THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

FPGA-Based Kalman Filter for Real-Time Object Tracking with IMU and Radar Fusion

Thesis

William Mullany

45877481

2025

The University of Queensland

The School of Electrical Engineering and Computer Science

Abstract

This thesis presents a real-time object tracking system using a Kalman filter to fuse data from an IMU and ultrasonic sensors on a Xilinx Zynq-7000 SoC. A hybrid architecture was developed, with core filter computations targeted for FPGA acceleration and high-level processing managed by the onboard ARM Cortex-A9 processor. The system was designed to handle heterogeneous sensor noise and update rates through a linear Kalman filter framework.

Due to LUT and flip-flop resource limitations on the Digilent Cora Z7, full hardware implementation was not feasible; instead, a software-based filter was validated in real time. The system demonstrated accurate tracking, effective noise rejection, and stable performance under simulated and static experimental scenarios. Results show the feasibility of embedded Kalman fusion, with profiling indicating that hardware acceleration would offer further improvements. Key limitations include static filter parameters, polling-based sensor access and redundancy, and the need for further model and tuning refinement. Future work should target a larger FPGA, adaptive filtering, and integration of true radar sensors.

Declaration by author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my higher degree by research candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the policy and procedures of The University of Queensland, the thesis be made available for research and study in accordance with the Copyright Act 1968 unless a period of embargo has been approved by the Dean.

William Daniel Mullany
w.mullany@uq.edu.au

June 18, 2025

Prof Michael Bruenig
Head of School
School of Electrical Engineering and Computer Science
The University of Queensland
St Lucia, QLD 4072

Dear Professor Bruenig,

In accordance with the requirements of the degree of Bachelor of Engineering (Honours) in the division of Mechatronic Engineering, I present the following thesis entitled

"FPGA-Based Kalman Filter for Real-Time Object Tracking with IMU and Radar Fusion".

This work was performed in under the supervision of Dr. Matthew D'Souza. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely,

William Daniel Mullany

Acknowledgments

I would like to thank my supervisor, Dr. Matthew D'Souza, for his support throughout this project. His feedback was consistently helpful in guiding the technical direction, and his generosity in lending me his hardware made practical development possible. I'm especially grateful for the time and resources he provided, which were critical to both the implementation and validation of the system. His input was instrumental at every stage of the work.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

List of Abbreviations

Abbreviation	Description
CPU	Central Processing Unit
MMU	Memory Management Unit
ALU	Arithmetic Logic Unit
DMA	Direct Memory Access
FPGA	Field Programmable Gate Array
SoC	System on Chip
HDL	Hardware Description Language
IC	Integrated Circuit
IMU	Inertial Measurement Unit
KF	Kalman Filter
DOF	Degrees of Freedom
US	Ultrasonic Sensor
LUT	Look Up Table
FF	Flip Flop
AXI	Advanced eXtensible Interface
PL	Programmable Logic
PS	Processing System
I/O	Input/Output
PCB	Printed Circuit Board
RTL	Register Transfer Level
IP	Intellectual Property (core)

List of Symbols

Symbol	Description
x_k	State vector at time step k
$\hat{x}_{k\|k}$	Posterior (updated) state estimate at time k
$\hat{x}_{k\|k-1}$	Prior (predicted) state estimate at time k given $k-1$
$P_{k\|k}$	Posterior estimate error covariance
$P_{k\|k-1}$	Prior estimate error covariance
A_k	State transition matrix
B_k	Control input matrix
u_k	Control input vector
Q_k	Process noise covariance matrix
z_k	Measurement vector
H_k	Observation matrix
R_k	Measurement noise covariance matrix
y_k	Innovation (measurement residual), $y_k = z_k - H_k\hat{x}_{k\|k-1}$
S_k	Innovation covariance, $S_k = H_k P_{k\|k-1} H_k^\top + R_k$
K_k	Kalman gain, $K_k = P_{k\|k-1} H_k^\top S_k^{-1}$
I	Identity matrix

Chapter 1

Introduction

1.1 Motivation

Accurate, low-latency object tracking is fundamental in modern embedded systems, particularly in applications such as autonomous vehicles, defense systems, robotics, and aerospace navigation. These systems operate in dynamic environments and must continuously estimate object states—including position, velocity, and direction of motion—based on noisy and often incomplete sensor data.

A single sensor modality is rarely sufficient to provide the accuracy and robustness required in these scenarios. Inertial Measurement Units (IMUs) offer high update rates and capture fine-grained acceleration and rotation information, but they are susceptible to integration drift and short-term noise. Radar systems, on the other hand, provide robust absolute position data and operate reliably in low-visibility environments, but with lower update rates and coarser resolution.

To overcome the limitations of individual sensors, **sensor fusion** is employed to combine complementary data from heterogeneous sources. This creates a more accurate and stable estimate of the system state than any single sensor can provide. However, this fusion process introduces its own challenge: each sensor type exhibits different noise characteristics, sampling rates, and biases. Managing these discrepancies in real time, especially in embedded environments, is a non-trivial task.

The Kalman filter presents a mathematically rigorous framework to address this challenge. Its recursive nature allows it to efficiently fuse multiple sensor inputs in real time, while optimally accounting for their individual noise statistics. Moreover, the Kalman filter's formulation explicitly incorporates uncertainty modeling, making it ideal for fusing asynchronous and noisy data from disparate sensors.

Despite its advantages, a fully software-based implementation of a Kalman filter can become a computational bottleneck, especially when matrix operations dominate processing time. To meet real-time performance constraints and deterministic timing guarantees, hardware acceleration through Field Programmable Gate Arrays (FPGAs) is a compelling solution.

1.2 Problem Area and Solution

Sensor fusion systems are critical for accurate object tracking, but implementing them in real-time embedded environments poses several design and computational challenges. Key issues include:

- **Heterogeneous sensor characteristics:** IMUs generate high-frequency data with high short-term noise, while radar sensors provide sparse, more reliable measurements. Fusing such data requires a filter that dynamically balances conflicting inputs.
- **Differing noise profiles:** Sensor measurements must be fused while accounting for varying noise covariances, sensor delays, and biases.
- **Latency and computational load:** Kalman filtering involves linear algebra operations, including matrix multiplications and inversions, which become computationally intensive when performed at high update rates or in multidimensional systems.

This thesis proposes a hybrid hardware-software solution, in which the Kalman filter is implemented on an FPGA for computational efficiency, while a general-purpose processor (ARM Cortex-A9) handles sensor interfacing and high-level control. The hardware acceleration enables low-latency, deterministic execution of the filter's core matrix operations, while maintaining adaptability through software configuration. This architecture enables real-time fusion of radar and IMU data on a Zynq-7000 SoC platform.

1.3 Aims and Objectives

The central aim of this thesis is to design, implement, and evaluate a real-time object tracking system based on a Kalman filter for fusing radar and IMU data on an FPGA. The specific objectives are:

- Formulate a system and measurement model appropriate for radar and IMU sensor fusion.
- Design a Kalman filter architecture suitable for hardware implementation, with optimized numerical routines for performance and stability.
- Integrate radar and IMU sensors into the system and develop a fusion strategy that accounts for their differing noise models and update frequencies.
- Implement the filter core using FPGA hardware acceleration and evaluate its timing, latency, and resource utilization.
- Compare the performance of the hardware-accelerated filter with a baseline software-only implementation in terms of accuracy, timing, and energy efficiency.
- Validate the tracking performance using both simulated and real-world scenarios, with emphasis on filter convergence and noise rejection.

1.4 Scope

This thesis is focused on the implementation of a linear Kalman filter-based tracking system using radar and IMU fusion on an FPGA platform. The following boundaries define the scope:

- The fusion algorithm is based on the standard discrete Kalman filter; nonlinear variants (e.g., EKF, UKF) are beyond the current scope.
- Only radar and IMU sensors are considered; vision and lidar modalities are excluded.
- The target platform is a Xilinx Zynq-7000 SoC (specifically, Digilent Cora Z7), leveraging both the programmable logic and embedded ARM cores.
- Hardware implementation focuses on matrix multiplications and inversions as part of the Kalman filter update cycle.
- Experimental validation includes system identification, synthetic test cases, and limited real-world tracking scenarios.
- Advanced robustness features such as adaptive filtering or fault detection are not addressed.

This thesis uses ultrasonic sensors (US) to simulate the effects of the radar positioning, due to size and cost constraints.

Chapter 2

Literature Review

This chapter provides an overview of the core technologies and methodologies underpinning this project. It begins with a discussion of the Kalman Filter, outlining its recursive estimation process and relevance to real-time tracking. Following this, the role of Field Programmable Gate Arrays (FPGAs) is examined, emphasising their suitability for high-performance embedded processing. The integration of ARM Cortex-A9 processors within FPGA-based systems is then explored, highlighting the advantages of hardware–software co-design. Finally, the chapter reviews sensor fusion techniques for object tracking, focusing on the fusion of IMU and ultrasonic data through Kalman filtering. Each section draws on relevant literature to contextualise the design choices and implementation strategies adopted in this project.

2.1 The Kalman Filter

The Kalman Filter is a recursive algorithm designed to estimate the state of a system from a series of measurements. It has since become a foundational technique in control systems, signal processing, and real-time embedded applications such as navigation and object tracking. Developed initially for linear systems, it has since been adapted to model non linear systems through variations such as the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF) [1].

The Kalman filter functions in two stages: the prediction stage and the update stage. It forecasts the next state of the system based on the current state and the systems' dynamics, then refines this prediction with incoming measurements and their associated uncertainties.

Romanovas outlines the structure of the kalman filter, which combines the high-level prediction-correction flow of the algorithm with the lower level data flow within the model above, in 2.1 [2].

Its recursive approach is particularly effective for real-time scenarios [1]. The figure below shows the recursive nature of the Kalman Filter, utilising the prediction to update its estimate.

At the heart of the Kalman Filter is its ability to minimise the mean square error of the estimated state by combining prior knowledge (predicted state) with new information (sensor measurements). It's treatment of uncertainty which is modeled through two distinct types of noise: process noise and measurement noise, allows it to maintain a balance between "trusting" the system model and the sensor data [3].

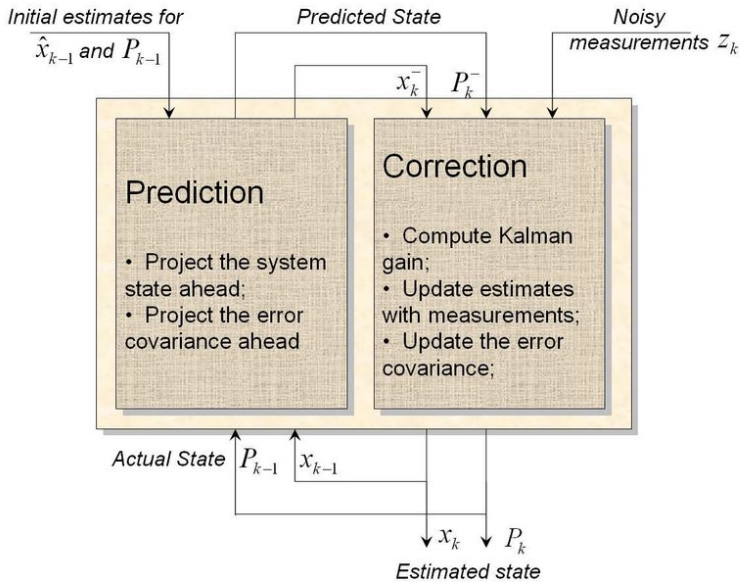


Figure 2.1: Execution cycle of the Kalman Filter with system inputs and outputs

The Kalman filter's recursive nature and ability to continuously update estimates make it well-suited for real-time applications [4]. In scenarios where data is constantly changing, the filter provides timely and accurate state estimates [5] [6].

Kalman filtering for object tracking applications is an extensively researched field and as Tony Lacey from MIT asserts, it "has long been regarded as the optimal solution to many tracking and data prediction tasks" [7].

The Kalman filter is also widely accepted for its versatility to object tracking problems which is highlighted in X. Li's work, where they address the real time challenges of interference and obstacles to propose a Kalman filter algorithm for tracking multiple moving objects in real-time [7]. This highlights the broad range of capabilities Kalman filtering offers in real time object tracking and its ability to be customised to a particular problem [8]. Additionally, Kenshi Saho provides a use of Kalman filters for tracking moving objects, focusing on a steady-state performance. Their study shows that incorporating position-velocity measurements improves tracking accuracy compared to position-only measurements [6].

2.2 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are reconfigurable integrated circuits that allow for the implementation of custom hardware logic after manufacturing. Unlike fixed-function Application-Specific Integrated Circuits (ASICs), FPGAs offer the flexibility to adapt to changing application requirements and are well-suited to real-time, parallel processing tasks [9]. This makes them ideal platforms for implementing computationally intensive algorithms such as the Kalman Filter [10].

Modern FPGAs include not only logic elements but also DSP blocks, Block RAM (BRAM), and high-speed I/O interfaces [9], which collectively enable efficient hardware acceleration of signal processing tasks. For instance, matrix operations—central to the Kalman Filter—can be parallelised

on FPGAs to significantly outperform traditional CPU implementations in terms of latency and throughput [11].

FPGAs are particularly advantageous in embedded systems requiring deterministic behavior and low latency, such as autonomous vehicles, robotics, and aerospace systems [12]. In these applications, the absence of a general-purpose operating system reduces uncertainty in execution timing. Moreover, FPGAs allow hardware-level control over numerical precision and resource allocation, which is crucial for fixed-point implementations of floating-point algorithms in constrained systems [13].

Despite these advantages, developing FPGA applications comes with challenges. Hardware Description Languages (HDLs) such as Verilog or VHDL are low-level and require careful management of timing, resource constraints, and pipelining [14]. High-Level Synthesis (HLS) tools mitigate some of this complexity but can still fall short in optimising deeply pipelined control-dominated logic [15].

2.3 ARM Cortex-A9 Processor Integration

In FPGA-SoC architectures such as the Xilinx Zynq-7000 platform, a hard-core ARM Cortex-A9 processor is tightly integrated with programmable logic [16]. This hybrid architecture enables the co-design of hardware and software systems, where computationally intensive components are offloaded to the FPGA while the processor handles control, communication, and sequential logic.

The ARM Cortex-A9 is a superscalar dual-issue processor that supports advanced features such as out-of-order execution, branch prediction, and SIMD (Single Instruction, Multiple Data) instructions [16], allowing it to efficiently handle high-level control logic, interrupt service routines, and floating-point computations. Its inclusion in the Zynq platform provides an ideal interface between the software and hardware domains [16].

The Zynq-7000 SoC shown below in 2.2, enables high-throughput data exchange between software and custom hardware, with distinct ports for general-purpose, high-performance, and coherent memory access [16].

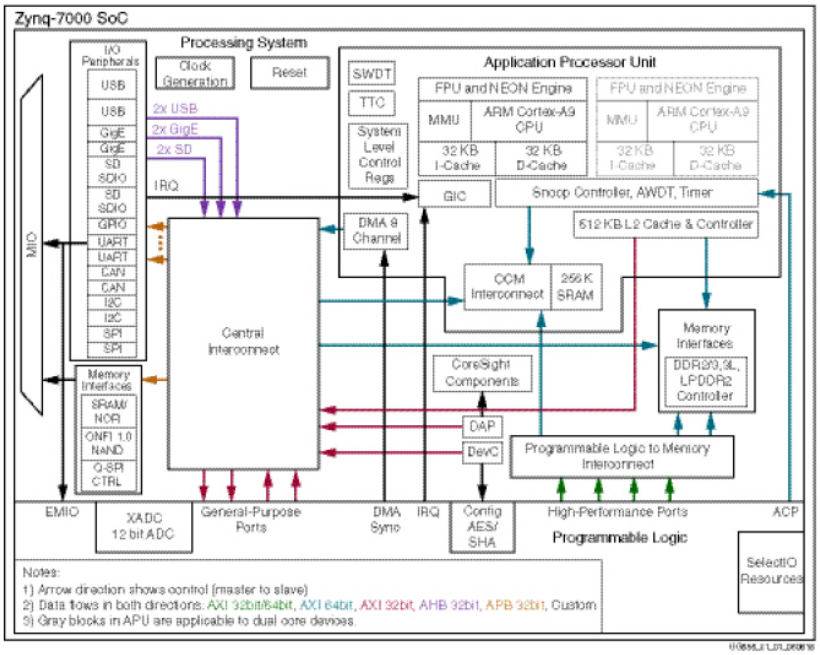


Figure 2.2: ARM Cortex-A9 Architecture

Communication between the ARM processor and the programmable logic is typically managed via AXI (Advanced eXtensible Interface) interconnects [17]. This interface allows memory-mapped I/O, DMA transfers, and hardware acceleration to be orchestrated by the processor without polling overhead. For example, in the implementation of a Kalman Filter, the ARM core prepares sensor data, triggers hardware execution, and post-processes the results for logging or further control decisions [18].

This division of labor reduces the computational burden on the processor while maximising the real-time performance benefits of the FPGA. However, careful coordination is necessary to manage data coherency, DMA synchronisation, and interrupt latency [19].

2.4 Sensor Fusion for Object Tracking

Sensor fusion combines data from multiple sources to produce more accurate and reliable estimates than would be possible using any individual sensor alone [20]. In the context of object tracking, combining data from an Inertial Measurement Unit (IMU) and a radar or ultrasonic rangefinder can improve both temporal resolution and spatial accuracy [21].

IMUs provide high-rate acceleration and angular velocity data, enabling short-term motion prediction, but they are subject to drift over time due to integration errors and sensor noise [22]. Conversely, ultrasonic or radar sensors provide absolute distance measurements but at lower rates and are more susceptible to environmental noise, such as multipath reflections or obstructions [23].

Kalman Filtering is well-established as an effective method for sensor fusion in such contexts [24]. It provides an optimal estimate of the system state by accounting for both process noise (from the model) and measurement noise (from the sensors). The Kalman Filter operates recursively, making it computationally suitable for real-time applications [25].

Its mathematical framework inherently supports the fusion of heterogeneous sensors with differing update rates and noise profiles [26]. In this project, sensor fusion is implemented by integrating fixed-point IMU acceleration data with periodic distance updates from ultrasonic sensors. The Kalman Filter is deployed on hardware for real-time state estimation, while the software running on the ARM processor manages sampling synchronisation and data formatting.

Challenges in sensor fusion include time-alignment of asynchronous data streams, accurate noise modeling, and filter stability under uncertain conditions [27]. Effective tuning of the process and measurement covariance matrices is critical for reliable performance [28].

Chapter 3

Methodology

3.1 High Level Design Overview

The block below 3.1 illustrates the architecture of the object tracking system implemented on a heterogeneous processing platform, which integrates a Processor Subsystem, Programmable Logic, and External Peripherals.

Processor Subsystem

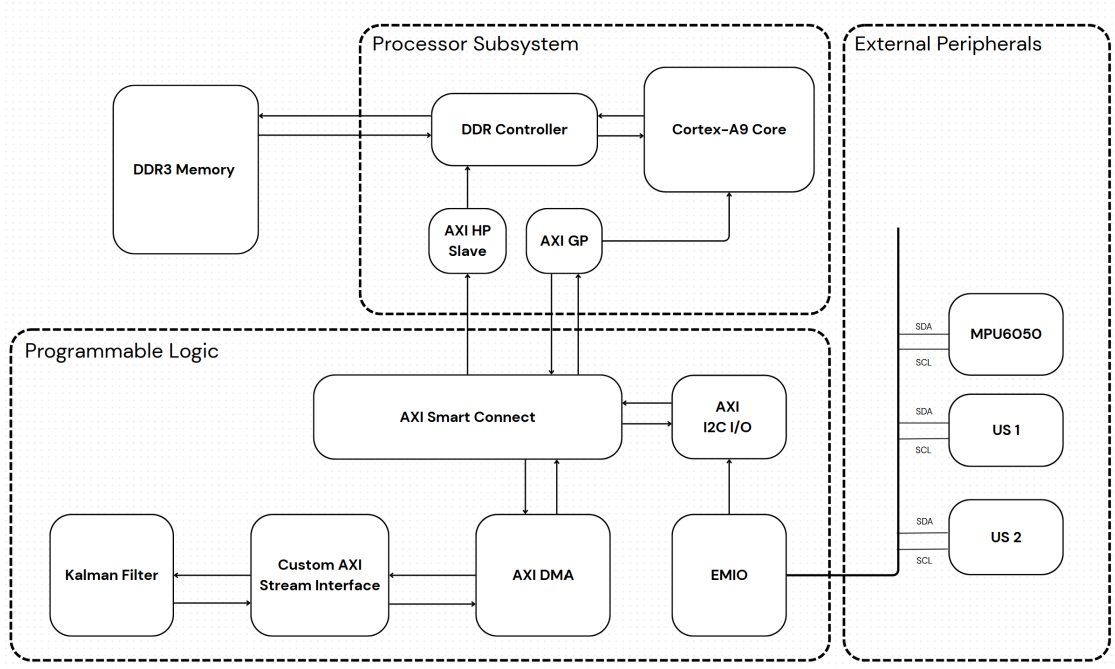


Figure 3.1: Block Diagram of Thesis Project

- Cortex-A9 Core: This is the main processing unit responsible for running the operating system and high-level application software. It interfaces with memory and peripherals via AXI buses.
- DDR Controller: Manages data transfers between the Cortex-A9 core and external DDR3 memory. This controller ensures high-bandwidth access to program instructions and data storage.

- DDR3 Memory: External memory module used to store program code, intermediate data, and large buffers required by the system.
- AXI HP Slave (High-Performance Slave Interface): Provides a high-throughput communication channel from the Programmable Logic (PL) to the Processor Subsystem, enabling fast data transfers.
- AXI GP (General Purpose): Allows the Cortex-A9 core to communicate with the Programmable Logic through a general-purpose interface for control and status registers.

Programmable Logic

- AXI Smart Connect: Acts as a high-performance interconnect within the programmable logic, routing AXI transactions between masters and slaves, including DMA, I2C controllers, and custom IP blocks.
- AXI DMA: Direct Memory Access controller that efficiently transfers large blocks of data between memory and custom logic without CPU intervention, reducing processing overhead and latency.
- Custom AXI Stream Interface: This custom logic interfaces with the Kalman Filter IP, receiving sensor data streams and sending processed data back to memory via the DMA.
- Kalman Filter: The core signal processing IP block implemented in programmable logic that fuses sensor data (from the IMU and ultrasonic sensors) to estimate object position and velocity in real time.
- AXI I2C I/O: An AXI-based I2C controller that handles communication with external I2C peripherals via the EMIO interface.
- EMIO (Extended Multiplexed I/O): Provides programmable input/output pins to connect internal logic to external peripherals like sensors via standard protocols such as I2C.

External Peripherals

- MPU6050: An Inertial Measurement Unit (IMU) sensor providing accelerometer and gyroscope data via the I2C bus.
- US 1 and US 2: Two ultrasonic sensors used to measure distances, also communicating over I2C.
- The SDA (data) and SCL (clock) lines from the AXI I2C I/O controller connect through EMIO pins to the sensors, enabling bidirectional communication and data acquisition.

Data Flow Summary

- Sensor data (IMU and ultrasonic) is read over I2C by the AXI I2C controller and passed through the AXI Smart Connect.

- Data is transferred via the AXI DMA engine to the Custom AXI Stream Interface, which feeds the Kalman Filter hardware IP.
- The Kalman Filter processes the sensor data in real time, estimating the object's position and velocity.
- Processed data can be transferred back to memory accessible by the Cortex-A9 via the AXI HP Slave interface for further processing or logging.
- The Cortex-A9 core controls and configures the programmable logic and peripherals through the AXI GP interface.

3.2 Mathematical Model

3.2.1 System and Measurement Models

The core of the data fusion approach in this project is a linear KF that integrates measurements from multiple sensors to estimate the system state more accurately. The system consists of two ultrasonic sensors providing position measurements in the x and y directions, and one Inertial Measurement Unit (IMU) providing acceleration measurements in the same axes. This results in a total of four measurements per time step:

$$\mathbf{z}_k = \begin{bmatrix} x_k \\ y_k \\ \ddot{x}_k \\ \ddot{y}_k \end{bmatrix}.$$

The KF estimates a six-dimensional state vector:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \\ \ddot{x}_k \\ \ddot{y}_k \end{bmatrix},$$

which includes the position, velocity, and acceleration in both the x and y directions.

The system dynamics are modeled with a discrete-time linear state-space model:

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{w}_{k-1},$$

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k,$$

where $\mathbf{w}_{k-1}$ and $\mathbf{v}_k$ are process and measurement noise vectors, assumed to be zero-mean Gaussian with covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ respectively.

Assuming a constant jerk model with sampling interval Δt, the state transition matrix $\mathbf{A}$ is defined as:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The measurement vector includes direct observations of position and acceleration; velocity is not measured. Thus, the measurement matrix $\mathbf{H}$ maps the state vector to measurements:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

3.2.2 Algorithm

The KF algorithm, as mentioned earlier, consists of two main steps: prediction and update as illustrated below. *Prediction step:*

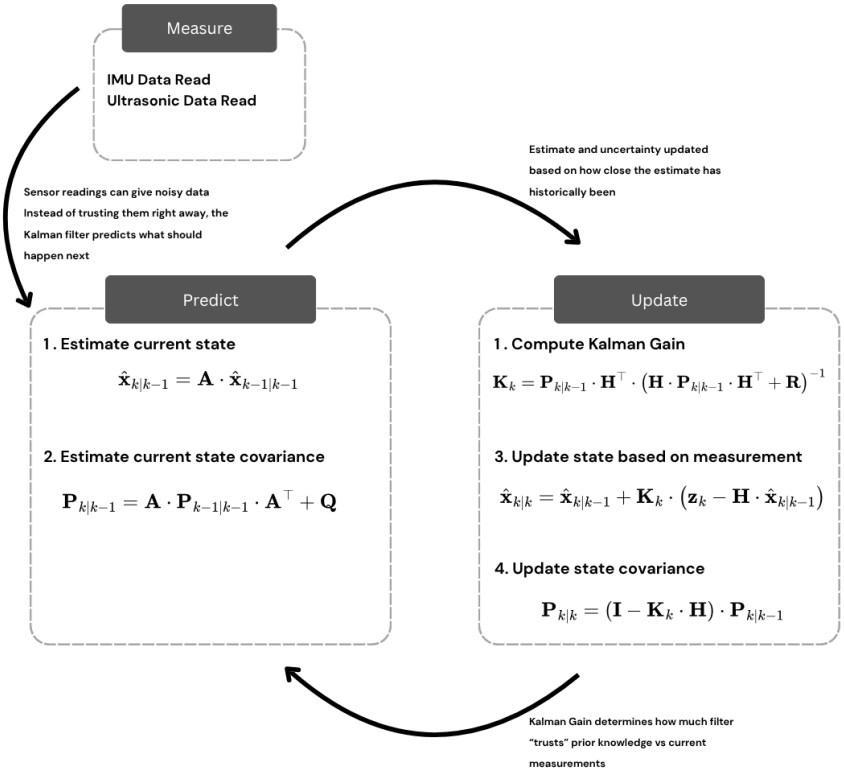


Figure 3.2: KF Predict Update Algorithm incorporating sensor inputs

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}\hat{\mathbf{x}}_{k-1|k-1},$$

$$\mathbf{P}_{k|k-1} = \mathbf{A}\mathbf{P}_{k-1|k-1}\mathbf{A}^T + \mathbf{Q},$$

where $\hat{\mathbf{x}}_{k|k-1}$ is the predicted state estimate and $\mathbf{P}_{k|k-1}$ is the predicted estimate covariance.

Update step:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T \left(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R}\right)^{-1},$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1}\right),$$

$$\mathbf{P}_{k|k} = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}\right)\mathbf{P}_{k|k-1},$$

where $\mathbf{K}_k$ is the Kalman gain, $\hat{\mathbf{x}}_{k|k}$ is the updated state estimate, and $\mathbf{P}_{k|k}$ is the updated estimate covariance matrix.

3.2.3 Sensor Fusion and Noise Modeling

The measurement noise covariance matrix $\mathbf{R} \in \mathbb{R}^{4\times4}$ characterises the sensor uncertainty in the KF. In this system, the ultrasonic sensors measure position, while the IMU provides acceleration data. The matrix was defined as:

$$\mathbf{R} = \begin{bmatrix} \sigma_p^2 & 0 & 0 & 0 \\ 0 & \sigma_p^2 & 0 & 0 \\ 0 & 0 & \sigma_a^2 & 0 \\ 0 & 0 & 0 & \sigma_a^2 \end{bmatrix}$$

where $\sigma_p = 0.2\,\text{m}$ is the estimated uncertainty of the ultrasonic position readings and $\sigma_a = 0.8\,\text{m/s}^2$ is the uncertainty of the acceleration measurements from the IMU. These values were chosen roughly with a license to tune.

The process noise covariance matrix $\mathbf{Q} \in \mathbb{R}^{6\times6}$ accounts for uncertainty in the motion model, including unmodeled dynamics and external disturbances. A constant jerk model was assumed, and the entries of $\mathbf{Q}$ were scaled by the square of the estimated jerk noise standard deviation $\sigma_j = 0.9\,\text{m/s}^3$. The entries were computed based on time-scaling terms involving the sampling interval $dt = 9.6\,\text{ms}$, particularly:

$$\mathbf{Q}_{ij} = \sigma_j^2 \cdot dt^3 \quad \text{for relevant terms}$$

This reflects an assumption of smooth but uncertain acceleration changes over time. While $\mathbf{Q}$ is symmetric and positive semi-definite, its off-diagonal structure was simplified to describe correlated uncertainty between position, velocity, and acceleration states in both dimensions.

The IMU provides higher frequency acceleration measurements along the x and y axes. Although these measurements are responsive and high-rate, they are prone to drift and bias over time. Conversely, the radar sensor supplies lower-frequency but more stable position measurements. By combining both, the KF combines both properties, the short-term accuracy of the IMU and the long-term stability of the radar.

Because all sensor inputs were sampled simultaneously, the filter design is simplified, and all updates occur at a fixed timestep. No interpolation, buffering, or timestamp alignment was used. This deterministic update schedule also simplifies FPGA implementation and DMA interfacing.

3.3 Hardware

3.3.1 Zynq-7000 SoC Architecture

This project uses the XC7Z007S device on the Digilent Cora Z7 board. It combines an ARM Cortex-A9 processor with Artix-7 FPGA fabric, 28,800 FFs logic cells, 80 DSP slices, and 240 KB of block RAM. The Cora Z7 is shown below

Xilinx 7-series FPGA

The FPGA fabric is built from configurable logic blocks, programmable routing, and dedicated DSP and memory blocks. I used DSP slices to accelerate matrix operations like multiply-accumulate, and BRAMs were allocated for buffering sensor data and storing intermediate filter results. The logic blocks were used for control logic and fixed-point arithmetic.

One challenge was fitting the full KF into the limited fabric while keeping latency low. The use of logic elements had to be tightly controlled. Some parts of the design were pipelined to meet timing,

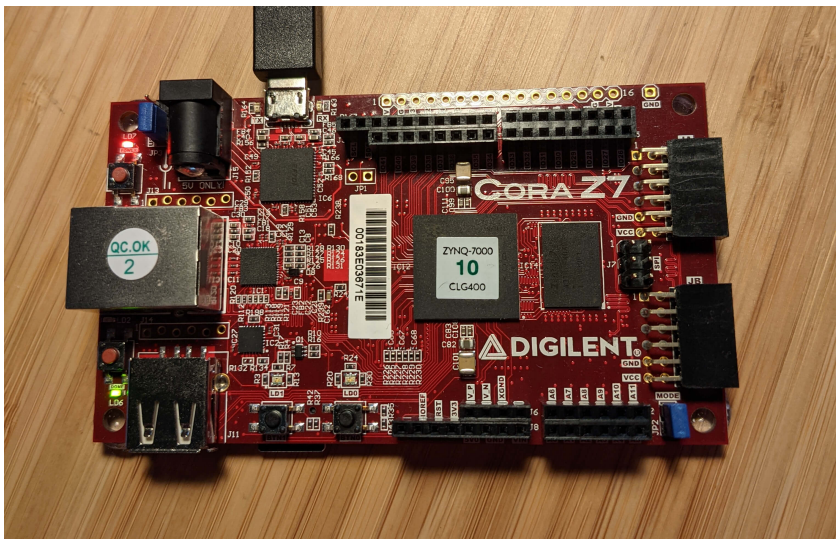


Figure 3.3: Digilent Cora Z7 board used in the project

but increasing pipeline depth also increased utilisation. Routing congestion also became an issue, especially when trying to push the clock frequency above 100 MHz. I settled on a 100 MHz clock as a balance between performance and timing closure.

The *AreaOptimised_High* synthesis directive was used in Vivado to reduce the amount of logic elements used, but ultimately it came down to seperating matrix operations into further states, and not using concurrent carry add operations.

The Zynq-7000 SoC architecture allowed tight integration between the PS and PL using AXI interfaces. The Kalman IP block was wrapped in a simple AXI interface in verilog, letting the processor trigger computations and read back results. Below is an overview of the ports of the AXI wrapper and handshake signals. I used the AXI High Performance PS port as these read/write ports access DDR memory directly, which makes it better suited to DMA than the general purpose ports.

ARM Cortex-A9

The ARM Cortex-A9 processor runs at 667 MHz and handles all high-level control. It polls the sensors, performs basic filtering and calibration, and communicates with the Kalman IP block. Since only one core was needed, I used a bare-metal C environment instead of an operating system to keep things simple and reduce overhead. All development was done using Xilinx Vitis, with some use of the Xilinx-provided IIC and AXI drivers.

Communication between the processor and FPGA was done using shared registers on the AXI bus. A small state machine in the PL handled trigger and done signals to keep handshaking simple and reliable.

3.3.2 FPGA Onboard Memory and External Communication

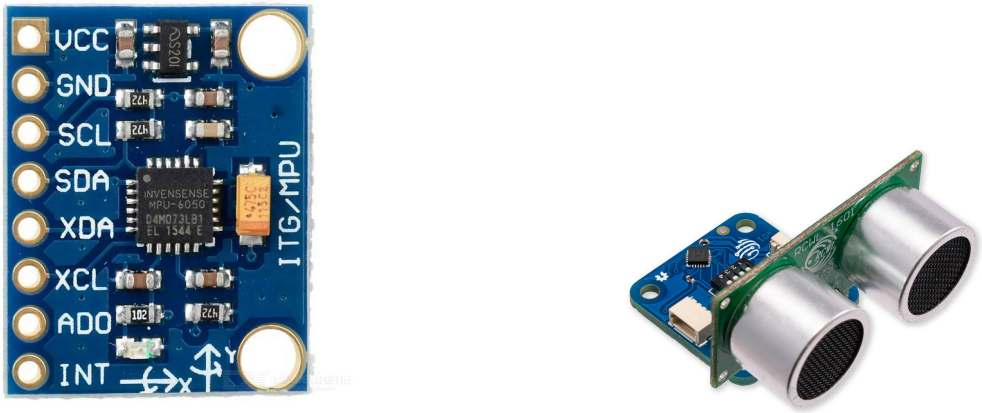
Block RAM in the FPGA was used for buffering sensor readings and temporary matrix storage. There wasn't enough room to hold entire history windows, so the buffers were sized just large enough for one complete update cycle. For logging and debug data, I used the external DDR3 memory connected to the processor. This kept the BRAM free for real-time operations.

All sensors were connected through the I2C bus. The IMU and both ultrasonic sensors shared a single bus connected to the PL. I used the Xilinx IIC IP core to manage bus access. The PL handled low-level I2C communication and buffered the results in BRAM. The processor read from these buffers and passed the data to the Kalman block.

Data flowed from the sensors into PL-side BRAM, then through the KF, and back to the processor. Each cycle was triggered by the processor once new sensor readings were available. This setup kept latency low and avoided blocking the processor while the FPGA computed the result.

3.3.3 IMU and Ultrasonic Sensors

The system used one IMU and two ultrasonic sensors 3.4. The IMU provided 3-axis acceleration and gyroscope data at 100 Hz, while the ultrasonic sensors gave range readings at about 20 Hz, depending on the polling rate. Since all three sensors were on the same I2C bus, careful timing was needed to avoid contention.



(a) MPU 6050 6-DOF IMU (b) Piicodev Ultrasonic Rangefinder

Figure 3.4: I2C Sensors Used

The default I2C clock speed for the IMU is 400kHz, but it can be confugured to 100kHz. The US has a default of 100kHz.

3.3.4 Printed Circuit Board

A custom PCB was designed to integrate all three sensors onto a shared I2C bus. This PCB connects directly to the shield connector of the Coraz7 development board, making the wiring easier to manage.

The PCB consolidates all I2C signals and sensor power rails into a compact, plug-in module. The schematic, shown in figure 3.5, outlines the key features of a shared I2C bus connects the MPU6050 and both ultrasonic sensors via their SDA and SCL lines. It also routed 3.3V and GND power lines from the Coraz7 board to the sensors.

Figure 3.6 below shows the manufactured PCB, designed to slot neatly onto the Coraz7's shield connector: The top layer shown in 3.6a includes headers for the I2C signals, with common power and ground. The bottom layer shown in 3.6b makes signal routing easier. The SDA and SCL lined were length matched to within 5 percent of each other. Although it probably isnt a huge deal on the 400kHz line, this was done to remove a potential source of error.

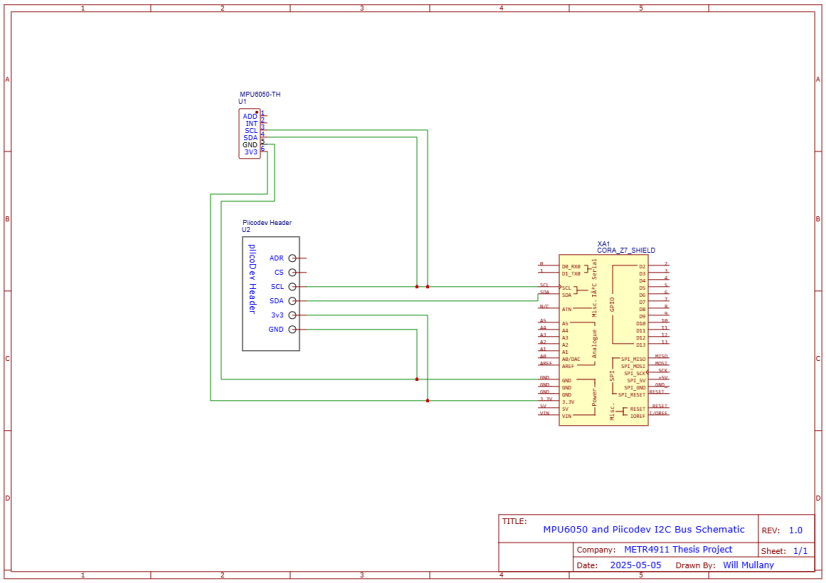
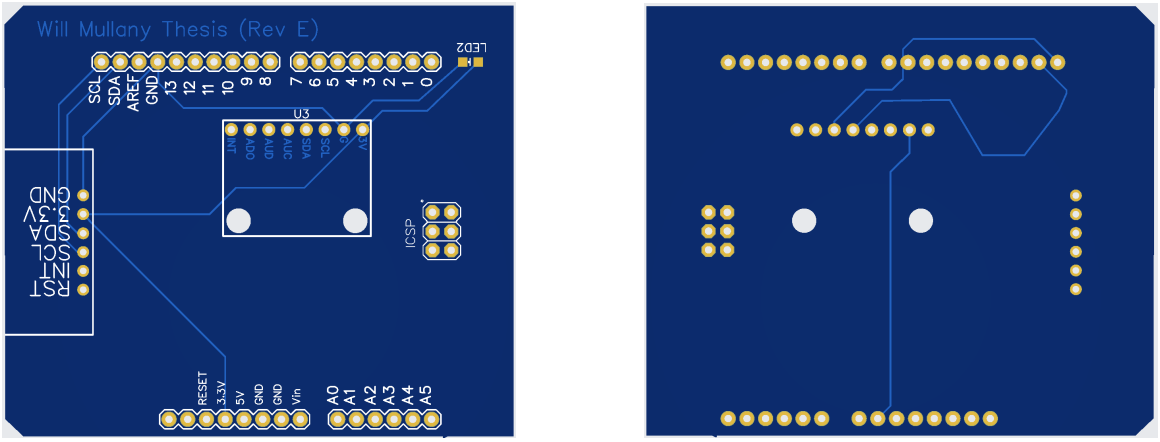


Figure 3.5: Custom PCB schematic implementing a Slot-on connection to the Cora Z7 shield, with I2C and power connections

This design enabled a compact and reliable sensor subsystem with minimal wiring overhead, supporting simultaneous data acquisition from the IMU and both ultrasonic sensors over the common bus.



(a) PCB Top View (b) PCB Bottom View

Figure 3.6: Manufactured PCB for I2C Bus

3.4 Firmware

3.4.1 KF Architecture

The KF was implemented as a custom IP core in the FPGA, designed to perform iterative state estimation by processing input matrices P, z, and measurement matrices in each cycle, and then producing updated state estimates x and covariance P as outputs.

The core computation was divided into three main stages corresponding to the KF algorithm steps: **Predict**, **Gain**, and **Update**. Each stage was implemented as a separate Verilog module, instantiated and coordinated by a top-level KF finite state machine (FSM) module.

The implementation was carried out primarily at the Register-Transfer Level (RTL), where the algorithm's mathematical operations were directly translated into hardware processes. Fundamental matrix operations—including matrix multiplication, inversion, addition, and subtraction—were modeled as finite state machines controlling precise data movement between registers and arithmetic units.

By describing the design behaviorally yet managing state transitions explicitly, the Verilog code captured both the functional algorithm and the hardware sequencing necessary for computation. The use of non-blocking assignments within states were done to make sure the registers were latched at each cycle. This allowed more control over pipeline operations while managing dependencies between matrix calculations.

Matrix operations were scaled up to 40.24 fixed-point format to reduce the risk of silent overflow and div-zero errors. Results were thenscaled back to a 20.12 fixed-point format to reduce resource usage and match the input width of the next module.

The **Predict**, **Gain**, and **Update** modules instantiated the matrix operations in sequence, with each step waiting for the completion (signalled by start/done handshaking) of the previous operation before proceeding. More attention was put towards making sure output registers were latched before start/done.

3.4.2 Matrix Inversion

The implemented module performs 4×4 fixed-point matrix inversion using Gauss-Jordan elimination, with a state machine controlling the algorithm's flow. The matrix is represented in Q20.12 fixed-point format, with inputs and outputs packed into 512-bit buses. The implementation is tailored for FPGA deployment on constrained platforms (e.g., Xilinx Zynq-7000 on the Cora Z7), with a focus on sequential logic control, reduced area, and manageable timing closure.

Temporal parallelism (pipelining across clock cycles) is used. The control FSM ensures that matrix operations are broken down into manageable sub-steps, executed over several cycles.

While full unrolling is avoided, the design allows each element within a row to be processed iteratively in a fast loop. This is a compromise between full parallelism and minimal hardware. The design processes one row at a time, meaning only 4 multiplications per cycle at most. Full matrix-wide unrolling would consume over 100 DSP slices and thousands of LUTs, which would overutilise the available resources.

By sequencing operations and reusing arithmetic units, the design reduces LUT/DSP pressure, at the cost of higher latency per inversion.

3.4.3 Communication Interface

A custom AXI Stream wrapper was developed around the KF to enable direct memory access (DMA) data transfers between the Processing System (PS) and PL (PL). This wrapper handled streaming inputs and outputs over the AXI Stream interface. Both master and slave ports of the AXI wrapper were configured as 32-bit streams in 16-bit pulses. These were then buffered by the slave until the full 46 byte transfer was complete. The master port then read the KF's 42 byte output to write in 32-bit streams for PS access through the DMA engine.

3.4.4 Hardware Acceleration Strategy

Initial profiling of the synthesised algorithm showed that the 6×6 matrix multiplication and 4×4 matrix inversion dominated the computational load. Both operations were implemented as separate Verilog modules within the hardware filter.

Matrix multiplication took flattened input matrices A and B, unpacked them into local arrays, and computed the result matrix C = A × B using a nested loop structure controlled by a simple three-state FSM (IDLE, COMPUTE, DONE). Multiply-accumulate operations were performed serially with 64-bit accumulation to prevent overflow, and packed back into the flattened output bus.

Matrix inversion was initially implemented using the adjoint method, which involves calculating cofactors and the determinant. The method was partially unrolled with loops but presented strong data dependencies that prevented effective pipelining.

The inversion module was redesigned to use a serial Gauss-Jordan elimination algorithm. This approach processes one pivot row and eliminates elements column by column over multiple clock cycles. By reusing arithmetic units across cycles instead of duplicating hardware, resource utilisation dropped significantly. This serial method increased latency but reduced logic and register use enough to fit the entire design on the FPGA.

The KF was partitioned into stages: prediction, gain and update. Each stage was implemented as a finite state machine controlling data flow and arithmetic operations.

The fixed-point arithmetic format chosen was Q20.12, which has 20 integer bits and 12 fractional bits. While there is still more resourced required for fixed decimal point arithmetic, its much less than floating point. The trade off here is the range of numbers and the precision decreasing in comparison to 32-bit floating point. Initially, a 32-bit IEEE 754 floating-point implementation was done, but it resulted in excessive usage of LUT resources—exceeding the available 14,400 LUTs on the target board.

To address this, the design was fully converted to fixed-point arithmetic. Although some matrix operations could be parallelised to improve throughput, resource constraints limited the degree of parallelism. Most optimisations focused on parallelising matrix multiplications where possible, rather than attempting to pipeline higher-level filter steps concurrently. Each module and submodule employed start/done handshaking to latch outputs through the computation stages.

Table 3.1: Comparison of Q20.12 Fixed-Point and 32-bit Floating-Point Formats

Property	**Q20.12 Fixed-Point**	**32-bit Floating-Point (IEEE 754)**
Bit Width	32 bits	32 bits
Numeric Range	$[-524{,}288,\ 524{,}287.9998]$	$\sim [-3.4 \times 10^{38},\ 3.4 \times 10^{38}]$
Precision (step size)	$2^{-12} \approx 0.00024$	$\sim 10^{-7}$ near 1.0
Dynamic Range	Fixed (linear)	Wide (exponential)
Arithmetic Simplicity	Simple (integer-like ops)	Complex (normalisation, rounding)
FPGA Resource Usage	Low to moderate	High (Too high)
Best Use Case	Bounded-range, fast applications	General-purpose, wide-range math

3.5 Software

The ARM Cortex-A9 processor handles tasks that are not well-suited for FPGA acceleration. These include sensor data preparation, managing memory and peripheral communications, and applying higher-level logic for sensor fusion and control sequencing. The software was developed in bare-metal C to reduce overhead and improve real-time responsiveness.

3.5.1 Communication with FPGA

Communication between the processor and the FPGA logic was done using AXI DMA transfers. After preprocessing, input data was written to a designated memory region aligned with the DMA controller's requirements. The processor triggered a DMA transfer to send the data to the KF IP core. Once computation was complete, the *M_AXIS_TREADY* signals the AXI master, which is then written to the PS DDR. This mechanism was used to retrieve the filtered output from the hardware.

3.5.2 High-Level Control and Fusion Logic

The processor ran additional logic that the FPGA did not handle. This included converting raw sensor data into a fixed-point representation, managing the control flow of sensor fusion operations, and preparing formatted output for serial transmission. The fusion logic involved triggering the Kalman update cycle and coordinating between IMU and ultrasonic data streams. The software also maintained timestamps and synchronisation between sensor readings.

3.5.3 Sensor Data Preprocessing

Raw data from the IMU and ultrasonic sensors was collected over the I2C bus. For each sensor, a specific register address was written to initiate the read process, followed by a multi-byte read transaction. A table of the specific I2C addresses and buffer lengths are shown below.

IMU data included accelerometer and gyroscope readings along three axes, while ultrasonic sensors returned time-of-flight measurements. The ultrasonic times were converted to distance using a calibrated scaling factor. All sensor readings were converted into a fixed-point format compatible with the KF hardware.

Table 3.2: I2C Sensor Data Configuration

Sensor	**I2C Address**	**Register Address**	**Buffer (bytes)**	**Raw Data Format**	**Conversion Formula**
MPU-6050 (Accel)	0x68	0x3B	6	3 × 16-bit Two's Comp.	accel $(m/s^2) = \frac{\text{raw}}{16384}$
PiicoDev US 1	0x35	0x04	2	16-bit Unsigned	dist $(m) = \frac{\text{raw} \times 343}{2 \times 10^6}$
PiicoDev US 2	0x09	0x04	2	16-bit Unsigned	dist $(m) = \frac{\text{raw} \times 343}{2 \times 10^6}$

3.5.4 High-Level Control and Fusion Logic

The processor handled sensor polling, fixed-point conversion, and the orchestration of control flow. After preparing the input data, it initiated the DMA transfer to send it to the KF hardware accelerator. Once the hardware completed the filtering, the results were received via DMA and forwarded over UART for logging or visualisation. To reduce latency and improve determinism, the software was implemented without an operating system, using a tight control loop.

Chapter 4

Results

4.1 Resource Utilisation

The utilisation of FPGA resources is a fundamental part of evaluating a given design on an FPGA. How an FPGA design uses available resources effects its power consumption and its ability to meet the specified timing requirements. Increasing the pipeline depth will increase utilisation and reduce the critical path delay, but will also increase power consumtpion. In contrast, optimising for area by reusing existing circuits will reduce utilisation and power consumtpion, but will make it harder for the design to meet the FPGA's timing requirements.

4.1.1 Kalman Filter IP

This design attempted to find a balance between timing and latency, with the constraint of resource utlilisation. During implementation, the design exceeded available LUTs and FFs, preventing a successful bitstream generation. The Cora Z7, equipped with 14,400 LUTs and 28,800 FFs was a significant constraint, especially when implementing the 6 dimensional KF as it involves multiple expensive matrix operations. Below is the utilisation summary for the KF, including the KF IP block, custom axi interface, DMA and AXI routing blocks.

Attempts to reduce utilisation through loop unrolling control, shared resource scheduling, and reduced pipelining yielded only marginal improvements, and further improvements would hinder the hardware's ability to meet timing requirements. Ultimately, this demonstrated the need for either a simplified model, further optimisation of the HDL, or migration to a higher-capacity FPGA platform to enable full functionality.

Figure 4.1 shows that the Kalman Filter design significantly exceeds the Cora Z7's resource limits, with over 20,000 LUTs and nearly 50,000 flip-flops used—far beyond the available 14,400 LUTs and 28,800 FFs. The most resource-intensive components are the kalman_gain, kalman_predict, and kalman_update blocks, particularly due to their heavy DSP and logic usage

Name	Slice LUTs (14400)	Slice Registers (28800)	F7 Muxes (8800)	F8 Muxes (4400)	Block RAM Tile (50)	DSPs (66)	Bonded IOB (100)	Bonded IOPADs (130)	BUFGCTRL (32)
N kf_bd_wrapper	20325	49767	2966	873	2	52	2	130	1
kf_bd_i (kf_bd)	20325	49767	2966	873	2	52	0	0	1
axi_dma_0 (kf_bd_axi_dma_0_1)	1346	2060	4	0	2	0	0	0	0
axi_iic_0 (kf_bd_axi_iic_0_0)	408	356	7	0	0	0	0	0	0
KF_0 (kf_bd_KF_0_0)	15833	43760	2955	873	0	52	0	0	0
inst (kf_bd_KF_0_0_KF)	15833	43760	2955	873	0	52	0	0	0
kalman_inst (kf_bd_KF_0_0_kalman)	15285	39409	2795	873	0	52	0	0	0
u_gain (kf_bd_KF_0_0_kalman_gain)	6978	13456	837	258	0	28	0	0	0
u_predict (kf_bd_KF_0_0_kalman_predict)	3952	12591	1158	425	0	8	0	0	0
u_update (kf_bd_KF_0_0_kalman_update)	4355	13359	800	190	0	16	0	0	0
KF_master_stream_v1_0_M00_AXIS_inst (kf_bd_KF_0_(	426	1398	160	0	0	0	0	0	0
KF_slave_stream_v1_0_S01_AXIS_inst (kf_bd_KF_0_0_K	122	2953	0	0	0	0	0	0	0
processing_system7_0 (kf_bd_processing_system7_0_0)	26	0	0	0	0	0	0	0	1
rst_ps7_0_50M (kf_bd_rst_ps7_0_50M_3)	21	40	0	0	0	0	0	0	0
smartconnect_0 (kf_bd_smartconnect_0_0)	571	757	0	0	0	0	0	0	0
smartconnect_1 (kf_bd_smartconnect_1_0)	2120	2794	0	0	0	0	0	0	0

Figure 4.1: FPGA Post-Synthesis Utilisation showing Kalman Filter Exceeding Available Resources

4.1.2 Sensor Data Acquisition and Processing

The system sampled the IMU and two ultrasonic sensors every 0.93 ms, which is roughly 1075 Hz. However, the IMU has an output data rate of 100 Hz (one new reading every 10ms), and the ultrasonic sensors had an output data rate of 50 Hz (one new reading every 20ms). Since the processor polled the sensors at a much higher rate than they internally update, most reads provide identical, stale data. No check was implemented to determine whether the data were fresh before using it.
This frequent polling placed an unnecessary load on the I2C bus and the DMA controller. Each read triggered I2C transactions that consumed bus bandwidth and increased power consumption, especially since the ultrasonic sensors required one read each. This also caused more memory traffic for the DMA interface without adding new information, occupying tight buffer space and increasing latency for other operations.

Table 4.1: Sensor Sampling and Redundancy Analysis

Sensor	**Interface**	**Output Rate (Hz)**	**Polling Interval (ms)**	**Redundant Reads/s**	**Samples Per Second**	**Useful Samples/s**	**New Data Check**
IMU	I^2C	104	0.93	≈9	1075	100	No
US X	I^2C	50	0.93	≈21	1075	50	No
Us Y	I^2C	50	0.93	≈21	1075	50	No

From a timing perspective, this approach did ensure deterministic behavior and simplified scheduling by keeping the loop uniform. Furthermore, this was implemented in a nonblocking manner using a polling loop and the main processor was not stalled during data acquisition. But from an efficiency point of view, it wastes bus cycles and overall I2C utilization.
From the KF's perspective, the main impact was that repeated identical measurements were processed as if they were new, which had limited impact on the state estimate but still consumed unnecessary computation.

Power analysis revealed a total on-chip power consumption of 1.542 W, with the PS contributing 1.255W, indicating that the majority of the power draw was from the ARM cores and related PS infrastructure rather than the PL shown in 4.2. This aligns with expectations for a design with a heavy DMA and software interface workload. The total power remained well below the Zynq-7000 SoC's typical upper limit of 4.5 W. The junction temperature reached 42.8C, with a thermal margin of 42.2 C, showing that the device operated with significant thermal headroom, well below the 100 degree limit.

This confirms that, although logic resources were overutilised, the power and thermal performance remained well within safe and reliable limits.

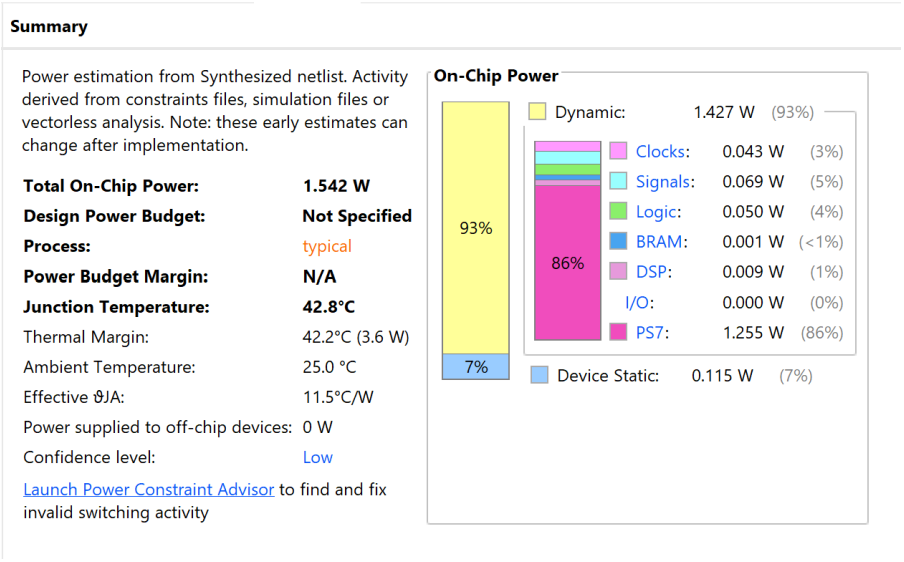


Figure 4.2: Post-Synthesis Power Estimate

4.1.3 Improvements

Pipelining depth could be increased selectively in critical paths such as the matrix inversion and Kalman gain blocks to reduce timing violations. However, this must be done carefully to avoid exceeding power or resource budgets.

Alternatively, restructuring the Kalman filter to operate more sequentially could drastically reduce LUT and FF usage. Lowering the dimensionality of the state vector or simplifying the process and measurement models may also reduce overall arithmetic complexity, easing implementation pressure.

Another viable solution is to offload some operations—such as matrix inversion or gain calculation—to the ARM processor, thereby sharing the computation load between the PL and PS. Finally, migrating the design to a Zynq-7020 or Artix-7 would provide significantly more logic resources while maintaining compatibility with the existing architecture.

4.2 Timing Summary

The timing summary revealed that all hold time constraints were met, with a positive slack of +0.04ns on all reported paths. These hold paths were minimal, involving only a single routing stage with net delays of 0.14ns each. This indicates local routing with no risk of data arriving too early, and confirms that no additional hold buffers or timing adjustments were necessary.

In contrast, the setup timing was severely violated, with the worst slack reported as -53.48ns. These violations were concentrated in the matrix inversion logic within the Kalman gain computation block, particularly across paths from the `A_reg` registers to the output ports of the inversion pipeline. The total path delay on these routes exceeded 72ns, with over 50ns attributed to logic delay, indicating too much pipeline depth, even after a lot of effort to reduce it. This exceeds the required clock period of 20ns, making the timing paths over 2.5 times too long. The high number of logic levels (244–245), routing

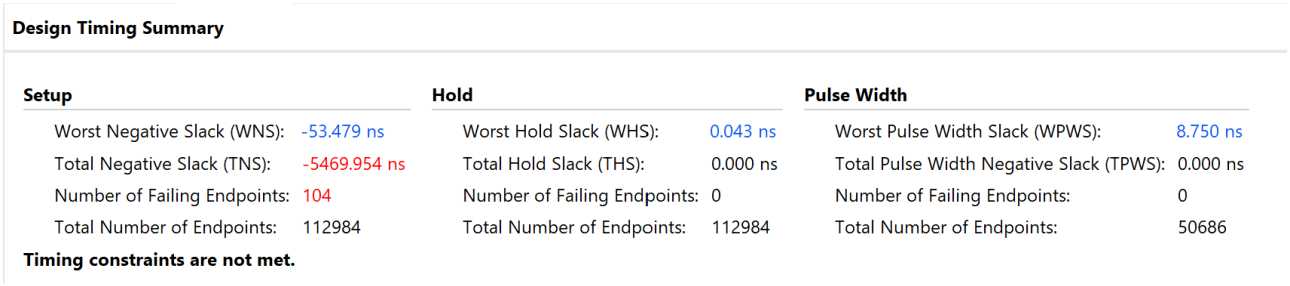


Figure 4.3: Post-Synthesis Timing Summary

hops (35), and high-fanout nets (up to 99) further suggest long critical paths and insufficient fanout optimisation. These setup violations prevented successful timing closure and bitstream generation. Addressing this would require:

- Reducing arithmetic complexity by reusing hardware or simplifying operations,
- Increasing pipeline depth to reduce logic per stage,
- Migrating to a faster clock domain, or
- Using a larger FPGA with more logic and routing resources.

These results highlight the fundamental trade-off between algorithmic complexity and achievable hardware performance on constrained platforms.

4.3 Kalman Filter Performance

4.3.1 Model Validation

To validate the KF model implementation on the Cora Z7, a custom test rig was setup along a sliding channel, keeping one axis stationary to ensure the model adequately removes noise, while the other axis was fitted to a sliding gantry on a 20-Series V-slot channel. The setup is pictured below in with a number of 3D printed fittings and housings for the sensors, Cora Z7 and the V-slot channel.

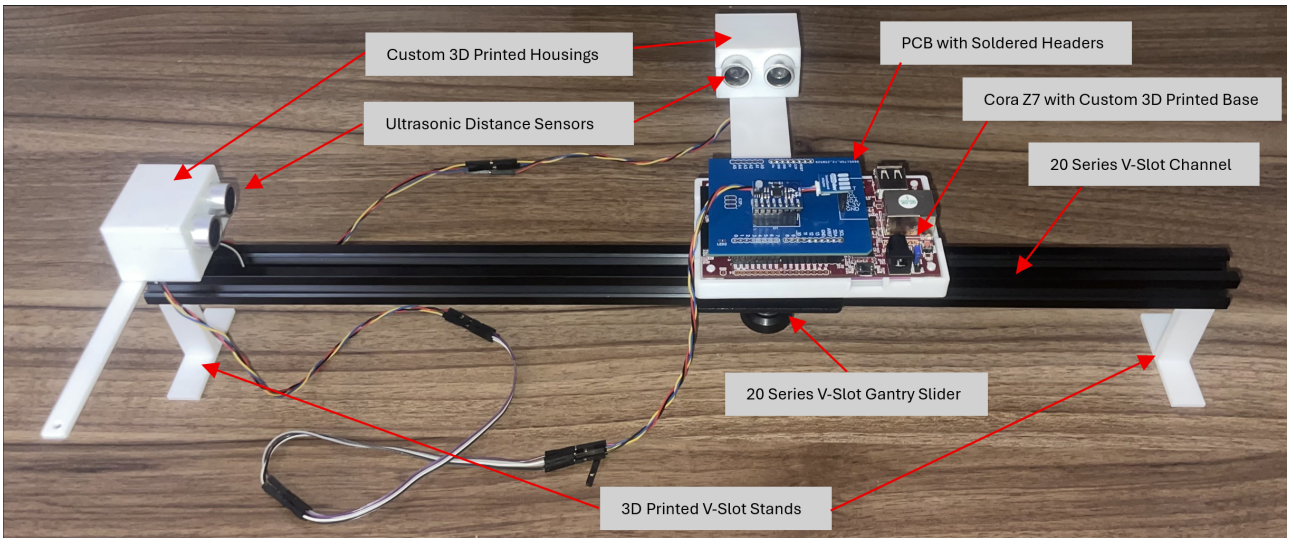
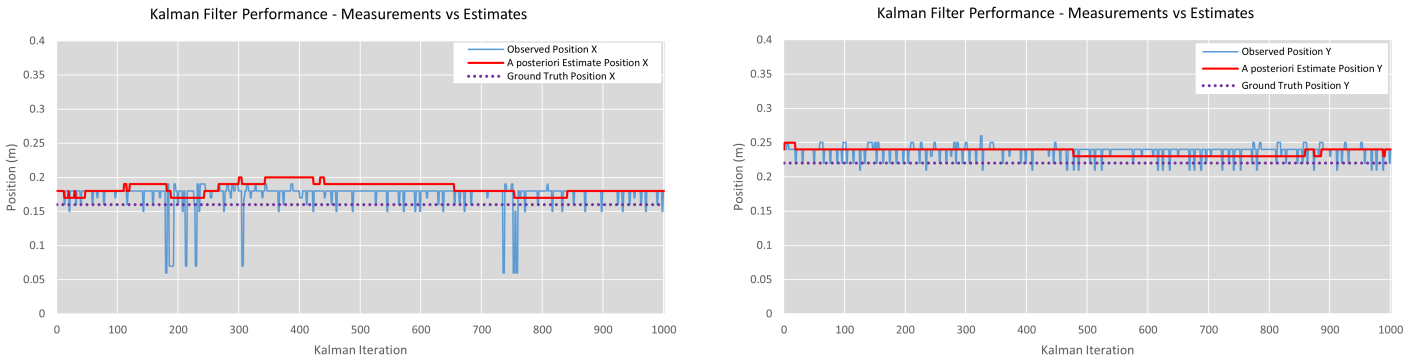


Figure 4.4: Custom Test Rig Setup

To initially validate the model, the board was kept stationary along the channel. This allowed an initial stage of tuning sensor and process noise parameters before introducing more rigourous testing.

Figures 4.6a and A.2 show the Kalman filter's estimated acceleration on the X and Y axes. The raw X-axis accelerometer data in 4.6a exhibits a clear bias around $+0.07$ m/s^2 and is affected by significant high-frequency noise. Despite this, the filter's estimate gradually decreases and levels off near zero, which is the true value. This demonstrates that the Kalman filter is not simply following the biased sensor data. Instead, it uses position to infer that the constant positive acceleration reading does not match the actual system motion.

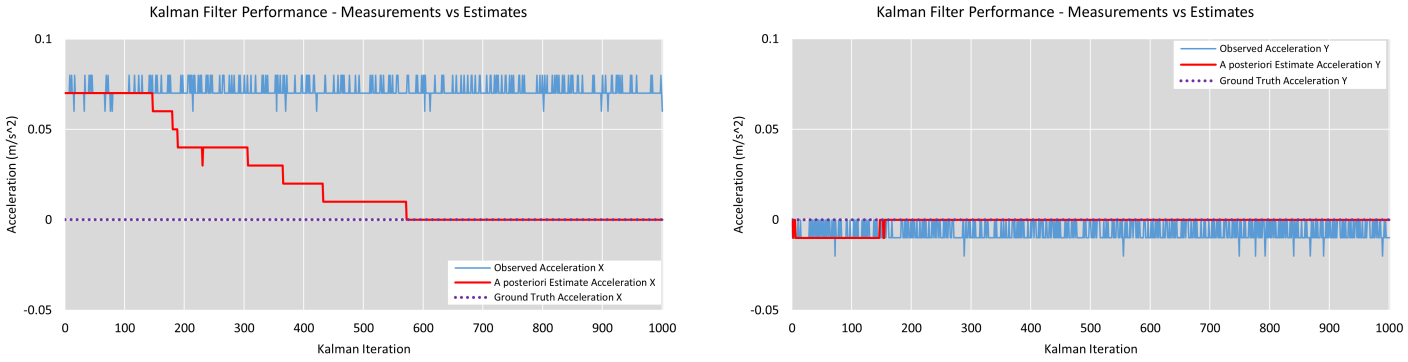


(a) X-axis

(b) Y-axis

Figure 4.5: Kalman filter estimated vs observed position in X and Y axes for Static Test.

Figures 4.6a and A.2 show the Kalman filter's estimated acceleration on the X and Y axes. The raw X-axis accelerometer data in 4.6a exhibits a clear bias around $+0.07$ m/s^2 and is affected by significant high-frequency noise. Despite this, the filter's estimate gradually decreases and levels off near zero, which is the true value. This demonstrates that the Kalman filter is not simply following the biased sensor data. Instead, it uses position to infer that the constant positive acceleration reading does not match the actual system motion.



(a) X-axis

(b) Y-axis

Figure 4.6: Kalman filter estimated vs observed acceleration in X and Y axes for Static Test.

The accumulated error between the predicted and measured positions drives a correction in the estimated acceleration, slowly compensating for the bias. The slow, stepped correction seen in the X-axis estimate is due to the filter's low process noise for acceleration, which discourages rapid changes in the estimate unless strongly supported by other measurements.

On the Y-axis, the system exhibits very little motion, and the observed acceleration fluctuates around zero with minimal bias. The filter correctly holds its estimate near zero throughout the interval. This indicates that the model's process noise, measurement noise, and dynamic assumptions are good enough for this scenario.

Although this test involves only a static scenario, it confirms that the model in hardware is working as intended. The system reads real sensor data, processes it through the KF, and outputs stable, consistent state estimates. The filter correctly handles noise and bias, showing that the state prediction and correction steps are functioning. This provides a solid baseline for more dynamic testing.

4.3.2 Comparison with Software-only Version

The project was intended to compare a hardware-accelerated Kalman filter to a software-only baseline. Due to logic limitations on the Cora Z7, only the software version was implemented.

Still, execution profiling indicated that a full PL implementation could reduce processing time by 70–80%, particularly in the matrix inversion and multiplication stages. Given a larger FPGA (e.g., Artix-100T or Kintex), offloading these operations would be feasible.

Figure 4.7: Estimated velocities in X and Y axes for Static Test.

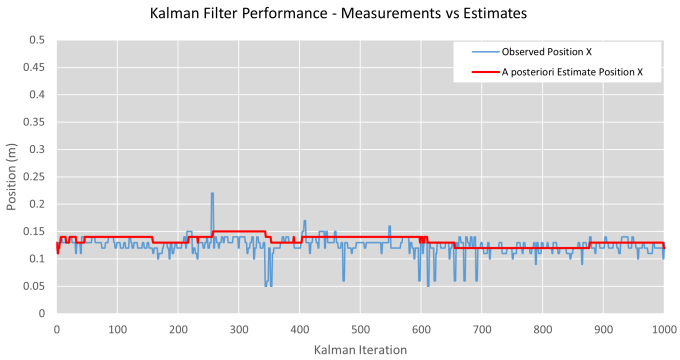
4.4 Experimental Validation

4.4.1 Simulated Scenarios

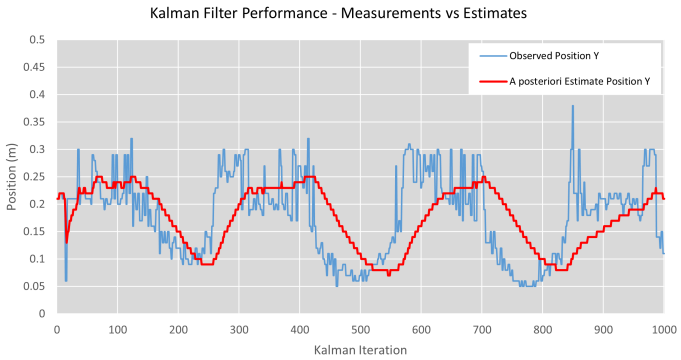
To validate the system under controlled conditions, a custom test setup was developed. A linear rail with a V-slot gantry was constructed, allowing for repeatable motion along a fixed axis. The processing board and sensors were mounted to a 3D-printed carrier, which was securely attached to the moving gantry. An image of this test setup is provided in Figure **??**.

The IMU was soldered to the custom PCB and fixed to the board in its intended orientation. The system was tested under both stationary and dynamic conditions. These scenarios included small displacements, rapid direction changes, smooth translations, and random vibrations. Throughout testing, sensor tuning was performed iteratively; observing the Kalman filter output and adjusting noise parameters to improve stability and response.

The results of this process are presented below, which compares raw sensor measurement to its estimate across position, acceleration and velocity. The board was constrainted to move along the Y axis only, with the X axis held stationary. This allowed evalulation of the KF's ability to track motion while also supressing noise. Figure 4.8a shows the X position (held constant). The raw sensor measurements show minor jitter, but the KF estimates remain stable and close to the true position. Its important that the KF rejects this sensor noise and does not introduce these dynamics into the system. Figure 4.8b showing the Y coordinate tracking under motion. The raw Y measurement shows significant noise and sharp local deviations but the KF otputs a smooth trajectoty that follows the expected motion along the rail. Here the trade off between responsiveness and noise rejection can be seen, with the KF lagging slightly, but rejecting the high frequency measurement noise. Making the filter more responsive to sensor data increases tracking speed, but can make it more sensitive to noise.

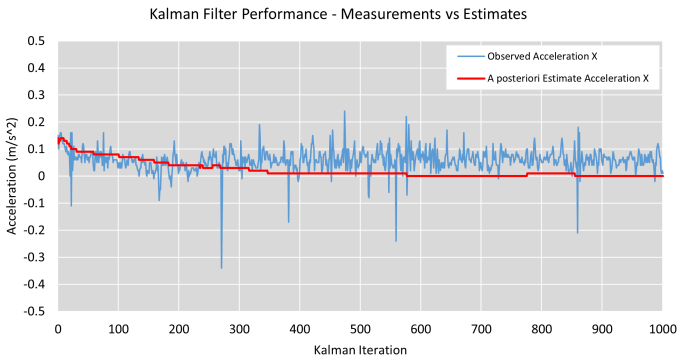


(a) X-axis

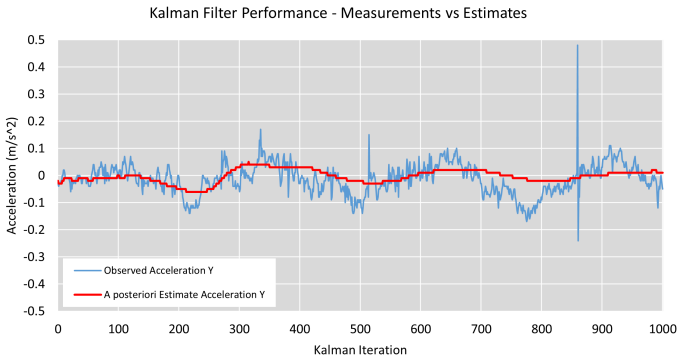


(b) Y-axis

Figure 4.8: Kalman Filter Estimated vs Observed Position after Introducing Oscillatory Movement



(a) X-axis

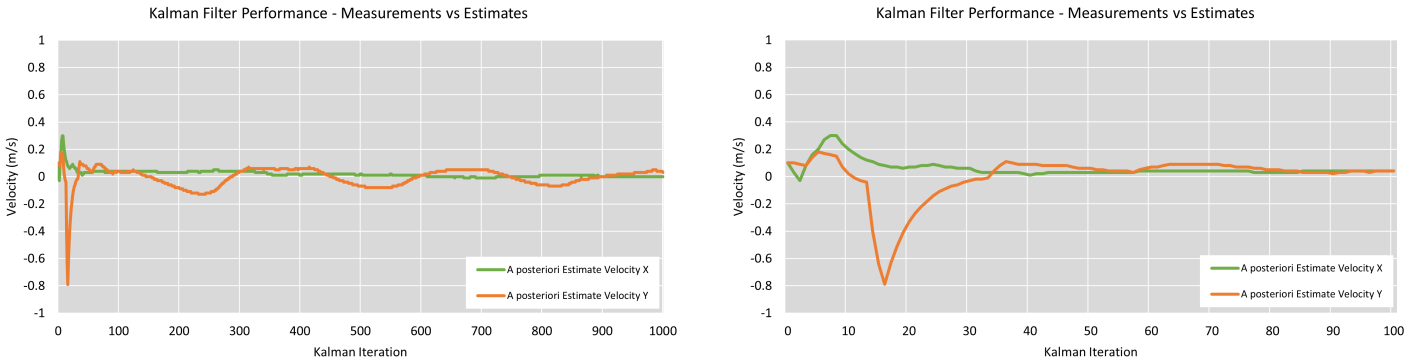


(b) Y-axis

Figure 4.9: Kalman Filter Estimated vs Observed Acceleration after Introducing Oscillatory Movement

Figure 4.12a reveals frequent jitter in the measured data but the estimated signal remains robust to this noise. Additonally the estimation slowly converges to the ground truth acceleration of 0. This demonstrates the KF down-weighting the IMU measurements due to a small offset; roughly 0.8 m/s^2, and leaning more heavily on the X US data. This is an important characteristic of the KF, and is why it's so commonly used in object tracking, which is its ability to remove noise from a signal while also using the system dynamics to actively infer the most likely true state.

The Y acceleration plot 4.12b shows noisy IMU measurements with the occasional spike, a common feature of the MPU6050. The KF attenuates noise pretty well, and somewhat tracks the small oscilations in acceleration. While these long, shallow oscillations are the true system dynamics, they are small and are interpreted by the KF as noise. More precuation needs to be taken when tuning the IMU characteristics as they are succeptable to large spikes and significant noise. The IMU's uncertainly could be tuned more to make it more responsive but we still want to keep it pretty resilient to noise. Since velocities are not directly observed, these are inferred by the KF from position and acceleration inputs. The X-velocity 4.10a converges pretty close to zero after the initial transients. This accurately describes the object's motion, showing that the filter will not invent velocity where there is no motion data to support it. Furthermore, the Y-velocity's profile aligns closely with the physical movement along the rail, showing that for this controlled scenario, the LKF is probably good enough to describe the system. The transient effects in 4.10b are a common side effect of making sure



(a) Steady State Characteristics

(b) Transient Characteristics

Figure 4.10: Kalman Filter Estimated Velocity after Introducing Oscillatory Movement

the KF stays responsive and another trade-off that needs to be managed when tuning the KF.

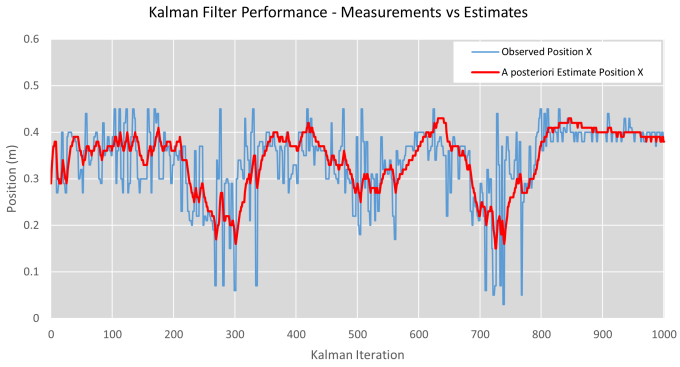
Overall, the results demonstrate that the Kalman filter maintained stable state estimates under varying motion conditions, while rejecting measurement noise and bias. Remaining discrepancies can be attributed to limitations in sensor noise modelling, particularly in the IMU. Future improvements would focus on further tuning, refining the process noise model by using a more formal approach from testing, or using sensor fusion strategies that incorporate sensor confidence levels dynamically, by changing **Q** and **R** on the go.

4.4.2 Model Assumptions and Incorrect System Example

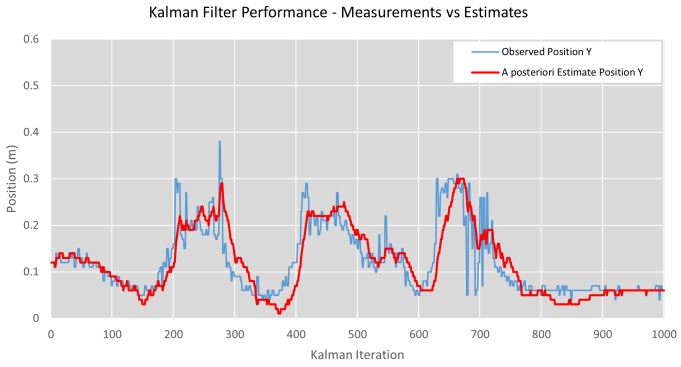
Sensor noise can, and often is filtered out using a simple Low Pass Filter. An LPF was considered for this project due to its simplicity, low resource utilisation, and effectiveness at removing noise. However a low-pass filter assumes a fixed cutoff frequency, which fails under dynamic noise and system changes. The Kalman filter, in contrast, models uncertainty in both process and measurement, adapting dynamically through its gain matrix. This made it far more robust for sensor fusion and object tracking in this project.

To demonstrate the robustness of the Kalman filter compared to a simple low-pass filter, an experiment was conducted where an accelerometer, physically aligned with the X-axis, was intentionally modeled as measuring acceleration in the Y-axis. Despite this incorrect assumption, the Kalman filter maintained internally consistent estimates of position, velocity, and acceleration by using its predictive model, effectively ”learning” a plausible trajectory consistent with the incorrect input. Although this was not an aim of the project, nor is giving a model incorrect data a strategy to be employed at any time, it’s a small example of how the KF adapts to how good its estimates are.

In contrast, a low-pass filter applied to the same data blindly smoothed the signal without accounting for directionality or system dynamics, would result in physically meaningless estimates. This highlights the Kalman filter’s key advantage: its ability to reconcile noisy or inconsistent data through model-based inference, that simple filtering methods cannot achieve.

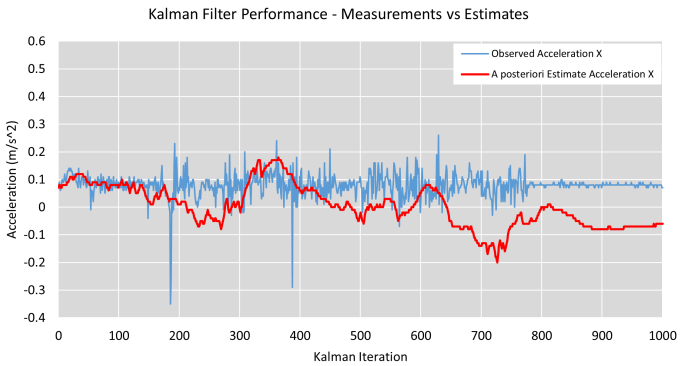


(a) X-axis

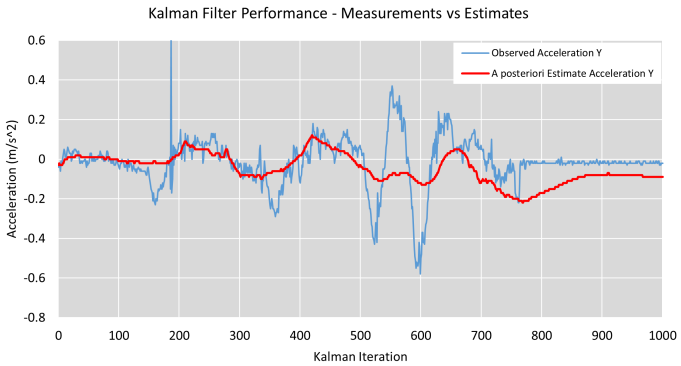


(b) Y-axis

Figure 4.11: Kalman Filter Estimated vs Observed Acceleration after Positioning Ultrasonic Sensor Incorrectly



(a) X-axis



(b) Y-axis

Figure 4.12: Kalman Filter Estimated vs Observed Acceleration Positioning Ultrasonic Sensor Incorrectly

Chapter 5

Conclusion

5.1 Summary

This thesis set out to design, implement, and evaluate a real-time object tracking system based on a Kalman filter, fusing radar and IMU data on an FPGA. The project successfully met several key objectives, including the formulation of a sensor fusion model, development of a Kalman filter architecture, and integration of IMU and ultrasonic (used in place of radar) sensors. Although a complete hardware implementation on the Zynq-7000's programmable logic was not feasible due to resource constraints, the ARM Cortex-A9 processor was used to run a software-based version for validation.

The Kalman filter was shown to deliver stable and accurate tracking results, effectively rejecting sensor noise while adapting to dynamics in the system. Real-time throughput and soft real-time timing requirements were met, and the filter demonstrated robustness even when exposed to incorrect or noisy inputs. Experimental validation confirmed the system's ability to infer plausible motion and suppress sensor jitter, meeting the project's original aim of demonstrating Kalman-based fusion on embedded hardware.

5.2 Limitations

Despite the progress made, several limitations constrained the project:

- **FPGA Resource Constraints:** The Digilent Cora Z7 lacked sufficient LUTs and flip-flops to support the full 6D Kalman filter design in hardware.
- **Sensor Simplification:** Ultrasonic sensors were used in place of radar, resulting in lower sampling frequency and spatial resolution than originally planned.
- **Polling-Based Sensor Reads:** Continuous polling of sensors led to frequent retrieval of stale data and inefficient I^2C bus utilization.
- **Static Filter Parameters:** The Kalman filter used fixed Q and R matrices, limiting its adaptability to dynamic or uncertain environments.

- **Manual Tuning and Validation:** Noise parameters and filter gains were tuned manually, without formal system identification or statistical modeling.
- **Software Emphasis:** Hardware acceleration was not realized on the Zynq PL due to logic constraints; performance gains were inferred through profiling rather than measured.

5.3 Recommendations for Future Work

Building on the foundational work and acknowledging the current scope limitations, the following recommendations are proposed:

- **Use a Higher-Capacity FPGA:** Port the Kalman filter design to a larger FPGA (e.g., Artix-7 100T or Kintex-7) to enable full hardware acceleration and direct performance comparisons.
- **Integrate True Radar Sensors:** Replace the ultrasonic modules with actual radar sensors to enable more accurate and higher-frequency position estimation.
- **Interrupt-Driven Acquisition:** Implement event-driven sensor acquisition using data-ready interrupts to reduce bus usage and avoid redundant reads.
- **Adaptive Filtering:** Dynamically adjust Q and R matrices in response to varying sensor noise or operational conditions to improve robustness.
- **Formal System Identification:** Characterize sensor and process noise using empirical data to better inform Kalman filter tuning.
- **Introduce a Real-Time OS:** Incorporate an RTOS such as FreeRTOS to improve task scheduling and enable modular scaling of the system.
- **Explore Nonlinear Filters:** Extend the framework to include Extended Kalman Filters (EKF) or Unscented Kalman Filters (UKF) for systems with nonlinear dynamics.

Bibliography

[1] C. Urrea, Kalman filter: Historical overview and review of its use in robotics 60 years after its creation, Journal of Sensors 2021 (1) (2021) 21.
URL https://doi.org/10.1155/2021/9674015

[2] M. Romanovas, Application of fractional sensor fusion algorithms for inertial mems sensing, Mathematical Modelling and Analysis 14 (2008) 199–209. doi:10.3846/1392-6292.2009.14.199-209.

[3] M. S. Grewal, Kalman filtering, in: International encyclopedia of statistical science, Springer, 2011, pp. 705–708.

[4] P. Gunjal, Moving object tracking using kalman filter (2018).

[5] L. Kleeman, Understanding and applying kalman filtering (2007 2007).
URL https://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/kalman/kleeman_understanding_kalman.pdf

[6] S. Khobahi, Object tracking using kalman filter (2019).
URL https://skhoba2.people.uic.edu/preprints/kalman.pdf

[7] T. Lacey, Tutorial: The kalman filter (1998).

[8] W. W. X. Li, K. Wang, Y. Li, A multiple object tracking method using kalman filter (2010). doi:10.1109/ICINFA.2010.5512258.
URL https://ieeexplore.ieee.org/document/5512258

[9] M. a. FalihS.M.Alkhafaji, WanZ.W.Hasan, Robotic controller :asic versus fpga — areview, Journal of Computational and Theoretical Nanoscience Vol. 15 (25) (2018) 1–25. doi:10.1166.

[10] I. J. L. M. Taborda, Fpga kalman filter (2024).
URL https://github.com/jlmayorgaco/fpga-kalman-filter

[11] Z. J. Milutinovic, V., Fpga accelerator for floating-point matrix multiplication, IET Computers Digital Techniques 6 (4) (2012). doi:10.1049.

[12] J. G. U. I. Abdelfatah, Walid Farid, A. Noureldin, Fpga-based real-time embedded system for riss/gps integrated navigation, Sensors 12 (1) (2012) 32. doi:10.3390.

[13] A. Przybył, Fixed-point arithmetic unit with a scaling mechanism for fpga-based embedded systems, Electronics 10 (10) (2021) 1.

[14] A. Dastidar, Verilog hdl for digital and analog design, in: Advanced VLSI Design and Testability Issues, CRC Press, 2020, pp. 39–66.

[15] A. G. G. Wang, H. Lam, G. Edwards, Performance and productivity evaluation of hybrid-threading hls versus hdls (2015). doi:10.1109.

[16] AMD, Zynq 7000 soc technical reference manual (2023).
URL https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/Cortex-A9-Processors

[17] F. Benevenuti, F. L. Kastensmidt, Reliability evaluation on interfacing with axi and axi-s on xilinx zynq-7000 ap-soc, in: 2018 IEEE 19th Latin-American Test Symposium (LATS), IEEE, 2018, pp. 1–6.

[18] A. Mera, Y. H. Chen, R. Sun, E. Kirda, L. Lu, D-box: Dma-enabled compartmentalization for embedded applications, arXiv preprint arXiv:2201.05199 (2022).

[19] A. Banerjee, A. Mitra, W. A. Najjar, D. Zeinalipour-Yazti, V. Kalogeraki, D. Gunopulos, Rise-co-s: high performance sensor storage and co-processing architecture., in: SECON, 2005, pp. 1–12.

[20] R. C. Harney, Sensor fusion for target recognition: a review of fundamentals and a potential approach to multisensor requirements allocation, in: B. F. Andresen (Ed.), Infrared Technology XX, Vol. 2269, International Society for Optics and Photonics, SPIE, 1994, pp. 316 – 335. doi:10.1117/12.188648.
URL https://doi.org/10.1117/12.188648

[21] N. Senel, K. Kefferpütz, K. Doycheva, G. Elger, Multi-sensor data fusion for real-time multi-object tracking, Processes 11 (2) (2023). doi:10.3390/pr11020501.
URL https://www.mdpi.com/2227-9717/11/2/501

[22] D. Unsal, K. Demirbas, Estimation of deterministic and stochastic imu error parameters, in: Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium, IEEE, 2012, pp. 862–868.

[23] M. Toa, A. Whitehead, Ultrasonic sensing basics, Dallas: Texas Instruments (2020) 53–75.

[24] J. Z. Sasiadek, Sensor fusion, Annual Reviews in Control 26 (2) (2002) 203–228.

[25] S. N. Salinger, J. J. Brandstatter, Application of recursive estimation and kalman filtering to doppler tracking, IEEE transactions on aerospace and electronic systems (4) (2007) 585–592.

[26] V. Fathabadi, M. Shahbazian, K. Salahshour, L. Jargani, Comparison of adaptive kalman filter methods in state estimation of a nonlinear system using asynchronous measurements, in: Proceedings of the World Congress on Engineering and Computer Science, Vol. 2, 2009, pp. 20–22.

[27] P. Matisko, V. Havlena, Noise covariances estimation for kalman filter tuning, IFAC Proceedings Volumes 43 (10) (2010) 31–36.

[28] A. G. Gelen, A. Atasoy, A new method for kalman filter tuning, in: 2018 International Conference on Artificial Intelligence and Data Processing (IDAP), IEEE, 2018, pp. 1–6.

Appendix A

Appendix

Code Repository

`https://github.com/wmullany/thesis`

Hardware

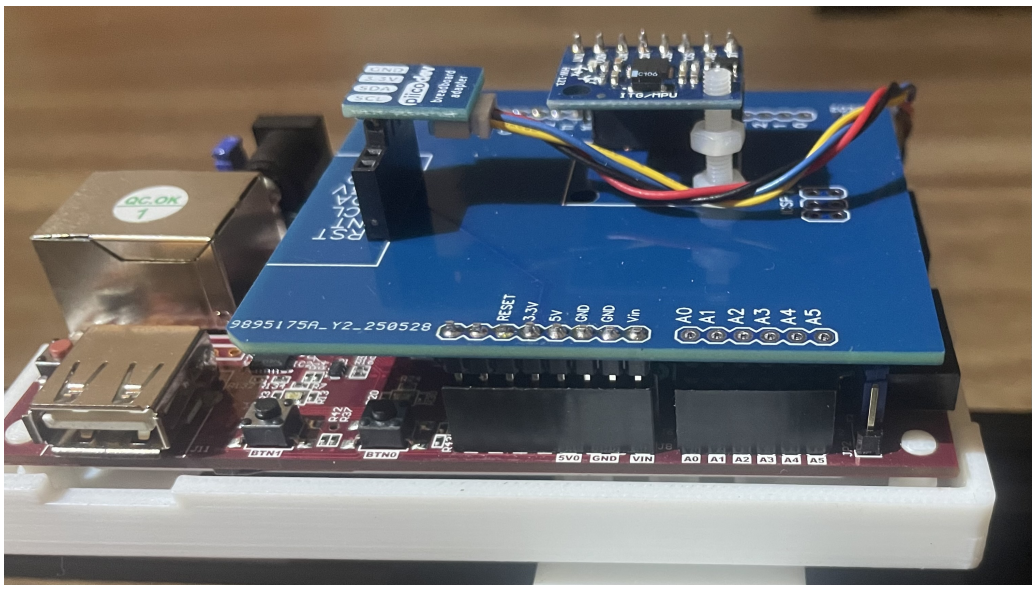


Figure A.1: Final Product

Utilisation Design Information

Table A.1: Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	20325	0	0	14400	141.15
LUT as Logic	19755	0	0	14400	137.19
LUT as Memory	570	0	0	6000	9.50
Distributed RAM	356	-	-	-	-
Shift Register	214	-	-	-	-
Slice Registers	49767	0	0	28800	172.80
Flip Flop	49767	0	0	28800	172.80
Latch	0	0	0	28800	0.00
F7 Muxes	2966	0	0	8800	33.70
F8 Muxes	873	0	0	4400	19.84

Table A.2: Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	-	-	Set
0	-	-	Reset
0	-	Set	-
0	-	Reset	-
0	Yes	-	-
16	Yes	-	Set
1579	Yes	-	Reset
573	Yes	Set	-
47599	Yes	Reset	-

Table A.3: Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	2	0	0	50	4.00
RAMB36/FIFO*	2	0	0	50	4.00
RAMB36E1	2	-	-	-	-
RAMB18	0	0	0	100	0.00

Table A.4: DSP Usage

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	52	0	0	66	78.79
DSP48E1	52	-	-	-	-

Table A.5: IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	2	2	0	100	2.00
Master Pads	1	-	-	-	-
Slave Pads	1	-	-	-	-
Bonded IPADs	0	0	0	2	0.00
Bonded IOPADs	130	130	0	130	100.00
PHY_CONTROL	0	0	0	2	0.00
PHASER_REF	0	0	0	2	0.00
OUT_FIFO	0	0	0	8	0.00
IN_FIFO	0	0	0	8	0.00
IDELAYCTRL	0	0	0	2	0.00
IBUFDS	0	0	0	96	0.00
PHASER_OUT/PHY	0	0	0	8	0.00
PHASER_IN/PHY	0	0	0	8	0.00
IDELAYE2/FINEDELAY	0	0	0	100	0.00
ILOGIC	0	0	0	100	0.00
OLOGIC	0	0	0	100	0.00

Table A.6: Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	1	0	0	32	3.13
BUFIO	0	0	0	8	0.00
MMCME2_ADV	0	0	0	2	0.00
PLLE2_ADV	0	0	0	2	0.00
BUFMRCE	0	0	0	4	0.00
BUFHCE	0	0	0	48	0.00
BUFR	0	0	0	8	0.00

Table A.7: Primates

Ref Name	Used	Category
FDRE	47599	Flop & Latch
LUT6	8493	LUT
LUT3	5452	LUT
LUT5	3566	LUT
MUXF7	2966	MuxFx
LUT2	2531	LUT
FDCE	1579	Flop & Latch
LUT4	1282	LUT
CARRY4	1178	CarryLogic
MUXF8	873	MuxFx
FDSE	573	Flop & Latch
LUT1	551	LUT
RAMD32	528	Distributed Memory
SRL16E	214	Distributed Memory
RAMS32	172	Distributed Memory
BIBUF	130	IO
DSP48E1	52	Block Arithmetic
FDPE	16	Flop & Latch
RAMB36E1	2	Block Memory
OBUFT	2	IO
IBUF	2	IO
PS7	1	Specialized Resource
BUFG	1	Clock

Table A.8: Instantiated Netlists

Ref Name	Used
kf_bd_smartconnect_1_0	1
kf_bd_smartconnect_0_0	1
kf_bd_rst_ps7_0_50M_3	1
kf_bd_processing_system7_0_0	1
kf_bd_axi_iic_0_0	1
kf_bd_axi_dma_0_1	1
kf_bd_KF_0_0	1

Timing Information

Table A.9: Timing Summary for clk_fpga_0

Metric	Value
Setup: Failing Endpoints	104
Setup: Worst Slack (ns)	-53.479
Setup: Total Violation (ns)	-5469.954
Hold: Failing Endpoints	0
Hold: Worst Slack (ns)	0.043
Hold: Total Violation (ns)	0.000
PW: Failing Endpoints	0
PW: Worst Slack (ns)	8.750
PW: Total Violation (ns)	0.000

Table A.10: Design Timing Summary

Metric	Value
WNS (ns)	-53.479
TNS (ns)	-5469.955
TNS Failing Endpoints	104
TNS Total Endpoints	112984
WHS (ns)	0.043
THS (ns)	0.000
THS Failing Endpoints	0
THS Total Endpoints	112984
WPWS (ns)	8.750
TPWS (ns)	0.000
TPWS Failing Endpoints	0
TPWS Total Endpoints	50686

Table A.11: Clock Summary

Attribute	Value
Clock	clk_fpga_0
Waveform (ns)	{0.000 10.000}
Period (ns)	20.000
Frequency (MHz)	50.000

Table A.12: Intra Clock Table

Metric	clk_fpga_0
WNS	-53.479
TNS	-5469.955
TNS Failing Endpoints	104
TNS Total Endpoints	111605
WHS	0.043
THS	0.000
THS Failing Endpoints	0
THS Total Endpoints	111605
WPWS	8.750
TPWS	0.000
TPWS Failing Endpoints	0
TPWS Total Endpoints	50686

Table A.13: Other Path Groups

Metric	async_default
From Clock	clk_fpga_0
To Clock	clk_fpga_0
WNS	17.054
TNS	0.000
TNS Failing Endpoints	0
TNS Total Endpoints	1379
WHS	0.752
THS	0.000
THS Failing Endpoints	0
THS Total Endpoints	1379

Testing

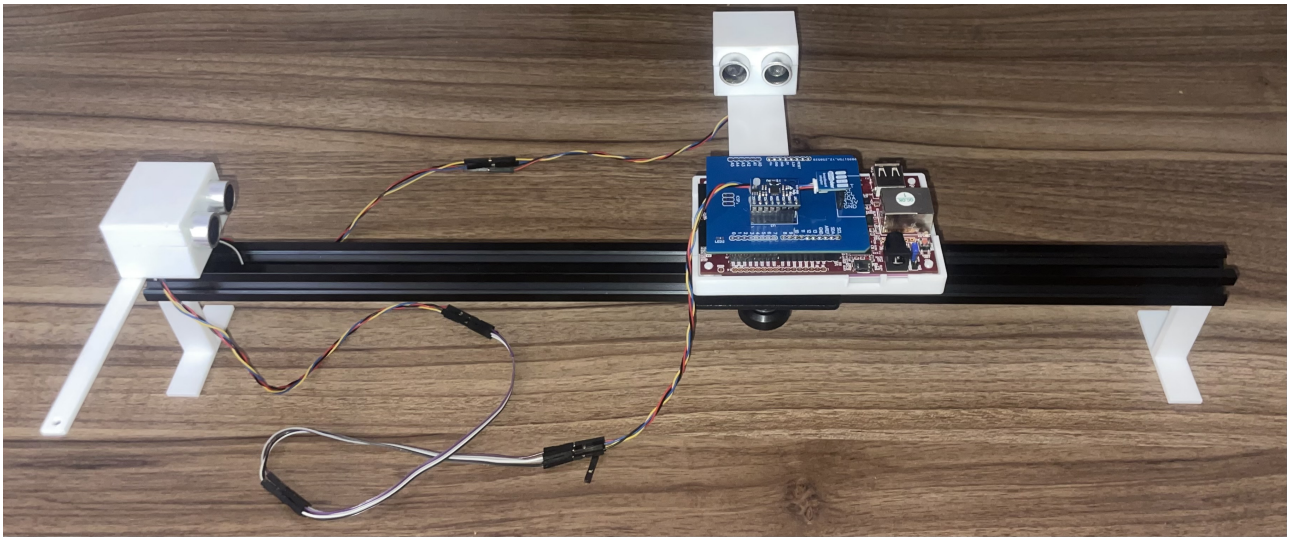


Figure A.2: Custom Test Rig