EMBEDDED SYSTEMS
RESEARCH
GROUP

University of Minho
School of Engineering

Integrated Master in Industrial Electronics and Computers Engineering

Project 1

Zephyrus

Hand Motion-Controlled Remote Car

Authors:
Hugo Carvalho A85156
José Mendes A85951

Professor:
Dr. Adriano Tavares

January, 2021

Acronyms

ACK Acknowledge. i, 30

ADC Analog-to-Digital Converter. i, vii, 72, 78, 81, 82, 95

API Application Programming Interface. i, 46, 79

BMS Battery Management System. i, 38, 39

BOM Bill Of Materials. i, 34, 40

CAD Computer Aided Design. i, 46

CMSIS Cortex Microcontroller Software Interface Standard. i, 79

COTS Commercial Off-The-Shelf. i, 28, 33, 34, 38, 46, 64

CRC Cyclic Redundancy Check. i

CS Chip Select. i, 31

FFNN Feed-Forward Neural Network. i, vi, 55

FPS Fixed-Priority Scheduling. i, 47

GPIO General-Purpose Input/Output. i, vii, 36, 53, 69, 76, 77, 92

HAL Hardware Abstraction Layer. i, 46

HMI Human-Machine Interface. i, 67

HSE High Speed External Clock. i, 77

HSI High Speed Internal Clock. i, 77

HW Hardware. i

I^{2}C Inter-IC Communication. i, v–viii, 29, 30, 51, 77, 80, 81, 92, 94, 95

I/O Input/Output. i

IC Integrated Circuit. i, 34, 36, 37, 41, 45, 59, 64, 66, 92

IDE Integrated Development Environment. i, 46

IMU Inertial Measurement Unit. i, 86, 91, 92, 98, 111

IR Infra-red. i, 35

ISM Industrial, Scientific and Medical. i, 34

ISR Interrupt Service Routine. i

LDO Low-Dropout Regulator. i, 41

LP Low-Power. i, 70–72

LSB Least Significant Bit. i

LSE Low Speed External Clock. i, 77

LSI Low Speed Internal Clock. i

MCU Microcontroller Unit. i, 29, 33, 42–45, 57, 70, 76

MIFA Meandered Inverted-F Antenna. i, 34, 64

MISO Master-In Slave-Out. i, 31

ML Machine Learning. i, 46

MOSFET Metal-Oxyde Semiconductor Field Effect Transistor. i, 36

MOSI Master-Out Slave-In. i, 31

MPU Motion Processing Unit. i, 34

MSB Most Significant Bit. i, 30

NN Neural Network. i, 54, 55

NRE Non-Recurring Engineering. i, 65

NVIC Nested Vectored Interrupt Controller. i, vii, 77, 92

PCB Printed Circuit Board. i, 46, 57, 61, 62, 66

PCC Power Consumption Calculator. i, vii, viii, 71, 72, 101–103

PLL Phase-Locked Loop. i, 78, 92

PWM Pulse-Width Modulation. i, 36, 78, 93

R&D Research and Development. i, 2

RAM Random Access Memory. i, viii, 107

RCC Reset and Clock Control. i, vii, 77

RF Radio Frequency. i, 40, 47, 65, 76

RL Reinforcement Learning. i, 47

RTOS Real-time Operating System. i, 9, 47, 77, 79, 83, 96, 108, 111, 113

RTX IntervalZero Real-time Operating System. i, 79

SCL Serial Clock. i, 29, 30

SCLK SPI Clock. i, 31

SDA Serial Data. i, 29, 30, 92

SMD Surface Mount Device. i, 59

SMT Surface Mount Technology. i, 59

SPI Serial Peripheral Interface. i, vii, 31, 32, 53, 69, 80, 94

SW Software. i, 68

USS Ultrasonic Sensor. i, 55

Contents

List of Figures

Chapter 1

Chapter 2

Chapter 3

Chapter 4

List of Tables

Chapter 1

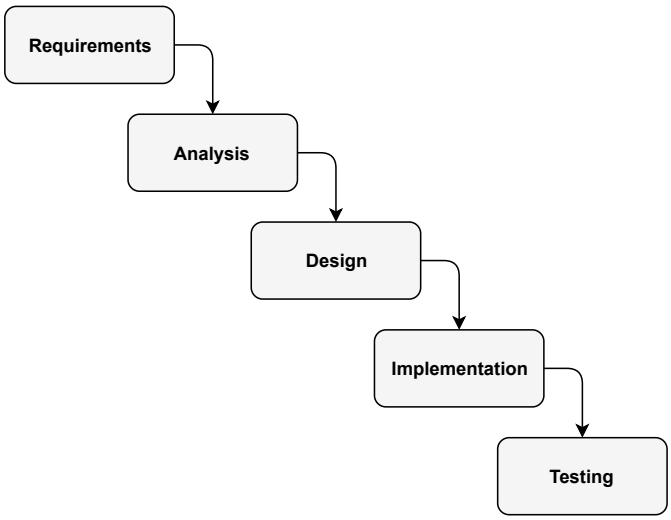
Introduction

This document consists of an analysis and presentation of the work done in the development of Zephyrus.

In the first chapters, the product will be put in the right context globally to understand the need for its existence and where it fits in the market. A high-level overview of the project will also be provided in these chapters. This is meant for future reference when designing the product as well as analysing the report.

Further in the document, one will be able to find documentation detailing the design and implementation processes, which will hopefully be simple to understand and reproduce.

The report will culminate in the test and verification of the results achieved in the development stages, in order to understand the impact of the decisions that were made and where to make improvements in future work.



1.1 Problem Statement

Recent developments in research of motion-tracking and gesture-controlled robotic products have shone a new light on entertainment, special needs teaching, and accessibility-focused applications. Products like motion-sensing console controllers, robotic arms, and prosthetics have popularized and extended the knowledge on gesture and impulse-based technologies throughout the past few years, and their market is predicted to continue to grow in the future, especially in Asia and Oceania [11], as presented in figure 1.1.1.

To provide a better understanding of the concepts associated with the forementioned technologies through Research and Development (R&D), the project will consist of a gesture-controlled car with a wristband as a remote controller. The user will be able to accelerate and steer the vehicle using natural wrist and finger movements.

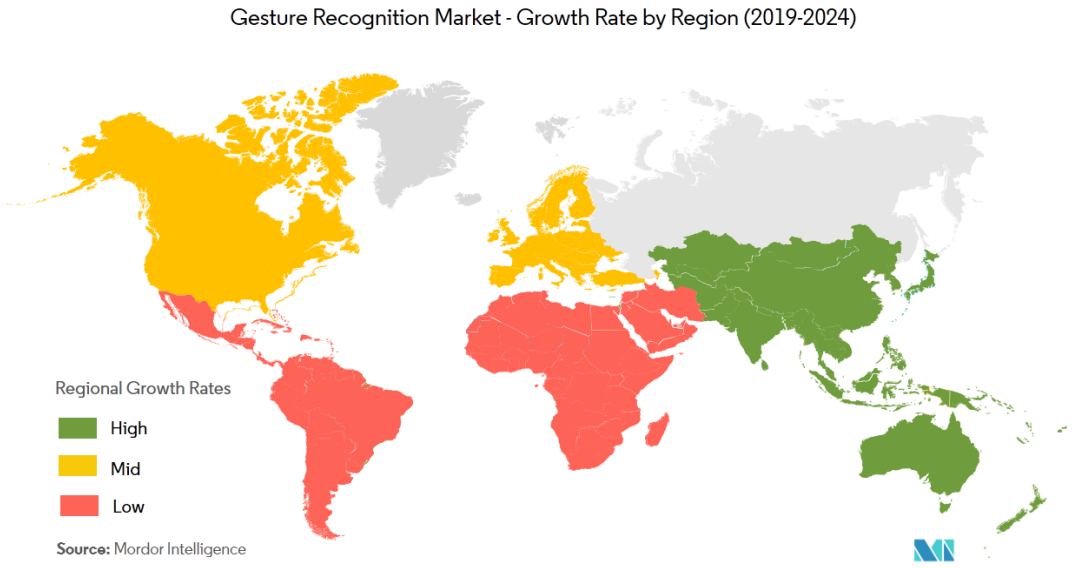


Figure 1.1.1: Gesture Recognition Market Growth

1.2 Problem Statement Analysis

From the Problem Statement Analysis, one can extrapolate an overview of the main components, features they provide and relationships between them. The schematic in figure 1.2.1 describes a system comprised by:

- A **Local Computational Unit**, which will control the wheels of the car in accordance to the user inputs, which it receives from the Remote Computational Unit;
- A **Remote Computational Unit**, which senses user input and converts it into a digital format, sending it to the Local Computational Unit.

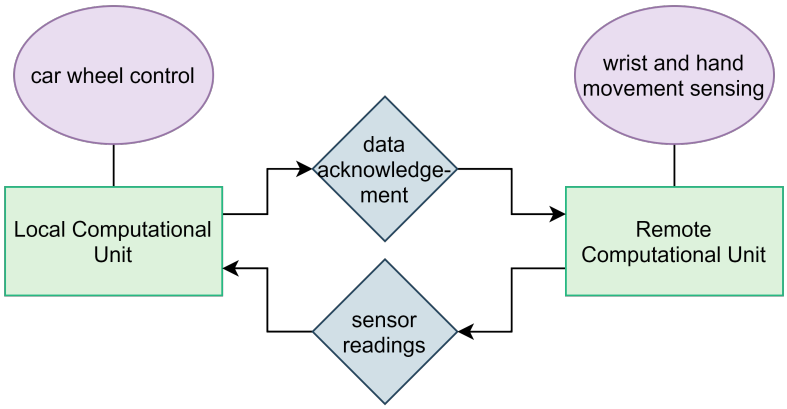


Figure 1.2.1: Problem Statement Diagram

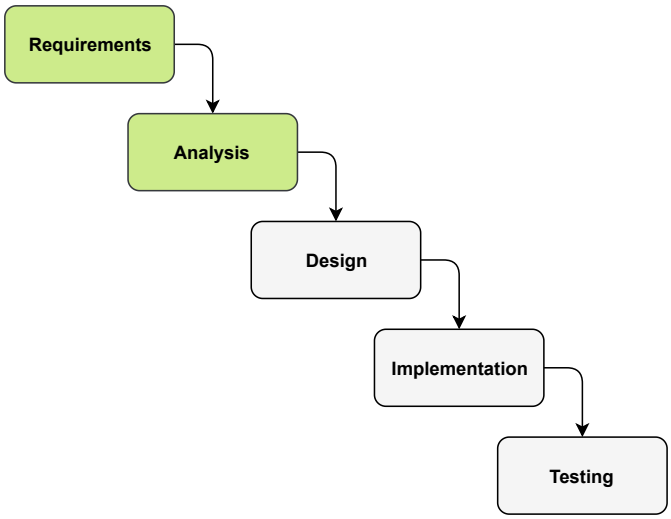
Chapter 2

Analysis

This chapter starts with brief market research about similar products to ultimately distinguish Zephyrus from its competitors.

The requirements and constraints of the project are presented as well as a general system overview to better visualize the objectives. Additionally, the system is analyzed in terms of its software and hardware architectures and divided into local and remote, proceeding with the presentation of various diagrams that allow further understanding of each system.

This stage ends with the proposed initial budget and the project's Gantt diagram.



2.1 Market Study

The market for gesture-controlled or gesture-recognition products is not very crowded, but it is competitive nonetheless [11], as displayed in figure 2.1.1. As stated, some of the best efforts in this area can be found in research and, in that sense, both strands will be analysed in this section, as a mean to understand, not only what this product might try to compete with, but also what technologies are being developed which might be used in future competing products or iterations of this same project.

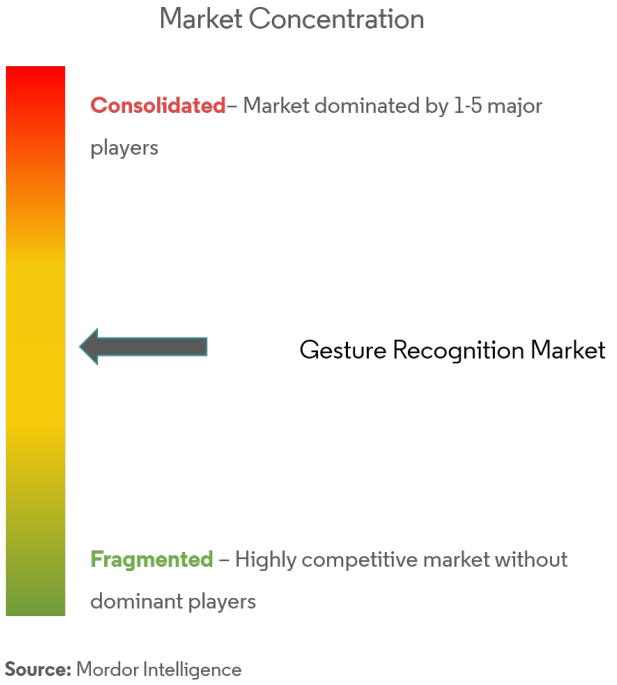


Figure 2.1.1: Gesture Recognition Market Placement

Ultigesture's Gesture controlled RC Car (Kickstarter)

This product was revealed in 2017 along with a Kickstarter campaign, with the promise of being easy and intuitive to control.

Figure 2.1.2: Ultigesture's arm gesture-controlled RC car

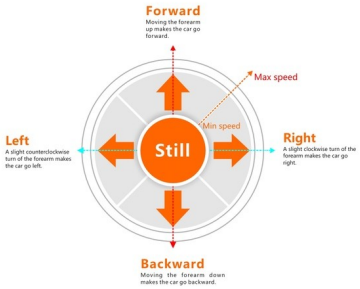


Figure 2.1.3: Ultigesture's gesture-controlled RC car's instructions

Pros:

- Practical setup;
- Appealing design;
- The use of Bluetooth allows the car to be controlled both by arm gestures and smartphone.

Cons:

- Unpractical and uncomfortable to command.

Specified range: $20m$

Predicted cost: 89$

Generic Glove-controlled RC Quadcopter Drone

This is a generic toy glove-controlled drone which can found in any Chinese website. It is controlled by hand movements and it also includes a button for keeping the drone hovering in the same place.

Figure 2.1.4: Generic Glove-controlled RC Drone

Pros:

- Practical and comfortable to command;
- Impactful first impression;
- Cheap.

Cons:

- Cheap-feeling;
- Can only be worn in the right hand.

Cost: 36$

EMG/EEG Controlled RC Car

In 2016 a team composed by Christian Ward, James Kollmer, Robert Irwin and Andrew Powell designed a quick prototype for a toy car which was controlled with arm movement and brain activity. Both sensing peripherals, as well as the car were connected to a computer, which ultimately controlled the movement of the car. It would be unfair to analyse this as a final product but it will serve as a reference for what can be done in practise.

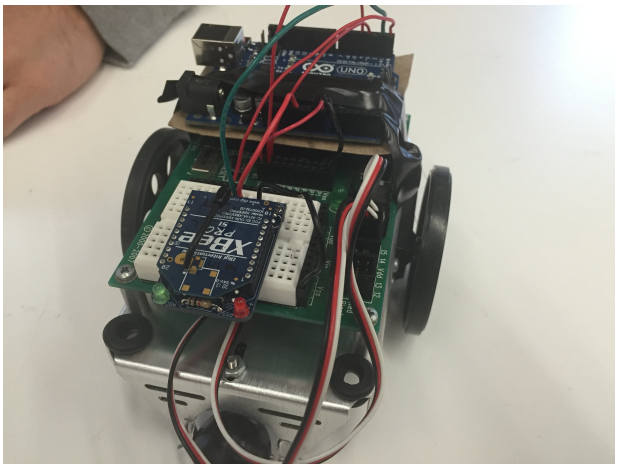


Figure 2.1.5: EMG/EEG Controlled RC Car

2.2 Added Value

Although there aren't many products in the market to compete with Zephyrus, strong efforts have been made in certain key aspects, such as good usability and competitive pricing, in the existing products. Zephyrus, specifically, aims to match all those products in those characteristics, but also be easy and versatile to set up, as well as comfortable to use and control.

2.3 Requirements and Constraints

The development requirements are divided into functional and non-functional if they are main functionalities or secondary ones, respectively. Additionally, the constraints of the project are classified as technical or non-technical.

2.3.1 Requirements

Functional Requirements

- Sense the user inputs and generate the band outputs;
- Take the wristband inputs and generate the desired motor outputs;
- Control the establishment and closure of the connection from the remote system.

Non-Functional Requirements

- Have low power consumption;
- Have low latency in the connection between local and remote;
- Have a comfortable and practical wristband/glove;

2.3.2 Constraints

Technical Constraints

- Use the Nucleo-STM32F767ZI as the development board;
- Use at least three different types of sensors;
- Use FreeRTOS as the RTOS;
- Use a controller based on Reinforcement Learning following the Actor-Critic method;
- Use Keil MDK-ARM as the development environment;
- Use Object-Oriented Programming Languages;

Non-Technical Constraints

- Project deadline at the end of the semester;
- Pair workflow;
- Limited budget;

2.4 System Overview

The diagram in figure 2.4.1 represents an initial big picture of the project to facilitate objective identification. Concerning the diagram, one can identify three main blocks:

- The **Local Board** block;
- The **Remote Board** block;
- The **External Environment** block.

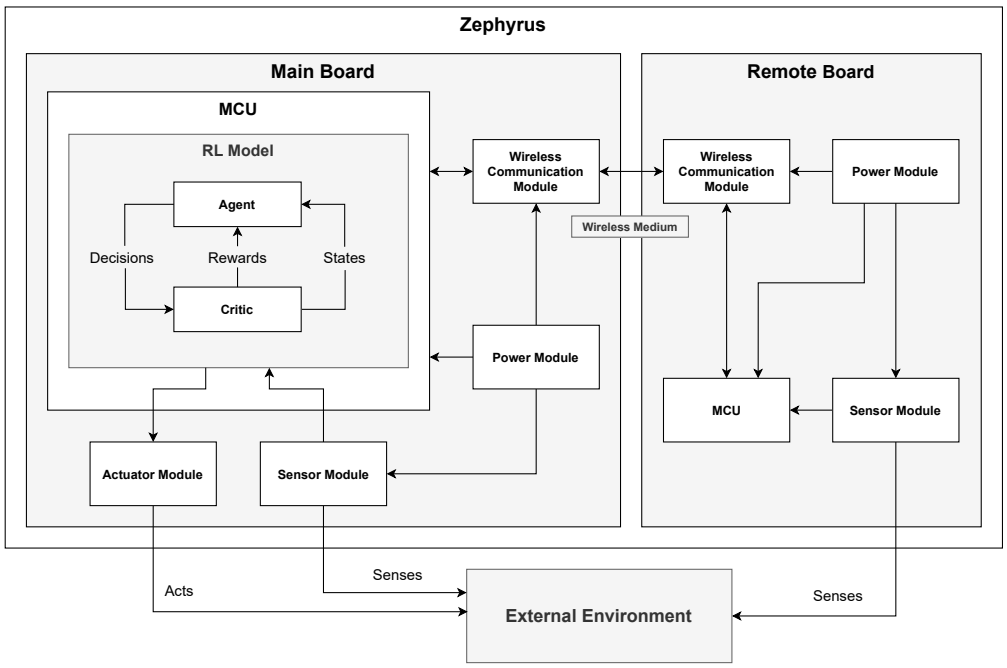


Figure 2.4.1: System Overview Diagram (Aug. in Appendix A.1)

2.5 System Architecture

2.5.1 Hardware Architecture

The diagram in figure 2.5.1 represents an initial hardware big picture to facilitate objective identification. Concerning the diagram, aside from the External Environment, one can identify two main blocks:

- The **Local Board**. This unit is comprised by the main MCU (in the STM32 board) for housing the RL Model and intermediate its interactions with the world, as well as the Actuator Module, for driving the wheel motors, the Power Module,

for power delivery and management, and the Wireless Communication Module, for facilitating the communication with the Remote Board through a wireless medium;

- The **Remote Board**. This unit contains the Sensor Modules network of the project for sensing the environment, its own MCU, whose job is to concentrate and multiplex the sensor network, sending data through its own Wireless Communication Module, and the Power Module which will provide all of the other modules with power.

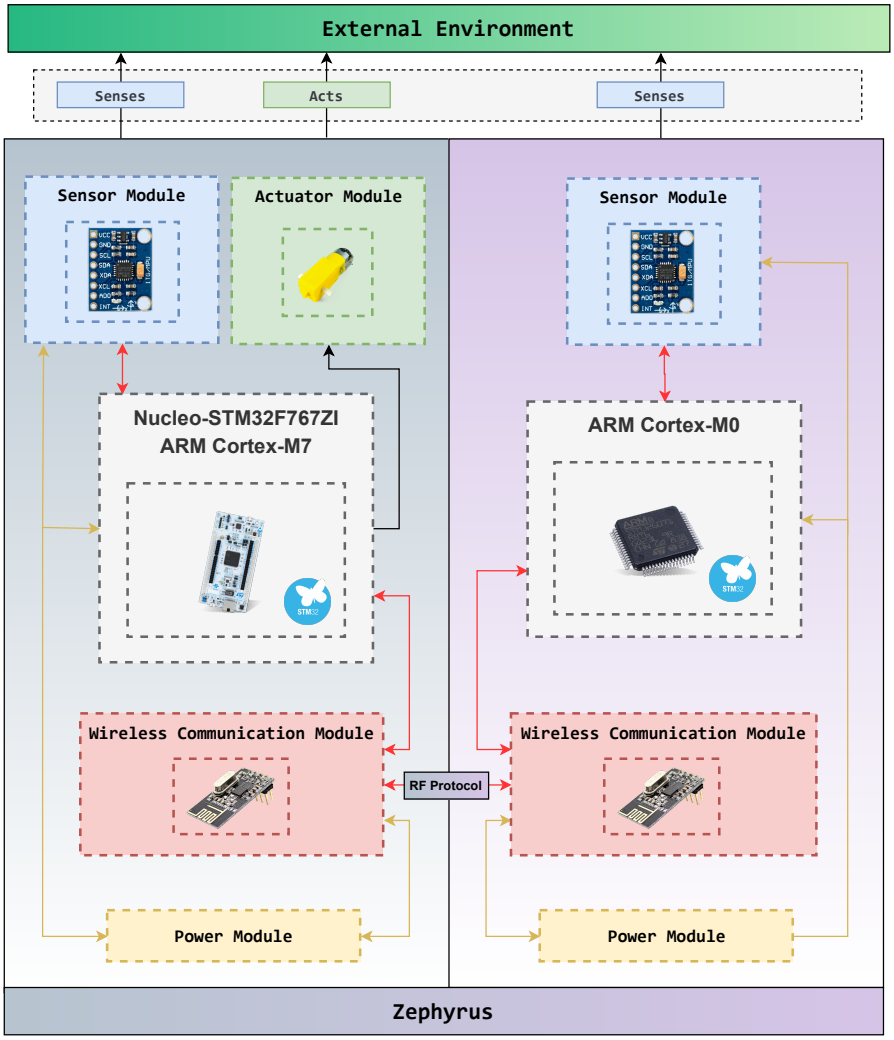


Figure 2.5.1: Hardware Architecture Diagram

2.5.2 Software Architecture

The software architecture for both the local and remote systems (figures 2.5.2 and 2.5.3, respectively) is divided into three layers, those being:

- The **Operating System** layer, which is comprised by the Operating System drivers and Board Support Packages;
- The **Middleware** layer, which includes software for abstracting the lower level packages and streamlining the development of complex algorithms;
- The **Application** layer, where the core functionality of the program is built, with resource to the API's in the lower level layers.

Both are soft real-time systems with, focused on low power consumption and effective communication, both with each other and with the sensor modules. As such, the analysis for both systems foresee the necessity for a Real-Time Operating System for managing real-time events and multitasking, drivers for serial communication and specific power management utilities.

The Remote System is mainly focused on sensing and processing values relative to the environment while sending information pertaining those values to the Local System.

The Local System, on the other hand, serves the specific purpose of hosting a controller that is composed by a Reinforcement Learning model. In that sense, it also needs specific, microcontroller-optimized middleware for RL and motor control.

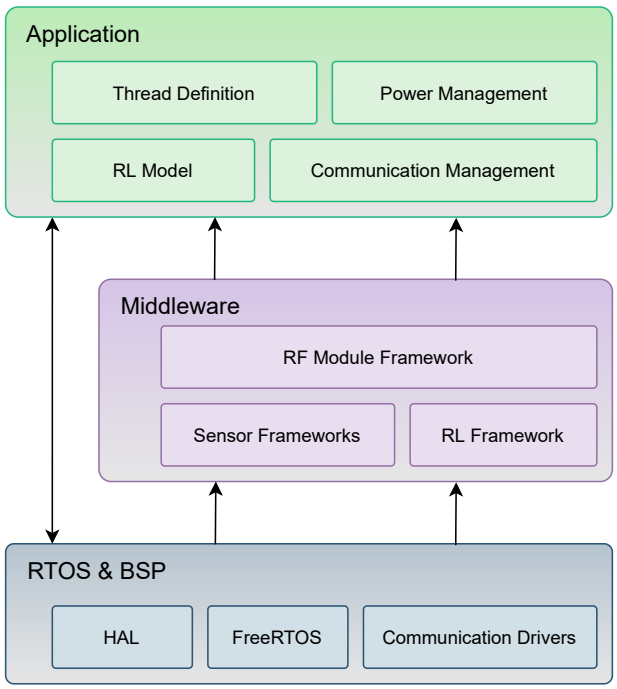


Figure 2.5.2: Software Architecture Diagram - Local System

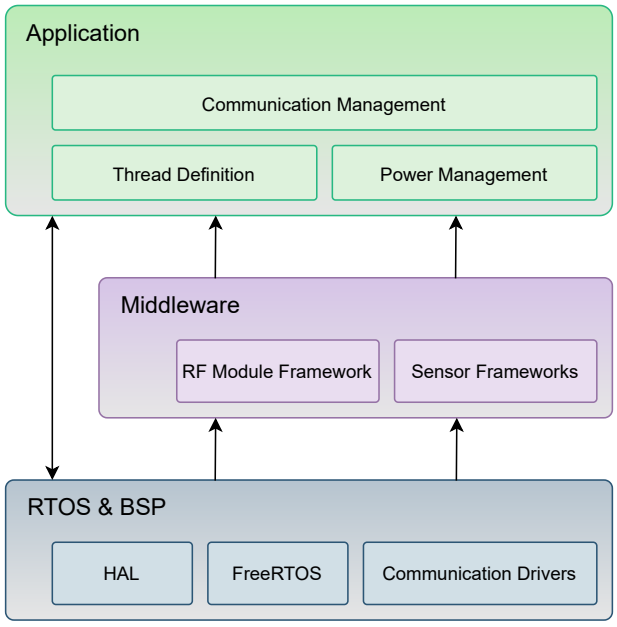


Figure 2.5.3: Software Architecture Diagram - Remote System

2.6 Local System

As referred to in section 2.4, the overall system includes a bulkier **Local System**. The analysis of the latter will be presented in this section.

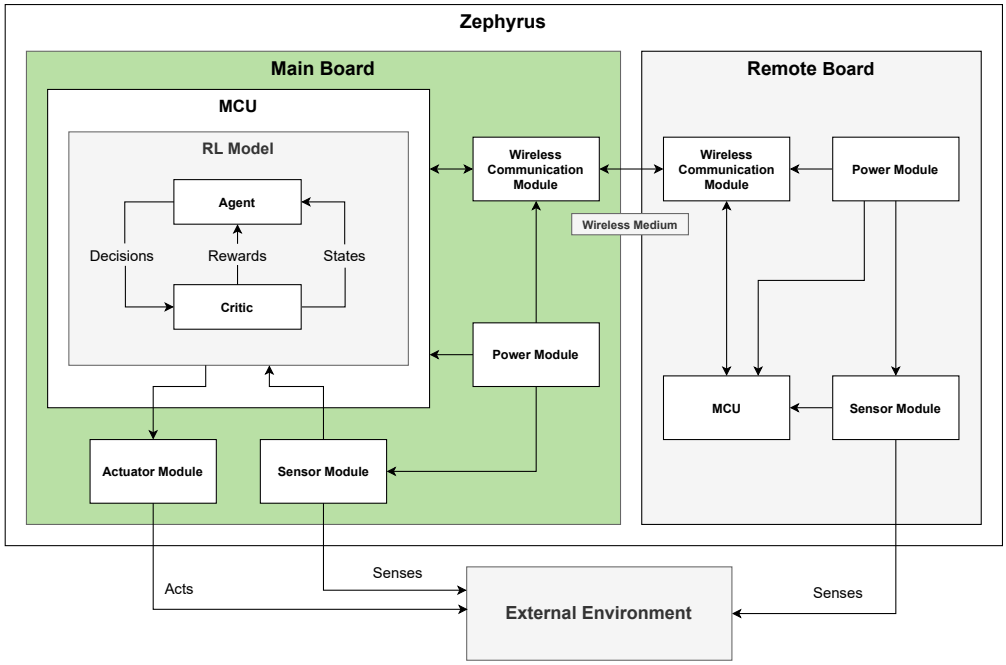


Figure 2.6.1: System Overview Diagram - Local

2.6.1 Events

In order to understand how the system works, it is important to know what are the most important events that can take place in it and how it will respond to each one of them. Table 2.1 highlights these events from the perspective of the Local System, categorizing them by their source and synchronism and linking it to the system's intended response.

Event	System Response	Source	Type
Power on	Initialize sensors and listen for connnection	User	Asynchronous
Connection Request	Verify credentials and respond to the request	Remote System	Asynchronous
Sampling Period Elapsed	Request sample from sensors	Local System	Synchronous
Command Received	Prepare the information to be part of the state observations to be considered by the agent	Remote System	Asynchronous
RL Timestep Elapsed	Calculate target value	Inference Engine	Synchronous
Target Value Calculated	Perform the action infered by the agent	Agent	Asynchronous

Table 2.1: Local Events

2.6.2 Use Cases

It is necessary to study the system's possible use cases to understand and model its intended behaviour. In the Local System's case, all the interactions happen with the Remote System. The diagram in figure 2.6.2 illustrates those interactions.

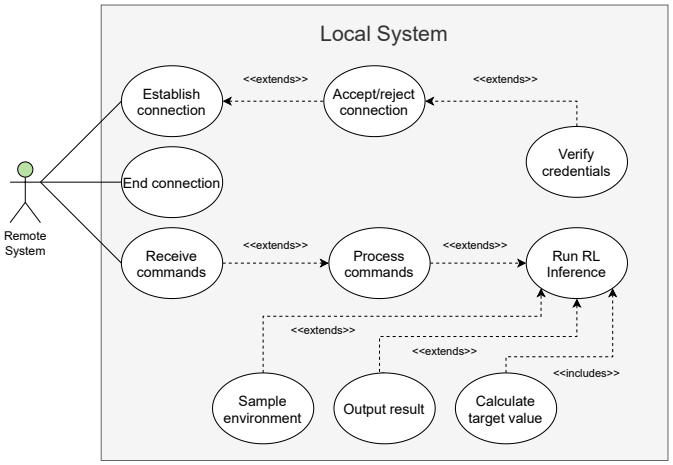


Figure 2.6.2: Local System's Use Case Diagram

2.6.3 State Machine Diagram

The two main functional states of the Local System - *Startup* and *Execution* - and their division into more complex state machines corresponding to each of its subsystems, are depicted in figure 2.6.3.

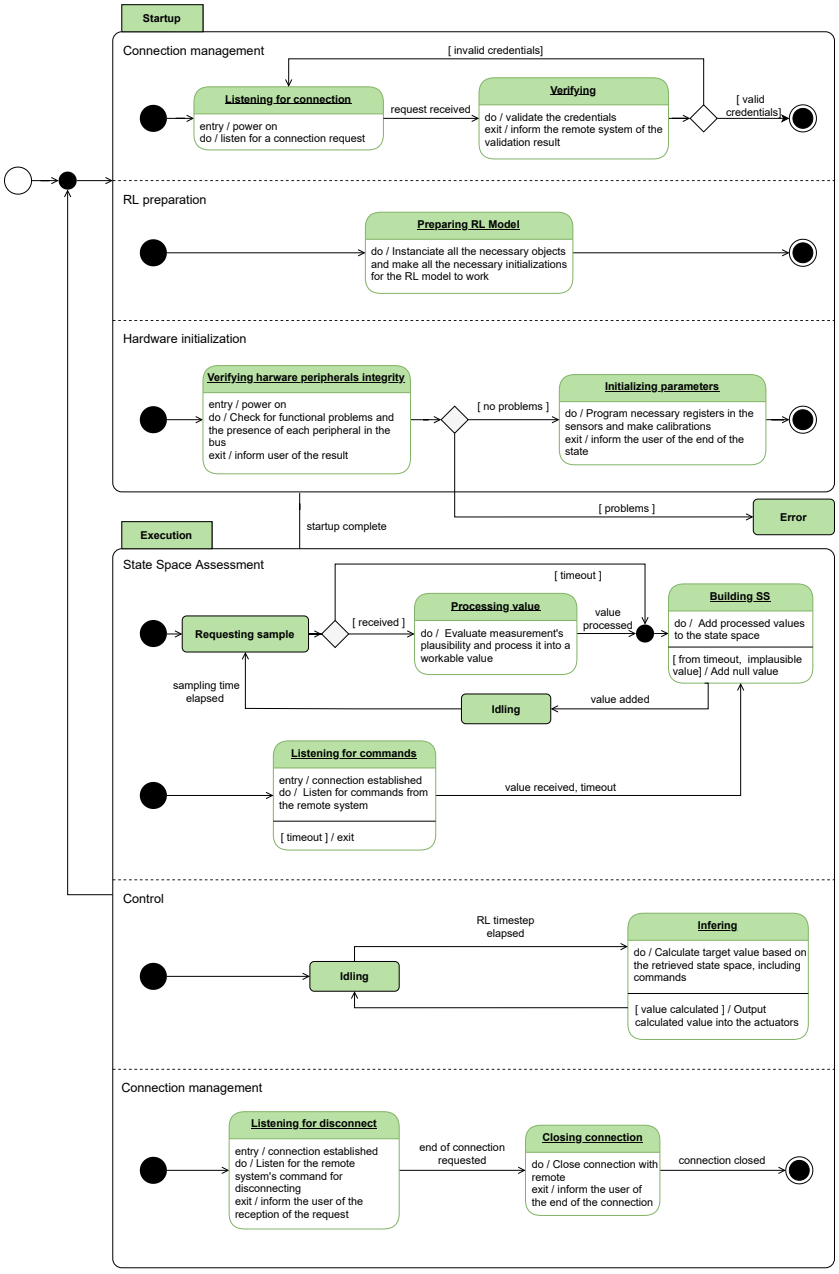


Figure 2.6.3: State Machine Diagram - Local (Aug. in Appendix A.6)

2.6.4 Sequence Diagram

The model for the Local System's intended response and behaviour relative to different stimuli dynamically, is presented as a sequence diagram in figures 2.6.4 and 2.6.5.

When powering on, the Local System will immediately initialize all required aspects of sensor-related hardware components and verify their integrity, to be able to start sampling as soon as it is necessary. It will also start listening to connection requests from the Remote System. Upon reception of one of such requests, it will accept or reject it based on previously established connection credentials. Whenever connected, the Local System will continuously sample the environment around it, changing the state observations for the Reinforcement Learning model.

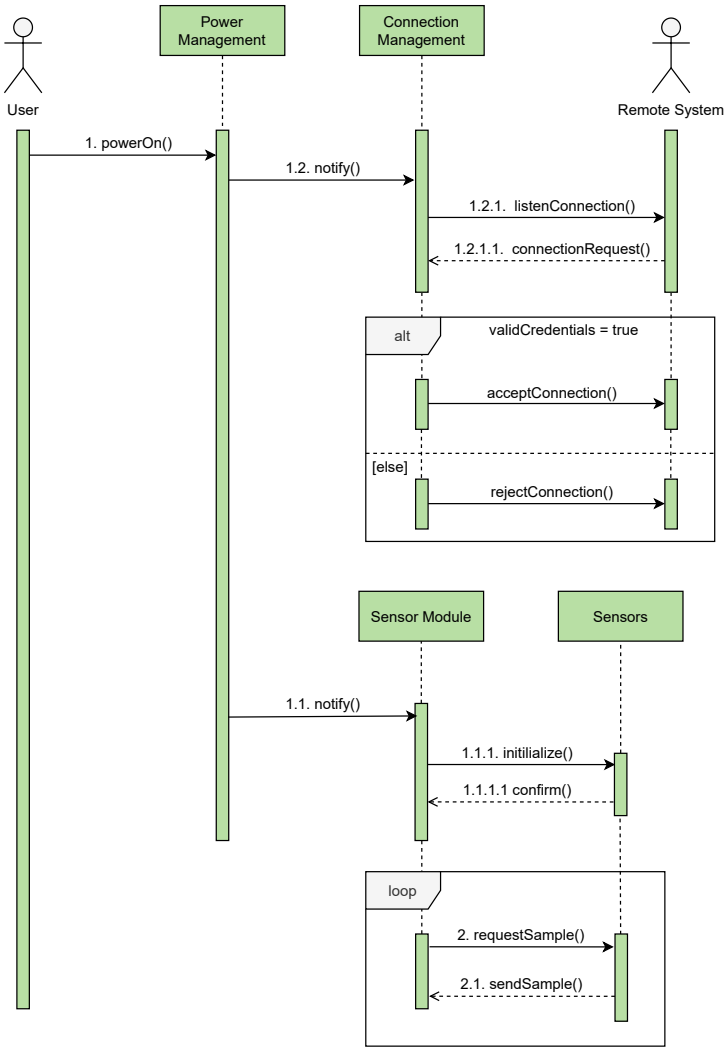


Figure 2.6.4: Local System Sequence Diagram (Aug. in Appendix A.2)

As previously mentioned in table 2.1, whenever a command from the Remote System is received, it is to be made part of the state observations for the Reinforcement Learning algorithm as well. In case a command is not received in time, the last command received will be repeated to preserve the predictability of the system.

Periodically, an iteration of the control routine is executed. The first logical step in that chain of events is the calculation of the target output values by the Agent, which will be followed by the output of said values and calculation of the reward that will allow the agent to update the policy.

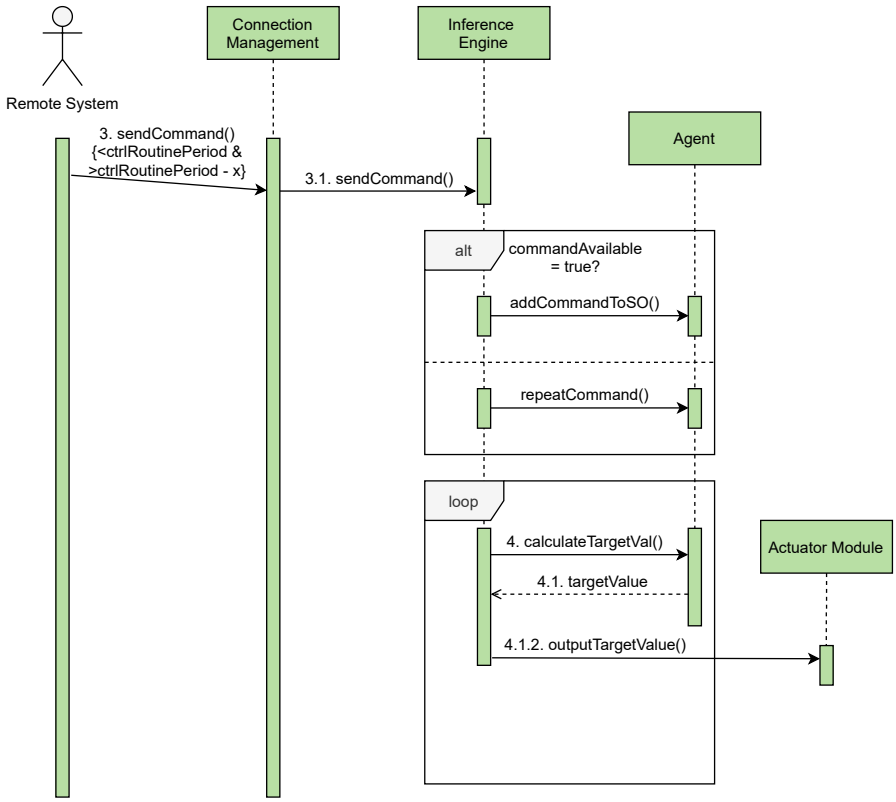


Figure 2.6.5: Local System Sequence Diagram (Aug. in Appendix A.3)

2.7 Reinforcement Learning

The **Reinforcement Learning Module** is highlighted in the system overview of figure 2.7.1.

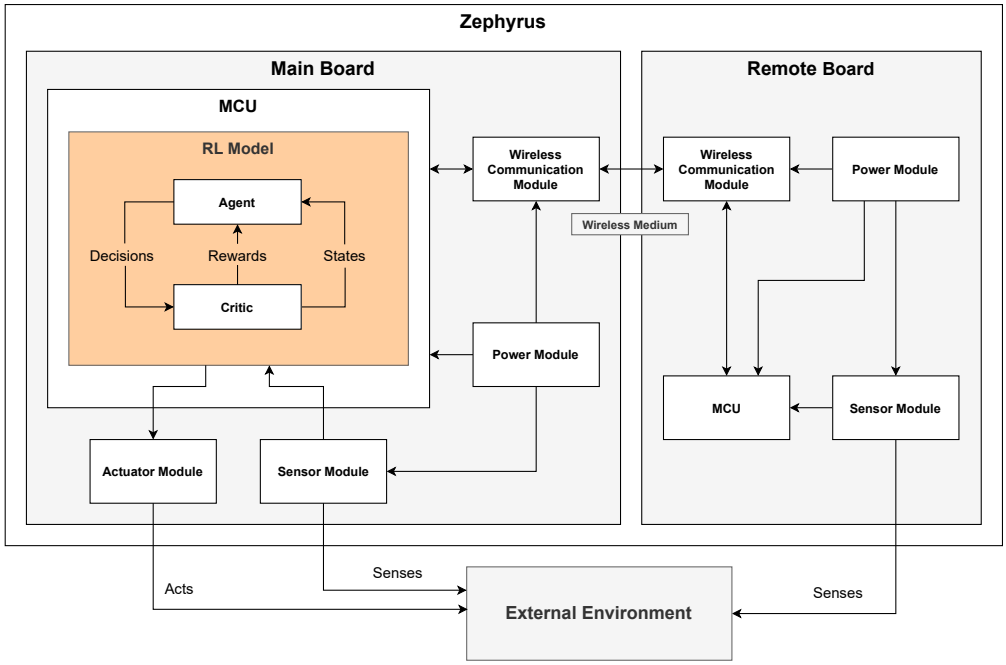


Figure 2.7.1: System Overview Diagram - Reinforcement Learning Model

2.7.1 Algorithm Selection

All the diverse Reinforcement Learning algorithms can be categorized into one of three main families [9]:

- **Action-value** family;
- **Actor-critic** family;
- **Policy-gradient** family;

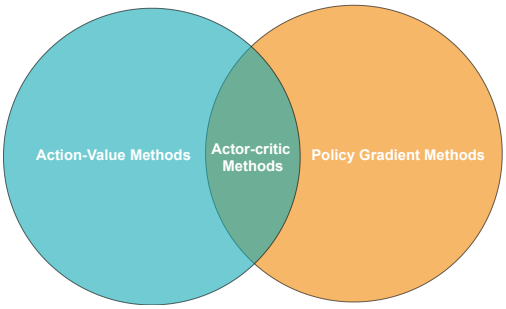


Figure 2.7.2: Reinforcement Learning Algorithm Families

As one can discern from figure 2.7.2, the Actor-critic method combines advantages from the other families allowing an algorithm in which a value function learns the appropriate policy. In this project, one will use the latter method going forward.

In figure 2.7.1 one can identify three main components of an Actor-Critic Reinforcement Learning Model:

- The **Actor/Agent** is the controlling entity of the system, which dictates actuator outputs based on the current state of the system, internal and external.
- The **Critic** is the entity that works to tune the behaviour of the Actor through a reward-based policy.
- The **External Environment** can be identified as everything that is not the Actor or Critic but is influenced by the Actor's actions and helps to stimulate it.

The model is time-dependent and sequential, since the reward calculation is based, not only on the assessment of the Agent's performance in the present, but also on past rewards. The environment is also stochastic because some inputs change randomly in time and it's also non-deterministic, meaning, the same inputs can generate different outputs as the neural network changes between policy updates.

2.7.2 Reinforcement Learning Workflow

After specifying the type of learning algorithm to utilize, one must follow a basic training workflow [6] as presented in figure 2.7.3.

Figure 2.7.3: Reinforcement Learning Workflow Schematic

2.7.3 Model Conversion & Deployment on Embedded Target

The final step of the mentioned process is the model conversion and deployment on the target device. In this case, the system is a resource-scarce embedded system [1], which demands a more careful deployment considering the compatibility between the model developing tool and the embedded target.

The tool selected for performing the model creation and training was **Keras**, which provides a wrapper on the **Tensorflow** API, that has been extensively tested with many processors based on the **Arm Cortex-M Series** architecture [5] and has been ported to several other architectures. Moreover, it is important to note that one of the supported developing boards listed by Tensorflow was the **STM32F746 Discovery kit** [4] because Zephyrus' Local Board will be an **STM32 NUCLEO-F767ZI**. This resolves any doubts about compatibility issues as the Tensorflow board support should extend to all the STM32F7 family.

For the deployment stage following this approach, one must convert the reinforcement learning model created and trained in TensorFlow to a TensorFlow Lite model specially made for microcontrollers. At this point, two choices were presented regarding deployment:

- Use TensorFlow Lite APIs
- Use STM32 X-Cube-AI APIs

After some research, one discovered that the deployment using X-Cube-AI was faster and took less flash memory than TensorFlow Lite deployment [7]. Additionally, using the same reinforcement learning-generated model it was also possible to speed up inference [7]. The X-Cube-AI core engine [10] is represented in figure 2.7.4, and the proposed **Edge Machine Learning Framework** [1][8] for Zephyrus is presented in figure 2.7.5.

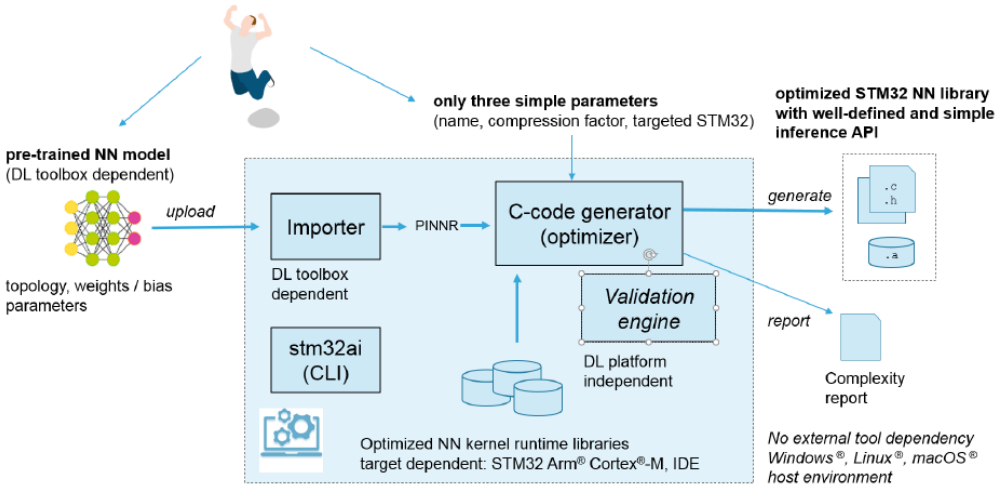


Figure 2.7.4: X-CUBE-AI Core Engine

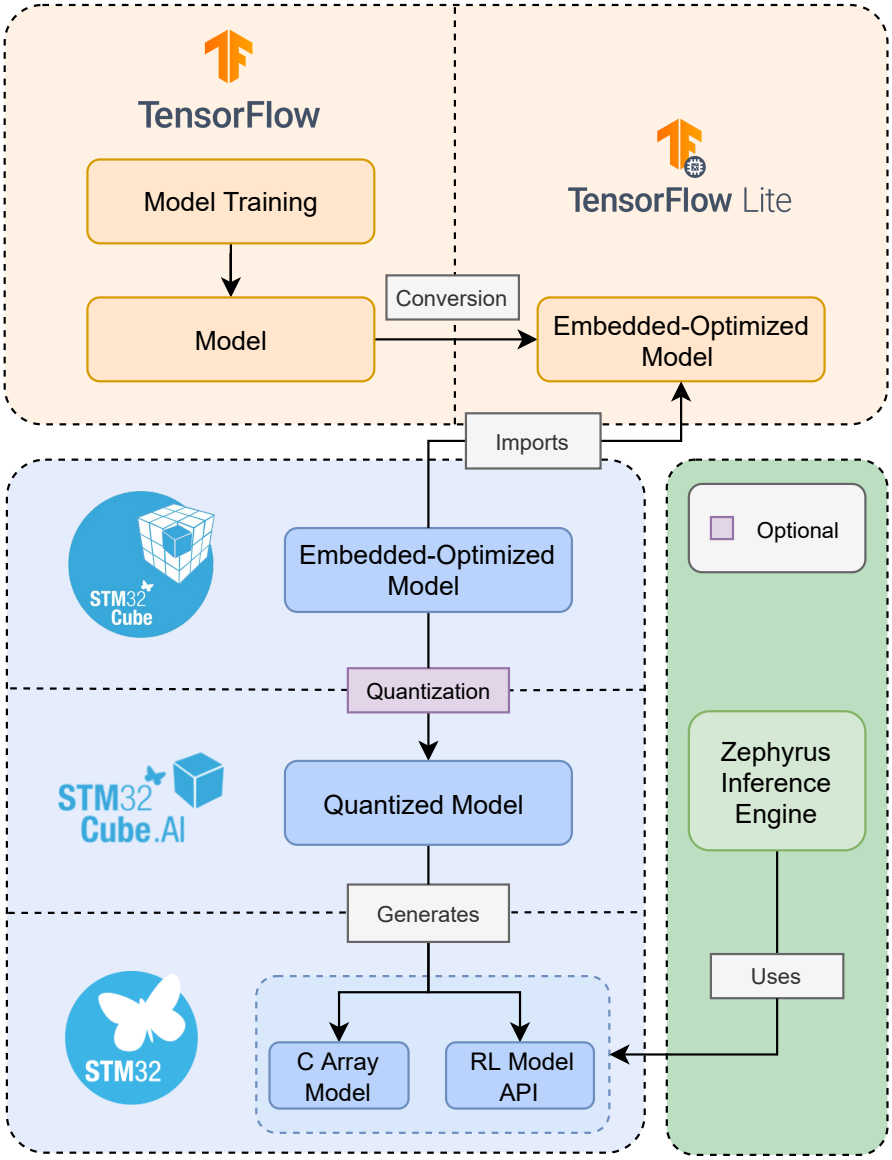


Figure 2.7.5: Zephyrus Edge Machine Learning Framework

2.8 Remote System

The control of the more complex system presented in section 2.6 is intended to be performed by a more compact **Remote System**, to be analysed in this section.

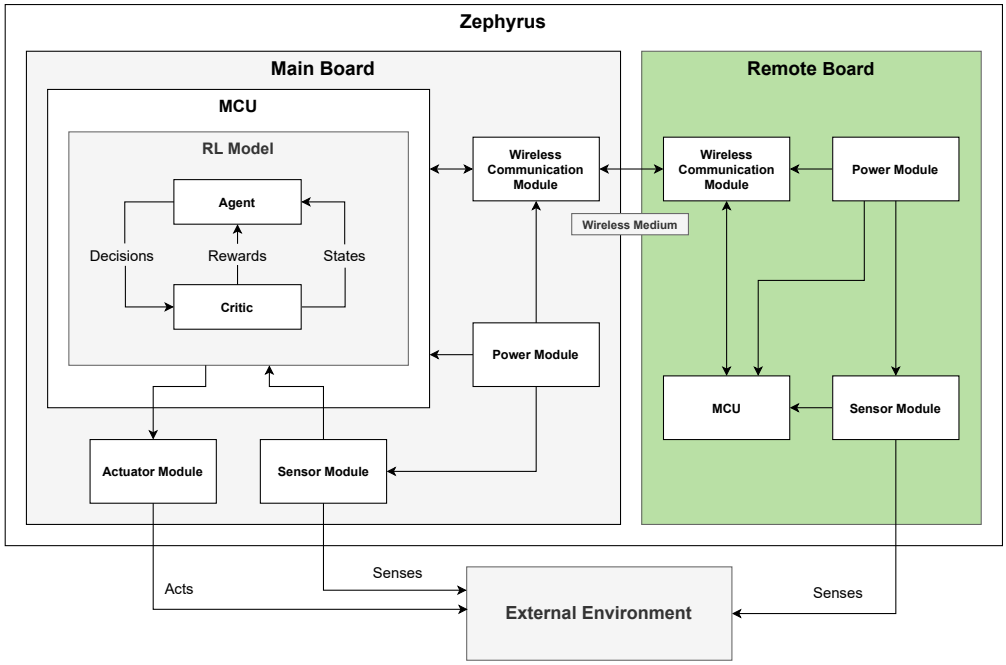


Figure 2.8.1: System Overview Diagram - Remote

2.8.1 Events

Table 2.2 presents the relationship between the most relevant events and their respective responses for the Remote System.

Event	System Response	Source	Type
Init connection button pressed	Try to establish a connection between local and remote	User	Asynchronous
Generate band movement	Send commands to local system	User	Asynchronous
Power on	Initialize sensors	User	Asynchronous
Sampling period elapsed	Request sample from sensors	Remote System	Synchronous

Table 2.2: Remote Events

2.8.2 Use Cases

The Remote System serves as a bridge between the user and the Local System. Thus, its behaviour is defined by its interaction with these two actors, as exemplified by the diagram in figure 2.8.2.

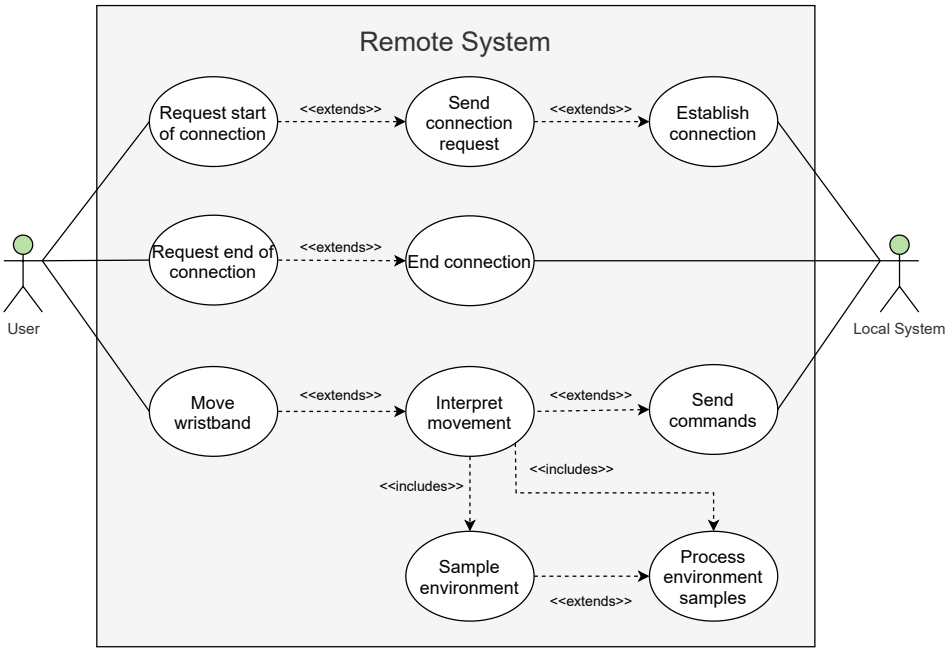


Figure 2.8.2: Remote System's Use Case Diagram

2.8.3 State Machine Diagram

The two main functional states of the Remote System - *Startup* and *Execution* - and their division into more complex state machines corresponding each of its subsystems, are depicted in figure 2.8.3.

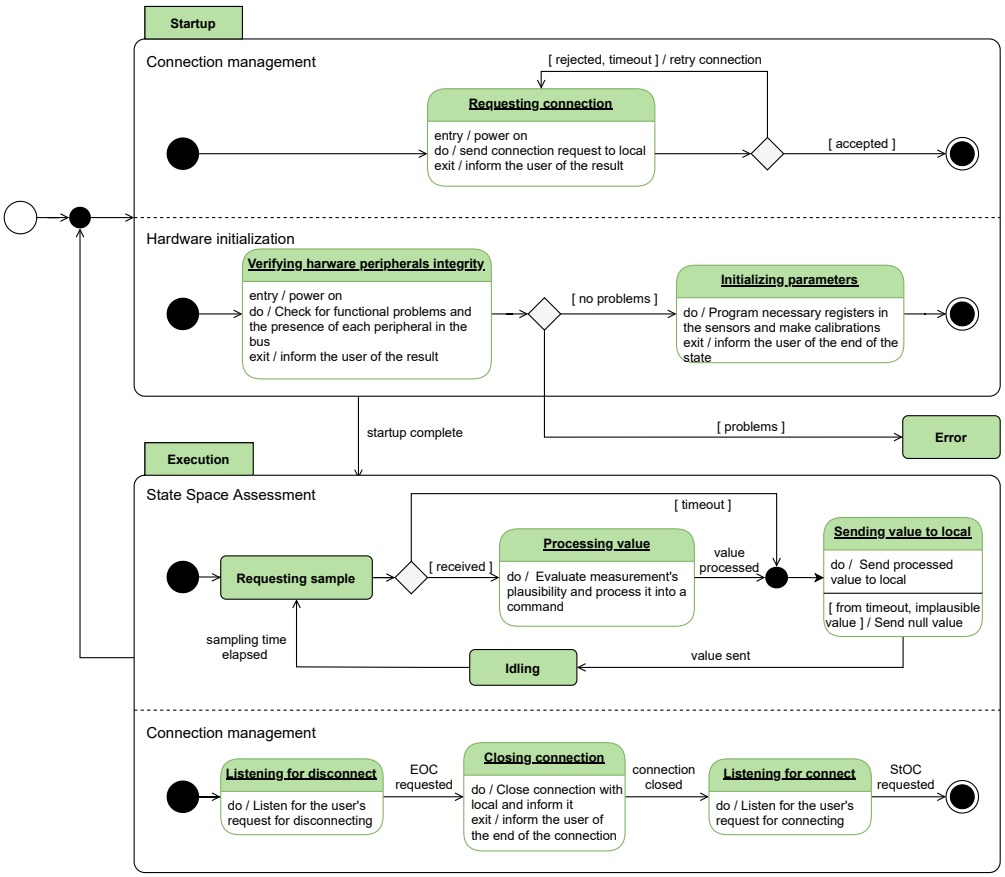


Figure 2.8.3: State Machine Diagram - Remote (Aug. in Appendix A.5)

2.8.4 Sequence Diagram

The Remote System's dynamic behaviour is presented as a sequence diagram in figure 2.8.4

When powering on, the Remote System initializes all required aspects of sensor-related hardware components and verifies their integrity. As soon as a connection with the Local System is established it will start sampling. The next step of the power-on sequence is sending a connection request to the Local System, to be accepted or rejected within an acceptable interval or repeated otherwise.

Periodically, the Remote System's controller will request a sample from the on-board sensors, to be pre-processed as necessary and sent to the Local System, where it will be interpreted as a command.

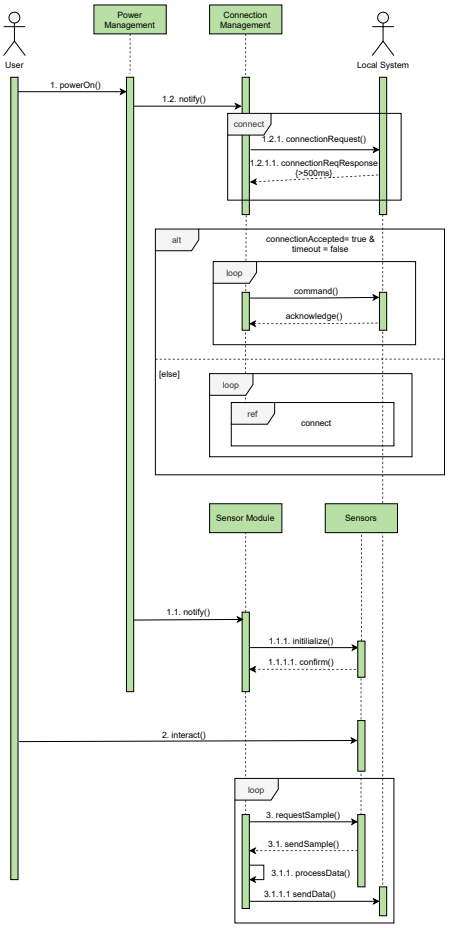


Figure 2.8.4: Remote System Sequence Diagram (Aug. in Appendix A.4)

2.9 Initial Budget

Table 2.3 represents an initial project budget estimate.

Item	**Cost**
STM32 NUCLEO-F767ZI	27€
STM32 F051	3€
Sensor Modules	20€
Power Modules	11€
Communication Modules	7€
Car Structure	14€
Total	82€

Table 2.3: Initial Budget

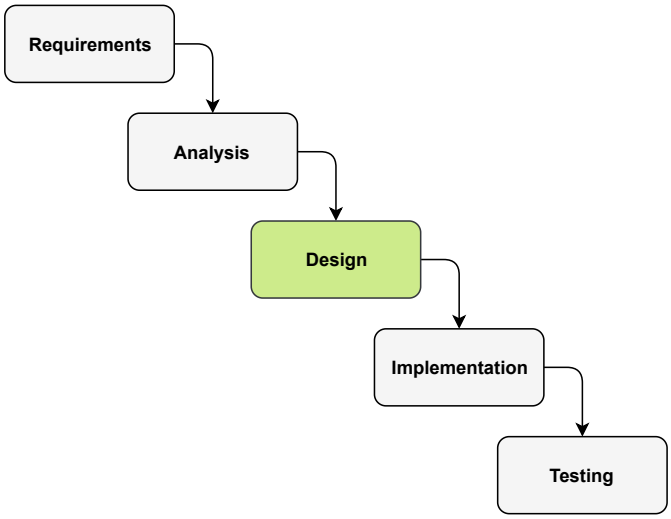
2.10 Gantt Diagram

The Gantt diagram for the project is represented in figure D.1, in the appendix chapter D.

Chapter 3

Design

In this stage, the hardware and software specifications will be approached for each Zephyrus subsystem, computational unit, and remote unit. As the design proceeds, one will specify their peripheral interface, main protocols, working sequences, and the tools and COTS leveraged for the design. At the end of this stage, the overall system must be completely clear for stakeholders, and the guidelines for implementation must be explicit.



3.1 Theoretical Foundations

In this section, some main concept background will be provided, namely relating to the communications protocols.

3.1.1 Protocols

In order to establish communication between the peripherals and the MCUs, the pre-existing and standard protocols that are supported by the peripheral devices will be used and their operation will be explained just as deeply as necessary to understand their how they are to be used in this application.

I^2C

The Inter-IC Communication (I^2C) bus is a half-duplex bidirectional bus that can connect, in a master-slave hierarchy, one or more master devices to one or more slave devices. Communication is done, logically, using two lines, Serial Data (SDA) and Serial Clock (SCL) and the slaves respond only when interrogated by the master, through their unique address.

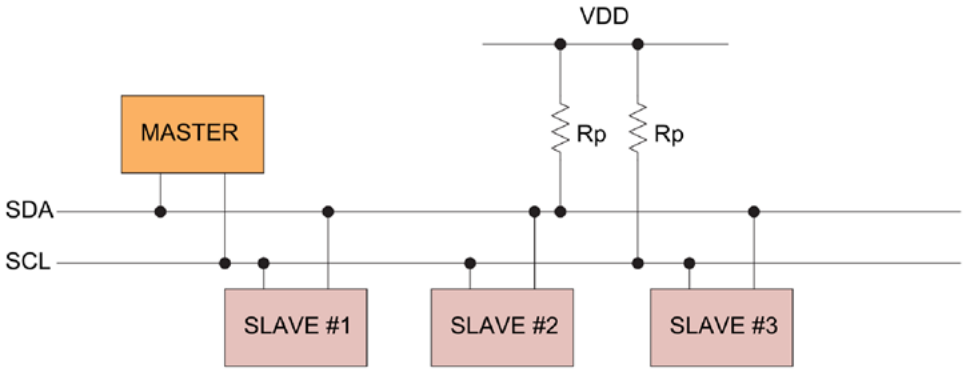
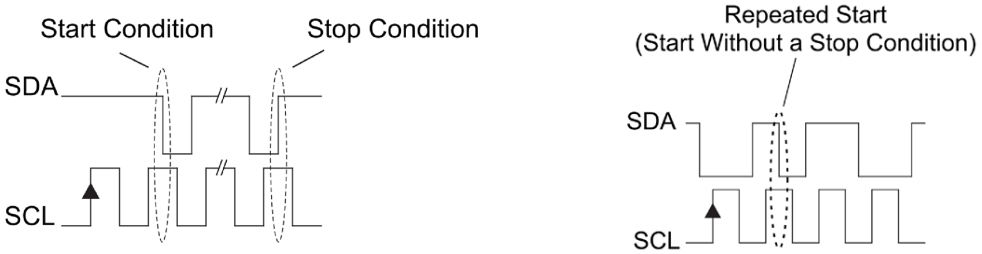


Figure 3.1.1: I^2C connection

In the physical layer, I^2C compatible devices connect to the bus with open collector or open drain pins which pull the line LOW, for writing a LOW bit into the bus, or leave the line HIGH for writing a HIGH bit. Electrically, this is configuration is made possible by pulling the communication lines to the supply voltage using pull up resistors. Bytes are clocked on falling clock edges.

I^2C supports 7, 8 or 10-bit addressing modes, allowing for the connection of up to 1023 devices, being address 0 a general call for all devices in the bus. I2C data packets are arranged in 8-bit bytes comprising slave address, register number, and data to be transferred.

I^{2}C transmissions always start with a "START Condition", to wake the slaves up from a sleeping state and end with a "STOP Condition", to tell the slaves to resume sleep, as exemplified in figure 3.1.2. These are characterized by changing the SDA line while SCL is high. When a Start condition is repeated during a transmission without the need for first terminating with a Stop condition, it's known as a "Repeated Start" and it can used to, for example, change data transmission direction or repeat transmission attempts.



(a) Start Condition And Stop Condition

(b) Repeated Start Condition

Figure 3.1.2: I^{2}C transition delimiter conditions

Data bits encode the actual transmission data and are transmitted in 8-bit byte format, starting with the MSB, and each bit is synchronized with SCL. Each byte must be followed by an Acknowledge (ACK) bit which is generated by the recipient of the data.

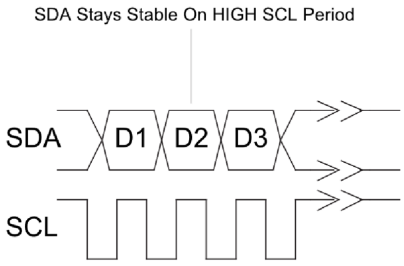


Figure 3.1.3: I^{2}C Bit Transition of Data Bits

When writing to or reading from a specific register in a SLAVE, the master must first point to the specific register by writing the register address once the SLAVE has been addressed. While the register address can be considered a data byte, to avoid confusion it is often classified as a Command byte.

S	ADDRESS	$\overline{W}$	A	COMMAND	A	S	ADDRESS	R	A	DATA	$\overline{A}$	P
	1 1 0 a3:a0	0	0	X X b5:b0	0		1 1 0 a3:a0	1	0	b7:b0	1	

Figure 3.1.4: I^{2}C Example Byte Read Transaction

SPI

Serial Peripheral Interface (SPI) is a synchronous, full duplex master-slave-based serial communication interface. It is widely used in microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, and others. This section will focus on the 4-wire interface of SPI, with separate Master-In Slave-Out (MISO) and Master-Out Slave-In (MOSI) wires, as opposed to the 3-wire interface [17], which will not be used in this project. This scheme is exemplified in figure 3.1.5.

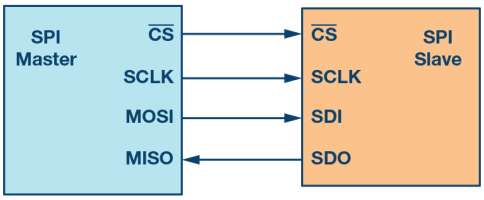


Figure 3.1.5: Single slave SPI connection

4-wire SPI devices have four signals:

1. **Clock (SCLK)**, controlled by the master to introduce synchrony to the communication process;
2. **Chip Select (CS)**, set by the master to select the active slave. In a multiple slave configuration, each slave should have its own CS wire;
3. **Master-Out Slave-In (MOSI)**, the data line that carries information from the master to the slave;
4. **Master-In Slave-Out (MISO)**, the data line that carries information from the slave to the master.

SPI has 4 transmission modes, characterized by the clock polarity and phase [17]. In this project's specific case, mode 3 will be used as it's the mode that is specified in the datasheet of nRF24L01+. It is schematized in the diagram in figure 3.1.6:

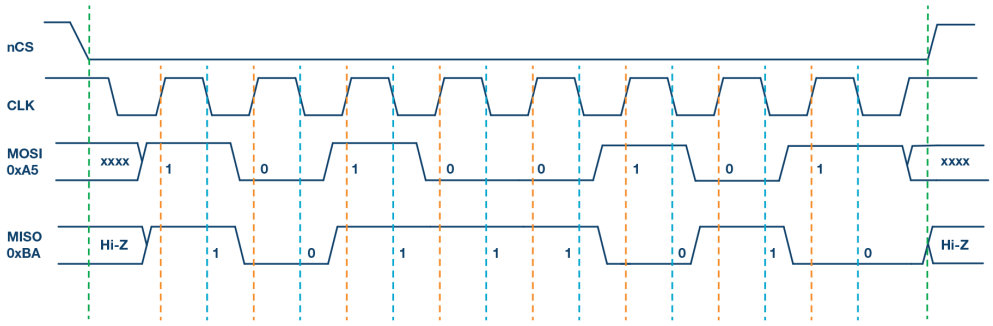


Figure 3.1.6: SPI Mode 3 Diagram

In this mode, the **clock polarity is** 1, which means that the **idle state of the clock signal is HIGH**. The **clock phase is 0**, which indicates that the data is **sampled on the rising edge** (shown by the orange dotted line) and the data is **shifted on the falling edge** (shown by the dotted blue line) of the clock signal [17].

In figure 3.1.7, is an example of a regular multiple slave SPI configuration.

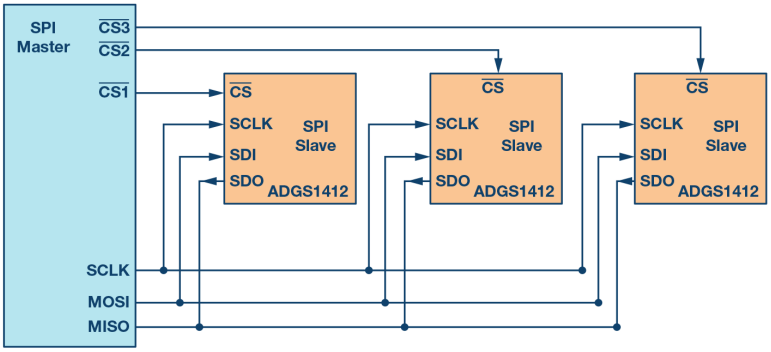
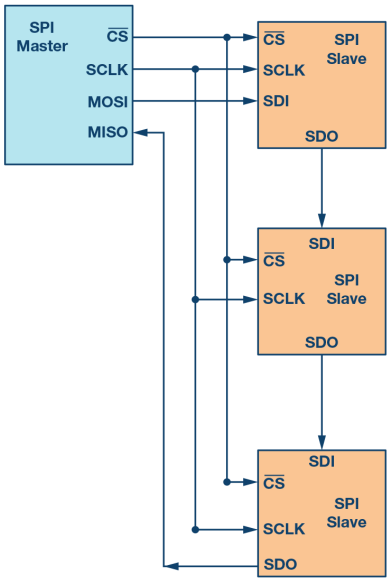
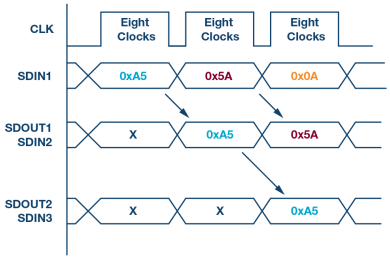


Figure 3.1.7: Multiple slave SPI configuration

Aside from the regular SPI configuration, another one exists in which the master is connected in a daisy-chain configuration to 2 or more slaves, configured such that the chip select signal for all slaves is tied together and data propagates from one slave to the next. This is called a Daisy-Chain configuration and is often used for daisy-chained shift registers and addressable devices.



(a) Multislave SPI daisy-chain connection



(b) Daisy-chain configuration: data propagation

Figure 3.1.8: Multislave SPI daisy-chain configuration

Enhanced Shockburst™

This protocol will be addressed later, since, relating to it, only key aspects matter. The workflow needs to be understood when one is designing the custom hardware.

3.2 Hardware Specification

This section will explore the chosen COTS hardware and other tools for use later in the implementation stage.

3.2.1 Development Board

This project includes a NUCLEO-F767ZI from ST Microelectronics, to speed up the development of the controller for the Local Board. This board includes an STM32F767ZI MCU, an on-board ST-LINK programmer/debugger, and a plethora of on-board peripherals and interfaces for connecting the MCU to external modules.

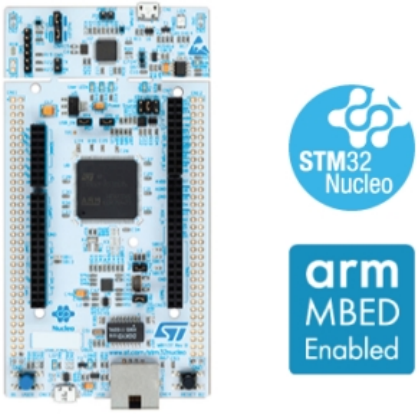


Figure 3.2.1: ST Microelectronics NUCLEO-F767ZI

The most relevant features of the MCU in question, for this project, are:

- 1MB FLASH memory;
- 512KB SRAM;
- 4 I^2C interfaces;
- 6 SPI interfaces;
- 12 general-purpose 16-bit timers.

3.2.2 Gyroscope and Accelerometer

For the measurement of changes in orientation and velocity, the car will incorporate a COTS module with an MPU-6050 accelerometer, gyroscope and Motion Processing Unit (MPU), a voltage regulator that allows the module to be powered by a 5V rail and all the complementary passive components.

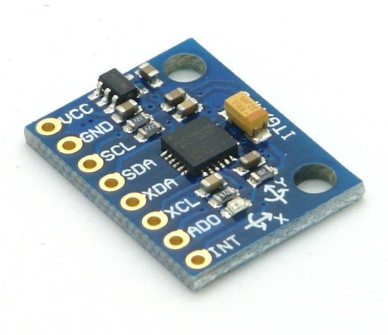


Figure 3.2.2: MPU-6050 module

Some of the features of the MPU-6050 IC that make it suitable for this project are:

- **Three-axis MEMS rate gyroscope** sensor with 16-bit ADC and signal conditioning;
- **Three-axis MEMS accelerometer** sensor with 16-bit ADCs and signal conditioning;
- **Digital Motion Processor™ (DMP)** engine for offloading computation of motion processing algorithms from the host processor;
- **Simple register-based I/O control system** system that can be undirectly accessed for reading/writing through an **I^2C interface**.

3.2.3 RF Communication Module

For communicating wirelessly with low power consumption, the component of choice was the nRF24L01+ RF transceiver IC by Nordic Semiconductor. This product communicates in the 2.4 GHz ISM band, allowing the car to communicate wirelessly with the wristband using a low amount of power and the RF communication solution to have a low-cost BOM. It uses Nordic's proprietary Enhanced ShockBurst™ protocol in the Data Link Layer.

To simplify the use of that component in this project, an integrated module was chosen, which includes the nRF24L01+ transceiver, as well as a crystal oscillator, a Meandered Inverted-F Antenna (MIFA), and all the necessary passive components for bypassing and antenna instrumentation.

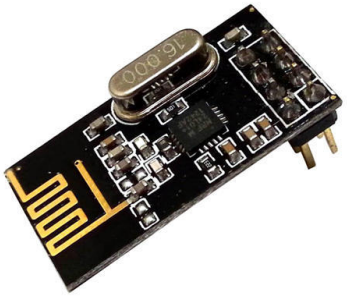


Figure 3.2.3: nRF24L01+ module

3.2.4 Ultrasonic Sensor

For obstacle detection, the ultrasonic sensor was the technology of choice. Besides ultrasonic sensing, IR triangulation was also considered. The reasoning behind the choice of technology is explained in the next paragraphs.

Although cheap ultrasonic sensors present considerable deviation when measuring the distance to objects made of materials like tile and paper, that deviation typically increases with distance and, at shorter distances, these sensors show very satisfactory results. Moreover, for materials like plastic, rubber, wood and cardboard, the technology outperforms distance measurement by IR light in measurement accuracy across all distances. A clear disadvantage of IR trangulation is the fact that sensors based on this technology tend to perform worst in close proximity with the obstacles ($5cm$-$20cm$) [12]. In sum, ultrasonic sensors outperform IR sensors at the measuring the distances expected for this application, to the most common day-to-day materials that the car may find along its way.

In addition to this, because they don't rely on infrared light, ultrasonic sensors are also virtually insensitive to factors that hinder the passage of light, such as dust, smoke and mist, as well as to interference by sunlight. All of these advantages to the this technology for this specific application outshine that fact that it is slightly more expensive than IR sensing.

The ultrasonic sensor module of choice was the HC-SR04, which includes a sensor comprised of a separate transmitter and receiver, which is rated for distances from $2cm$ to $450cm$ and has a claimed accuracy of 0.3cm.

Figure 3.2.4: HC-SR04

For generating a trigger signal for this sensor, a $5V$ pulse with a duration of at least $10\mu s$ is required. This creates the need for converting the $3.3V$ logic level of the GPIO in the STM32F767ZI MCU to $5V$, and the $5V$ level of the output of the module to $3.3V$, securely. For this purpose, a simple and inexpensive module based on a set of 4 BSS138 N-type MOSFETs with gate discharge resistors can be used.

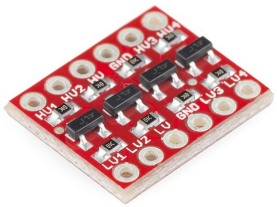


Figure 3.2.5: Quad-BSS138 module

3.2.5 Motor Driver

For driving the motor wheels, the car will include a motor driver module based on the TB6612FNG dual motor driver IC by Toshiba, which includes features such as:

- Integrated PWM logic input pin under-voltage and over-voltage protection;
- Integrated motor control logic;
- Integrated H-Bridge for each motor to be controlled;
- Standby mode.

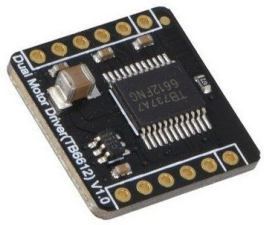


Figure 3.2.6: TB6612 module

The board itself has some advantages, such as simplifying connections, logically and physically, but also symplifying mode selection for the motor control logic. The only disadvantage found was the inexistence of an exposed interface in the board for controlling the standby mode of the IC.

3.2.6 Power Supply

The power supply for the car needs to allow it to be mobile and never present an obstacle to its movement. It also needs to allow for low operational cost without losing its practicality. With that in mind, the decision was made for the car to be battery-fed and rechargeable, an the objective in this section is to find the best tradeoff between the cost of the materials on the total budget and the practicality of the solution.

Battery

The batteries in the power supply for the car need to strike a good balance between size, energy storage capacity and price. For that, a pack of 2 rechargeable Li-Ion 18650 batteries in series will be used. This solution will allow for a supply voltage range of $6V$ at full-discharge to $8.4V$ at full-charge, allowing the car to supply the motors close to the maximum rated voltage.

Figure 3.2.7: 2 pack of 18650 batteries

BMS

For protection and management of any battery pack during charge and discharge cycles, a dedicated Battery Management System (BMS) is required. To accelerate the development of the project, a COTS solution shall be used. With this in mind, the car will include a WH-2S80A board, which is designed to manage 2 cells in series and includes some features that make it suitable for this project:

- Low, $6\mu A$ Quiescent Current;
- $4A$ maximum sustained discharge current, well over the current needed to drive the motors;
- Overcharge protection ($4.29V$);
- Over-discharge protection ($3.00V$);
- Over-current protection;
- Short-circuit protection;

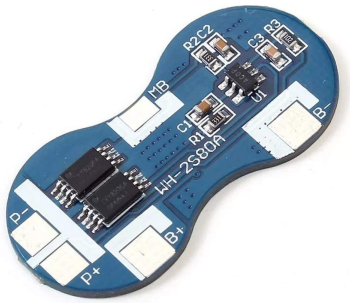


Figure 3.2.8: WH-2S80A BMS module

Step-down DC/DC converter

In order to fix the output voltage of the power supply in a constant value that can be used by the motor as well as the NUCLEO board and all the peripherals, a step-down converter will be needed, since the output voltage of the BMS will vary from approximately $6V$ to $8.4V$. For that the choice was a simple variable voltage switching regulator. This will allow the overall solution to keep a high value of efficiency. For this project, the chosen solution was a generic board based on the LM2596 DC/DC converter IC.

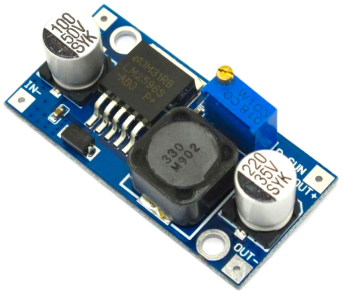


Figure 3.2.9: LM2596 module

Overview

Figure 3.2.10 shows an overview of the power supply module with all the aforementioned components and their intended connections.

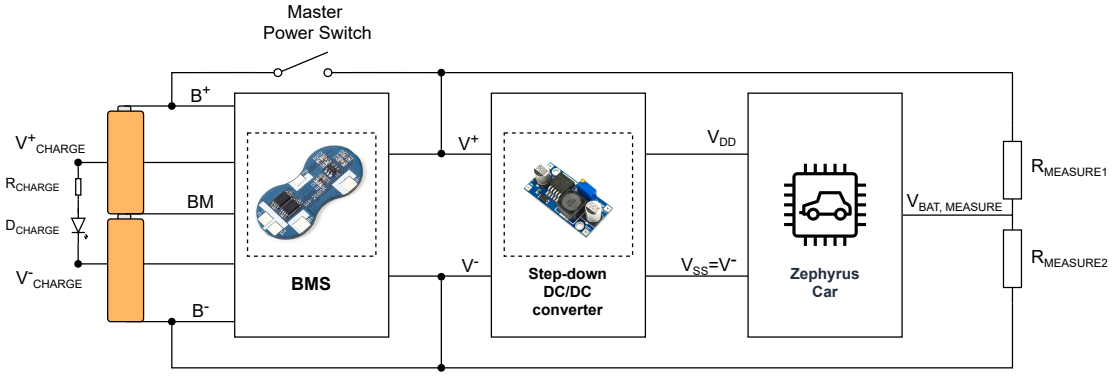


Figure 3.2.10: Power Supply Overview

Aside from the previously explored modules and their direct connections, one can identify in the diagram a series of signals/components:

- V^+_{CHARGE} and V^-_{CHARGE}, which represent the electric potential at the charger input pins. According to the WS-2S80A BMS module's specification, V^+_{CHARGE} and V^-_{CHARGE} should be such that $9 \leq V^+_{CHARGE} - V^-_{CHARGE} \geq 12$;
- D^+_{CHARGE}, which works to indicate that the charger is plugged in, since the BMS module lacks one. R^+_{CHARGE} should be such that it can protect the LED at any voltage inside the allowed range of $V^+_{CHARGE} - V^-_{CHARGE}$;
- $R^+_{MEASURE1}$ and $R^-_{MEASURE2}$, which divide $V^+ - V^-$ into $V_{BAT,MEASURE}$, a value that can be safely connected to the microcontroller's analog inputs.

3.2.7 Remote Power Supply

The power supply for the wristband should be designed to follow the portable, compact and low power design philosophy of the rest of the system. In this section, the aim is to find te best balance between the material cost, usability and general portability of the power supply system.

Battery

In an effort to make the wristband in Zephyrus as light and compact as possible while compromising as little as possible on low BOM costs, the decision was made to have it be fed by a Lithium Coin cell.

As small-size and low-power RF communication devices become more and more efficient, small and cheap Lithium coin cells start to become more viable, serving as a good alternative to rechargeable batteries, since they imply less of an initial monetary cost for themselves and their charge/protection circuitry. The downside here is, of course, that they need to be replaced, so systems that experience long usage times may refrain from using these devices altogether. However, in a low-power system such as Zephyrus, that remains turned off entirely for most of the time, this is less of a problem.

Aside from being one of the most popular models of Lithium Coin cells, produced by all major brands, CR2032 batteries also have among the highest energy capacity ratings[14] of all coin cells. Their size is also a good fit for a wristband, and so that will be the supported battery type.

Figure 3.2.11: Typical CR2032 coin cell

Internal coin cells' relatively high internal impedance, low energy capacity and the fact that they will not be recharged mean that they do not need charge/protection circuitry where it can be left out, such as in Zephyrus' wristband. They do, however, suffer from high drop in voltage when exposed to higher current draws, such as the ones RF devices produce intermittently while sending or receiving data. To mitigate this issue, a small bypass capacitor can be placed parallel to the cell, to allow it to supply larger amounts of current in short bursts without degradation or noticeable voltage drops.

Power Switch

In order to turn the wristband's power on and off, Zephyrus implements a push button power switch. This is an alternative to the typical toggle switch, as found in the car and it's meant to give the product a modern appeal, as well as delegate as much power as possible to the processor when turning off, giving it time to terminate tasks and close previously open connections. Another benefit of this is allowing the product to turn itself down when not in use for long, as it helps to achieve even greater power savings. For this, the solution needs to have as low $I_{Quiescent}$ as possible.

Although there are a few integrated solutions on the market, none quite satisfy the specific functionality, space, energy and cost-constrained needs of the project, and as such a custom solution is needed. This is indeed the route that was taken, as a simple, cost-efficient and compact solution was found that solved this problem in just the right way.

This solution is based on the Maxim MAX16150 Pushbutton On/Off Controller and Battery Freshness Seal IC and follows the recommendation in the "Typical Application Circuit" section of the product's own datasheet [15], found in figure 3.2.12.

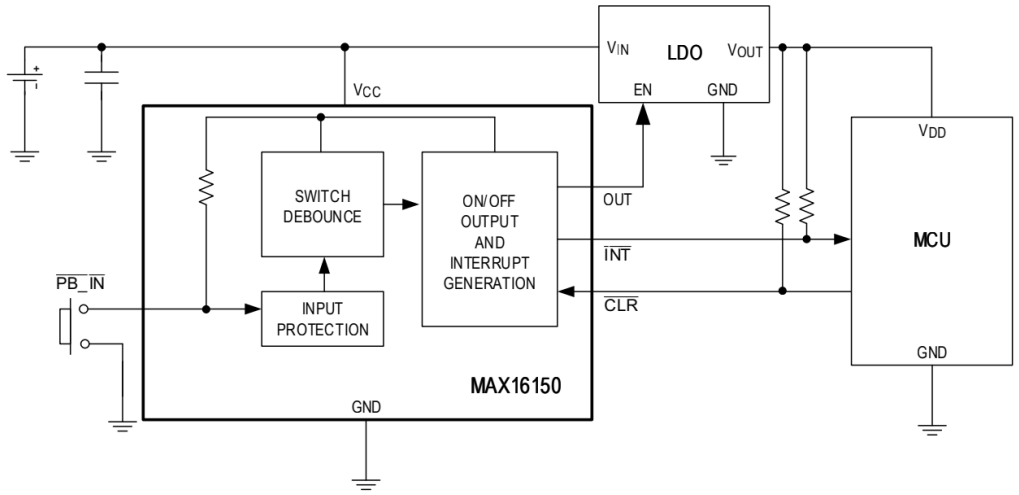


Figure 3.2.12: Typical Application Circuit for MAX16150

In this specific application, the LDO is replaced by an integrated load switch, for adequate load switching, as well as discharging after disabling. Ideally, this will eventually be removed after all optimizations in terms of power management have been implemented, in such a way that the circuit always draws less than the $20mA$ that the push button controller is capable of delivering. Since this is a provisionary solution, and in an effort to save time and money during development, the same integrated circuit for load switching will be used in this project and in the parallel project of the Embedded Systems special-

ization. This should help in streamlining the development of the power switch for both of them. As the other project's power demands are far more restrictive than Zephyrus's, it will dictate the power delivery capabilities of the load switch, and the most important decisions about it will be explained there. The chosen solution is based on the Diodes Incorporated AP22814 Single Channel Power Distribution Load Switch [16].

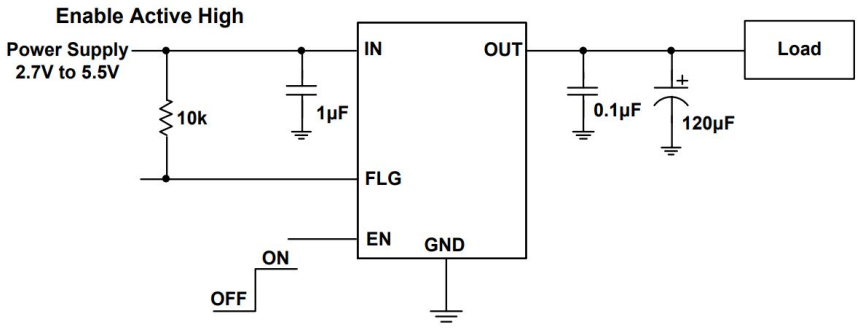


Figure 3.2.13: Typical Application Circuit for AP22814

3.3 Peripheral Interface

This section will explore the meaningful connections that will be made in sequence to hardware explored in the Hardware Overview, that haven't been explored in that section already.

3.3.1 Gyroscope and Accelerometer

Apart from the power supply lines, the most important connections on this board are:

- The **I²C interface** pins, SCL and SDA;
- The $AD0$ pin, which is an eighth address pin on the MPU-60x0 family of devices, that allows for connecting two of such devices in the same bus. This will be connected to ground;
- The INT pin, that serves to notify the MCU of the occurrence of one or more of the events that trigger the various configurable interrupts.

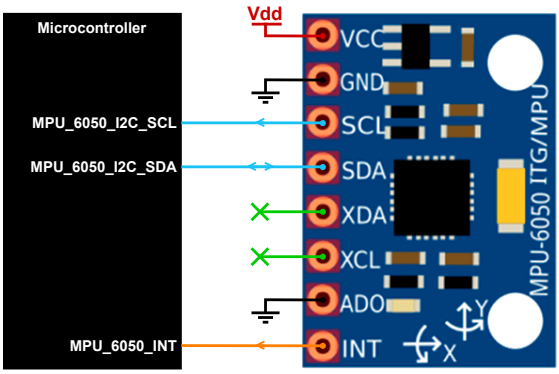


Figure 3.3.1: Gyroscope and Accelerometer Interface

3.3.2 RF Communication Module

The most important connections on the RF Communication Module are:

- The **SPI interface** pins, *MOSI*, *MISO*, *SCK* and *CSN*. The latter is the Chip-select pin will serve to put the device in or take the device of Stand-by Mode I;
- The *INT* pin, that serves to notify the MCU of the occurrence of one or more of the events that trigger the various configurable interrupts;
- The *CE* pin, which activates RX/TX mode when high.

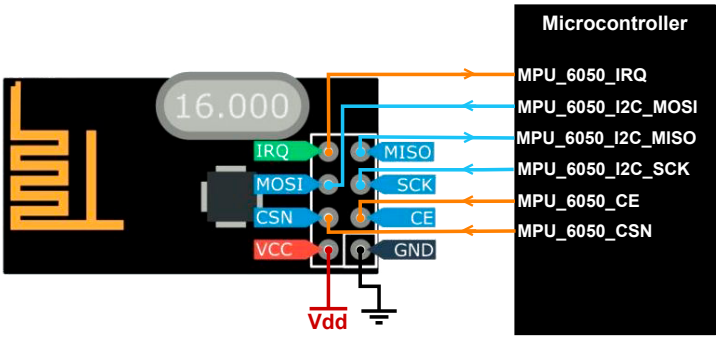


Figure 3.3.2: Gyroscope and Accelerometer Interface

3.3.3 Ultrasonic Sensor

The most important connections on the Ultrasonic Sensor module are:

- *Trig*, which can be driven driven high for more than $10\mu s$ will stimulate the module to start ranging.

- *Echo*, which after starting a ranging cycle will stay high for a period of time such that the distance in meters to the closest object is determined by:

$$d = \frac{T_{EchoHigh} \times v_{sound}}{2} \tag{3.1}$$

where $v_{sound} = 340m/s$.

The level shifter module intermediates the connection between the MCU and the ultrasonic sensor. It is worth mentioning that rise and fall times in the transistors may affect the timing in the distance measurement by introducing a delay in both signals and should thus be taken into consideration when implementing the system.

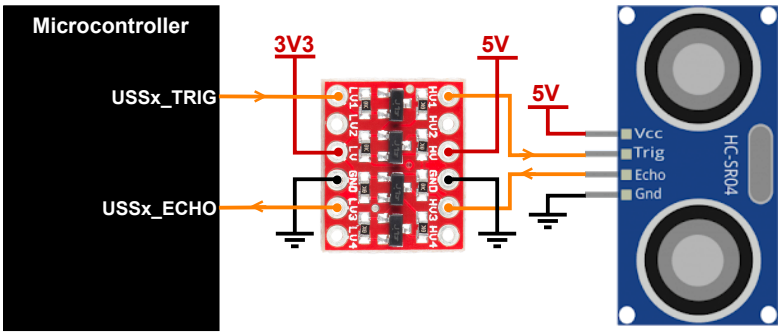


Figure 3.3.3: Ultrasonic Sensor Interface

3.3.4 Motor Driver

Apart from the power supply lines, the most important connections on this board are:

- $PWM1$ and $PWM2$, which allow for controlling the speed at which each motor rotates;
- $DIR1$ and $DIR2$, which allow for controlling the direction in which each motor rotates (HIGH for CW rotation and LOW for CCW rotation [20]);
- $M1+$, $M1-$, $M2+$, $M2-$, which drive the motors according to the previously established inputs.

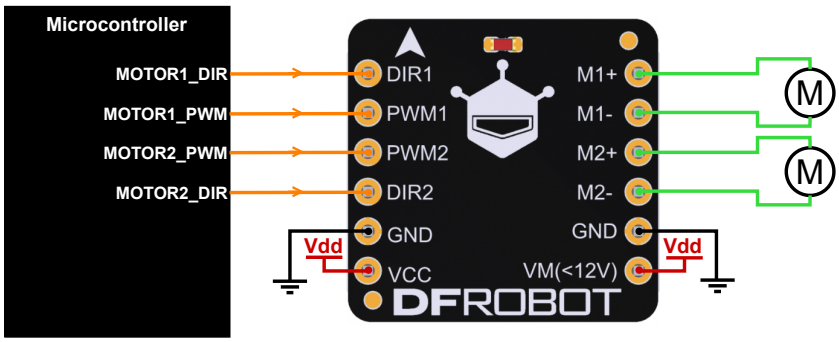


Figure 3.3.4: Motor Driver Interface

3.3.5 Push Button Power Switch

The MAX16150 IC's interface with the user and the MCU is done through the following connections:

- $\overline{PB\_IN}$, which signals a button press when driven LOW;
- $\overline{INT}$, which must be passively pulled HIGH and signals to the MCU that the button has been pressed when actively pulled LOW;
- $\overline{CLR}$, which must be passively pulled HIGH and signals to the MAX16150 that the MCU is ready to shut down;
- OUT, which is the means by which the IC controls the power supply of the MCU.

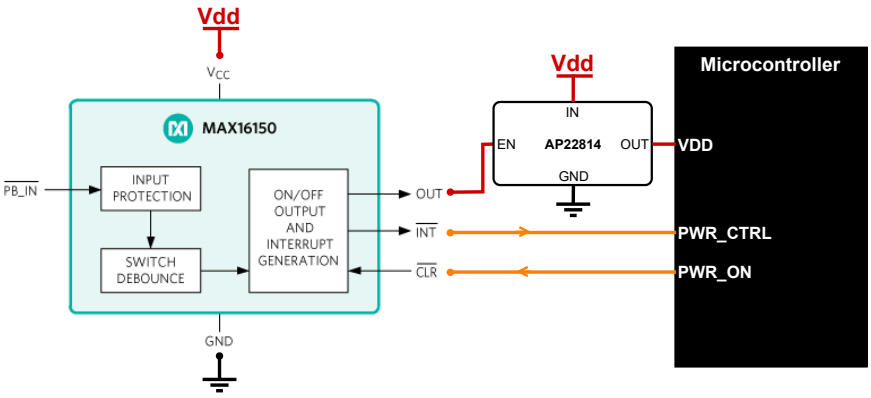


Figure 3.3.5: Push Button Power Switch Interface

3.4 Tools and COTS

The software tools and COTS used for the project are described in section 3.4.1 and section 3.4.2, respectively.

3.4.1 Tools Summary

- **C** - Programming language chosen for both systems;
- **STM32CubeMX** - Configuring tool for the STM boards;
- **X-CUBE-AI** - STM32CubeMX extension for Machine Learning applications;
- **Keil μVision5** - Programming IDE used for the STM Boards;
- **Eagle CAD** - IDE utilized for PCB design and layout;
- **TensorFlow Lite** - Tensorflow for microcontrollers to deploy the model into the local system;
- **Autodesk Fusion 360** - CAD tool used for designing the PCBs' 3D model;
- **Jupyter** - Web application that allows the creation and sharing of documents that contain live code;
- **Anaconda** - Tool utilized in this project for creating virtual environments for running Jupyter Notebooks;

3.4.2 COTS Summary

- **FreeRTOS** - Real-time Kernel on top of which embedded applications can be built.
- **Keras** - TensorFlow wrapper API used for designing Zephyrus' Neural Network;
- **TensorFlow** - Machine Learning API used for designing Zephyrus' Neural Network;
- **HAL** - STM32 abstraction layer for embedded software that ensures maximized portability across the STM32 portfolio;

3.5 System Tasks

The use of Real-time Operating System (RTOS) is fairly common in embedded software designs, as it allows code division into smaller blocks, tasks, which execute seemingly in parallel.

Such designs usually implement preemptive multi-tasking, which demands for *a priori* priority definition, when following an Fixed-Priority Scheduling (FPS) (Zephyus case). With that in mind, one must establish the main system tasks, their priorities, how they communicate with each other, and design the intended workflow for each individual task through flowcharts.

3.5.1 Task List

The task list for the **Local System** is introduced below:

- **zMain** - local system main;
- **zGyroAccelerometerManager** - gyroscope and accelerometer module communication and management task;
- **zRFManager** - RF module management task;
- **zInferenceManager** - Reinforcement Learning inference management task;
- **zUltrasonicManager** - ultrasonic sensor module communication and management task;

The task list for the **Remote System** is introduced below:

- **zMain** - remote system main;
- **zGyroAccelerometerManager** - gyroscope and accelerometer module communication and management task;
- **zRFManager** - RF module management task;

3.5.2 Task Priority Level Assignment

As foresaid, one needs to assign each task a static priority level to indicate their relative urgency. Moreover, the scheduler will always pick the task that is ready to execute with the highest priority level. Note that an unsuitable assignment of task priorities might result in system performance loss and unresponsiveness, since the processor could be overtaken by a single high hierarchy task. Considering this, the thought out priority assignment diagrams are represented in figures 3.5.1 and 3.5.2.

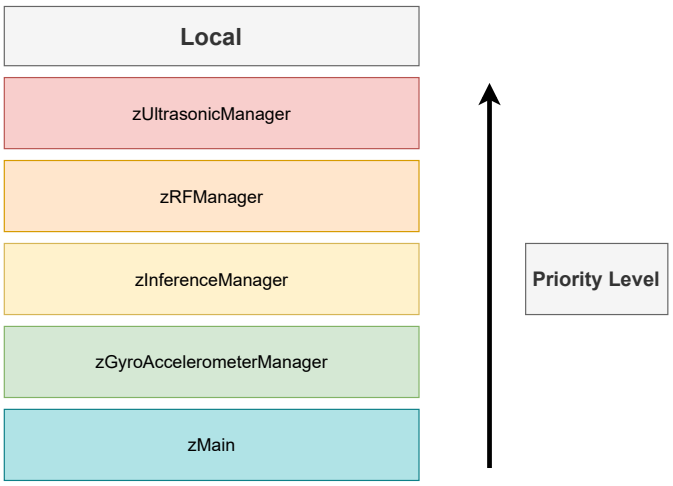


Figure 3.5.1: Priority Assignment Schematic - Local

The priority assignment in the Local System prioritizes tasks that are quick to execute and have strict timing requirements. This means that although more important tasks will still be executed in a logical and timely manner, they may have to give in to other tasks that take samples they need synchronously.

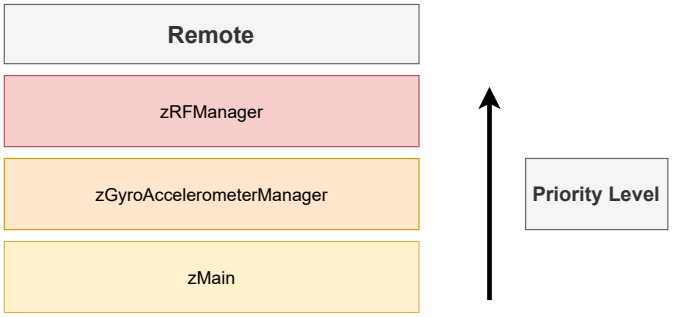


Figure 3.5.2: Priority Assignment Schematic - Remote

The priority assignment in the Remote System prioritizes communication since sampling is done independently by external modules.

3.5.3 Local Task Timeline

The task timeline for the local system is represented in figure 3.5.3.

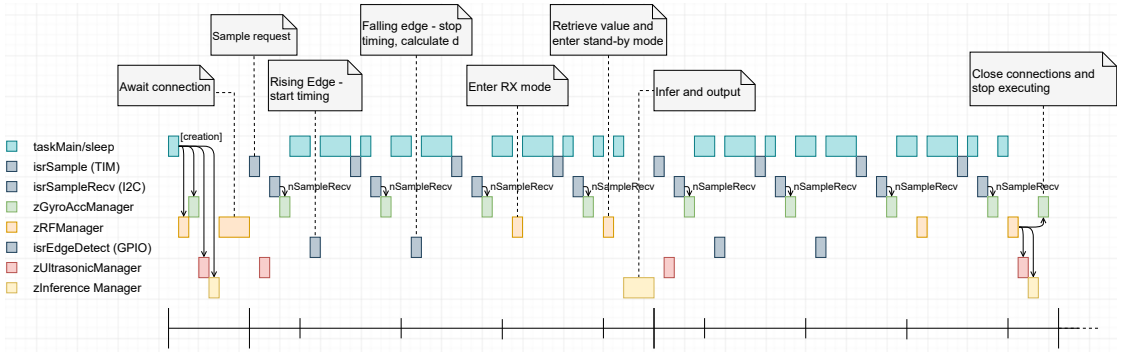


Figure 3.5.3: Local Task Timeline (Aug. in Appendix C.1)

3.5.4 Remote Task Timeline

The task timeline for the remote system is represented in figure 3.5.4.

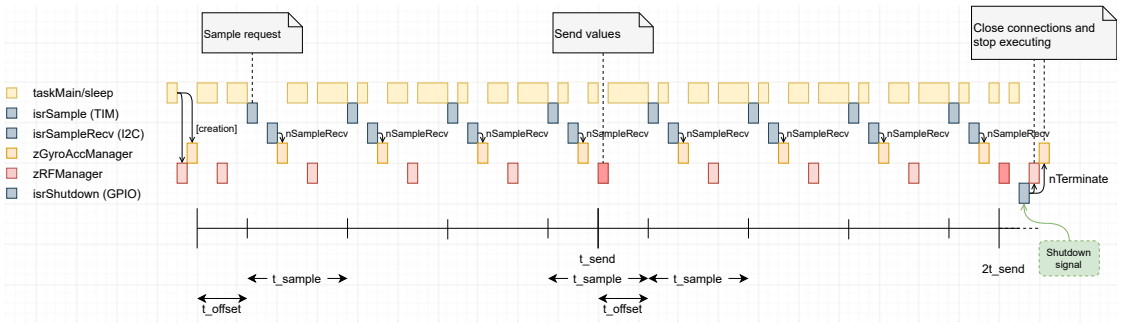


Figure 3.5.4: Remote Task Timeline (Aug. in Appendix C.2)

3.5.5 Task Communications

The global task communications schematic is illustrated in figure 3.5.5.

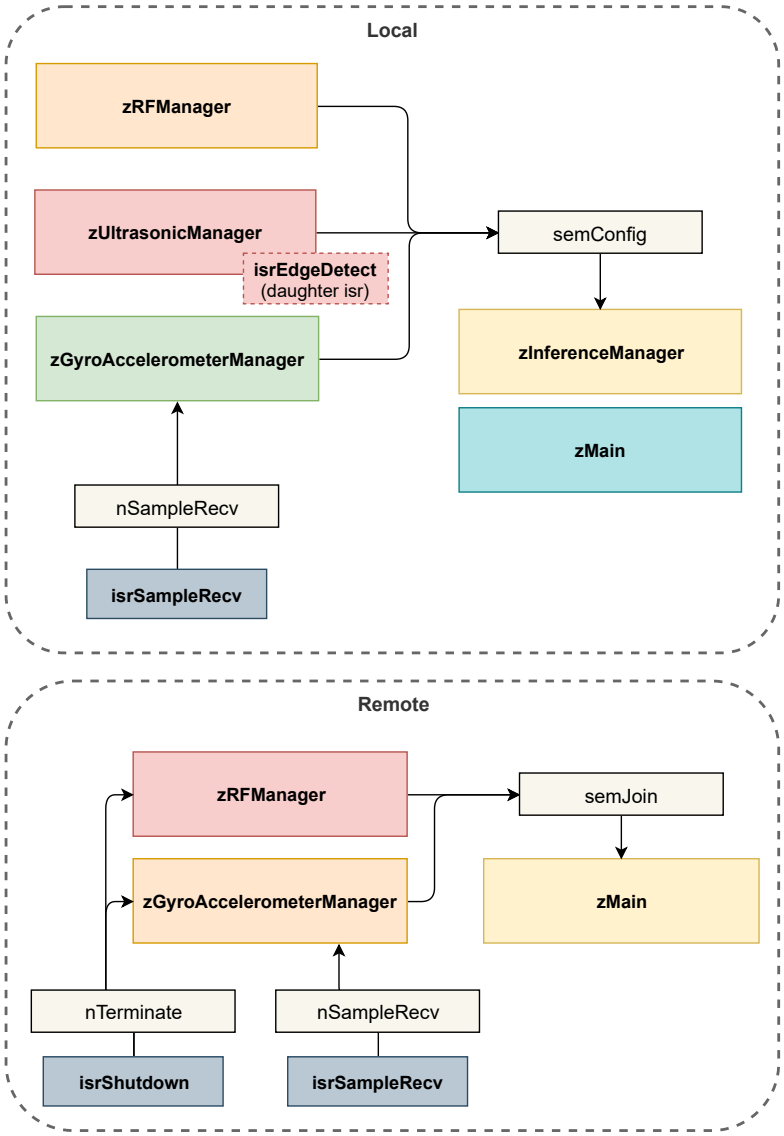


Figure 3.5.5: Initial Task Communications Schematic

3.6 Local System: Software Specification

In the Software Specification phase, each software module must be identified and defined. Note that one will only focus on the more complex tasks and the ones where commentary is necessary.

3.6.1 zGyroAccelerometerManager

The flowchart for the loacl gyroscope and accelerometer module Task is illustrated in figure 3.6.2, based on the predefined state diagram of figure 2.6.3.

Startup

The task must initialize I^2C and configure the necessary registers for the Data Ready detection interrupt, which can be identified in figure 3.6.1 [13]. The configuration of the latter will be part of the flowchart of this section in an implicit way under the hardware configuration block. After, the task should wait for the remote's connection signal and decrement the semConfig semaphore to indicate that it is ready to start exchanging messages.

Interrupt Name	Module
Free Fall Detection	Free Fall
Motion Detection	Motion
Zero Motion Detection	Zero Motion
FIFO Overflow	FIFO
Data Ready	Sensor Registers
I^2C Master errors: Lost Arbitration, NACKs	I^2C Master
I^2C Slave 4	I^2C Master

Figure 3.6.1: MPU-6050 Interrupt Table

Execution

Initially, the timer interrupt should check if the predefined sample period elapsed, and request a sample to the sensors, sending a notification (nSampleRecv) to the zGyroAccManager task. Upon a new data scenario, it should store the values of the sensors and if the predetermined number of oversamples per sample was reached the task will calculate the mean with the latter. The mean value will be processed to attain the state variables of the system. With the mState mutex, one will be able to control the access to the state data shared with the inference task. In the scenario in which there isn't any new data, the task should check for timeout and increment the timeouts counter accordingly. At every decision point, the task will be continuously checking for the terminate signal,

which indicates whether it should increment the aforementioned semConfig semaphore and return to a waiting state or not.

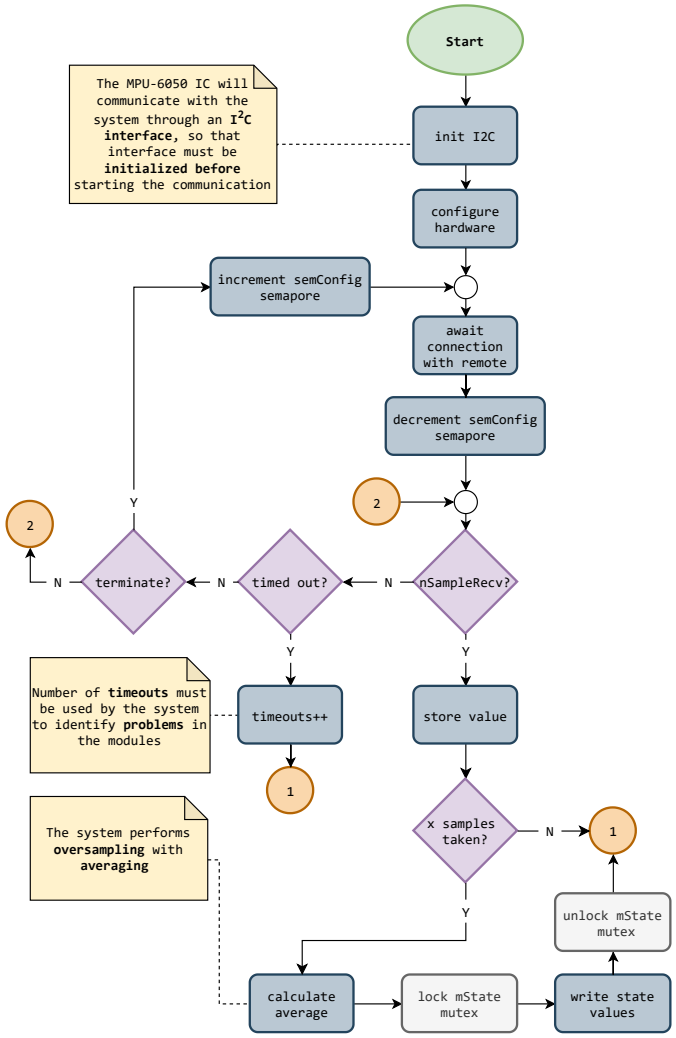


Figure 3.6.2: zGyroAccelerometerManager Task Flowchart

3.6.2 zRFManager

The flowchart for the local zRFManage Task is depicted in figure 3.6.3.

Startup and Execution

At startup, the zRFManager task first configures the SPI and GPIO peripherals to then perform the configuration of the physical RF Module, selecting parameters such as auto-acknowledgement, auto re-transmit delay and count and radio power. After receiving a connection request and going through a handshake process, it gives the appropriate signal enters the continuous execution state.

During this stage, the task continuously polls for the reception of information, upon which it decodes the message and redirects it to the appropriate receptor. In the case of it being a request for terminating the connection the task returns to a state in which it is awaiting a connection request.

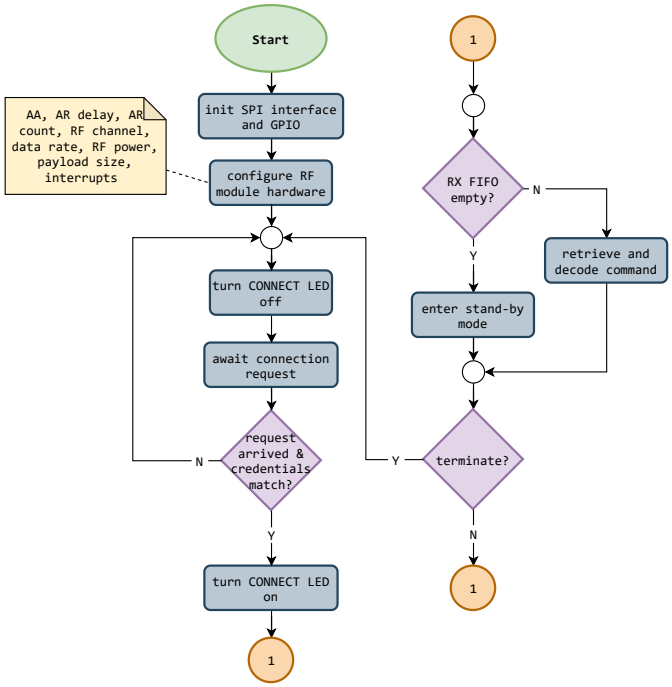


Figure 3.6.3: zRFManager Task Flowchart

3.6.3 zInferenceManager

The inference task will be responsible for preparing the field for the inference to take place and for running the latter on an adequate time. Considering this, this task will first create an instance of the Neural Network (NN) and make some configurations that facilitate the interaction with the NN. The task will use a downwards semaphore that will indicate that all the tasks performed their configurations when its value is zero. Moreover, it will perform all the necessary data accesses to read and write values concerning its inputs/outputs, respectively, in shared areas. The task will also have error handling capabilities and will be reactive to the terminate signal issued when one presses the Remote System's power off button.

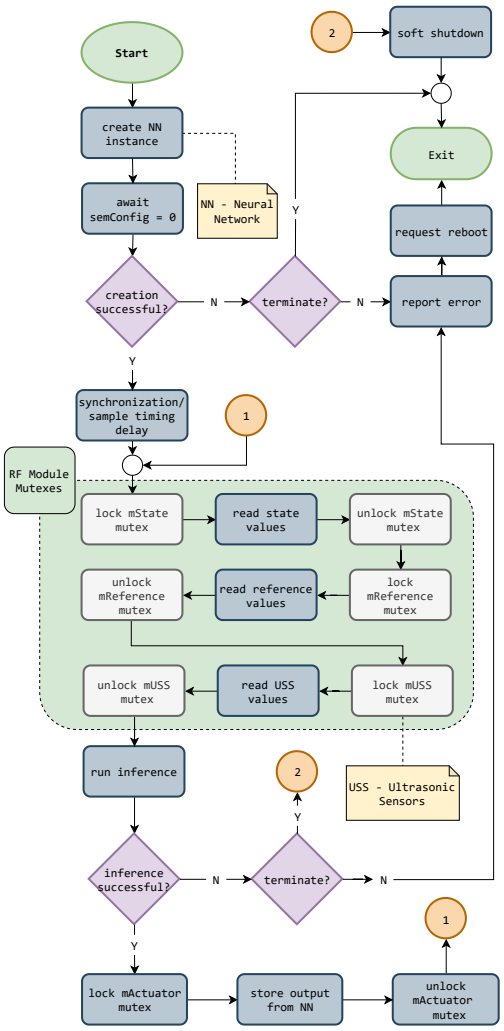


Figure 3.6.4: zInferenceManager Task Flowchart

3.7 Reinforcement Learning

The initial design of the project's Neural Network that will serve as the controller for the car is represented in figure 3.7.1.

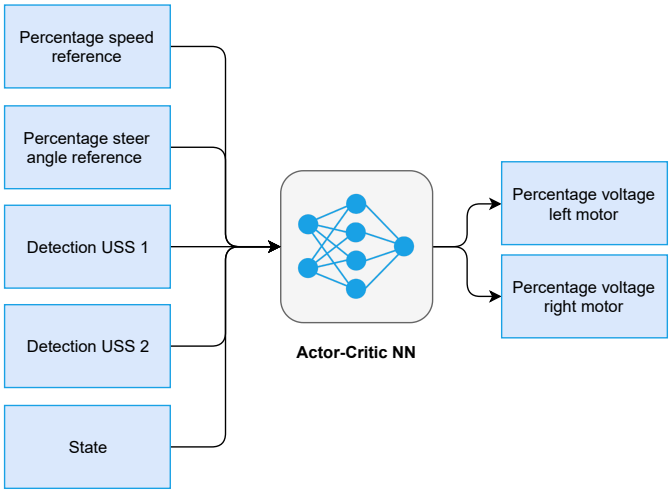


Figure 3.7.1: Zephyrus Feed-Forward Neural Network (FFNN)

The neural network will receive as inputs a percentage of total speed and steer angle, as well as a value representing the detections of objects by the front and back ultrasonic sensors (USS). Since one is using reinforcement learning techniques, the state will also have to be an input of the network. Additionally, the NN will comprise both the actor and the critic as a way to condensate the model in only one file for the deployment stage. The outputs will be percentages of the voltages applied to the left and right motors to allow portability to loads that require different working voltages ($\neq 6V$).

With this in mind, the training of the agent will require the following steps [19]:

1. Run the agent on the environment to collect training data per episode;
2. Compute expected return at each time step;
3. Compute the loss for the combined actor-critic model;
4. Compute gradients and update network parameters;
5. Repeat 1-4 until either success criterion or max episodes has been reached;

3.7.1 Car Kinematics

In order to achieve a reinforcement learning model using python and Tensorflow, one first needs to devise a physical model of the system to be used later as a way to calculate the new states and rewards. The remote car can be considered as a generic tricycle, and as such, the formulas used reflect that [29].

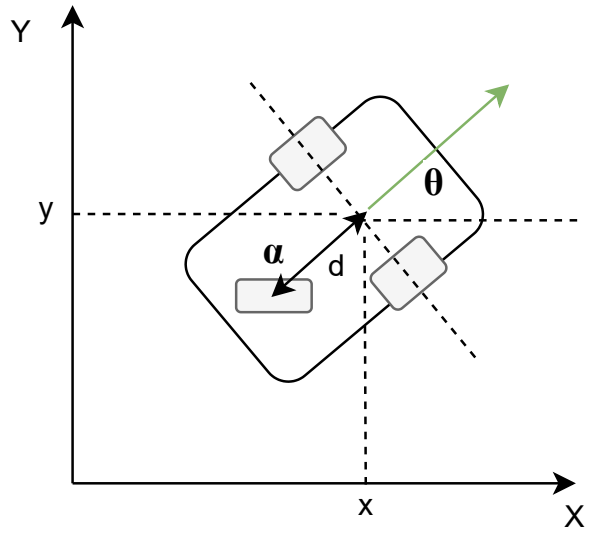


Figure 3.7.2: Kinematic Car Model

$$v_s(t) = w_s(t) * r \tag{3.2}$$

$$w(t) = \frac{v_s(t)}{d} sin\alpha(t) \tag{3.3}$$

Kinematic model in the car frame:

$$v_x(t) = v_s(t) cos\alpha(t) \tag{3.4}$$

$$v_y(t) = 0 \tag{3.5}$$

$$\frac{d\theta}{dt} = \frac{v_s(t)}{d} sin\alpha(t) \tag{3.6}$$

Kinematic model in the world frame:

$$\frac{dx}{dt} = vs(t) cos\alpha(t) cos\theta(t) \tag{3.7}$$

$$\frac{dy}{dt} = vs(t) cos\alpha(t) sin\theta(t) \tag{3.8}$$

$$\frac{d\theta}{dt} = \frac{v_s(t)}{d} sin\alpha(t) \tag{3.9}$$

3.8 Remote System: Prototype Alpha

3.8.1 Custom Hardware

Whether it be in order to accelerate development or materialize the product, allowing it to reach its full potential, some freedom was taken into designing specific solutions on PCBs. This section explores the details on those designs.

STM32F0 Development Board PCB

In order to develop for a smaller, cheaper and lower-power machine, a simple custom solution was divised. In order to speed up development, an MCU from the same STM32 platform was chosen, as it is a familiar platform. Although there were certainly existing solutions for that specific platform, few allowed for as close exploration of the hardware for later integration into a larger system and understanding of its interfaces as a custom solution would. The divised solution is compatible with all products from the F0 family of the STM32 lineup of products with a QFP-32 pinout. Aside from the I/O, it includes only the most essential components for the microcontroller to boot and run smoothly, such as bypass capacitors and an option to manipulate the BOOT0 pin to boot from any region of memory.

The figures presented below concern the board's dimensions, as well as the layout from the top and bottom perspectives. The development board schematic is represented in figure B.1.

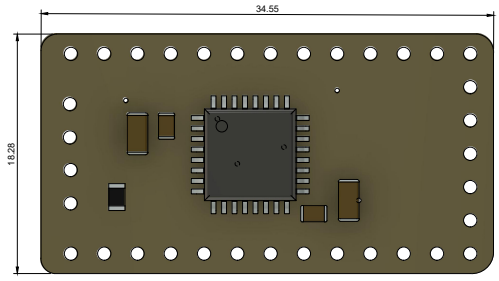
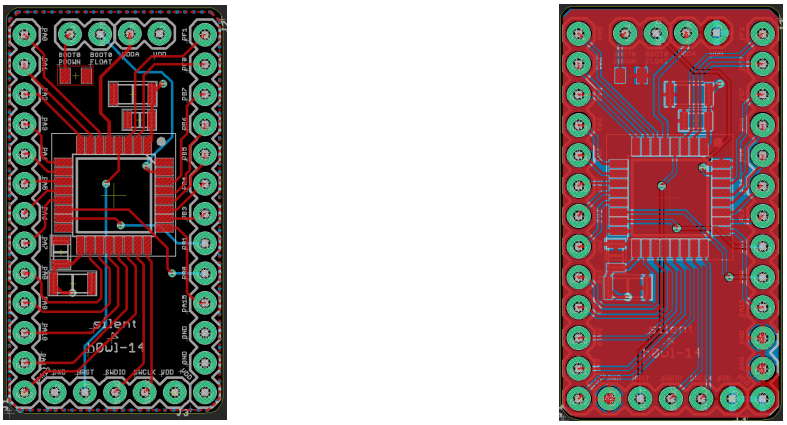
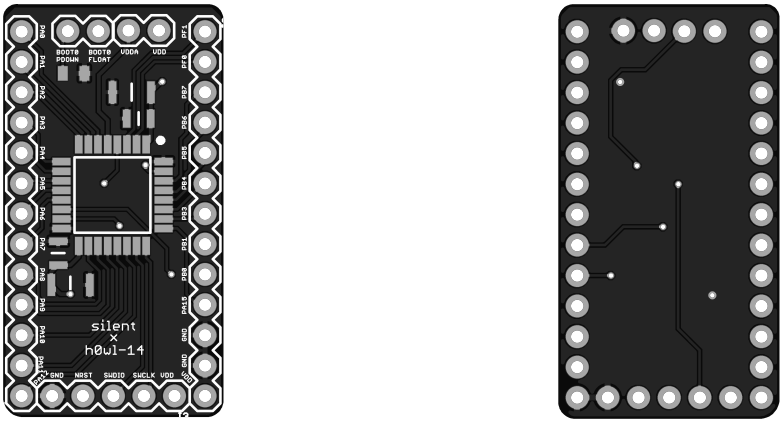


Figure 3.8.1: STM32F0 Development Board - Dimensions (millimeters)



(a) Without GND plane (b) With GND plane

Figure 3.8.2: STM32F0 Development Board - Layout



(a) Top view (b) Bottom view

Figure 3.8.3: STM32F0 Development Board - PCB preview

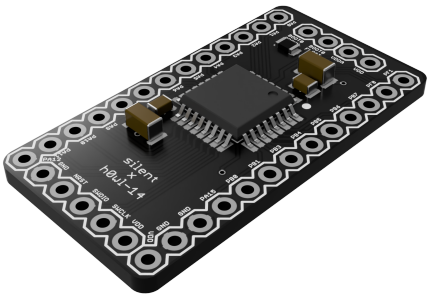


Figure 3.8.4: STM32F0 Development Board - 3D View

Push Button PCB

The power management section of the remote system's design is implemented using two physically separate ICs, of very small SMD packages. It is thus impossible to connect them to a breadboard directly and the number of passive components that it requires means that the module may work inconsistently in the alpha prototype, even when using SMT sockets for the ICs. And most importantly, as mentioned before, the same solution for ON/OFF control is also needed in another project, as a separate module so, implementing it for this one and ordering the components at the same time as those for Zephyrus would mean saving time, money and hassle during development of both projects.

With that in mind, this board contains both the push button ON/OFF controller and the load switch ICs chosen in the hardware design, as well as all the required passive components, a push button and an 0.1" pitch connector that exposes all of the useful power and I/O features of both controllers.

The figures presented below concern the board's dimensions, as well as the layout from the top and bottom perspectives. The board's schematic is represented in B.3

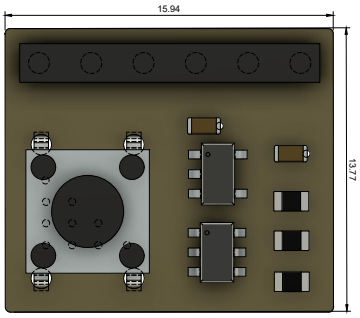


Figure 3.8.5: Push Button Board - Dimensions (millimeters)

(a) Without GND plane

(b) With GND plane

Figure 3.8.6: Push Button Board - Layout

(a) Top view (b) Bottom view

Figure 3.8.7: Push Button Board - PCB preview

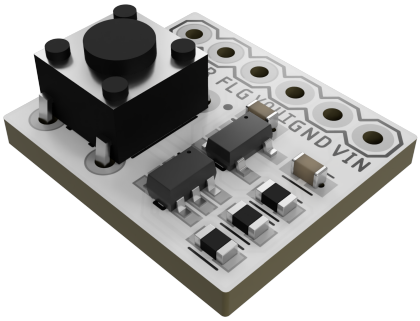


Figure 3.8.8: Push Button Board - 3D View

3.9 Remote System: Prototype Beta

3.9.1 Custom Hardware

One intends to migrate from the remote system's Alpha prototype to one with all the components tightly integrated into the same package to improve reliability and usability, named Beta prototype. However, as the latter depends on the hardware and software implementation and testing of the Alpha, one opted to finish its design at a later stage. For now, an initial schematic was devised. Take note that some of the integrated modules presented in section 3.2 had to be re-engineered to be fully applicable to Zephyrus. The initial design for the wristband PCB, based on the labeling of figure 3.9.1 is represented in B.2.

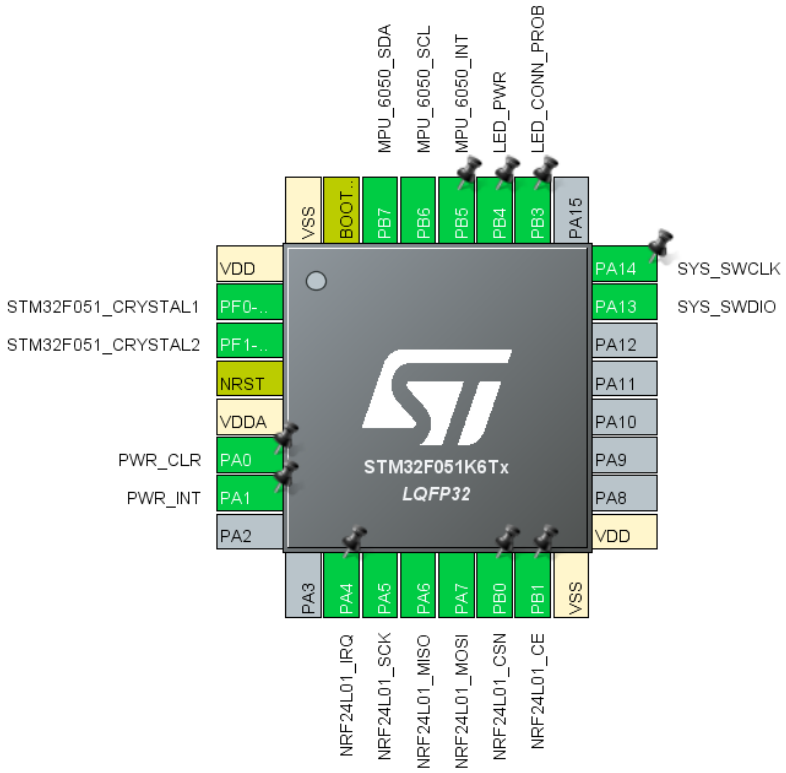


Figure 3.9.1: Prototype Beta: MCU pin net labels (initial)

3.9.2 Wristband PCB

As previously mentioned, the remote system should implemented in a single, watch-sized package, exception made for the gyroscope/accelerometer module, which should extend from the wristband to the finger of the user. With the intent of honoring those choices, the rest of the system was implemented in a single PCB, with two sets of 0.1"-spaced pads, one for programming the on-board STM32F0 microcontroller and another for it to interact with the external MPU-6050 module. Keeping the funcionality present in the Alpha prototype in its entirety was the biggest concern while designing the Beta prototype, although aesthetics and user-friendliness were also kept in mind throughout its design.

The figures presented below concern the board's dimensions, as well as the layout from the top and bottom perspectives. The board's schematic is represented in B.2

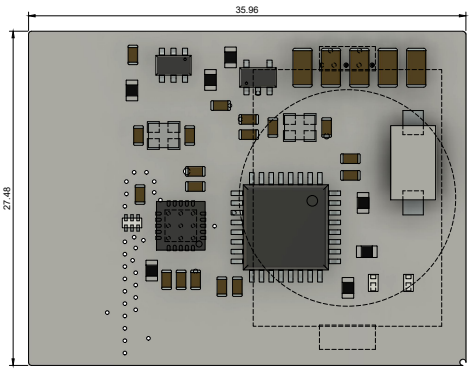
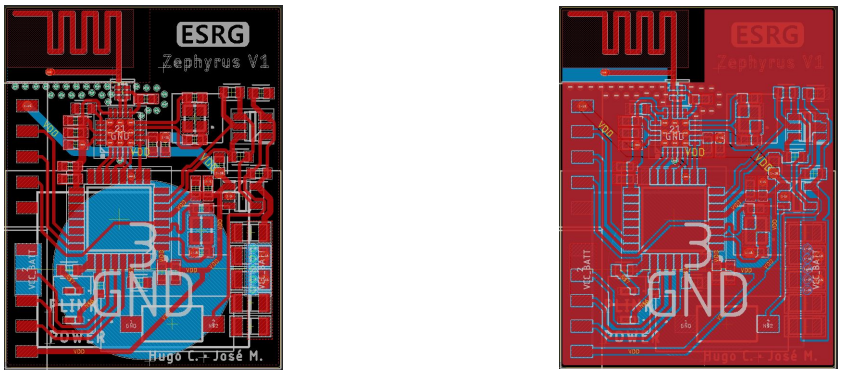


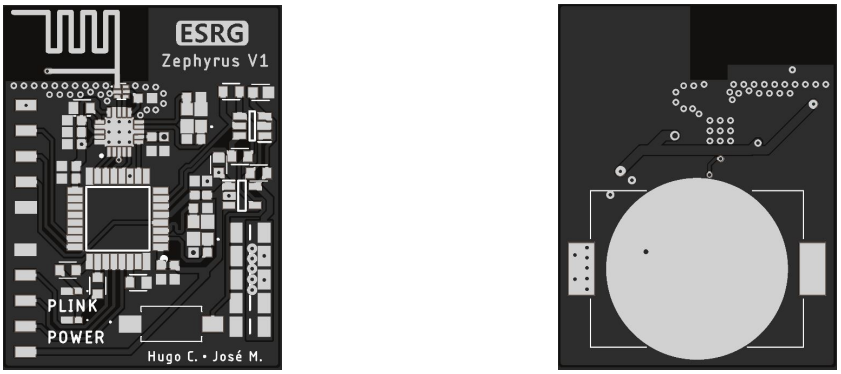
Figure 3.9.2: Wristband Board - Dimensions (millimeters)



(a) Without GND plane

(b) With GND plane

Figure 3.9.3: Wristband Board - Layout



(a) Top view (b) Bottom view

Figure 3.9.4: Wristband Board - PCB preview

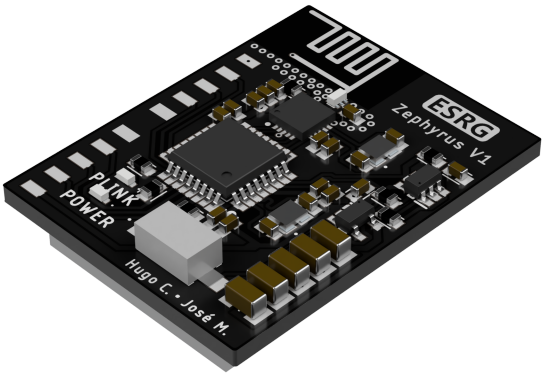


Figure 3.9.5: Wristband Board - 3D View

Sectioning and Design Considerations

The board for Zephyrus' wristband was divided into sections with different sets of hardware components in order to maximize efficiency, minimizing trace length and indirections, leading to less signal propagation problems and granting better noise immunity (figure 3.9.6).

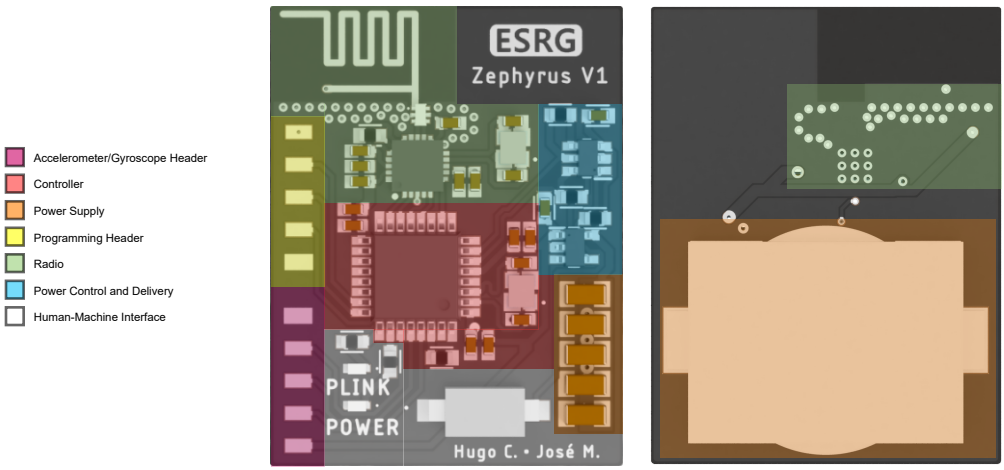


Figure 3.9.6: Prototype Beta Sections

Power Supply The power supply section is comprised by the Lithium-Ion battery that powers the entire system and its bypass capacitors. These are of size 1206 (imperial) and space was guaranteed to accommodate five of them. The size of the capacitors is justified by the uncertainty of their values in the final prototype, as it is hard to tell with the time given to finish the prototype what was the most appropriate value for the bypass capacitance without testing it on the real system. This way, either an 0805 or a 1206-sized capacitor can comfortably be placed in each socket. Five spots have been guaranteed in an effort to maximize the flexibility in terms of the the reduction of the total ESR of the bank.

Power Control and Delivery The Power Control and Delivery section is essentially a transposition of the Push Button design from the Alpha prototype to the Beta prototype. It fits in line with the second part of the Power Supply module to keep the power supply constrained in one tight space along the right side of the board - as far from the Radio as possible. The load switch sub-section was purposefully placed towards the middle of the board to keep the power traces as short as possible. These were also made thicker than the signal traces to reduce losses due to the Joule Effect, although this is a fairly low-power system.

Radio The top most section of the board pertains to the Radio module. This is comprised by the nRF24L01+ IC, the integrated balun and filter network, the antenna and the necessary decoupling/feedback passive components.

The chosen antenna type for this design was the Meandered Inverted-F Antenna (MIFA) type [22]. It was chosen for its small size, very cheap cost and comparatively good performance, which make it a very popular design, being implemented, inclusively, in the COTS module used in the alpha prototype of this project. The specific design

used in this project is one presented and well documented by Cypress Semiconductor [22], reproduced in EAGLE CAD's library editor to fit the specific needs of this project.

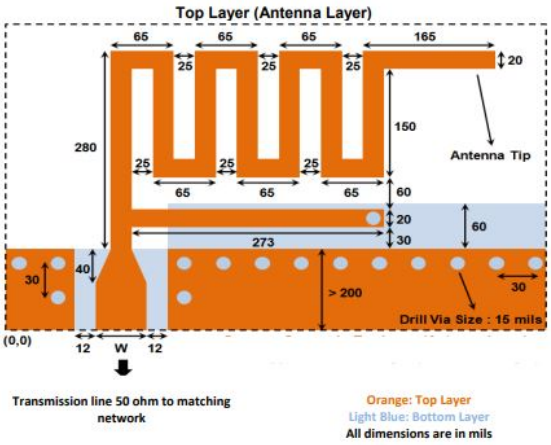


Figure 3.9.7: MIFA layout guideline [22]

Its positioning was chosen taking into account the specific directionality of this antenna, with its area of most intense radiation (the tip) facing the supposed direction of the car.

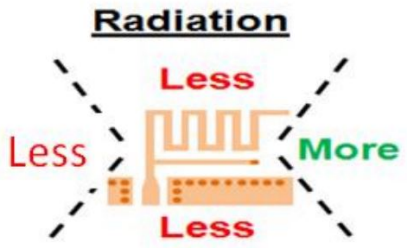


Figure 3.9.8: MIFA directionality schematic

As already established, the balun (balanced-unbalanced transformer), which transforms differencial signals into single-ended ones, isolating both of them to improve signal integrity, as well as the RF filter are woven into the same, tightly integrated package, as opposed to the predominantly discrete approach suggested in the radio IC's product specification [27]. This was done so to reduce performance variability between multiple units, to have adequate filter characteristics for the nRF24L01+'s radio and adequate impedance matching, reducing return loss. Most importantly, though, it helps reduce time-to-market and Non-Recurring Engineering (NRE) costs, reallocating time that would otherwise have to be spent researching on the deep field of RF design to be

used productively in more important features of the design. The most relevant disadvantage of this approach is the lack of freedom in filter tuning, which is well beyond the scope of this project in any case.

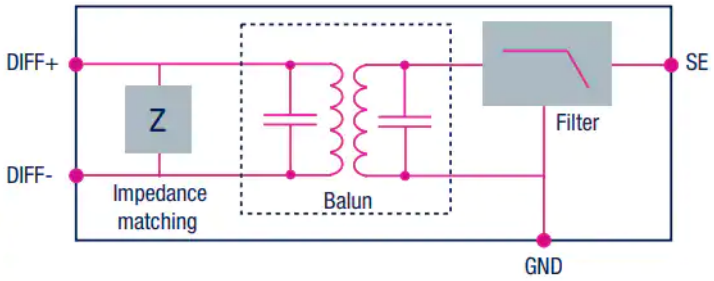


Figure 3.9.9: Block diagram of a balun transformer [28]

The chosen model for this application was the Johanson Technology 2450BM14A0002, which is impedance-matched at the input to a myriad of Nordic chips, the nRF24L01+ being one of them and at the output to a 50Ω transmission line. It was placed as close as possible to the chip to reduce the length of the traces carrying the differential antenna signal coming from it. These traces were also length-matched, following well-known high speed interface layout design guidelines [23]. Care was also taken to place the antenna right beside the IC to circumvent the need for a thick and long transmission line. The layout otherwise follows the company's own application note pertaining specifically to designs using nRF24L01 and nRF24L01+ chipsets [24].

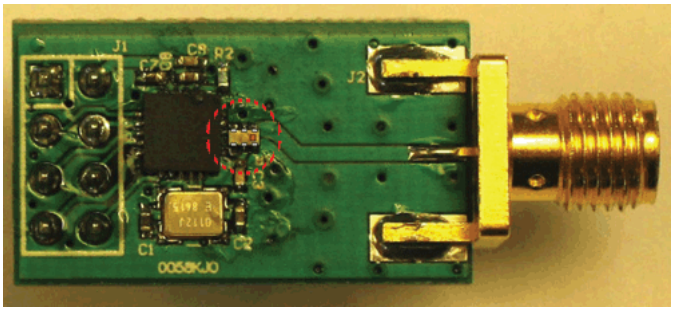


Figure 3.9.10: Suggested layout for the 2450BM14A0002 chip when matched with the nRF24L01+

Finally, the RF filter and the antenna itself are covered in a via fence, which helps reduce signal strength loss through radially propagated energy reaching the edge of the board. This type of via is usually flooded with solder mask (tented vias), although in the prototyping stage this wasn't possible as the PCB manufacturer didn't support the manufacturing of such vias due to clearance restrictions [26].

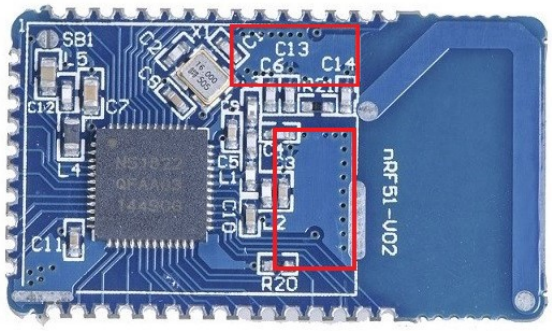


Figure 3.9.11: Example of an application of via fencing

Programming Header To the left most of the board lies the programming header. This was strategically done to have the space around the antenna free from any other signals during normal execution while still not leaving it underutilized in the prototyping stage. It is meant to be left desoldered after programming of the prototype has been done.

Accelerometer/Gyroscope Header The Programming header shares the left side of the board with the Accelerometer/Gyroscope header. This one is strategically located in the lower half of the board to be closer to the index finger, where the Accelerometer/-Gyroscope module will be tied during normal usage.

Human-Machine Interface The lower sector of the board is where the whole of the Human-Machine Interface (HMI) is located. This places it closer to the user, making it easier and more comfortable to view and access.

Controller The controller is located in the center of the board, roughly equidistant to every peripheral, as it is the link that connects the multiple edges of the system and weaves it into a cohesive package. Thus, its orientation and pin assignment also follow the location of each peripheral as close as possible to minimize trace lengths and unnecessary vias. The type of package chosen for it is the same as the one in the development board to maintain cohesion between prototypes and ease in hand soldering.

3.10 Remote System: Software Specification

This section will present the SW Specification for the **Remote System** presented in section 2.8.

3.10.1 zGyroAccelerometerManager

The flowchart for the remote gyroscope and accelerometer Task is represented in figure 3.10.1.

Startup and Execution

This module initialization is similar to the local one 3.6.1, since it requires the protocol initialization and hardware configuration. Although, in this case, the system requests a connection with the local instead of awaiting it.

The execution stage of the remote flowchart for this sensor is pretty much equivalent to the one presented in the local system 3.6.4. The only differences are that the mean value will be processed and stored in the task-shared queue (qRF) by other tasks to be later sent to the local board by the RF module, and the terminate signal will close the connection with the local system and decrement a downwards semaphore that will show that all the tasks performed cleanup and that the overall system can power itself off.

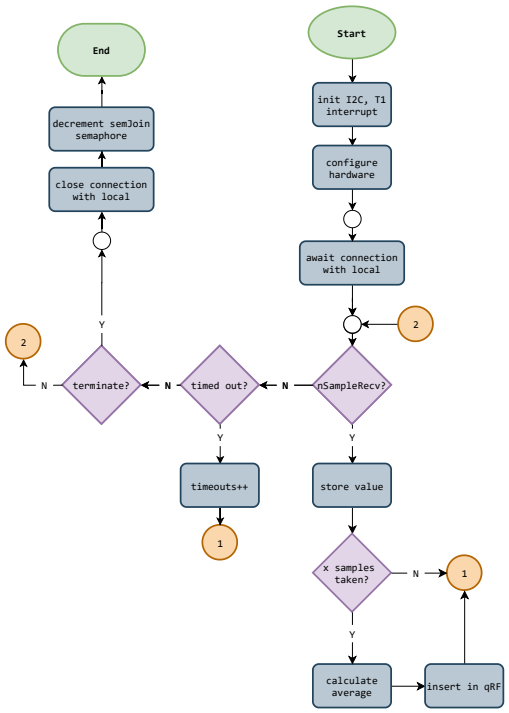


Figure 3.10.1: zGyroAccelerometerManager Task Flowchart (Remote)

3.10.2 zRFManager

The zRFManager task (figure 3.10.2) only initiates the SPI and GPIO peripherals to perform the configuration of the physical RF Module.

Startup

Starting up the RF communication mechanism in the remote system is very similar to the same process in the Local System, the difference being that the wait to go into the execution state is for the Local System's response to a connection request.

At startup, the zRFManager task first configures the SPI and GPIO peripherals to then perform the configuration of the physical RF Module, selecting parameters such as auto-acknowledgement, auto re-transmit delay and count and radio power. After receiving a connection request and going through a handshake process, it gives the appropriate signal enters the continuous execution state.

Execution

After establishing a connection with the Local System, it continuously checks if there are messages ready to send, sending them if so is the case. It also checks if a message has reached the maximum number of re-transmissions, in which case it must signal connection problems to the user. If at a certain point there are no more tasks for the hardware RF module to execute, it must be sent into stand-by mode. The response to a shutdown request is closing the connection to the local system.

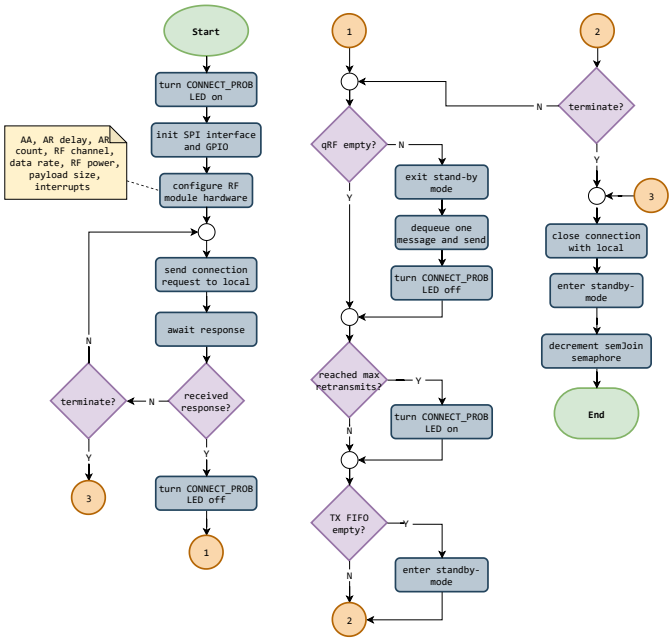


Figure 3.10.2: zRFManager Task Flowchart (Remote)

3.11 Low Power Design

Regarding **Low Power Applications**, there isn't a one-size-fits-all solution that is applicable in all instances. Power management is one of the top criteria that affects all areas of the design, including microcontroller selection [21]. With that in mind, a low power MCU was preemptively selected to favor the **power-performance pairing**.

3.11.1 Challenges

Concerning the all the LP parameters and the trade-offs between them, one can identify four high-level questions to take into account [21]:

1. Will the system be able to perform the specified task in the within the maximum available time?
2. Will the energy source support the required peak consumption?
3. Will the energy source achieve the expected operational lifetime of the application?
4. Does the system fulfill the three above criteria under typical and worst-case scenarios?

All these concepts will serve as the bedrock for some decisions made when one approaches the implementation stage.

3.11.2 Power Profiling

Generating a power profile is very important, since it allows for interval average current calculation and a better decision making when it comes to what should be disabled to save power.

Power Modes Overview

The specifications for all the power modes can be found on the **RM0410 Reference manual - STM32F76xxx**, figure 3.11.1. After analysing all the features each mode presents, one opted for the **Sleep Mode** as the Low-Power mode for this application. The summary for the latter can be observed in figure 3.11.2.

Peripheral	Run	Sleep	Stop		Standby		VBAT
				Wakeup		Wakeup	
CPU	Y	-	-	-	-	-	-
Flash access	Y	Y	-	-	-	-	-
DTCM RAM	Y	Y	Y	-	-	-	
ITCM RAM	Y	Y	Y	-	-	-	
SRAM1	Y	Y	Y	-	-	-	-
SRAM2	Y	Y	Y	-	-	-	-
FMC	O	O	-	-	-	-	-
QUADSPI	O	O	-	-	-	-	-
Backup Registers	Y	Y	Y	-	Y	-	Y
Backup RAM	Y	Y	Y	-	Y	-	Y
Brown-out reset (BOR)	Y	Y	Y	Y	Y	Y	
Programmable Voltage Detector (PVD)	O	O	O	O	-	-	-
High Speed Internal (HSI)	O	O	(2)	-	-	-	-
High Speed External (HSE)	O	O	-	-	-	-	-
Low Speed Internal (LSI)	O	O	O	-	O	-	-
Low Speed External (LSE)	O	O	O	-	O	-	O
RTC	O	O	O	O	O	O	O

(a) Part 1

Peripheral	Run	Sleep	Stop		Standby		VBAT
				Wakeup		Wakeup	
Number of RTC tamper pins	3	3	3	3	3	3	2
CRC calculation unit	O	O	-	-	-	-	-
GPIOs	Y	Y	Y	Y		6 pins	2 tamper
DMA	O	O	-	-	-	-	-
Chrom-Art Accelerator (DMA2D)	O	O	-	-	-	-	-
LCD-TFT	O	O	-	-	-	-	-
DCMI	O	O	-	-	-	-	-
USARTx (x=1..8)	O	O	-	-	-	-	-
I2Cx (x=1,2,3,4)	O	O	-	-	-	-	-
SPIx (x=1..6)	O	O	-	-	-	-	-
SAIx (x=1,2)	O	O	-	-	-	-	-
SPDIFRX	O	O	-	-	-	-	-
ADCx (x=1,2,3)	O	O	-	-	-	-	-
DACx (x=1,2)	O	O	-	-	-	-	-
Temperature sensor	O	O	-	-	-	-	-
Timers (TIMx)	O	O	-	-	-	-	-
Low-power timer 1 (LPTIM1)	O	O	O	O	-	-	-
Independent watchdog (IWDG)	O	O	O	O	O	O	-
Window watchdog (WWDG)	O	O	-	-	-	-	-
Systick timer	O	O	-	-	-	-	-
Random number generator (RNG)	O	O	-	-	-	-	-
Cryptographic processor (CRYP)	O	O	-	-	-	-	-
Hash processor (HASH)	O	O	-	-	-		-
SDMMC1, SDMMC2	O	O	-	-	-	-	-
CANx (x=1,3)	O	O	-	-	-	-	-
USB OTG FS	O	O	-	O	-	-	-
USB OTG HS	O	O	-	O	-	-	-
Ethernet	O	O	-	O	-	-	-
HDMI-CEC	O	O	-	-	-	-	-
DSI-Host	O	O	-	-	-	-	-

(b) Part 2

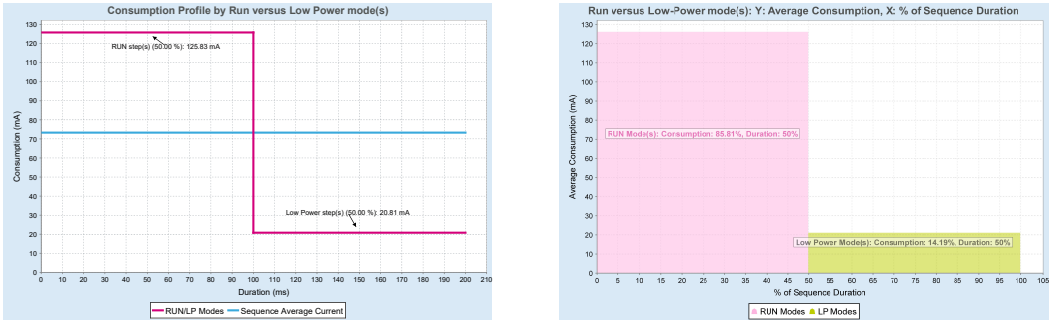
Figure 3.11.1: Power Mode Features

Mode name	Entry	Wakeup	Effect on 1.2 V domain clocks	Effect on V_{DD} domain clocks	Voltage regulator
Sleep (Sleep now or Sleep-on-exit)	WFI	Any interrupt	CPU CLK OFF no effect on other clocks or analog clock sources	None	ON
	WFE	Wakeup event			

Figure 3.11.2: Sleep Mode Summary

PCC: Power Consumption Calculator

The Power Consumption Calculator (PCC) is a tool that allows for embedded application's estimated power consumption simulations. With it one can compare the expected power consumption of the system when switching between running mode and Sleep (Low-Power) mode.



(a) Part 1 (b) Part 2

Figure 3.11.3: PCC Run vs Sleep mode simulations

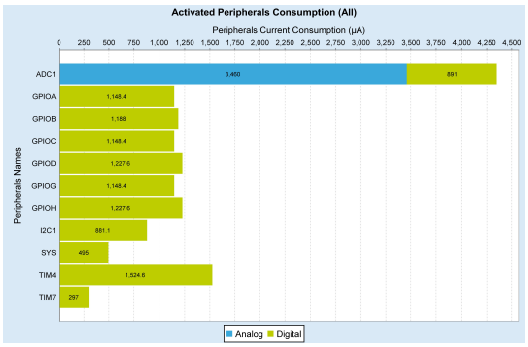


Figure 3.11.4: PCC peripheral consumption simulations

As expected, one can notice the difference between using a Low-Power mode in the application, with, teoretically, 72% less consumption (mA) in the Sleep mode. Additionally, it is also possible to identify that the ADC is one of the peripherals with higher power consumption. This information is relevant to reduce even more the power consumption of the overall application when the peripheral isn't performing any conversions.

3.12 Local Test Cases

3.12.1 Local Unit Tests

Local: Unit Tests	Expected Results	Real Results
Run inference	Inference successful	
Configure nRF24L01+ SPI	Configuration successful and status read	
Configure MPU-6050	Configuration successful and status read	
Make ultrasonic sensor readings	Able to read	
Verify task synchronism	Synchronized and timely correct tasks	

Table 3.1: Local Unit Test Specification

3.13 Remote Test Cases

3.13.1 Remote: Prototype Alpha Unit Tests

Remote: Prototype Alpha Unit Tests	Expected Results	Real Results
Test STM32F051 dev board PCB continuity	No unwanted continuities	
Try to read STM32F051 memory w/ flash read utility	Read flash	
Configure nRF24L01+ SPI	Configuration successful and status read	
Configure MPU-6050	Configuration successful and status read	
Press the power button while the device is off	Successful power on sequence	
Press the power button while the device is on	Successful power off sequence	
Verify LEDs responsiveness and current draw	Able to control the LEDs and get a current draw within the specified limits	
Verify task synchronism	Synchronized and timely correct tasks	

Table 3.2: Prototype Alpha Unit Tests Specification

3.13.2 Remote: Prototype Beta Unit Tests

Remote: Prototype Beta Unit Tests	Expected Results	Real Results
Test wristband PCB continuity	No unwanted continuities	
Try to read STM32F051 memory w/ flash read utility	Read flash	
Configure nRF24L01+ SPI	Configuration successful and status read	
Configure MPU-6050	Configuration successful and status read	
Press the power button while the device is off	Successful power on sequence	
Press the power button while the device is on	Successful power off sequence	
Verify LEDs responsiveness and current draw	Able to control the LEDs and get a current draw within the specified limits	
Verify task synchronism	Synchronized and timely correct tasks	

Table 3.3: Prototype Beta Unit Tests Specification

3.14 Integration Tests

3.14.1 Prototype Alpha Integration Tests

Prototype Alpha Integration Tests	**Expected Results**	**Real Results**
Verify connection	Connection successful	
Make the car follow a predetermined path	Car follows the instructions	
Verify object detection	Car stops after the detection of an obstacle	
Press the power button while the device is off	Successful power on sequence	
Press the power button while the device is on	Successful power off sequence	

Table 3.4: Prototype Alpha Integration Tests Specification

3.14.2 Prototype Beta Integration Tests

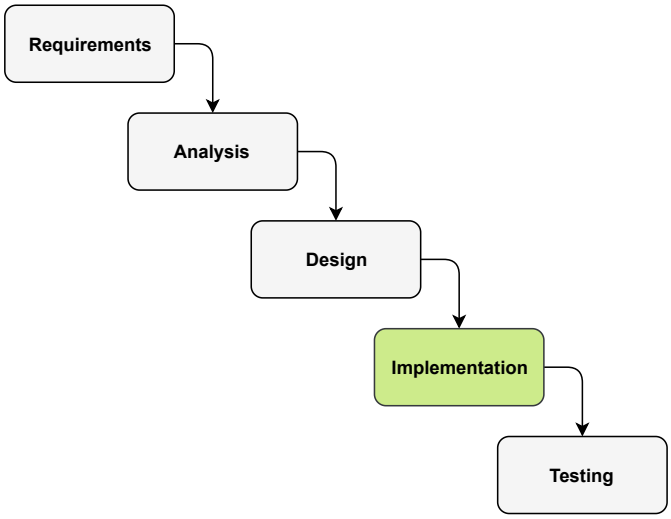
Prototype Beta Integration Tests	**Expected Results**	**Real Results**
Verify connection	Connection successful	
Make the car follow a predetermined path	Car follows the instructions	
Verify object detection	Car stops after the detection of an obstacle	
Press the power button while the device is off	Successful power on sequence	
Press the power button while the device is on	Successful power off sequence	

Table 3.5: Prototype Beta Integration Tests Specification

Chapter 4

Implementation

This chapter will present the implementation guidelines one must follow to achieve the intended prototype attending to what was stipulated in the design phase. All the steps will be described in detail to ensure rich product documentation. This matters, as the document might be used for future reference.



4.1 Local System Configurations

Considering that the project accents onto two different ST Microelectronics systems, the breakdown of the implementation will start from the necessary **STM32CubeMX configurations** for the local system.

4.1.1 System Core

In figure 4.1.1 one can see the final pinout layout for the local system MCU.

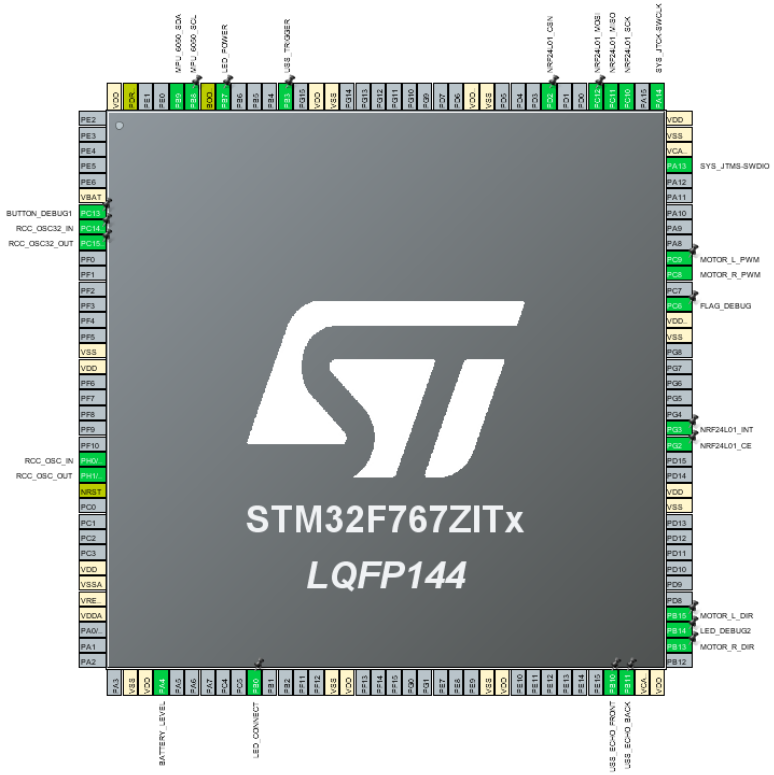


Figure 4.1.1: STM32F767Zi Pinout View

Additionally, the GPIO pins for the system can be observed in figure 4.1.2. For the matter, `PB0` was defined as the LED (green) that indicates a successful connection with the remote system, and `PB13` and `PB15` as the output pins that indicate the right and left motor directions, respectively. The `PB14`, `PC13`, and `PC6` pins were used for debug purposes when testing the alpha and beta prototypes. The last three pins represented, `PD2`, `PG2`, and `PG3`, are related to the RF module. Note that it is mandatory that the pin `PG3` is set to external interrupt mode with falling edge detection since the `NRF24L01+` interrupt line is active low.

Pin Name	Si...	GPIO output level	GPIO mode	GPIO Pull-up/Pull-down	Maximum ...	Fast ...	User Label	Modified
PB0	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	LED_CONNECT	☑
PB13	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	MOTOR_R_DIR	☑
PB14	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	LED_DEBUG2	☑
PB15	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	MOTOR_L_DIR	☑
PC6	n/a	Low	Output Push Pull	No pull-up and no pull-down	Very High	n/a	FLAG_DEBUG	☑
PC13	n/a	n/a	External Interrupt Mode with Rising edge trigger det...	No pull-up and no pull-down	n/a	n/a	BUTTON_DEBUG1	☑
PD2	n/a	High	Output Push Pull	Pull-up	Low	n/a	NRF24L01_CSN	☑
PG2	n/a	Low	Output Push Pull	Pull-up	Low	n/a	NRF24L01_CE	☑
PG3	n/a	n/a	External Interrupt Mode with Falling edge trigger det...	Pull-up	n/a	n/a	NRF24L01_INT	☑

Figure 4.1.2: GPIO Configuration

Concerning the Nested Vectored Interrupt Controller (NVIC), one can notice five enabled interrupts. Two of them are for the external interrupts mentioned earlier (`PC13` - EXTI line[15:10] and `PG3` - EXTI line3), other two for the timer 2 and timer 6 interrupts, and the last one is the the I^2C 1 interrupt. Note that there is also an important interrupt related to the timer 7 that is generated for every RTOS kernel tick.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority	Uses FreeRTOS functions
Non maskable interrupt	☑	0	0	☐
Hard fault interrupt	☑	0	0	☐
Memory management fault	☑	0	0	☐
Pre-fetch fault, memory access fault	☑	0	0	☐
Undefined instruction or illegal state	☑	0	0	☐
System service call via SWI instruction	☑	0	0	☐
Debug monitor	☑	0	0	☐
Pendable request for system service	☑	15	0	☑
System tick timer	☑	15	0	☑
EXTI line3 interrupt	☑	5	0	☑
TIM2 global interrupt	☑	5	0	☑
I2C1 event interrupt	☑	5	0	☑
EXTI line[15:10] interrupts	☑	5	0	☑
TIM6 global interrupt, DAC1 and DAC2 u...	☑	5	0	☑
Time base: TIM7 global interrupt	☑	0	0	☐

Figure 4.1.3: NVIC Configuration

This nucleo board provides two external clock sources, High Speed External Clock (HSE) and Low Speed External Clock (LSE). In this case, the two were enabled in crystal resonator mode just to open more options when choosing the clock source for the system and other peripherals.

Figure 4.1.4: RCC Configuration

4.1.2 Clock Tree

For the clock tree, one routed the path that was more suitable for the application, in this case, choosing the High Speed Internal Clock (HSI) for the source, since the

dependency on the external clock could prove itself troublesome if the peripheral were to fail because of the board environment exposure. Moreover, the PLL was used in this case to easily achieve the desired frequency for the system. Of course that the use of the PLL comes with an increase in power consumption, but considering that the local side has bulkier batteries, this increase didn't jeopardize the overall low-power profile one was trying to accomplish. With that in mind, the system runs at $216MHz$ with the peripheral/timer frequencies specified in figure 4.1.5.

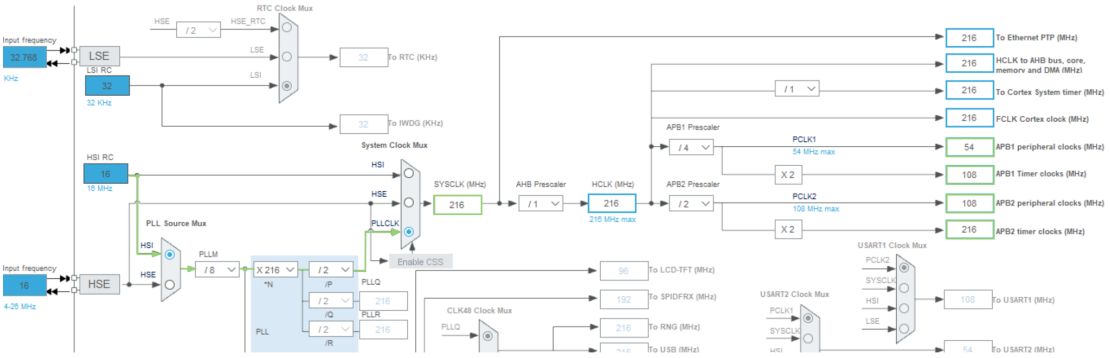


Figure 4.1.5: Clock Tree

4.1.3 Timers

The plethora of timers used for the local system is represented in table 4.1 and the timer-associated pins can be seen in figure 4.1.6.

Timer	Purpose
2	Ultrasonic sensor timing
3	Dual channel PWM generation for the motors
4	Power LED
6	ADC trigger
7	FreeRTOS kernel timebase

Table 4.1: Local System Timer Configuration

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pu...	Maximum output...	Fast Mode	User Label	Modified
PB3	TIM2_CH2	n/a	Alternate Functi...	Pull-down	Low	n/a	USS_TRIGGER	☑
PB10	TIM2_CH3	n/a	Alternate Functi...	No pull-up and n...	Low	n/a	USS_ECHO_FRONT	☑
PB11	TIM2_CH4	n/a	Alternate Functi...	No pull-up and n...	Low	n/a	USS_ECHO_BACK	☑
PC8	TIM3_CH3	n/a	Alternate Functi...	No pull-up and n...	Medium	n/a	MOTOR_R_PWM	☑
PC9	TIM3_CH4	n/a	Alternate Functi...	No pull-up and n...	Medium	n/a	MOTOR_L_PWM	☑
PB7	TIM4_CH2	n/a	Alternate Functi...	No pull-up and n...	Low	Disable	LED_POWER	☑

Figure 4.1.6: Timer Pin Configuration

4.1.4 Native FreeRTOS

Instead of relying on the cube-generated and RTX-based CMSIS RTOS V1 designed for Cortex-M0/M0+/M3/M4/M7 processors, one decided to bypass the initialization of the latter whilst still using Cube for some configurations (listing 4.1). To replace CMSIS, one devised a RTOS wrapper (`_rtos_wrapper.h`) to simplify the interaction with the native RTOS API. Furthermore, the Cube configurations consisted of enabling preemption, and defining the kernel frequency to $200Hz$, which corresponds to a $5ms$ tick (figure 4.1.7).

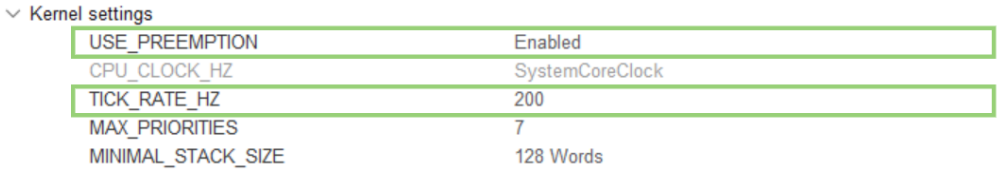


Figure 4.1.7: RTOS Configutation

```
#if 0   /* Bypass CMSIS Definition */

  MX_FREERTOS_Init();
  osKernelStart();

#endif

/* Create Tasks */
THREADS_create();
/* Start scheduler */
THREADS_startScheduler();
```

Listing 4.1: Initial RTOS Configuration

4.1.5 Edge Machine Learning

To run inference with a machine learning model in the nucleo board, one had to enable the Cube.AI extension that allows for model analysis and further deployment onto the edge device.

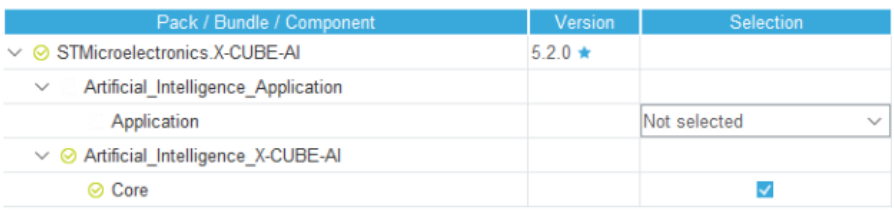


Figure 4.1.8: Cube.AI Extension Configuration

4.1.6 SPI

The SPI configuration (figure 4.1.9) necessary for interfacing with the `NRF24L01+` module is depicted in figure 4.1.9. Note that the module's baudrate cap for on air communications is $2Mbps$, hence the prescaler chosen to keep the baudrate below this value. The pin configurations for SPI are depicted in figure 4.1.10.

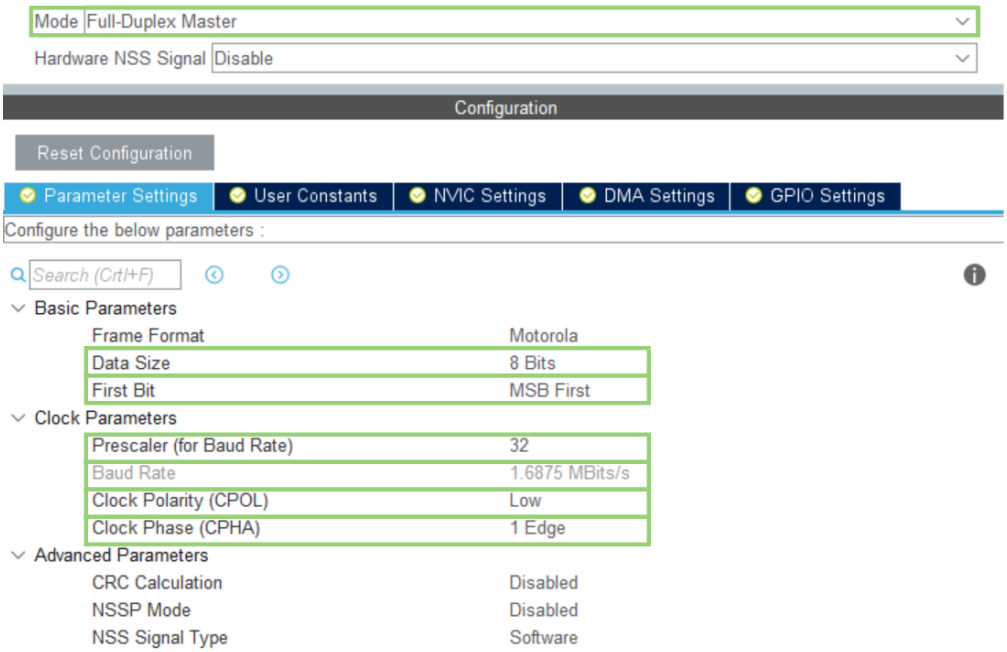


Figure 4.1.9: SPI Configuration

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pu...	Maximum output...	Fast Mode	User Label	Modified
PC11	SPI3_MISO	n/a	Alternate Functi...	No pull-up and n...	Very High	n/a	NRF24L01_MISO	☑
PC12	SPI3_MOSI	n/a	Alternate Functi...	No pull-up and n...	Very High	n/a	NRF24L01_MOSI	☑
PC10	SPI3_SCK	n/a	Alternate Functi...	No pull-up and n...	Very High	n/a	NRF24L01_SCK	☑

Figure 4.1.10: SPI Pin Configuration

4.1.7 I^2C

The I^2C configuration necessary for communicating with the gyroscope and accelerometer module is represented in figure 4.1.11. One is using fast mode I^2C with a frequency of $400kHz$. The pin configurations for I^2C are represented in figure 4.1.12.

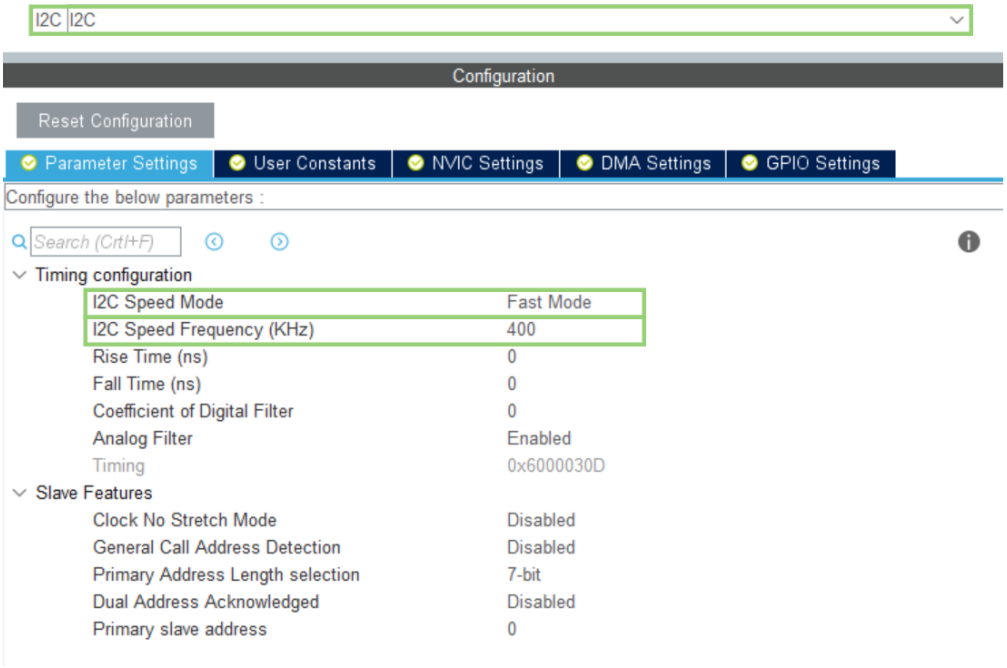


Figure 4.1.11: I^{2}C Configuration

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pu...	Maximum output...	Fast Mode	User Label	Modified
PB8	I2C1_SCL	n/a	Alternate Functi...	Pull-up	Very High	Disable	MPU_6050_SCL	☑
PB9	I2C1_SDA	n/a	Alternate Functi...	Pull-up	Very High	Disable	MPU_6050_SDA	☑

Figure 4.1.12: I^{2}C Pin Configuration

4.1.8 ADC

The configuration for the ADC is depicted in figure 4.1.13.

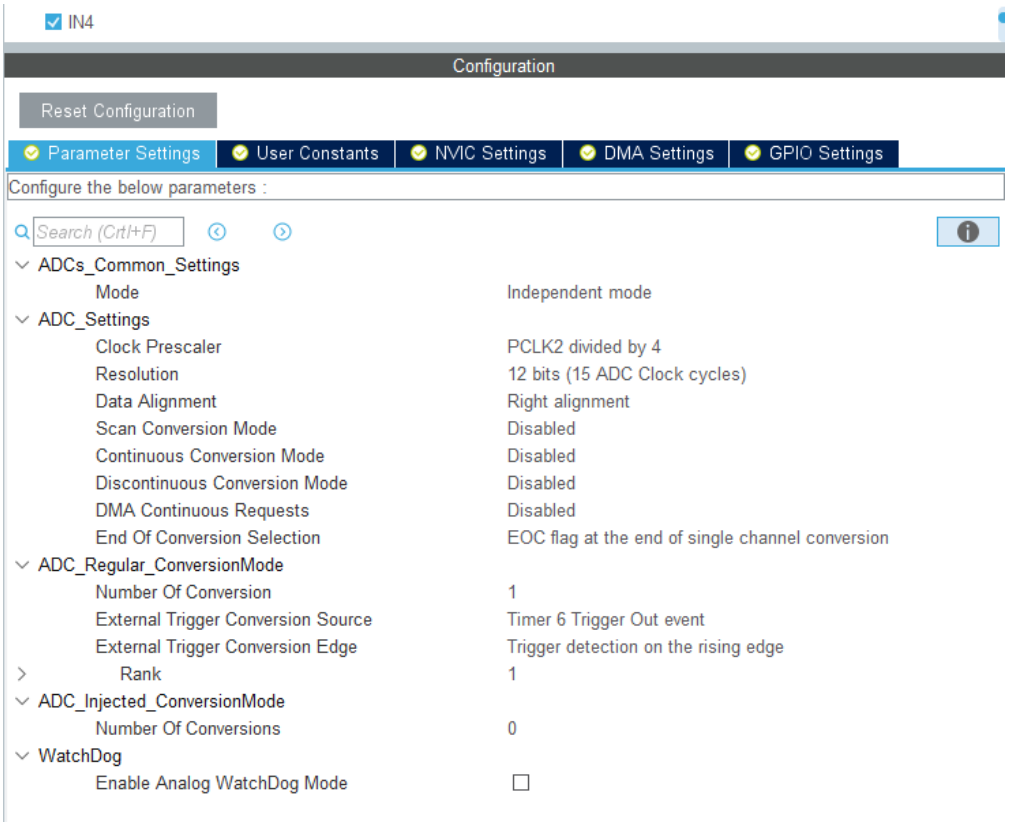


Figure 4.1.13: ADC Configuration

4.2 Local System: Timeline Structure

Taking into consideration the more abstract version of the local system's task timeline, referred in section 3.5.3, one devised a refined version of the latter based on the implementation of the local RTOS system. Note that, for intelligibility, the name of the gyroscope and accelerometer task changed from zGyroAccManager to zIMUManager.

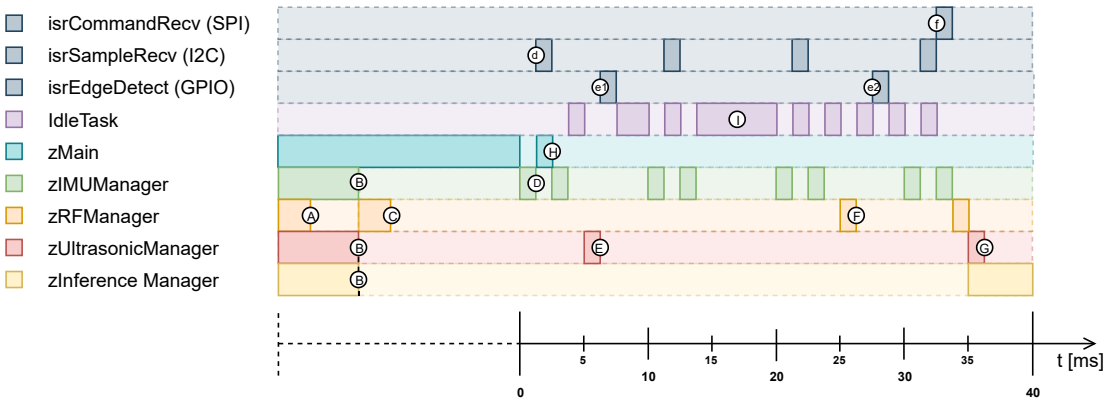


Figure 4.2.1: Local Task Timeline - Refined (Aug. in Appendix C.3)

In the table 4.2 are described all the letter-marked events of the local task timeline of figure 4.2.1.

Event ID	Event Description	Event ID	Event Description
Ⓐ	Wait for nConfig	ⓔ2	USS finished raging
Ⓑ	Increment semConfig and await nConnected	Ⓕ	Enter RX mode
Ⓒ	Send nConnected	ⓕ	Command received
Ⓓ	IMU sample requested	Ⓖ	USS distance ready
ⓓ	IMU sample received	Ⓗ	Battery check
Ⓔ	USS triggered	Ⓘ	Sleep
ⓔ1	USS started ranging		

Table 4.2: Local Timeline Events

In the event Ⓐ, the zRFManager task waits for the notification `nConfig` that indicates that all the other tasks have finished their configurations (listing 4.2).

```
/*!< System Notifications Creation */
RTOS_NOTIFICATION(nConfig, 0);
RTOS_NOTIFICATION(nConnected, 1);
RTOS_NOTIFICATION(nIMURx, 2);
RTOS_NOTIFICATION(nIMUTx, 3);
RTOS_NOTIFICATION(nRF, 4);

...

RTOS_TASK_FUN(zRFManager) {
    ...

    /*!< Wait for all configurations to be complete */
    RTOS_AWAIT(nConfig);

    ...
}
```

Listing 4.2: Notification creation and zRFManager waits for the `nConfig`

Additionally, in event Ⓑ, after their initial setup, the tasks zIMUManager, zUltrasonicManager, and zInferenceManager increment a counting semaphore `semConfig` and start waiting for the `nConnected` notification that specifies a successful connection with the wristband (listing 4.3).

```
static void THREADS_incSemConfig(void) {

        RTOS_SEMAPHORE_INC(semConfig);

        if (RTOS_SEMAPHORE_GET_COUNT(semConfig) == 3)
                RTOS_NOTIFY(zRFManager, nConfig);
}

RTOS_TASK_FUN(zIMUManager) {

    /*!< Configuration Section */

    /*!< Signal another configuration complete */
    THREADS_incSemConfig();
    /*!< Await the establishment of a connection to continue */
    RTOS_AWAIT(nConnected);

    ...
}
```

```
RTOS_TASK_FUN(zInferenceManager) {

    /*!< Configuration Section */

    /*!< Signal another configuration complete */
    THREADS_incSemConfig();
    /*!< Await the establishment of a connection to continue */
    RTOS_AWAIT(nConnected);

    ...
}

RTOS_TASK_FUN(zUltrasonicManager) {

    /*!< Configuration Section */

    /*!< Signal another configuration complete */
    THREADS_incSemConfig();
    /*!< Await the establishment of a connection to continue */
    RTOS_AWAIT(nConnected);

    ...
}
```

Listing 4.3: Tasks increment `semConfig` and wait for `nConnected`

When this semaphore reaches the value of three, the notification `nConfig` is sent to the zRFManager task, which then initializes the `NRF24L01+` module (listing 4.4). The `NRF24L01+` is initialized by default with $2Mbps$ of data rate, $0dBm$ of output power, and in the RX mode. The zRFManager task proceeds to wait for an order from the remote system to establish a handshake with it.

```
RTOS_TASK_FUN(zRFManager) {
    ...

    /*!< Wait for all configurations to be complete */
    RTOS_AWAIT(nConfig);

    /* NRF24L01+ inititialization section*/

    /* Handshake section */
    while(conn_state == NOT_CONNECTED) {
        /* Try to establish connection */
    }
    ...
}
```

Listing 4.4: `NRF24L01+` after receiving the `nConfig`

If the handshake turns out to be succeesful, the event Ⓒ occurs and the zRFManager sends the notification `nConnected` to the zIMUManager, zUltrasonicManager, and zInferenceManager tasks.

```
        /*!< Notify tasks of a successful connection */
        RTOS_NOTIFY(zIMUManager, nConnected);
        RTOS_NOTIFY(zInferenceManager, nConnected);
        RTOS_NOTIFY(zUltrasonicManager, nConnected);
```

Listing 4.5: zRFManager notifies the other tasks of a successful connection with the remote

The event Ⓓ represents a sample request to the IMU by the zIMUManager task. The IMU task waits to be notified by the IMU module that the request has been received and then, upon a received sample, event ⓓ, the task continues to fetch the data (listing 4.6).

```
void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef *hi2c){
        if(hi2c->Instance == I2C1)
                RTOS_NOTIFY_ISR(zIMUManager, nIMURx);
}

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef *hi2c){
        if(hi2c->Instance == I2C1)
                RTOS_NOTIFY_ISR(zIMUManager, nIMUTx);
}

RTOS_TASK_FUN(zIMUManager) {
    ...

    /*!< Await the establishment of a connection to continue */
    RTOS_AWAIT(nConnected);
    ...
    while(1) {
                /*!< Request sensor data */
                IMU_dataRequest(&hi2c1);
                /*!< Await reception notification */
                RTOS_AWAIT(nIMURx);
                /*!< Fetch received data */
                IMU_dataFetch(&mpu6050);

            ...
    }

    ...
}
```

Listing 4.6: IMU data request and data fetch

The event Ⓔ represents the trigger generation of the ultrasonic sensor, which translates into putting the trigger pin (PB3) to the high state (listing 4.7).

```
/*!< Pull trigger pin HIGH */
USS_TRIGGER_GPIO_Port->BSRR = USS_TRIGGER_Pin;
```

Listing 4.7: Ultrasonic sensor trigger generation

Consequently, the events ⓔ1 and ⓔ2 represent the start and end of the distance evaluation process (ranging).

```
void THREADS_tim2IRQHandler(void) {

        /*!< Read the SR register only once as the interrupt
        flags are cleared when this is done. */
        uint32_t sr = TIM2->SR;

        /*!< If it is a channel 3 (front sensor) interrupt */
        if (READ_BIT(sr, TIM_SR_CC3IF)) {

                /*!< In case of a rising edge, start counting */
                if (READ_BIT(USS_ECHO_FRONT_GPIO_Port->IDR, \
                        USS_ECHO_FRONT_Pin))
                        USS_startCount(&uss_front, TIM2->CCR3);
                else
                        USS_stopCount(&uss_front, TIM2->CCR3);

        /*!< Else if it is a channel 4 (back sensor) interrupt*/
        } else if (READ_BIT(sr, TIM_SR_CC4IF)) {

                /*!< In case of a rising edge, start counting*/
                if (READ_BIT(USS_ECHO_BACK_GPIO_Port->IDR, \
                        USS_ECHO_BACK_Pin))
                        USS_startCount(&uss_back, TIM2->CCR4);
                else
                        USS_stopCount(&uss_back, TIM2->CCR4);
        }
}

RTOS_TASK_FUN(zUltrasonicManager) {
     ...

    /*!< Await the establishment of a connection to continue */
    RTOS_AWAIT(nConnected);
    ...
    while(1) {
                ...

        /*!< Enable the Output Capture channel 2 and
```

```
            write CNT + 15us to CCR2 register */
            SET_BIT(TIM2->CCER, TIM_CCER_CC2E);
            WRITE_REG(TIM2->CCR2, TIM2->CNT + 1620);
            /*!< Sleep */
            RTOS_DELAY_UNTIL(tsUSSMan, USS_CAPTURE_TIMEOUT_MS);
            /*!< Disable the Output Capture channel 2 */
            CLEAR_BIT(TIM2->CCER, TIM_CCER_CC2E);

                ...
        }

}
```

Listing 4.8: Ultrasonic sensor ranging

The event Ⓕ represents the start of a listening period, in which the controller is waiting the reception of a command while the nRF24L01+ peripheral is in RX mode and the event ⓕ represents a command received.

```
RTOS_TASK_FUN(zRFManager) {

    ...

    while(1) {

        /*!< Enter RX mode */
        RF_PowerUpRx();

        /*!< Await command */
        RTOS_AWAIT_TIMEOUT(nRF, 5);

        /*! < Poll interrupt flags for DataReady */
        RF_Read_Interrupts(&nrf_irq);

        ...

        /*!< Enter sleep mode */
        RF_PowerDown();

        /*!< Sleep */
        RTOS_DELAY_UNTIL(tsRFMan, MASTER_CYCLE_PERIOD_MS);
    }

    ...
}
```

The event Ⓗ represents the battery check process, wherein, after a certain period has elapsed, the controller checks the voltage measured in the battery, calculates the

charge level in it and outputs a warning representing an appropriate or low battery level.

```
RTOS_TASK_FUN(zMain) {

    /*!< ADC1: Clear regular group conversion flag and overrun
    * flag */
    CLEAR_BIT(ADC1->SR, ADC_FLAG_EOC | ADC_FLAG_OVR);

    /*!< ADC1: Enable end of conversion interrupt for regular
    * group */
    SET_BIT(ADC1->CR1, ADC_IT_EOC | ADC_IT_OVR);

    /*!< ADC1: Enable ADC */
    SET_BIT(ADC1->CR2, ADC_CR2_ADON);

static double THREADS_checkForLowBattery(void) {

#define EVAL_LOW            1
#define EVAL_HIGH           0
#define VBATT_RSCALE        3
#define VBATT_MAX           8200
#define VBATT_MIN           6000
#define VBATT_EXCURSION     (VBATT_MAX-VBATT_MIN)
#define LOW_BATT_THRSLD     15
#define HYSTERESIS          2

    /*!< Evaluation from last iteration */
    static uint32_t last_evaluation = EVAL_HIGH;
    uint32_t evaluation;    /*!< Evaluation to be made */
    int32_t adc_value;      /*!< Raw ADC reading        */
    int32_t v_batt;         /*!< Battery voltage */
    int32_t batt_lvl;       /*!< Calculated battery level */

    /*!< Fetch ADC conversion value */
    adc_value = READ_REG(ADC1->DR);

    /*!< Calculate real battery voltage */
    v_batt = adc_value * 3300 / 4096 * VBATT_RSCALE;

    /*!< Extrapolate 0-100 battery level assuming linear
    * voltage drop */
    batt_lvl = 100 * (v_batt - VBATT_MIN) / VBATT_EXCURSION;

    /*!< If last evaluation was LOW, go to HIGH only once the
     *    hysteresis margin has been overcome in the UP
     *    direction
     *    If last evaluation was HIGH, go to LOW only once the
     *    hysteresis margin has been overcome in the DOWN direction */
    if (last_evaluation == EVAL_LOW)
```

```
        evaluation = batt_lvl < LOW_BATT_THRSLD + HYSTERESIS;
    else
        evaluation = (batt_lvl <= LOW_BATT_THRSLD - HYSTERESIS);

    /*!< Keep calculated value for future reference */
    last_evaluation = evaluation;

    return evaluation;
}
```

Finally, the event Ⓘ represents the sleep phase of the system and was intentionally placed in the in the system's idle task.

```
static void THREADS_sleep(void){

        /* Clear SLEEPDEEP bit of Cortex System Control
        Register */
        CLEAR_BIT(SCB->SCR, SCB_SCR_SLEEPDEEP_Msk);

        /* Request Wait For Interrupt */
        __WFI();
}
```

4.3 Remote System Configurations

This section will refer to the remote system configurations, valid for the alpha and beta prototypes.

4.3.1 System Core

In figure 4.3.1 one can observe the final pinout layout for the remote system.

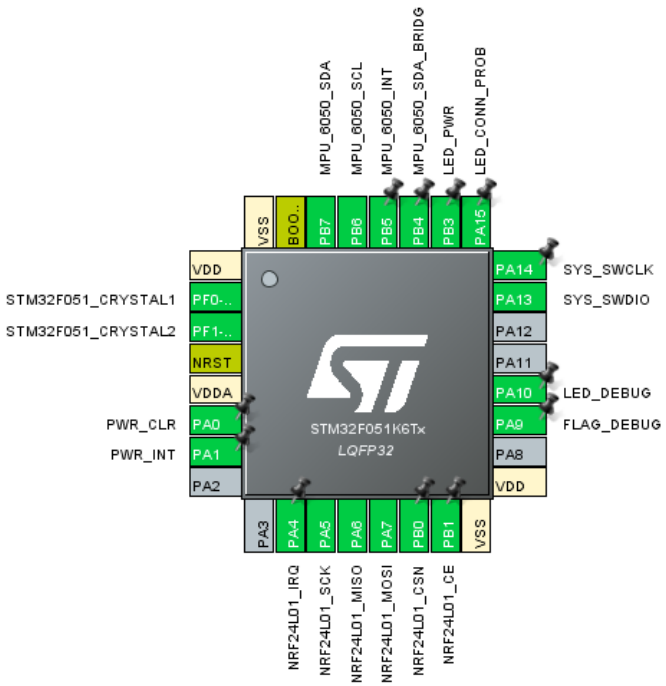


Figure 4.3.1: STM32F051KxTx Pinout View

This pinout differs from the initial one presented in section 3.9.1 due to a design error in the conception of the beta prototype's board, which was only detected after the boards had arrived. It consisted in all the connections of the pins from 26 (PB3) to 31 (BOOT0) being shifted one logical position down. This affected the position of the LED, I^2C and BOOT0 pins.

In short, the solution that was found to at least get the LEDs and interface with the IMU working completely without damaging the board, as well as the MCU working consistently was to:

- Remove BOOT0's pull-down resistor, since it was now connected to PB7, which is an I^2C communication pin;
- Short BOOT0 and VSS using a solder blob, ensuring BOOT0's connection to

ground, although without a pull-down resistor, which would be the ideal situation;

- Short the PB7 and PB4 pins with a small copper wire, using PB4 as a bridge for the SDA line in the connection with the IMU IC;
- Change the LED_POWER control CC channel to TIM2'S CC2, in order to control it from using the PB3 pin.

In figure 4.3.2 one can see the all the GPIO pins used for the remote system.

Pin N...	Signal...	GPIO outpu...	GPIO mode	GPIO Pull-up/Pull-down	Ma...	Fas...	User Label	Modif...
PA1	n/a	n/a	External Interrupt Mode with Falling edge trigger detection	Pull-up	n/a	n/a	PWR_INT	☑
PA4	n/a	n/a	External Interrupt Mode with Falling edge trigger detection	Pull-up	n/a	n/a	NRF24L01_IRQ	☑
PB4	n/a	n/a	Analog mode	No pull-up and no pull-down	n/a	n/a	MPU_6050_SDA_BRIDGE	☑
PB5	n/a	n/a	External Interrupt Mode with Falling edge trigger detection	Pull-up	n/a	n/a	MPU_6050_INT	☑
PA0	n/a	High	Output Push Pull	No pull-up and no pull-down	Medium	n/a	PWR_CLR	☑
PA15	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	LED_CONN_PROB	☑
PB0	n/a	High	Output Push Pull	Pull-up	Low	n/a	NRF24L01_CSN	☑
PB1	n/a	Low	Output Push Pull	Pull-up	Low	n/a	NRF24L01_CE	☑
PA9	n/a	Low	Output Push Pull	No pull-up and no pull-down	High	n/a	FLAG_DEBUG	☑
PA10	n/a	Low	Output Push Pull	No pull-up and no pull-down	High	n/a	LED_DEBUG	☑

Figure 4.3.2: Remote - GPIO Configuration

Additionally, in figure 4.3.3 are displayed the pins necessary for flashing and debugging the remote system.

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pul...	Maximum output ...	Fast Mode	User Label	Modified
PA13	SYS_SWDIO	n/a	n/a	n/a	n/a	n/a		☐
PA14	SYS_SWCLK	n/a	n/a	n/a	n/a	n/a		☐

Figure 4.3.3: Remote - SYS Pin Configuration

Depicted in figure 4.3.4 are the the enabled interrupts for the application. The most important are the ones considering external interrupts from the line 0 and 1 (power interrupt), line 4 to 15 (`MPU6050` and `nRF24L01+` interrupts), and the I^2C interrupt.

NVIC Interrupt Table	Enabled	Preemption Priority	Uses FreeRTOS functi...
Non maskable interrupt	☑	0	☐
Hard fault interrupt	☑	0	☐
System service call via SWI instruction	☑	0	☐
Pendable request for system service	☑	3	☑
System tick timer	☑	3	☑
EXTI line 0 and 1 interrupts	☑	3	☑
EXTI line 4 to 15 interrupts	☑	3	☑
Time base: TIM6 global and DAC underrun error interrupts	☑	0	☐
I2C1 event global interrupt / I2C1 wake-up interrupt through EXTI line 23	☑	3	☑

Figure 4.3.4: Remote - NVIC Configuration

4.3.2 Clock Tree

In terms of the clock tree, once again, one chose the path that was more convenient to the application, as a $48MHz$ system clock was necessary, the path includes a PLL.

Although this can increase the power consumption, this was a trade-off one was willing to make to increase performance in some key aspects.

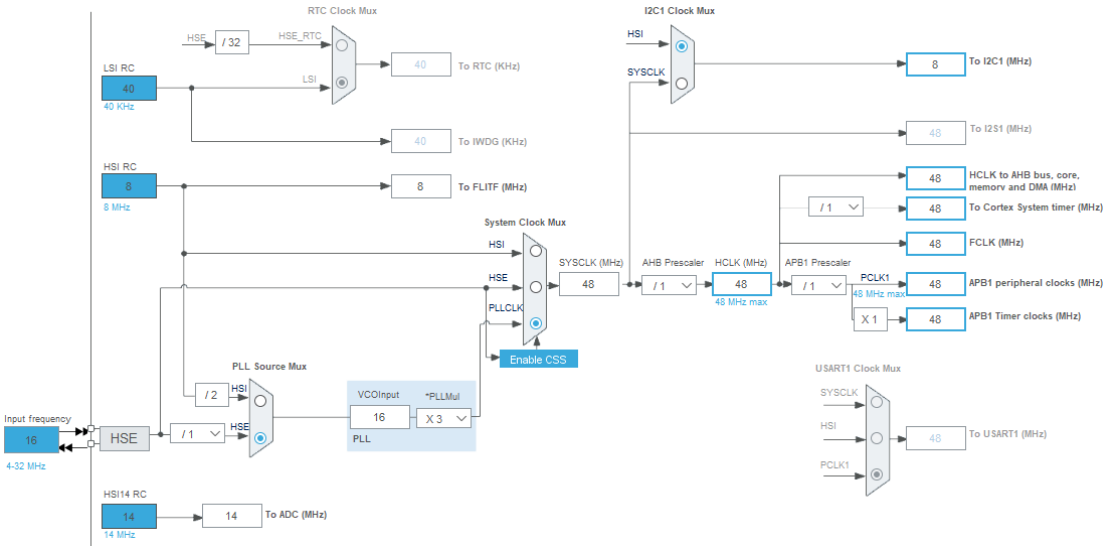


Figure 4.3.5: Remote - Clock Tree

4.3.3 Timers

The timers used for the remote system is represented in table 4.3 and the timer-associated pins can be seen in figure 4.3.6.

Timer	Purpose
2	PWM generation for the low battery LED
6	FreeRTOS kernel timebase

Table 4.3: Remote - Local System Timer Configuration

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pul...	Maximum output ...	Fast Mode	User Label	Modified
PB3	TIM2_CH2	n/a	Alternate Functio...	No pull-up and n...	Low	n/a	LED_PWR	☑

Figure 4.3.6: Remote - Timer Pin Configuration

4.3.4 Native FreeRTOS

As in the local system, the native FreeRTOS was used, so the configuration of section 4.1.4 applies here as well. This also means that a $5ms$ tick was used.

4.3.5 SPI

The SPI configuration (figure 4.3.7) necessary for interfacing with the `NRF24L01+` is represented in figure 4.3.7. Note that the module's baudrate cap for on air communications is $2Mbps$, hence the prescaler chosen to keep the baudrate below this value. The pin configurations for SPI are depicted in figure 4.3.8.

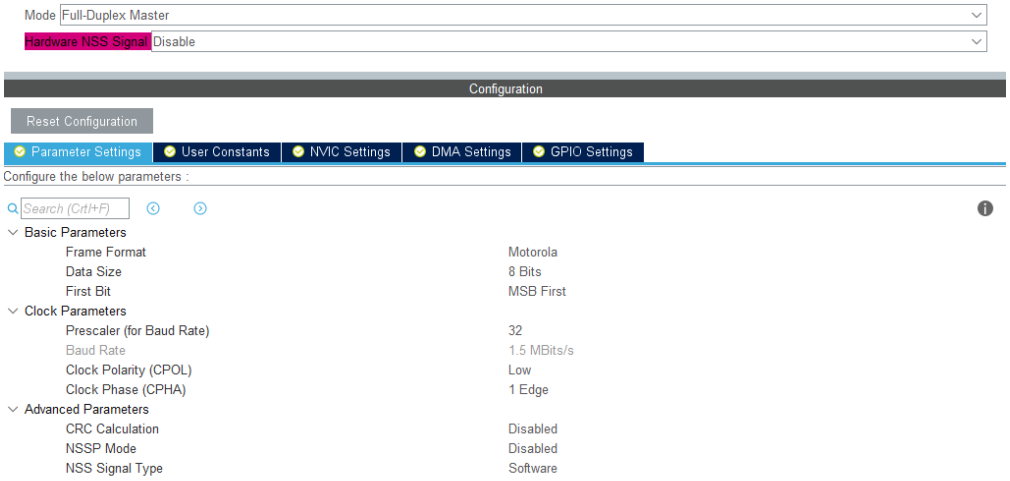


Figure 4.3.7: Remote - SPI Configuration

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pul...	Maximum output ...	Fast Mode	User Label	Modified
PA5	SPI1_SCK	n/a	Alternate Functio...	No pull-up and n...	High	n/a	NRF24L01_SCK	☑
PA6	SPI1_MISO	n/a	Alternate Functio...	No pull-up and n...	High	n/a	NRF24L01_MISO	☑
PA7	SPI1_MOSI	n/a	Alternate Functio...	No pull-up and n...	High	n/a	NRF24L01_MOSI	☑

Figure 4.3.8: Remote - SPI Pin Configuration

4.3.6 I^2C

The I^2C configuration necessary for communicating with the gyroscope and accelerometer is represented in figure 4.3.9. One is using fast mode I^2C with a frequency of $400kHz$. The pin configurations for I^2C are represented in figure 4.3.10.

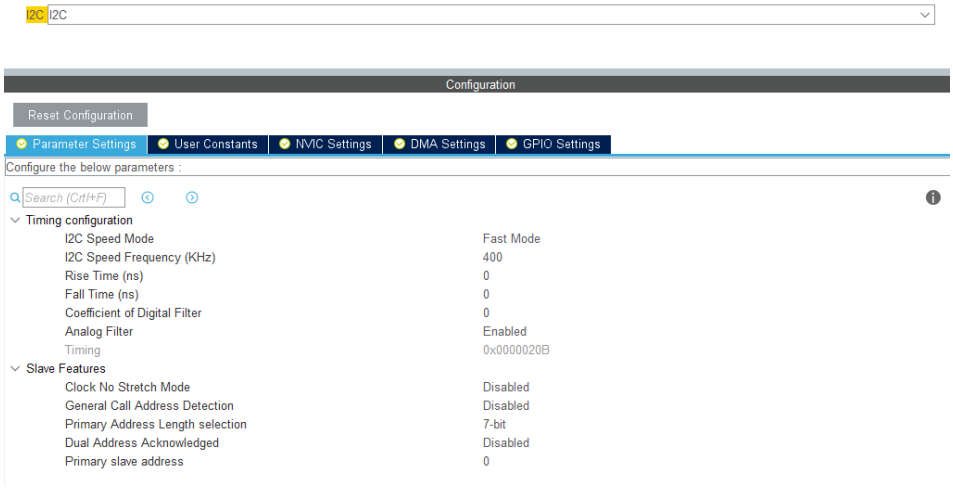


Figure 4.3.9: Remote - I^{2}C Configuration

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pul...	Maximum output ...	Fast Mode	User Label	Modified
PB6	I2C1_SCL	n/a	Alternate Functio...	Pull-up	High	Disable	MPU_6050_SCL	☑
PB7	I2C1_SDA	n/a	Alternate Functio...	Pull-up	High	Disable	MPU_6050_SDA	☑

Figure 4.3.10: Remote - I^{2}C Pin Configuration

4.3.7 ADC

The configuration for the ADC is presented in figure 4.3.11. Note the enabling of the `Low Power Auto Power Off` that allows the ADC to be disabled by default to reduce power consumption.

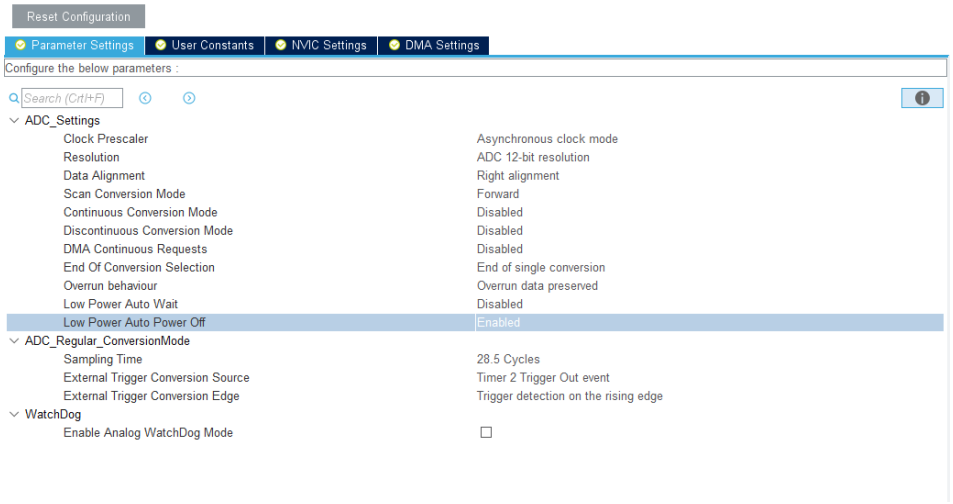


Figure 4.3.11: Remote - I^{2}C Pin Configuration

4.4 Remote System: Timeline Structure

Taking into consideration the more abstract version of the remote system's task timeline, referred in section 3.5.4, one devised a refined version of the latter based on the implementation of the local RTOS system. Just as is the case with the Local System and for the same reason, the name of the gyroscope and accelerometer task changed from zGyroAccManager to zIMUManager.

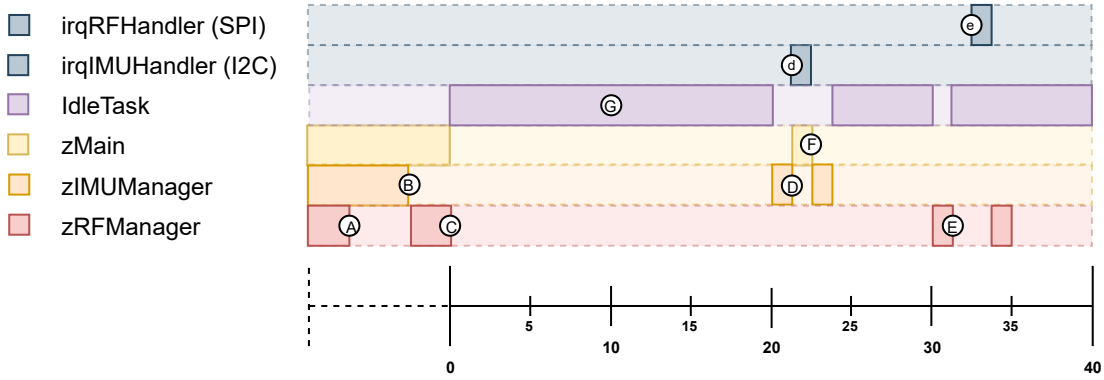


Figure 4.4.1: Local Task Timeline - Refined (Aug. in Appendix C.4)

In the table 4.2 are described all the letter-marked events of the local task timeline of figure 4.2.1.

Event ID	Event Description
Ⓐ	Wait for nConfig
Ⓑ	Send nConfig and await nConnected
Ⓒ	Send nConnected
Ⓓ	IMU sample requested
ⓓ	IMU sample received
Ⓔ	Command sent
ⓔ	Acknowledgement received
Ⓕ	Battery check
Ⓖ	Sleep

Table 4.4: Remote Timeline Events

From event Ⓐ, until ⓓ, inclusive, all of them have the same meaning as in the local system, as explained in section 4.2

Here event Ⓑ symbolizes an action done solely by thread zIMUManager

```
static void THREADS_incSemConfig(void) {

    RTOS_SEMAPHORE_INC(semConfig);

    if (RTOS_SEMAPHORE_GET_COUNT(semConfig) == 3)
        RTOS_NOTIFY(zRFManager, nConfig);
}

RTOS_TASK_FUN(zIMUManager) {

    /*!< Configuration Section */

    /*!< Signal another configuration complete */
    THREADS_incSemConfig();

    /*!< Await the establishment of a connection to continue */
    RTOS_AWAIT(nConnected);

    ...
}
```

Listing 4.9: zIMUManager increments `semConfig` and waits for `nConnected`

The initialization of the nRF24L01+ module is made with the exact same parameters as in the local system, although it goes into TX mode instead. Following that, the remote system systematically and periodically advertises its code and waits for an acknowledgement, after which it waits for a short while for an answer with the car's code. At that point, and if both codes have been accepted by the receiving entities, a connection is considered established.

In event Ⓒ, the main difference in comparison with the Local system is that only zIMUManager is notified of the connection, since there are no other tasks that would benefit from it.

```
        /*!< Notify tasks of a successful connection */
        RTOS_NOTIFY(zIMUManager, nConnected);
```

Listing 4.10: zRFManager notifies zIMUManager of a successful connection with the local

Events Ⓓ and ⓓ happen only once per cycle here, unlike in the local system. This is motivated by the high power consumption of the MPU6050 module when not in standby mode, which would prove too intense for a battery such as the one in the remote system to stand permanently. For that same reason, the module also needs to be put into low power mode every time it is not in used.

```
void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef *hi2c){
    if(hi2c->Instance == I2C1)
        RTOS_NOTIFY_ISR(zIMUManager, nIMURx);
}

void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef *hi2c){
    if(hi2c->Instance == I2C1)
        RTOS_NOTIFY_ISR(zIMUManager, nIMUTx);
}

RTOS_TASK_FUN(zIMUManager) {
    ...

    /*!< Await the establishment of a connection to continue */
    RTOS_AWAIT(nConnected);
    ...
    while(1) {

        /*!< Exit low power mode */
        IMU_exitLowPowerMode(&hi2c1);

        /*!< Request sensor data */
        IMU_dataRequest(&hi2c1);

        /*!< Await reception notification */
        RTOS_AWAIT(nIMURx);

        /*!< Fetch received data */
        IMU_dataFetch(&mpu6050);

        /*!< Enter low power mode */
        IMU_enterLowPowerMode(&hi2c1);

        ...
    }

    ...
}
```

Listing 4.11: IMU data request and data fetch

The events Ⓔ and ⓔ represents a command being sent. The command will be repeatedby the nRF24L01+ chip up to a total of 15 re-transmissions, separated by 250μs until an acknowledgement is received. Whether the command reaches the destination successfully until the specified time ends or not, a notification will be received, that specifies the result. If no acknowledgement is received, it means that either the car is not available for communication or synchronism is lost. In any case, the result is signaled to

the user. All of this can be seen in listing 4.12

```
RTOS_TASK_FUN(zRFManager) {

    ...

    while(1) {

        /*!< Get imu readings */
        RTOS_QUEUE_RECV_TIMEOUT(qIMU, meas_angle, \
            Z_RF_QUEUE_TIMEOUT);
        RTOS_QUEUE_RECV_TIMEOUT(qIMU, meas_angle+1, \
            Z_RF_QUEUE_TIMEOUT);

        /*!< Wake nRF24L01 up and request to send data */
        RF_Transmit((uint8_t*)meas_angle);

        /*!< Await ACK */
        RTOS_AWAIT_TIMEOUT(nRF, 5);

        /*!< Read interrupt flags for DataSent */
        RF_Read_Interrupts(&nrf_irq);

        /*!< Announce connection problem if communication was
            unsuccessful */
        if (nrf_irq.F.DataSent)
            SET_BIT(LED_CONN_PROB_GPIO_Port->BSRR, \
                (uint32_t)LED_CONN_PROB_Pin<<16);
        else
            SET_BIT(LED_CONN_PROB_GPIO_Port->BSRR, \
                LED_CONN_PROB_Pin);

        ...

        /*!< Send nRF24L01 into sleep mode */
        RF_PowerDown();

        ...

        }

    }
```

Listing 4.12: RF command sending

The event Ⓕ represents the battery check process, wherein, after a certain period has elapsed, the controller checks the voltage measured in the battery, calculates the charge level in it and outputs a warning representing an appropriate or low battery level.

```
RTOS_TASK_FUN(zMain) {

    ...

    /*!< Configure the ADC and start conversion */

    while(1) {

        /*!< If a conversion is complete, interpret it */
        if (READ_BIT(ADC1->ISR, ADC_FLAG_EOC)) {

            /*!< Clear EOC flag */
            CLEAR_BIT(ADC1->ISR, ADC_FLAG_EOC);

            /*!< Read battery voltage and change led indicator \
             *   accordingly */
            if (THREADS_checkForLowBattery())
                WRITE_REG(TIM2->CCR2, POWER_LED_PULSE_LOW_BATT);
            else
                WRITE_REG(TIM2->CCR2, POWER_LED_PULSE_NORMAL);
        }

        ...
    }

...
}

...

static double THREADS_checkForLowBattery(void) {

#define EVAL_LOW            1
#define EVAL_HIGH           0
#define VBATT_MAX           3300
#define VBATT_MIN           2000
#define VBATT_EXCURSION     (VBATT_MAX-VBATT_MIN)
#define LOW_BATT_THRSLD     15
#define HYSTERESIS          2
#define VREFINT_CAL_ADDR    ((uint16_t*)((uint32_t)0x1FFFF7BA))

    /*!< Evaluation from last iteration */
    static uint32_t last_evaluation = EVAL_HIGH;
    uint32_t evaluation;  /*!< Evaluation to be made */
    int32_t adc_value;    /*!< Raw ADC reading */
    int32_t v_batt;       /*!< Battery voltage */
    int32_t batt_lvl;     /*!< Calculated battery level */
```

```
    /*!< Fetch ADC conversion value */
    adc_value = READ_REG(ADC1->DR);

    /*!< Calculate real battery voltage
     *   VDDA = 3.3 V x VREFIN_CAL / VREFINT_DATA */
    v_batt = 3300*(*VREFINT_CAL_ADDR)/adc_value;

    /*!< Extrapolate 0-100 battery level assuming linear
     *   voltage drop */
    batt_lvl = 100 * (v_batt - VBATT_MIN) / VBATT_EXCURSION;

    /*!< If last evaluation was LOW, go to HIGH only once the
     *   hysteresis margin has been overcome in the UP direction
     *   If last evaluation was HIGH, go to LOW only once the
     *   hysteresis margin has been overcome in the DOWN
     *   direction */
    if (last_evaluation == EVAL_LOW)
        evaluation = batt_lvl < LOW_BATT_THRSLD + HYSTERESIS;
    else
        evaluation = (batt_lvl <= LOW_BATT_THRSLD - HYSTERESIS);

    /*!< Keep calculated value for future reference */
    last_evaluation = evaluation;

    return evaluation;
}
```

Listing 4.13: Battery level check mechanism

The event Ⓖ represents the system's sleep state activated in the idle task. This state is similar to the one presented in event Ⓘ of the local task timeline (4.2.1).

4.5 Remote System: Power Consumption Simulations

After choosing all of the hardware and planning a complete strategy for the software implementation, a set of power consumption simulations were run to find the best profile for this application. This was done using STMCubeMX's Power Consumption Calculator (PCC) tool.

The best profile tested included running the peripherals with a clock frequency of $8MHz$ and the CPU at a frequency of $16MHz$ when not in sleep mode. All the peripherals would be kept running at all times except for the ADC, which would only be turned on when a conversion was needed and turned off right after. The external hardware peripherals would also be put in their respective sleep modes when not in use. The parameter table is shown in the table in figure 4.5.3 and the result is plotted in figure 4.5.2. This would give a wort-case scenario estimated battery life of 45h, with an average

estimated current draw of $4.92mA$ at $3.3V$, for all the hardware in the board. An ideal solution would be an average power consumption closer to $3mA$, but this is a worst-case scenario prediction and can be further improved. In 2 second intervals, an ADC conversion occurs which increases the average power consumption of the whole system by an estimated $0.06mA$ in that cycle.

Step	Mode	Vdd	Range/Scale	Memory	CPU/Bus...	Clock Co...	Peripherals	Step Current	Duration
1	RUN	3.3	No Scale	FLASH	8 MHz	HSI	ADC GPIOA GPIOB G...	6.32 mA	5 µs
2	SLEEP	3.3	No Scale	RAM/FLASH	8 MHz	HSI	GPIOA GPIOB GPIOF...	3.28 mA	4.995 ms
3	RUN	3.3	No Scale	FLASH	16 MHz	HSE PLL	GPIOA GPIOB GPIOF...	8.14 mA	500 µs
4	SLEEP	3.3	No Scale	RAM/FLASH	8 MHz	HSI	GPIOA GPIOB GPIOF...	3.28 mA	14.5 ms
5	RUN	3.3	No Scale	FLASH	16 MHz	HSE PLL	GPIOA GPIOB GPIOF...	12.64 mA	1 ms
6	RUN	3.3	No Scale	FLASH	16 MHz	HSE PLL	GPIOA GPIOB GPIOF...	8.14 mA	1 ms
7	SLEEP	3.3	No Scale	RAM/FLASH	8 MHz	HSI	GPIOA GPIOB GPIOF...	3.28 mA	8 ms
8	SLEEP	3.3	No Scale	RAM/FLASH	8 MHz	HSI	GPIOA GPIOB GPIOF...	16.08 mA	4 ms

Figure 4.5.1: PCC Power Consumption Simulations Goal: Table of Parameters

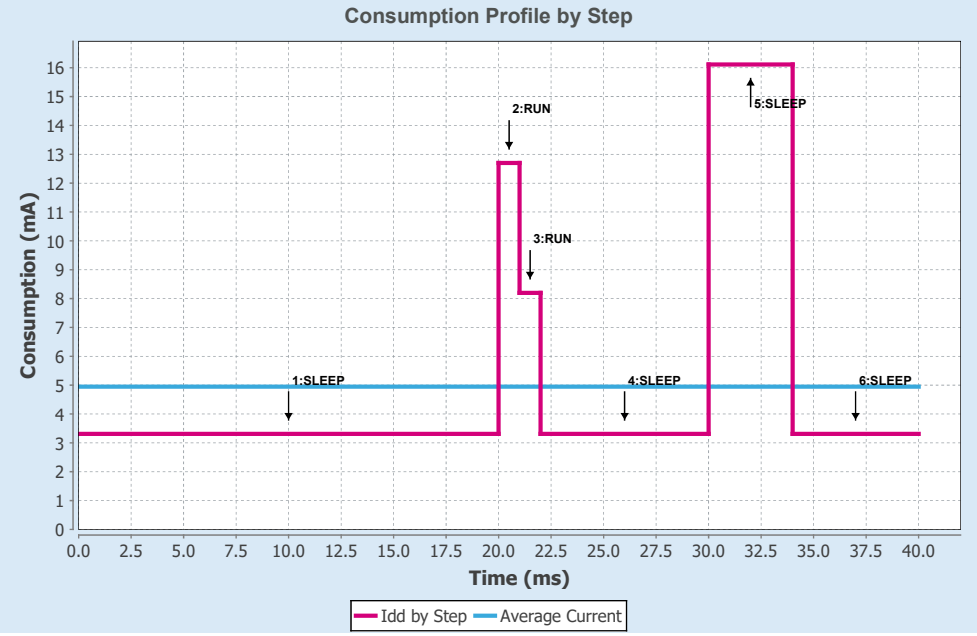


Figure 4.5.2: PCC Power Consumption Simulations Goal: Plot

However, the solution could not be fully implemented, lacking the clock frequency reduction feature, which could not be properly tested in time. Everything else was

successfully implemented, though, resulting in a profile for which the simulation is plotted in figure 4.5.4. This would give an estimated battery life of 21h, with an average estimated current draw of $10.58mA$ at $3.3V$,

Step	Mode	Vdd	Range/Scale	Memory	CPU/Bus...	Clock Con...	Peripherals	Step Current	Duration
1	SLEEP	3.3	No Scale	RAM/FLASH	48 MHz	HSI PLL	ADC GPIOA GPIOB G...	9.93 mA	20 ms
2	RUN	3.3	No Scale	FLASH	48 MHz	HSI PLL	ADC GPIOA GPIOB G...	22.43 mA	1 ms
3	RUN	3.3	No Scale	FLASH	48 MHz	HSI PLL	ADC GPIOA GPIOB G...	17.93 mA	1 ms
4	SLEEP	3.3	No Scale	RAM/FLASH	48 MHz	HSI PLL	ADC GPIOA GPIOB G...	9.93 mA	8 ms
5	SLEEP	3.3	No Scale	RAM/FLASH	48 MHz	HSI PLL	ADC GPIOA GPIOB G...	22.73 mA	4 ms
6	SLEEP	3.3	No Scale	RAM/FLASH	48 MHz	HSI PLL	ADC GPIOA GPIOB G...	9.93 mA	6 ms

Figure 4.5.3: PCC Power Consumption Simulations Realistic: Table of Parameters

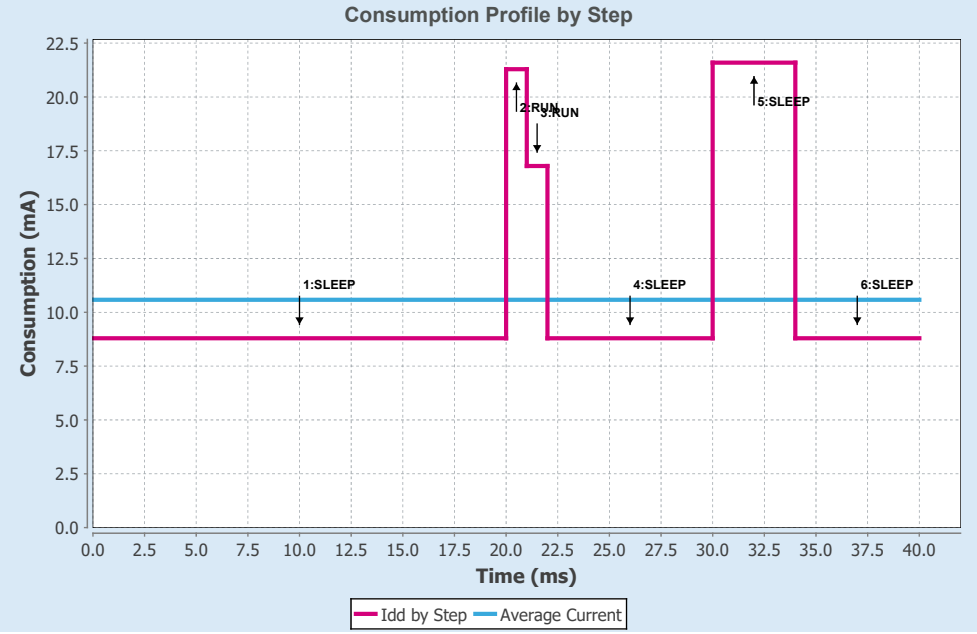


Figure 4.5.4: PCC Power Consumption Simulations Realistic: Plot

4.6 Idle Task Hook

As mentioned before, part of the power saving techniques used in the application for both systems Zephyrus is a sleeping mechanism that will trigger when all tasks are suspended. The simplest way to implement this is through the idle task that FreeRTOS provides and guarantees is always running. Without writing code on top of what the kernel provides, one can implement a callback function that is called in every cycle of the idle task. This function has the prototype `void vApplicationIdleHook(void)` and will be implemented in Zephyrus to put the CPU core to sleep. This way it is guaranteed to only happen when no user task is executing and stop when they are.

In the remote system, it is also used to pull the power control line low when all the user tasks have responded to the termination signal

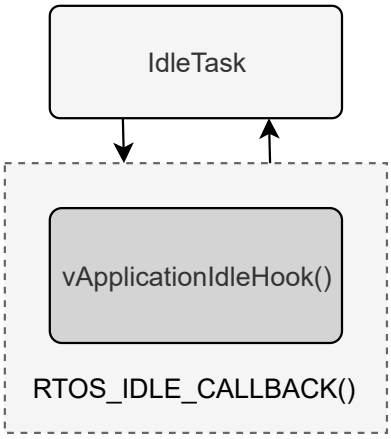


Figure 4.6.1: Idle Task Hook

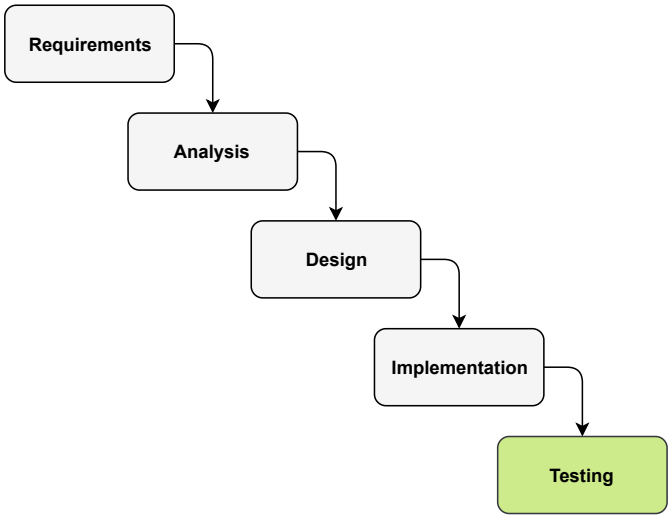
```
/* In FreeRTOSConfig.h */
#define configUSE_IDLE_HOOK 1s
/* Local */
RTOS_IDLE_CALLBACK() {
    THREADS_sleep();
}
/* Remote */
RTOS_IDLE_CALLBACK() {

        /*!< When all tasks have terminated, shut down */
    if (RTOS_SEMAPHORE_GET_COUNT(semTerminate) == \
            RTOS_SEMAPHORE_MAX_COUNT(semTerminate))
        PWR_CLR_PULL_LOW();

    THREADS_sleep();
}
```

Listing 4.14: Idle Task Callback

Chapter 5

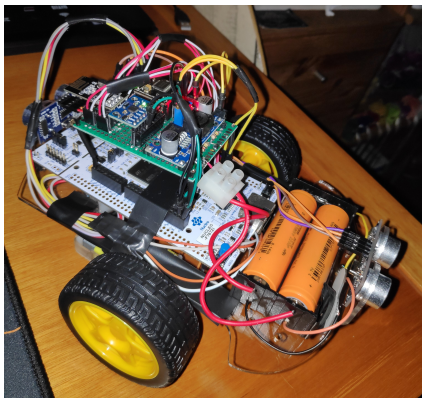
Testing

This phase will present the results of the unit and integration tests performed on each Zephyrus subsystem. Hence, the expected results will be compared to the real results to gain insight into what could be improved in the future through a verification stage.

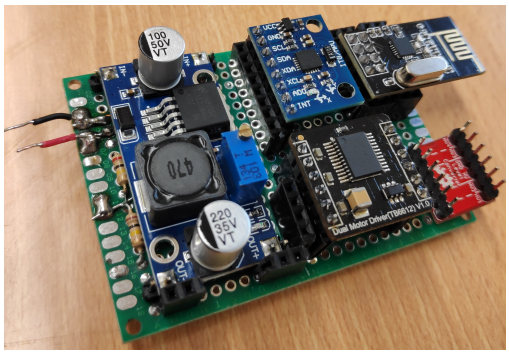


5.1 Local System: Prototype Alpha

Firstly, the Local System prototype implementation will be described for both hardware and software. This Zephyrus system only consists of one prototype, that is the alpha one (implicit), and so it will be referred to as such.



(a) Remote Car



(b) Modules

Figure 5.1.1: Local System Alpha Prototype

5.2 Local System: Test Setup and Results

As specified in section 3.12, some alpha tests were conducted on the local system.

5.2.1 Unit Tests

The unit tests performed were the following:

1. Inference tests;
2. `nRF24L01+` tests;
3. `MPU-6050` tests;
4. `HC-SR04` tests;
5. Task synchronism tests;

Inference Tests

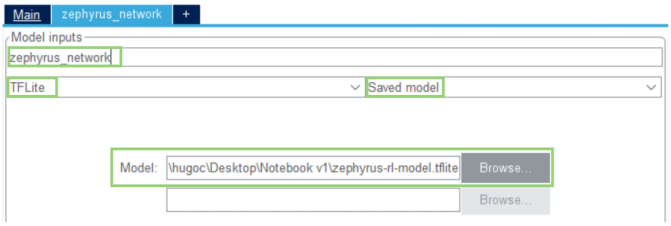


Figure 5.2.1: Model Import

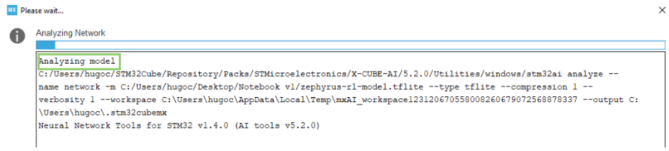


Figure 5.2.2: Model Analysys

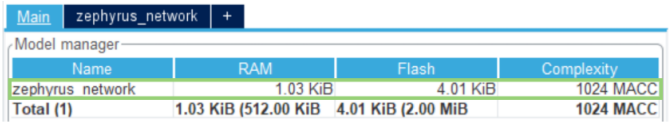


Figure 5.2.3: Model Size (RAM/Flash)

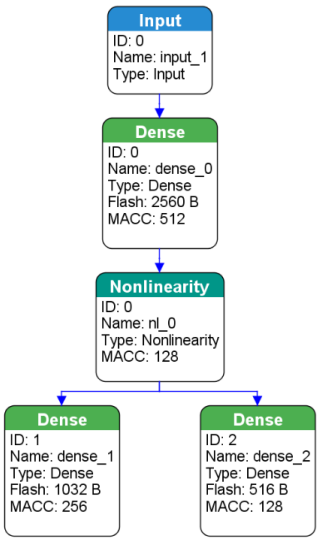
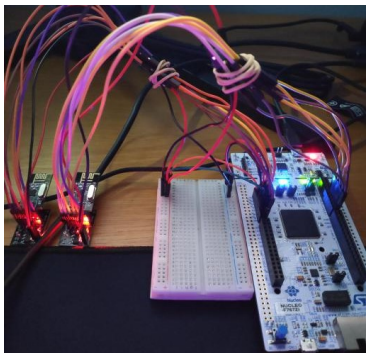
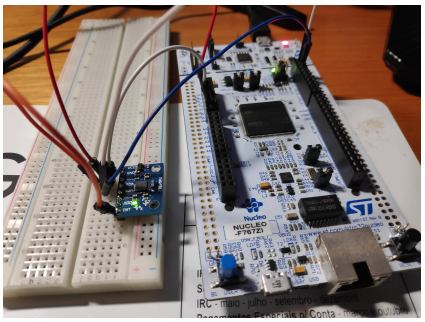


Figure 5.2.4: Zephyrus Network

Module Tests



(a) Local - `nRF24L01+` Tests



(b) Local - `MPU-6050` Tests

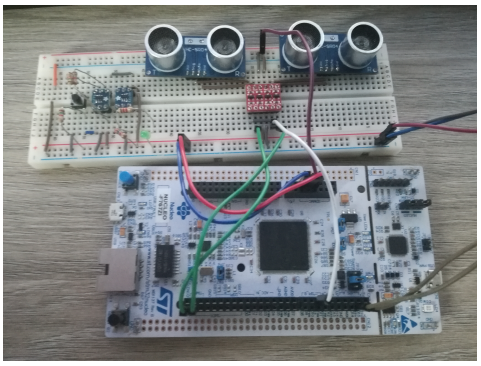


Figure 5.2.6: Local - `HC-SR04` Tests

Task Synchronism Tests

For the task synchronism tests, one verified if the local RTOS system respected the guidelines settled in the timeline of figure 4.2.1.

5.3 Remote System Prototypes

5.3.1 Prototype Alpha

Initially an alpha prototype was constructed using all the modules that constitute the remote system. Note that, for the controller section a custom STM32F0 development board was used (section 5.3.1).

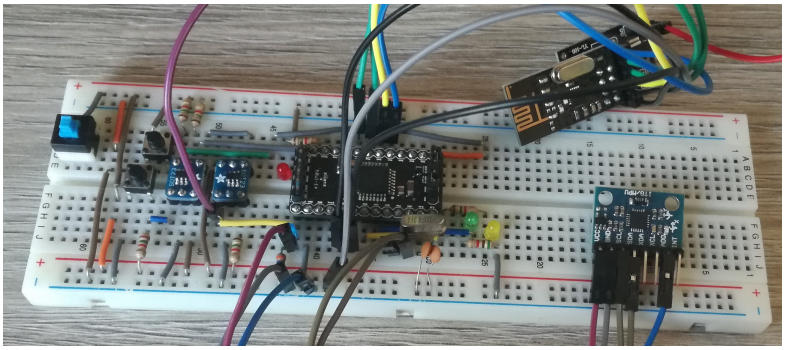


Figure 5.3.1: Remote System Alpha Prototype

STM32F0 Development Board

As mentioned before in section 3.8.1, a custom solution was devised so one could expose the STM32F0 pinout. The implementation of the latter with all the components soldered is represented in figure 5.3.2.

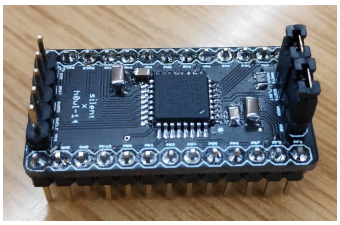


Figure 5.3.2: Custom STM32F0 Development Board

5.3.2 Prototype Beta

The beta prototype for the remote system is depicted in figure 5.3.3.

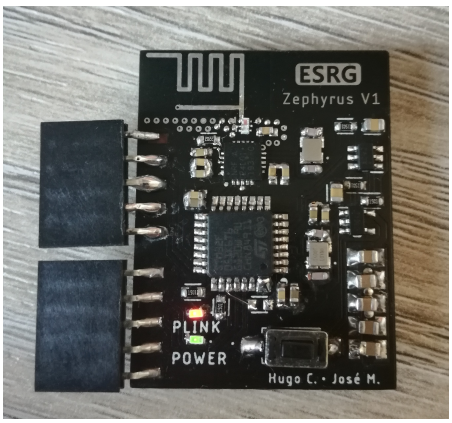


Figure 5.3.3: Remote System Beta Prototype

5.4 Remote System: Test Setup and Results

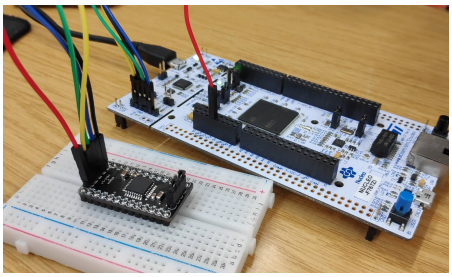
As specified in section 3.13.1, several alpha and beta tests were conducted on the remote system.

5.4.1 Alpha: Unit Tests

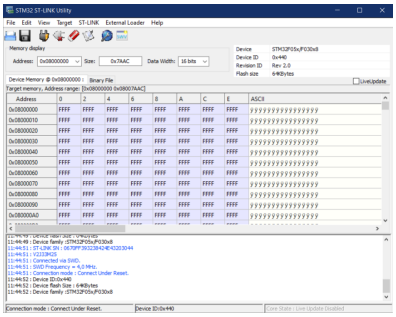
The alpha unit tests performed were the following:

1. STM32F0 custom development board tests;
2. `nRF24L01+` tests;
3. `MPU-6050` tests;
4. Power on/off sequence tests;
5. Task synchronism tests;

STM32F0 Custom Development Board Tests



(a) Test Setup



(b) Flash Read Test

`nRF24L01+` Tests

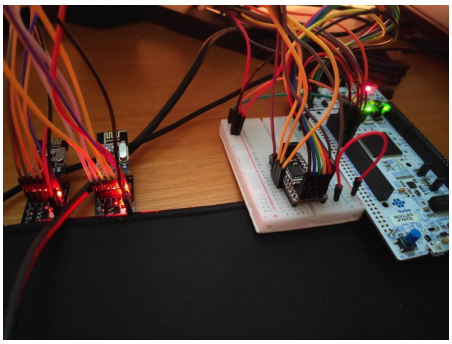


Figure 5.4.2: Remote - `nRF24L01+` Tests

`MPU-6050` Tests

For the `MPU-6050` tests, one verified if the remote alpha could configure the IMU and access the intended values.

Power On/Off Sequence Tests

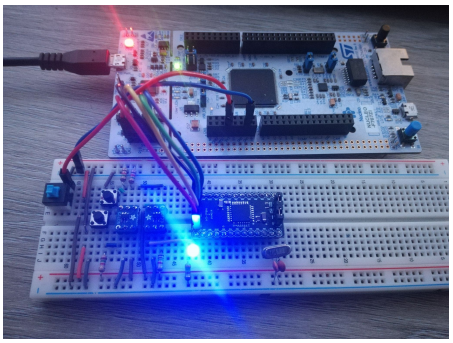


Figure 5.4.3: Remote - Power On/Off Sequence Tests

Task Synchronism Tests

For the task synchronism tests, one verified if the remote alpha RTOS system respected the guidelines settled in the timeline of figure 4.4.1.

5.4.2 Beta: Unit Tests

The beta unit tests performed were the following:

1. Wristband board tests;
2. `nRF24L01+` tests;
3. `MPU-6050` tests;
4. Power on/off sequence tests;
5. Task synchronism tests;

Wristband Board Tests

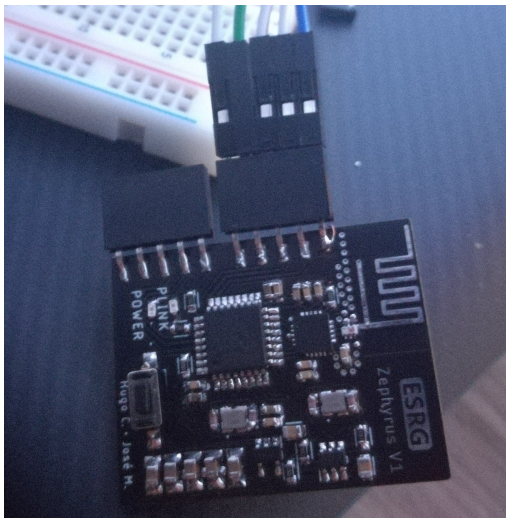


Figure 5.4.4: Remote - Wristband board Tests

`nRF24L01+` Tests

For the `nRF24L01+` tests, one verified if the remote alpha could configure the and exchange messages with the local system.

`MPU-6050` Tests

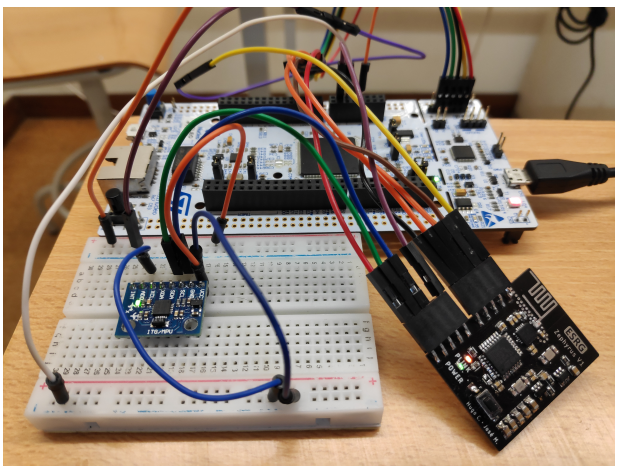
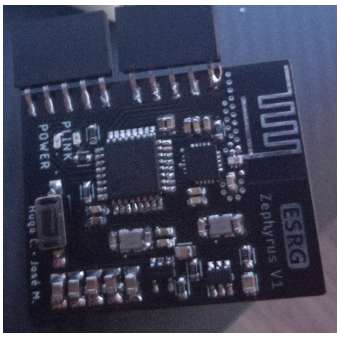
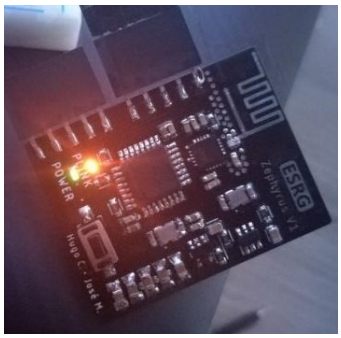


Figure 5.4.5: Remote Beta - `MPU-6050` Tests

Power On/Off Sequence Tests



(a) Remote Beta - Power Off



(b) Remote Beta - Power On

Figure 5.4.6: Remote Beta - Power On/Off Sequence Tests

Task Synchronism Tests

For the task synchronism tests, one verified if the remote beta RTOS system respected the guidelines settled in the timeline of figure 4.4.1.

5.4.3 Alpha: Integration Tests

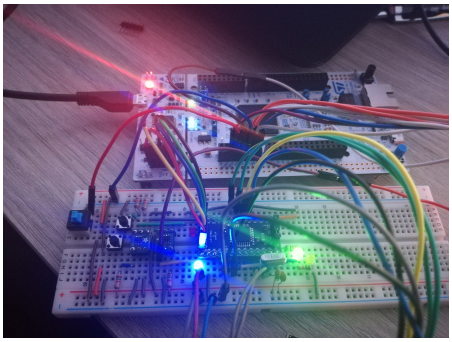


Figure 5.4.7: Alpha Integrated Test

5.4.4 Beta: Integration Tests

The integration tests for the remote beta prototype failed most of the times, likely because of the antenna filter, more precisely because of the soldering work.

5.5 Verification

At this stage one compares the expected test results with the actual results in order to identify points to improve in the next chapter.

5.6 Local Test Cases

5.6.1 Local Unit Tests

Local: Unit Tests	**Expected Results**	**Real Results**
Run inference	Inference successful	Inference unsuccessful
Configure nRF24L01+ SPI	Configuration successful and status read	Configuration successful and status read
Configure MPU-6050	Configuration successful and status read	Configuration successful and status read
Make ultrasonic sensor readings	Able to read	Able to read
Verify task synchronism	Synchronized and timely correct tasks	Synchronized and timely correct tasks

Table 5.1: Local Unit Test Specification

5.7 Remote Test Cases

5.7.1 Remote: Prototype Alpha Unit Tests

Remote: Prototype Alpha Unit Tests	**Expected Results**	**Real Results**
Test STM32F051 dev board PCB continuity	No unwanted continuities	No unwanted continuities
Try to read STM32F051 memory w/ flash read utility	Read flash	Read flash
Configure nRF24L01+ SPI	Configuration successful and status read	Configuration successful and status read
Configure MPU-6050	Configuration successful and status read	Configuration successful and status read
Press the power button while the device is off	Successful power on sequence	Successful power on sequence
Press the power button while the device is on	Successful power off sequence	Successful power off sequence
Verify LEDs responsiveness and current draw	Able to control the LEDs and get a current draw within the specified limits	Able to control the LEDs and get a current draw within the specified limits
Verify task synchronism	Synchronized and timely correct tasks	Synchronized and timely correct tasks

Table 5.2: Prototype Alpha Unit Tests Specification

5.7.2 Remote: Prototype Beta Unit Tests

Remote: Prototype Beta Unit Tests	**Expected Results**	**Real Results**
Test wristband PCB continuity	No unwanted continuities	No unwanted continuities
Try to read STM32F051 memory w/ flash read utility	Read flash	Read flash
Configure nRF24L01+ SPI	Configuration successful and status read	Configuration successful and status read
Configure MPU-6050	Configuration successful and status read	Configuration successful and status read
Press the power button while the device is off	Successful power on sequence	Successful power on sequence
Press the power button while the device is on	Successful power off sequence	Successful power off sequence
Verify LEDs responsiveness and current draw	Able to control the LEDs and get a current draw within the specified limits	Able to control the LEDs and get a current draw within the specified limits
Verify task synchronism	Synchronized and timely correct tasks	Synchronized and timely correct tasks

Table 5.3: Prototype Beta Unit Tests Specification

5.8 Integration Tests

5.8.1 Prototype Alpha Integration Tests

Prototype Alpha Integration Tests	**Expected Results**	**Real Results**
Verify connection	Connection successful	Connection successful
Make the car follow a predetermined path	Car follows the instructions	Not tested
Verify object detection	Car stops after the detection of an obstacle	Not tested
Press the power button while the device is off	Successful power on sequence	Successful power on sequence
Press the power button while the device is on	Successful power off sequence	Successful power off sequence

Table 5.4: Prototype Alpha Integration Tests Specification

5.8.2 Prototype Beta Integration Tests

Prototype Beta Integration Tests	**Expected Results**	**Real Results**
Verify connection	Connection successful	Irregular connection
Make the car follow a predetermined path	Car follows the instructions	Not tested
Verify object detection	Car stops after the detection of an obstacle	Not tested
Press the power button while the device is off	Successful power on sequence	Successful power on sequence
Press the power button while the device is on	Successful power off sequence	Successful power off sequence

Table 5.5: Prototype Beta Integration Tests Specification

Chapter 6

Conclusion

6.1 Future Works

Although the group is left satisfied with the state of the project at the time of delivery, the work is not yet concluded until the project works just as it was initially intended. Thus, to make the product more appealing, some work could certainly be done in the future.

Arrive to a working reinforcement learning model As the use the use of Reinforcement Learning is a technical constraint of the project, it is perhaps the most important feature to get working first. The fact that one of the most important features of the project is associated with it makes it all the more urgent.

Synchronism between the local and remote systems One of the problems identified during the implementation of this project was the fact that, as of the time of delivery of the project, the difference between the time when the local system is ready to receive a command and the time when the remote system is ready to send it varies wildly, always in the same direction. This pertains to one of the most important features of the project.

Enclosure for the wristband As there was a need to push less relevant features of the project back in favour of more important ones, the enclosure for the wristband was left undone. Although this doesn't break the functionality of the prototype, it is important for the finished product to have a proper interface and a shield for the board.

Get the radio working consistently As of the time of delivery, the radio for the remote system is working inconsistently. Although it was already figured to be hard to get working consistently with the previously acquired experience, it was a necessary condition for a final working product.

Further power consumption optimizations In both systems, especially the wristband, good efforts were done to achieve good power consumption figures. However, both still have work that could be done to improve them, perhaps through the disabling of certain power-hungry peripherals and use of other power-saving ones like the DMA.

Bibliography

[1] Branco, S., Ferreira, A. and Cabral, J., 2019. Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey. Electronics, 8(11), p.1289.

[2] Sommerville, I., 2011. Software Engineering. Boston: Pearson.

[3] STMicroelectronics. 2020. X-CUBE-AI - Stmicroelectronics. [online] Available at: <https://www.st.com/en/embedded-software/x-cube-ai.html#overview> [Accessed 30 October 2020].

[4] TensorFlow. 2020. Tensorflow Lite For Microcontrollers. [online] Available at: <https://www.tensorflow.org/lite/microcontrollers> [Accessed 31 October 2020].

[5] Ltd., A., 2020. Cortex-M – Arm Developer. [online] Arm Developer. Available at: <https://developer.arm.com/ip-products/processors/cortex-m> [Accessed 31 October 2020].

[6] Mathworks.com. 2020. What Is Reinforcement Learning?- MATLAB & Simulink. [online] Available at: <https://www.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html> [Accessed 31 October 2020].

[7] DigiKey. 2020. Tinyml: Getting Started With STM32 X-CUBE-AI. [online] Available at: <https://www.digikey.com/en/maker/projects/tinyml-getting-started-with-stm32-x-cube-ai/f94e1c8bfc1e4b6291d0f672d780d2c0> [Accessed 31 October 2020].

[8] Sakr, F., Bellotti, F., Berta, R. and De Gloria, A., 2020. Machine Learning on Mainstream Microcontrollers. Sensors, 20(9), p.2638.

[9] Lonza, A., 2019. Reinforcement Learning Algorithms With Python. Birmingham: Packt Publishing, Limited.

[10] St.com. 2020. X-Cube.AI User Manual: Getting started with X-CUBE-AI Expansion Package for Artificial Intelligence (AI). [online] Available at: <https://www.st.com/en/embedded-software/x-cube-ai.html#documentation> [Accessed 31 October 2020].

[11] Mordorintelligence.com. 2020. Gesture Recognition Market - Growth, Trends and Forecast (2020 - 2025) [online] Available at: <https://www.mordorintelligence.com/industry-reports/gesture-recognition-changing-the-way-humans-interact-with-devices-industry> [Accessed 2 November 2020].

[12] Adarsh, S., Kaleemuddin, S., Bose, D. and Ramachandran, K., 2016. Performance comparison of Infrared and Ultrasonic sensors for obstacles of different materials in vehicle/ robot navigation applications. IOP Conference Series: Materials Science and Engineering, [online] 149, p.012141. Available at: <https://www.researchgate.net/publication/309001685_Performance_comparison_of_Infrared_and_Ultrasonic_sensors_for_obstacles_of_different_materials_in_vehicle_robot_navigation_applications> [Accessed 22 November 2020].

[13] 2020. MPU-6000 And MPU-6050 Product Specification Revision 3.1. [ebook] Available at: <http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Components/General%20IC/PS-MPU-6000A.pdf> [Accessed 22 November 2020].

[14] 2018. Lithium Coin Handbook And Application Manual. [ebook] Energizer. Available at: <https://data.energizer.com/pdfs/lithiumcoin_appman.pdf> [Accessed 23 November 2020].

[15] 2020. MAX16150 Nanopower Pushbutton On/Off Controller And Battery Freshness Seal. 3rd ed. [ebook] Maxim Integrated. Available at: <https://datasheets.maximintegrated.com/en/ds/MAX16150.pdf> [Accessed 24 November 2020].

[16] 2016. AP22814 Single Channel Power Distribution Load Switch. 2nd ed. [ebook] Diodes Incorporated. Available at: <https://www.diodes.com/assets/Datasheets/AP22804_14.pdf> [Accessed 9th January 2021].

[17] Dhaker, P., 2018. Introduction To SPI Interface. [ebook] Analog Devices. Available at: <https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf> [Accessed 24 November 2020].

[18] Afzal, S., 2020. I2C Primer: What Is I2C? (Part 1) | Analog Devices. [online] Analog.com. Available at: <https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html> [Accessed 26 November 2020].

[19] TensorFlow. 2020. Playing Cartpole With The Actor-Critic Method | Tensorflow Core. [online] Available at: <https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic> [Accessed 5 December 2020].

[20] GitHub. 2020. Arduinolibrary Dfrobot 2X1.2A DC Motor Driver - TB6612FNG. [online] Available at: <https://github.com/Arduinolibrary/DFRobot_2x1.2A_DC_Motor_Driver_TB6612FNG/blob/master/Dual%20Motor%20Driver(TB6612)(V1.0).PDF> [Accessed 5 December 2020].

[21] Mouser. 2021. Low Power MCU Design. [online] Available at: <https://eu.mouser.com/applications/low-power-ewc-overview/> [Accessed 1 January 2021].

[22] Pattnayak, T. and Thanikachalam, G., 2014. Antenna Design And RF Layout Guidelines. 9th ed. [ebook] Cypress Semiconductor. Available at: <https://www.cypress.com/file/136236/download> [Accessed 9 January 2021].

[23] 2014. High-Speed Interface Layout Guidelines. 9th ed. [ebook] Texas Instruments. Available at: <https://www.ti.com/lit/an/spraar7h/spraar7h.pdf?ts=1610212863527> [Accessed 9 January 2021].

[24] 2009. Johanson Technology, Inc. 2450BM14A0002 Matched Balun For The Nordic Nrf24l01/Nrf24l01+ Chipsets. 1st ed. [ebook] Johanson Technology. Available at: <https://www.johansontechnology.com/downloads/technotes/Johanson-2450BM14A0002-App-Note-for-nRF24L01+.pdf> [Accessed 9 January 2021].

[25] W. Lindseth, "Effectiveness of PCB perimeter Via fencing: Radially propagating EMC emissions reduction technique," 2016 IEEE International Symposium on Electromagnetic Compatibility (EMC), Ottawa, ON, 2016, pp. 627-632, doi: 10.1109/ISEMC.2016.7571721.

[26] jlcpcb.com. 2021. JLCPCB - PCB Capabilities. [online] Available at: <https://jlcpcb.com/capabilities/Capabilities> [Accessed 10 January 2021].

[27] 2007. Nrf24l01 Single Chip 2.4Ghz Transceiver Product Specification. 2nd ed. [ebook] Nordic Semiconductor. Available at: <https://www.mouser.com/datasheet/2/297/nRF24L01_Product_Specification_v2_0-9199.pdf> [Accessed 10 January 2021].

[28] 2018. Tiniest Matched Baluns For Wifi, Bluetooth Sub-Ghz. 1st ed. [ebook] ST Microelectronics. Available at: <https://www.st.com/content/ccc/resource/sales_and_marketing/promotional_material/flyer/group0/25/09/bc/ab/d6/ac/45/1f/FLBALUN1018/files/FLBALUN1018.pdf/jcr:content/translations/en.FLBALUN1018.pdf> [Accessed 10 January 2021].

[29] Ribeiro, M. and Lima, P., 2002. KINEMATICS MODELS OF MOBILE ROBOTS.

Appendices

Appendix A

Augmented Figures

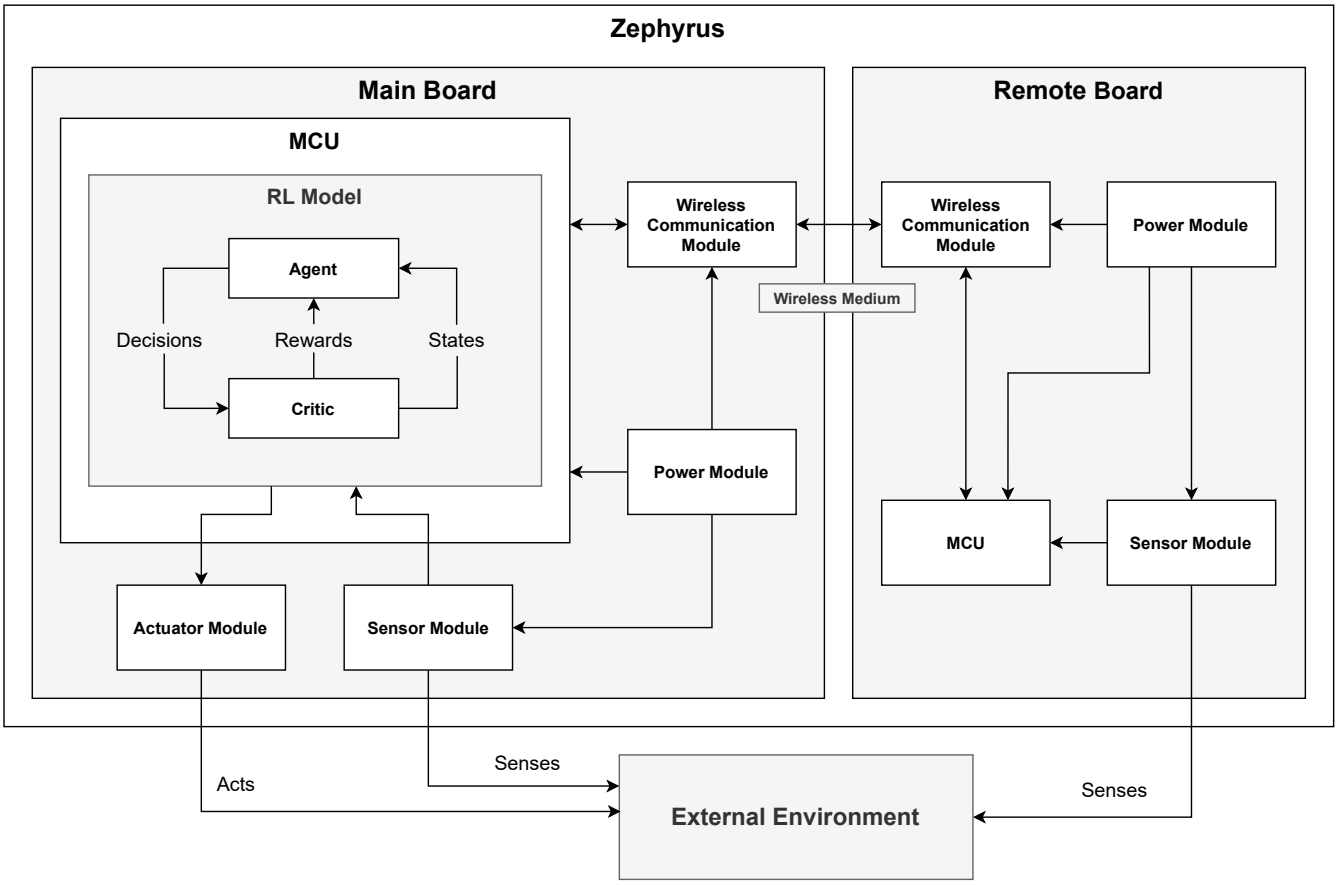


Figure A.1: System Overview Diagram Augmented

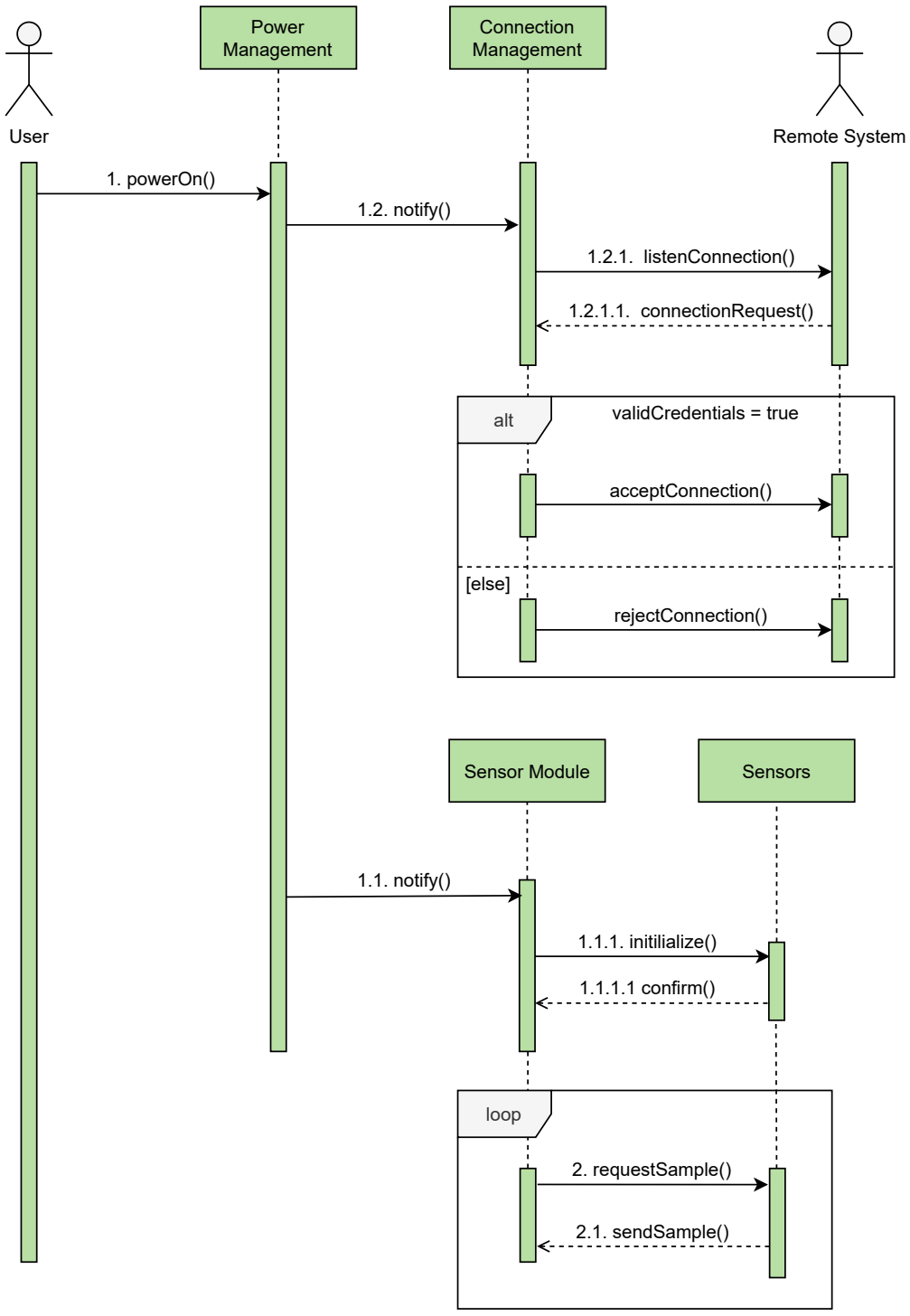


Figure A.2: Local System Sequence Diagram Augmented

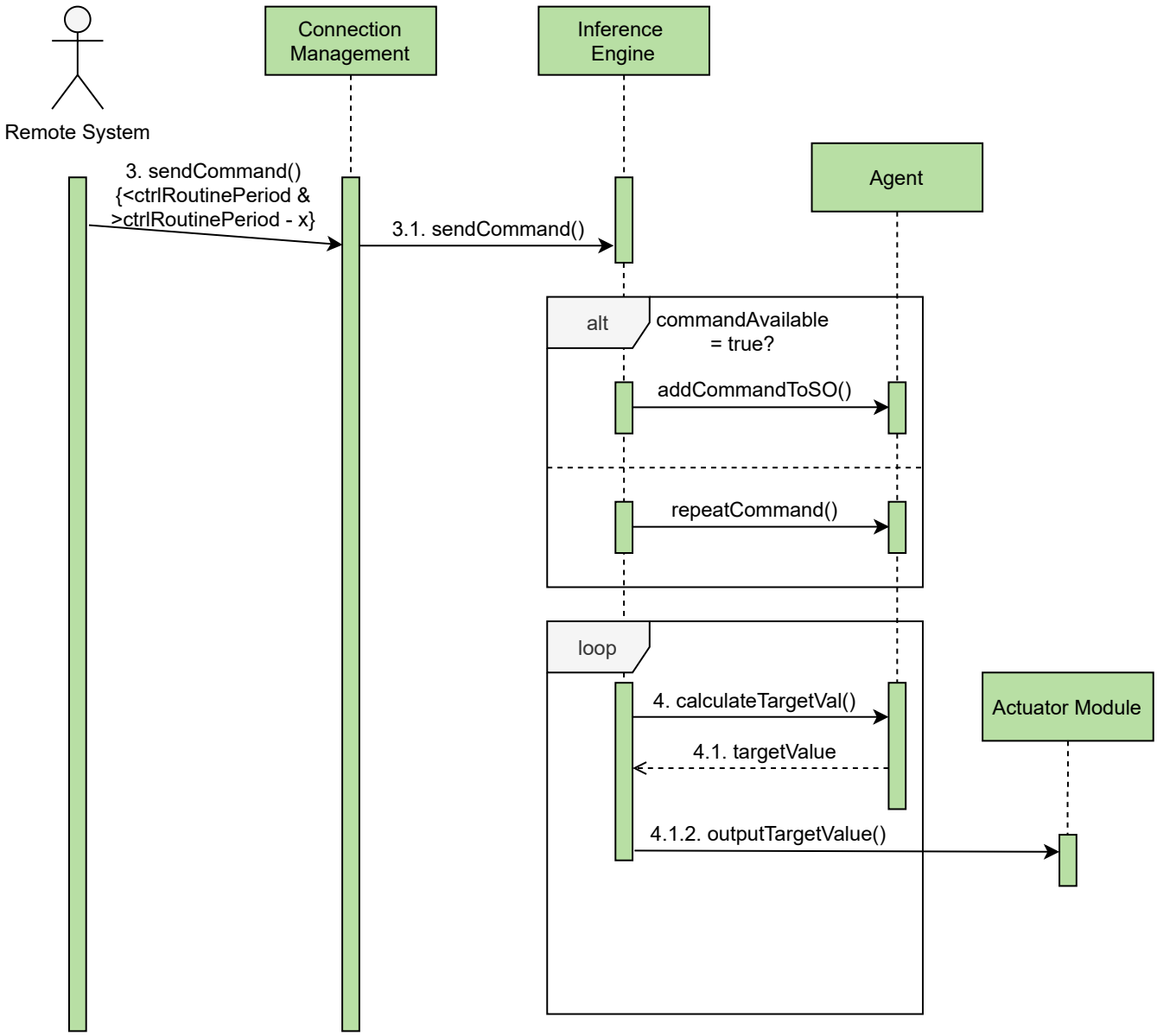


Figure A.3: Local System Sequence Diagram Augmented

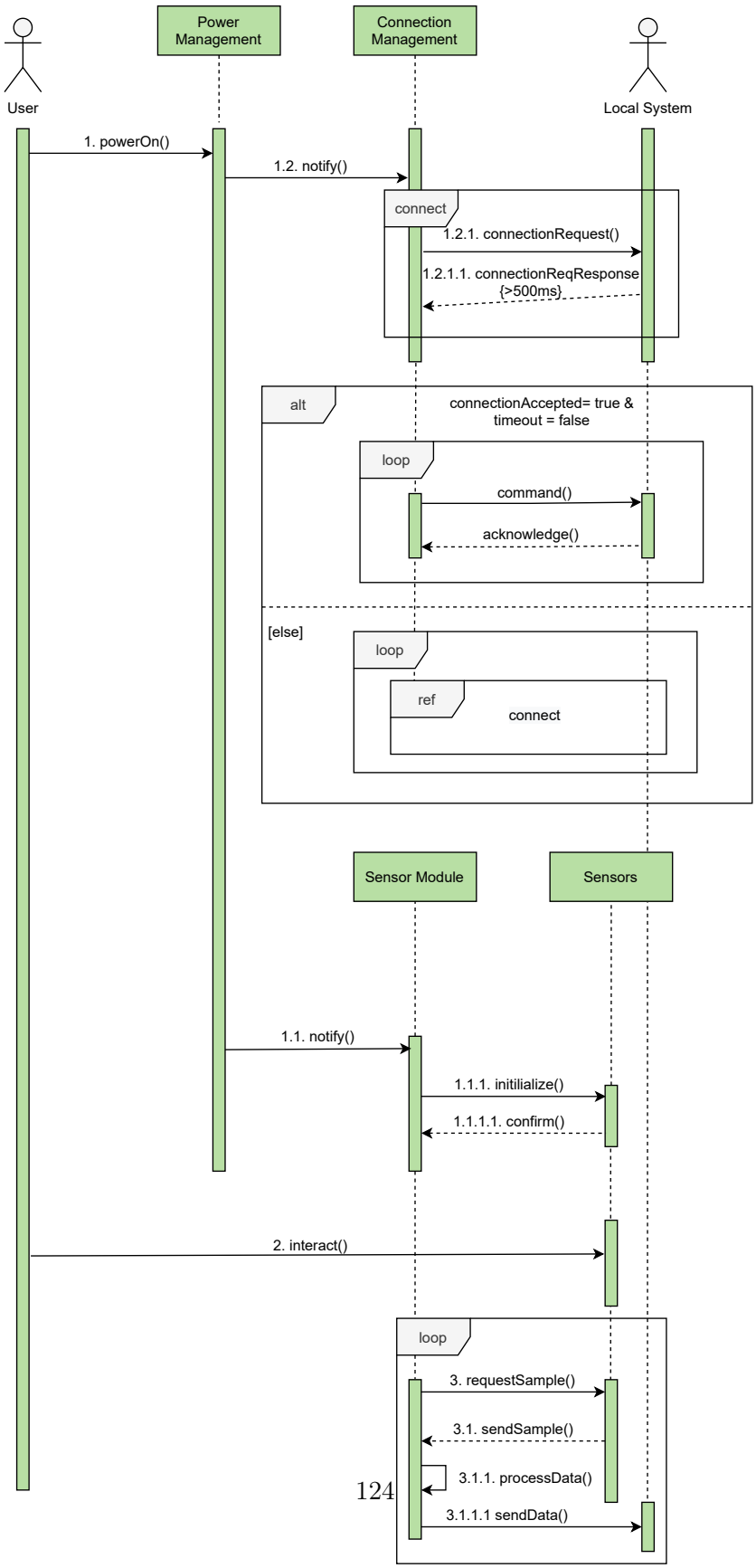


Figure A.4: Remote System Sequence Diagram Augmented

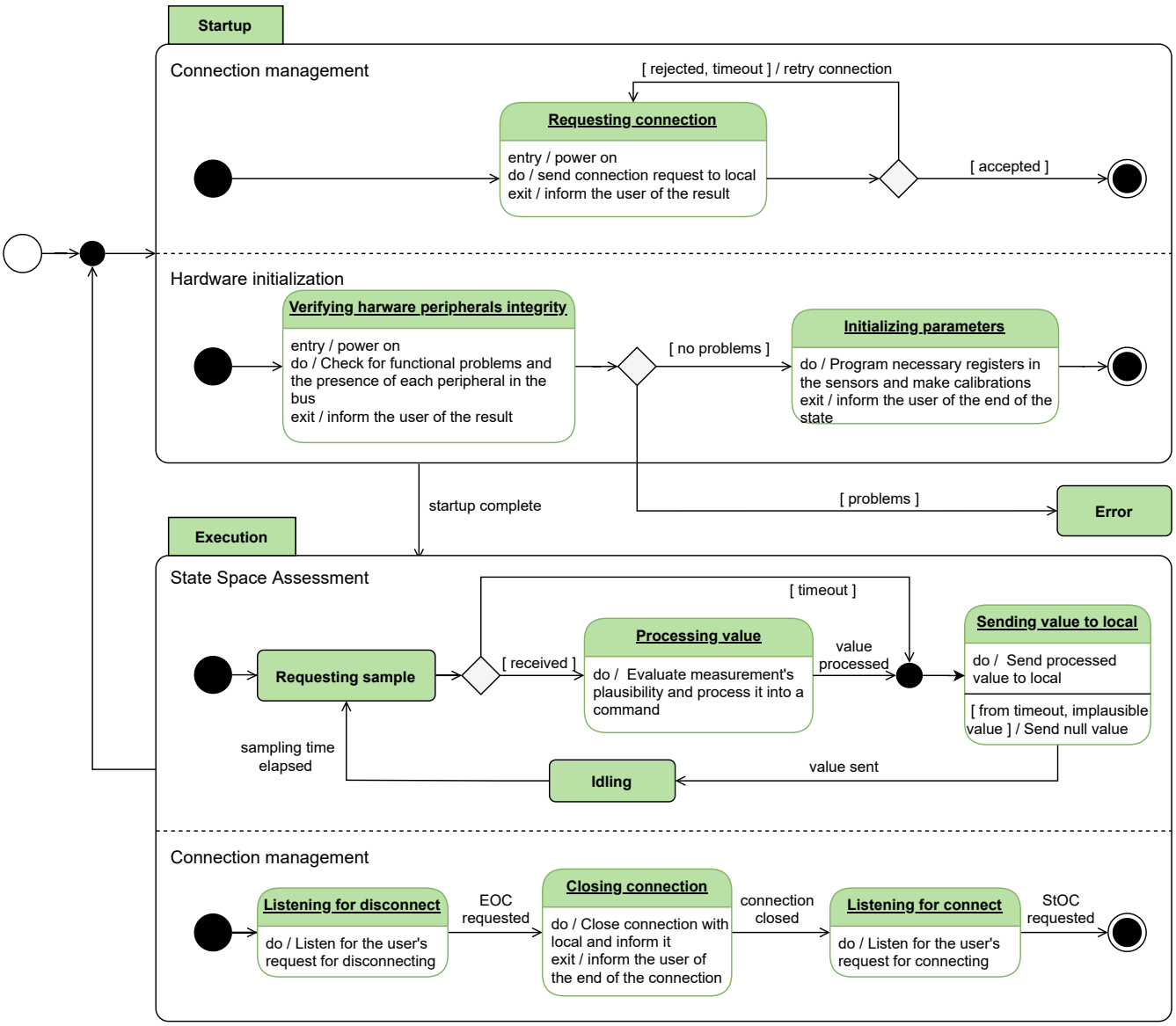


Figure A.5: State Machine Diagram - Remote Augmented

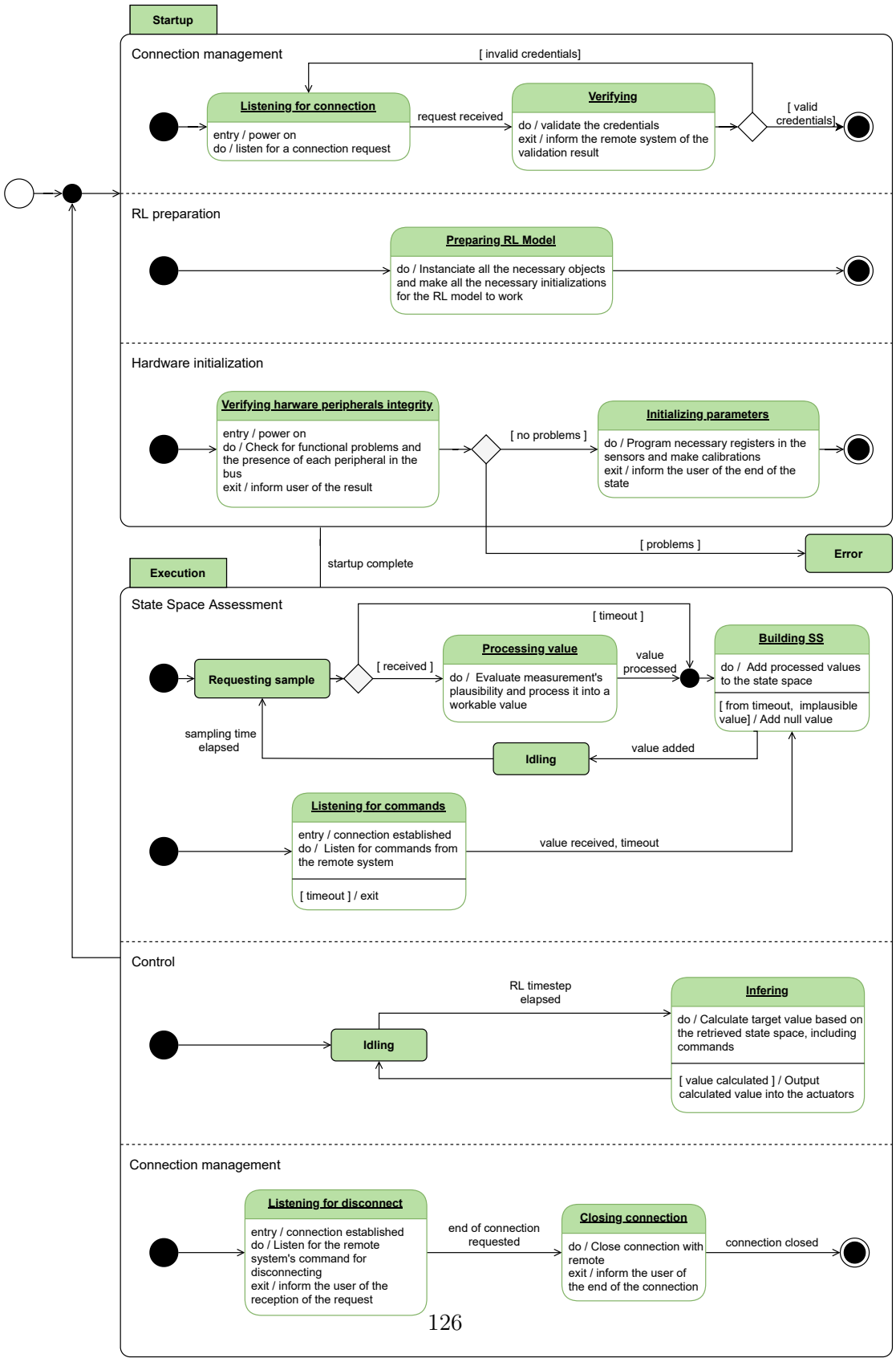


Figure A.6: State Machine Diagram - Local Augmented

Appendix B

Prototype Shematics

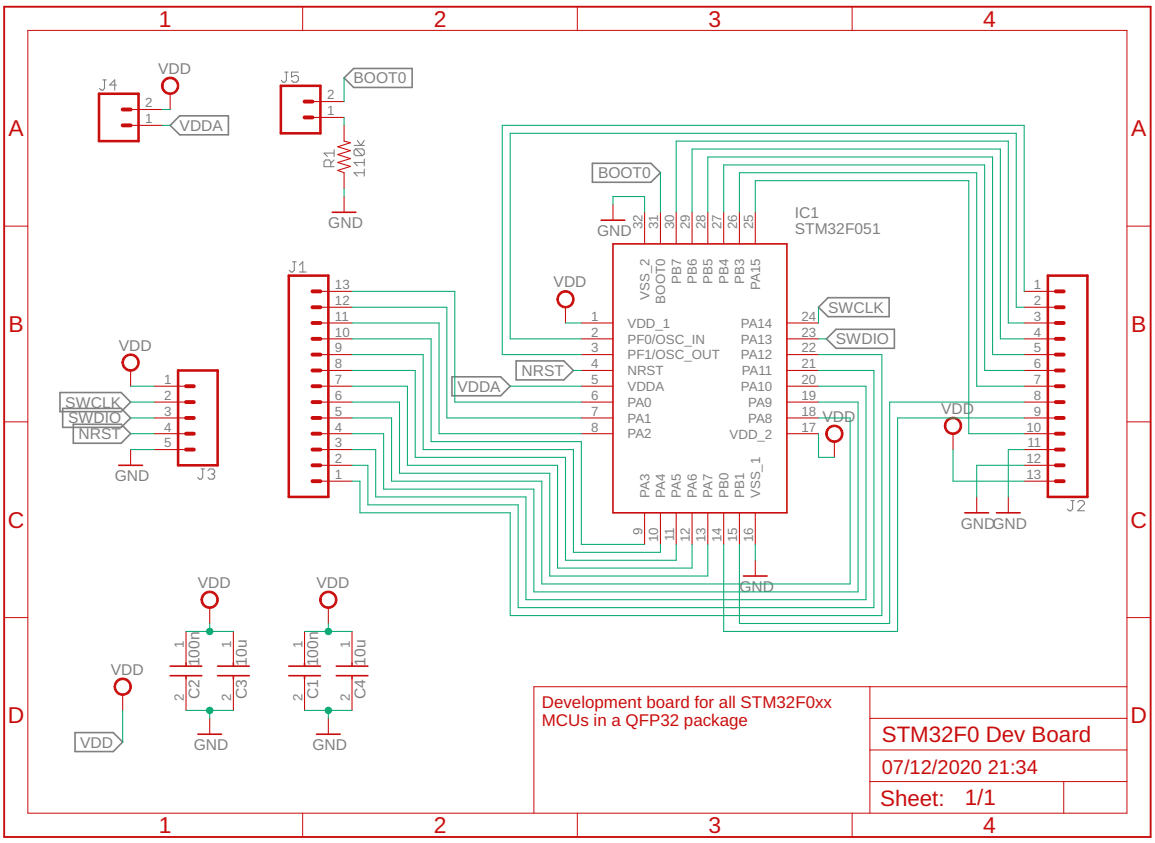


Figure B.1: STM32F0 Development Board - Schematic

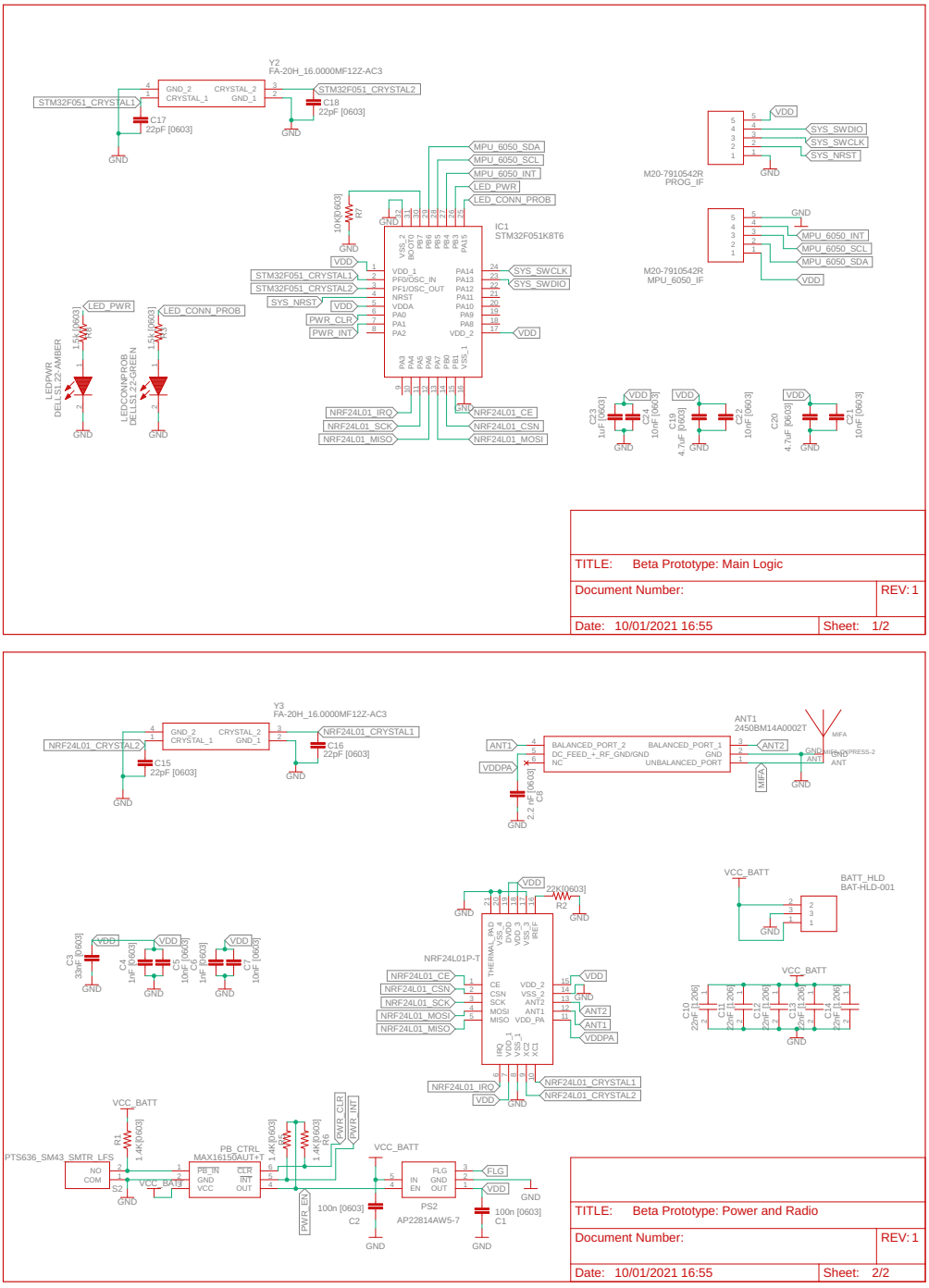


Figure B.2: Prototype Beta: Initial Design

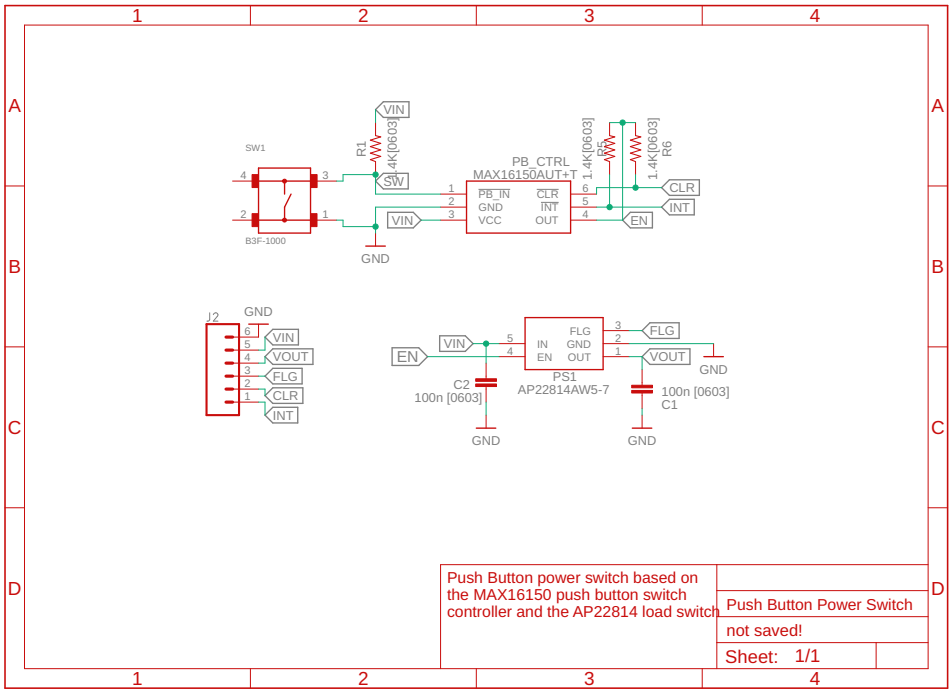


Figure B.3: Push Button Switch Board - Schematic

Appendix C

Task Timelines

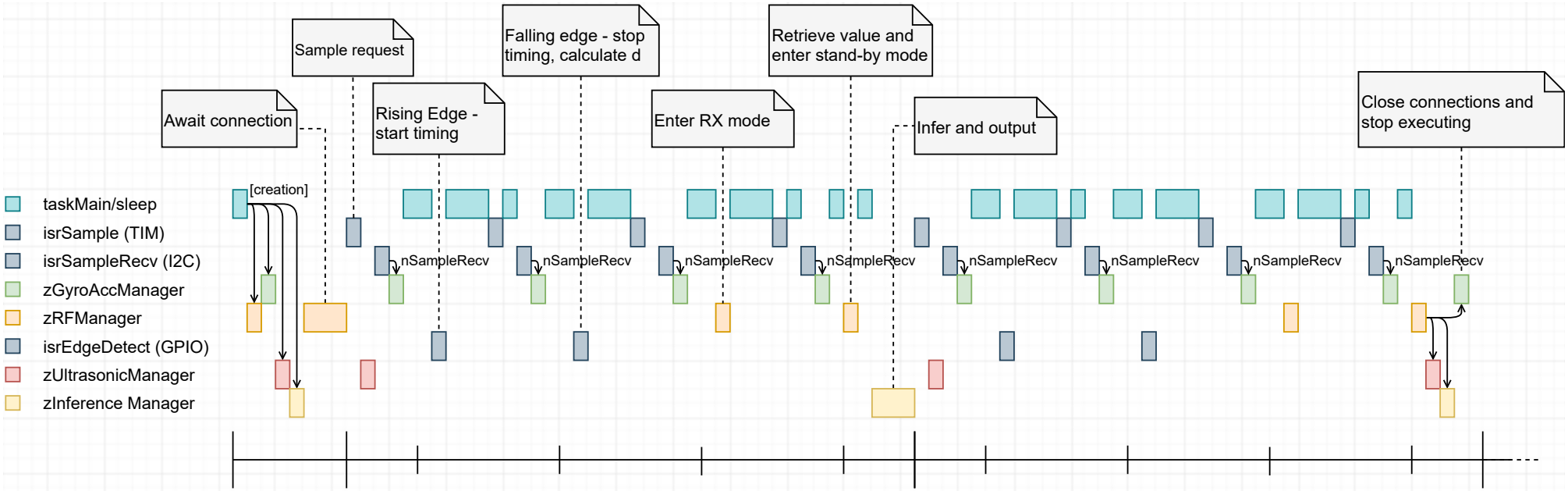


Figure C.1: Local Task Timeline

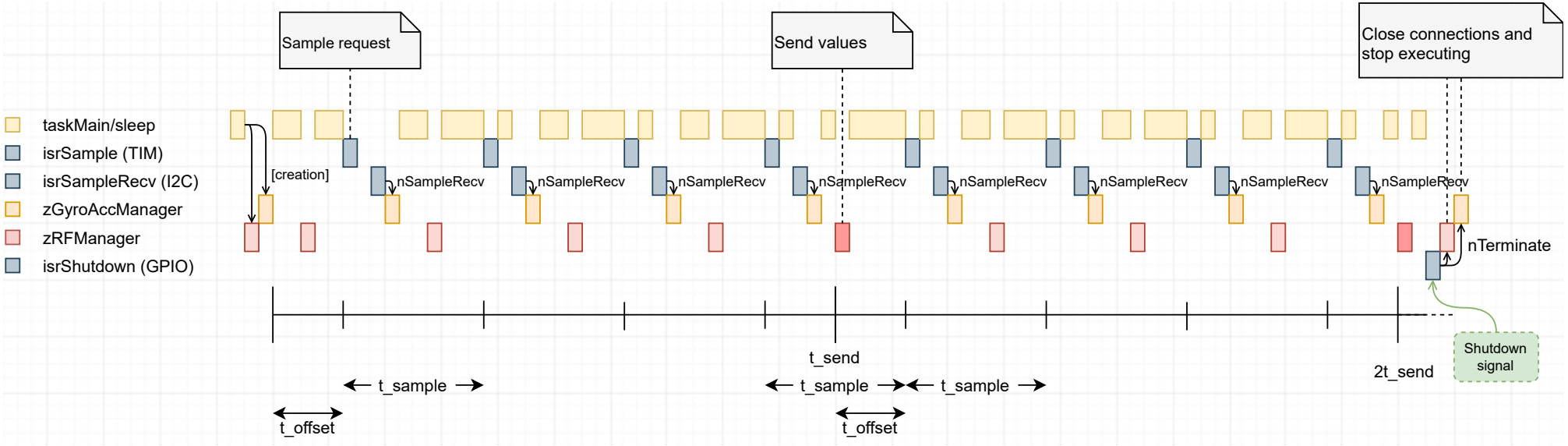


Figure C.2: Remote Task Timeline

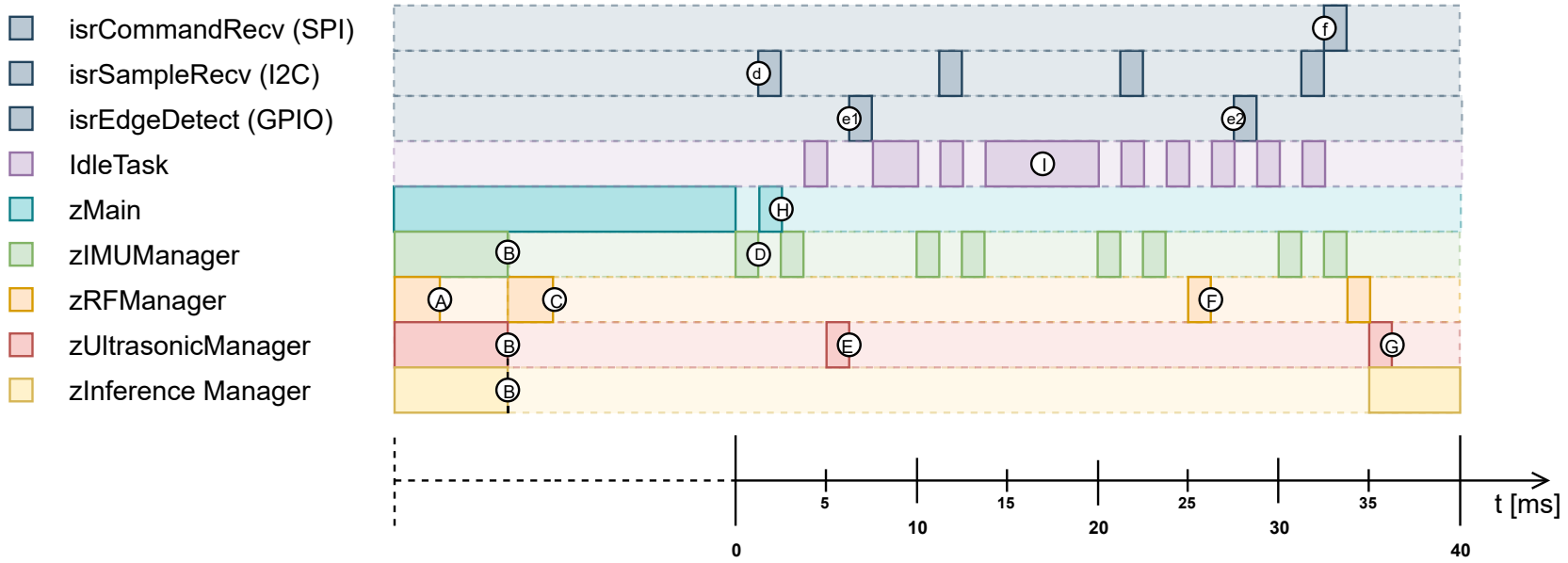


Figure C.3: Local Task Timeline - Refined

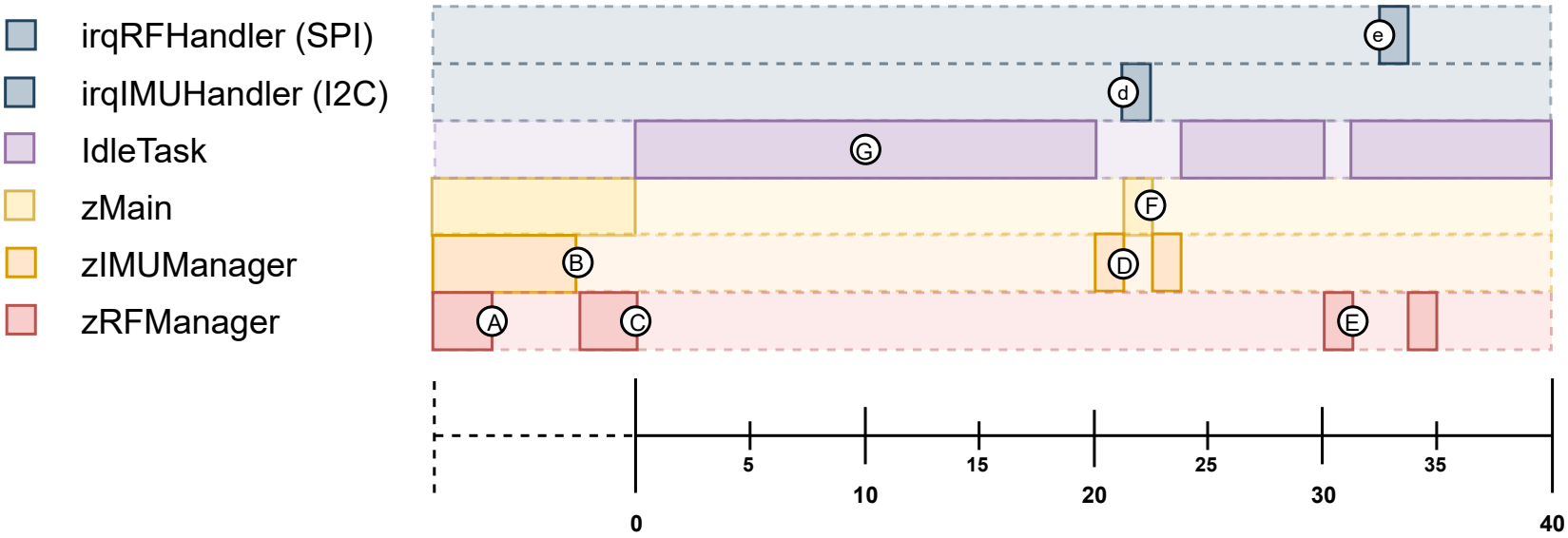


Figure C.4: Remote Task Timeline - Refined

Appendix D

Project Planning

	Task Name	Duration	Start	ETA
1	**M0: Planning**			
1.1	Project Choice	2 days	15.10.20	16.10.20
1.2	Problem Statement	3 days	17.10.20	19.10.20
1.3	Project Plan	7 days	20.10.20	26.10.20
2	**M1: Analysis**			
2.1	System Overview	2 days	20.10.20	21.10.20
2.2	Requirements and constraints	1 day	22.10.20	22.10.20
2.3	System Architecture	2 days	23.10.20	24.10.20
2.4	System Analysis	2 days	25.10.20	26.10.20
2.5	Market Research	2 days	27.10.20	28.04.12
2.6	Estimated Budget	2 days	29.10.20	30.10.20
2.7	Diagrams	10 days	31.10.20	9.11.20
3	**M2: Design**			
3.1	Analysis Review	3 days	10.11.20	12.11.20
3.2	Hardware Specification	10 day	13.11.20	22.11.20
3.3	Component Shipping	14 days	23.11.20	7.12.20
3.4	Software Specification	10 days	18.11.20	27.11.20
3.5	RL Model Design and Training	8 days	28.11.20	5.12.20
4	**M3: Implementation**			
4.1	Hardware Testing	7 days	8.12.20	11.12.20
4.2	Software Implementation	32 days	1.12.20	31.12.20
4.3	Hardware Implementation	32 days	1.12.20	31.12.20
4.4	RL Model Conversion	5 days	1.12.20	5.12.20
5	**M4: Testing & Verification**			
5.1	Software Unit Testing	4 days	1.1.21	4.1.21
5.2	Software Integrated Testing	5 days	4.1.21	8.1.21
5.3	Hardwareware Unit Testing	4 days	1.1.21	4.1.21
5.4	Hardware Integrated Testing	5 days	4.1.21	8.1.21
5.5	Verification	3 days	9.1.21	11.1.21

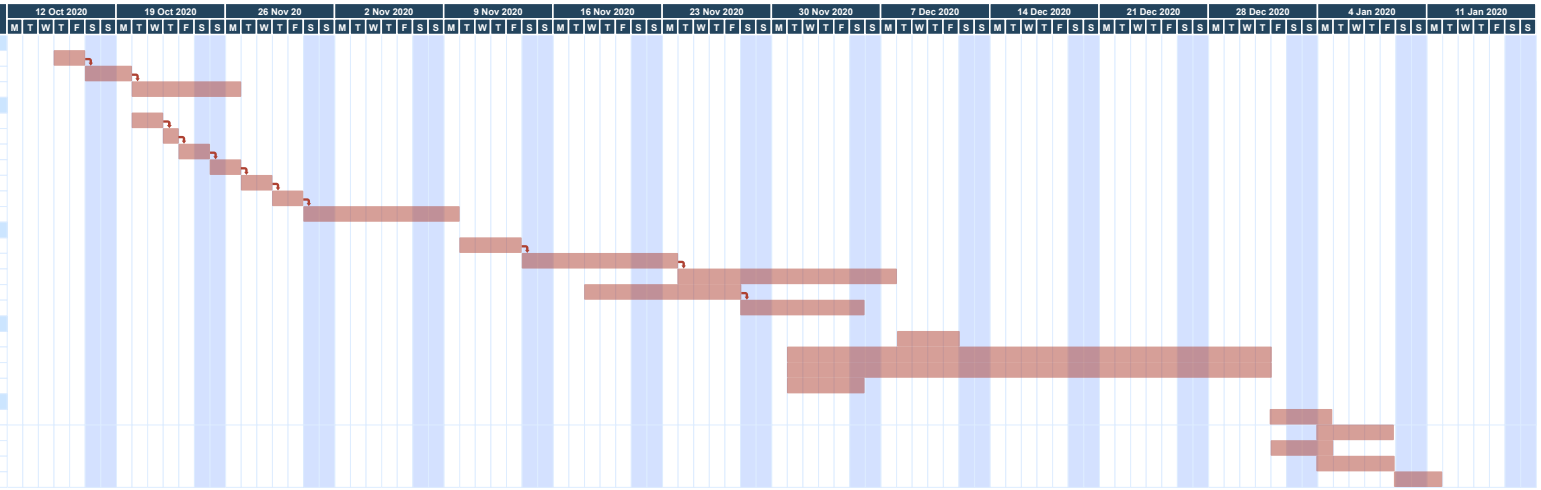


Figure D.1: Gantt Diagram

Appendix E

FreeRTOS Wrapper

```
#ifndef INC__RTOS_WRAPPER_H_
#define INC__RTOS_WRAPPER_H_

#include "cmsis_os.h"

/*!< ---------------------------------- TOOLS ------------------------------- */

/*! Concatenates x and y. It is meant to support other macro definitions */
#define CAT(x, y) x ## y

/*! Returned by some methods as a way to convey success in their operation */
#define RTOS_TRUE                       pdTRUE

/*! Returned by some methods as a way to convey failure in their operation */
#define RTOS_FALSE                  pdFALSE

/*! Returned by some methods as a way to convey a timeout in their operation */
#define RTOS_TIMEOUT    pdFALSE

/*! Converts a value from milliseconds to its equivalent in system ticks */
#define RTOS_TOOLS_MS_TO_TICKS(__ms__)  pdMS_TO_TICKS(__ms__)


/*!< ------------------------------- SCHEDULER ------------------------------ */

#define RTOS_SCHEDULER_START() \
        vTaskStartScheduler();

/*!< -------------------------------- DELAYS -------------------------------- */

#define RTOS_TIMESTAMP(__timestamp__)   \
        TickType_t __timestamp__

#define RTOS_KEEP_TIMESTAMP(__timestamp__) \
        __timestamp__ = xTaskGetTickCount()

#define RTOS_GET_TIMESTAMP() \
        xTaskGetTickCount()

/*!< Delay task execution __msdelay__ ms */
#define RTOS_DELAY(__msdelay__) \
        vTaskDelay(RTOS_TOOLS_MS_TO_TICKS(__msdelay__))

/*!< Delay task execution __msdelay__ ms past __timestamp__*/
#define RTOS_DELAY_UNTIL(__timestamp__, __msdelay__)    \
        vTaskDelayUntil(&__timestamp__, RTOS_TOOLS_MS_TO_TICKS(__msdelay__))


/*!< ----------------------------- NOTIFICATIONS ---------------------------- */

/*!< Create a notification with an __id__ in the range [0, 31] */
#define RTOS_NOTIFICATION(__notification__, __id__) \
uint32_t __notification__ = (1 << __id__);

/*!< Notifies a task */
#define RTOS_NOTIFY(__task__, __notification__) \
xTaskNotify(__task__, __notification__, eSetBits)

/*!< Notifies a task */
#define RTOS_NOTIFY_ISR(__task__, __notification__) \
xTaskNotifyFromISR(__task__, __notification__, eSetBits, NULL)

/*!< Awaits a notification */
#define RTOS_AWAIT(__notification__)            \
xTaskNotifyWait(0, __notification__, NULL, portMAX_DELAY)

/*!< Awaits a notification */
#define RTOS_AWAIT_TIMEOUT(__notification__, __timeout__)               \
xTaskNotifyWait(0, __notification__, NULL, RTOS_TOOLS_MS_TO_TICKS(__timeout__))


/*!< ------------------------------ SEMAPHORES ------------------------------ */
```

```
74  /*!< Get the representation of the mutex buffer associated with __semaphore__*/
    #define RTOS_SEMAPHORE_BUFFER(__semaphore__)    \
            CAT(__semaphore__, _SemaphoreBuffer)

    /*!< Get the representation of the handle associated with __semaphore__ */
79  #define RTOS_SEMAPHORE_HANDLE(__semaphore__)    \
            __semaphore__

    /*!< Get __semaphore__'s maximum count */
    #define RTOS_SEMAPHORE_MAX_COUNT(__semaphore__) \
84          CAT(__semaphore__, _MaxCount)

    /*!< Declare the necessary objects for creating and manipulating a mutex */
    #define RTOS_SEMAPHORE_STATIC(__semaphore__, __MAX_COUNT__) \
    const uint32_t RTOS_SEMAPHORE_MAX_COUNT(__semaphore__) = __MAX_COUNT__; \
89  SemaphoreHandle_t RTOS_SEMAPHORE_HANDLE(__semaphore__); \
    StaticSemaphore_t RTOS_SEMAPHORE_BUFFER(__semaphore__);

    /*!< Wrapper macro for 'xSemaphoreCreateMutexStatic' */
    #define RTOS_SEMAPHORE_CREATE_STATIC(__semaphore__, __INIT_COUNT__)     \
94          RTOS_SEMAPHORE_HANDLE(__semaphore__) = xSemaphoreCreateCountingStatic(\
                RTOS_SEMAPHORE_MAX_COUNT(__semaphore__), __INIT_COUNT__, \
                &RTOS_SEMAPHORE_BUFFER(__semaphore__))

    /*!< Wrapper macro for 'xSemaphoreTake' for specific use for semaphores
99   *      created through this API */
    #define RTOS_SEMAPHORE_DEC(__semaphore__) \
            xSemaphoreTake(__semaphore__, portMAX_DELAY)

    /*!< Wrapper macro for 'xSemaphoreGive' for specific use for semaphores
104  *      created through this API */
    #define RTOS_SEMAPHORE_INC(__semaphore__) \
            xSemaphoreGive(__semaphore__)

            /*!< Wrapper macro for 'xSemaphoreTakeFromISR' for specific use for
109  * semaphores created through this API */
    #define RTOS_SEMAPHORE_INC_ISR(__semaphore__) \
            xSemaphoreGiveFromISR(__semaphore__, NULL)

    /*!< Wrapper macro for 'xSemaphoreGiveFromISR' for specific use for
114  * semaphores
     *      created through this API */
    #define RTOS_SEMAPHORE_DEC_ISR(__semaphore__) \
            xSemaphoreTakeFromISR(__semaphore__, NULL)

119 /*!< Get count of __semaphore__ */
    #define RTOS_SEMAPHORE_GET_COUNT(__semaphore__) \
            uxSemaphoreGetCount(RTOS_SEMAPHORE_HANDLE(__semaphore__))


124 /*!< ------------------------------- QUEUES ------------------------------- */

    #define RTOS_QUEUE_BUFFER(__queue__) \
            CAT(__queue__, _QueueBuffer)

129 #define RTOS_QUEUE_SIZE(__queue__) \
            CAT(__queue__, _QueueSize)

    #define RTOS_QUEUE_HANDLE(__queue__)    \
            __queue__
134
    #define RTOS_QUEUE_STORAGE(__queue__) \
            CAT(__queue__, _QueueStorage)

    #define RTOS_QUEUE_TYPE_SIZE(__queue__) \
139         CAT(__queue__, _QueueTypeSize)

    #define RTOS_QUEUE_STATIC(__queue__, __type__, __size__)        \
            QueueHandle_t RTOS_QUEUE_HANDLE(__queue__); \
            StaticQueue_t RTOS_QUEUE_BUFFER(__queue__); \
144         const uint32_t RTOS_QUEUE_SIZE(__queue__) = __size__; \
            const uint32_t RTOS_QUEUE_TYPE_SIZE(__queue__) = sizeof(__type__); \
```

```
        __type__ RTOS_QUEUE_STORAGE(__queue__)[__size__];

#define RTOS_QUEUE_CREATE_STATIC(__queue__) \
        RTOS_QUEUE_HANDLE(__queue__) = \usepackage{}xQueueCreateStatic \
            (RTOS_QUEUE_SIZE(__queue__),\
            RTOS_QUEUE_TYPE_SIZE(__queue__), \
                (uint8_t*)RTOS_QUEUE_STORAGE(__queue__), \
                            &RTOS_QUEUE_BUFFER(__queue__));

#define RTOS_QUEUE_RECV(__queue__, __dest_ptr__) \
        xQueueReceive(__queue__, __dest_ptr__, portMAX_DELAY)

#define RTOS_QUEUE_RECV_TIMEOUT(__queue__, __dest_ptr__, __timeout__) \
        xQueueReceive(__queue__, __dest_ptr__, RTOS_TOOLS_MS_TO_TICKS \
        (__timeout__))

#define RTOS_QUEUE_SEND_BACK(__queue__, __source__) \
        xQueueSendToBack(__queue__, __source__, portMAX_DELAY)

#define RTOS_QUEUE_SEND_BACK_TIMEOUT(__queue__, __source__, __timeout__) \
        xQueueSendToBack(__queue__, __source__, RTOS_TOOLS_MS_TO_TICKS \
            (__timeout__))

#define RTOS_QUEUE_SEND_BACK_ISR(__queue__) \
        xQueueSendToBackFromISR(__queue__, __source__, NULL)

#define RTOS_QUEUE_IS_FULL(__queue__) \
        (uxQueueMessagesWaiting(__queue__) == RTOS_QUEUE_SIZE(__queue__))

#define RTOS_QUEUE_IS_FULL_ISR(__queue__) \
        (uxQueueMessagesWaitingFromISR(__queue__)== RTOS_QUEUE_SIZE(__queue__))

#define RTOS_QUEUE_IS_EMPTY(__queue__) \
        (uxQueueMessagesWaiting(__queue__) == 0)

#define RTOS_QUEUE_IS_EMPTY_ISR(__queue__) \
        (uxQueueMessagesWaitingFromISR(__queue__) == 0)

/*!< ------------------------------- MUTEXES ------------------------------- */

/*! Get the representation of the mutex buffer associated with __mutex__ */
#define RTOS_MUTEX_BUFFER(__mutex__)    CAT(__mutex__, _MutexBuffer)

/*! Get the representation of the handle associated with __mutex__ */
#define RTOS_MUTEX_HANDLE(__mutex__)    __mutex__

/*! Declare the necessary objects for creating and manipulating a mutex. */
#define RTOS_MUTEX_STATIC(__mutex__) \
SemaphoreHandle_t RTOS_MUTEX_HANDLE(__mutex__); \
StaticSemaphore_t RTOS_MUTEX_BUFFER(__mutex__);

/*! Wrapper macro for 'xSemaphoreCreateMutexStatic' */
#define RTOS_MUTEX_CREATE_STATIC(__mutex__)     \
        RTOS_MUTEX_HANDLE(__mutex__) = xSemaphoreCreateMutexStatic( \
                        &RTOS_MUTEX_BUFFER(__mutex__))

/*! Wrapper macro for 'xSemaphoreTake' for specific use for mutexes
 *      created through this API */
#define RTOS_MUTEX_LOCK(__mutex__) \
        xSemaphoreTake(__mutex__, portMAX_DELAY)

/*! Wrapper macro for 'xSemaphoreTake' for specific use for mutexes */
#define RTOS_MUTEX_UNLOCK(__mutex__) \
        xSemaphoreGive(__mutex__)


/*!< ------------------------------- TASKS ------------------------------- */

/*! Get the representation of the stack array associated with __task__ */
#define RTOS_TASK_STACK(__task__)               CAT(__task__, _Stack)

/*! Get the representation of the handle associated with __task__ */
#define RTOS_TASK_HANDLE(__task__)      \
```

```
        __task__

/*! Get the representation of the function associated with __task__ */
#define RTOS_TASK_FUNCTION(__task__)    \
        CAT(__task__,_Fun)

/*! Get the representation of the stack buffer associated with __task__ */
#define RTOS_TASK_BUFFER(__task__) \
        CAT(__task__, _TaskBuffer)

/*! Get the representation of the stack size associated with __task__ */
#define RTOS_TASK_STACK_SIZE(__task__) \
        CAT(__task__, _StackSize)

/*! Get the representation of __task__'s priority */
#define RTOS_TASK_PRIORITY(__task__) \
        CAT(__task__, _Priority)

/*! Get the representation of __task__'s informal name */
#define RTOS_TASK_NAME(__task__) \
        CAT(__task__, _Name)

/*! Declare the necessary objects for creating and manipulating a task. */
#define RTOS_TASK_STATIC(__task__, __STACK_SIZE__, __PRIORITY__, __NAME__) \
        const uint32_t RTOS_TASK_STACK_SIZE(__task__) = __STACK_SIZE__/4;\
        const uint32_t RTOS_TASK_PRIORITY(__task__) = __PRIORITY__;\
        const char RTOS_TASK_NAME(__task__)[] = __NAME__;\
        StaticTask_t RTOS_TASK_BUFFER(__task__);\
        StackType_t RTOS_TASK_STACK(__task__)[__STACK_SIZE__/4];\
        TaskHandle_t RTOS_TASK_HANDLE(__task__);

/*! Wrapper macro for 'xTaskCreateStatic' */
#define RTOS_TASK_CREATE_STATIC(__task__)       \
        RTOS_TASK_HANDLE(__task__) = xTaskCreateStatic(\
                        RTOS_TASK_FUNCTION(__task__), \
                        RTOS_TASK_NAME(__task__), \
                        RTOS_TASK_STACK_SIZE(__task__),\
                        NULL, RTOS_TASK_PRIORITY(__task__), \
                        RTOS_TASK_STACK(__task__), \
                        &RTOS_TASK_BUFFER(__task__))

/*! Default task parameter */
#define RTOS_TASK_STATIC_DEFAULT_PARAM  void* empty_param

/*! Declare task function */
#define RTOS_TASK_FUN(__task__) \
        void CAT(__task__,_Fun)(RTOS_TASK_STATIC_DEFAULT_PARAM)

#define RTOS_TASK_DELETE() \
        vTaskDelete(NULL)

#define RTOS_IDLE_CALLBACK()    \
        void vApplicationIdleHook(void)

/*! RTOS priorities enumeration */
typedef enum RTOSPriorities {
        RP_IDLE = 0,                    /*!< IDLE priority */
        RP_LOW,                                 /*!< LOW priority */
        RP_BELOW_NORMAL,        /*!< BELOW NORMAL priority */
        RP_NORMAL,                              /*!< NORMAL priority */
        RP_ABOVE_NORMAL,        /*!< ABOVE NORMAL priority */
        RP_HIGH,                                /*!< HIGH priority */
        RP_REAL_TIME                    /*!< REAL TIME priority */
} RTOSPriorities_t;


#endif /* INC__RTOS_WRAPPER_H_ */
```

Listing E.1: Own-made