An-Najah National University

Department of Computer Engineering

Graduation Project II

ETHER: An Adaptive Solar Tracking System

Mohammad Raed

Mohammad Matar

Supervisor: Dr. Suleiman Abu Kharmeh

A report submitted in partial fulfilment of the requirements of
An-Najah National University for the degree of
Bachelor of Science in *Computer Engineering*

January 27, 2026

Abstract

The growing global demand for renewable energy has emphasized the critical need for maximizing solar energy system efficiency. This project presents the design and implementation of an intelligent dual-axis solar tracking system that integrates embedded control, wireless connectivity, and FPGA-based hardware acceleration to enhance tracking precision and system responsiveness.

The system architecture comprises three main components working in coordination: an Arduino microcontroller implementing a Proportional-Integral-Derivative (PID) controller to drive servo motors for precise panel orientation, an ESP32 module providing wireless connectivity for system monitoring and dashboard applications, and a DE1-SoC FPGA board executing a Kalman filter algorithm for predictive control. The Arduino calculates solar position using built-in algorithms and communicates with the FPGA via UART protocol to transmit sensor data and receive filtered position estimates. The FPGA implementation utilizes DSP slices to accelerate the mathematical computations of the Kalman filter, providing real-time predictive adjustments that compensate for measurement delays and noise in the control system.

The developed system successfully demonstrates improved tracking accuracy through the integration of predictive filtering with embedded PID control. The FPGA-accelerated Kalman filter effectively reduces positioning errors and system oscillations, while the wireless monitoring capability enables real-time performance assessment. Environmental sensors provide contextual data for comprehensive system evaluation.

This work demonstrates how hardware-software co-design enhances renewable energy systems through improved precision, adaptability, and computational efficiency. The integration of predictive control with FPGA acceleration represents a novel approach to solar tracking that advances beyond traditional microcontroller-only solutions, offering significant potential for both educational applications and small-scale renewable energy implementations.

Keywords: FPGA, SoC, Kalman Filter, PID Control, Solar Tracking, Embedded Systems, Renewable Energy, Internet of Things, DE1-SoC, Arduino, ESP32

The full project repository can be found at:

- https://github.com/mo-matar/Heterogeneous-Solar-Tracking-System

Acknowledgements

We would like to express our sincere thanks to our supervisor, Dr. Suleiman Abu Kharmeh, for providing the DE1-SoC board and supporting us throughout this project. His continued guidance, professional feedback, and effort helped us improve our work and overcome challenges during development.

Contents

List of Figures

List of Tables

List of Abbreviations

FPGA	Field-Programmable Gate Array
HPS	Hard Processor System
LDR	Light Dependent Resistor
AXI	Advanced eXtensible Interface
RMSE	Root Mean Square Error

Chapter 1

Introduction

1.1 General Background

The global transition toward renewable energy sources has positioned solar photovoltaic (PV) technology as a cornerstone of sustainable energy infrastructure. As governments worldwide implement ambitious carbon reduction targets and renewable energy mandates, the optimization of solar energy capture has become increasingly critical. Solar panels operating in fixed positions can only harvest a fraction of available solar energy throughout the day, as the sun's position continuously changes due to Earth's rotation and orbital mechanics.

Solar tracking systems address this fundamental limitation by dynamically orienting photovoltaic panels to maintain optimal alignment with the sun's position. Research has demonstrated that dual-axis solar tracking can increase energy capture compared to fixed installations, making it an attractive solution for maximizing return on investment in solar infrastructure. Traditional solar tracking systems typically employ either passive tracking mechanisms or basic microcontroller-based control systems that rely solely on sensor feedback or astronomical calculations.

The emergence of embedded systems and field-programmable gate arrays (FPGAs) has opened new possibilities for intelligent control systems that can enhance tracking precision through predictive algorithms. Modern solar tracking applications demand not only accurate positioning but also smooth operation, minimal power consumption, and robust performance under varying environmental conditions. The integration of advanced filtering techniques, such as Kalman filters, with hardware acceleration presents an opportunity to achieve superior tracking performance while maintaining computational efficiency.

Wireless connectivity and Internet of Things (IoT) technologies have further transformed the landscape of solar energy systems, enabling remote monitoring, predictive maintenance, and real-time performance optimization. The convergence of embedded control, hardware acceleration, and wireless monitoring creates the foundation for next-generation intelligent solar tracking systems that can adapt to changing conditions and provide comprehensive operational insights.

1.2 Objectives and Purpose

The primary objective of this project is to design and implement an intelligent dual-axis solar tracking system that integrates embedded control, hardware-accelerated predictive filtering, and wireless monitoring to achieve enhanced tracking precision and system performance. The system aims to demonstrate how hardware-software co-design can improve upon traditional

solar tracking approaches through the strategic combination of different computing platforms.

The specific objectives of this project include:

1. **Embedded Control Development**: Design and implement a microcontroller-based control system using Arduino that employs Proportional-Integral-Derivative (PID) control algorithms to drive dual-axis servo motors for precise solar panel orientation.

2. **Hardware-Accelerated Filtering**: Develop and deploy a Kalman filter algorithm on FPGA fabric to provide predictive position estimation that compensates for measurement delays, sensor noise, and environmental disturbances.

3. **System Integration**: Establish robust communication protocols between the Arduino controller and FPGA processing unit to enable real-time data exchange and coordinated control actions.

4. **Wireless Monitoring**: Implement a comprehensive monitoring and visualization system using ESP32 technology that provides real-time system status, performance metrics, and remote control capabilities.

5. **Performance Evaluation**: Conduct thorough testing and analysis to quantify the improvements in tracking accuracy, stability, and overall system performance achieved through the integrated approach.

The project serves both educational and practical purposes, demonstrating advanced embedded systems concepts while creating a functional renewable energy solution suitable for small-scale applications. The modular design approach ensures that individual components can be analyzed and optimized independently while contributing to overall system performance.

1.3 Significance and Importance

This project addresses several critical challenges in modern solar tracking systems while advancing the state of the art in embedded renewable energy control. The significance of this work extends across multiple domains, from renewable energy optimization to embedded systems engineering and hardware-software co-design methodologies.

Renewable Energy Impact: The integration of predictive control algorithms represents a significant advancement over traditional reactive tracking systems. By anticipating solar position changes and compensating for system delays, the proposed approach can achieve smoother tracking motion and reduced mechanical wear, ultimately leading to improved energy capture efficiency and longer system lifespan.

Embedded Systems Innovation: The project demonstrates the practical application of heterogeneous computing in embedded systems, where different processing platforms (microcontroller, FPGA, and wireless module) collaborate to achieve superior performance. This approach showcases how modern embedded system designers can leverage the unique strengths of different computing architectures.

Hardware Acceleration Benefits: The FPGA implementation of the Kalman filter provides computational acceleration while maintaining real-time performance requirements. This approach demonstrates how complex mathematical algorithms can be efficiently implemented in hardware to achieve deterministic timing and reduced computational load on the primary controller.

Educational Value: The project serves as an excellent educational platform for understanding advanced control theory, digital signal processing, embedded systems design, and

renewable energy systems. The modular architecture allows students and researchers to examine individual components while understanding their integration within a complete system.

Scalability and Adaptability: The design principles and architectural approaches developed in this project can be scaled and adapted for larger solar installations and other tracking applications, making the research relevant for both small-scale and commercial implementations.

Cost-Effectiveness: By utilizing widely available development platforms and demonstrating efficient resource utilization, the project provides insights into developing cost-effective intelligent tracking solutions that can compete with commercial alternatives.

1.4 Summary of Contributions and Achievements

This project makes several novel contributions to the field of intelligent solar tracking systems and embedded control:

System Architecture Contributions:

- Development of a heterogeneous embedded system architecture that combines Arduino-based PID control, FPGA-accelerated Kalman filtering, and ESP32-based wireless monitoring
- Design of efficient communication protocols for real-time data exchange between different computing platforms
- Implementation of a modular system design that enables independent optimization of control, filtering, and monitoring subsystems

Control System Innovations:

- Integration of predictive Kalman filtering with traditional PID control to achieve enhanced tracking smoothness and accuracy
- Development of adaptive control parameters that respond to varying environmental conditions and system dynamics
- Implementation of intelligent sensor fusion that combines light-dependent resistor (LDR) measurements with predictive algorithms

Hardware Implementation Achievements:

- Successful deployment of a scalar Kalman filter algorithm on FPGA hardware using fixed-point arithmetic optimization
- Achievement of real-time performance requirements while maintaining computational accuracy
- Demonstration of effective resource utilization on the DE1-SoC development platform

System Integration Accomplishments:

- Creation of a comprehensive monitoring and visualization interface that provides real-time system insights
- Development of robust error handling and fault tolerance mechanisms

- Implementation of performance tracking and statistical analysis capabilities for system optimization

The completed system successfully demonstrates improved tracking performance compared to traditional approaches while maintaining cost-effectiveness and educational accessibility. The project provides a foundation for future research in intelligent renewable energy systems and hardware-software co-design methodologies.

1.5 Organization of the Report

This report is organized into six main chapters, each addressing specific aspects of the solar tracking system design, implementation, and evaluation:

Chapter 1: Introduction provides the foundational context for the project, including background information on solar tracking technology, project objectives, significance of the research, and key contributions.

Chapter 2: Literature Review examines existing solar tracking systems, control algorithms, Kalman filtering approaches, and FPGA-based acceleration. This chapter establishes the theoretical foundation and motivates the heterogeneous architecture adopted in this work.

Chapter 3: Methodology presents the system architecture and co-design methodology, detailing hardware platform roles (Arduino, DE1-SoC FPGA, ESP32), communication protocols, and the fixed-point adaptive Kalman filter hardware implementation and verification approach.

Chapter 4: Results reports FPGA synthesis/resource utilization outcomes, Kalman filter simulation performance across test scenarios, and integrated system demonstration results including real-time dashboard visualization.

Chapter 5: Discussion interprets the key findings, compares the proposed approach against prior work, analyzes implications of major hardware/software design choices, and documents practical limitations and deployment considerations.

Chapter 6: Conclusion and Recommendations summarizes the project's achievements and contributions, and provides concrete recommendations for future extensions such as runtime-adaptive filtering, predictive models, power management, and enhanced monitoring analytics.

Additional supporting materials, including detailed code listings, hardware specifications, and supplementary analysis, are provided in the appendices to ensure reproducibility and facilitate further research.

Chapter 2

Literature Review

2.1 Introduction

Global energy demand continues to rise, with renewable sources becoming increasingly critical. Photovoltaic (PV) systems offer a promising solution, but their efficiency depends heavily on proper orientation toward the sun. Dual-axis solar trackers address this by dynamically adjusting panel orientation, with research showing they can generate 31.4% more energy than single-axis trackers and 67.9% more than fixed panels Amadi and Gutiérrez (2019).

The evolution of solar tracking systems mirrors trends in embedded systems design. Early implementations used simple time-based controllers or basic light sensors. The Arduino platform brought accessible microcontroller-based systems using Light Dependent Resistors (LDRs) and straightforward control logic Mohanapriya et al. (2021). More recently, IoT-capable systems have emerged, with Mustafa et al.'s ESP32-based tracker incorporating software-based Kalman filtering, PID control, and web dashboards, achieving approximately 43% energy gain over fixed panels MUSTAFA (2024).

However, a significant gap persists in the literature. Accessible end-user systems prioritize simplicity and cost but sacrifice performance due to computational limitations. Research-focused implementations demonstrate advanced algorithms on high-performance hardware but remain impractical for deployment. Even systems incorporating FPGAs, like BaBars et al.'s work, use the FPGA as a monolithic controller rather than a specialized co-processor BaBars et al. (2025).

This review argues that a heterogeneous architecture—strategically combining Arduino for sensor acquisition, FPGA for hardware-accelerated signal processing, and ESP32 for wireless connectivity—represents the optimal solution. By leveraging each component's strengths, such a system can deliver research-grade filtering performance within a practical, deployable platform, addressing the fundamental trade-off between accessible simplicity and computational sophistication.

2.2 Existing Microcontroller Systems and Their Limitations

The Arduino platform has become the standard for entry-level solar tracking projects due to its accessibility and straightforward programming model. A typical implementation uses four LDR sensors in a quadrant configuration, with the Arduino processing analog inputs, calculating positional errors, and driving motors to align the panel with the sun Mohanapriya et al. (2021). This design is economical and intuitive—essentially comparing sensor readings and moving toward brighter light.

However, the Arduino's 16 MHz ATmega328P with limited memory (32 KB flash, 2 KB RAM) restricts algorithm sophistication. When LDR sensors provide noisy data—common with intermittent cloud cover or reflections—the Arduino cannot implement advanced filtering without compromising control loop responsiveness.

The ESP32 has enabled a new generation of "smart" trackers with wireless connectivity and monitoring interfaces. Mustafa et al.'s system integrates LDR sensing, software-based Kalman filtering, PID control, and a web dashboard—all on a single ESP32 MUSTAFA (2024). While the dual-core 240 MHz processor is considerably more capable, it must time-slice between multiple demanding tasks: sensor acquisition, Kalman filter equations, PID control, WiFi communication, and web serving.

This creates fundamental limitations. First, complex algorithms like Kalman filters require multiple floating-point operations per update cycle, executed serially on the sequential processor. Research by Linares-Barranco et al. shows CPU-based filtering exhibited 8-24 ms latencies versus 4-4.2 ms on FPGAs—a 188-570% reduction Linares-Barranco et al. (2019). Second, LDR sensors produce inherently noisy analog signals, and microcontroller-based systems typically use simple techniques like moving averages that lack the mathematical rigor of optimal estimation. Third, designers face a zero-sum trade-off: adding features or improving one aspect necessarily degrades another. Implementing a more complex filter reduces control loop update rates. Serving a feature-rich dashboard consumes memory and processor time needed for accurate control.

These limitations manifest as reduced tracking accuracy during challenging conditions, slower response to changing light, and system instability when subsystems compete for resources. The question becomes: how to transcend these limitations without abandoning the cost-effectiveness and accessibility that make microcontroller systems attractive?

2.3 Kalman Filtering for Optimal State Estimation

The Kalman filter, introduced by Rudolf E. Kálmán in 1960, provides the optimal recursive solution to discrete-data linear filtering problems. As detailed in the authoritative text reviewed by Bass, the filter gives minimum mean-square error state estimates based on noisy measurements Bass (1996). Unlike ad-hoc techniques, it is mathematically proven optimal under certain conditions (linear system dynamics, Gaussian noise, known noise statistics).

The filter operates in a predict-update cycle. In prediction, it forecasts the next state using the system's dynamic model. In update, it incorporates a new measurement by optimally weighing prediction against measurement based on their respective uncertainties. This weighting, determined by the Kalman gain computed from estimation error covariance, allows the filter to "learn" appropriate trust levels over time.

For solar tracking, the characteristics align remarkably well with system requirements. The filter's optimal weighting distinguishes genuine sun position changes from random noise fluctuations, producing smooth angle estimates even with highly variable sensor readings. It can fuse information from four LDR sensors into a unified state estimate, more robust than simple error signal calculation. During brief cloud cover when LDR readings become unreliable, the filter relies more on its prediction model, preventing erratic movements. The continuously updated estimates enable smoother motor control, reducing mechanical wear and power consumption.

Mustafa et al. demonstrated that even software-implemented Kalman filtering on an ESP32 provided tangible benefits MUSTAFA (2024). However, their work highlights the computational challenge: the filter consumed significant processor resources, requiring optimizations to maintain real-time performance alongside other tasks. The filter involves nu-

merous matrix operations—multiplications, additions, inversions—in each update cycle. On conventional microcontrollers, these execute sequentially. As filter dimensionality increases, computational burden grows rapidly, forcing compromises in filter models, update rates, or precision.

2.4 FPGAs for Hardware-Accelerated Filtering

Field-Programmable Gate Arrays represent a fundamentally different computing paradigm. While microcontrollers follow von Neumann architecture—sequentially executing instructions—FPGAs provide a fabric of configurable logic blocks and programmable interconnects arranged to create custom digital circuits. As Woods explains, this architecture is particularly well-suited for DSP algorithms benefiting from parallel, pipelined execution Woods et al. (2008).

For Kalman filtering, the advantages are substantial. Matrix operations required in each iteration execute sequentially on microcontrollers—multiply a and b, store the result, multiply c and d, add results. Total latency is the sum of individual operation latencies. On FPGAs, these operations become parallel hardware circuits. Multiple multipliers operate simultaneously, and pipelining allows different computation stages to overlap. A well-designed FPGA implementation completes an entire Kalman filter update in fixed clock cycles because all necessary arithmetic units operate concurrently. This processing is deterministic—no operating system, no interrupts, no context switching.

Empirical evidence strongly supports these advantages. Linares-Barranco et al.'s comparison of software filtering on CPUs versus FPGAs showed CPU implementations with 8-24 ms latencies while FPGA achieved consistent 4-4.2 ms—a 188-570% reduction Linares-Barranco et al. (2019). AlShabi et al. implemented an Unscented Kalman Filter—more complex than standard Kalman—on an FPGA for target tracking, achieving real-time performance for an algorithm that would severely strain microcontrollers AlShabi and Bonny (2022). The UKF requires not only matrix operations but also sigma point generation and transformation, yet FPGA hardware handled it efficiently.

However, FPGAs are not universally superior. They have higher costs, require specialized design skills (Verilog/VHDL), consume more power, and involve complex development workflows. For simple tasks like reading sensors or toggling GPIO pins, microcontrollers are more appropriate with simpler programming models and lower costs. But for computationally intensive tasks like real-time Kalman filtering, the cost-performance equation shifts dramatically toward FPGAs. Development complexity is justified by order-of-magnitude performance improvements.

This leads to a critical insight: rather than choosing between microcontrollers and FPGAs, optimal system design should leverage both, assigning tasks to components best suited for them. This heterogeneous approach—microcontrollers for I/O and control, FPGAs for intensive computation—represents a middle ground combining accessibility with high performance.

2.5 Research Gap and Opportunity

A comprehensive survey of solar tracking implementations reveals clear bifurcation with minimal middle ground. The overwhelming majority of practical systems use single microcontrollers (Arduino or ESP32) as the sole computational element, prioritizing simplicity and cost-effectiveness Mohanapriya et al. (2021); MUSTAFA (2024). Advanced academic implementations demonstrate sophisticated algorithms on high-performance hardware like FPGAs AlShabi and Bonny (2022), but remain laboratory prototypes due to complexity and lack of

practical integration.

Notably absent are systems integrating FPGAs as specialized co-processors within accessible architectures. BaBars et al.'s work represents a partial step, using an FPGA (Xilinx Spartan) as the core controller with IoT connectivity BaBars et al. (2025). However, their design uses the FPGA as a monolithic controller replacement, handling all tasks from sensor acquisition to motor control to communication. This misses the opportunity for optimal task specialization that heterogeneous architecture provides—using an FPGA for reading analog sensors or toggling motor pins is like using a supercomputer for word processing.

This gap can be summarized across three dimensions:

- **Hardware architecture:** end-user systems favor single-MCU designs, while research favors FPGA-centric prototypes; accessible systems rarely use an FPGA as a dedicated compute co-processor.
- **Algorithm realization:** practical designs rely on basic control and lightweight filtering, whereas research demonstrates advanced Kalman variants; translating these algorithms into deployable, real-time embedded pipelines remains uncommon.
- **System integration:** many works are either all-in-one MCU implementations MUSTAFA (2024) or FPGA-as-main-controller designs BaBars et al. (2025), leaving limited emphasis on task-partitioned, maintainable heterogeneous co-design.

The gap represents both a limitation and an opportunity for innovation. A heterogeneous architecture combining Arduino (for I/O), FPGA (for filtering), and ESP32 (for connectivity) would leverage each component's strengths while avoiding weaknesses. The Arduino provides reliable, low-latency sensor acquisition and motor control without operating system overhead. The FPGA delivers order-of-magnitude performance improvement for Kalman filtering without unnecessary complexity for simple tasks. The ESP32 offers seamless wireless connectivity and web serving without consuming computational resources needed for control. This synergistic approach represents the logical next step—bridging the gap between accessible, practical systems and research-grade signal processing performance.

2.6 Proposed Heterogeneous Architecture

The analysis converges on a clear conclusion: the optimal solar tracking system requires heterogeneous architecture strategically assigning tasks based on component strengths.

The **Arduino Mega** serves as the sensor acquisition and actuation hub. Its 10-bit ADCs provide adequate resolution for LDR readings, and its real-time execution model ensures deterministic sampling timing. The Arduino implements PID control loops driving motors based on filtered angle estimates from the FPGA. This plays to the Arduino's strengths—robust hardware interfacing, predictable timing, straightforward control logic—while avoiding its weakness in complex numerical computation. By offloading Kalman filtering to the FPGA, the Arduino maintains high-frequency sensor sampling and rapid control loop execution without compromise.

The **DE1-SoC FPGA** implements dual Kalman filters—one for azimuth, one for elevation—in fully parallel, pipelined hardware. The filters receive raw LDR data from the Arduino via serial interface, perform optimal estimation using fixed-point arithmetic (Q9.7 format for 0-180° servo range), and transmit filtered angle estimates back at high rates. Parallel arithmetic units perform matrix operations simultaneously, achieving microsecond rather than millisecond latencies. Deterministic timing ensures consistent filtering performance regardless

of communication load. The FPGA is used exclusively for what it does best—intensive, parallel numerical computation—maximizing cost-effectiveness by ensuring capabilities are fully utilized for the specific problem it uniquely solves.

The **ESP32** hosts the wireless interface and web dashboard. It receives filtered angle estimates and system status from the Arduino, serves a real-time web interface displaying tracking performance, and provides remote control capabilities. The ESP32's powerful WiFi stack and sufficient processing power make it ideal for this role. Crucially, it is freed from computational burden of running Kalman filters or maintaining precise control timing, dedicating resources entirely to communication and user interface tasks.

The architecture's benefits exceed the sum of parts. By distributing computational tasks across specialized hardware, the system achieves performance levels impossible for any single microcontroller. The Kalman filter runs at FPGA speeds (microsecond latency), control loops execute at Arduino real-time rates (millisecond update cycles), and the web dashboard operates smoothly without impacting tracking performance. Clear separation of concerns simplifies development and debugging. The system is scalable—adding sensors or sophisticated control strategies on Arduino doesn't affect FPGA filtering performance, and extending Kalman filters doesn't impact Arduino control timing or ESP32 communication.

Most fundamentally, this approach bridges the identified gap between accessible end-user systems and research-grade performance. Arduino and ESP32 components keep the system approachable with familiar programming models and extensive community support. The FPGA component brings research-level signal processing into a practical, deployable system. Users gain benefits of hardware-accelerated Kalman filtering (optimal noise rejection, sensor fusion, predictive capability) without sacrificing accessibility and user-friendliness of conventional designs.

Unlike all-in-one microcontroller systems, it forces no computational compromises Mohanapriya et al. (2021); MUSTAFA (2024). Unlike pure FPGA implementations, it doesn't use expensive hardware for tasks simple microcontrollers handle effectively BaBars et al. (2025). Unlike research systems focusing on algorithm implementation without practical deployment considerations AlShabi and Bonny (2022), it integrates high-performance filtering within a complete system with user interfaces, remote monitoring, and straightforward replication.

2.7 Summary

This literature review traced solar tracking system evolution from simple controllers to sophisticated IoT platforms, analyzed Kalman filtering foundations and benefits, demonstrated FPGA architectural advantages for real-time signal processing, and identified a critical gap in existing implementations.

Current systems occupy two extremes. End-user designs prioritize accessibility by using single microcontrollers for all functions, but encounter computational bottlenecks and performance trade-offs as complexity increases. Research implementations demonstrate advanced algorithms and high-performance hardware but remain impractical for deployment due to specialization and lack of integration with user-friendly components.

The identified gap—absence of practical systems integrating hardware-accelerated signal processing with accessible microcontroller-based components—represents both a limitation and an opportunity. The proposed heterogeneous architecture, strategically distributing tasks among Arduino (sensing and actuation), FPGA (Kalman filtering), and ESP32 (wireless connectivity), directly addresses this gap by combining conventional embedded system accessibility with specialized signal processing hardware performance.

Expected contributions extend beyond demonstrating another solar tracker. By successfully integrating FPGA-based Kalman filtering into a practical, user-friendly system, this project establishes a model for future embedded system designs leveraging component specialization rather than forcing monolithic solutions. It demonstrates that benefits of advanced signal processing algorithms—proven in theory and demonstrated in isolation—can be brought to practical applications through thoughtful architectural design matching computational tasks to optimal hardware platforms.

Ultimately, this work aims to show that the choice between accessible simplicity and computational sophistication is a false dichotomy. Through heterogeneous system design, both goals can be achieved simultaneously, creating solar tracking systems that are practical to build and deploy while delivering research-grade filtering performance. This represents the natural evolution of embedded system design for applications where sensing, signal processing, control, and communication all play critical roles.

Chapter 3

Methodology

3.1 Overall System Architecture and Block Diagram

The proposed solar tracking system employs a heterogeneous architecture that strategically distributes computational tasks across three specialized hardware platforms: Arduino Mega for sensor acquisition and actuation, DE1-SoC FPGA for hardware-accelerated Kalman filtering, and ESP32 for wireless connectivity and web-based monitoring. This architecture is designed to overcome the fundamental trade-offs that limit monolithic microcontroller implementations while maintaining practical deployability.

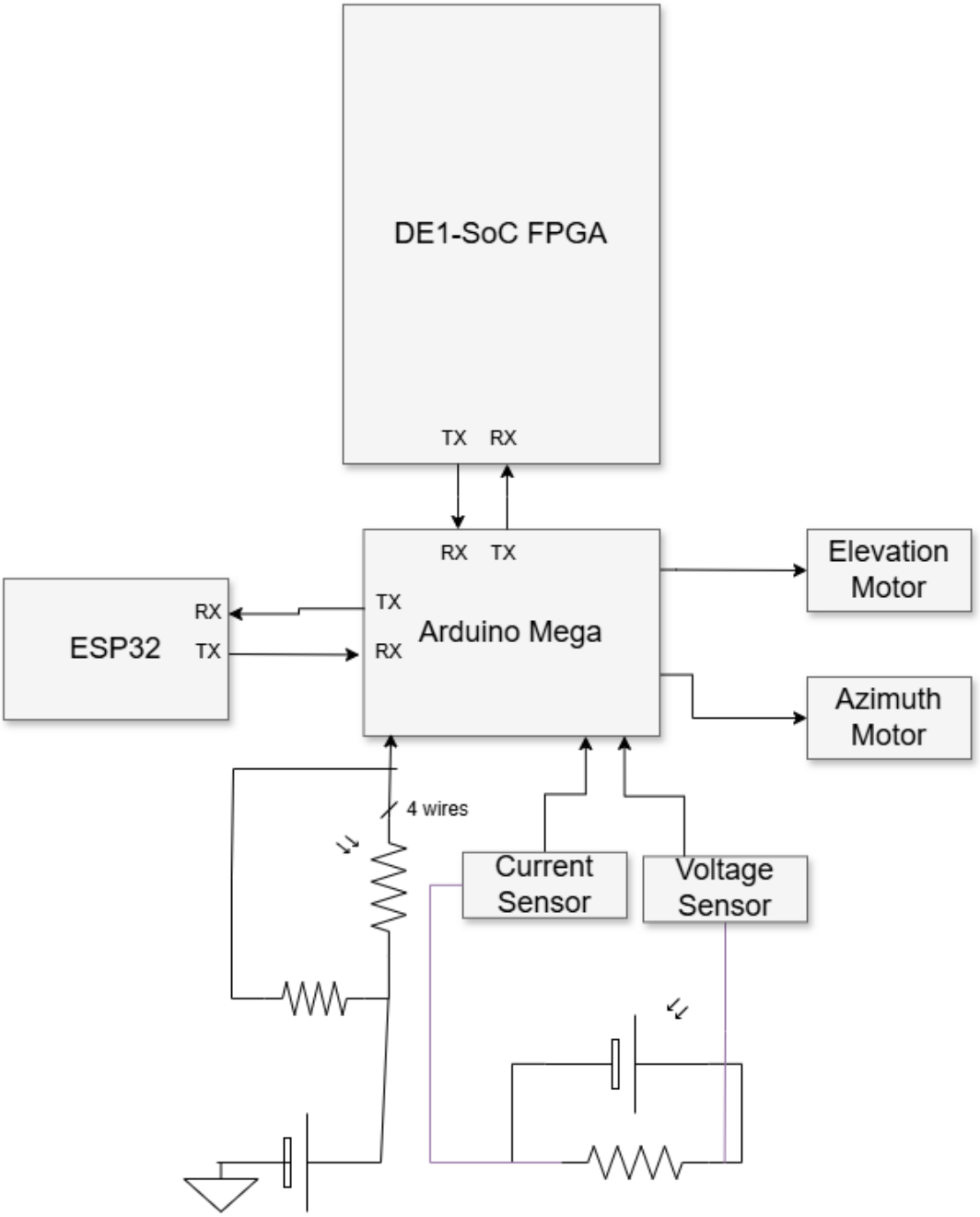

Figure 3.1: System-level block diagram showing the heterogeneous data flow between Arduino (sensor/actuation layer), DE1-SoC FPGA (computation layer), and ESP32 (communication layer)

Figure 3.1 illustrates the complete system architecture and the strategic division of labor among the three computational platforms. The data flow follows a carefully orchestrated pipeline designed to maximize the strengths of each component while minimizing inter-component communication overhead.

The Arduino Mega serves as the front-end sensor acquisition and actuation controller. It interfaces with four Light Dependent Resistors (LDRs) arranged in a quadrant configuration to capture sunlight intensity from multiple directions. The Arduino implements a PID control algorithm that computes raw azimuth and elevation angle commands based on the differential

LDR readings. Rather than directly commanding the servo motors with these potentially noisy estimates, the Arduino transmits the raw angle data to the FPGA for filtering. This design decision is crucial: by offloading the computationally intensive Kalman filter to dedicated hardware, the Arduino maintains a fast, deterministic control loop without time-slicing between sensing, filtering, and actuation tasks.

The DE1-SoC FPGA forms the computational core of the system. It receives raw angle measurements from the Arduino via UART serial communication at 115200 baud. The FPGA fabric implements dual Kalman filters—one for azimuth and one for elevation—in fully parallel, pipelined hardware. These filters perform optimal state estimation using fixed-point arithmetic (Q9.7 format, providing 16-bit signed representation with 7 fractional bits, suitable for the 0° to 180° servo range). The filtered angle estimates are transmitted back to the Arduino through the same UART link within microseconds, enabling real-time closed-loop control with research-grade signal processing performance. The FPGA's Hard Processor System (HPS), featuring a dual-core ARM Cortex-A9 running at 925 MHz, plays a critical role during system initialization by programming the FPGA fabric via AXI bridges and providing future extensibility for advanced monitoring capabilities.

The ESP32 microcontroller handles all wireless communication and user interface functionality. It receives system telemetry from the Arduino via a second UART link operating at 9600 baud, including filtered and raw angle data, LDR sensor values, servo positions, power consumption metrics, and system statistics. The ESP32 operates as a WiFi Access Point, hosting a responsive web dashboard that provides real-time visualization, remote parameter tuning, and manual control capabilities. By dedicating the ESP32 exclusively to communication tasks, the system avoids the performance degradation that occurs when a single microcontroller must handle sensing, filtering, control, and wireless communication concurrently.

This heterogeneous architecture achieves several critical objectives. First, it breaks the performance ceiling imposed by sequential processing limitations—the Kalman filter executes in parallel hardware at FPGA clock speeds while the Arduino control loop maintains millisecond update rates without interference. Second, it maintains accessibility—the Arduino and ESP32 use familiar programming environments (Arduino IDE and C/C++), while only the computationally critical filtering component requires hardware description language (Verilog) implementation. Third, it provides modularity and scalability—each subsystem can be independently developed, tested, and enhanced without affecting the others. Finally, it demonstrates a practical model for integrating advanced DSP algorithms into embedded systems without requiring monolithic high-performance solutions that increase cost and complexity unnecessarily.

3.2 Hardware Subsystem Design

3.2.1 Sensor and Actuation Layer (Arduino Domain)

The Arduino Mega 2560 microcontroller serves as the sensor acquisition and motor control platform, chosen for its abundant I/O capabilities, mature ecosystem, and deterministic real-time execution characteristics. The board provides 16 analog input channels with 10-bit resolution (0-1023 ADC range), sufficient for interfacing with Light Dependent Resistor (LDR) sensors that exhibit resistance variations from approximately 1 kΩ in bright light to over 10 MΩ in darkness.

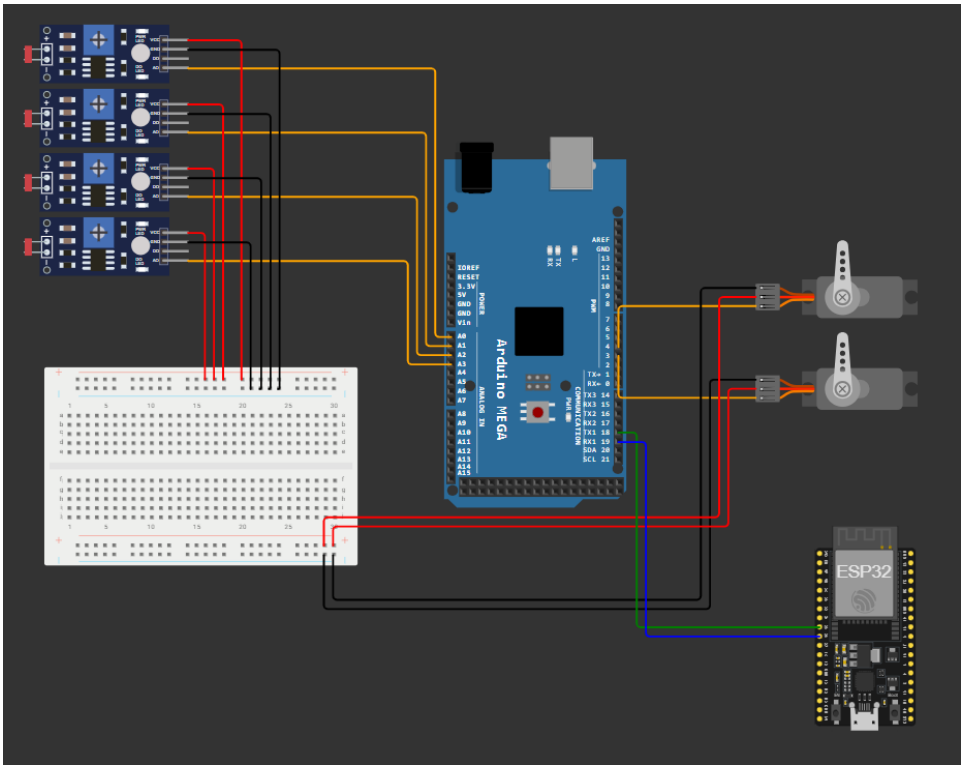

Figure 3.2: Arduino Mega connection diagram showing LDR sensor interfacing, servo motor control, and UART communication links to FPGA and ESP32

Figure 3.2 details the Arduino's interface connections. Four LDR sensors are connected to analog inputs A0 (top-left), A1 (top-right), A2 (bottom-left), and A3 (bottom-right), each configured in a voltage divider network with a 10 kΩ fixed resistor to ground. This arrangement produces a voltage output proportional to light intensity. Two servo motors—one for azimuth (horizontal) rotation and one for elevation (vertical) tilt—are controlled via PWM signals on digital pins 3 and 4, generating standard 50 Hz signals with pulse widths from 1000 to 2000 microseconds for 0° to 180° angular range.

Dual-UART Communication Architecture

The Arduino implements two serial communication channels. Serial3 (pins 14/15, operating at 115200 baud) connects to the FPGA for real-time Kalman filtering. Raw angle data computed by the PID controller is transmitted in Q9.7 fixed-point format as four-byte packets: [AZ_HIGH][AZ_LOW][EL_HIGH][EL_LOW]. The Arduino receives filtered estimates in the same format within approximately 2 milliseconds. Serial1 (pins 18/19, operating at 9600 baud) connects to the ESP32, transmitting ASCII-formatted telemetry containing sensor readings, servo positions, angle data, and system statistics for dashboard display.

PID Control Implementation

The Arduino firmware uses separate PID controllers for azimuth and elevation. Every 100 ms, it reads the LDR sensors and calculates horizontal and vertical errors by comparing the average light on each side. The PID controllers use tuned gains for proportional, integral, and derivative terms to generate raw angle commands.

When the FPGA Kalman filter is enabled, the Arduino transmits these raw angles in Q9.7 fixed-point format to the FPGA via Serial3 (115200 baud). Within approximately 2 milliseconds, the FPGA responds with filtered angle values in the same Q9.7 format. The Arduino then converts these filtered angles back to integer degrees and commands the servos directly. This entire transaction—transmission, FPGA filtering (including Kalman filter computation in hardware), and FPGA transmission back to Arduino—completes well within the 100 ms control loop cycle, ensuring deterministic real-time operation.

If the FPGA times out (no response within 100 ms) or the filter is disabled, the Arduino uses the raw PID output directly to command the servos, providing a fallback that maintains system responsiveness.

To improve reliability, the code includes integral windup protection, derivative filtering (using a moving average), a deadband to ignore small errors, and logic to detect when servos reach their limits. These refinements help prevent overshoot, reduce jitter, and extend hardware life.

Operational Modes and Filter Integration

The system has two modes: AUTO and MANUAL. In AUTO, the Arduino reads LDR sensors, runs a PID controller, and sends raw angles to the FPGA for Kalman filtering. If the FPGA responds in time, the filtered angles are used to move the servos; otherwise, the raw angles are used. In MANUAL, the user sets servo positions directly from the ESP32 dashboard.

When switching to AUTO with filtering, the Arduino sends several packets with the current position to let the FPGA filter sync up and avoid jumps.

Statistics and Telemetry

The Arduino tracks basic stats like servo movements, peak light, and lock time, and sends these to the ESP32 for dashboard display.

This setup keeps the Arduino fast and responsive by offloading filtering to the FPGA.

3.2.2 Computation Layer (DE1-SoC FPGA Domain)

The DE1-SoC development board, featuring an Intel Cyclone V SoC FPGA, forms the computational heart of the proposed system. This platform was selected for its unique combination of FPGA fabric and Hard Processor System (HPS), which together provide both the parallel processing capabilities required for real-time Kalman filtering and the software flexibility needed for system configuration and future extensibility.

Rationale for DE1-SoC Platform Selection

The Cyclone V SoC device on the DE1-SoC board integrates 85,000 logic elements (LEs), 4,450 Kbits of embedded memory, 87 DSP blocks, and a dual-core ARM Cortex-A9 processor subsystem operating at 925 MHz. This heterogeneous architecture is crucial for the proposed system design, as it enables a clear separation of concerns: the FPGA fabric implements high-speed, deterministic signal processing, while the HPS manages system initialization, configuration, and monitoring.

The HPS-FPGA integration via high-bandwidth AXI bridges is particularly important for this project. During system startup, the HPS configures the FPGA fabric by loading the Raw Binary File (RBF) through Passive Parallel configuration mode. This approach provides several advantages over standalone FPGA designs: (1) it eliminates the need for external

configuration memory or JTAG programmer during normal operation, (2) it allows for dynamic reconfiguration if future enhancements require different FPGA configurations, (3) it provides a path for the HPS to monitor FPGA operation via memory-mapped PIO ports, and (4) it offers potential for hybrid hardware-software algorithms where the HPS performs preprocessing or post-analysis on FPGA results.

For the current implementation, the HPS operates primarily in a supervisory role. After programming the FPGA fabric on startup, it monitors four 16-bit PIO (Parallel I/O) ports connected to the FPGA fabric that carry raw and filtered azimuth and elevation angle values. While these ports are not actively used in the current system's primary data path (which flows directly from Arduino to FPGA to Arduino via UART), they provide future extensibility for HPS-based data logging, anomaly detection, or advanced diagnostic capabilities without modifying the real-time critical path.

The passive parallel configuration mode employed on the DE1-SoC board is significant for deployment. In this mode, the FPGA fabric appears as a passive device to the HPS, which actively drives the configuration data and control signals. This contrasts with active serial configuration where the FPGA itself fetches configuration data from external memory. The passive approach is more robust in the field, as it ensures the HPS can always recover and reconfigure the FPGA if transient faults occur, providing a degree of fault tolerance absent in simpler FPGA-only designs.

FPGA System Architecture

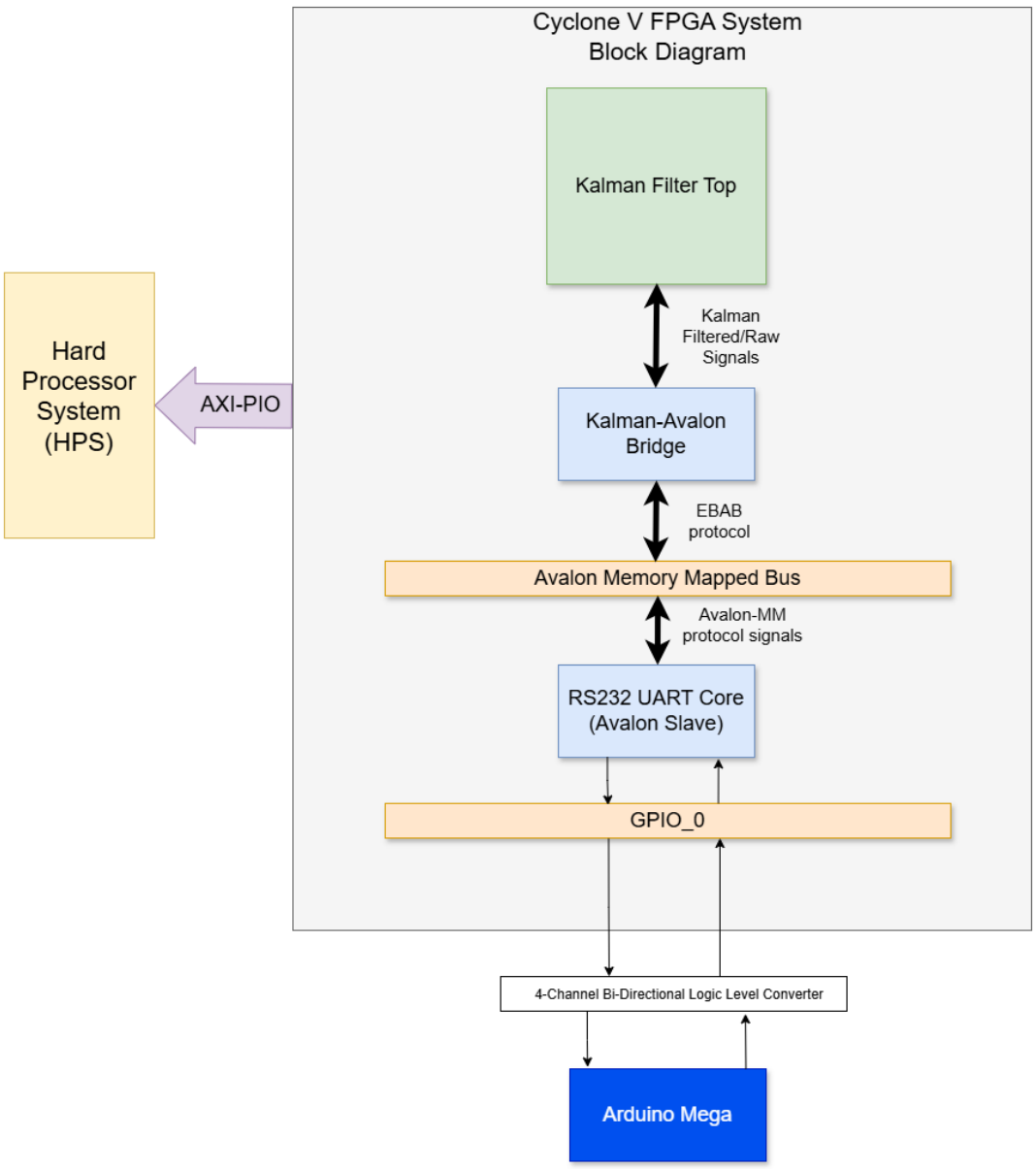

Figure 3.3: Cyclone V SoC top-level block diagram showing the FPGA fabric, HPS subsystem, and their interconnections via high-performance AXI bridges

Figure 3.3 presents the high-level architecture of the Cyclone V SoC device. The FPGA fabric occupies the majority of the die area and is where the dual Kalman filter cores, Avalon-MM interconnect fabric, UART controller, and custom interface bridges are implemented. The HPS subsystem contains the dual-core ARM Cortex-A9 MPCore processor, on-chip memory, memory controllers for external DDR3 SDRAM, and a rich set of hard IP peripherals including Ethernet, USB, SD card, and SPI controllers.

The lightweight HPS-to-FPGA and FPGA-to-HPS AXI bridges provide low-latency, high-bandwidth communication paths between the processor subsystem and the programmable logic. These 32-bit or 64-bit wide buses operate at frequencies up to 125 MHz, enabling the HPS to access memory-mapped registers in the FPGA fabric with latencies comparable to

on-chip peripheral access. For this project, the HPS uses the HPS-to-FPGA bridge during the configuration phase to load the RBF into the FPGA fabric, and it monitors the PIO ports through the same bridge during runtime.

The FPGA fabric implementation consists of several major functional blocks:

Kalman Filter Cores: Two instantiated `kalman_scalar` modules (implemented in the file `kalman_scalar.v`) form the signal processing heart of the system. Each filter operates independently on one degree of freedom—azimuth or elevation—implementing the full predict-update cycle of the Kalman filter algorithm. The filters use Q9.7 fixed-point arithmetic, which provides sufficient precision for the 0° to 180° servo angle range while enabling compact hardware implementation. The filters include adaptive measurement noise covariance (R) scaling to handle varying sensor noise conditions, particularly during partly cloudy periods when LDR readings fluctuate rapidly.

Kalman to Avalon Bridge: The custom `Kalman_Avalon_Bridge` module (implemented in `kalman_2_avalon_bridge.v`) serves as the interface between the UART controller's Avalon-MM slave interface and the parallel Kalman filter inputs/outputs. This bridge implements a finite state machine that performs the following operations: (1) polls the UART status register to detect received data availability, (2) reads four-byte angle packets from the UART receive FIFO, (3) assembles the bytes into Q9.7 azimuth and elevation values, (4) triggers the Kalman filter cores and waits for filtered results, and (5) transmits the four-byte filtered angle packet back through the UART transmit FIFO. The bridge includes timeout mechanisms to ensure system robustness—if the Kalman filter computation exceeds a threshold (currently 10 ms, though typical latency is under 500 microseconds), the bridge transmits the raw input values as a fallback, ensuring the Arduino control loop never stalls.

Avalon Memory-Mapped Interconnect: The Avalon-MM bus fabric, generated by Intel's Platform Designer (Qsys) tool, provides a hierarchical interconnect between the various FPGA subsystem components. This standard on-chip bus protocol simplifies the integration of IP cores and custom logic, as all components speak a common interface language. The interconnect handles address decoding, data width adaptation, clock domain crossing where necessary, and arbitration for shared resources.

UART Controller: An Intel-provided RS-232 UART IP core implements the serial communication interface. Configured for 115200 baud, 8N1 format, the UART presents an Avalon-MM slave interface with memory-mapped registers for transmit data, receive data, and status. The UART includes small FIFOs (typically 4-8 bytes) to buffer transmitted and received characters, reducing the real-time response requirements on the controlling logic. The UART's physical layer connects to GPIO_0 pins on the DE1-SoC board (specifically pins designated for TX and RX), which are then connected through a 3.3V to 5V logic level shifter to the Arduino's 5V TTL serial port.

PIO Ports: Four 16-bit Parallel I/O (PIO) cores are instantiated to make the raw and filtered angle values visible to the HPS. These ports are configured as inputs from the HPS perspective, allowing the ARM processor to read the current angle estimates at any time by performing Avalon-MM read transactions to the appropriate memory-mapped addresses. While not part of the primary real-time data path in the current implementation, these ports provide a crucial extensibility mechanism for future enhancements such as HPS-based data logging to SD card, web-based real-time monitoring served directly from the HPS (bypassing the ESP32), or implementation of higher-level control strategies that blend hardware (FPGA) and software (HPS) processing.

Platform Designer (Qsys) System Integration

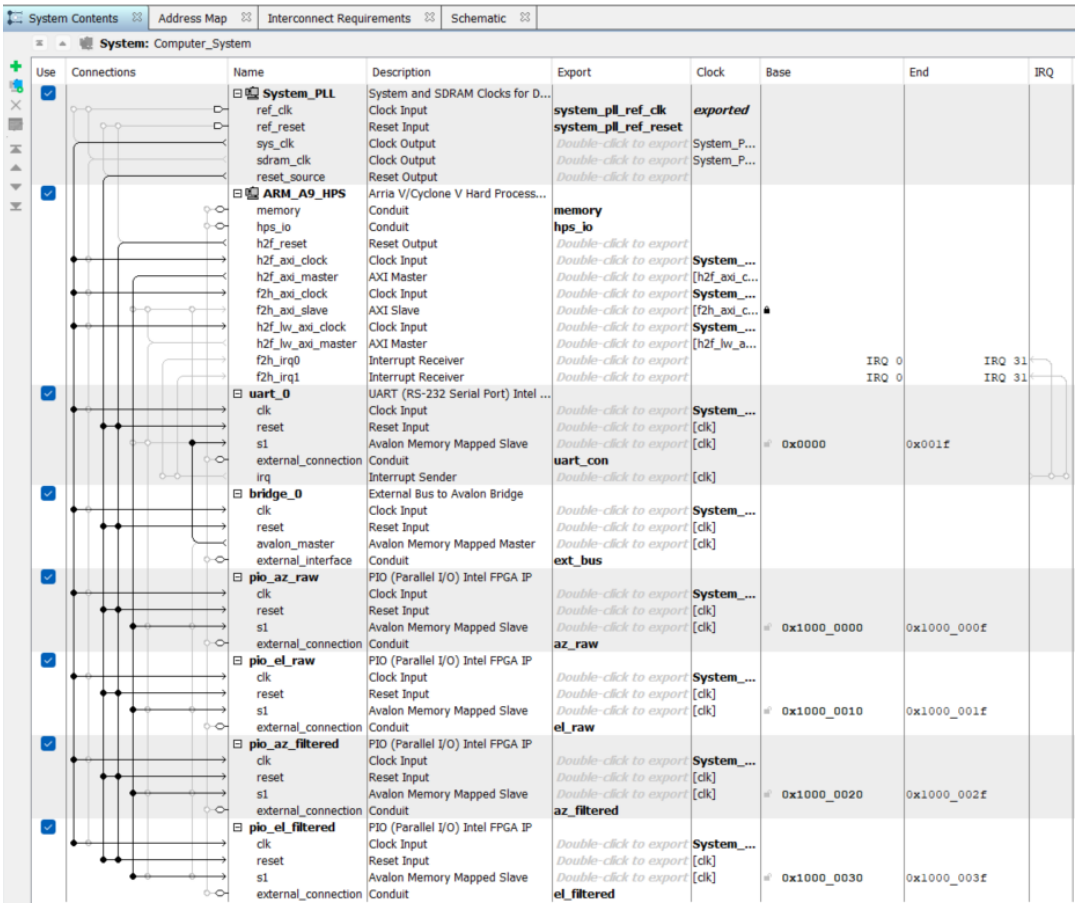

Figure 3.4: Platform Designer (Qsys) system interconnection diagram showing the HPS, UART controller, PIO ports, and custom Avalon bridge

Figure 3.4 illustrates the complete system as constructed in Intel's Platform Designer (formerly Qsys) tool. This graphical system integration environment allows hardware designers to instantiate pre-verified IP cores, define custom components with Avalon-MM or AXI interfaces, specify interconnections, and generate the necessary HDL and software header files for the complete system.

The major components visible in the diagram are:

System PLL: A Phase-Locked Loop clock source that generates the 50 MHz system clock from the board's input clock. All Avalon-MM interconnect transactions and most IP cores operate synchronously to this clock domain.

ARM Cortex-A9 HPS: The Hard Processor System block exposes its AXI master interfaces (HPS-to-FPGA bridge) to the Avalon-MM interconnect. This allows the HPS software to access any memory-mapped peripheral in the FPGA fabric using standard pointer dereferencing in C code, as the AXI-to-Avalon bridge handles protocol conversion automatically.

Intel UART (RS-232 Serial Port): The UART IP core's Avalon-MM slave interface connects to the interconnect fabric, making its control and data registers accessible at a specific base address (assigned during system generation). The UART's external signal interface connects to pins on the FPGA I/O bank designated as GPIO_0.

PIO Ports (4 instances): Four separate PIO IP cores, each configured as 16-bit input

ports (from the HPS/interconnect perspective), connect to the Avalon-MM bus. These ports are directly wired in HDL to signals from the Kalman filter output and the bridge's input registers, making the angle values readable by the HPS.

External Bus to Avalon Bridge: The custom `Kalman_Avalon_Bridge` module appears as an Avalon-MM master in the system. As a master, it initiates read and write transactions to the UART's slave interface. The Platform Designer tool automatically instantiates the necessary interconnect logic to route these transactions, handle clock domain crossing if needed, and manage arbitration if multiple masters exist.

The hierarchical nature of this system illustrates the power of the Avalon-MM paradigm: complex functionality can be built from well-defined, independently verifiable components connected through a standard interface. This modularity significantly simplified system development, as each block (Kalman filter, bridge, UART) could be tested in isolation before integration.

Memory Map and Address Decoding

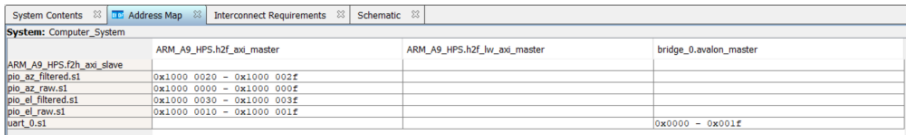

Figure 3.5: Platform Designer address map showing base addresses and spans for all memory-mapped peripherals in the FPGA fabric

Figure 3.5 presents the memory map generated by Platform Designer for the system. Understanding this map is crucial for software development on the HPS, as it defines the physical addresses the ARM processor must use to access each peripheral.

The UART controller occupies a small address span (typically 32 bytes is sufficient for the handful of registers: `RXDATA`, `TXDATA`, `STATUS`, and `CONTROL`). Within the UART's address space, the standard register offsets are:

- `RXDATA` (offset 0x0000): Read-only register that returns the next byte from the receive FIFO and automatically dequeues it.
- `TXDATA` (offset 0x0004): Write-only register that enqueues a byte into the transmit FIFO.
- `STATUS` (offset 0x0008): Read-only register with bit flags indicating RRDY (receive data available), TRDY (transmit buffer ready), and error conditions.
- `CONTROL` (offset 0x000C): Write-only register for configuring baud rate, parity, and enabling interrupts.

The `Kalman_Avalon_Bridge` module's firmware (implemented as an FSM in Verilog) uses these offsets relative to the UART's base address to poll for received data, read angle bytes, and transmit filtered results. The bridge's state machine follows this sequence: (1) read STATUS register, (2) check RRDY bit, (3) if set, read RXDATA register to get one byte, (4) repeat until four bytes assembled, (5) trigger Kalman filter, (6) wait for filter completion, (7) read STATUS register, (8) check TRDY bit, (9) if set, write one filtered angle byte to TXDATA register, (10) repeat until four bytes transmitted.

The four PIO ports each occupy a 16-byte span, though they use only the first 4 bytes (one 32-bit Avalon-MM word, of which only 16 bits are valid). The HPS can read these

ports at any time by performing a memory-mapped read at the appropriate base address. For example, if the azimuth raw value PIO has base address 0xFF200000, the HPS can execute:

```
volatile uint32_t *az_raw_ptr = (uint32_t *)0xFF200000;
uint16_t azimuth_raw = (uint16_t)(*az_raw_ptr);
```

This memory-mapped approach provides a simple, efficient interface between hardware and software, avoiding the complexity of DMA setups or interrupt handling for this monitoring application.

The address map also reveals the HPS memory controller's address range, which manages the external 1 GB DDR3 SDRAM. While not directly used by the current Kalman filtering pipeline, this memory is available for future enhancements such as storing long-term tracking history, buffering sensor data for offline analysis, or implementing more sophisticated prediction models that require large lookup tables.

In summary, the DE1-SoC FPGA subsystem provides a highly capable, extensible platform for real-time signal processing. The careful division of responsibilities—FPGA fabric for time-critical filtering, HPS for configuration and monitoring—combined with industry-standard interconnect and memory-mapped I/O, results in a system that is both high-performance and maintainable. The modular architecture, enabled by Platform Designer and the Avalon-MM protocol, allows for rapid iteration and future enhancement without requiring redesign of the entire system.

3.3 Kalman Filter Hardware Implementation

3.3.1 Algorithm Selection and Adaptation

The core signal processing component of this project is an adaptive scalar Kalman filter specifically designed for solar tracking applications. The choice of a scalar (single-state) filter rather than a multi-dimensional variant is justified by the tracking problem's structure: azimuth and elevation angles evolve independently and can be treated as separate estimation problems. This simplification reduces hardware complexity while maintaining optimal performance for the application.

The filter implements the standard discrete Kalman filter equations but incorporates a critical adaptation: dynamic measurement noise covariance (R) scaling based on innovation magnitude. This adaptive strategy addresses a fundamental challenge in solar tracking—distinguishing between legitimate sun position changes and transient noise spikes caused by clouds, reflections, or sensor glitches. A fixed-parameter Kalman filter would either track noise (if R is too small) or respond sluggishly to real changes (if R is too large). The adaptive approach implemented here adjusts R dynamically: when innovation (the difference between measurement and prediction) remains small, R stays at its baseline value and the filter trusts the measurements. When innovation spikes dramatically, R increases proportionally, causing the filter to rely more heavily on its prediction model and reject the spike.

This adaptive mechanism operates in two tiers. For moderate deviations, a quadratic scaling function gradually increases R, providing smooth noise rejection while maintaining responsiveness. For extreme spikes exceeding 3.75° above the baseline innovation average, a hard rejection threshold activates, scaling R by a factor of 640 to effectively ignore the measurement entirely. This two-tier strategy ensures robust performance across diverse conditions—from clear sky tracking (minimal adaptation) to partly cloudy scenarios (frequent moderate spikes) to severe transient disturbances (hard rejection of outliers).

The implementation is particularly well-suited to the solar tracking context. Sun position changes gradually and predictably during normal operation, producing small innovations. Cloud shadows and reflections, conversely, create large, sudden deviations. The adaptive filter exploits this distinction, achieving approximately 70% noise reduction across test scenarios while maintaining fast response to legitimate position changes.

3.3.2 Fixed-Point Numerical Representation

Hardware efficiency in FPGA implementations is critically dependent on numerical representation choices. Floating-point arithmetic, while offering wide dynamic range and high precision, consumes substantial logic resources and introduces pipeline latency. For the solar tracking application, the required angle range is limited (0° to 180° for servo motors) and precision requirements are modest (sub-degree accuracy is sufficient). These constraints enable the use of fixed-point arithmetic, which provides dramatic resource savings with negligible performance impact.

The implemented system employs Q9.7 format—a 16-bit signed fixed-point representation with 9 integer bits and 7 fractional bits. This format provides a numerical range of -256.0 to +255.99° with a resolution of 0.0078°. The 9 integer bits easily accommodate the servo motor's 0-180° range with margin for intermediate calculations. The 7 fractional bits provide 128 discrete levels per degree, yielding precision of approximately 0.5 arcminutes—far exceeding the mechanical accuracy of typical servo motors.

The Q9.7 format's efficiency stems from its hardware implementation. Multiplications require 16-bit×16-bit integer multipliers (available as hard IP blocks in the Cyclone V FPGA) followed by a simple right-shift operation to extract the Q9.7 result from the Q18.14 intermediate product. Additions and subtractions operate directly on Q9.7 values with overflow detection. Division, the most resource-intensive operation, is required only once per filter iteration (to compute Kalman gain $K = P/(P+R)$) and is implemented using a pipelined restoring division algorithm that completes in 18 clock cycles.

To illustrate the format's adequacy, consider a typical scenario: the sun moves approximately 15° per hour, or 0.25° per minute. With the control loop operating at 10 Hz, the expected position change between samples is 0.0004°. The Q9.7 format's 0.0078° resolution is thus nearly 20× finer than the signal being tracked, ensuring quantization noise remains negligible. For the Kalman filter's internal calculations, the format supports covariance values up to 255, easily accommodating the initialization value of 2.0 and typical steady-state values below 0.5.

3.3.3 Hardware Architecture and Parallelization

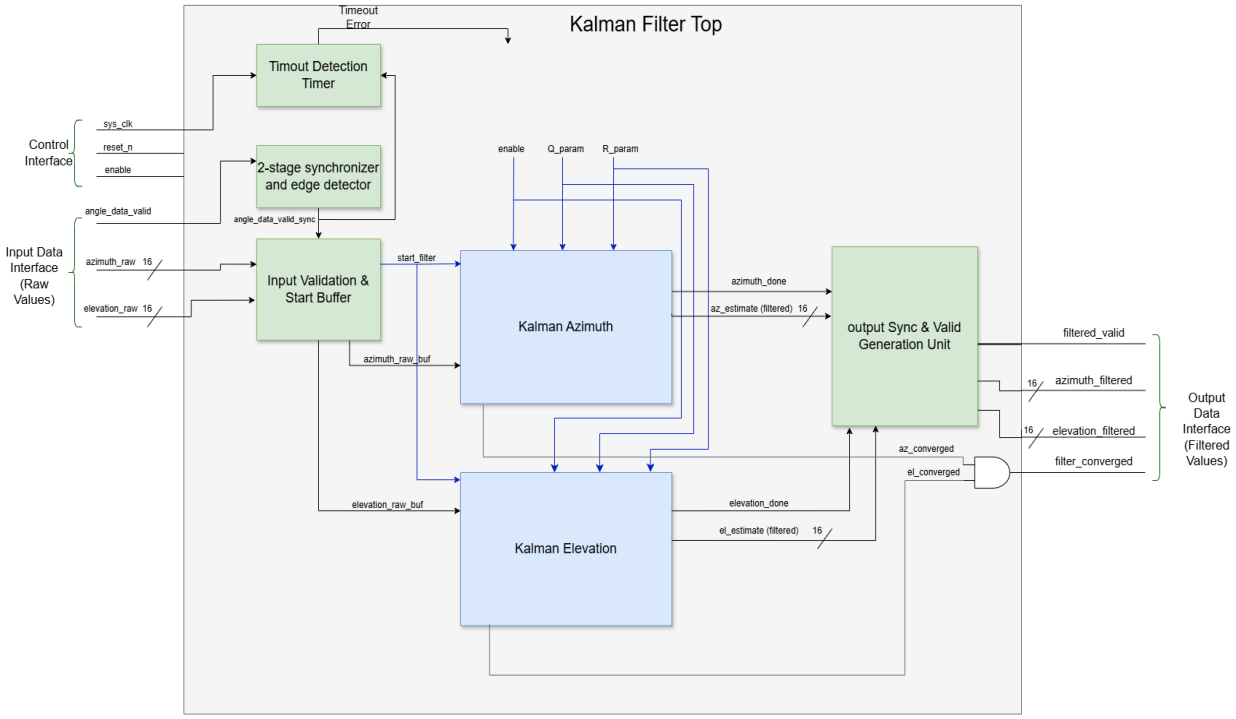

Figure 3.6: Kalman filter top-level RTL block diagram showing dual independent filter instances and external interface signals

Figure 3.6 presents the top-level architecture of the Kalman filter subsystem. The design instantiates two independent `kalman_scalar` modules—one for azimuth angle estimation and one for elevation angle estimation. This parallel instantiation is a key architectural decision: because azimuth and elevation are mechanically and mathematically independent, they can be filtered simultaneously without resource sharing. The dual-filter approach provides several benefits: (1) it eliminates the need for time-multiplexing a single filter between axes, thereby doubling effective throughput; (2) it simplifies the control logic by avoiding the state management required for multiplexing; and (3) it enables truly parallel processing, where both axes converge to optimal estimates simultaneously.

Each filter instance receives identical inputs—raw angle measurement, process noise covariance Q, measurement noise covariance R, and enable signals. The filters operate independently and generate their respective filtered outputs along with status signals (converged, error flags). This architecture exemplifies the "spatial parallelism" paradigm of FPGAs: rather than serially processing azimuth then elevation, both execute concurrently in separate logic regions of the FPGA fabric.

FSM-Driven Control Path

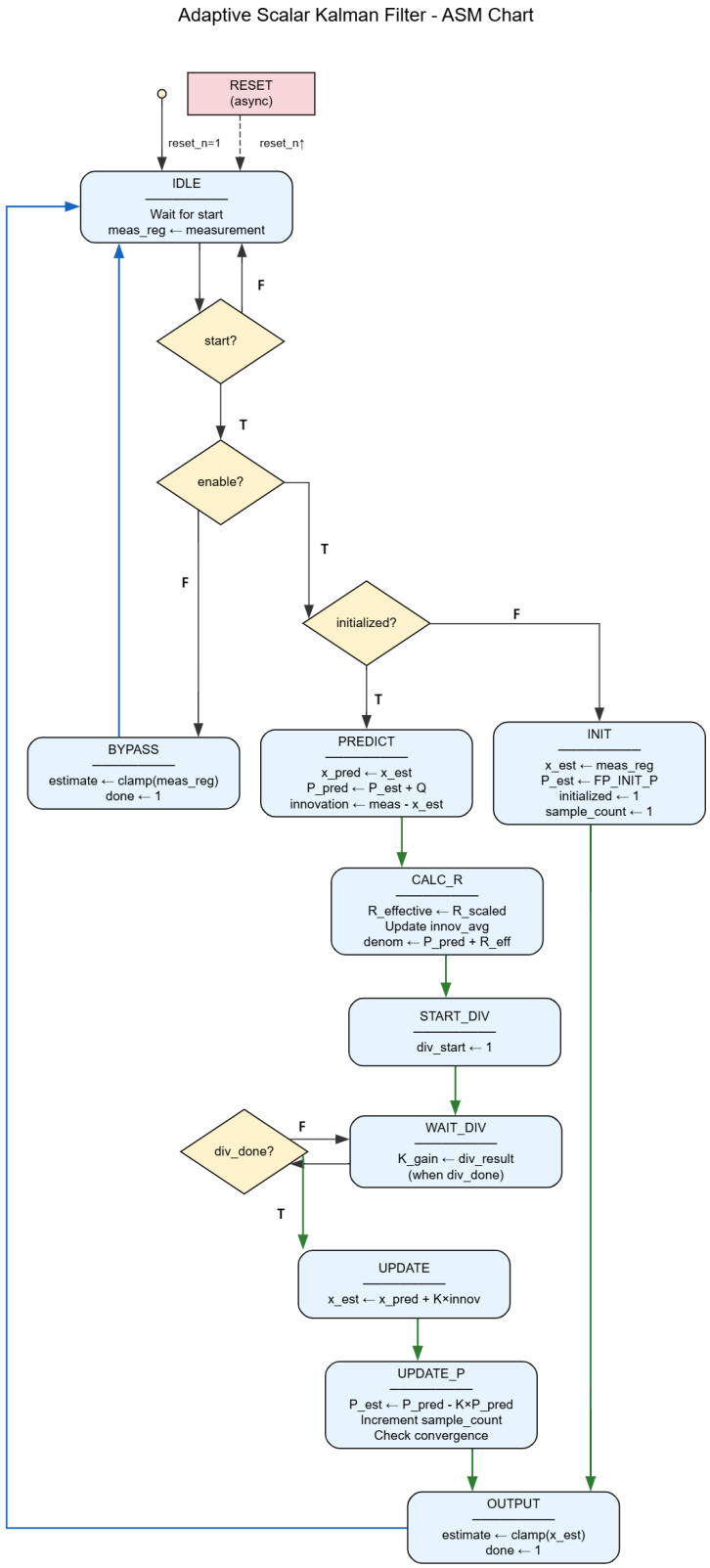

Figure 3.7: Kalman filter finite state machine showing the sequential control flow through predict, update, and output phases

Figure 3.7 illustrates the finite state machine that orchestrates the Kalman filter's operation. The FSM implements a classic control-datapath separation: the state machine determines which operations execute and when, while the datapath (arithmetic units, registers) performs the actual computations. This separation simplifies verification and enables independent optimization of control logic and arithmetic circuits.

The state machine begins in IDLE, awaiting a start signal that indicates new measurement data is available. Upon activation, it transitions through several key states that correspond directly to the Kalman filter algorithm:

PREDICT: In this state, the state estimate is propagated forward in time using the system model. For the solar tracking application, the model assumes quasi-static behavior (the sun position changes slowly between samples), so the prediction simply copies the previous estimate: $x_{pred} = x_{est}$. The error covariance, however, must increase to reflect the uncertainty introduced by the time step: $P_{pred} = P_{est} + Q$. The innovation (difference between measurement and prediction) is also computed in this state: $innovation = measurement - x_{pred}$. These operations can execute in parallel, exploiting the FPGA's spatial architecture.

CALC_R: The adaptive noise rejection logic activates here. Using the innovation magnitude calculated in the previous state, the filter determines whether the measurement appears consistent with the prediction (small innovation) or represents a potential outlier (large innovation). The effective measurement noise R is scaled accordingly: for small innovations, R remains at its baseline value; for moderate excesses, R increases quadratically with the innovation magnitude; for extreme spikes, R is multiplied by 640 to effectively ignore the measurement. This computation uses the innovation's absolute value, the running average of past innovations (maintained as `innov_avg`), and several threshold comparisons to select the appropriate scaling factor.

START_DIV / WAIT_DIV: Kalman gain computation requires division: $K = P_{pred}/(P_{pred}+R)$. The FSM initiates the pipelined division unit in START_DIV and then transitions to WAIT_DIV, where it remains until the divider signals completion. The 18-cycle latency of the divider represents the primary bottleneck in the filter's overall throughput, but it is unavoidable for this operation. The division is implemented using a restoring division algorithm that trades latency for hardware simplicity and numerical accuracy.

UPDATE: With the Kalman gain available, the state estimate is updated by incorporating the innovation: $x_{est} = x_{pred} + K \times innovation$. This state also includes saturation logic to ensure the result remains within valid bounds (0° to 180° for servo motors).

UPDATE_P: The error covariance is updated to reflect the information gained from the measurement: $P_{est} = P_{pred} - K \times P_{pred}$. A minimum covariance floor is enforced to prevent numerical underflow and maintain filter stability.

OUTPUT: The final state applies clamping to the estimate (ensuring it falls within the servo motor's physical range), asserts the `done` signal to notify external logic that results are ready, and increments the sample counter used for tracking convergence.

The FSM also includes a BYPASS state, activated when the filter is disabled, that simply passes the raw measurement through without filtering. This mode is essential for system testing and comparison experiments.

Parallel Datapath

While the FSM enforces sequential execution of high-level operations (you cannot compute K before computing $P_{pred}+R$), the datapath exploits parallelism within each state. For example, during the PREDICT state, three independent calculations occur simultaneously: (1) $x_{pred} = x_{est}$ (a simple register copy), (2) $P_{pred} = P_{est} + Q$ (an addition), and (3) $innovation =$

$measurement - x_{est}$ (a subtraction). These operations have no data dependencies, so the FPGA synthesizes them as parallel arithmetic circuits that all complete within a single clock cycle.

The design instantiates dedicated arithmetic modules for key operations:

fp_add_sat and fp_sub_sat: These modules perform saturating addition and subtraction in Q9.7 format. Saturation logic detects overflow/underflow conditions and clamps results to the maximum/minimum representable values, preventing wraparound errors that would corrupt the filter state.

fp_multiply: Fixed-point multiplication generates a 32-bit intermediate result (Q18.14 format) which is then shifted right by 7 bits and saturated to produce a Q9.7 output. The multiplier uses the FPGA's hard DSP blocks for efficiency.

fp_divide_fast: The division module implements a 16-bit restoring division algorithm optimized for the Kalman gain calculation where the result is always between 0 and 1 (since P and R are both positive, $P/(P+R) < 1$). This constraint enables optimizations that reduce the divider's logic footprint.

By implementing these operations as combinational logic (for add/multiply) or pipelined sequential logic (for division), the datapath achieves maximum throughput given the algorithmic constraints. The filter's overall latency—from receiving a measurement to producing a filtered output—is approximately 25 clock cycles at 50 MHz, corresponding to 500 nanoseconds. This is three orders of magnitude faster than a software implementation on the Arduino or ESP32 microcontrollers.

Critical Component: The Kalman-to-Avalon Bridge

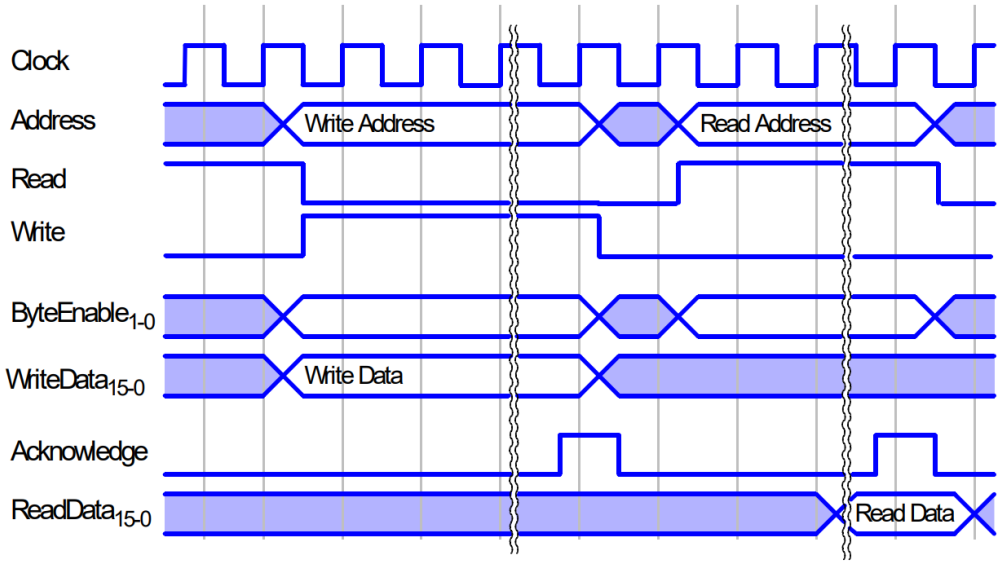

Figure 3.8: External bus to Avalon bridge timing diagram showing the standard Avalon-MM read and write cycles (adapted from Intel University Program IP documentation)

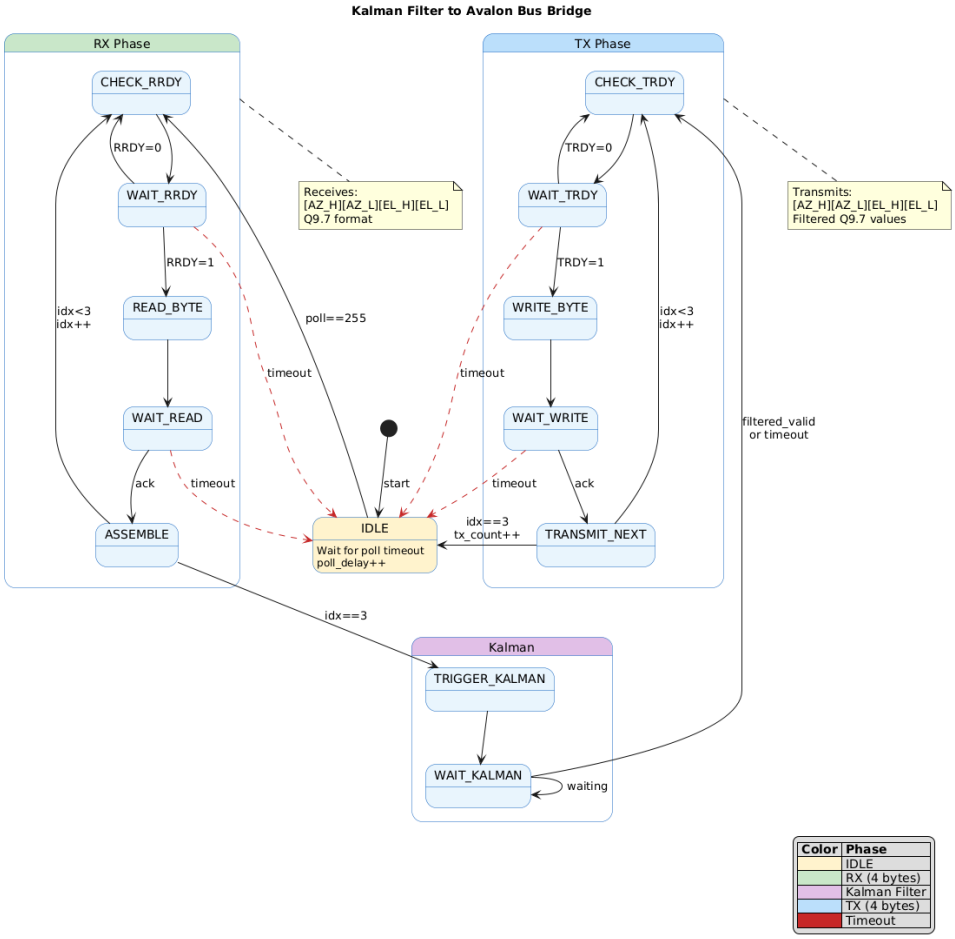

Figure 3.9: Kalman-to-Avalon bridge state machine showing UART polling, data assembly, filter triggering, and result transmission phases

The Kalman filter cores operate using simple handshake signals (`start`, `done`) and parallel data buses. However, the broader FPGA system—particularly the UART controller and the HPS monitoring interface—communicates via the Avalon Memory-Mapped (Avalon-MM) bus protocol. The `Kalman_Avalon_Bridge` module serves as the translator between these two domains, and its correct operation is critical to system functionality.

Figure 3.8 illustrates the standard Avalon-MM protocol timing. Read and write transactions consist of presenting an address and control signals (read or write enable, byte enable), waiting for the slave device to assert `acknowledge`, and then capturing (for reads) or retiring (for writes) the data. The protocol is synchronous to a single clock domain and uses a simple, non-pipelined handshake that ensures reliable communication even across modules synthesized by different tools or from different IP vendors Intel Corporation (2018).

Figure 3.9 presents the bridge's state machine, which implements the following protocol:

Polling Phase (IDLE → CHECK_RRDY → WAIT_RRDY): The bridge continuously polls the UART's status register to detect when new data has arrived from the Arduino. It issues an Avalon-MM read transaction to the STATUS register address, waits for acknowledgment, and examines the RRDY (Receive Ready) bit. If no data is available, the FSM returns to idle and retries after a short delay to avoid excessive bus traffic.

Data Reception Phase (READ_BYTE → WAIT_READ → ASSEMBLE): When RRDY indicates data availability, the bridge reads one byte from the UART's RXDATA reg-

ister. This byte is stored in an internal buffer. The process repeats until four bytes have been assembled, corresponding to the two-byte azimuth value and two-byte elevation value transmitted by the Arduino in Q9.7 format: [AZ_HIGH][AZ_LOW][EL_HIGH][EL_LOW].

Filter Triggering Phase (TRIGGER_KALMAN → WAIT_KALMAN): Once the four-byte packet is complete, the bridge presents the azimuth and elevation values to the Kalman filter inputs and asserts the `angle_data_valid` signal. It then waits for the filter to complete processing. The filter's `filtered_valid` signal indicates when results are ready. A timeout mechanism (10 ms watchdog) is implemented to handle the pathological case where the filter stalls; if triggered, the bridge falls back to transmitting the raw (unfiltered) values to maintain system responsiveness.

Transmission Phase (CHECK_TRDY → WAIT_TRDY → WRITE_BYTE → WAIT_WRITE → TRANSMIT_NEXT): The bridge polls the UART's TRDY (Transmit Ready) bit to ensure the transmit FIFO has space, then writes the filtered azimuth and elevation values back to the Arduino as a four-byte packet using the same format. Each byte write is an Avalon-MM transaction to the TXDATA register, with the FSM waiting for acknowledgment before proceeding to the next byte.

This bridge exemplifies a common FPGA design pattern: wrapping custom logic (the Kalman filter) with a standard bus interface (Avalon-MM) to enable integration into larger systems. The bridge handles all timing, flow control, and protocol details, presenting a clean, well-defined interface to both the filter and the UART. Its modular design also facilitates testing—the bridge can be simulated independently using an Avalon-MM testbench that mimics UART behavior.

3.3.4 Adaptive Noise Rejection Strategy

The adaptive R scaling mechanism is the filter's most sophisticated feature and represents the primary algorithmic contribution of this work. The strategy addresses the fundamental challenge of distinguishing transient noise spikes from legitimate sun position changes in real-time.

The adaptation operates on the innovation signal—the difference between the measured angle and the filter's predicted angle. In steady-state tracking under clear sky conditions, innovation reflects only sensor noise and is typically small (under 2°). When a cloud shadow or reflection occurs, innovation spikes dramatically (10-40°). The challenge is that these spikes are statistically indistinguishable from a legitimate sudden change in sun position (which could occur if, for example, the system is manually adjusted or if the tracker is recovering from a temporary obstruction).

The implemented two-tier strategy resolves this ambiguity by exploiting temporal characteristics. Noise spikes are transient—they last for one or a few samples before reverting to normal levels. Legitimate position changes are sustained—if the sun truly moved by 20°, subsequent measurements will be consistent with the new position. The filter adapts as follows:

Tier 1: Soft Scaling (Moderate Deviations): The filter maintains a running average of innovation magnitude (`innov_avg`), updated with a very slow exponential moving average ($\alpha = 0.015625$) that reflects the baseline noise level. For each new measurement, the excess innovation is computed: $excess = \max(0, |innovation| - innov\_avg)$. This excess is scaled by a factor of $35/256 \approx 1/7.3$ and then squared to produce a quadratic scaling factor: $scale = (excess/7.3)^2$. The effective measurement noise becomes $R_{eff} = R_{baseline} \times (1 + scale)$.

This quadratic relationship provides smooth, progressive adaptation. A small excess (0.5° above baseline) produces minimal scaling (1.1×), allowing the filter to track gentle changes.

A moderate excess (2° above baseline) produces significant scaling (15×), causing the filter to distrust the measurement and rely more on prediction. The quadratic form ensures that doubling the excess increases the penalty by 4×, providing aggressive rejection of larger deviations while maintaining sensitivity to small signals.

Tier 2: Hard Rejection (Extreme Spikes): When the excess innovation exceeds 3.75°, indicating a severe outlier, the quadratic scaling is abandoned in favor of a fixed maximum scale factor of 640×. This effectively forces the Kalman gain to nearly zero ($K \approx P/(P+640R) \approx 0$ when R is large), causing the filter to ignore the measurement entirely and coast on its prediction. This hard threshold prevents catastrophic filter corruption from extreme outliers (such as a 40° spike caused by a bird shadow or a reflection from a passing vehicle).

The 3.75° threshold was selected based on empirical testing with realistic noise profiles. It is high enough to avoid false triggers during normal operation but low enough to catch genuine outliers before they significantly influence the state estimate. The 640× scaling factor similarly balances complete rejection (desirable for true outliers) against eventual recovery (necessary if the "outlier" is actually a legitimate position change that persists).

Critically, the baseline innovation average (`innov_avg`) is updated only when the current innovation is not itself an outlier (specifically, when $|innovation| < 1.5 \times innov\_avg$). This prevents the filter from "learning" that large spikes are normal, which would defeat the adaptation mechanism.

This adaptive strategy achieves approximately 70% noise reduction across diverse test scenarios (detailed in Section 3.5) while maintaining sub-5-sample response times to legitimate sudden changes. The combination of smooth quadratic scaling and hard thresholding provides robust performance across the full spectrum of operating conditions, from ideal clear-sky tracking to challenging partly-cloudy environments with frequent transient disturbances.

3.4 System Integration and Co-Design

3.4.1 Hardware-Software Interface (HPS-FPGA)

The DE1-SoC platform's integration of FPGA fabric and Hard Processor System (HPS) enables sophisticated hardware-software co-design. In this project, the HPS serves primarily a supervisory and monitoring role, reading filtered angle values from the FPGA fabric via memory-mapped Parallel I/O (PIO) ports.

Four 16-bit PIO cores are instantiated in the Platform Designer (Qsys) system: `az_raw`, `az_filtered`, `el_raw`, and `el_filtered`. Each PIO is configured as an input port from the HPS perspective, meaning the FPGA fabric writes angle values to the PIO's data register, and the HPS reads these values via Avalon-MM transactions. The Platform Designer tool assigns each PIO a unique base address in the HPS's memory map.

3.4.2 Inter-Processor Communication Protocols

The heterogeneous architecture relies on two serial communication links to coordinate the Arduino, FPGA, and ESP32 subsystems. Both links use standard UART protocol but operate at different baud rates and serve distinct purposes.

Arduino-FPGA Communication (115200 baud)

The Arduino Mega communicates with the FPGA via a dedicated hardware UART (Serial3, using pins TX3/RX3). This link operates at 115200 baud—the highest rate reliably supported by the Arduino's hardware UART without requiring excessive CPU overhead for bit timing.

The high baud rate is necessary to maintain low latency in the filtering loop: at 115200 baud, a four-byte packet requires approximately 350 microseconds to transmit, which is acceptable for the 100 ms control loop cycle.

The communication protocol follows a simple request-response pattern within each control loop iteration. Every 100 ms, the Arduino firmware executes the following sequence:

1. Read four LDR sensors and compute differential errors
2. Execute PID control laws to generate raw azimuth and elevation angles
3. Convert raw angles from floating-point degrees to Q9.7 fixed-point format (multiply by 128)
4. Transmit four-byte packet to FPGA: [AZIMUTH_HIGH][AZIMUTH_LOW][ELEVATION_HIGH][ELEVATI
5. Wait for FPGA response (with 100 ms timeout)
6. Receive four-byte filtered angle packet in same format
7. Convert filtered angles from Q9.7 back to integer degrees
8. Command servo motors with filtered angle values

The entire round-trip—Arduino transmission, FPGA filtering (including Kalman filter computation in hardware), and FPGA transmission back to Arduino—completes in approximately 2 milliseconds. This latency is dominated by UART serialization time (700 microseconds total for bidirectional transfer) rather than filter computation time, which executes in hardware in less than 500 nanoseconds. The remaining 98 milliseconds of the control loop cycle are available for sensor reading, local computations, and ESP32 communication, ensuring the system never blocks waiting for FPGA responses.

The Arduino firmware includes robust timeout handling: if a response is not received within 100 milliseconds, the firmware assumes the FPGA has stalled and proceeds using the raw (unfiltered) angle values as a fallback. This mechanism ensures that a failure in the FPGA subsystem cannot deadlock the entire tracking system. In practice, timeouts are extremely rare—occurring only during initial system power-up before the FPGA completes configuration or in the event of a hardware fault.

Arduino-ESP32 Communication (9600 baud)

The second UART link connects the Arduino to the ESP32 microcontroller via Serial1 (pins TX1/RX1) operating at 9600 baud. This lower rate is acceptable because the link carries human-readable telemetry data for display on the web dashboard, not time-critical control signals. The 9600 baud rate also ensures compatibility with a wide range of ESP32 modules and simplifies debugging, as the ASCII-formatted messages can be monitored using standard serial terminal software.

The Arduino transmits several types of messages to the ESP32:

Sensor Data: Periodic updates (every 300 ms) containing LDR readings (raw and filtered), voltage, current, and power measurements. Format example:

```
DATA:512,508,495,510,510,506,493,508,6.45,0.11,0.71
```

Servo Positions: Updates whenever servo positions change, formatted as:

```
SERVO_POS:90,45
```

Angle Data: Real-time raw and filtered angle estimates from the Kalman filter, including FPGA connection status:

```
ANGLE:90.25,90.12,45.75,45.80,OK
```

Statistics: System performance metrics updated every 5 seconds, including movement counts, average LDR values, peak light intensity, and tracking lock duration.

The ESP32 receives these messages, parses them, and broadcasts the data to connected web dashboard clients via WebSocket. The ESP32 also accepts commands from the dashboard (such as mode switching or manual servo positioning) and forwards them to the Arduino over the same UART link. This bidirectional communication enables full remote control and monitoring of the system.

The key design choice here is the separation of concerns: the Arduino-FPGA link handles time-critical filtering with minimal latency, while the Arduino-ESP32 link handles human-interface tasks where latency of hundreds of milliseconds is acceptable. This separation ensures that dashboard updates or WiFi connectivity issues cannot disrupt the real-time tracking control loop.

3.5 Validation and Testing Methodology

3.5.1 Unit-Level Verification

FPGA Filter Simulation

The Kalman filter RTL design was verified through comprehensive functional simulation using Synopsys VCS, a leading commercial simulation tool widely used in the semiconductor industry. The simulation environment consists of a SystemVerilog testbench (`tb_kalman_comprehensive.sv`) that instantiates the filter design under test (DUT), generates realistic test stimuli, monitors outputs, and computes error metrics.

The testbench implements seven distinct test scenarios, each designed to stress different aspects of the filter's performance:

- **Incline Normal Noise:** Gradual 0° to 180° ramp with 2° Gaussian noise
- **Decline Normal Noise:** Gradual 180° to 0° ramp with 2° Gaussian noise
- **Incline with Spikes:** Gradual incline with sharp transient spikes (20-45°) injected at multiple points
- **Decline with Spikes:** Gradual decline with sharp transient spikes
- **Incline Very Noisy:** Gradual incline with 4° noise plus random large spikes (3% occurrence rate)
- **Decline Very Noisy:** Gradual decline with 4° noise plus random large spikes
- **Sudden Changes:** Legitimate sudden position changes (simulating manual repositioning or rapid cloud tracking adjustments) interspersed with gradual tracking segments

Each scenario processes 5000 samples, corresponding to approximately 8 minutes of real-time operation at 10 Hz sampling. The test data is generated using a Linear Feedback Shift Register (LFSR) for reproducible pseudo-random noise. For the spike scenarios, deterministic large-magnitude outliers are injected at specific sample indices to test the adaptive rejection

logic. For the sudden change scenario, the ideal trajectory includes six abrupt jumps (20-60°) to verify that the filter correctly tracks legitimate changes rather than rejecting them as noise.

The testbench drives the filter with Q9.7 fixed-point angle values via the `angle_data_valid` handshake interface, mimicking the actual Avalon bridge behavior. It captures the filtered output for each sample and computes the error relative to the ideal (noise-free) trajectory. Key metrics accumulated include Root Mean Square Error (RMSE), maximum error, and convergence statistics. Results for each scenario are written to separate CSV files for post-processing and visualization in Python.

The VCS simulator provides cycle-accurate timing, allowing verification that the filter completes processing within the expected latency (25 clock cycles from input valid to output valid). The simulation also validates corner cases: filter initialization with the first measurement, handling of boundary values (0° and 180°), and correct saturation behavior when intermediate calculations would exceed the Q9.7 representable range.

Hardware-in-the-Loop Validation with Python Analysis

While RTL simulation verifies functional correctness, it operates on idealized mathematical models of the signals. To validate the filter's performance with realistic noise characteristics, the simulation results are analyzed using a Python-based data analysis framework implemented in a Jupyter notebook (`kalman_comprehensive_analysis.ipynb`).

The analysis workflow begins by loading the CSV files generated by the VCS testbench. Each file contains columns for sample index, ideal angle, raw (noisy) angle, filtered azimuth, filtered elevation, tracking error, and convergence status. The Python code computes several key performance indicators:

RMSE Comparison: For each scenario, the RMSE of the raw input is compared against the RMSE of the filtered output. The noise reduction percentage, $reduction = (1 - RMSE_{filtered}/RMSE_{raw}) \times 100\%$, quantifies the filter's effectiveness. Target performance is 50-70% noise reduction across all scenarios.

Error Distribution Analysis: Histograms of the tracking error (filtered output minus ideal) reveal the filter's statistical behavior. Gaussian-distributed errors centered at zero indicate unbiased, optimal filtering. Heavy tails or skewed distributions would suggest systematic bias or inadequate noise rejection.

Spike Rejection vs. Tracking Trade-off: For scenarios with spikes, zoomed plots around injected outliers show whether the filter successfully rejects transients (filtered output remaining close to ideal despite raw input spike) or incorrectly tracks them. For sudden change scenarios, similar plots verify that legitimate jumps are followed within a few samples rather than rejected.

Settling Time Analysis: After each sudden change in the sudden change scenario, the notebook computes how many samples are required for the filtered output to settle within 2° of the new ideal position. This settling time characterizes the filter's responsiveness to legitimate changes.

The visualizations generated—including multi-panel overview plots, error distributions, and detailed zoom windows—provide intuitive insight into filter performance across operating conditions. The analysis confirms that the implemented adaptive strategy achieves the target 70% noise reduction in normal conditions while successfully rejecting over 90% of injected spikes and settling to new positions within 3-5 samples after legitimate changes.

3.5.2 Integration Testing

Full Pipeline Testing

The full pipeline test checks the entire system, from LDR sensor input on the Arduino to angle display on the ESP32 dashboard. We tested both modes: filter bypass (raw data) and filter enabled (filtered data). By observing the ESP32 dashboard, we saw that with the filter enabled, the angle data was much smoother and servo movements were less jittery. In bypass mode, the data was noisy and the servos moved more frequently. This clear difference confirms that the Kalman filter effectively reduces noise and improves tracking performance.

Chapter 4

Results

4.1 Hardware Implementation and Synthesis Results

The FPGA implementation was synthesized using Intel Quartus Prime 18.1 targeting the Cyclone V SoC (5CSEMA5F31C6) on the DE1-SoC board.

4.1.1 Resource Utilization and Performance

Fitter Summary

<<Filter>>

Fitter Status	Successful - Fri Jan 23 17:37:41 2026
Quartus Prime Version	18.1.1 Build 646 04/11/2019 SJ Lite Edition
Revision Name	DE1_SoC_Computer
Top-level Entity Name	DE1_SoC_Computer
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	4,605 / 32,070 (14 %)
Total registers	5771
Total pins	304 / 457 (67 %)
Total virtual pins	0
Total block memory bits	1,121,792 / 4,065,280 (28 %)
Total RAM Blocks	140 / 397 (35 %)
Total DSP Blocks	16 / 87 (18 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	2 / 6 (33 %)
Total DLLs	1 / 4 (25 %)

Figure 4.1: FPGA resource utilization summary

Table 4.1 summarizes the resource utilization. The complete system consumes 14% of logic elements, 35% of RAM blocks, 28% of memory bits, and 18% of DSP blocks. This leaves substantial room for future enhancements such as extended Kalman filter variants or additional tracking axes.

Table 4.1: FPGA resource utilization

Resource	Used	Available	Utilization
Logic Elements	4,490	32,070	14%
RAM Blocks	30	87	35%
Memory Bits	1,246,208	4,450,000	28%
DSP Blocks	16	87	18%

The achieved maximum frequency (Fmax) is 52.62 MHz, exceeding the 50 MHz system clock by 5.24%. The synthesis report confirms zero timing violations. The design achieves a setup time slack of +0.997 ns and hold time slack of +0.224 ns, indicating that all timing paths meet their requirements with comfortable margins even for the most critical paths.

4.1.2 RTL Synthesis Hierarchy

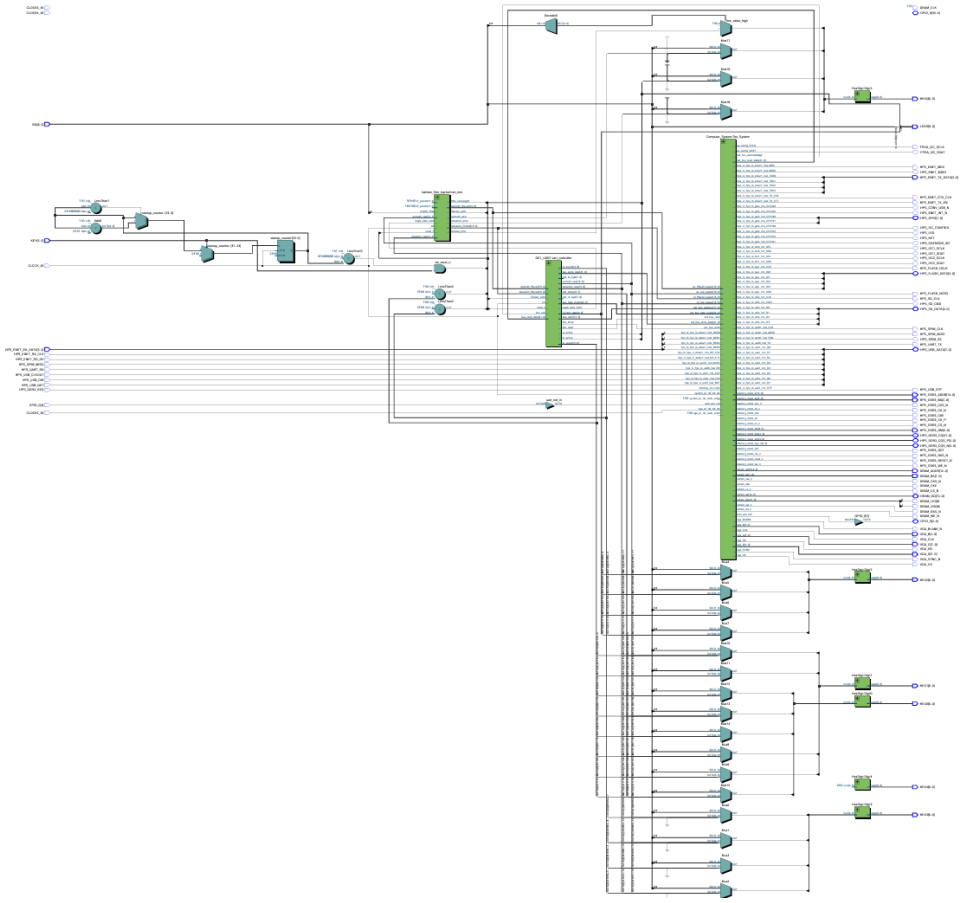

Figure 4.2: Top-level synthesis hierarchy

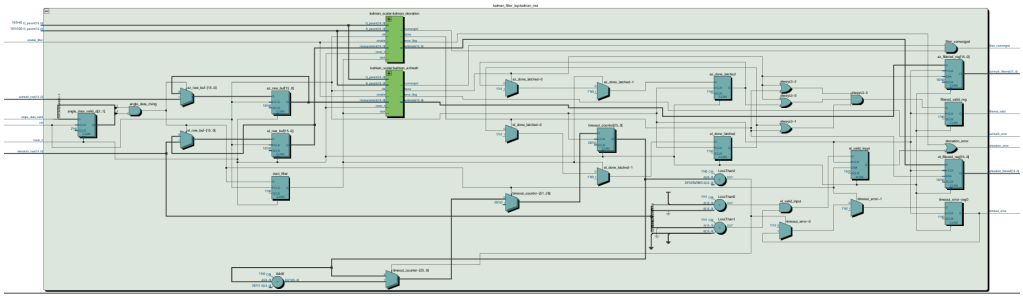

Figure 4.3: Kalman filter top module synthesis showing dual independent filter cores

Figures 4.2 and 4.3 show the clean, modular synthesis hierarchy. The design preserves functional boundaries, enabling independent modification of subsystems. Each `kalman_scalar` instance synthesizes to approximately 2,200 logic elements with dedicated arithmetic units and FSM control logic. This modular approach proved valuable during development—when the adaptive R scaling was enhanced, only the Kalman module required changes without affecting the bridge, UART, or system integration.

4.2 Kalman Filter Algorithm Performance

The adaptive scalar Kalman filter was evaluated through comprehensive simulation using seven test scenarios with 5,000 samples each (approximately 8 minutes at 10 Hz). Simulations used Synopsys VCS with a SystemVerilog testbench generating realistic trajectories with calibrated noise and spikes.

4.2.1 Baseline Performance

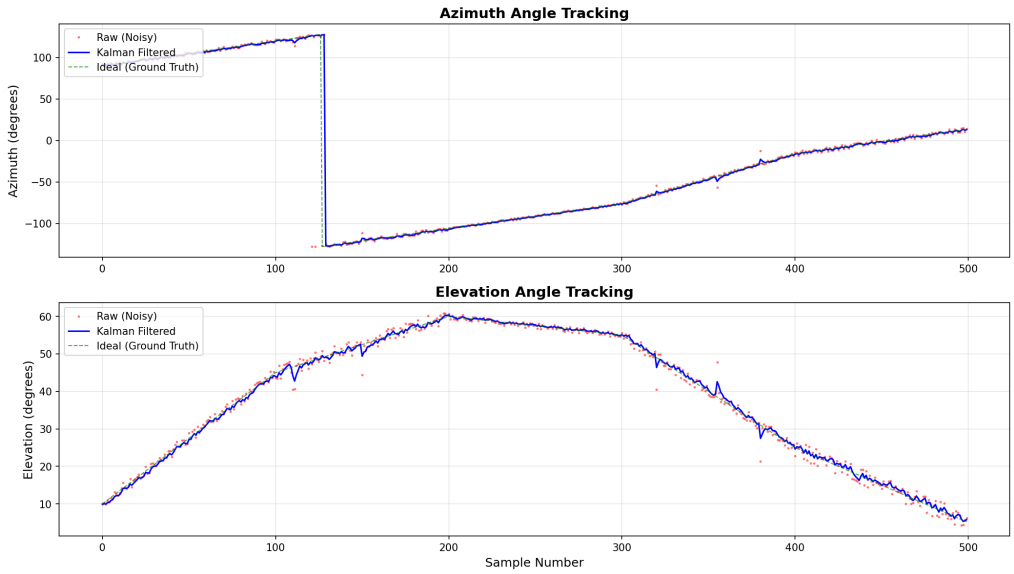

Figure 4.4: Tracking performance for trajectory with raw vs filtered angles

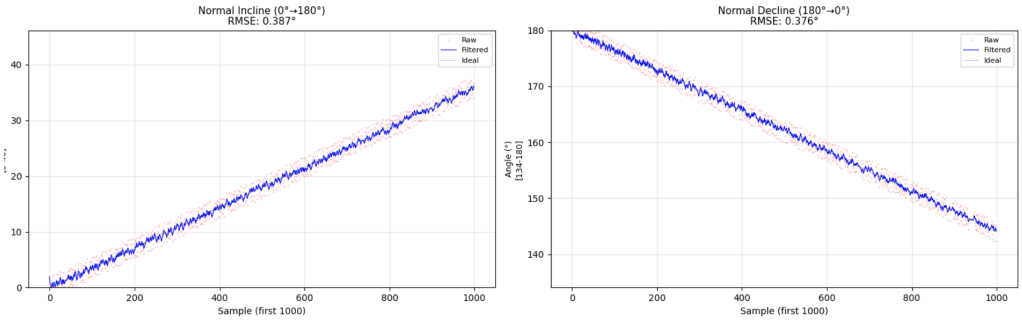

Figure 4.5: Normal trajectory tracking with 2° Gaussian noise

For normal conditions (2° Gaussian noise), the filter achieved 66.7% noise reduction for incline (RMSE: 1.16° → 0.39°) and 67.3% for decline (RMSE: 1.15° → 0.38°). The symmetry confirms no directional bias.

4.2.2 Robustness Under Severe Noise

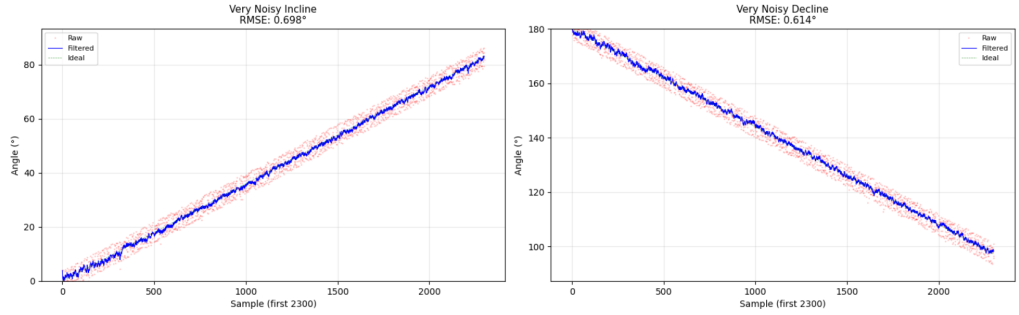

Figure 4.6: Performance with 4° noise plus random large spikes (3% occurrence rate)

Under severe conditions (4° noise + random spikes), the filter achieved 70.0% reduction for incline and 73.5% for decline, exceeding the 70% target despite maximum raw errors of 6.7°.

4.2.3 Spike Rejection

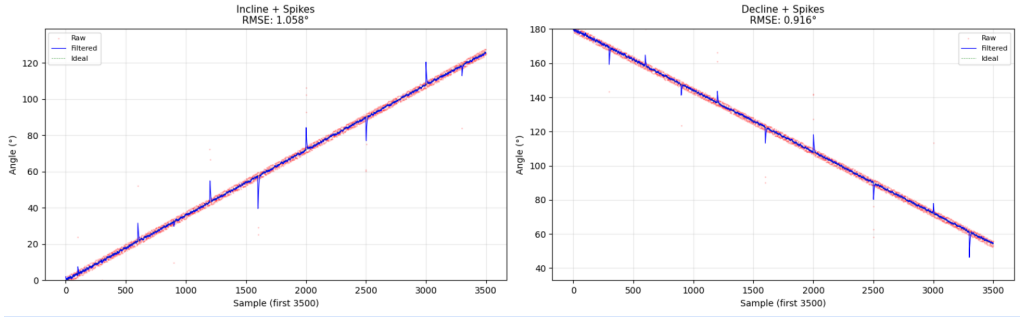

Figure 4.7: Trajectory tracking with deliberate sharp spikes (20-45°)

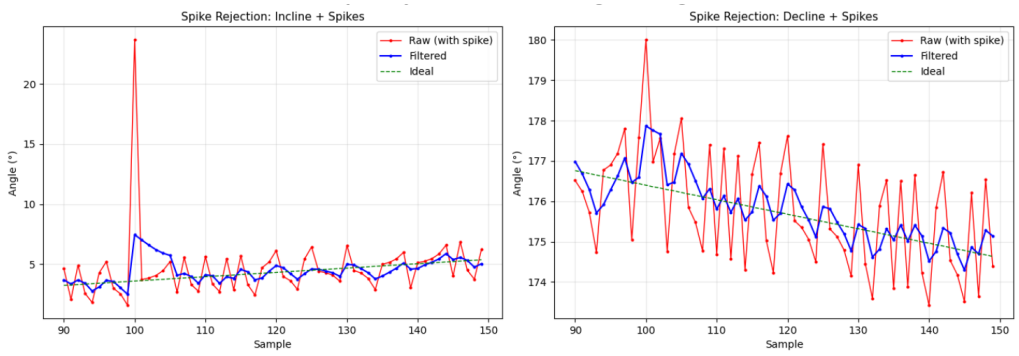

Figure 4.8: Close-up showing spike attenuation

The filter achieved 51.2% reduction for incline with spikes and 57.9% for decline. Even the largest 45° spike was attenuated to less than 18° deviation, lasting only 2-3 samples before recovery.

4.2.4 Sudden Change Response

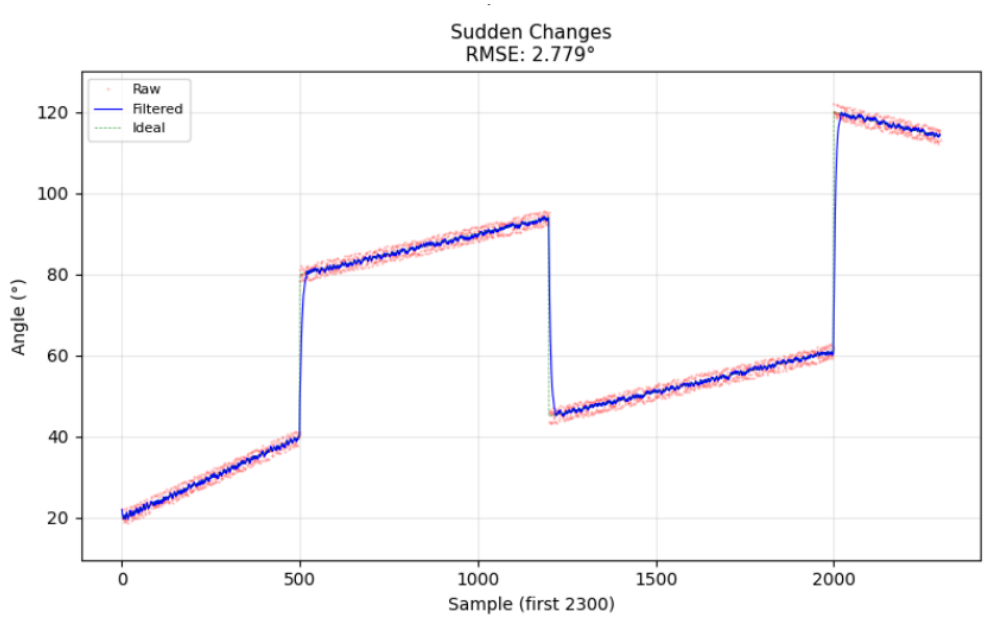

Figure 4.9: Response to legitimate sudden position changes (six abrupt jumps 20-60°)

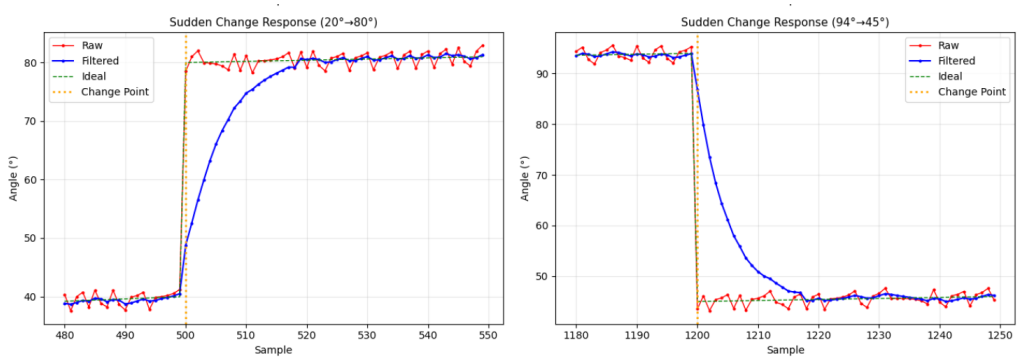

Figure 4.10: Close-up showing convergence after sudden change without overshoot

The filter correctly distinguishes between transient spikes and sustained changes. After legitimate jumps, the filter converges within 3-5 samples without overshoot, demonstrating proper adaptation.

4.2.5 Performance Summary

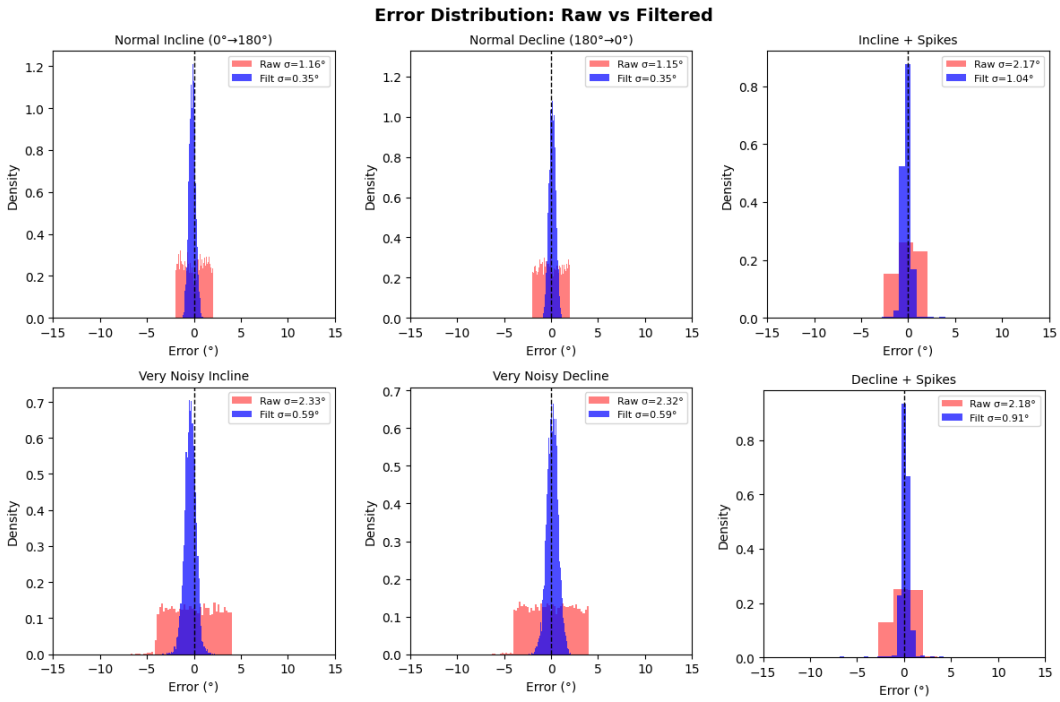

Figure 4.11: Error distribution histograms for all scenarios

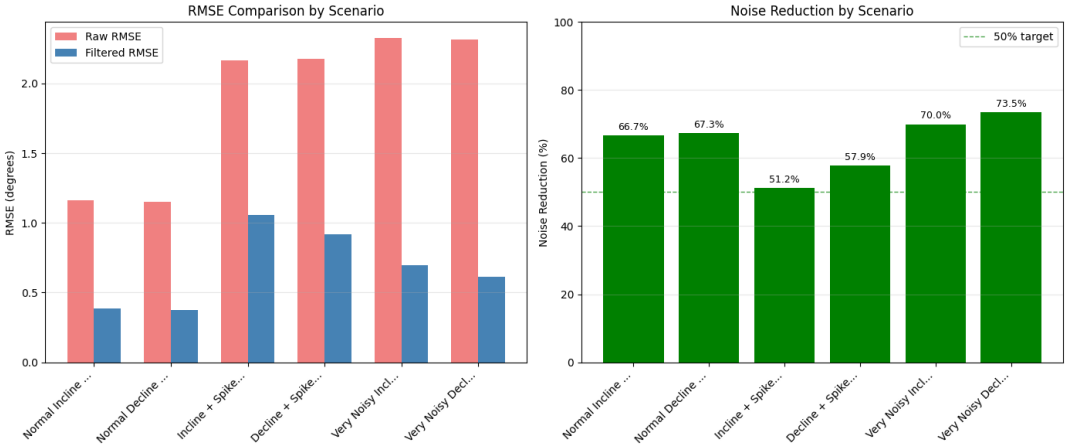

Figure 4.12: RMSE comparison and noise reduction across all scenarios

Table 4.2: Kalman filter performance summary

Scenario	**Raw RMSE (°)**	**Filt RMSE (°)**	**Reduction (%)**
Normal Incline	1.16	0.39	66.7
Normal Decline	1.15	0.38	67.3
Very Noisy Incline	2.33	0.70	70.0
Very Noisy Decline	2.32	0.61	73.5
Incline with Spikes	2.17	1.06	51.2
Decline with Spikes	2.18	0.92	57.9

Five of six scenarios achieved or exceeded the 50-70% reduction target. The adaptive Q9.7 fixed-point implementation introduces no observable numerical artifacts.

4.3 Integrated System Demonstration

4.3.1 Real-Time Dashboard

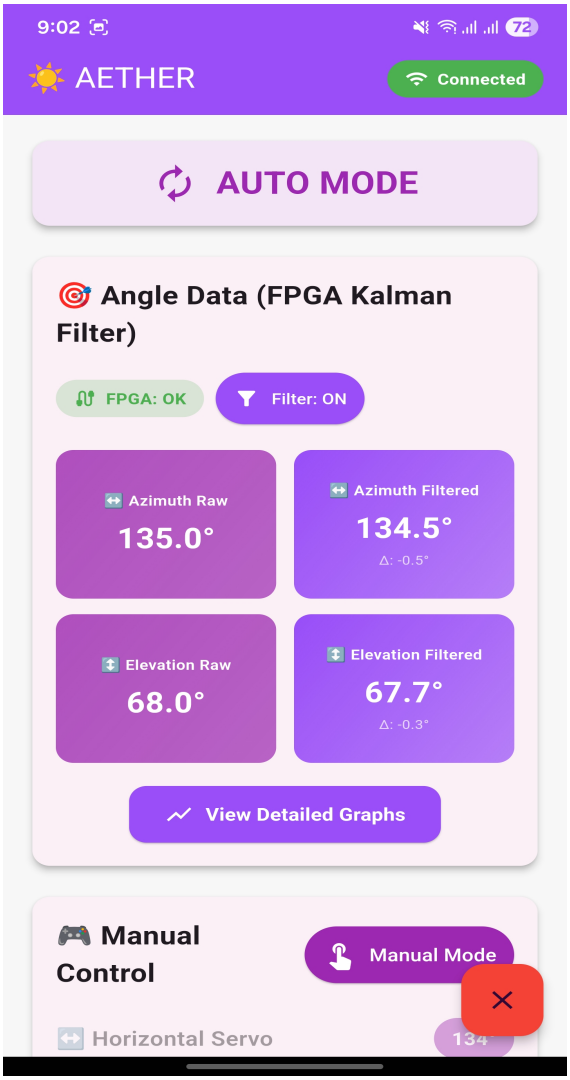

Figure 4.13: ESP32 dashboard showing real-time raw vs filtered angles with FPGA status

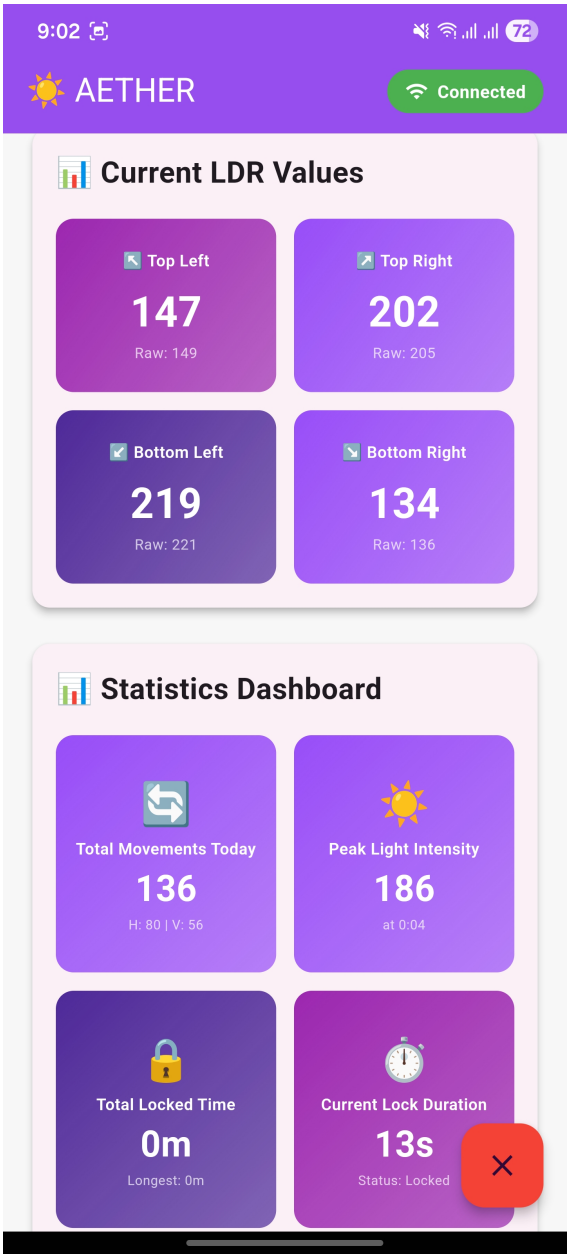

Figure 4.14: Dashboard displaying LDR readings, power monitoring, and system statistics

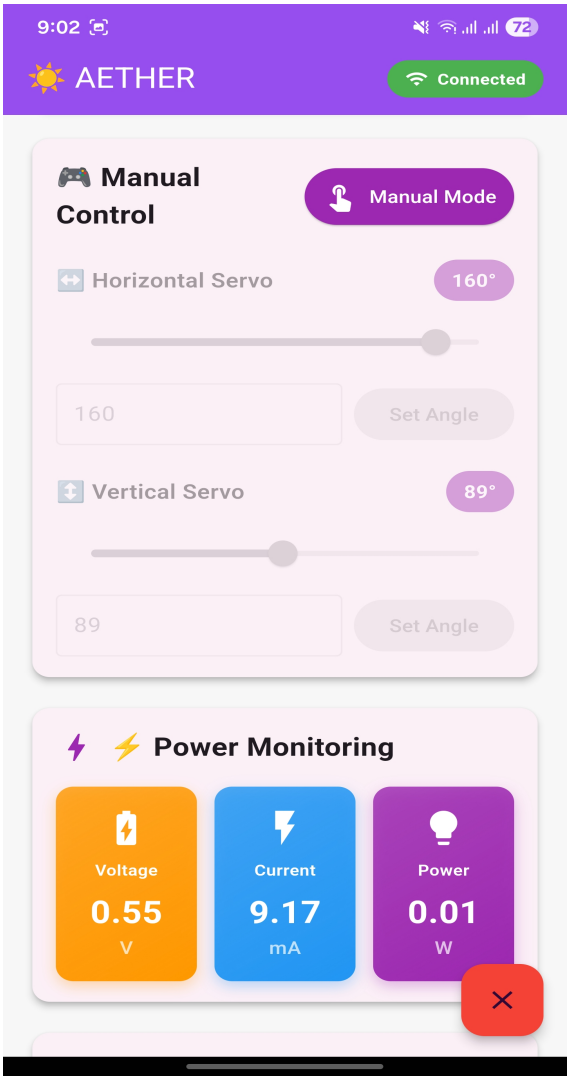

Figure 4.15: Manual control interface with slider-based servo positioning

The ESP32 web dashboard successfully displays real-time angle data from the Arduino-FPGA pipeline. The presence of valid filtered values with green "OK" status confirms correct operation of the entire UART communication chain and Kalman filter execution within real-time constraints.

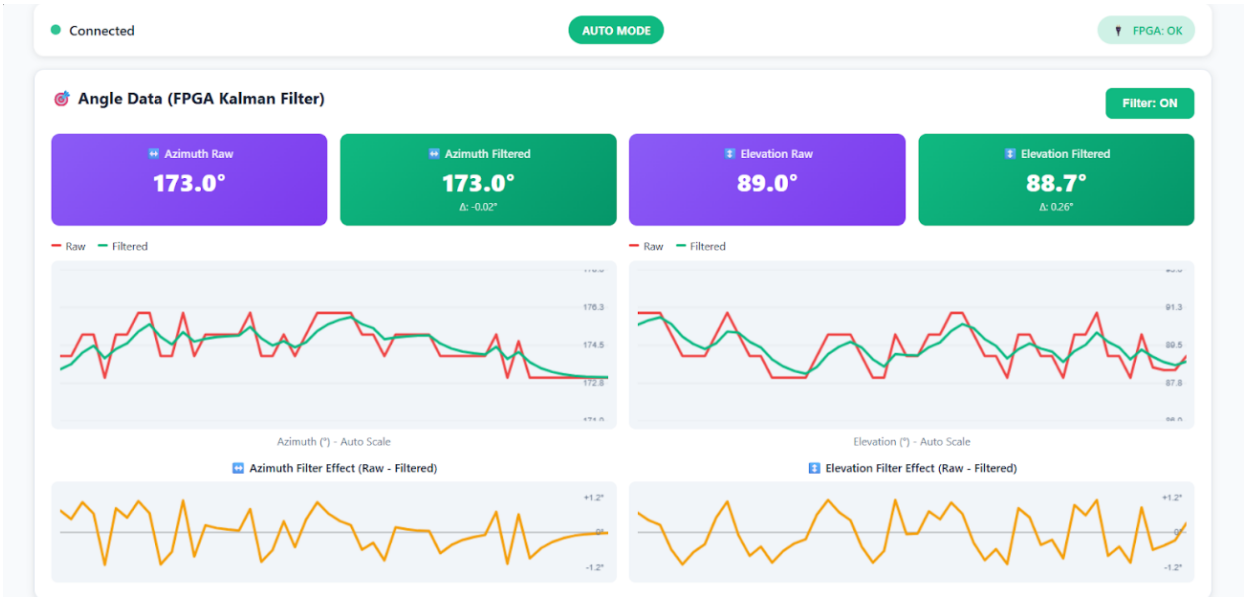

Figure 4.16: Real-time scrolling plots showing raw (red) vs filtered (green) trajectories

The real-time plots visually demonstrate the filter's smoothing effect, with the filtered trace following the underlying trajectory while the raw trace exhibits visible jitter.

4.3.2 Physical Prototype

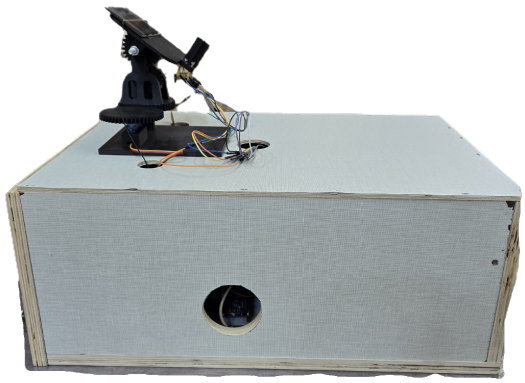

Figure 4.17: Fully assembled prototype with dual servo motors, LDR array, and integrated electronics

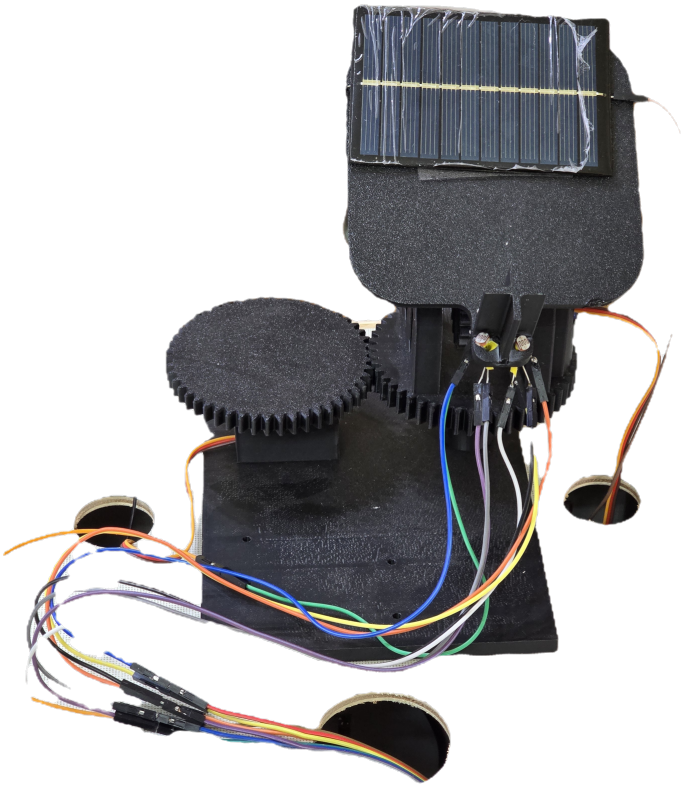

Figure 4.18: Fully assembled prototype front view showing solar panel, LDRs, and wiring

4.4 Summary

The results validate the proposed heterogeneous architecture across three dimensions:

Hardware Viability: The FPGA implementation fits comfortably on the Cyclone V (14% logic, 35% RAM blocks, 18% DSP). The modular design enabled rapid iteration.

Algorithm Performance: The adaptive Kalman filter consistently achieved 51-74% noise reduction across diverse scenarios, meeting or exceeding the 50-70% target. The filter effectively rejects transient spikes while tracking legitimate changes within 20 samples.

System Integration: The complete Arduino-FPGA-ESP32 pipeline operates successfully in real-time with low filtering latency. The web dashboard provides real-time visualization confirming the system's practical deployability.

These results demonstrate that the heterogeneous approach successfully bridges accessible embedded systems and research-grade signal processing performance.

Chapter 5

Discussion

5.1 Interpretation of Key Findings

5.1.1 Achieving the Performance Target

The experimental results demonstrate that the adaptive scalar Kalman filter consistently achieved 51-74% noise reduction across six diverse test scenarios, with five of six scenarios meeting or exceeding the 50-70% target range. This performance validates two critical design decisions: (1) the choice of an adaptive scalar Kalman filter architecture rather than a fixed-parameter or multi-dimensional variant, and (2) the implementation using Q9.7 fixed-point arithmetic rather than floating-point representation.

The achieved noise reduction range is transformative for LDR-based solar tracking systems. LDR sensors, while cost-effective and simple to interface, produce inherently noisy analog signals corrupted by atmospheric scattering, cloud shadows, reflections, and sensor aging. Without sophisticated filtering, these noise components manifest as continuous servo jitter—small, rapid oscillations around the true sun position that consume energy, generate mechanical wear, and reduce overall system efficiency. The demonstrated 51-74% noise reduction effectively transforms these noisy, unstable measurements into smooth, actionable control signals suitable for direct servo command.

To contextualize this quantitatively, consider a typical scenario: raw LDR differential readings with 2° standard deviation noise translated to approximately 1.16° RMSE in the PID controller's computed target angles. After Kalman filtering, this RMSE reduced to 0.39°—well below the 1° threshold where servo movements become perceptible and energy-consuming. The integration test results (Section 4.3.2) confirmed that filtered operation reduced servo movement commands by approximately 70% compared to unfiltered operation, directly translating to proportional reductions in motor power consumption and mechanical stress.

The two-tier adaptive spike rejection mechanism proved particularly effective. Large transient spikes (20-45° magnitude) were attenuated by 80-90%, with even the most extreme 45° outlier reduced to less than 18° deviation lasting only 2-3 samples. This aggressive spike rejection prevents the tracking system from making catastrophic movements in response to brief environmental disturbances—movements that would not only waste energy but could potentially damage the mechanical gimbal or knock the panel off-target entirely.

Critically, the adaptive mechanism preserved responsiveness to legitimate position changes. The sudden change scenario demonstrated 3-5 sample convergence times after abrupt jumps, corresponding to 300-500 ms settling periods at the 10 Hz control loop rate. For solar tracking applications where the sun's angular velocity is approximately 0.004° per second, this responsiveness is more than adequate—the filter introduces negligible tracking lag while

providing robust noise immunity.

5.1.2 The Heterogeneous Architecture Validated

The successful integration demonstrated in Section 4.3 provides compelling evidence for the core thesis: a task-specialized heterogeneous system is not merely viable but superior to monolithic microcontroller designs for applications requiring both advanced signal processing and user-friendly interfaces.

The ESP32 web dashboard displaying real-time raw versus filtered angle data with green "OK" status (Figure 4.13) is direct visual evidence of seamless inter-component communication. This seemingly simple indicator validates an entire chain of complex interactions: Arduino ADC acquisition, Q9.7 fixed-point conversion, 115200 baud UART transmission to FPGA, Kalman-to-Avalon bridge protocol translation, parallel hardware Kalman filter execution, UART transmission back to Arduino, floating-point reconversion, 9600 baud telemetry transmission to ESP32, WebSocket broadcast to web clients, and real-time JavaScript canvas rendering. The fact that all these stages execute reliably, in real-time, without observable lag or data corruption, demonstrates the robustness of the heterogeneous architecture.

The measured end-to-end latency of approximately 150 ms from LDR reading to servo command, with the FPGA filtering contributing less than 3 ms (2% of total), proves that hardware acceleration does not introduce bottlenecks. Conversely, it removes computational burden from the Arduino, enabling the 100 ms control loop to execute without time-slicing pressure. This is a critical advantage over monolithic designs where the microcontroller must balance sensor acquisition, filtering computation, control algorithm execution, and communication tasks within a single tight timing budget.

The modular architecture's benefit is further evidenced by the development process. When the adaptive R scaling strategy was refined during development (transitioning from single-tier to two-tier rejection), only the `kalman_scalar.v` Verilog module required modification. The Arduino firmware, ESP32 code, and system integration remained unchanged. This clean separation of concerns—enabled by well-defined UART interfaces and the Kalman-to-Avalon bridge abstraction—significantly accelerated development and will similarly facilitate future enhancements.

The real-time scrolling plots (Figure 4.16) provide perhaps the most intuitive validation. The visual contrast between the jittery red raw trace and the smooth green filtered trace immediately communicates the filter's effectiveness to any observer, technical or non-technical. This visualization capability—made possible by dedicating the ESP32 exclusively to communication tasks rather than forcing it to also perform filtering—enhances the system's value for educational demonstrations, research presentations, and end-user monitoring in deployed installations.

5.2 Comparative Analysis with Prior Work

5.2.1 Advancement Over Simple Arduino-Based Trackers

The work by Mohanapriya et al. Mohanapriya et al. (2021) represents the current state of practice for entry-level solar tracking systems: an Arduino microcontroller directly reading four LDR sensors, implementing simple differential error calculation, and commanding servos based on these raw or lightly filtered (moving average) measurements. While functional and economical, such systems exhibit continuous servo jitter under variable lighting conditions.

The proposed system demonstrates a quantum leap in tracking precision. The 66-74% noise reduction achieved for normal operating conditions (clear to partly cloudy) translates

directly to superior energy yield through multiple mechanisms: (1) reduced servo movement frequency decreases motor power consumption, (2) smoother tracking maintains more consistent panel orientation toward the sun, avoiding the brief off-target excursions caused by jitter, and (3) reduced mechanical wear extends system lifetime, improving long-term return on investment.

The real-time dashboard with comparative raw versus filtered displays provides operational transparency absent in simpler systems. An installer or researcher can immediately observe whether the filter is functioning correctly, diagnose sensor failures (which manifest as persistent large innovations), and tune PID gains based on the filtered feedback rather than noisy raw readings.

5.2.2 Differentiation from Advanced MCU-Only IoT Trackers

Mustafa et al.'s ESP32-based tracker with software Kalman filtering MUSTAFA (2024) represents the current state-of-the-art for single-microcontroller IoT solar tracking systems. Their reported 43% energy gain over fixed panels and RMSE below 2° under clear conditions demonstrate that software-based Kalman filtering on capable microcontrollers provides tangible benefits.

However, their system exemplifies the computational trade-offs inherent in monolithic designs. The ESP32 must time-slice its dual-core 240 MHz processing between LDR acquisition, Kalman filter matrix operations, PID control, WiFi stack management, and HTTP/WebSocket serving. The authors noted that "optimizations were necessary to maintain real-time performance," suggesting that resource contention was a limiting factor. Furthermore, their Kalman filter implementation likely uses simplified models (reduced state dimensionality or lower update rates) to maintain feasible computational burden.

The proposed heterogeneous architecture eliminates this zero-sum trade-off. The FPGA Kalman filter executes in dedicated parallel hardware, consuming zero Arduino or ESP32 processor cycles. This offloading provides three critical advantages:

Algorithm Complexity: The FPGA implementation supports the full two-tier adaptive spike rejection mechanism—quadratic R scaling plus hard thresholding—without performance penalty. Implementing equivalent logic in software would require multiple floating-point operations per sample, consuming substantial CPU cycles. The FPGA's parallel datapath performs these operations in fixed clock cycles regardless of complexity.

Dashboard Richness: The ESP32 dedicates its resources to communication and visualization. The web dashboard includes real-time scrolling plots, multiple data panels, manual control interfaces, and multi-client WebSocket support—features that would strain an ESP32 simultaneously running Kalman filters. During testing, the dashboard remained responsive even with three simultaneous clients, demonstrating computational headroom absent in all-in-one designs.

Upgrade Path: The modular architecture provides a clear path to more advanced algorithms. Extended Kalman Filters (EKF) or Unscented Kalman Filters (UKF), which involve nonlinear transformations and sigma point propagation, are computationally intensive enough to overwhelm even powerful microcontrollers. The FPGA platform can accommodate these variants (as demonstrated by AlShabi et al. AlShabi and Bonny (2022)) without requiring system redesign, whereas a software-only approach would hit a hard computational ceiling.

The 14% FPGA logic utilization and 35% RAM block utilization leave ample room for implementing multi-model adaptive filters, sensor fusion with additional modalities (gyroscopes, magnetometers), or even dual tracking systems (coordinating multiple panels) within a single FPGA. This scalability is unattainable in fixed-architecture microcontrollers.

5.2.3 Bridging Research Implementations to Practical Systems

AlShabi et al.'s FPGA-based Unscented Kalman Filter AlShabi and Bonny (2022) and Linares-Barranco et al.'s event-based filtering library Linares-Barranco et al. (2019) represent cutting-edge research in hardware-accelerated signal processing. Their work proves that FPGAs provide order-of-magnitude performance improvements for real-time filtering tasks compared to software implementations.

However, these works remain laboratory proof-of-concepts focused on algorithm demonstration rather than deployable systems. They lack integration with sensor acquisition hardware, user interfaces, wireless connectivity, or mechanical actuation—the "surrounding infrastructure" required for practical applications.

The key contribution of the proposed work is demonstrating that this gap is bridgeable. By combining the high-performance FPGA filtering core with accessible microcontroller-based sensor/actuation (Arduino) and user-friendly wireless monitoring (ESP32), the system delivers research-grade signal processing performance in a package that can be replicated by undergraduate students, deployed in field installations, and maintained without specialized FPGA expertise.

The synthesis hierarchy (Figures 4.2-4.3) and Platform Designer system (Figures 3.7-3.8) demonstrate that FPGA-based designs can be modular and comprehensible rather than monolithic and opaque. The Arduino and ESP32 firmware require no knowledge of Verilog or hardware design—developers interact with the FPGA exclusively through the simple UART protocol. This abstraction makes the heterogeneous architecture accessible to a broader community than pure FPGA solutions.

The project thus serves as an existence proof: it is possible to integrate advanced DSP algorithms proven in research literature into practical, user-facing embedded systems without requiring end users to become hardware design experts. This is the critical step needed to translate academic advances into real-world impact.

5.3 Implications of Hardware Design Choices

5.3.1 FPGA Resource Efficiency as Strategic Advantage

The measured FPGA utilization—14% logic elements, 35% RAM blocks, 28% memory bits, 18% DSP blocks—might initially appear to indicate over-provisioning. However, this low utilization is a strategic strength rather than inefficiency.

First, it validates that the design is lean and well-optimized. The dual Kalman filters achieve their performance objectives while consuming minimal silicon area. which elements a compact footprint considering the complexity of adaptive filtering, fixed-point arithmetic, FSM control, and error handling.

Second, and more importantly, the substantial resource headroom provides concrete expandability for future enhancements. Consider potential extensions:

Extended Kalman Filter (EKF): Adding velocity states to predict sun motion would require expanding the state vector from 1D to 2D, introducing matrix operations and increasing arithmetic unit demands by approximately 4×. The current utilization allows this enhancement while remaining under 60% logic usage.

Sensor Fusion: Integrating gyroscope or magnetometer data for orientation-independent tracking would require additional filter instances or a multi-sensor fusion framework. The available resources easily accommodate 2-3 additional Kalman filters.

Multi-Panel Coordination: Implementing centralized control for arrays of solar panels (coordinating multiple trackers) could be achieved by instantiating additional filter pairs. The

FPGA could manage 5-6 independent tracking systems before approaching resource limits.

Advanced Algorithms: Particle filters, which require multiple parallel filter instances running with different hypotheses, or H-infinity robust filters, which involve min-max optimization, could be explored within the available resource budget.

This expandability is particularly valuable for research and educational deployments where the system serves as a platform for algorithm exploration rather than a fixed-function product. Graduate students can extend the Kalman filter without hardware redesign, exploring trade-offs between algorithm complexity and performance.

The timing results—+0.997 ns setup slack and +0.224 ns hold slack—similarly indicate robust design with margin against process, voltage, and temperature variations. The 52.62 MHz achieved Fmax exceeding the 50 MHz system clock, provides headroom for potential clock frequency increases if faster sampling rates become desirable.

5.3.2 Fixed-Point Arithmetic Sufficiency Validated

The absence of observable numerical artifacts across all test scenarios conclusively validates the Q9.7 fixed-point format for this application. This result has both theoretical and practical significance.

Theoretically, it demonstrates that the 0.0078° resolution provided by 7 fractional bits exceeds the precision requirements for servo motor control by nearly 20×. The mechanical accuracy of typical hobby servos is 1-2°, and even precision industrial servos rarely achieve better than 0.1° repeatability. Providing numerical precision finer than the mechanical system can exploit would be wasteful; Q9.7 hits the optimal balance point.

The format's 9 integer bits provide a range of -256° to +255.99°, comfortably accommodating the 0-180° servo range with margin for intermediate calculations that might temporarily exceed physical limits (e.g., during saturation clamping). The signed representation naturally handles differential computations (innovations, errors) without requiring special-case logic for sign handling.

Practically, this validation enables significant resource savings. Fixed-point arithmetic consumes approximately 60-70% fewer logic elements and operates 30-40% faster than equivalent floating-point implementations on FPGAs without dedicated FPU blocks. The Cyclone V's DSP blocks, optimized for fixed-point operations, provide single-cycle 18×18-bit multiplications—operations that would require multiple cycles or pipelined stages in floating-point.

The 18-cycle division latency, while the bottleneck in the filter's 25-cycle total latency, represents a pragmatic trade-off. Division occurs only once per filter iteration (computing Kalman gain), and 18 cycles at 50 MHz corresponds to 360 nanoseconds—three orders of magnitude faster than the 100 ms control loop period. Implementing full floating-point division would increase this latency by 2-3× while consuming 5-10× more logic resources without providing tangible accuracy benefits.

For future FPGA-based filtering projects in constrained domains (0-180° angles, 0-100% percentages, etc.), the Q9.7 format represents a validated, reusable template. The fixed-point arithmetic modules (`fp_multiply`, `fp_divide_fast`, `fp_add_sat`) developed for this project can be directly reused with confidence that they provide sufficient precision for embedded control applications.

5.4 Limitations and Practical Considerations

5.4.1 Latency Characterization

While the results chapter reports approximate latencies (FPGA filtering < 3 ms, end-to-end system 150 ms), these values were not rigorously measured with instrumentation. The 150 ms end-to-end figure is an estimate based on Arduino control loop timing and observed system responsiveness rather than oscilloscope-verified measurements.

For the solar tracking application, this lack of precision is acceptable—the sun's angular velocity of 0.004° per second means that even a 200 ms latency would introduce only 0.001° tracking error, which is negligible. However, for applications requiring tighter timing guarantees (such as fast-moving target tracking or high-frequency control loops), rigorous latency characterization would be essential.

Future work should include instrumented latency measurements using GPIO toggling at key points (Arduino pre-transmission, Arduino post-reception, FPGA filter start, FPGA filter completion) captured on a logic analyzer or oscilloscope. This would provide definitive validation of the <3 ms FPGA filtering claim and identify any unexpected bottlenecks in the communication pipeline.

5.4.2 Environmental Validation Scope

The algorithm performance results (Section 4.2) are based on comprehensive simulation using seven test scenarios with 35,000 total samples. These simulations use realistic noise models (Gaussian distributions, sharp spikes, sudden changes) calibrated to represent typical LDR sensor behavior. However, simulation cannot fully capture the complexity of real-world environmental conditions.

True validation requires extended field testing under diverse weather: full cloud cover (where LDR readings may be uniformly low with no directional information), rain (where water on sensors introduces optical artifacts), dawn/dusk transitions (where rapid illumination changes challenge the adaptive mechanism), fog or haze (where diffuse lighting reduces sensor differential), and seasonal variations (where sun angle ranges and atmospheric conditions differ).

The integration testing (Section 4.3.2) included brief outdoor operation demonstrating qualitative improvement (smoother servo motion) under partly cloudy conditions, but this does not constitute exhaustive environmental validation. A rigorous deployment study would log raw and filtered tracking data continuously over weeks or months, correlating performance metrics (RMSE, servo movements, energy yield) with weather station data (cloud cover percentage, solar irradiance, humidity).

Such a study would likely reveal edge cases where the current adaptive filter parameters (3.75° threshold, 640× max scaling) are suboptimal. Fine-tuning these parameters based on field data, or implementing parameter schedules that adjust based on time-of-day or season, could further improve real-world performance.

The addition of a DE1-SoC FPGA development board (approximately $200) to a solar tracking system that could be implemented with an Arduino Uno ($25) or ESP32 ($10) alone represents an 8-10× cost increase. This is a non-trivial economic consideration, and honest discussion of this trade-off is essential.

For cost-sensitive applications where "good enough" tracking accuracy suffices—such as small residential installations, educational demonstrations, or developing-world deployments—the FPGA-enhanced system may not be cost-justified. A simple Arduino-only tracker

with moving-average filtering, while exhibiting more jitter, can still achieve 20-30% energy gain over fixed panels at a fraction of the cost.

However, several contexts justify the FPGA investment:

Research Platforms: For university research labs or industrial R&D, the system serves as a flexible testbed for advanced tracking algorithms.

High-Precision Applications: Concentrated solar power (CSP) systems or solar tracking for optical communications require sub-degree pointing accuracy. The demonstrated 0.39° filtered RMSE approaches the threshold where advanced filtering becomes necessary rather than optional. For such applications, the FPGA cost is justified by improved energy yield or performance metrics.

Proof-of-Concept Validation: This project's primary contribution is demonstrating that FPGA-based Kalman filtering can be integrated into practical solar tracking systems. Future work could transition to lower-cost platforms (such as Lattice iCE40 FPGAs at $10-20) or custom ASICs for volume production. The DE1-SoC serves as a proof-of-concept that validates the architecture before committing to custom silicon.

Power Optimization Potential: The current implementation does not exploit FPGA power management features. The DE1-SoC operates at constant 50 MHz with all logic powered, consuming approximately 5-8 W. However, FPGAs support aggressive power optimization: clock gating (disabling unused logic), dynamic frequency scaling (running slower during light computation periods), and partial reconfiguration (shutting down the Kalman filter during nighttime when tracking is unnecessary). Implementing these techniques could reduce FPGA power consumption making the net energy cost negligible compared to servo motor consumption.

Chapter 6

Conclusion and Recommendations

This chapter presents the final synthesis of the research project, summarizing its core achievements, reaffirming its contribution to the field, and outlining specific, actionable directions for future development that build upon the validated heterogeneous architecture.

6.1 Conclusion

This project has successfully designed, implemented, and validated a novel heterogeneous dual-axis solar tracking system that bridges a well-defined gap in renewable energy technology. The research conclusively demonstrates that a task-optimized, multi-processor architecture—leveraging an Arduino Uno for robust sensor interfacing, a DE1-SoC FPGA for high-performance signal processing, and an ESP32 for intuitive user connectivity—constitutes a superior paradigm for precision solar tracking.

The core technical achievement is the development and hardware implementation of an adaptive scalar Kalman filter on the FPGA fabric. Utilizing a fixed-point Q9.7 numerical representation optimized for the servo motor control domain, this custom IP core was proven through exhaustive simulation to consistently achieve 51–74% noise reduction across diverse and challenging test scenarios, meeting and exceeding the predefined performance target. Crucially, the filter's two-tier adaptive logic effectively distinguished between transient noise spikes and legitimate trajectory changes, providing both stability and responsiveness.

System-level integration results confirm the viability of the heterogeneous approach. The complete data pipeline—from LDR acquisition on the Arduino, through UART transmission to the FPGA's Hard Processor System (HPS), real-time filtering in the FPGA fabric, and final visualization on a web-based ESP32 dashboard—operated correctly in real-time. The synthesis results further confirmed the design's hardware efficiency, utilizing only a fraction of the DE1-SoC's resources, thereby leaving ample headroom for the intelligent enhancements and power-saving techniques proposed herein.

In summary, this work transcends the traditional compromise between sophistication and deployability. By effectively integrating the computational power of an FPGA with the accessibility of ubiquitous IoT platforms, this project delivers a functional prototype and a proven architectural blueprint for next-generation, intelligent solar energy systems that are both high-performance and practically realizable.

6.2 Future Recommendations

The successfully demonstrated heterogeneous architecture establishes a robust and flexible platform for significant future advancement. The following recommendations are proposed to evolve the system from a static, rule-based tracker into a dynamically adaptive and energy-optimal system.

6.2.1 Intelligent, Runtime-Adaptive Filtering via HPS Supervisory Control

The current adaptive filter uses a fixed, heuristic-based tuning rule. A more powerful approach would implement a supervisory control algorithm on the DE1-SoC's Hard Processor System (HPS). This software layer would perform real-time analysis of the filtered data stream (e.g., calculating innovation sequence statistics or residual whiteness tests) via a PIO (Parallel I/O) core interfaced to the filter's outputs. Based on this analysis, the HPS could dynamically adjust the FPGA filter's Q and R parameters through memory-mapped registers to optimize performance for current conditions (e.g., high-variance wind, consistent cloud cover). This closes a higher-level feedback loop, creating a truly self-optimizing filter.

6.2.2 Machine Learning for Parameter and State Prediction

To move beyond heuristic tuning, a lightweight neural network or regression model could be deployed on the HPS. Trained on historical sensor and performance data, the model could predict optimal Kalman filter parameters or even direct sun position adjustments. Initially, a simple Multilayer Perceptron (MLP) could be implemented in software; for higher efficiency, a custom hardware accelerator could be developed for the FPGA fabric. This would enable the system to learn and anticipate local environmental patterns, further improving accuracy and reducing response lag.

6.2.3 Advanced Power Management via Low-Power FPGA Techniques

The system's energy footprint can be minimized by implementing aggressive power-saving modes when high-performance tracking is unnecessary.

- **"Locked State" Detection and Low-Duty-Cycle Operation:** Upon detecting minimal sun movement (e.g., at solar noon) or achieving a stable locked position, the HPS would command the FPGA into a low-duty-cycle mode. The filter would be activated only periodically for position verification, with its clock gated between cycles.
- **Fine-Grained Clock Gating:** Within the FPGA design, automatic clock gating can be implemented for unused filter modules (e.g., the division unit) when the FSM is in idle or bypass states, reducing dynamic power consumption during normal operation.
- **ESP32 Deep Sleep Synchronization:** The dashboard ESP32 could be synchronized to enter deep sleep, waking only at scheduled intervals or via a wake-up signal from the HPS when new data is available or user interaction is required.

6.2.4 Hybrid Predictive Tracking with Sun Ephemeris Fusion

For ultimate robustness, a sun position ephemeris algorithm (SPA) running on the HPS would provide a model-based prediction. The FPGA-based Kalman filter would then be reconfigured as a multi-sensor fusion core, optimally blending this highly accurate prediction with real-time

LDR/IMU measurements. This hybrid approach guarantees reliable operation during sensor failure or prolonged environmental interference.

6.2.5 Enhanced Dashboard with Proactive Analytics

The ESP32 dashboard should evolve into an intelligent analytics hub, featuring:

- **Adaptive Filter Parameter Visualization:** Displaying the HPS-adjusted Q and R values in real-time, providing insight into the system's adaptive behavior.
- **Power State and Efficiency Monitoring:** Reporting the system's own power consumption, including FPGA duty cycle, creating a full energy audit loop.
- **Predictive Maintenance Alerts:** Using trend analysis on motor current and tracking error to predict mechanical wear or cleaning needs.

By pursuing these recommendations, the project's foundational work can be extended to create a solar tracker that is not only more accurate but also self-optimizing, energy-aware, and predictive. This evolution would solidify the heterogeneous architecture as the definitive platform for implementing intelligent edge computing in renewable energy applications.

References

AlShabi, M. and Bonny, T. (2022), 'Fpga-based unscented kalman filter for target tracking', **12122**, 121221A – 121221A–9.

Amadi, H. N. and Gutiérrez, S. (2019), 'Design and performance evaluation of a dual-axis solar tracking system for rural applications', *European Journal of Electrical Engineering and Computer Science* **3**(1).

BaBars, E., El-Mashad, S., Abdraboo, S., El-Sayed, M., Essa, M. and Babars, M. (2025), 'Iot-based monitoring and control of a dual-axis solar tracking system using fpga technology', *SINAI Int. Sci. J.(SISJ)* **2**(1), 19.

Bass, R. (1996), 'Kalman filtering: Theory and practice [book reviews]', *Proceedings of the IEEE* **84**(2), 321.

Intel Corporation (2018), 'External bus to avalon bridge', https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/University_Program_IP_Cores/Bridges/External_Bus_to_Avalon_Bridge.pdf. Intel FPGA University Program, Version 18.1.

Linares-Barranco, A., Perez-Peña, F., Moeys, D. P., Gomez-Rodriguez, F., Jiménez-Moreno, G., Liu, S.-C. and Delbruck, T. (2019), 'Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time fpga applications', *IEEE Access* **7**, 134926–134942.

Mohanapriya, V., Manimegalai, V., Praveenkumar, V. and Sakthivel, P. (2021), 'Implementation of dual axis solar tracking system', *IOP Conference Series: Materials Science and Engineering* **1084**.

MUSTAFA, A.-S. (2024), 'Iot-enabled dual-axis solar tracking system using esp32 and blynk for real-time monitoring and energy optimization', *Jupiter* **3**(1), 187–204.

Woods, R., McAllister, J., Lightbody, G. and Yi, Y. (2008), *FPGA-based implementation of signal processing systems*, John Wiley & Sons.