UNIVERSITY OF SÃO PAULO

POLYTECHNIC SCHOOL OF THE UNIVERSITY OF SÃO PAULO - POLI USP

TECHNICAL REPORT
ADAPTIVE FILTERING SYSTEM

Gabriel Kenji Godoy Shimanuki

Rafael Yuji Yokowo

Dr. Alexandre Moreira Nascimento

Dr. Lúcio Flávio Vismari

Prof. Paulo Sérgio Cugnasca

SÃO PAULO

2021

TECHNICAL REPORT
ADAPTIVE FILTERING SYSTEM

This documentation aims to present general and technical information about the final project of a digital circuit responsible for an Intelligent Adaptive Filtering system, developed as part of the Digital Laboratory II course at the Polytechnic School of USP.

SÃO PAULO

2021

SUMMARY

Overview

This document aims to provide a comprehensive description of the Adaptive Filtering System with a Bayesian Filter, detailing the stages of its development and its key features.

The project, conceived and developed as part of the Digital Laboratory II course at the Polytechnic School of the University of São Paulo, consists of a digital circuit implemented in VHDL using Intel Quartus Prime software. The hardware utilized in the project includes an FPGA board, model DE0-CV, a WeMos D1 board, and an HC-SR04 ultrasonic sensor, along with other additional components that facilitate the control of input signals to the FPGA.

Due to the pandemic, the activities were carried out remotely, with the support of the LabHome infrastructure.

1 Project Specification

The project involves the implementation, with the necessary modifications, of an adaptive filtering system utilizing a recursive estimation technique—specifically, a Bayesian filter—to enhance the reliability of readings from low-cost ultrasonic sensors in measuring distances to solid objects. This work is an extension of a study conducted by the Security Analysis Group [1], which includes the simulation of filter tests. The implementation will use the sonar system developed in the first part of the course as the basis for the ultrasonic sensor. Modifications to the base designs will be made to ensure that the consistency verification tests and the implementation itself align with physical constraints.

Ultrasonic distance sensors for detecting solid objects are used in various everyday applications, such as detecting people, measuring object dimensions, and positioning objects, among others. In addition to these, another potential application of these sensors is in vehicle parking systems. However, sensors used in cars generally have a high cost due to the accuracy and efficiency required. Therefore, low-cost sensors, which may exhibit lower reliability, are typically avoided in applications like the ones mentioned above.

The purpose of this work is to present a *proof of concept* for the use of low-cost sensors in higher-sensitivity applications by employing intelligence techniques, such as the Bayesian filter—Hidden Markov Model. Therefore, the goal of the proposed project prototype is to validate the enhancement of the reliability of low-cost sensors, improving distance measurements through the application of an intelligence-based approach developed in hardware.

1.1 Project Scope

1.1.1 Week 1 – Ideation and Validation Tests

In Week 1, the project proposal was validated through a presentation and discussion with the course professors. The first activity involved verifying the error associated with the ultrasonic sensor (HC-SR04) used in the experiments from Part 1 of the course. The purpose of the validation was to assess the noise signal, which is essential for verifying

the proper functioning of the filtering that will be implemented throughout the project development. The initial tests were static, meaning the sensor was fixed in one position and remained stationary relative to the solid obstacle used as a target. A series of data acquisitions were conducted to observe the sensor's noise. Figure 1 illustrates one of the tests conducted – with the sensor positioned 25 cm from the fixed obstacle. It is evident that even in a static position, the sensor's data acquisition shows deviations.

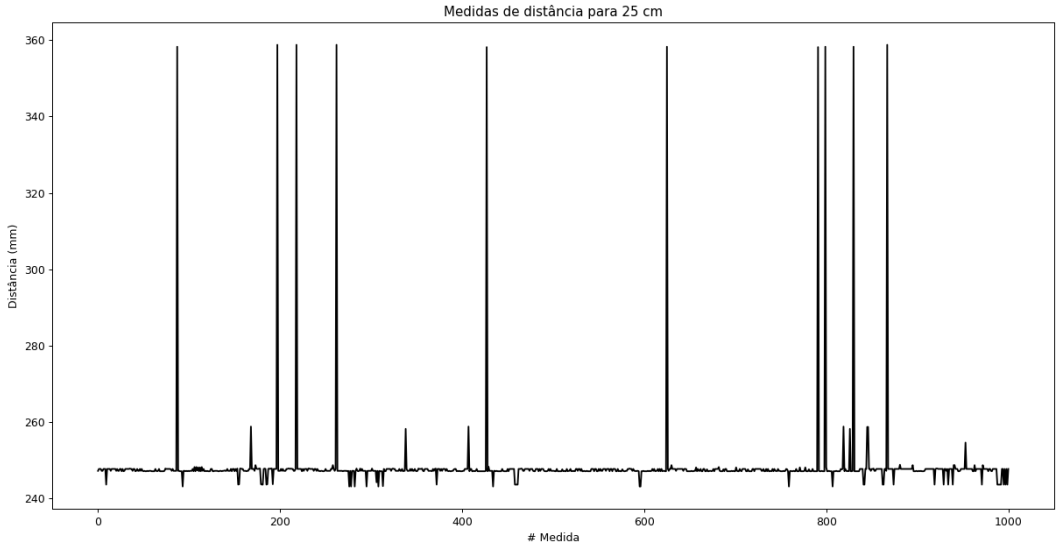

Figure 1: Distance measurement acquisition from the HC-SR04 sensor, positioned 25 cm from the fixed target, with a measurement precision of one-tenth of a millimeter.

In addition, the project requirements for the implementation were outlined as follows:

- Develop the logic for the Bayesian filter in hardware. The filter logic will be implemented in two phases:
  - Implementation of the adaptive filter in software. The programming language used will be Python.
  - After validating the feedback logic, implement the filter in hardware - FPGA.
- Develop an effective data presentation method to help verify the correct operation of the circuit and the filter. This stage involves the following activities:
  - Develop serial communication between the MQTT broker (used in the lab) and a Python notebook.

  - Develop the logic for storing all test logs.
  - Develop a real-time application to visualize sensor measurement curves – observed vs. estimated.

- Test the HC-SR04 sensor under noise injection. This test involves assessing the impact of physical disturbances on the sensor. The activity will be divided into the following steps:
  - Develop a small angle oscillator using a servomotor with an Arduino UNO.

- Sonar system response module. To attenuate noisy signal acquisition, the sonar should be able to identify points where objects are approaching and reduce its rotation speed.

- Development of the physical test scenario. The physical test scenario will be carried out in person, using the setup (FPGA + ESP8266 module + sensors) of one team member. The physical test scenario has the following specifications:
  - The HC-SR04 sensor has the following features:
    * Measurement range: 2 to 400 cm
    * Measurement angle: 15^{o}
  - Physical constraint on the movement of the test equipment.
  - The need for a reliable system for approach and retreat. The initial idea is to use a second servomotor equipped with a fixed target.

Objectives

Thus, the objectives to be achieved are:

- Perform validation tests on the hypothesis regarding distance acquisition errors.
- Implement serial communication between the FPGA and a Python notebook via the MQTT protocol.

Requirements

Given these objectives, the following requirements must be met:

- Modify the counting module to adjust the measurement unit from centimeters to tenths of a millimeter.
- Modify the BCD-to-4-digit conversion component.
- Implement the Python notebook to acquire distance data.
- Adapt the data transmission format from the sonar to include additional decimal places resulting from the increased measurement precision.
- Develop calculation modules for the Bayesian filter.

Out of Scope

Having defined the project scope, we must also specify what is out of scope and what is not expected after the development and application of the project.

- Acquisition of distances at high speeds.
- Testing with different ultrasonic sensors (other manufacturers or models).
- Distance calculations above 4 meters.

1.1.2 Week 2 – Development of the Bayesian Filter in Software and Planning of Hardware Modules

To create a consistent model capable of generating fast feedback, it was decided to initially develop a Bayesian filter model in software that simulates real-world scenarios and performs data acquisition with the FPGA. This model will allow a better understanding of the impact of parameter changes and help determine the hardware structure of the filter.

Objectives

The objectives for this week are as follows:

- Test the HC-SR04 sensor with small physical perturbations (simulating the sensor embedded in a moving car on an irregular road).

- Implement the Bayesian filter calculations in Python (as outlined in [1]).
- Conduct distinct test cases for data acquisition with the FPGA.
- Plan the construction of the Bayesian filter circuit in hardware.

Requirements

To achieve the above objectives, the following requirements need to be met:

- Identify libraries that can assist with the filter modeling.
- Determine the most efficient way to simulate physical disturbances.
- Implement fixed-point or floating-point calculation modules for data manipulation.
- Create methods to adapt the software model for hardware implementation.

Out of Scope

The following items are out of scope for this phase of the project:.

- Continuous calculation of Bayesian filter parameters in hardware (the values for σ will be tabulated).
- Data manipulation with words larger than 64 bits.

1.1.3 Week 3 – Implementation of the Bayesian Filter in Hardware

After verifying the correct functioning of the software model, the next step is to implement the Bayesian filter on the FPGA. This hardware-based model will be fully implemented in hardware, and all validation tests will be performed using the FPGA setup. The HC-SR04 sensor will be attached to the servomotor for data acquisition in the sonar system. The sonar system should be sensitive enough to reduce its rotation speed to mitigate disturbances during the acquisition of proximity data.

Objectives

The objectives for Week 3 are as follows:

- Implement arithmetic blocks, components (memory, registers), and state machines for the Bayesian filter calculations.
- Implement dynamic velocity determination for the sonar's rotation speed.
- Conduct distinct test cases for data acquisition using the FPGA.
- Plan the final verification test cases for the proposed solution.

Requirements

The following requirements must be met to achieve the objectives:

- Identify and build the necessary arithmetic blocks for constructing the filter.
- Integrate the FPGA board and all components used in the project proposal.
- Assess the impact on data acquisition by the sonar system.
- Develop methods to validate acquired data during the prototyping validation tests.
- Evaluate the suitability of the hardware for performing the proposed validation tests.

Out of Scope

The following item is out of scope for this phase:

- Performing all validation tests for the proposed solution.

1.1.4 Week 4 – Final Testing and Analysis of Results

In Week 4, the planned tests will be conducted to validate the proposed solution in [1]. It is expected that the hardware integration of all components will be functioning correctly. Based on the data collected during testing, the proof of concept presented in the project will be analyzed.

Objectives

The objectives for this week are:

- Build the physical experimental environment for controlled data acquisition.
- Validate the final test plans.
- Conduct the tests, collect data, and analyze the results.

Requirements

The following requirements must be met:

- Identify all materials required to build the testing environment.
- Conduct the tests within the controlled environment.
- Analyze the data collected during the controlled tests.
- Validate the results with course collaborators.

Out of Scope

The following item is out of scope for this phase:

- Execution of the test plan on a real vehicle.

1.2 Method and Approach

To successfully complete the final project for the course, a consistent method coupled with a practical, direct, and efficient approach is essential. This allows for the smooth execution of activities and the proper functioning of the developed components. Given that the team has access to a lab-replica setup, most of the application tests will be conducted at one team member's home. This decision minimizes the team's dependence on course collaborators, as the project requires multiple tests, validations, and repositioning of objects to demonstrate the success of the proposed solution.

The circuit, programmed and activated on an FPGA board, will be designed using **Intel Quartus Prime** for creation and assembly, and **ModelSim** for testing. For physical

prototyping, the system will use an ultrasonic sensor (HC-SR04), servomotor (SG90), a Wemos D1 board (ESP8266), and various components for assembly and positioning.

The project design process is built on four essential pillars to achieve the final goal:

- *Iterative and incremental development.*
- *Consistent and comprehensive testing.*
- *Documentation of each completed stage and modification.*
- *Good communication among team members.*

For the successful completion of the project, it is assumed that the implementation will meet all listed requirements, making it possible to collect and analyze data that confirms the hypothesis from [1]. For each new layer of complexity added, new tests will be created to ensure proper integration. If any implementation problems arise, minor adjustments will be made to the schedule to avoid delaying the final submission.

1.3 Discussions

The development of this project involves certain risks and unresolved questions, which will be addressed and mitigated throughout the development process.

1.3.1 Risks

Creating and implementing hardware requires extra caution, as the cost of errors can be significant, especially in large-scale production. Some of the risks inherent in the design and use of hardware components in the project include:

- Non-functioning FPGA board or Analog Discovery (if used), which could delay the development, testing, and verification processes.
- Conflicts between different versions of Intel Quartus Prime, which could corrupt files or cause malfunctioning components on computers with non-standard versions.
- New implementations modifying the pre-existing circuit, causing previously functioning parts to fail.

- Internet or connection issues, given the context of the pandemic, which could cause delays in the project.
- Problems with the logistics of acquiring physical components, such as delivery issues or component malfunction.

1.3.2 Open Questions

Some aspects of the circuit's development are still under discussion to ensure that the final implementation can effectively verify the proper functioning of the Bayesian filtering process:

- Does the addition of other sensors improve the implementation's performance (e.g., temperature sensors, accelerometers)?
- What is the impact of remote data acquisition? Specifically, the implementation of a remote interface with the FPGA board via MQTT protocol.

2 Model Description

2.1 Bayesian Networks

The modeling is based on a series of Bayesian networks, or a hidden Markov model, as shown in Figure 2.

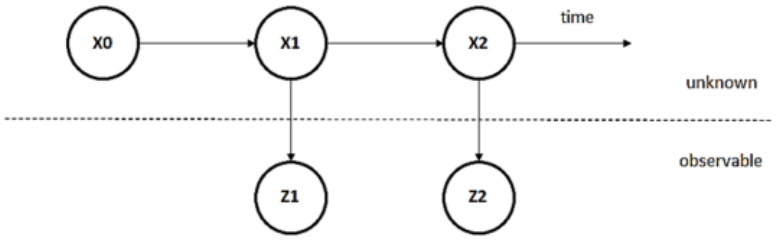

Figure 2: Model based on series of Bayesian networks.

The Markov chain modeling applied to the context of the problem presented relies on two considerations, as shown in Figure 3. The first is that each estimate depends exclusively on the previous estimate. The second is that the estimate from the filter depends on the value acquired by the sensor.

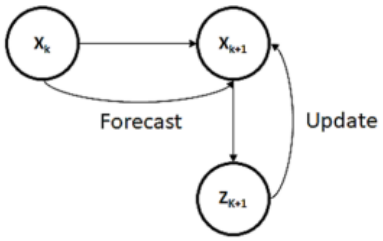

Figure 3: Prediction and update model.

The filtering iteration process uses five variables: σ_p^2 (process variance), σ_m^2 (measurement uncertainty), $k_{k|1}$ (constant associated with the estimate at iteration k), x_k (estimated value at iteration k), and $z_{k|1}$ (observed measure by the sensor at iteration k). These variables are used in equations 1, 2, and 3.

Equation 1 shows how the constant $k_{k|1}$ is calculated, which is used in determining the estimated measure from equation 3. Notice that the dependency of $k_{k|1}$ is only on the k iteration (previous iteration).

$$k_{k|1} = \left(\frac{\sigma_k^2 + \sigma_p^2}{\sigma_m^2 + \sigma_p^2 + \sigma_k^2} \right) \tag{1}$$

Equation 2 shows how the variance $\sigma_{k|1}^2$ is calculated. Notice that the variable $\sigma_{k|1}^2$ depends on the variable calculated in 1 and σ_k^2 from the previous estimate.

$$\sigma_{k|1}^2 = (1 - k_{k|1})(\sigma_k^2 + \sigma_p^2) \tag{2}$$

Finally, equation 3 results in the determination of the inferred value through filtering, using the variables from both the current and previous states.

$$x_{k|1} = k_k z_{k|1} + (1 - k_{k|1}) x_k \tag{3}$$

2.2 Hybrid Implementation - Data Acquisition and High-Level Processing

2.2.1 Design Decisions and Operation

The circuit design is based on previously developed elements from earlier experiments in the course. The development of the new circuit is supported by modules that already have some sonar functionalities. The sonar circuit thus encapsulates these elements by applying an organization divided into the control unit and the data flow. The reused modules available for this project are:

- UART 8N2
- Ultrasonic sensor interface - HC-SR04

The modules used are declared in the data flow of the main entity, with all control signals coming from the control unit. The control unit is composed of four states: *Initial*, *Transmission*, *Wait*, and *Final*. The expected sequence of operations involves the measurement of distance using the ultrasonic sensor, followed by the transmission of the data (distance in millimeters) via serial.

In summary, the organization of all the modules and the base code structure were implemented with the aim of speeding up the process of modification and integration of new components and functionalities within the time constraints set by the course instructors.

The next step in the project is to utilize the previously developed structure to achieve new results and conduct further analysis. By automating the acquisition of measurements through a Python script using the *paho.mqtt* library, it will be possible to reproduce real-world scenarios of circuit use. In addition to these components, the project also relies on auxiliary projects, such as Arduino-based setups, responsible for the movement of a device to acquire relative distance measurements with a moving sensor.

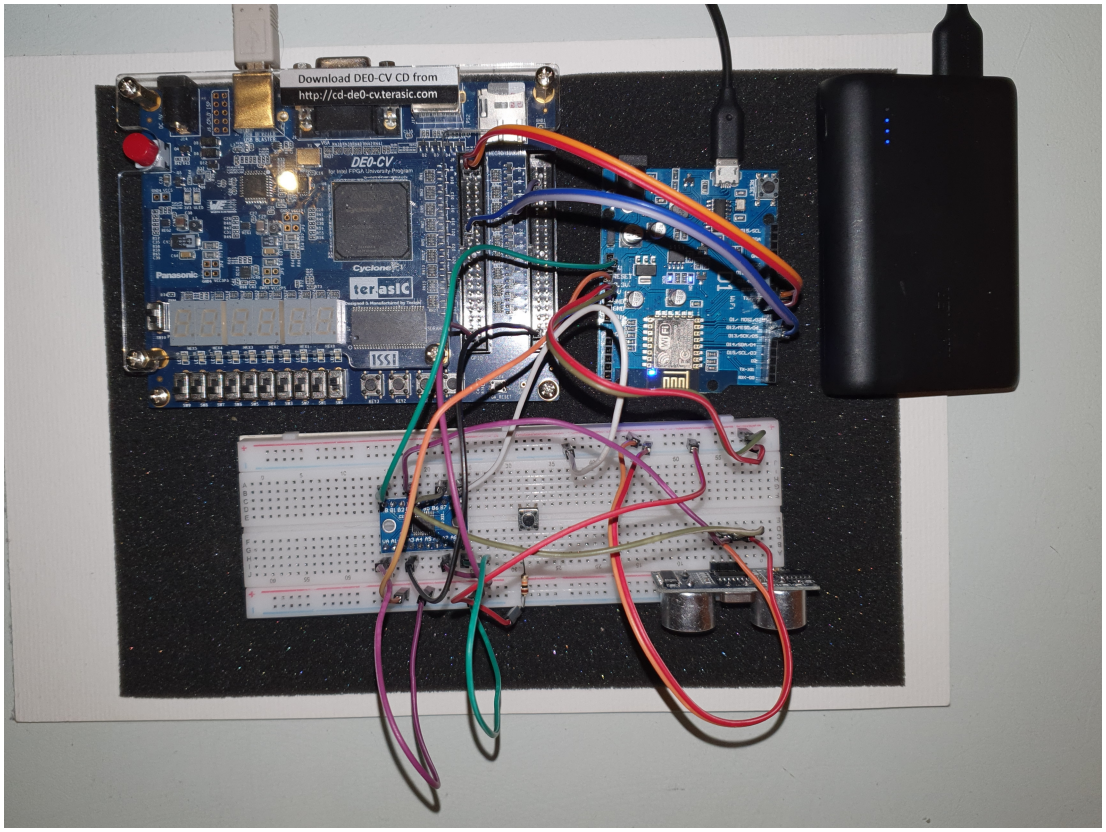

Figure 4: Photo of the FPGA circuit built by the team.

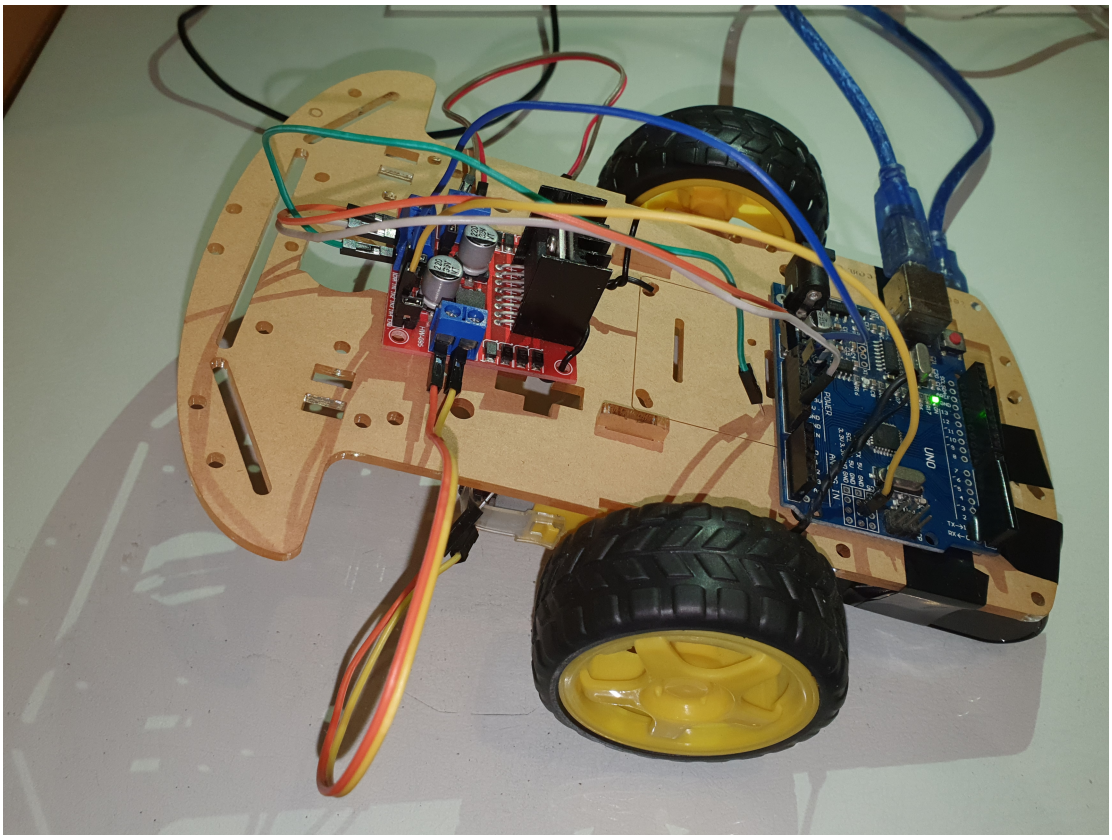

Figure 5: Photo of the cart built by the team for acquiring relative distance measurements.

2.2.2 Data Flow of the Pilot Project

The organization of the circuit design was carried out by dividing it into two blocks: the Control Unit and the Data Flow, as shown in the figure below.

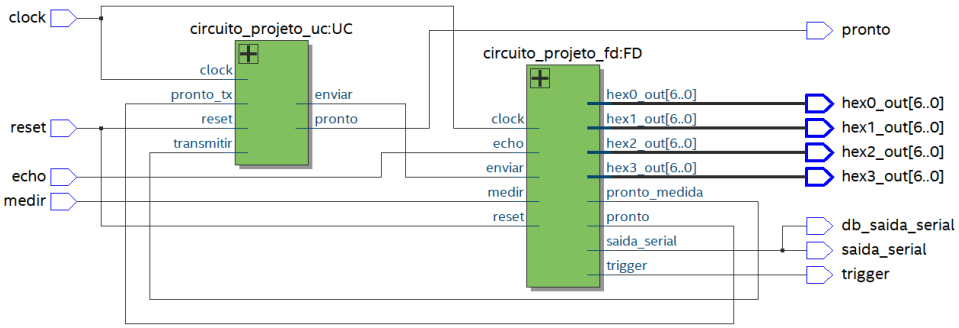

Figure 6: Block diagram of the intelligent sonar project.

The data flow design consists of three main elements:

- Ultrasonic sensor interfacing module
- Serial transmission module (in the ”d3d2d1.d0” *”d3d2d1.d0”* format);

- BCD Counter

Unlike previous implementations in the course, the format of the transmitted data has been altered for two main reasons:

- To align the transmitted distance measurement data with the floating-point format used in Python (the transmitted data arrives in the Python script already in *float* format).
- To increase the measurement precision in order to identify noise more clearly.

These modules execute the primary functionality of the test circuit, which is acquiring the distance measurements in tenths of millimeters, transmitting the data in serial format, and making it available for interfacing with a Python script. The diagram with the data flow components of the project can be observed below.

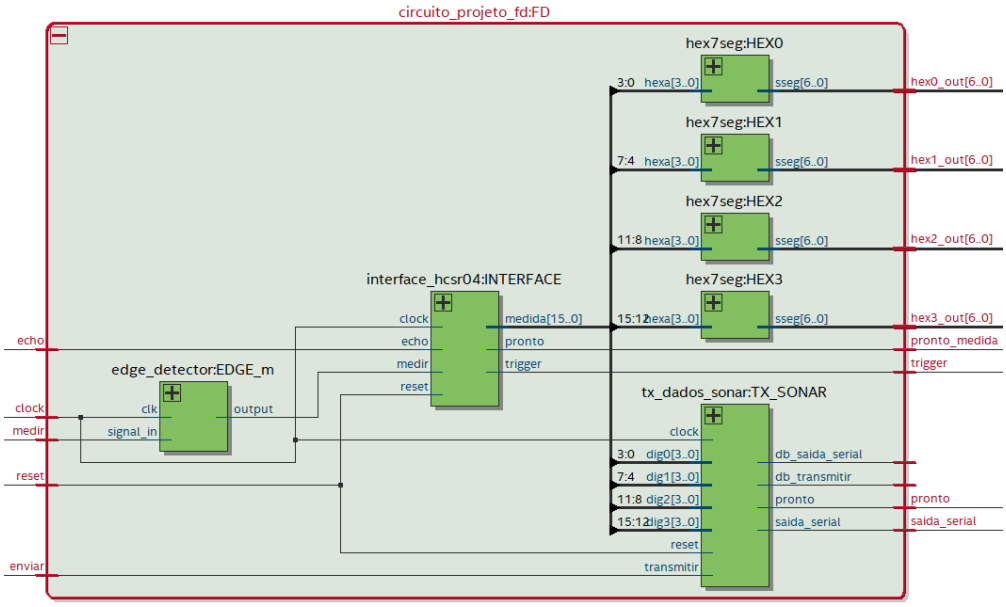

Figure 7: Data flow of the intelligent sonar circuit.

2.2.3 Control Unit of the Pilot Project

The Control Unit of the pilot project consists of four distinct states responsible for controlling the input signals of the system components. Overall, its operation consists of repeating three main execution blocks of the circuit: *distance measurement by the ultrasonic sensor* and *data transmission via UART 8N2*.

The circuit begins operation with the activation of the *measure* signal, which initiates the distance measurement. Then, the data transmission begins with the activation of the *measurement_ready* signal. Once the transmission is complete, the circuit triggers the *ready* output.

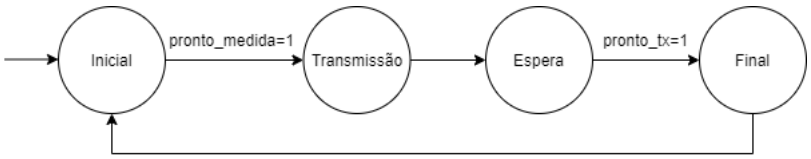

Figure 8: State transition diagram of the control unit.

2.2.4 Data Processing

In the current stage of development, the project simplifies the implementation of high-level logic. The data acquisition layer is performed by the FPGA hardware, through the interfacing with the ultrasonic sensor (HC-SR04). The measured data is transmitted via the UART module, using the MQTT protocol. Data reception is handled by a Python script subscribed to the transmission topic of the broker. Through this subscription, data is accessed through a Byte-type variable. In this Python script, in addition to the initial data processing (conversion from Byte to float), the Bayesian filter algorithm described in [1] is implemented.

2.2.5 Data Acquisition Test

Given that the team has a setup similar to that of LabEAD, with an FPGA board, Wemos D1 board, and the ultrasonic sensor (along with other auxiliary components like a level-shifter for voltage adjustments between components), automated tests were conducted with the proposed distances.

In these tests, 1,000 distance samples were automatically acquired. The graphs can be observed below:

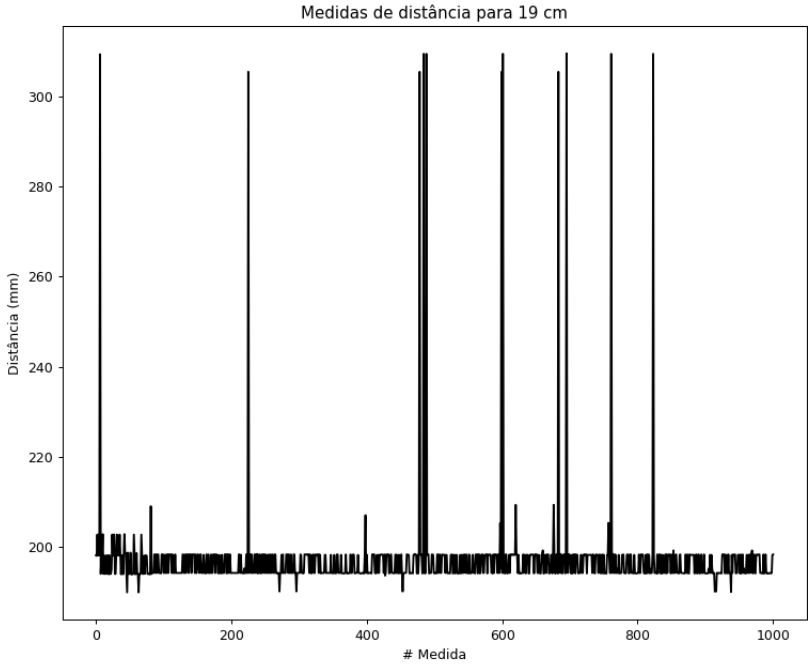

Figure 9: Testing of data acquisition for distances in the range of 19 cm.

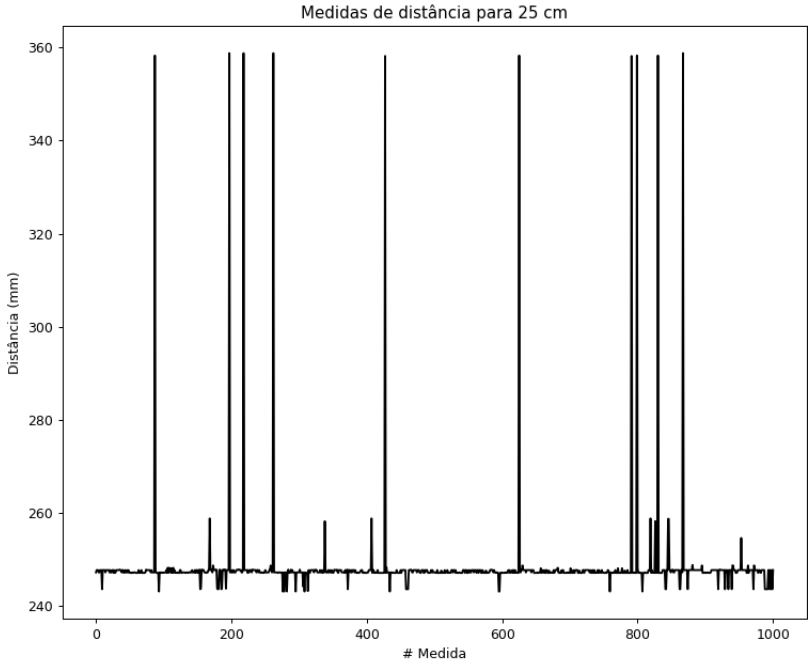

Figure 10: Automated acquisition of distances with the FPGA circuit for an object positioned 25 cm from the sensor.

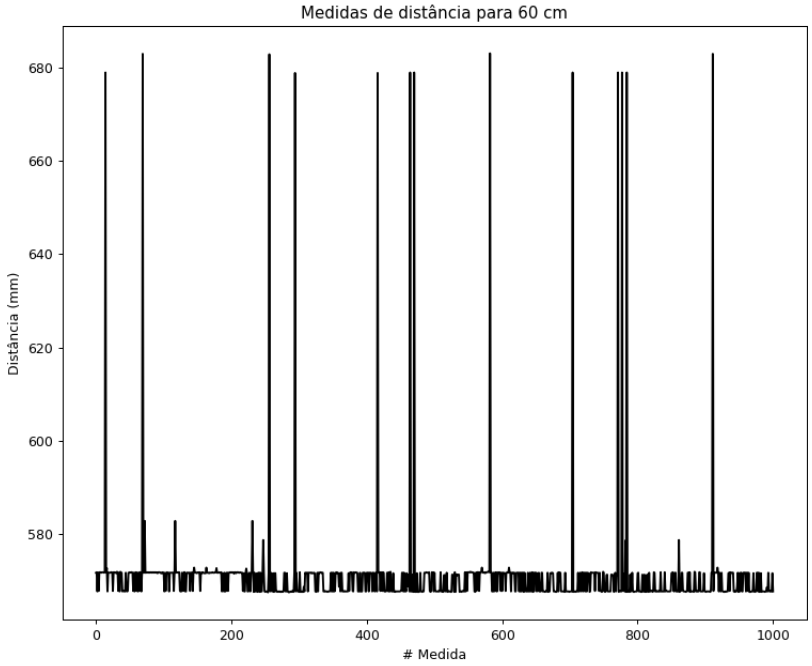

Figure 11: Automated acquisition of distances with the FPGA circuit for an object positioned 60 cm from the sensor.

It is noticeable that, in the case where the object is positioned 60 cm away from the sensor, there is a measurement error of 3 cm. This discrepancy may stem from several factors that were not initially considered, such as temperature and humidity in the experimental environment. These factors influence the speed of sound, which in turn affects the calculated distance between the sensor and the object.

2.3 Initial Testing and Validation

2.3.1 Bayesian Filter Applied in Software

Based on the distance data obtained through the serial transmission performed by the circuit, graphs were plotted comparing the curves of the sensor's measured data (in red) with those obtained from the filter output—i.e., the estimates—represented in black.

The Bayesian filter implemented utilizes the equations 4, 5, and 6 presented in the previous section. To execute the filter, several initial parameters are provided: a .csv file with the measurements obtained by the circuit, values for σ_p^2 (process variance), σ_m^2 (measurement uncertainty), a seed distance to start the process, and an initial value for

σ_0^2 for the iterations of σ_k^2.

***** *The code used is available in the file folder with the titles "mqttinterface.py" and "dataplot.py".*

2.3.2 Sensor Test with Physical Perturbations

1. **Tests with Arduino UNO**

   The tests conducted on the Arduino UNO aim to serve as a reference for the implementation on the FPGA board. Two test scenarios were performed, with the object placed at 31 cm and 71 cm. In both scenarios, the sensor was fixed on a breadboard. For each distance, two tests were conducted: the first without physical disturbance and the second with a physical disturbance generated by a mobile phone vibration motor.

   In each of the four tests, one thousand measurements were taken. Data acquisition was automated using the Arduino in a loop connected to a Python script via a serial port. This Python script stores the data for each test. The results are summarized in Table 1 below.

	31 cm (Without Vibration)	**31 cm (With Vibration)**	**71 cm (Without Vibration)**	**71 cm (With Vibration)**
Mean	31.0	31.0	71.5	70.7
Median	30.9	30.9	71.8	72.5
Variance	0.04	0.03	47.24	50.52
Standard Deviation	0.2	0.2	6.9	7.1

Table 1: Test results for ultrasonic sensor (HC-SR04) - Arduino UNO - 10k acquisitions for each column.

From the collected data, it can be observed that for the measurements taken at 31 cm from the wall, the scenario with vibration exhibits, counterintuitively, a smaller variance than the scenario without vibration. On the other hand, the test at 71 cm from the wall presents results consistent with expectations, showing a higher variance in the case with vibration. Finally, the average observed for all cases is very close to the expected value. Only the two cases with greater distance show a slight deviation from the expected mean.

***** *The simulation environments, both static and under vibration, were used without*

external interferences such as changes in temperature, position, or humidity. The cause of the outliers is still unknown.

2. **Tests on the FPGA Board**

   The tests with the FPGA board were divided into two stages: (i) acquisition of distance measurements with the static sensor, and (ii) acquisition of distance measurements under forced perturbation conditions. Stage (ii) aims to simulate real-world conditions in which the ultrasonic sensor may be part of a vehicle, drone, or other electronic measurement component. In such cases, adverse conditions like bumpy roads, strong winds, and motor vibrations are commonly observed, which can impact the reliability of the measurements.

   For the case study in this project, the circuit with the ultrasonic sensor was placed on a surface in contact with equipment that generates forced vibrations (a massager).

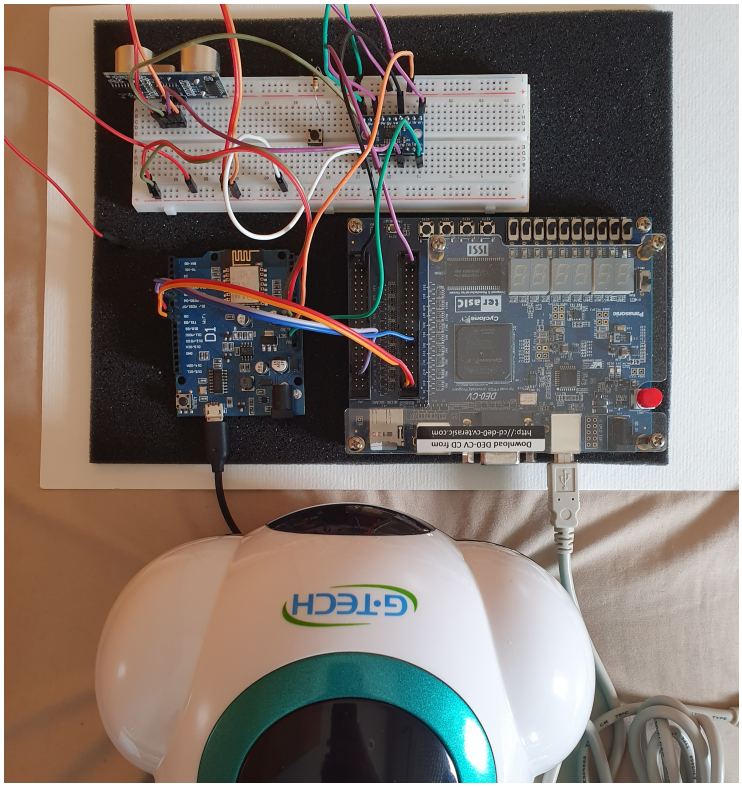

Figure 12: Vibration test environment.

The following results were observed:

**Note: The data in red were obtained by the FPGA and transmitted to the script, while the black data represents the estimates produced by the Bayesian filter.*

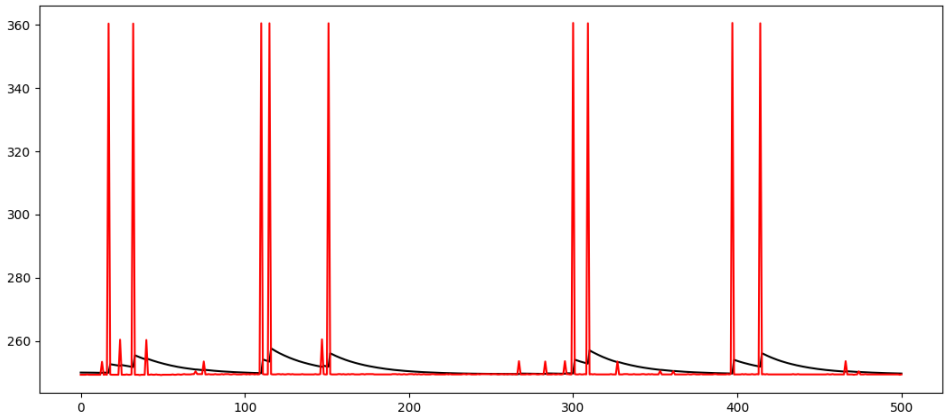

Figure 13: Distance measurements acquisition with the object positioned 25 cm from the sensor, in the absence of vibration.

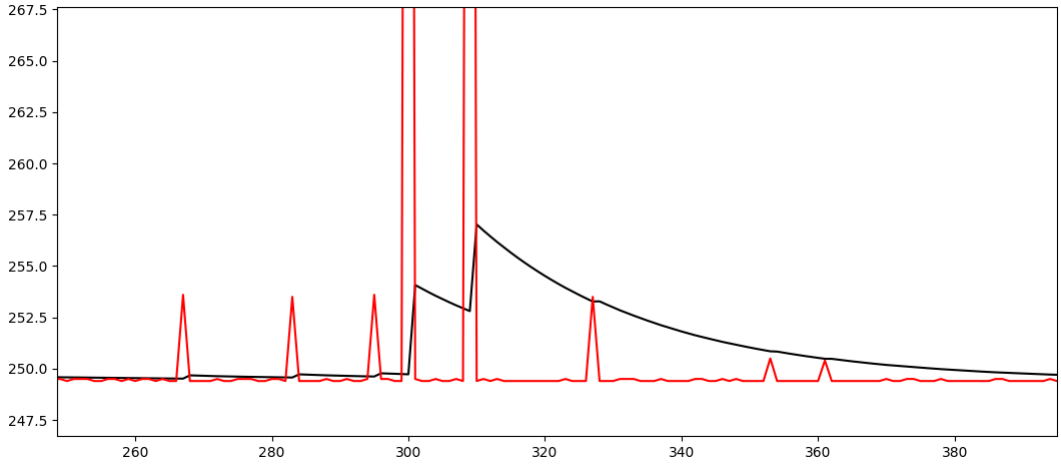

Figure 14: Zoomed-in view of distance measurements acquisition, with the object positioned 25 cm from the sensor, in the absence of vibration.

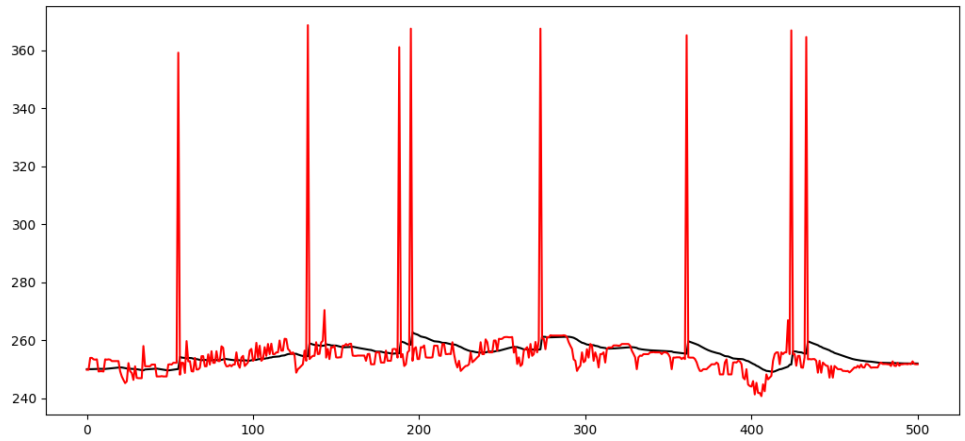

Figure 15: Distance measurements acquisition with the object positioned 25 cm from the sensor, under vibration.

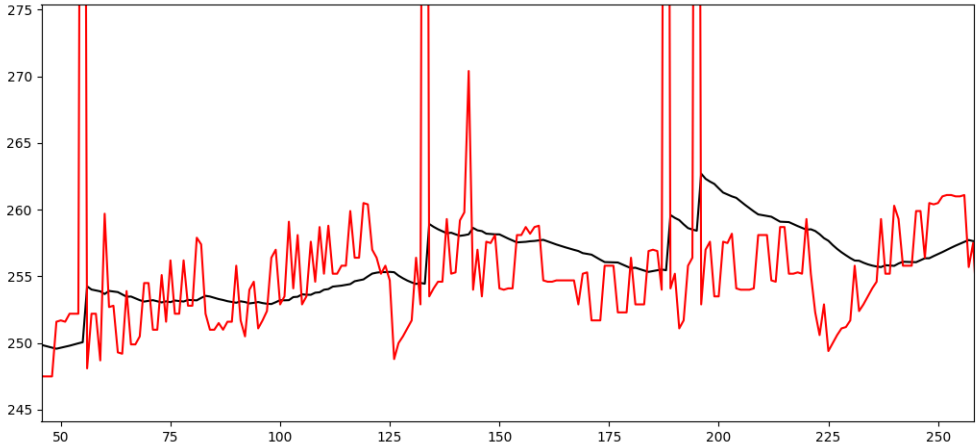

Figure 16: Zoomed-in view of distance measurements acquisition, with the object positioned 25 cm from the sensor, under vibration.

In both FPGA circuit test cases, it can be observed that the Bayesian filter delivered good results, effectively attenuating small vibrations and outliers. Moreover, the estimated values produced by the filter appear to be more stable than those obtained in the field, showing minimal variations. The variance of the estimates observed was approximately 10 times smaller than that of the field measurements.

One issue that remains to be analyzed and discussed is the behavior of the filter after the occurrence of an outlier. In Figure 14, this characteristic can be observed in more detail. With the current sampling rate, the filter takes several seconds to

return to its baseline values. Since the considered period includes the time for data transmission and estimation calculations, it is possible that hardware implementation may improve the results.

***** *The simulation environments, both static and under vibration, were used without external interferences such as changes in temperature, position, or humidity. The cause of the outliers is still unknown.*

3 Description of the Digital Circuit

3.1 Presentation of the Models of the Main Circuit Components

For the construction and development of the circuit, the IEEE-754 standard for single precision floating-point data and operations was adopted.

The modules are as follows:

1. Floating-point Bayesian filtering calculation module
2. Ultrasonic sensor signal conversion module to floating-point
3. Adaptive control module - modification of the Bayesian filter parameters

The subsequent sections will present and discuss the key concepts used in the development of each module.

a) **Floating-Point Bayesian Filter Calculation Module**

(a) **Design Decisions and Operation**

The calculation module consists of the implementation of the three main equations that comprise the Bayesian filter [1], as presented below:

$$k_{k|1} = \left(\frac{\sigma_k^2 + \sigma_p^2}{\sigma_m^2 + \sigma_p^2 + \sigma_k^2} \right) \tag{4}$$

$$\sigma_{k|1}^2 = (1 - k_{k|1})(\sigma_k^2 + \sigma_p^2) \tag{5}$$

$$x_{k|1} = k_k z_{k|1} + (1 - k_{k|1}) x_k \tag{6}$$

The distance values, seed (initial value to start the first iteration), and sigmas are provided by other components of the intelligent sonar system.

The development was carried out modularly, with each equation implemented in a distinct component. The outputs of these components were thoroughly

tested to ensure correct operation. The following figures present the RTL (Register Transfer Level) diagrams of each of these components:

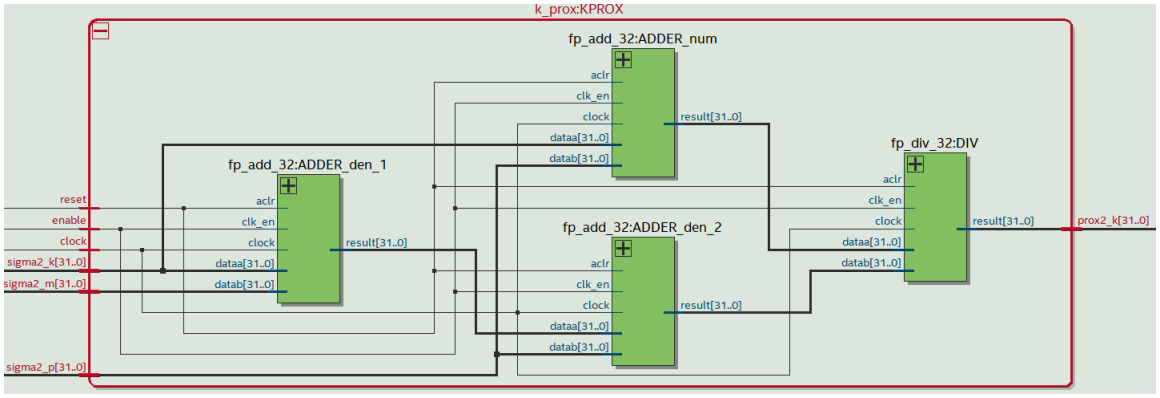

Figure 17: RTL Viewer of the component implementing equation 4.

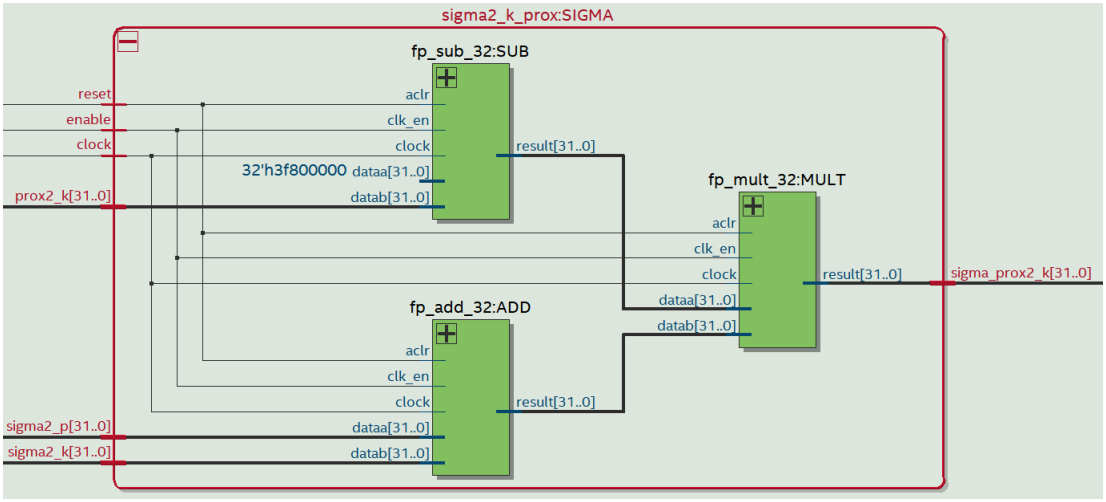

Figure 18: RTL Viewer of the component implementing equation 5.

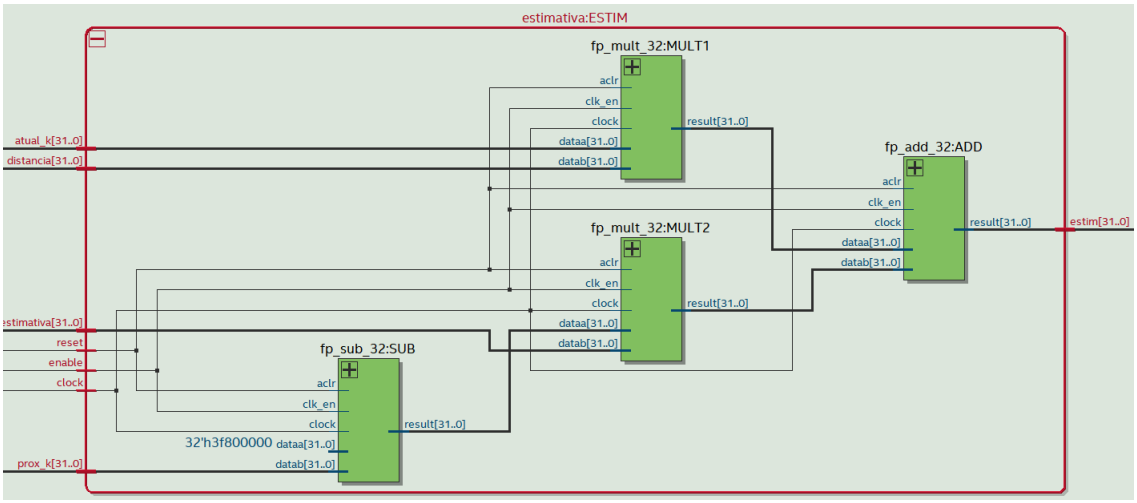

Figure 19: RTL Viewer of the component implementing equation 6.

After confirming that the components functioned correctly, they were integrated into the calculation module. This integration required adding auxiliary registers and multiplexers (MUXs) to select and store values for use in the iterations of the calculations. Additionally, a control unit was developed to manage the selection signals of the MUXs and register write signals.

The details of the complete assembly, including the Data Flow and Control Unit, will be presented below.

(b) **Complete Circuit of the Calculation Module**

The organization of the floating-point Bayesian filter calculation module was achieved by dividing it into two blocks: the Control Unit and the Data Flow, as shown in the figure 20.

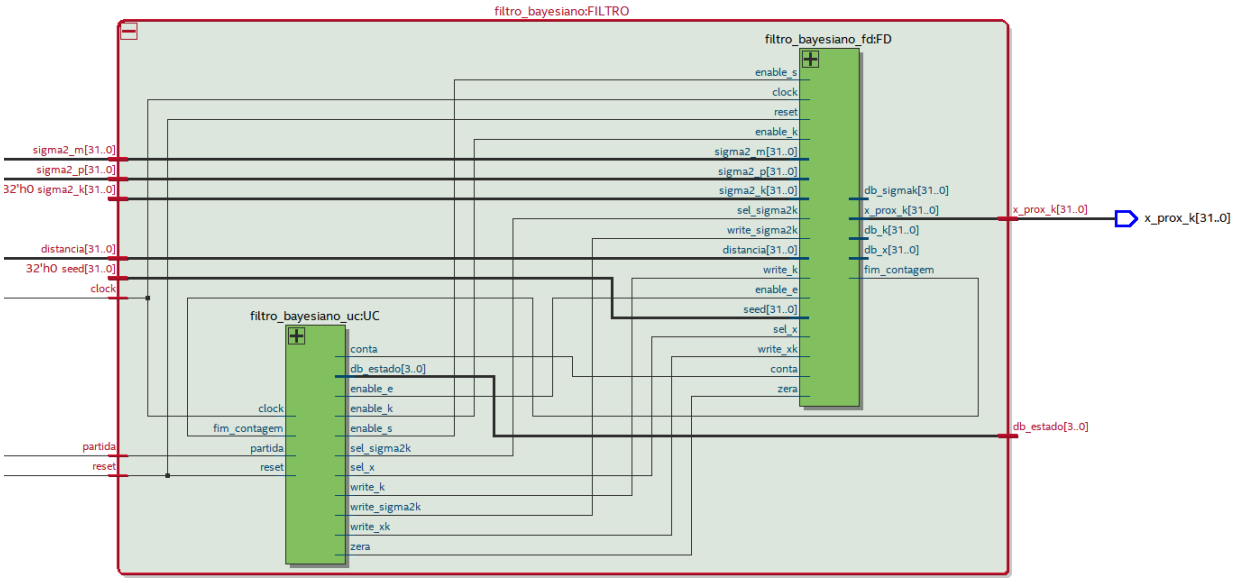

Figure 20: Complete RTL Viewer of the Bayesian Filter Calculation Module.

(c) **Data Flow of the Floating-Point Calculation Module**

The Data Flow of the calculation module encompasses the components that implement the three equations presented earlier, along with auxiliary components for storing values from different iterations.

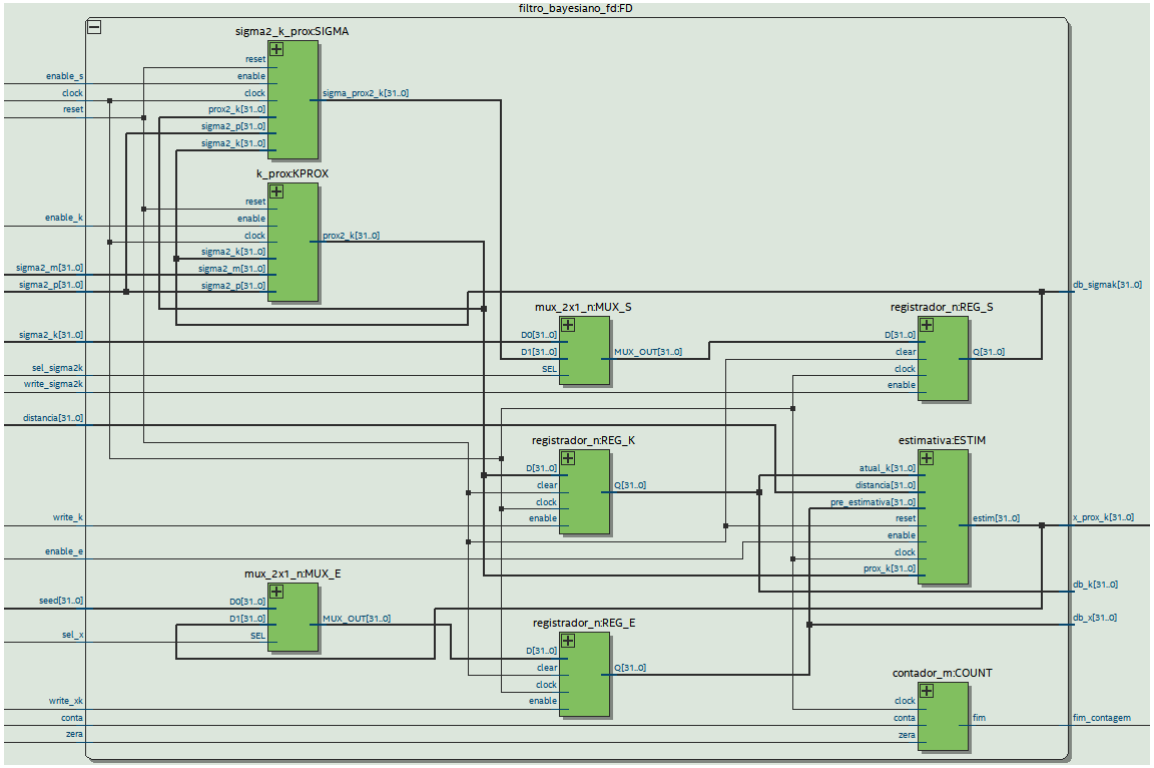

Figure 21: RTL Viewer of the Data Flow of the Bayesian Filter Calculation Module.

(d) **State Diagram of the Control Unit**

The Control Unit of the design consists of 11 distinct states responsible for managing the input signals to the components of the Bayesian filter calculation module.

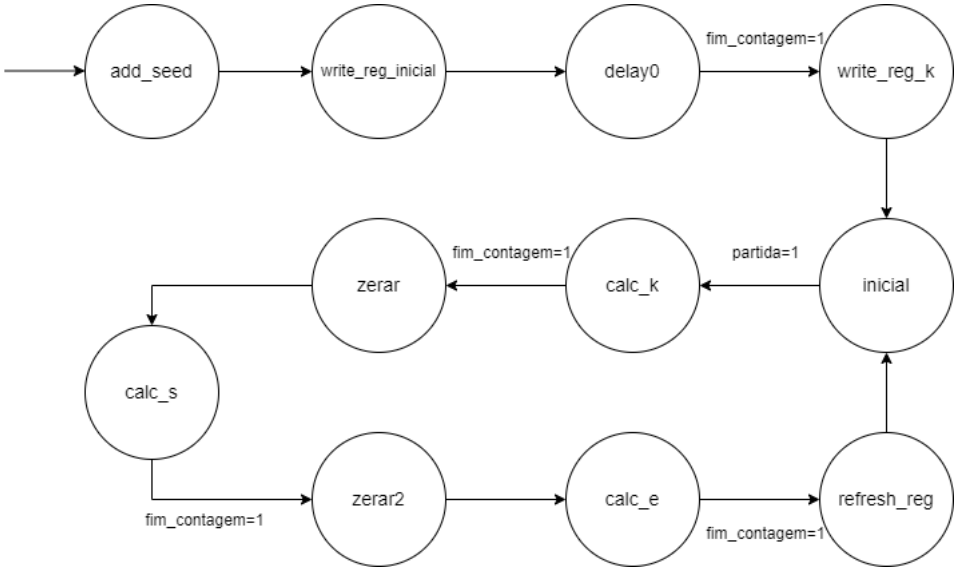

Figure 22: State Transition Diagram of the Control Unit.

The first four states correspond to the storage of the initial iteration values of

the Bayesian filter (seed, k_0, and σ_0). Once these values are stored, the circuit waits for a distance value (from the distance calculation module), which triggers the *start* signal. The *calc* states represent 1.5-microsecond delays added to ensure no ambiguity in the output values of the modules implementing the filter equations. The reset states clear the delay counter for the next state, and finally, the *refresh_reg* state updates the values of σ, k, and x in the registers.

b) **Floating-Point Distance Measurement Module**

(a) **Design Decisions and Operation**

The conversion module implements the transformation of the measurement taken by the ultrasonic sensor into a floating-point representation. The sensor interface component (hc-sr04) used in this development is based on the one designed in Experiment 4 of Part 1 of the course. No structural changes were made to the interface; instead, a 32-bit counter was added to count the number of clock cycles corresponding to each measurement.

In the previous implementation (Week 4), the requirement was to convert the measurements to BCD format to display the result in decimal on the FPGA's displays. Additionally, the distance data was transmitted via serial communication using the MQTT protocol. The measurement's precision was on the order of centimeters, with a clock count of 2941 cycles representing approximately 1 centimeter of distance. This count corresponds to the FPGA's clock frequency (50 MHz) and the speed of sound in air (340 m/s at 20°C).

In the floating-point conversion module, there is no need for BCD conversion. Only the total clock count from the sensor's measurement is required. To achieve this, a 32-bit counter was added to the data flow of the sensor interface (see Figure 23), along with a 32-bit register to store the clock count (see Figure 24), which is then used as input to a floating-point multiplier for the conversion.

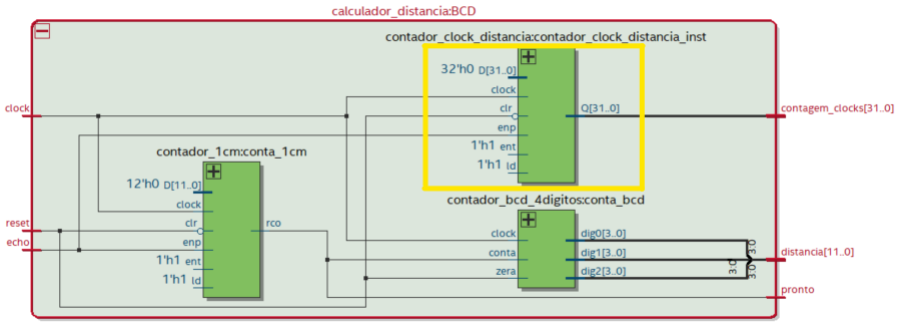

Figure 23: Block diagram showing the addition of the 32-bit counter to the distance calculation module of the hc-sr04 sensor interface.

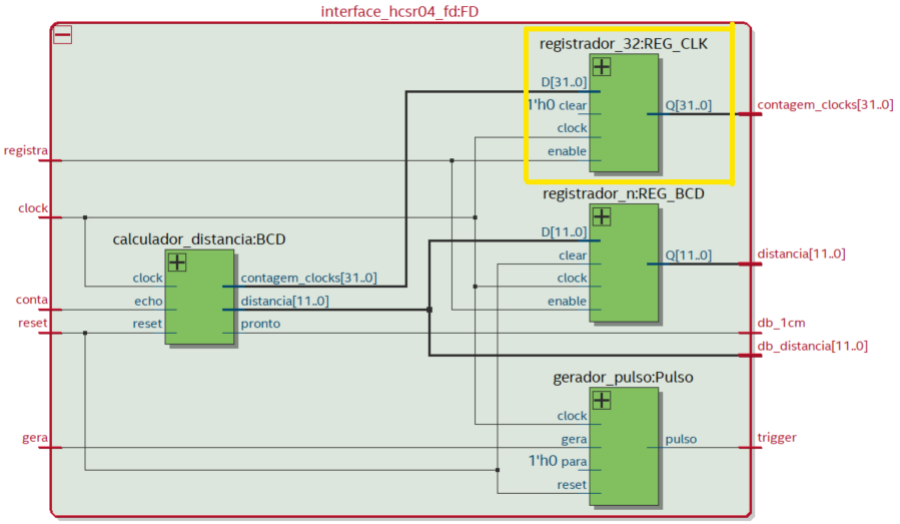

Figure 24: Block diagram showing the addition of the 32-bit register to the sensor interface's data flow.

The distance measurement module uses three main components: the modified hc-sr04 sensor interface, and the clock-to-floating-point conversion module.

The floating-point conversion module is a combinational circuit and, therefore, does not require a control unit. It consists of three components:

- An integer-to-single precision floating-point converter
- A 32-bit RAM
- A single-precision floating-point multiplier

Both the integer-to-floating-point converter and the single-precision multiplier were constructed using libraries provided by **Intel Quartus**.

The RAM stores constants used in the conversion from clock cycles to distance, allowing the measurement units (meters, centimeters, millimeters, etc.) to be

adjusted. The current implementation supports distances down to 10μ meters. The six constants stored in RAM are used in the transformation between clock counts and distance. The speed of sound in air is currently assumed to be 340 m/s at 20°C. If necessary, a temperature sensor could be added to adjust the constants.

The operation of the clock-to-distance floating-point converter is described as follows (Figure 25):

i. The integer-to-floating-point converter receives the 32-bit clock count

ii. The precision unit is selected in the main interface of the floating-point measurement module

iii. The conversion result, along with the constants from RAM, is fed into the single-precision multiplier

iv. The multiplier performs the operation, and the output is the floating-point representation of the distance measured by the sensor

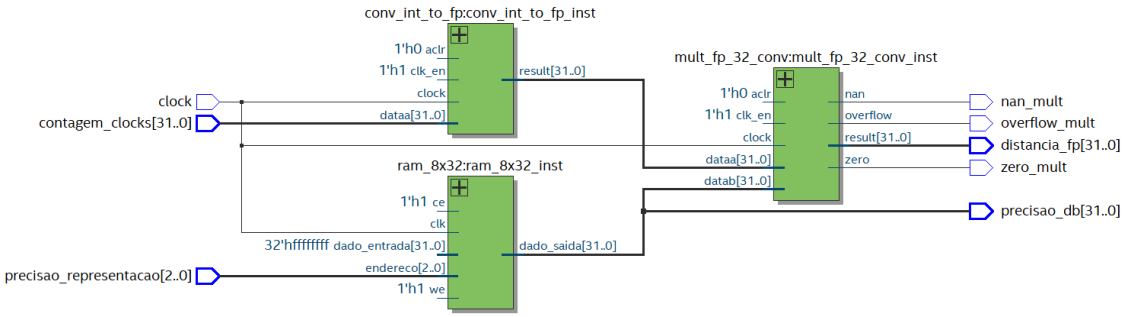

Figure 25: Block diagram of the clock-to-floating-point distance conversion module with single-precision (32 bits).

(b) **Data Flow of the Floating-Point Distance Measurement Module**

The floating-point distance measurement module is a combinational circuit that does not require a control unit.

The module's operation, shown in Figure 26, involves two components: the sensor interface and the conversion module for single precision. The interface performs the measurement and provides the result. The conversion module takes this value and converts it into floating-point representation.

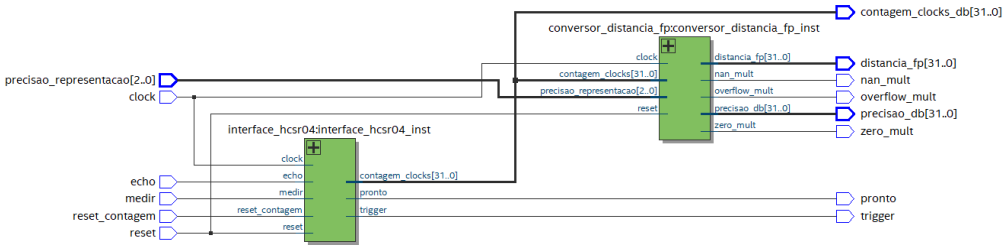

Figure 26: Block diagram of the main floating-point distance measurement module.

This floating-point distance measurement module serves as input to the Bayesian filter calculation module, as shown in Figure 20.

(c) **Control Unit of the Floating-Point Distance Measurement Module**

As the circuit of this module is combinational, there is no control unit within the module itself. The control unit shown in Figure 27 is part of a component used by the distance measurement module. This control unit manages the measurement process, which indirectly controls the main conversion module.

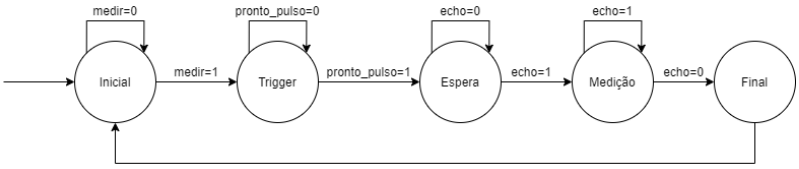

Figure 27: State diagram of the control unit for the hc-sr04 sensor interface circuit.

The Control Unit consists of five distinct states responsible for managing the ultrasonic sensor's measurement data. The initial state waits for the measurement to begin, controlled by the *measure* signal. The trigger state generates the trigger pulse in the data flow. After the pulse is generated, the circuit enters a wait state, waiting for the *echo* signal from the sensor. When the *echo* signal is HIGH, the distance measurement begins. Finally, when the *echo* signal goes LOW, the circuit finishes the execution, triggers the ready signal, and stores the measured distance value.

c) **Adaptive Control Module of the Bayesian Filtering Circuit**

(a) **Design Decisions and Operation**

The adaptive control module provides the values of σ_P and σ_M used by the Bayesian filter calculation module. Each RAM stores a set of values for each

sigma. The addressing of both RAMs is shared by the same input signal. This module, when combined with environmental adaptation logic, will enable the circuit to automatically determine the best parameters used in the Bayesian filter calculations presented in equations 4, 5, and 6.

It is important to note that the circuit is not yet fully automated. However, this module has already been constructed to vary the parameters for sensitivity testing of the FPGA implementation.

(b) **Construction of the Adaptive Control Module**

The module, shown in Figure 28, consists of two RAMs. By selecting the input address of the module, the RAMs provide the chosen σ_P and σ_M parameters.

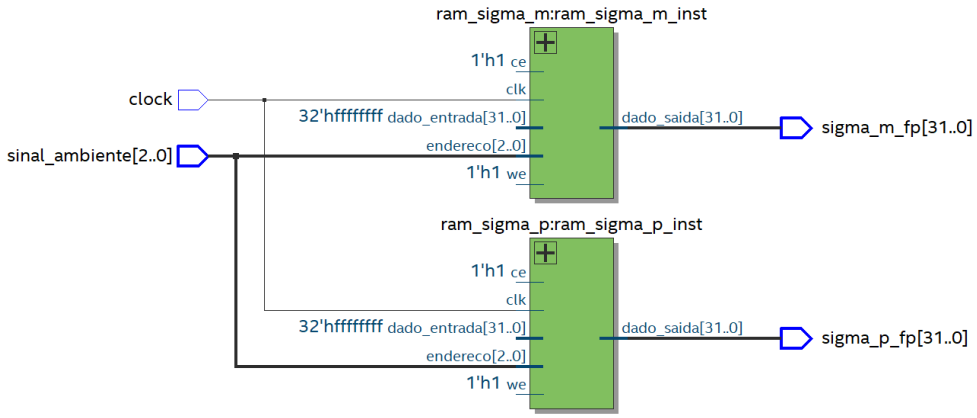

Figure 28: Block diagram of the environment decoding module - stores parameters σ_P and σ_M.

At the current stage of development, the parameters used are those presented in [1]. In the future, new parameters will be determined through simulations to improve the filter's performance in the physical test environment that will be constructed.

3.2 Development Tests

Below, the tests conducted for each module through simulations will be presented.

a) **Test of the Bayesian Filter Calculation Module in Floating-Point**

The tests involving the calculation module were divided into two main parts:

- **Modular Tests of the Components Implementing the Filter Equations**; For each component (KPROX, SIGMA, and ESTIM - see Figures 17 to 19), an operation was performed with floating-point precision, considering the following input data: $\sigma_{k0} = 0$, $\sigma_p = 0.01$, $\sigma_m = 0.25$, $seed = 250$, and $distancias = 250 \pm 0.5$. The results obtained were as follows:

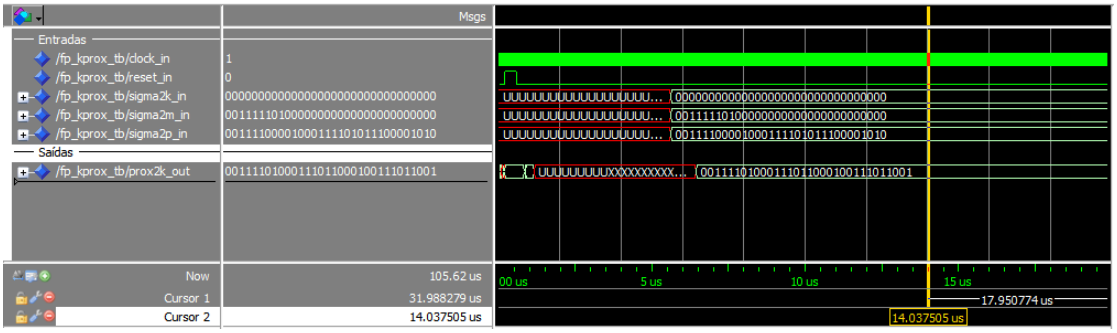

Figure 29: Test case with KPROX.

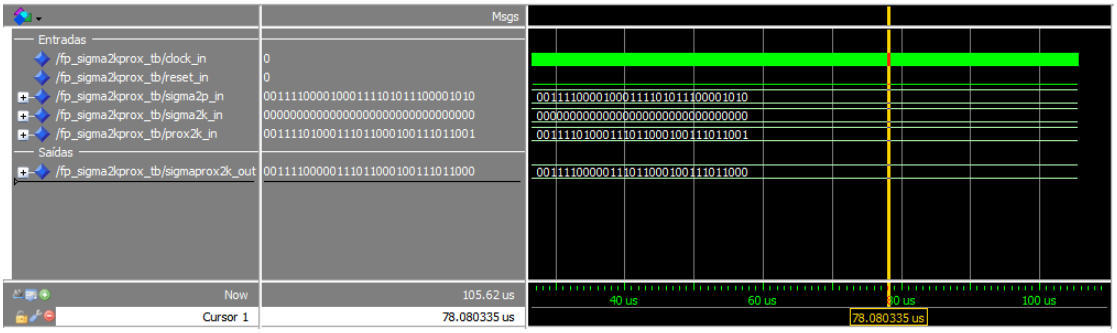

Figure 30: Test case with SIGMA.

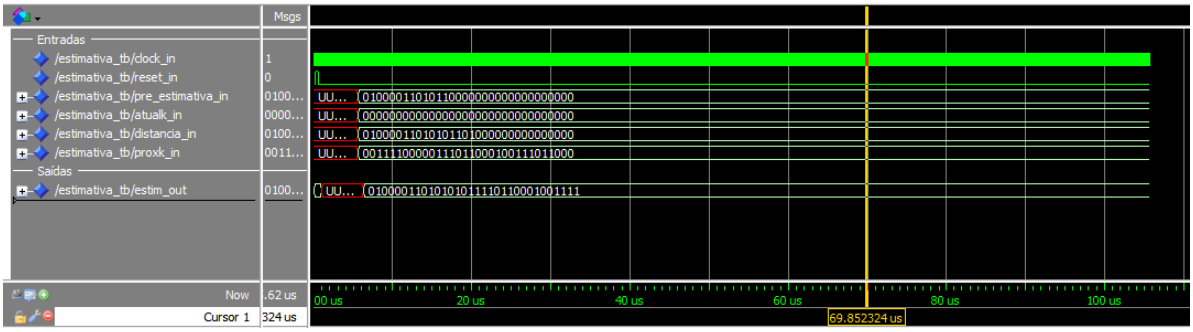

Figure 31: Test case with ESTIM.

When the floating-point IEEE-754 output values were converted to integers, the results were found to be as expected.

- **Integration Tests of the Components**

Once the modules were verified as working correctly, the behavior of the circuit integrating all components, along with auxiliary components such as multiplexers and registers, was tested.

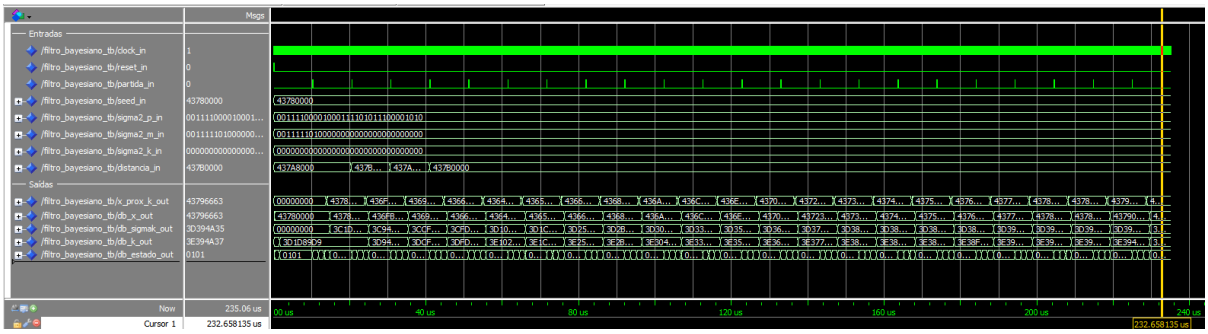

Figure 32: Integration test of the calculation module #1.

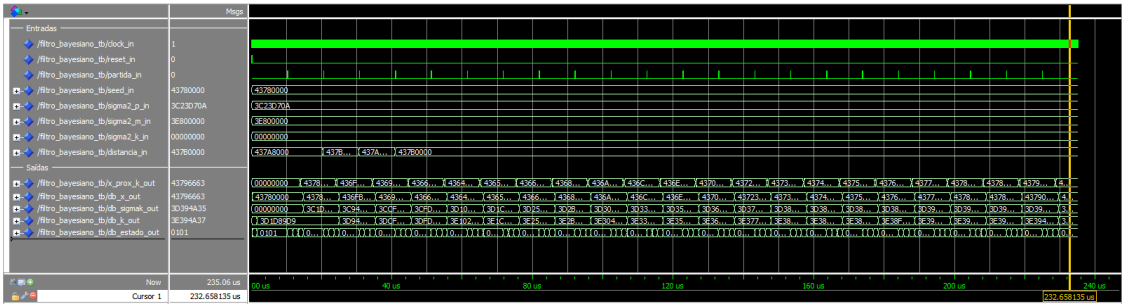

Figure 33: Integration test of the calculation module #2.

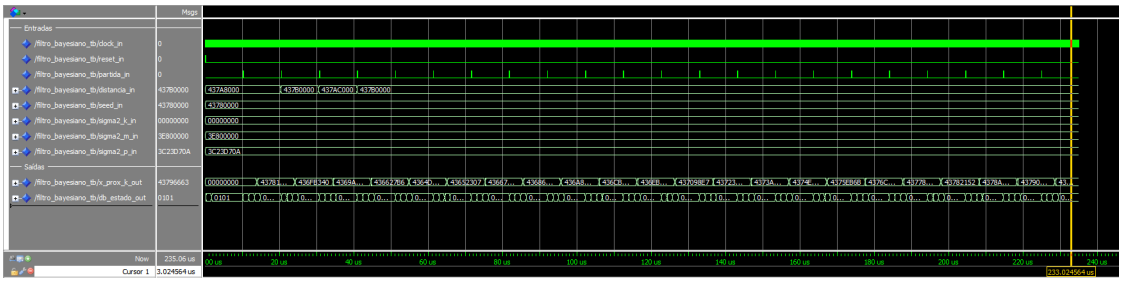

Figure 34: Integration test of the calculation module #3.

b) **Tests of the Ultrasonic Sensor Signal Conversion Module to Floating-Point**

The tests for the sensor signal conversion module were divided into two groups. The first group presents tests for the conversion to floating-point representation, while the second group shows tests for the distance measurement module, which integrates the HC-SR04 interface and the conversion module. For each test, the timing diagram and a table with the obtained and expected values are shown.

- **Test of the Clock Count to Floating-Point Conversion Module - Single Precision**

  The first test, shown in Figure 35 and Table 2, presents initial tests for conversion to centimeters. It can be observed that all results were compatible with expectations. Minor precision errors arise due to the digital circuit discretization and the single-precision representation. The input signal (clock count) and the output signal (conversion) from the module are both in hexadecimal format.

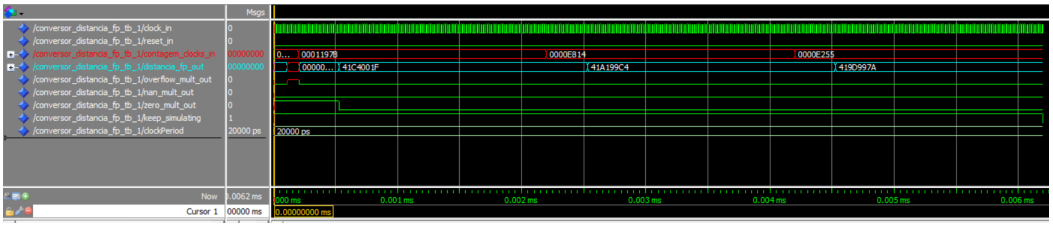

Figure 35: Test of the conversion module with precision in centimeters.

FP Hexadecimal Representation	Corresponding Decimal (cm)	Expected Result (cm)
41C4001F	24.50005913	24.5
41A199C4	20.20008087	20.2
419D997A	19.69993973	19.7

Table 2: Converter test - 3 measurements in centimeters.

Validation tests were then performed to check the inclusion of the RAM, which determines the representation type of the signals.

The first test of this batch is the operation test 19700, shown in Figure 36 and Table 3.

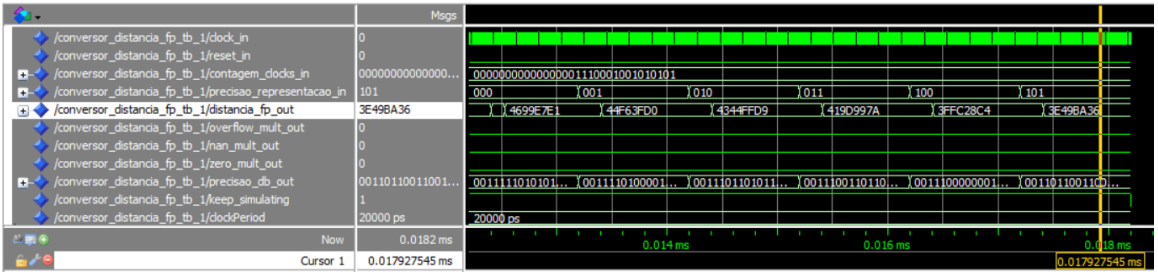

Figure 36: Test #1 - 19700 - variation of the 6 representation types.

FP Hexadecimal Representation	Converter Output	Corresponding Decimal (m)	Expected Result
4699E7E1	19699.94	19699.94 x 10E-5	19700.00
44F63FD0	1969.994	1969.994 x 10E-4	1970.000
4344FFD9	196.9994	196.9994 x 10E-3	197.0000
419D997A	19.69994	19.69994 x 10E-2	19.70000
3FFC28C4	1.969994	1.969994 x 10E-4	1.970000
3E49BA36	0.196999	0.196999	0.197000

Table 3: Converter test - measurement 19700 with 6 representations (hundredth of a millimeter, tenth of a millimeter, millimeter, centimeter, decimeter, meter).

The second test is the operation test 20200, shown in Figure 37 and Table 3.

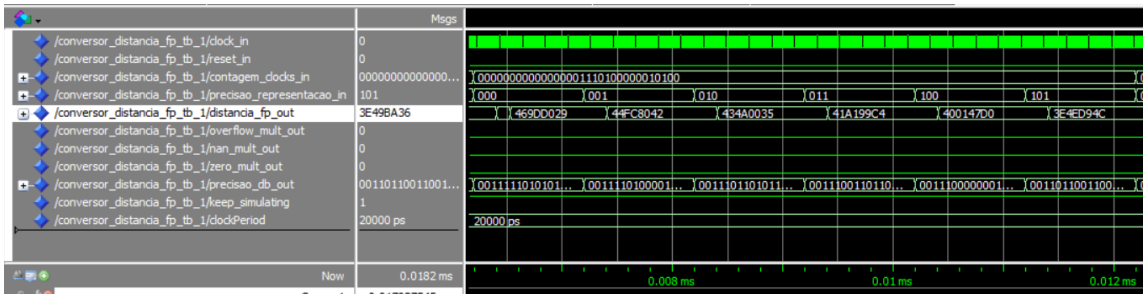

Figure 37: Test #2 - 20200 - variation of the 6 representation types.

FP Hexadecimal Representation	Converter Output	Corresponding Decimal (m)	Expected Result
469DD029	20200.08	20200.08 x 10E-5	20200.00
44FC8042	2020.008	2020.008 x 10E-4	2020.000
434A0035	202.0008	202.0008 x 10E-3	202.0000
41A199C4	20.20008	20.20008 x 10E-2	20.20000
400147D0	2.020008	2.020008 x 10E-1	2.020000
3E4ED94C	0.202001	0.202001	0.202000

Table 4: Converter test - measurement 20200 with 6 representations (hundredth of a millimeter, tenth of a millimeter, millimeter, centimeter, decimeter, meter).

The third test is the operation test 24500, shown in Figure 38 and Table 5.

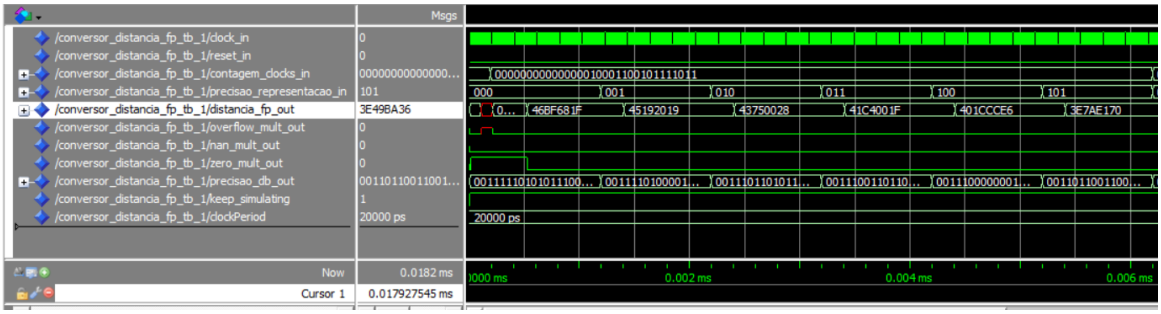

Figure 38: Test #3 - 24500 - variation of the 6 representation types.

FP Hexadecimal Representation	Converter Output	Corresponding Decimal (m)	Expected Result
46BF681F	24500.06	24500.06 x 10E-5	24500.00
45192019	2450.006	2450.006 x 10E-4	2450.000
43750028	245.0006	245.0006 x 10E-3	245.0000
41C4001F	24.50006	24.50006 x 10E-2	24.50000
401CCCE6	2.450006	2.450006 x 10E-1	2.450000
3E7AE170	0.245001	0.245001	0.245000

Table 5: Converter test - measurement 24500 with 6 representations (hundredth of a millimeter, tenth of a millimeter, millimeter, centimeter, decimeter, meter).

- **Integration Tests of the HC-SR04 Interface and Conversion to Floating-Point - Single Precision**;

  After validating the conversion module, integration tests were conducted between the HC-SR04 ultrasonic sensor interface and the floating-point converter. The figures 39, 40, and 41 show the integration tests, along with Table 6.

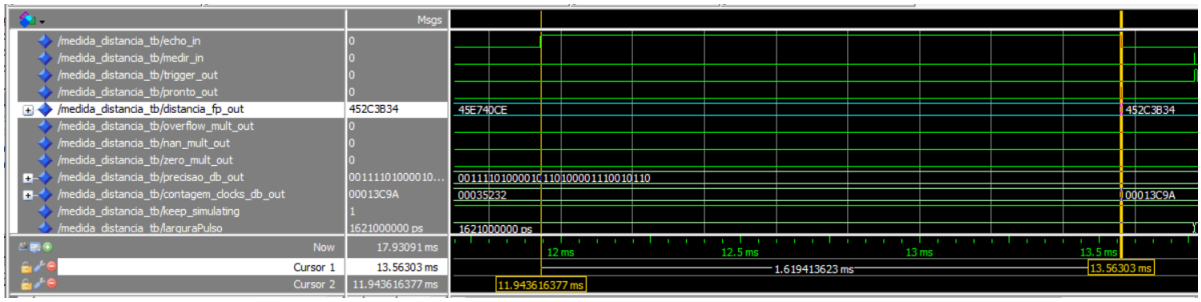

Figure 39: Integration test #1 - measurement of 2756 tenths of millimeters.

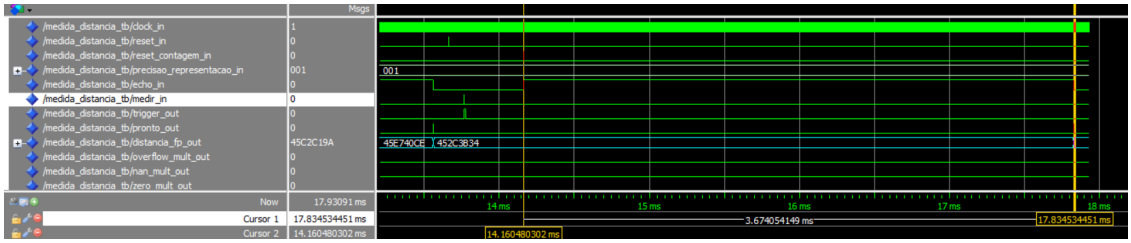

Figure 40: Integration test #2 - measurement of 62.33 centimeters.

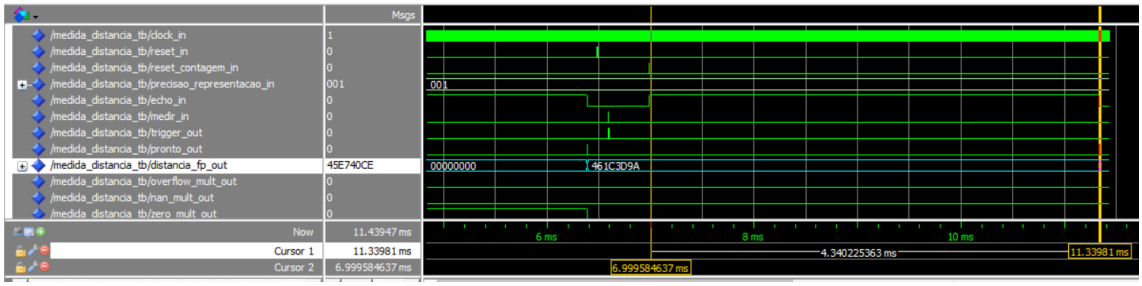

Figure 41: Integration test #3 - measurement of 74.00 centimeters.

Image Reference	FP Hexadecimal Representation	Corresponding Decimal	Expected Result (dec mm)
Figure 39	452C3B34	2755.7002	2755.00
Figure 40	45C2C19A	6232.20019	6233.00
Figure 41	45E740CE	7400.100586	7400.00

Table 6: Converter test - 3 measurements in tenths of millimeters.

These integration tests confirm the correct functioning of the components.

c) **Test of the Environment Decoding Module - Adaptation of the Bayesian Filter**

The final module tested is the environment decoding module. This module consists of two RAM units, which hold the single-precision floating-point values of σ_P and σ_M. Figure 7 and Table 7 confirm the correct operation of the addressing module used for determining the input parameters of the Bayesian filter.

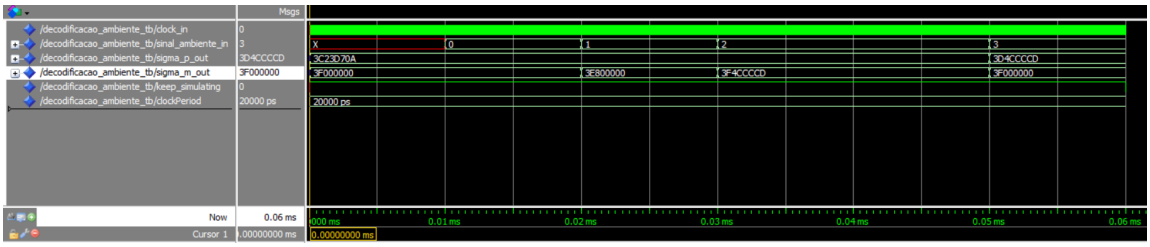

Figure 42: Test of the environment decoding module.

3.3 Construction of the Main Circuit

The integration phase involves the assembly of the three main parts of the intelligent sonar system presented earlier: the Bayesian filter calculation module, the ultrasonic

Measured Values		Expected Values		Decimal Values	
σ_P	σ_M	σ_P	σ_M	σ_P	σ_M
3C23D70A	3F000000	3C23D70A	3F000000	0.01	0.50
3C23D70A	3E800000	3C23D70A	3E800000	0.01	0.25
3C23D70A	3F4CCCCD	3C23D70A	3F4CCCCD	0.01	0.80
3D4CCCCD	3F000000	3D4CCCCD	3F000000	0.05	0.50
3DCCCCCD	3F000000	3DCCCCCD	3F000000	0.10	0.50

Table 7: Obtained and expected values for decoding the RAM memories for σ_P and σ_M.

sensor signal conversion module, and the adaptive control module (responsible for selecting the circuit parameters). Since all parts have been properly tested and showed correct results, the next step is to validate the operation of the entire circuit.

The structure of the intelligent sonar system can be seen in the following figure:

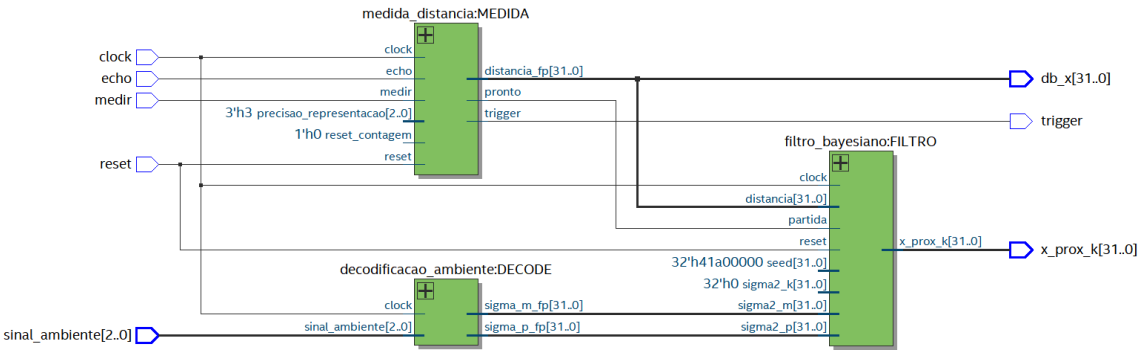

Figure 43: RTL Viewer of the complete circuit.

3.3.1 Tests and Simulations

Once the circuit was assembled, a testbench was created to verify the system's operation. The test case consists of acquiring 50 distance samples, with the initial parameters set as follows: $\sigma_{k_0} = 0$, $\sigma_p = 0.01$, $\sigma_m = 0.25$, $seed = 20$, and $distancias = 27$ and 27.8 cm. The results obtained were as follows:

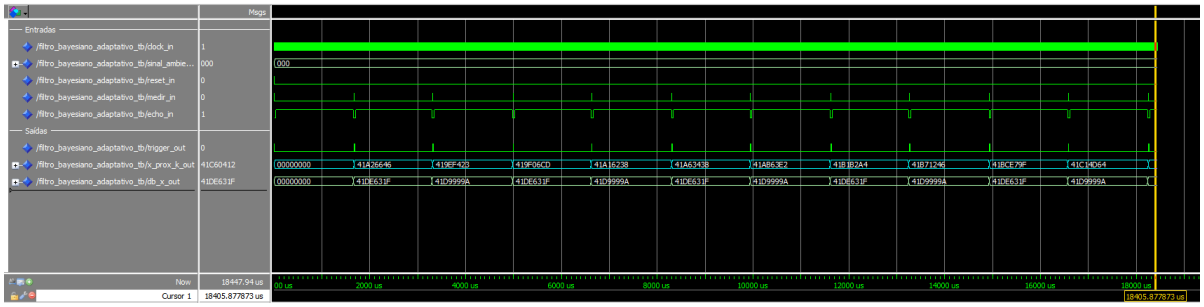

Figure 44: Beginning of the simulation.

At the start of the simulation, the estimates generated by the Bayesian filter are close to the provided seed and begin to converge towards the measured distances. The filter output, identified as *x_prox_k_out*, begins with a value of 20.29 cm and, after 10 iterations, estimates a distance of 25.15 cm. It is noted that the closer the seed value is to the actual measured value, the closer the estimates will be to the obtained measurements. The goal of these tests is to verify the correct operation of the circuit. For this purpose, an arbitrary value was assigned to the seed. However, with further refinement in the upcoming weeks, the results are expected to improve.

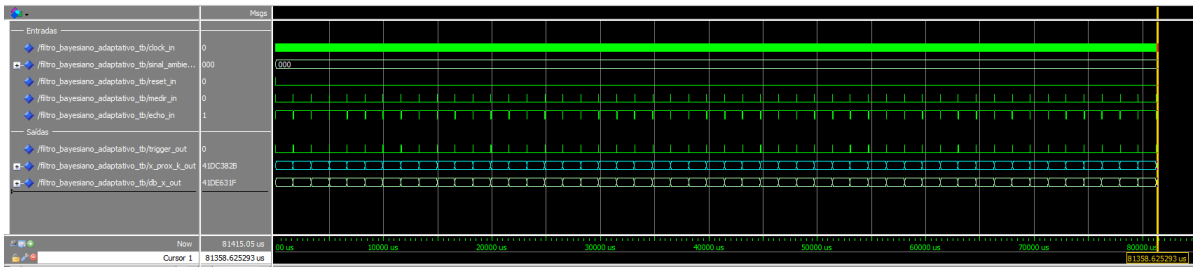

Figure 45: Complete simulation with 50 acquisitions.

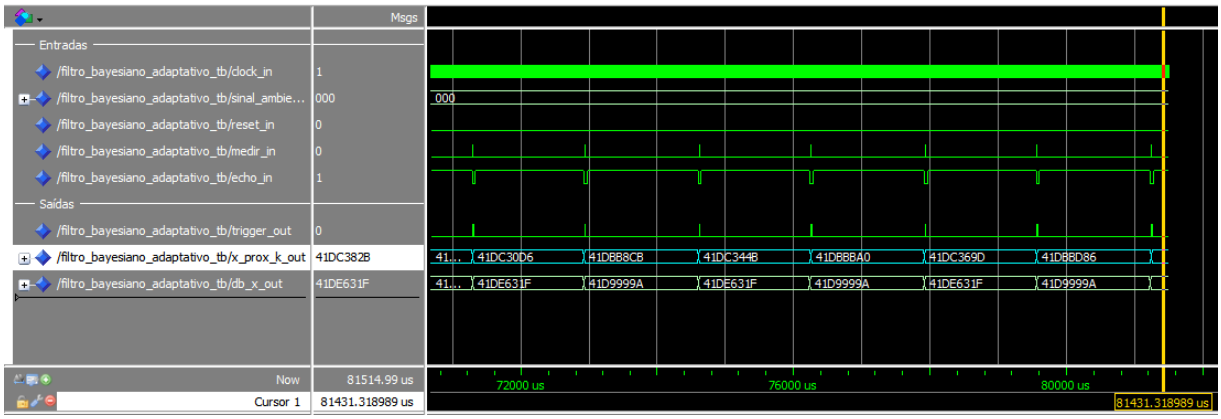

Figure 46: Zoom of the end of the complete simulation with 50 acquisitions.

At the end of the 50 iterations, the estimates show little variance and more stably

represent the distance received as input by the circuit. The final value obtained was 27.53 cm.

3.4 Integration Tests

The preparation of the project for synthesis on the FPGA board and the results of the tests conducted are presented in the following sections.

3.4.1 Validation Test Plan for the Bayesian Filtering Circuit

The test plan consists of measurements taken by the ultrasonic sensor through the FPGA board. The raw measurement data is processed in two ways. The first method is simple serial transmission of the data to a Python interface, where the data is stored, and filtering is performed in software. The software-based estimate is used for comparison with the hardware filtering results. The second method involves processing the raw data directly in FPGA. The results of the filtering are converted to decimal representation and also transmitted via serial communication (the serial communication utilizes the infrastructure provided by the course, Broker MQTT with a fixed IP, where the Python script accesses the transmission and reception topics to acquire the transmitted data).

At the current stage of the project, the tests are partially static, meaning that some tests are conducted using measurements of static objects, while others are conducted with objects that have some degree of freedom to move. Real-world test protocols have not yet been developed. However, since raw measurement data from the ultrasonic sensor is accessible, the software filtering results are used as a benchmark for FPGA filtering, which does not hinder the validation of the results that will be observed in the test phase.

Note: All FPGA implementation tests will be conducted using the laboratory kit provided to the team. This kit includes all the components of the homelab, as well as an FPGA board and components for integrating the FPGA board with the computer.

3.4.2 Development of New Components

During the course, the need arose to create a new conversion component for the output of the Bayesian filter calculation. The conversion required is from single-precision floating-point to BCD encoding.

The construction of the new module is divided into two main blocks: a control unit and a data flow. The data flow consists of a floating-point to integer converter (built using Intel Quartus software libraries), a 4-digit BCD counter (reused from previous projects), a 32-bit decrement counter, a delay counter, and a register to store the BCD output. The data flow of the module is shown in Figure 47.

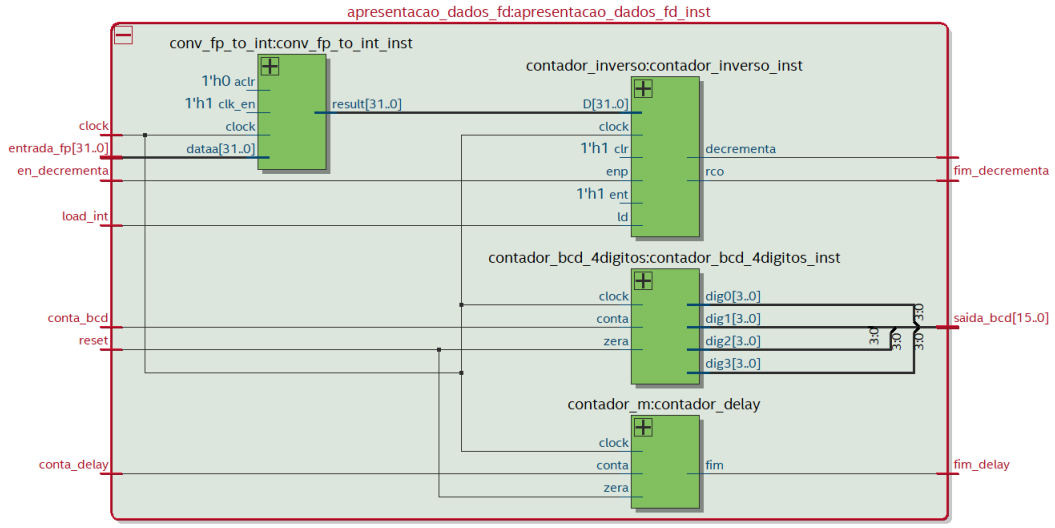

Figure 47: RTL Viewer of the data flow for the floating-point to BCD conversion module.

The control unit, shown in Figure 48, consists of six states: *Start*, which waits for a signal to begin the conversion; *Delay*, which adds time to ensure that the floating-point to 32-bit integer conversion is completed; *Register*, which stores the filtering estimate as an integer; *Decrement*, which performs the integer-to-floating-point conversion iteratively; *E_register_BCD*, which stores the filtering estimate in 16-bit BCD encoding; and *Ready*, which signals the end of the conversion operation.

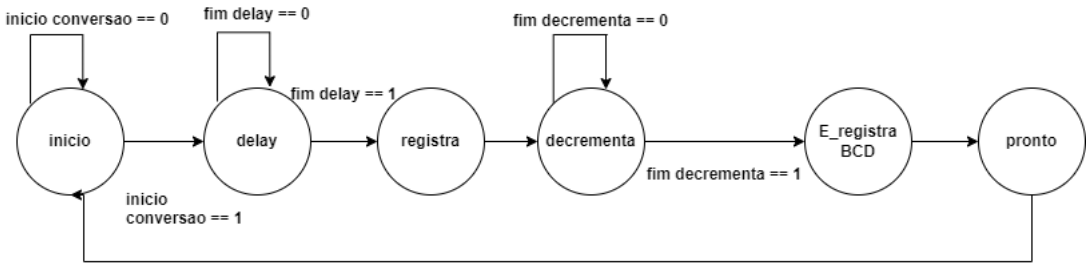

Figure 48: State diagram of the floating-point to BCD conversion module.

In Figure 49, we can see the integration between the data flow and the control unit described above.

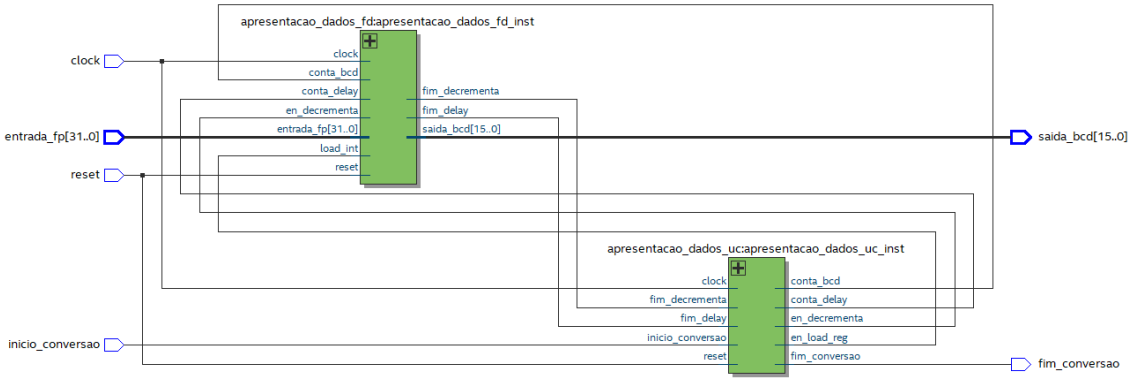

Figure 49: Block diagram of the floating-point to BCD conversion module.

Subsequent tests were conducted on the conversion module to validate the circuit's operation. In Figure 50, we show the conversion test. The input is a 32-bit floating-point value, 01000101001011001101000000000000, representing the number 2765.0. The output is 0010011101100101, which is the BCD-encoded value, confirming the module's correct operation.

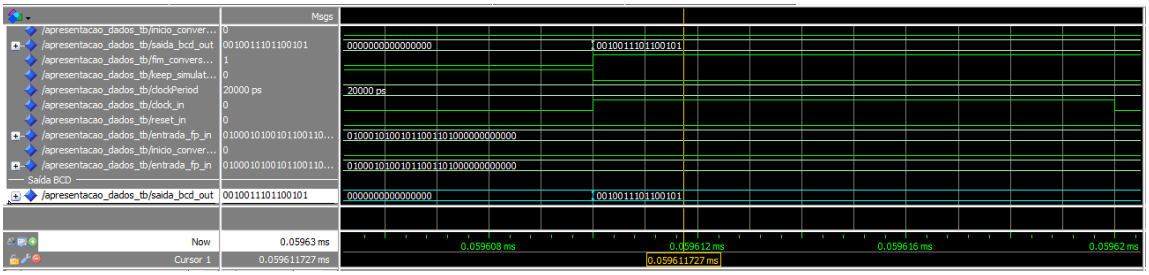

Figure 50: Floating-point to BCD conversion test.

3.4.3 Integration of New Components into the Main Circuit

Following the construction and testing of all the previously described modules, the final circuit diagram is presented in Figure 51. The new main circuit consists of a control unit and a data flow. The data flow of the main circuit, shown in Figure 52, consists of the components designed during the planning phase, integrated with the floating-point conversion module and a serial transmission component.

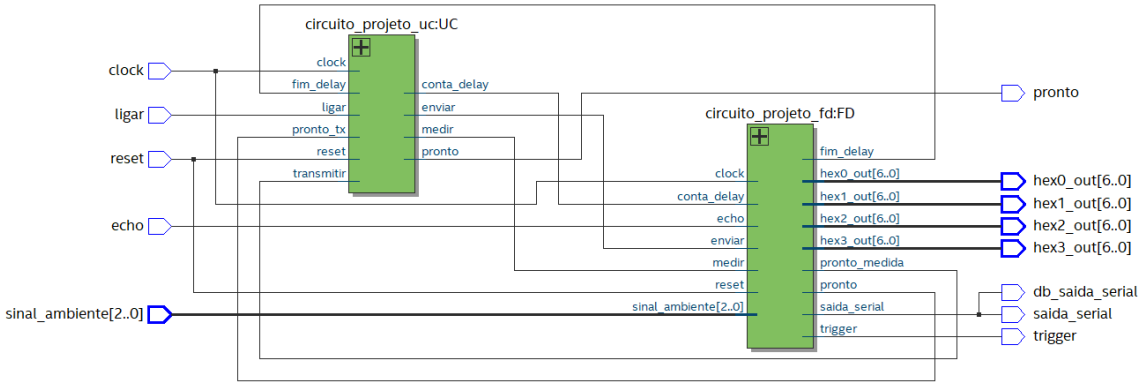

Figure 51: Main circuit diagram.

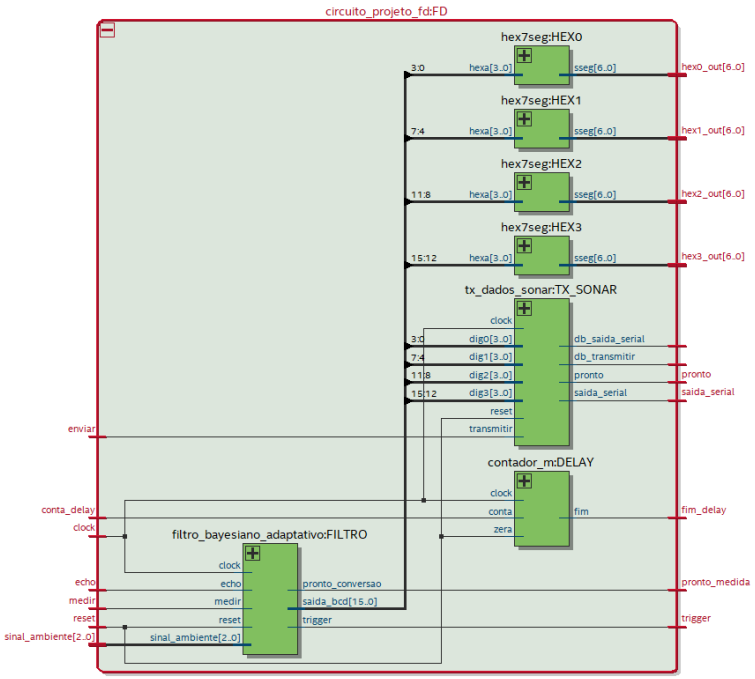

Figure 52: Main circuit data flow diagram.

The control unit, shown in Figure 53, consists of seven states: *Off*, *Delay*, *Measure*, *Initial*, *Transmission*, *Wait*, and *End*.

The *Off* state checks whether the circuit is active and ready to measure. The *Delay* state ensures that two consecutive measurements do not overlap. The *Measure* state starts the distance measurement, and the four subsequent states perform the same functions as in Week 1's circuit: transmitting the data in the *d3d2d1.d0* format and waiting for new data.

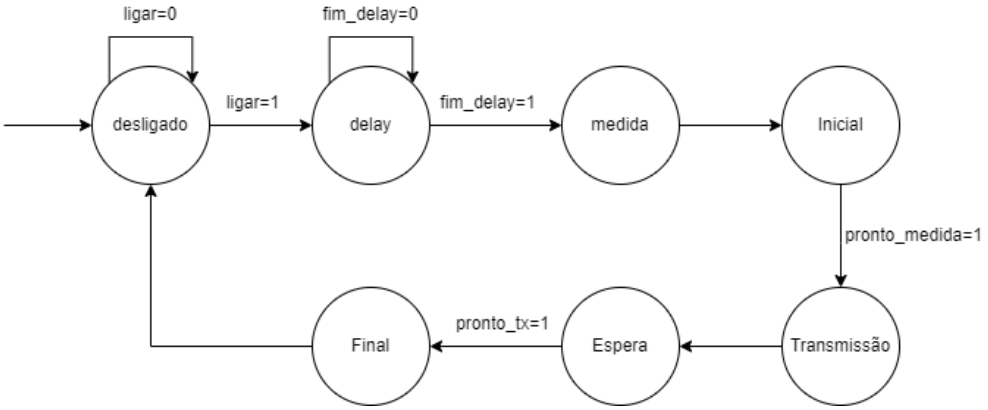

Figure 53: State diagram of the main circuit control unit.

After integration, tests were conducted to validate the correct operation of the Bayesian filtering circuit. The input consists of measurements from the HC-SR04 sensor, and the output includes both the raw, unfiltered distance measurement and the filtered estimate. Figure 54 shows the integration test results. Two tests are shown, with *sinal_ambiente* consistently at 000 (with 5 decimal places). The result is satisfactory, as the input pulse signal (representing the HC-SR04 sensor echo) is correctly converted to BCD output.

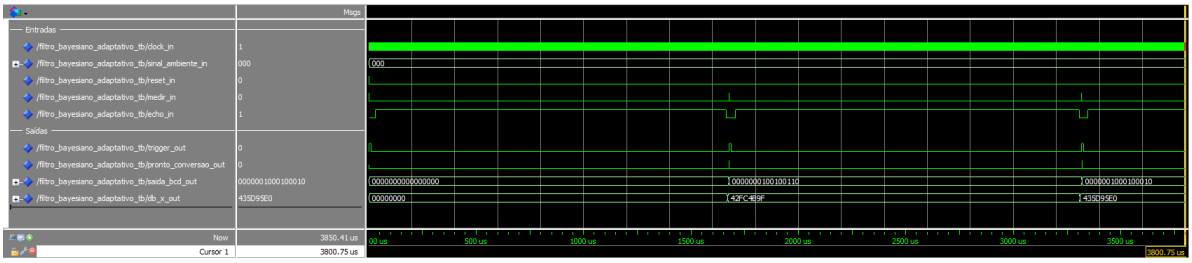

Figure 54: Main circuit integration validation test.

3.4.4 Results and Observations from Laboratory Tests

It was possible to successfully implement a manually adaptive Bayesian filter in hardware. Figures 55 and 56 show comparative graphs of distances measured by the ultrasonic sensor, both without filtering (in *red*) and with the filter applied (in ***black***). The data acquisition was done through serial transmission of estimates, using the same standard and tools from previous weeks.

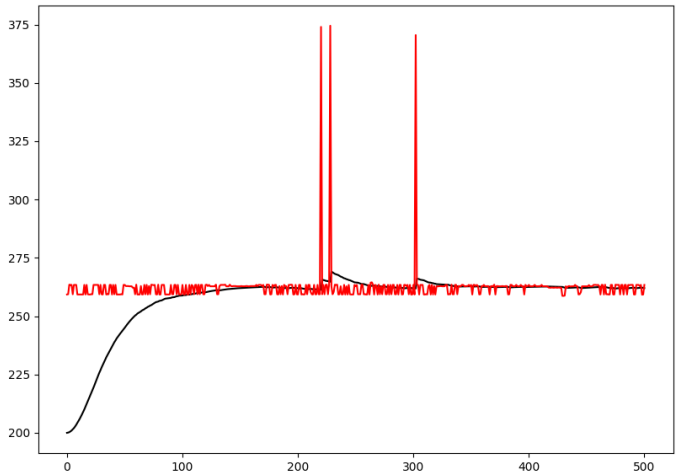

Figure 55: Test with Bayesian adaptive filter implemented in **software**.

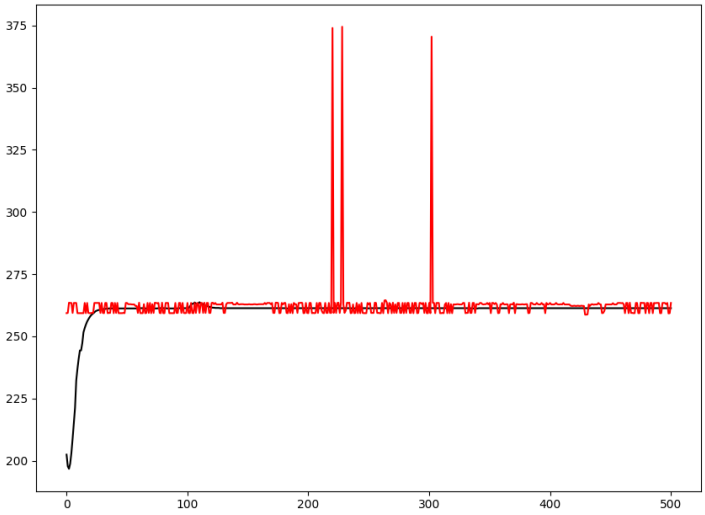

Figure 56: Test with Bayesian adaptive filter implemented in **hardware**.

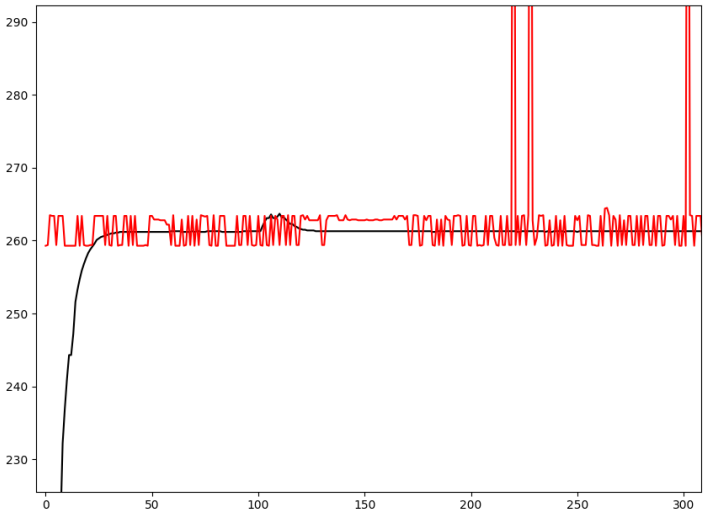

Figure 57: Zoom of the test with Bayesian adaptive filter implemented in hardware.

The variation in the output values of the filter, observed in the two figures, is due to the *seed* provided to the circuit (in this case, 20 cm). As the circuit receives new distance measurements from the ultrasonic sensor, its estimates begin to converge towards the expected value.

It is noteworthy that the results are very satisfactory, as the estimates provided by the Bayesian filter significantly reduced the variations previously present in unfiltered acquisitions. The next step is to set up a physical environment for testing and observe the behavior of the smart sonar during movement.

In comparison with the results obtained in software, the main difference observed lies in the convergence time of the filter's estimates. The hardware implementation was able to perform the task considerably faster. Further evaluation of different behaviors is still required to draw a clear and accurate conclusion about both the filter and the circuit that implements it.

NOTE:** *As it was not possible to acquire the filter estimates with the same distances obtained from the circuit without the filter, due to physical and technical constraints, the values obtained in hardware are not the direct application of the filter to the red data, but rather a new distance acquisition under* ***the same conditions *of distance, temperature, humidity, and speed.*

***** *External interferences, such as changes in temperature, position, or humidity, were not considered in the simulation environments.*

3.5 Modeling of New Components

To adapt the automated determination of the parameters σ_P and σ_M, used in Bayesian filtering, to hardware, three components were developed to be integrated into the main circuit. The first component is a modulus subtractor, the second component implements the logic for parameter addressing in hardware (based on the software test presented in the previous section), and the last module is a sequential register bank that stores and updates the values of the last N measurements taken.

3.5.1 Floating-Point Modulus Subtractor

To perform comparison operations between sequential distance measurements, a subtractor that operates in modulus was developed. Modulus operations were chosen because the construction of components performing floating-point operations is of the *unsigned* type. Therefore, adopting a new component that generates only positive values avoids conflicts with the base of the main circuit developed in the previous stages. To execute modulus subtraction, a floating-point comparator, two 32-bit 2x1 multiplexers, and a floating-point subtractor were used.

This new component is combinational, meaning its output depends solely on the input signals. The circuit begins with the comparison of two floating-point numbers. At this stage, the signal is generated to determine the relationship (greater than, less than, or equal) between the two data points, which in turn generates the selection signal for the two multiplexers. Each of the multiplexers receives the same data as the comparator, but necessarily in the same order. However, the selection signals are complementary, meaning if the selection signal of multiplexer A is *1*, the selection signal of multiplexer B must be *0*, ensuring that the first value entering the subtractor is always greater than the second. The final output of this component produces the modulus subtraction result.

Figure 58 shows the RTL diagram of the modulus subtraction component.

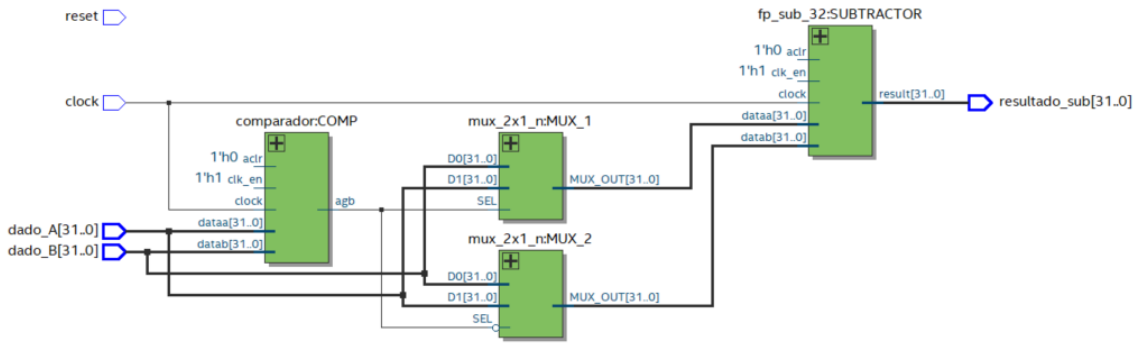

Figure 58: Block diagram of the modulus subtraction module.

Figure 59 shows the test of the floating-point modulus subtractor. In the first test, subtraction occurred within a range from 0 to 0.01 ms, with input data $43A00000$ (320 decimal) and $440AC000$ (555 decimal). The result was $436B0000$ (235 decimal). In the second test, between 0.02 ms and 0.03 ms, the same floating-point values were input but in reversed order, resulting in the same value $436B0000$ (235 decimal).

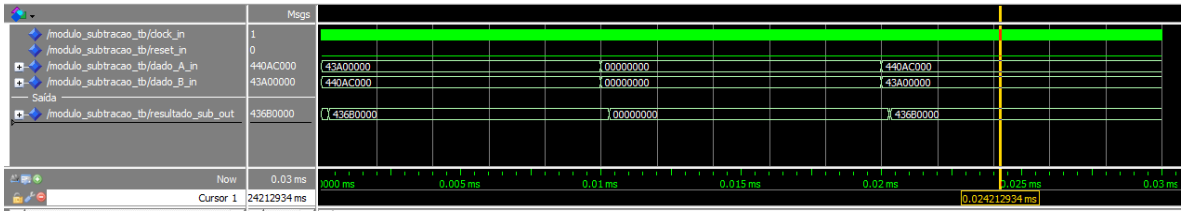

Figure 59: Timing chart of the modulus subtraction test.

As observed in the figure above, the test results are consistent with the expectations.

3.5.2 Setting Module Parameters

The parameter addressing module implements the selection logic that determines the parameters σ_P and σ_M. The implementation follows the logic outlined in Table 10, where, after determining a value from four data acquisitions, some calculations are performed, resulting in a value that could represent acceleration if each operation step involves division by time. This final value is compared to a set of tabulated values that define specific intervals. Each interval is associated with addressing a pair of parameters.

In the implementation of the parameter addressing module, a simplification was adopted to avoid performing division operations by time, considering the assumption that sequential acquisitions occur at fixed time intervals. This hypothesis has not been fully validated yet, but the preliminary implementation tests will follow this reasoning. In

the tests shown in Figures 80 and 81, fixed intervals of 10 ms were used. If significant deviations are observed, modifications will be made to include the division by acquisition time between measurements.

The parameter addressing module consists of three absolute value floating-point subtractors. Two of these components are used to perform subtraction: the first receives data for the current and last measurements, while the second receives data for the penultimate and antepenultimate measurements. The output of each of these blocks generates the result of the absolute value subtraction. Finally, the last subtraction block is fed by the output of the previous blocks, resulting in the final subtraction measure. This subtraction measure is then used in the interval table that determines the parameters. To determine the final result of the subtractions in relation to the table, comparators are used. Since a table with four intervals was created, four comparators are used. However, if an adaptive module with more intervals is desired, additional counters can be added. Each of the four counters is fed by two signals: the first is the final subtraction result, and the second is a fixed value. The goal is to determine in how many comparisons the final result is greater compared to the increasing fixed values determined by the arbitrarily constructed table. The output of the four comparators is used as input to a final component that decodes the information (n x m). In the current implementation, four comparators are used, so n equals 4. The m-bit output of the decoder is related to the number of bits required for addressing. Although four parameter pairs are used, which could be addressed by 2 bits, 3 bits were used to facilitate the inclusion of more intervals if needed. Finally, through this decoding, the parameters used in the filtering process are addressed.

Figure 60 shows the RTL diagram of the parameter addressing component.

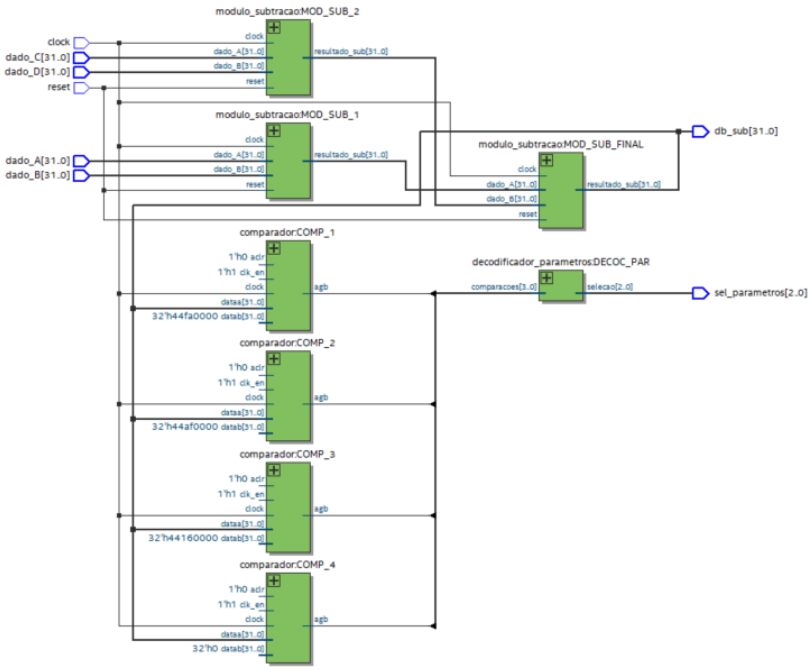

Figure 60: Block diagram of the parameter addressing module.

Table 8 includes the addressing column corresponding to the interval resulting from the subtraction operations.

Intervalos (Acel em unidade U)	σ_P	σ_M	**Endereçamento**
1400 < Acel < 2000	0.3	0.25	000
600 < Acel < 1400	0.01	0.1	001
0 < Acel < 6000	0.1	0.75	010
0 < Acel	0.05	0.5	011

Table 8: Relação de endereçamento dos parâmetros.

Figure 61 shows the test of the absolute value floating-point subtractor.

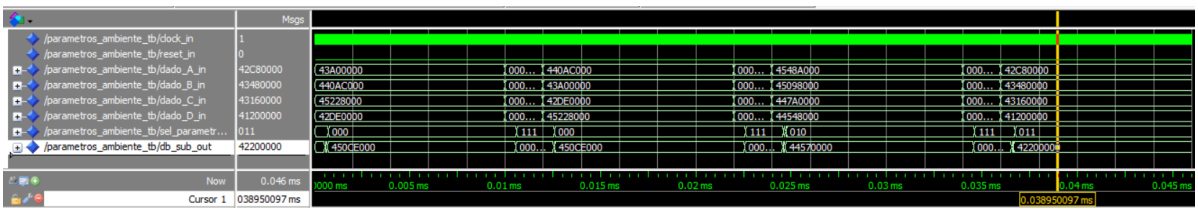

Figure 61: Timing diagram of the parameter decoding/addressing module test.

The test shown above involves four experiments. The four input data *data_A_in*, *data_B_in*, *data_C_in*, and *data_D_in* are floating-point measurement values. The output *db_sub_out* is a debugging signal resulting from the three subtraction operations, and the

signal *sel_parametros* is the result of the addressing performed by the module. Table 9 summarizes the results shown in the timing diagram.

Dado A	Dado B	Dado C	Dado D	Operação Decimal	DB_subtração	Endereçamento
43A00000 (320)	440AC000 (555)	45228000 (2600)	42DE0000 (111)	||320 - 555| - |2600 - 111|| = 2254	450CE000	000
440AC000 (555)	43A0000 (320)	42DE0000 (111)	45228000 (2600)	||555 - 320| - |111 - 2600|| = 2254	450CE000	000
4548A000 (3210)	45098000 (2200)	447A0000 (1000)	44548000 (850)	||3210 - 2200| - |1000 - 850|| = 860	44570000	010
42C80000 (100)	43480000 (200)	43160000 (150)	41200000 (10)	||100 - 200| - |150 - 10|| = 40	42200000	011

Table 9: Resultados de endereçamento de parâmetros - número entre parênteses em representação decimal.

As observed in the table above, the result indicates the correct operation of the new component.

3.5.3 Sequential Register Bank

The register bank is a sequential component divided into a control unit and data flow, as shown in Figure 62. The register bank is used to store the last N measured data points. In the context of the implementation, only four points are required, including the most recent data, and thus three historical data points. The register data feed the input of the parameter addressing component.

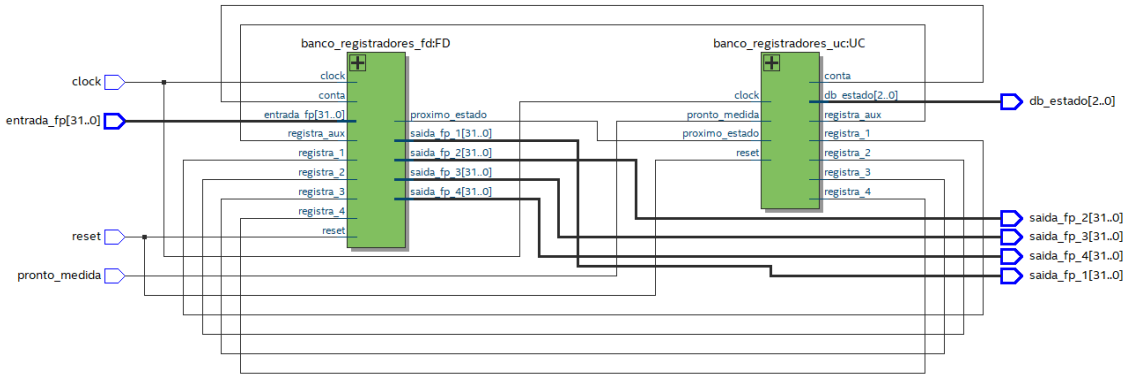

Figure 62: Block diagram of the main register bank circuit.

The data flow, shown in Figure 63, consists of five 32-bit registers. Four registers store the acquisition values, in floating-point format, for feeding the parameter addressing module. An auxiliary register is used to store the most recent acquisition data. Although not explicitly shown in the block diagram, the output of each register is connected in series to the input of the subsequent register in the order $Registrador_{aux}$ -> $Registrador_1$ ->

Registrador$_2$ -> *Registrador*$_3$ -> *Registrador*$_4$. This connection is made so that the data update happens sequentially.

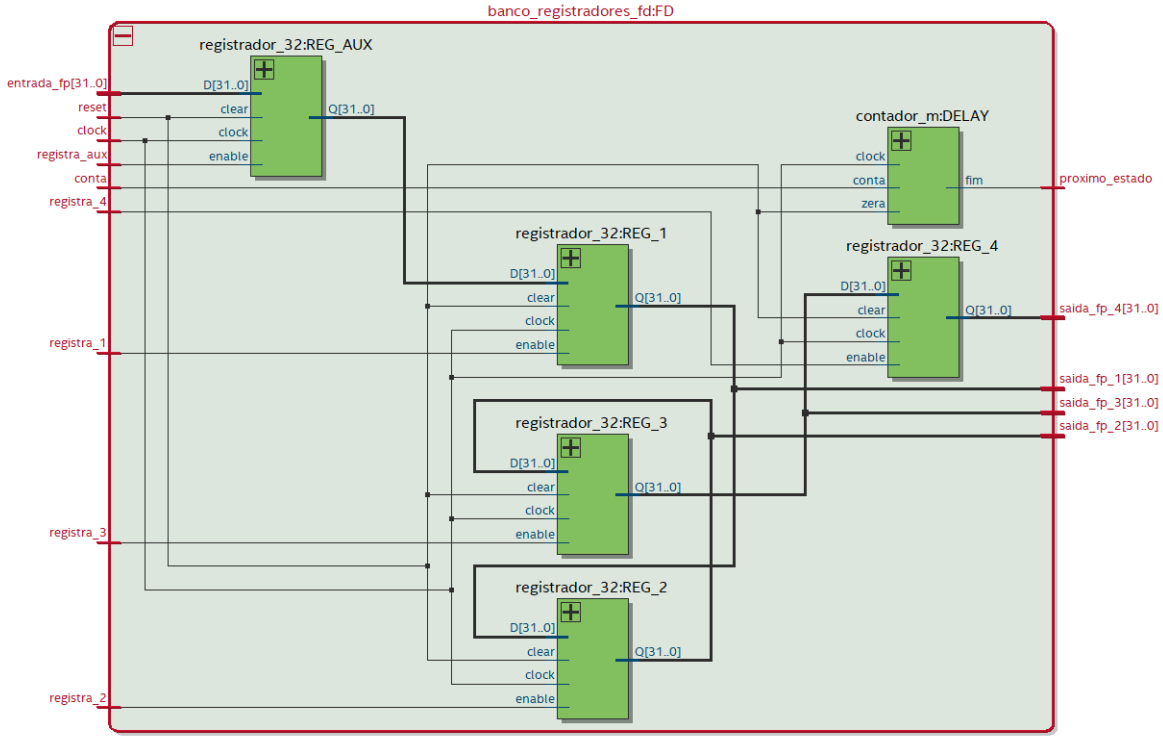

Figure 63: Block diagram of the data flow in the register bank.

The control unit, shown in Figure 64, consists of six states: *Initial*, REG_{aux}, REG_1, REG_2, REG_3, REG_4. Each of the register states activates the register signal for one of the registers, following a sequential logic. Initially, the most recent data is recorded in the auxiliary register, and then, in the order *Register_4* -> *Register_3* -> *Register_2* -> *Register_1*, the values of all the registers are updated.

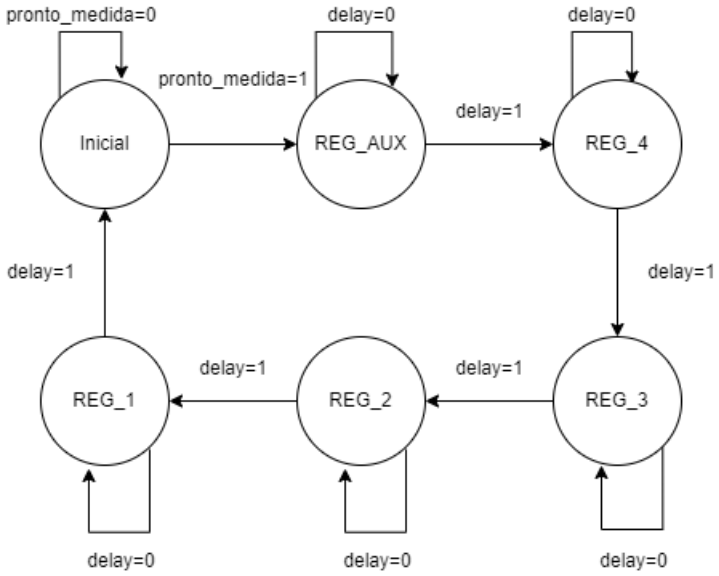

Figure 64: State diagram of the control unit of the register bank.

Figures 65 and 66 show the functional tests of the register bank.

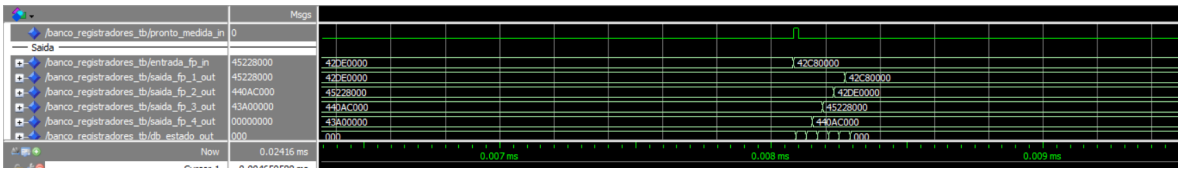

Figure 65: Timing diagram #1 - test of the parameter decoding/addressing module.

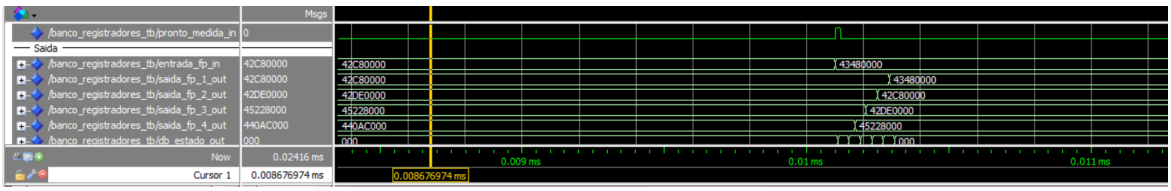

Figure 66: Timing diagram #2 - test of the parameter decoding/addressing module.

In Figure 65, it is possible to observe that the first register contains the value 42DE0000, the second contains 4522800, the third contains 44DAC000, and the fourth contains 43A00000. Upon receiving the pulse signal from *pronto_medida*, the four registers are updated, following the logic that the $N+1$ register stores the value of register N until it reaches the value of the input to the sequential register bank.

Similarly, in Figure 66, the operation is similar to what was previously described, but with a variation in the values stored in the registers, further corroborating the hypothesis of the correct operation of the new component.

4 Data Collection and Analysis of Bayesian Filter Estimates

4.1 Experiments for Determining Filter Parameters

To determine the filter parameters, initial experiments were carried out using software. Some approximation and deviation curves were constructed, with the presence of noise, simulating the behavior of a potential physical test. The tests used Python notebooks, which included an implementation of the Bayesian filter, with three main inputs: the data sequence (simulating the continuous data acquisition from the ultrasonic sensor), and the two parameters σ_P and σ_M. Some of the test curves are shown in Figures 67 to 73 below. In these images, the ordinate axis is in tenths of millimeters, and the abscissa axis corresponds to the count of the observed measurement.

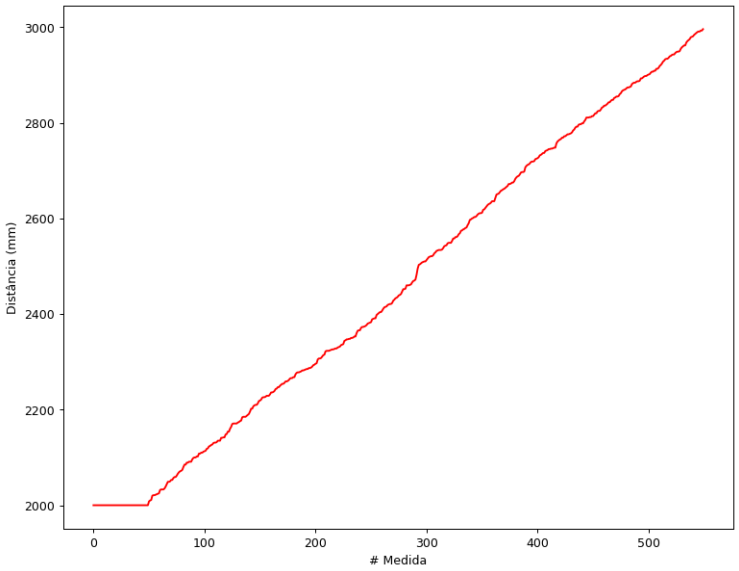

Figure 67: Simulated curve #1 - Deviation with stable plateau.

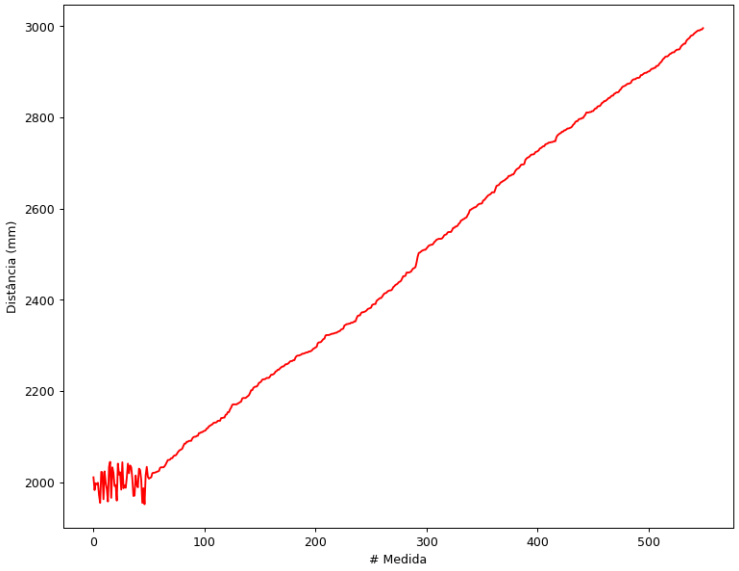

Figure 68: Simulated curve #2 - Deviation with noisy plateau.

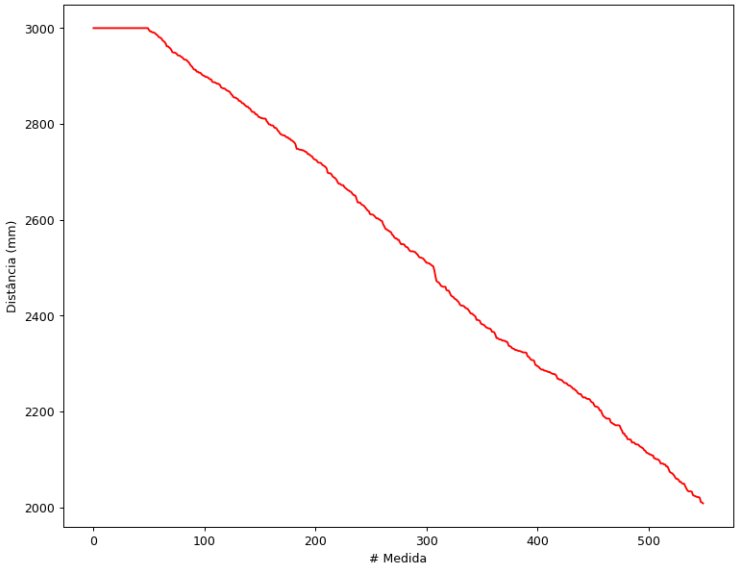

Figure 69: Simulated curve #3 - Approach with stable plateau.

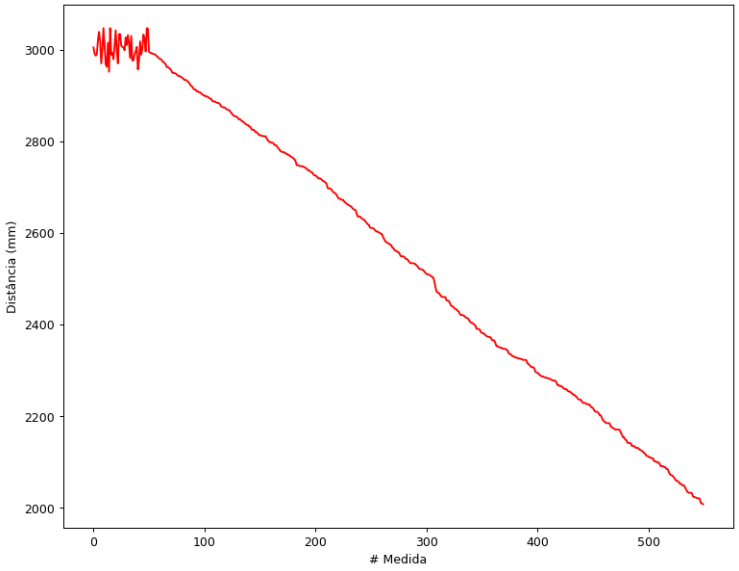

Figure 70: Simulated curve #4 - Approach with noisy plateau.

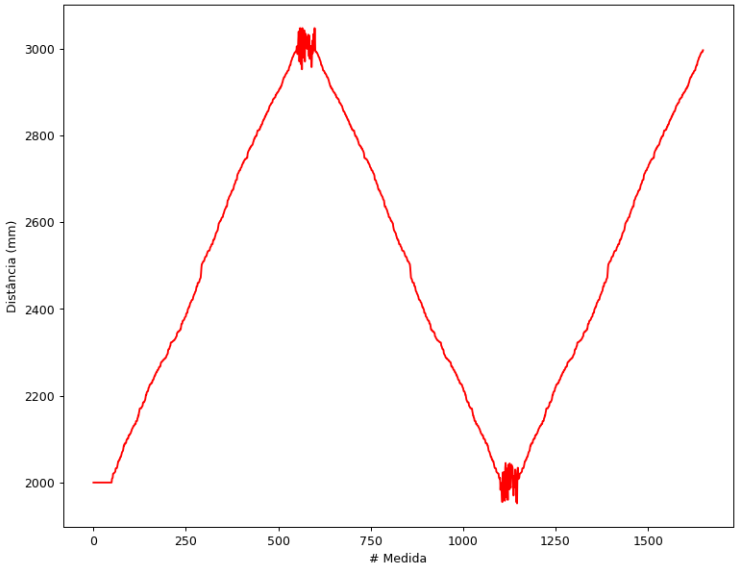

Figure 71: Simulated curve #5 - Approach and deviation with noise.

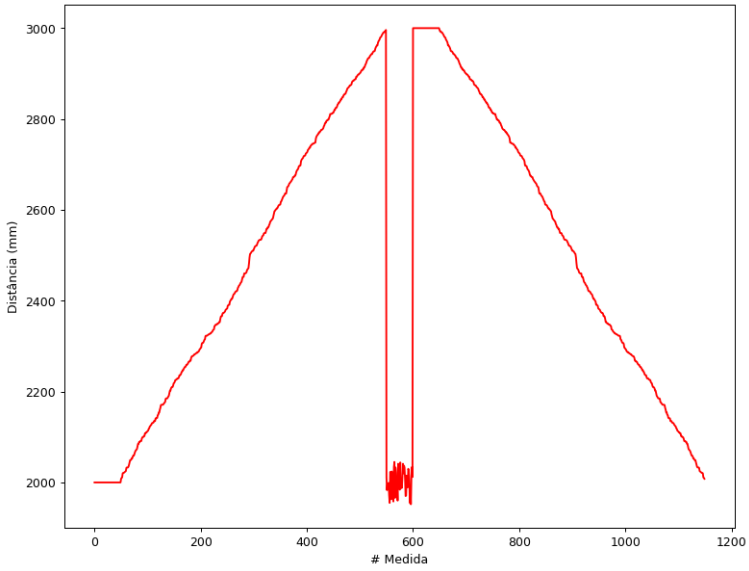

Figure 72: Simulated curve #6 - Abrupt approach and deviation with noise.

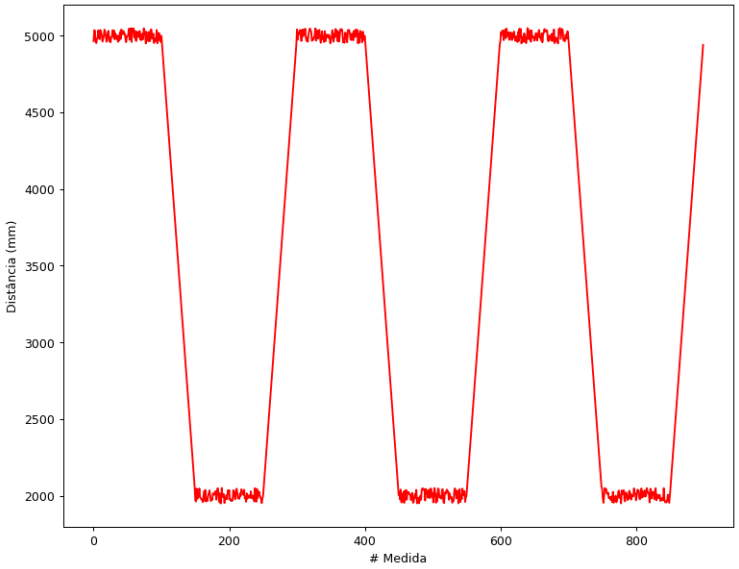

Figure 73: Simulated curve #7 - Periodic approach and deviation with noise.

Through these curves, numerous executions were performed to observe the effect of the parameters σ_P and σ_M. Depending on the choices, the estimated curve (i.e., the output of the Bayesian filter) shows greater or lesser error compared to the original data. Successful filtering is related to an estimated curve that exhibits a lower variance than the real curve and accommodates abrupt measurement variations, which is often linked to the intrinsic error of the ultrasonic sensor.

Next, in Figures 74 to 79, some curves with fixed parameter tests are shown. The tests shown here use parameters presented in [1]. In these figures, the curves in **red** are

associated with the raw data, while the curves in **black** are associated with the filtered result.

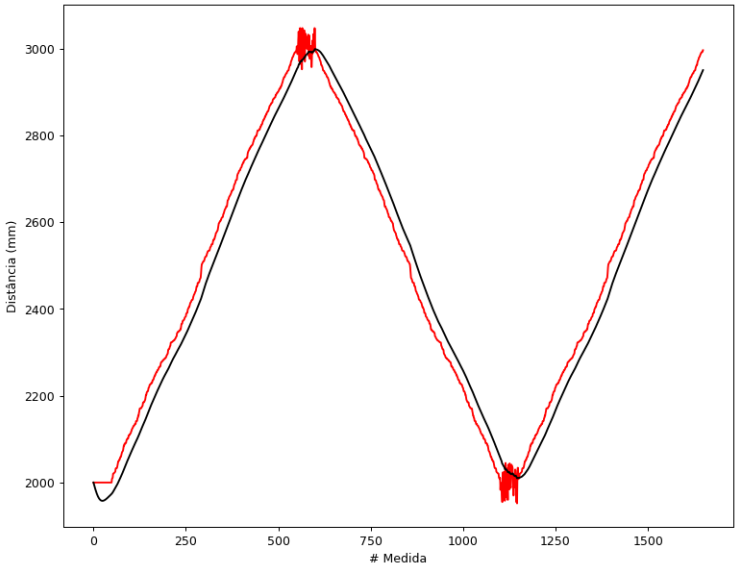

Figure 74: Fixed parameter test 1 - $\sigma_P = 0.01$ and $\sigma_M = 0.25$ - Approach and deviation with noise - Real variance: 98978, Filter variance: 98427.

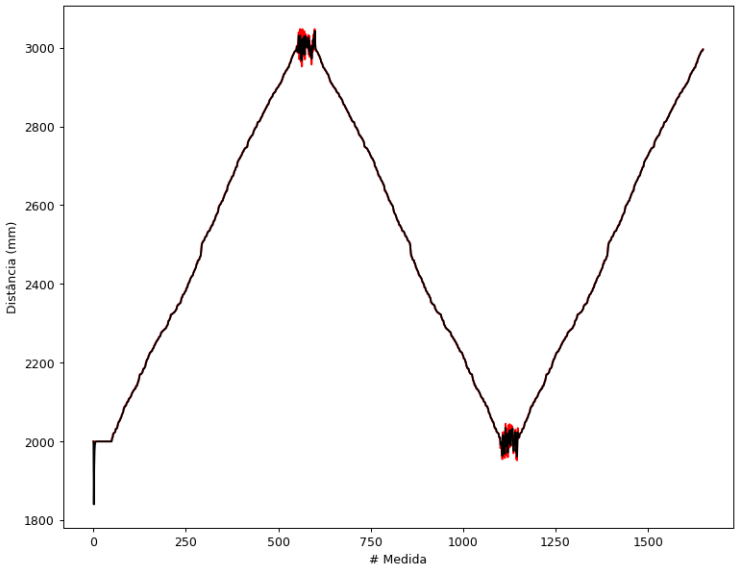

Figure 75: Fixed parameter test 2 - $\sigma_P = 0.30$ and $\sigma_M = 0.25$ - Approach and deviation with noise - Real variance: 98978, Filter variance: 99126.

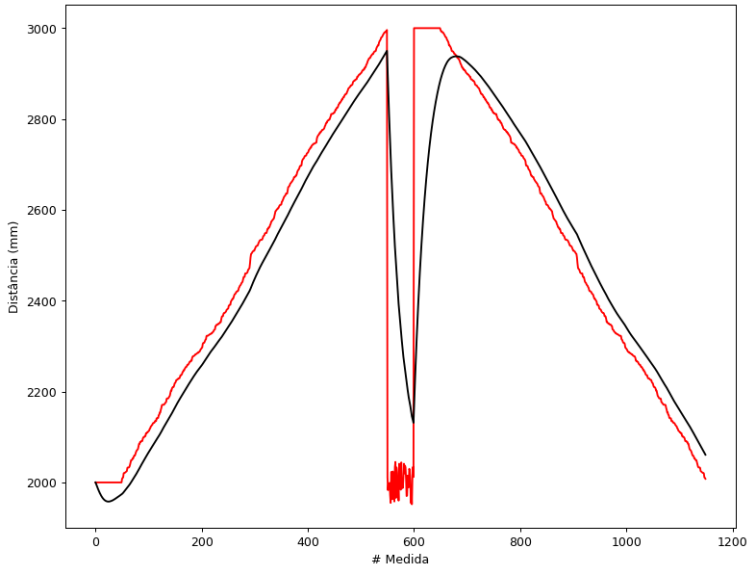

Figure 76: Fixed parameter test 3 - $\sigma_P = 0.01$ and $\sigma_M = 0.25$ - Abrupt approach and deviation with noise - Real variance: 105217, Filter variance: 88627.

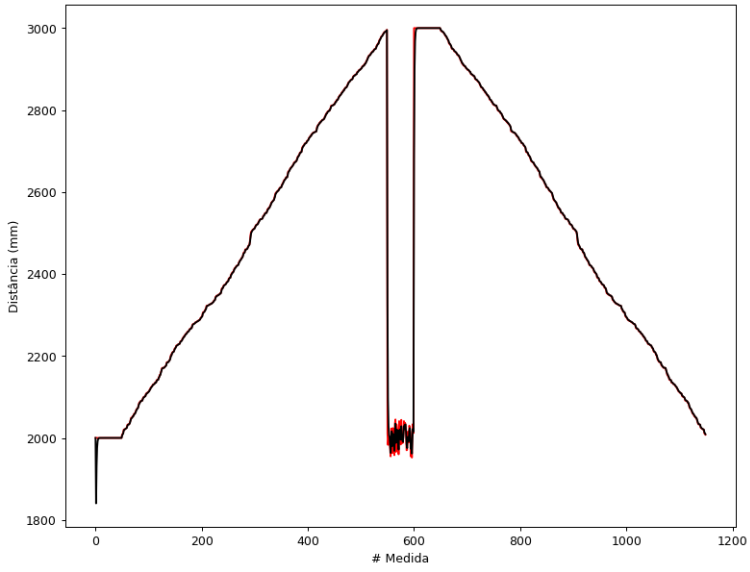

Figure 77: Fixed parameter test 4 - $\sigma_P = 0.30$ and $\sigma_M = 0.25$ - Abrupt approach and deviation with noise - Real variance: 105217, Filter variance: 104843.

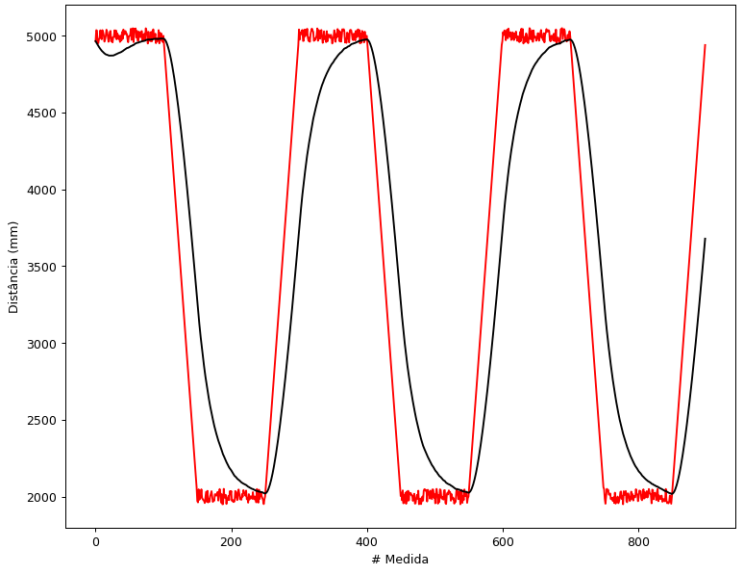

Figure 78: Fixed parameter test 5 - $\sigma_P = 0.01$ and $\sigma_M = 0.25$ - Periodic approach and deviation with noise - Real variance: 1749998, Filter variance: 1389014.

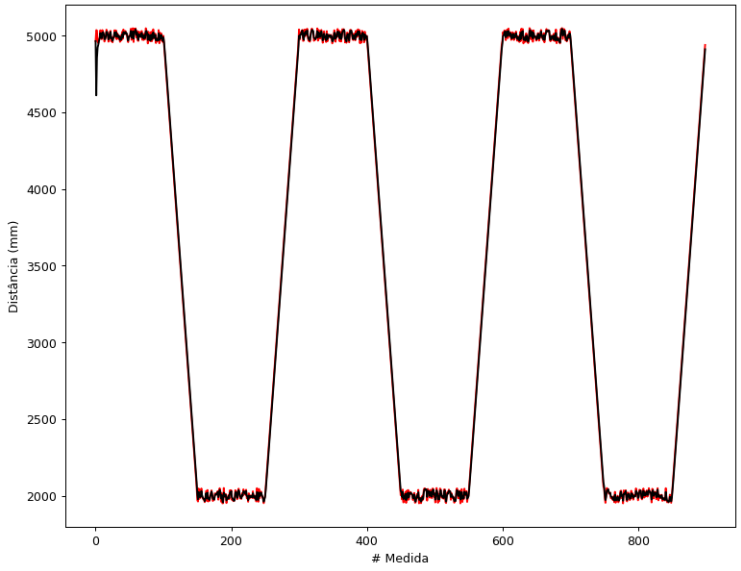

Figure 79: Fixed parameter test 6 - $\sigma_P = 0.30$ and $\sigma_M = 0.25$ - Periodic approach and deviation with noise - Real variance: 1749998, Filter variance: 1746975.

From Figures 75, 77, and 79, it is possible to observe a non-filtering behavior. Additionally, the filtering performed in Figure 75 increased the variance relative to the raw data curve.

After the filtering tests with fixed parameters, a logic was developed that automatically selects σ_P and σ_M through comparisons between variations in sequential measurement points. The idea is analogous to the determination of acceleration, since acceleration, in a discrete sampling method, uses the difference between two distance measurements

divided by the time between acquisitions to determine velocity, and the difference between velocities divided by the time between the acquisition of the first and last points used in determining the velocities to calculate, approximately, the acceleration.

For simulation purposes, four acceleration values were adopted, which determine four regions associated with the filtering parameters. The relationship used in the tests that will be shown below is shown in Table 10. The determination of the four parameter pairs was generated by permuting 21 parameters into pairs. For each parameter, the filtering test was conducted, and the generated curve was observed along with the variance comparison between the filtered data and the real data.

Intervalos (Acel em unidade U)	σ_P	σ_M
1400 < Acel < 2000	0,3	0,25
600 < Acel < 1400	0,01	0,1
0 < Acel < 600	0,1	0,75
0 < Acel	0,05	0,5

Table 10: Relação entre regiões e parâmetros da filtragem Bayesiana.

Figures 80 and 81, shown below, use the parameters from Table 10 for controlling the automation process.

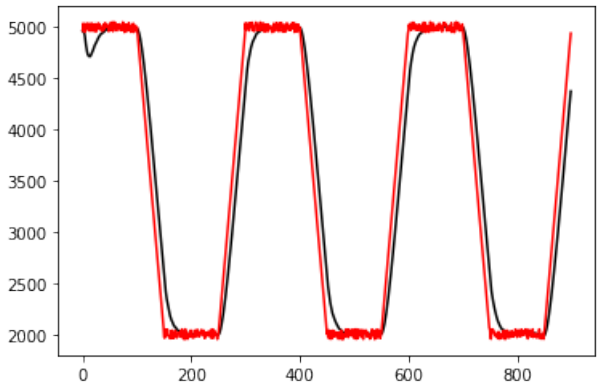

Figure 80: Adaptive parameter test 1 - Abrupt approach and deviation with noise - Real variance: 1749998, Filter variance: 1650891.

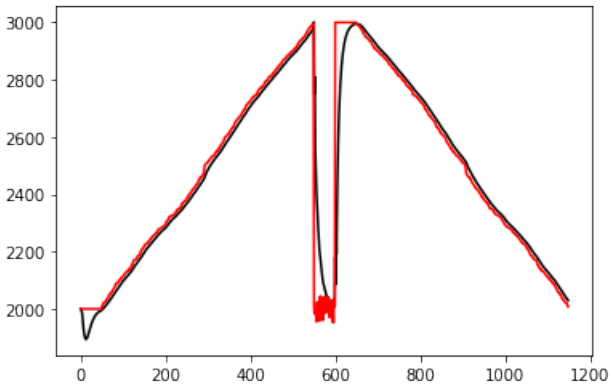

Figure 81: Adaptive parameter test 2 - Periodic approach and deviation with noise - Real variance: 105217, Filter variance: 99747.

As can be observed, the adoption of the test parameters successfully achieved a filtering curve close to the real data, with the characteristic of reducing the noise.

4.2 Circuit Adaptation for Operation

To prepare the circuit for presentation and for an operational scenario, some structural modifications were made. The first modification was the reduction of the distance acquisition interval to 1 ms, and the second modification involved adding a component that refreshes the circuit every 20 measurements. This last modification aims to address a systematic issue observed since Experiment 4: for an unknown reason, the main control unit freezes in one of its states without any external interference. Furthermore, this interruption occurs randomly and unpredictably (the number of measurements taken before the issue arises is variable).

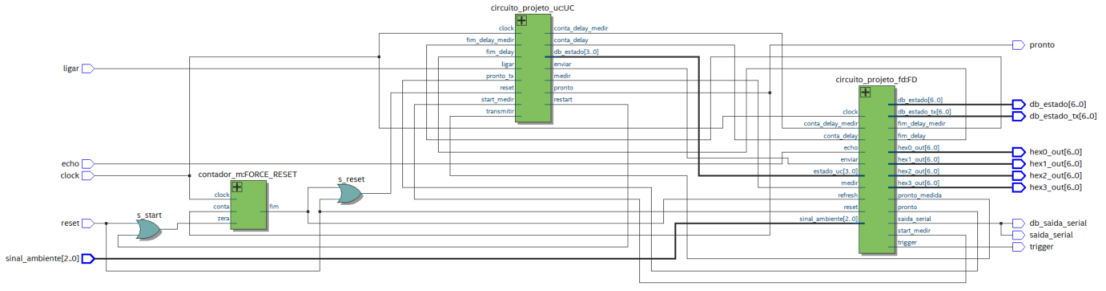

Figure 82: RTL Viewer of the complete circuit.

5 Tests - Validation of Final Results

The following presents the tests to be conducted on the bench during the experiment.

5.1 Comparison of Output Values from the Circuit with and without the Full Bayesian Filter

As observed in previous weeks, it is expected that the Bayesian filter will reduce noise from the ultrasonic sensor and environmental vibrations, as well as attenuate potential outliers measured by the sensor.

For this activity, three distinct test cases will be considered, involving changes in the filter parameters to evaluate the circuit's response. The acquisition of measurements will be done by altering the position of the circuit relative to the setup to simulate a moving condition. Additionally, the adaptive Bayesian filter's ability to attenuate intense vibrations and oscillations will also be tested.

*****_In the following graphs, the measurements taken by the circuit without the filter will be presented in red, and those taken with the application of the Bayesian filter will be presented in_ ***black***_._

*****_Note: The large variations at the beginning and end of the measurements are due to a reset that was applied at the start and end of each test case._

The test cases are as follows:

1. **Test Case #1:** $\sigma_p = 0.01$ **and** $\sigma_m = 0.25$

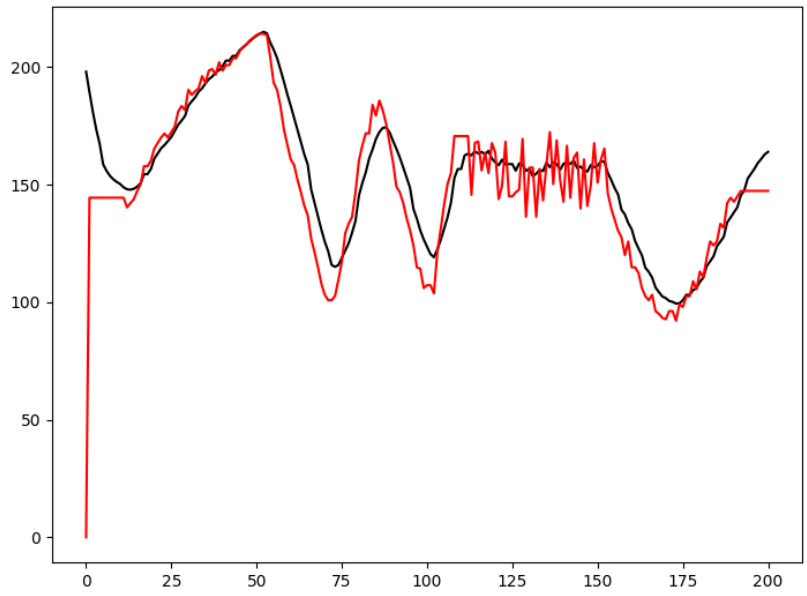

Figure 83: Comparison of outputs for the circuit with and without filter - Test Case #1.

The filter was able to significantly reduce the variations in distance measured by the circuit, especially between measurements 100 and 150. However, there is still a slight delay in the estimates during the more intense ascent and descent segments, which is due to the selected parameters.

In a parking sensor application, this configuration could be interesting, given that the vehicle speed is low in this condition (allowing a slight delay in the filter's response) and that precision is important to avoid collisions (attenuation of variations is essential).

2. **Test Case #2:** $\sigma_p = 0.05$ **and** $\sigma_m = 0.5$

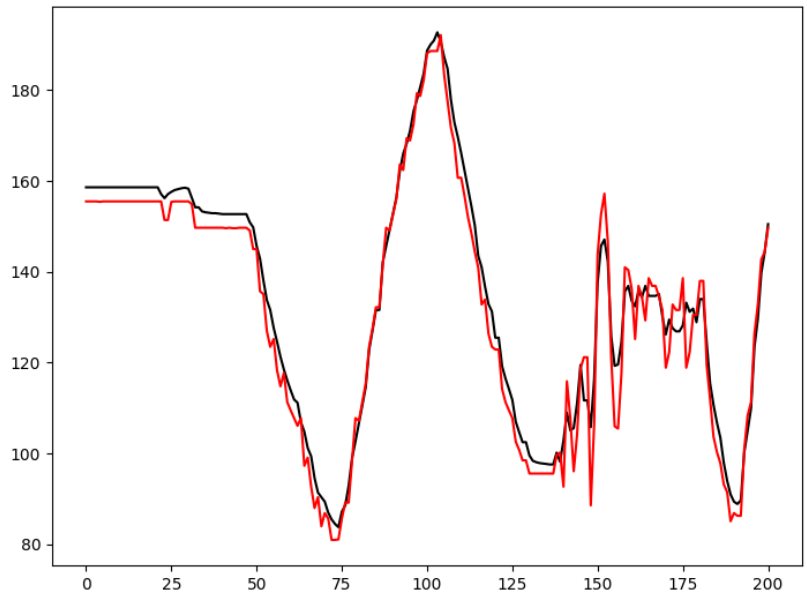

Figure 84: Comparison of outputs for the circuit with and without filter - Test Case #2.

3. **Test Case #3:** $\sigma_p = 0.1$ **and** $\sigma_m = 0.8$

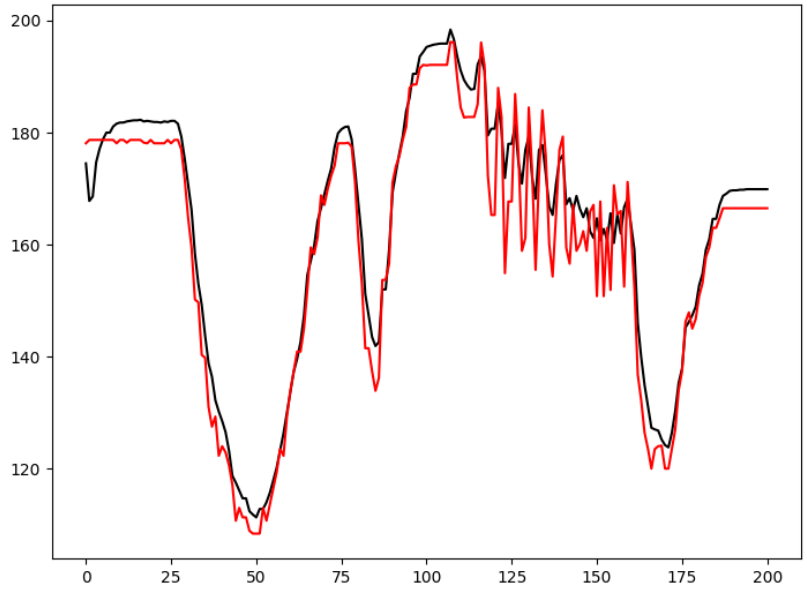

Figure 85: Comparison of outputs for the circuit with and without filter - Test Case #3.

In Test Cases #2 and #3, good results were also observed regarding the attenuation of the measured values, although it was not as intense as in the first case. However,

the delay previously observed was considerably reduced, showing a better adherence between the estimated values and those obtained by the circuit without the filter.

5.2 Verification and Validation of the Adaptive Module's Functioning in the Integration

The functionality tests conducted after the integration of the adaptive module implemented in the FPGA were inconsistent with the results observed in software tests, as shown in Figures 80 and 81. The selection of the Bayesian filtering parameters did not reduce the variance of the estimated curve compared to the raw data curve. Figure 86 shows the test conducted, where the raw data curve is presented in **red** and the filtered data curve is presented in **black**.

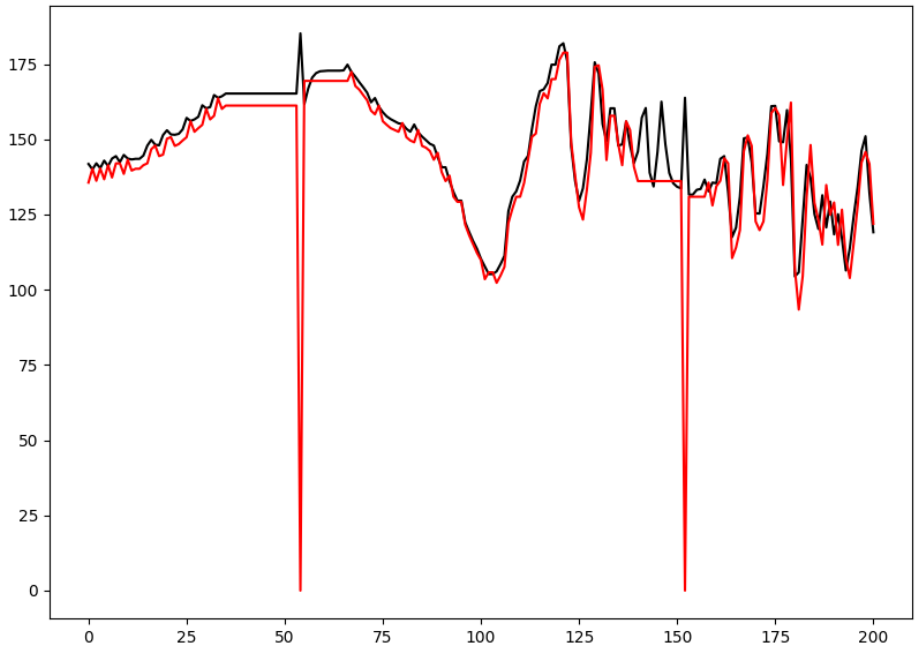

Figure 86: Test of the adaptive filtering in FPGA.

**Note: The sharp drops in red represent forced resets applied to the circuit to refresh the measurements of the unfiltered interface.*

The adaptation module was tested in the previous activity using the generation of synthetic data curves. There are two hypotheses for the problem that will be analyzed in

the future, before the project's completion. The first is a malfunction in the addressing logic of the module itself. This malfunction is related to an incorrect approach to solving the problem and/or the use of data incompatible with the tests conducted in the physical implementation, which is the second hypothesis. The tests during the planning phase did not involve the use of real data collection, as all the curves were generated based on some approximation and deviation logic. The tests and parameter choices were based on these simulated curves, which may be incompatible with the real data collection.

Therefore, all future tests will involve curves derived from real tests conducted through the FPGA implementation. Based on these curves, all tests will be re-executed, and the validation of the adaptive parameter addressing logic will be carried out. If the new simulations indicate correct operation, the next step will be testing on the FPGA. However, if an error is detected in the logic, modifications will be made to both the software and hardware implementations. From this approach, it is expected that the parameter addressing adaptation module will result in better filtering than experiments using fixed parameters throughout their execution.

5.3 Discussion on the Integration of Modules for the Final Circuit

The integration test of the modules in the final circuit concluded the iterative approach of validation tests in software and subsequent construction of the respective hardware modules. To date, the Bayesian filter is functioning exactly within the specifications, considering manual control of the environmental signals. The automated adaptation module is still undergoing review and verification.

6 Demonstration - Final Product Presentation

6.1 Revisiting the Context and Motivations of the Project

Distance sensors are commonly used for obstacle detection and distance measurement, providing data for various applications such as collision prevention, object detection, and terrain mapping. However, these sensors have a wide range of quality and accuracy, with prices ranging from USD 0.55 to over USD 100 per unit. The simple choice of lower-cost sensors, however, directly impacts the reliability of the measurements, as the noise levels in the calculated values can be significant [1]. In this context, this project aims to present a model of an adaptive Bayesian filter as a *proof of concept* for improving the reliability of measurements from low-cost sensors.

6.2 Proposed Technical Solution

In this project, floating-point (IEEE 754) calculation modules were developed to implement the Bayesian filter in hardware, a module for converting the sensor measurements to floating-point format, and a module for controlling the adaptation of the Bayesian filter. It is worth noting that the development of these calculation and signal conversion components, specifically the floating-point arithmetic components, was supported by the libraries available in the Intel Quartus development software - IP Catalog.

The system is expected to perform a distance measurement using an ultrasonic sensor, carry out all the Bayesian filtering calculations, and transmit the measured and estimated data to an interface that allows for the demonstration of the circuit's functionality. Finally, the adaptive control module enables the adjustment of the sensor parameters, altering its filtering characteristics to provide either more or less sensitivity.

6.3 Improvements and Future Work

The design of the project involved the use of several modules developed during the first part of the course's experiments. These experiments addressed the construction of a

servomotor control (via PWM modulation), the creation of serial transmission and reception modules, and the interfacing of an ultrasonic sensor (HC-SR04). While reusing the source code from previous work accelerated the development of new components, it also carried over small unresolved issues from the initial module development. These issues were minor at the time since they did not affect the execution of the early experiments. However, the application of the Bayesian filter requires uninterrupted circuit operation. In this new context, these inherited small issues became amplified, as the filter's operation increased the frequency of component interactions, partially preventing the correct functioning of the circuit.

The primary issue identified with the reuse of the modules is the interruption of the main circuit's operation—essentially, a freezing of the circuit, in this case, halting the acquisition of new measurements. This problem was identified around the middle of Experiment 4. From the start of the project, it was known that a freezing issue arose from the integration of some components developed in the first part of the course. Weekly attempts were made to address this issue, alongside the development of validation tests, module construction, and the execution of tasks outlined in the project specification document. However, no patterns were identified in the gaps observed during testing on the FPGA board.

Several alternative solutions were explored to bypass the issue. Bypassing refers to any measure that appeared to resolve the problem without identifying its root cause. In these attempts, each module was inspected, and unit tests were run. Despite this, no component failures were detected. It was observed that the integration of the components is what triggers the main circuit's failure.

One hypothesis is that there is a synchronization failure that causes one or more of the control units or the main circuit to enter an exception state. The entry into an exception state would result in a freeze, as the component would be unable to resume the process of acquisition, filtering, and transmission of measurements. The exception state could be directly related to the lack of synchronization between components, as sometimes the transition of one state depends on generating a signal to start or end the operation of another component. This hypothesis has not yet been confirmed, but efforts to resolve the issue are following this line of reasoning.

7 Conclusion

Based on the activities carried out, a successful Adaptive Filtering System with a Bayesian Filter was developed, capable of generating estimates with the potential to enhance the reliability of low-cost sensors. Additionally, it was possible to validate the behavior of the adaptive filtering circuit in hardware, improving the performance of its output. The executions and tests conducted throughout the experiments involved the use of tools such as the *Python Notebook* for controlling and verifying input and output data, *MQTT Dash* for remote communication between nodes and the digital lab (which includes the transmission and reception of data to verify the functionality of the developed circuits), as well as *Intel Quartus Prime* and *ModelSim* for the development of the code and circuit testing (using *testbenches*), respectively.

All of these activities and this report have been successful, and the results obtained are in full compliance with the established specifications.

Bibliography

[1] A. M. Nascimento, P. S. Cugnasca, L. F. Vismari, J. B. C. Junior, and J. R. de Almeida, "Enhancing the accuracy of parking assistant sensors with bayesian filter," in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, IEEE, 2018, pp. 1–6.

[2] A. M. Nascimento, P. S. Cugnasca, L. F. Vismari, J. B. C. Junior, and J. R. de Almeida, "Research notes on low-cost sonar distance sensor," *Unpublished*, n.d. Available upon request.

[3] F. de Almeida, L. Sato, and E. Midorikawa, "Tutorial for creating digital circuits in vhdl on quartus prime 16.1. digital laboratory handbook," Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da USP, Technical Report, 2017.

[4] ALTERA-Intel, *De0-cv user manual*, Manual, 2015.

[5] ALTERA-Intel, *Quartus prime introduction using vhdl designs*, Manual, 2016.

[6] ALTERA-Intel, *Quartus prime introduction to simulation of vhdl designs*, Manual, 2016.

[7] R. D'Amore, *VHDL - Description and Synthesis of Digital Circuits*, 2nd. LTC, 2012.

[8] B. Mealy and F. Tappero, *Free Range VHDL - The no-frills guide to writing powerful code for your digital implementations*, 1.21. Free Range Factory, 2018.

[9] R. J. Tocci, N. S. Widmer, and G. L. Moss, *Digital Systems: Principles and Applications*, 11th. Prentice-Hall, 2011.

[10] J. F. Wakerly, *Digital Design Principles & Practices*, 4th. Prentice Hall, 2006.