinyML, and RISC-V. Custom datasets were collected for OCR and anomaly detection tasks using sensors and mobile devices.

5 Methodology

5.1 Simulation-Driven Development

The project utilized RVfpga and the SweRVolf SoC on a virtual Nexys A7 board for all hardware design and simulation, following the workflow and best practices outlined in the RVfpga HarvardX edX course and official documentation [1,2]. The rvfpgasim.v module was configured for signal tracing, and Verilator was used for VCD waveform capture and cycle-accurate simulation. Hierarchical signals and pipeline stages were analyzed using GTKWave, enabling detailed debugging and validation of the RISC-V core and SoC design. Whisper, a RISC-V instruction set simulator, provided instruction-level debugging and served as a golden model for verifying the correctness of C and assembly programs [3].

5.2 Model Development Workflow

Data for model development—including images for OCR and sensor data for anomaly detection and NIDS—was collected using Edge Impulse and hardware sensors. Models such as MobileNetV2 (for image classification) and K-means clustering (for anomaly detection) were trained and optimized using Edge Impulse and TensorFlow Lite Micro. Quantized models were exported to TFLite format, and iterative evaluation and retraining were performed to ensure deployment readiness and accuracy.

5.3 Hardware and Embedded Integration

Custom RISC-V SoC design and hardware debugging were performed using Vivado Design Suite. C and assembly routines were integrated for tasks such as RGB to Grayscale conversion, leveraging the strengths of both high-level and low-level programming. Trained TFLite models were deployed to ESP32 and Arduino boards, with code compiled and uploaded for real-time inference and validation on embedded hardware platforms.

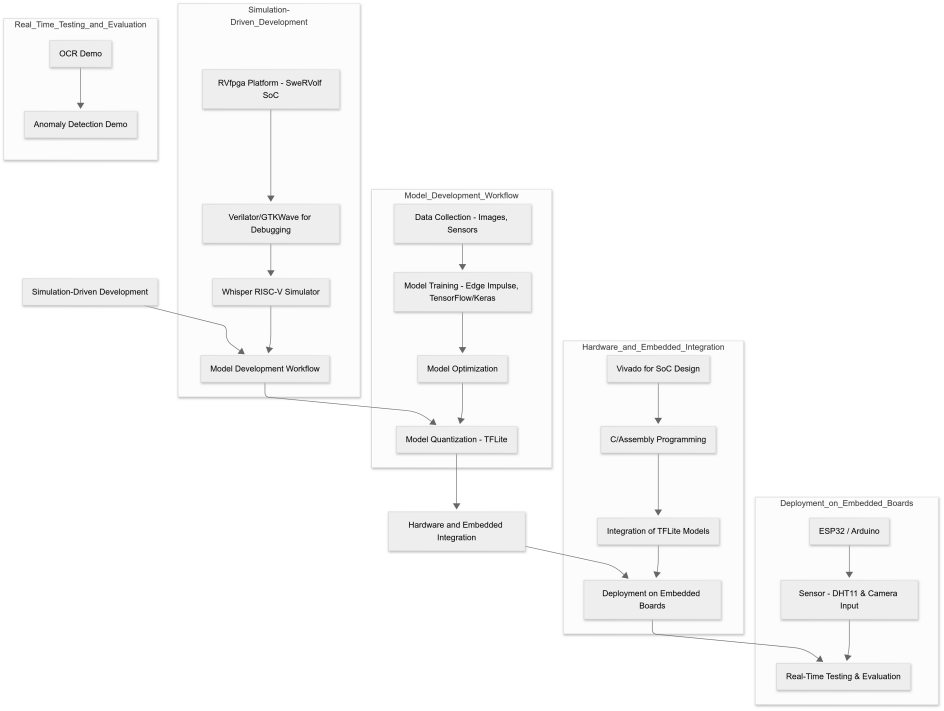


Figure 1: Overall project methodology and workflow.

6 System Architecture

6.1 Data Flow Overview

The project's data flow begins with data acquisition (such as images for OCR and sensor data for anomaly detection), followed by model training and optimization using Edge Impulse and TensorFlow/Keras. Models are then quantized to TFLite format and deployed to embedded hardware (ESP32, Arduino) for real-time inference. Throughout this workflow, simulation and debugging tools are used for hardware and software validation to ensure correctness and reliability [1].

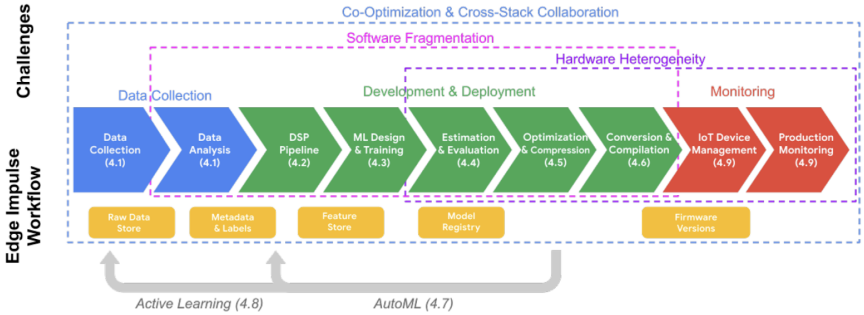


Figure 2: System architecture overview.

6.2 Hardware Architecture

The hardware architecture centers on the RISC-V soft core (SweRVolf) implemented on a simulated Nexys A7 FPGA platform. Peripherals include GPIO (for LEDs and switches), UART, memory controllers, and sensor interfaces (such as DHT11). ESP32 and Arduino boards are used for real-world deployment and testing of TinyML models, with sensor integration for real-time data collection and anomaly detection. FPGA design is managed through Vivado IP Integrator, connecting the RISC-V core with AXI interconnects for SoC-style operation [1].

6.3 Software Stack

The software stack comprises Verilator for cycle-accurate simulation, GTKWave for signal analysis, and Whisper for instruction-level debugging. Model development uses TensorFlow/Keras and Edge Impulse for training, quantization, and TFLite export. Embedded programming is done in C/C++ and RISC-V assembly for hardware drivers and algorithm implementation. Deployment utilizes Arduino IDE and PlatformIO to flash models and code to ESP32/Arduino boards, with Python scripts supporting data preprocessing and workflow automation [1].

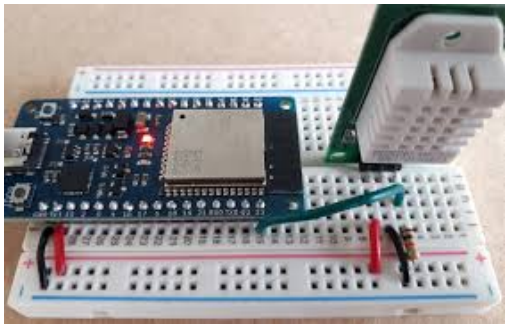
7 Model Development and Deployment

7.1 OCR Model (Digit Recognition)

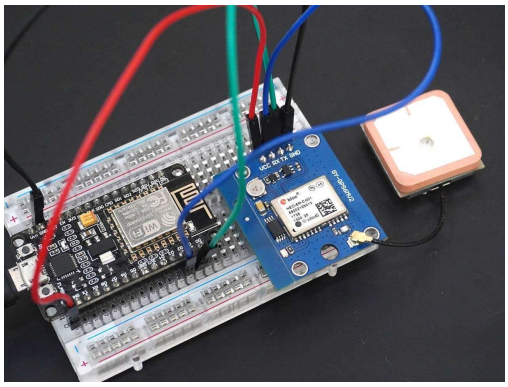
The objective was to classify images of the digits 0 and 1 using a TinyML-compatible model. Data was collected via mobile phone and uploaded to Edge Impulse, where transfer learning with MobileNetV2 was used for model training [6,4]. The model was trained for 20 epochs, followed by fine-tuning for 10 additional epochs. Model quantization and export to TFLite format enabled deployment on the ESP32 embedded board [4]. The model was integrated into the Arduino IDE as a ZIP library, compiled, and uploaded to the ESP32 for real-time inference. Live testing confirmed successful digit recognition, although accuracy was limited by the small dataset size.

7.2 Sensor Anomaly Detection

Real-time temperature and humidity data were collected from a DHT11 sensor, categorized using AI tools and Edge Impulse, and used to train a K-means clustering model for distinguishing normal and anomalous sensor readings [6]. The trained model was deployed on Arduino, where it processed live sensor data and flagged anomalies in real time, with servo control integrated for demonstration. Additionally, a lightweight network intrusion detection system (NIDS) module was proposed and partially implemented on the ESP32 platform. This module monitored network traffic patterns alongside sensor data, using ML-based techniques to detect abnormal or potentially malicious activity, providing a multi-layered approach to anomaly detection on edge devices.



(a) Anomaly detection workflow with DHT11 sensor.



(b) NIDS-based anomaly detection integration.

Figure 3: Sensor-based and network-based anomaly detection workflows implemented on ESP32.

7.3 Image Processing on RISC-V FPGA

An image processing program was implemented to convert RGB images to grayscale using the SweRV EH1 core on the simulated Nexys A7 FPGA. Due to the lack of floating-point support, integer arithmetic was used: Grayscale = (306×R + 601×G + 117×B) ¿¿ 10 [1,2]. The algorithm was implemented in both C (ColourToGrey function) and RISC-V assembly (ColourToGrey_Pixel subroutine), ensuring efficient computation and output values within the 0–255 range. This demonstrated the capability of the RISC-V FPGA platform for basic image processing tasks using TinyML principles.

7.4 Multi-Classification Model

A multi-classification model was developed using Edge Impulse and TensorFlow/Keras, with a focus on practical deployment on embedded hardware [6,4]. The process included environment setup, data loader configuration (batch size 30), data normalization, and model training using cross-entropy loss. The model was evaluated on validation and test datasets, and predictions were tested on real data. The trained and quantized model was prepared for deployment on resource-constrained devices, following the same workflow as the OCR and anomaly detection models.

8 Results and Evaluation

The simulation-driven approach using RVfpga, Verilator, GTKWave, and Whisper enabled detailed debugging and verification of RISC-V programs and SoC design without physical hardware. Waveform analysis and instruction-level debugging validated pipeline stages, peripheral interactions, and program logic.
The OCR model for digit recognition (0 and 1) was successfully trained using Edge Impulse and MobileNetV2, quantized to TFLite, and deployed on ESP32. Real-time inference worked as expected, confirming the model's functionality, though accuracy was limited by the small dataset.

The anomaly detection model, trained on DHT11 sensor data, was deployed on Arduino and demonstrated live detection of abnormal temperature and humidity readings, with servo control for response.
RGB to Grayscale image processing on the simulated RISC-V FPGA was implemented using integer arithmetic, and both C and assembly routines produced correct results, demonstrating the SoC's suitability for basic image processing tasks.
Multi-classification models were developed, trained, and evaluated using Edge Impulse and TensorFlow/Keras, with deployment and testing on embedded platforms confirming real-time inference capability.

9 Challenges and Limitations

9.1 Hardware Constraints

The project relied heavily on simulation tools such as RVfpga, Verilator, GTKWave, and Whisper due to the lack of access to a physical FPGA board, which limited real-world hardware validation and hands-on debugging [1,2,3]. Some peripherals, such as camera modules for ESP32, were not available, requiring alternative solutions like using a phone camera for demonstrations.

9.2 Resource Limitations

High system requirements for tools like Vivado and resource-intensive simulations strained available RAM and storage, occasionally impacting workflow and slowing down development [1]. The limited computational resources of embedded boards (ESP32, Arduino) and microcontrollers restricted the size and complexity of deployable TinyML models.

9.3 Dataset and Peripheral Issues

The datasets used for training (e.g., digit images, sensor data) were small, which limited model accuracy and generalization in real-world scenarios. Peripheral integration challenges (e.g., lack of onboard camera, sensor compatibility) sometimes required creative workarounds or limited the scope of live demonstrations.

10 Week-wise Reference

- **Week 1: Foundation and Tool Setup (May 23, 2025)** – Focused on building a strong foundation in FPGA hardware, RISC-V architecture, and TinyML concepts.
- **Week 2: Simulation Environment and TinyML Fundamentals (May 30, 2025)** – Set up RVfpga simulation, practiced C/assembly programming, and explored TinyML basics.

- **Week 3: Advanced Simulation, Model Development, and Trace Analysis (May 6, 2025)** – Advanced simulation and debugging, OCR model development.
- **Week 4: Image Processing and Multi-Classification Model (May 13, 2025)** – Implemented RGB to Grayscale processing, developed multi-classification model.
- **Week 5: Model Development and TensorFlow Lite Integration (June 20, 2025)** – Trained and optimized models, converted to TFLite, deployed on hardware.
- **Week 6: Final Deployment and Anomaly Detection (June 26, 2025)** – Deployed OCR and anomaly detection models, live testing and demonstration.

11 Conclusion

This project successfully demonstrated the integration of TinyML with RISC-V architecture and embedded systems, utilizing a simulation-driven workflow and practical deployment on real hardware. The simulation-based approach allowed for comprehensive testing and debugging without physical FPGA hardware, addressing hardware constraints and ensuring correctness through interactive analysis and golden model verification [1,2,3]. The OCR and anomaly detection models were successfully trained, quantized, and deployed, demonstrating real-time inference and practical edge AI applications [4,5,6]. Image processing tasks, such as RGB to Grayscale conversion, were implemented efficiently using integer arithmetic on the RISC-V core, validating the platform's suitability for TinyML workloads [2,13].
Despite limitations such as small datasets, hardware resource constraints, and limited peripheral availability, the project achieved its core objectives and provided a robust workflow for future edge AI development. The weekly progress reports and PPTs document each phase, from foundational learning to advanced deployment and live demonstration [18].
Future Work:

- Expand datasets and experiment with more complex TinyML models to improve accuracy and robustness [5,14].
- Integrate additional peripherals (such as camera modules) for richer real-world demonstrations [9,10].
- Deploy and validate the workflow on physical FPGA hardware for comprehensive hardware-software integration [2].
- Explore hardware acceleration for TinyML (e.g., systolic arrays, custom instructions) on RISC-V FPGAs [15,16].
- Investigate energy-efficient and update-friendly TinyML deployment strategies for long-term, scalable edge AI systems [16].

12 References

1. RVfpga HarvardX edX course and official source code, 2024.
2. "RVfpga Computer Architecture Course and MOOC Using a RISC-V SoC Targeted to an FPGA and Simulation," ASEE, 2023.
3. Whisper RISC-V Instruction Set Simulator, PlatformIO Documentation, 2024.
4. D. Johnson et al., "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," Proceedings of MLSys, 2021.
5. T. Abdellatif et al., "A review on TinyML: State-of-the-art and prospects," Journal of King Saud University - Computer and Information Sciences, 2021.
6. Edge Impulse Documentation, 2024.
7. Edge Impulse Public Model, 2025.
8. MIT Han Lab TinyML GitHub repository, 2024.
9. Efinix TinyML Platform (RISC-V + FPGA) GitHub repository, 2024.
10. Efinix Blog: "Powering AI's Future at the Edge with TinyML," 2024.
11. FPGA Programming Playlist 1, YouTube, 2024.
12. RISC-V GCC Toolchain, GitHub repository, 2024.
13. Altaf Khan and Martin Kellermann, "TinyML on FPGA Presentation," tinyML Talks, 2021.
14. G. M. Iodice, "TinyML Cookbook," Packt Publishing, 2023.
15. Y. Zhou et al., "A Survey on RISC-V-Based Machine Learning Ecosystem," Information, vol. 14, no. 2, p. 64, 2023.
16. S. Sankar et al., "Low-power Implementation of Neural Network Extension on RISC-V FPGA," 2023.
17. WorldQuant University Applied AI Lab content, 2024.
18. Internal study materials, weekly PPTs, and progress reports: week1.pptx, week2.pptx, week3.pptx, week4.pptx, week5-1.pptx, week6.pptx, 2025.