



Norwegian University of
Science and Technology

Development of a Real-Time Embedded Control System for SLAM Robots

Johan Korsnes

Master of Science in Cybernetics and Robotics

Submission date: June 2018

Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Formulation

A project with the goal of developing robots for Simultaneous Localization and Mapping (SLAM) was started in 2004 at NTNU. Throughout the years since its inception, several different robots have been designed. The majority of these robots employ 8-bit AVR microcontrollers for their control systems. Common to all of these robots is the Nordic Semiconductor nRF Bluetooth dongle used for communication with a central server. This dongle contains an nRF51-series System-on-Chip (SoC).

In this project, a real-time embedded control system for the SLAM robots is to be developed from the ground up. It shall be based on Nordic Semiconductor's more recent nRF52-series SoC for both robot control and communication. The new nRF52-series SoC features sophisticated debugging functionality combined with a powerful 32-bit ARM Cortex M4F processor with a dedicated floating point unit.

A new printed circuit board shall be designed, integrating the new SoC and additional components as seen fit. A development environment with proper debugging facilities shall be set up, as well as a template application running a real-time operating system. To facilitate future SLAM application development, priority should be given to the following aspects: Proper debug facilities, such as trace and single-stepping; a unified and robust design, based on a single printed circuit board; autonomous capabilities, in the form of assessment of computing abilities.

The specific tasks outlined for the project are as follows:

- Review and derive specifications and requirements
- Develop schematics and printed circuit board layout
- Perform complete board bring-up, including assembly and soldering
- Set up and configure the development environment with debug facilities
- Develop drivers for relevant components
- Set up a template application that will serve as a starting point for future SLAM application development

Preface

This thesis is not the continuation of a specialization project. Neither does it directly continue on any previous work. I was assigned one of the robots part of the SLAM project, and this robot has been used to form parts of the specifications for the new control system.

I am thankful for the feedback I have received from my supervisor, Professor Tor Onshus.

I would like to thank the personnel at the ITK workshop and Omega workshop, for letting me use their equipment.

Having been able to design and bring up a complete embedded system all the way from schematics to firmware has been a very rewarding process. With embedded systems as a study specialization and hobby, I cannot imagine anything more exciting. It has given me invaluable insight into the world of embedded systems.

Johan Korsnes
Trondheim 14.06.2018

Summary

This thesis encompasses all phases in the development of a real-time embedded control system designed for simultaneous localization and mapping (SLAM) robots. This work includes system design, schematic capture, printed circuit board (PCB) design, assembly using precision soldering, and embedded software development. A template application running on a real-time operating system (RTOS) has been set up with device drivers. Real-time aware debug facilities have been implemented, giving complete insight into the RTOS' behavior.

A set of specifications and requirements have been derived for the new control system, ensuring compatibility with the equipment used on current SLAM robots. The nRF52 System-on-Chip (SoC) used has been thoroughly assessed. New functionality has been introduced, such as an onboard display and a microSD slot for portable storage. A two-stage power supply has been designed with a focus on efficiency and a stable output voltage.

Complete schematic capture of the electrical control system has been performed, for which a four-layer PCB layout has been designed. While the fabrication of the PCB itself is outsourced to an external fabricator, the assembly process is performed as part of the thesis work. This process involves different precision soldering techniques and electrical verification.

A low-cost development environment is set up, where the development kit used for flash programming and debugging is the only expense. The integrated development environment used, Segger Embedded Studio, and the advanced real-time aware tracing software used, Tracealyzer, are both available in educational licenses free of charge. Both a set of drivers and a template application incorporating these, have been developed. The template application is set up to showcase different synchronization mechanisms, and to serve as a template for future work.

The new control system features a robust and unified single PCB design, something which addresses concerns raised about the hardware reliability with previous control systems. The new development environment features modern real-time aware debugging facilities, improving upon the inadequacies in previous control systems. With a template application with drivers set up, the control system is viable for implementation in a SLAM robot.

Oppsummering

Denne avhandlingen tar for seg alle faser i designet og utviklingen av et sanntids innvevd kontrollsystem til bruk på kartleggings-roboter (SLAM). Arbeidet omfatter systemdesign, utvikling av skjematikk, design av kretskort, montasje, presisjonsloddning og programvareutvikling for tilpassede datasystem. En applikasjon kjørende på et sanntids-operativsystem er utviklet med tilhørende enhetsdrivere. Feilsøknings-verktøy designet for sanntidsapplikasjoner har blitt implementert, noe som gir omfattende innsikt i systemets oppførsel.

Et sett med spesifikasjoner og krav er utarbeidet for det nye kontrollsystemet, med fokus på kompatibilitet med det utstyret som idag benyttes på SLAM-robotene. Det har blitt utført en grundig evaluering av nRF System-på-Chip (SoC)-en som skal benyttes. Ny funksjonalitet har blitt introdusert, slik som et integrert display og mikroSD støtte for portable lagring av større mengder data. Et to-steps strømforsyning er designet med fokus på virkningsgrad og stabil utgangsspenning.

Elektrisk skjematikk tilhørende det elektriske kontrollsystemet har blitt utviklet, som så har dannet grunnlaget for designet av et fire-lags kretskort. Selve produksjonen av kretskortet er utført av tredjepart, men all sammensetting og loddning av komponenter er utført som en del av prosjektet. Denne prosessen involverer forskjellige teknikker for presisjons-loddning samt verifikasjon av det elektriske systemet.

Et lavkostnads utviklingsmiljø er satt opp, hvor et nRF utviklerbrett som benyttes til programmering og debugging utgjøre eneste kostnad. Det integrerte utviklingsmiljøet som benyttes, Segger Embedded Studio, og det sanntids-tilpassede tracing-verktøyet Tracealyzer som også benyttes, er begge tilgjengelig i en gratis akademisk lisens. Et sett med drivere er utviklet, samt en applikasjon basert på FreeRTOS som benytter disse. Applikasjonen er satt opp for å vise frem noen av de forskjellige synkroniseringsmekanismene tilgjengelig, og for å danne et grunnlag for fremtidig arbeid.

Det nye kontrollsystemet består av et robust og enhetlig design på ett enkelt kretskort. Dette adresserer bekymringer vedrørende robusthet til hardware som har vært fremmet rundt tidligere kontrollsystem. Det nye utviklingsmiljøet stiller med moderne sanntids-tilpassede verktøy for debugging. Med en eksempelapplikasjon med driver satt opp, har kontrollsystemet vist seg å være et lovende system for implementering i SLAM-robotene.

Conclusion

The control system developed in this thesis addresses primarily two growing concerns regarding the state of the current SLAM robots: robustness of hardware, and the lack of more sophisticated real-time debugging facilities. In addition, the new platform significantly increases the computational power available to the application developer, greatly facilitating the migration towards more autonomous robots.

It is now a unified design with all connections and most components integrated into a single printed circuit board. Distrust in the hardware should be a non-issue in the new system, as application developers will no longer have to worry about loose wires or hardware faults when debugging. This unification does come at the cost of complicating future prototyping, as the process of exchanging components will require drop-in compatible alternatives.

The migration to the nRF52 SoC comes with an increase in RAM and program storage capacities. Running with a clock frequency about four times that of the ATmega, combined with 32-bit architecture and dedicated floating point unit, this all ensures that the new control system makes for a solid foundation for the more computing intensive autonomous robots under development.

Instead of relying on rudimentary debugging facilities such as light emitting diodes and printing to console, the debug features of the new SoC are leveraged to offer modern and more powerful debugging facilities. These facilities include real-time monitor debugging and RTOS aware trace functionality. Combined, this gives a much more complete insight into the behavior of the RTOS and should accelerate future development.

Combined, the aforementioned aspects of the new control system certainly goes a long way to alleviate deficiencies curbing efficient SLAM development in the contemporary control systems. While no SLAM application is developed as part of this thesis work, the new development environment set up with RTOS aware analysis—made possible with the migration to the new SoC—should be able to accelerate future development significantly. The new debugging features combined with unified hardware design and a new and more powerful SoC should make for a more stable and suitable platform.

Contents

Problem Formulation	i
Preface	iii
Summary	v
Conclusion	ix
List of Tables, Figures an Acronyms	xv
1 Introduction	1
1.1 Background	1
1.1.1 Simultaneous Localization and Mapping	1
1.1.2 Template Robot	2
1.2 Previous Work	3
1.3 Motivation	5
1.4 Objective and Scope	5
1.5 Outline	6
2 Specifications and Theory	7
2.1 System-on-Chip	7
2.1.1 Peripherals	10
2.1.2 Storage	11
2.1.3 Compute Performance	13
2.1.4 Debugging	15

2.1.5	Summary	16
2.2	System Configuration	16
2.3	New Functionality	20
2.3.1	Display	20
2.3.2	Portable Storage	20
2.3.3	Connectors and Test Points	21
2.4	Power Supply	23
2.4.1	Component Selection	24
2.4.2	Performance	26
2.5	PCB Fundamentals	27
2.6	Development Environment	29
2.6.1	Build Process	29
2.6.2	IDE and Flash Programmer	30
2.6.3	Debugging Facilities	31
3	Schematics and Layout	33
3.1	Altium Designer	33
3.2	Component Library	35
3.2.1	Low Drop-out Regulator Heatsinking	36
3.2.2	System-on-Chip Vias	37
3.3	Schematic Capture	38
3.3.1	Power Supply	38
3.3.2	System-on-Chip	40
3.3.3	Analog-to-Digital Conversion	42
3.3.4	Bus Devices	43
3.4	Printed Circuit Board	44
3.4.1	Dimensions and Layers	44
3.4.2	Initial Configuration	46
3.4.3	Design Process	48
3.5	Second Revision	53
3.5.1	Corrections and Improvements	53
3.5.2	Ease of Manufacturing	54
3.5.3	Inrush Current	54
4	Hardware	55
4.1	Equipment	55
4.2	PCB Fabrication	55

4.3	System-on-Chip	59
4.4	Remaining Components	62
4.5	Power Supply Verification	64
5	Software	67
5.1	Development Environment	67
5.1.1	Host to Target Interface	67
5.1.2	Setting up a Minimal Example	68
5.1.3	Debugging	69
5.2	Driver Development	72
5.2.1	Display	72
5.2.2	microSD	75
5.2.3	Motors	76
5.2.4	Encoders	77
5.2.5	IR	78
5.2.6	Sensor Tower Servo	78
5.2.7	Magnetometer	79
5.3	Template Application	80
5.3.1	Gatekeepers	80
5.3.2	System Architecture	81
5.3.3	Autonomous Scan Cycle	82
6	Results	85
6.1	Hardware	85
6.2	Software	87
6.2.1	Development Environment	87
6.2.2	Drivers	87
6.2.3	Template Application	89
7	Discussion and Further Work	93
7.1	Discussion	93
7.2	Further Work	96
7.2.1	Completing the Control System	96
7.2.2	Second Revision	97
7.2.3	SLAM Application	97
	Bibliography	99

A Media Attachment **A-1**

A.1 Schematics and PCB Layout A-1

A.2 Software A-2

A.3 Tutorials A-2

B Schematics and PCB Layout **B-1**

Listings

5.1	FreeRTOSConfig.h: Increase heap size for more tasks.	68
5.2	sdk_config.h: SDK low-frequency clock source configuration.	69
5.3	sdk_config.h,main.c: Enabling and using RTT.	70
5.4	trcConfig.h Trace library target configuration.	71
5.5	FreeRTOSConfig.h: Enabling trace in FreeRTOS.	71
5.6	main.c: RTT interlock and Tracealyzer usage.	72
5.7	oled_driver.c: Part of the SPI configuration used.	74
5.8	oled_driver.c: SPI initialization routine.	74
5.9	oled_driver.c: Setting or resetting a single pixel.	75
5.10	task_display.c: Display Task structure.	81

List of Tables

1.1	Control system revisions timeline.	4
2.1	Platform peripheral comparison.	11
2.2	Memory capacities for the different platforms.	12
2.3	SLAM robot peripheral and pin requirements.	18
2.4	Control system current consumption estimates.	25
2.5	DC-DC module used for the first stage.	25
2.6	LDO used for the second stage.	26
3.1	Layer stack configuration in AD.	47
4.1	Equipment used during soldering, assembly and verification.	56
5.1	OLED driver callback functions	73
5.2	SDK graphics library drawing operations.	75
5.3	microSD-card driver interface.	76
5.4	Motor driver interface.	77
5.5	Encoder initialization functions.	77
5.6	IR driver interface functions.	78
5.7	Servo driver interface functions.	79
5.8	Magnetometer driver interface functions.	79
A.1	Description of the content in the pcb folder.	A-1
A.2	Description of the content in the sw folder.	A-2

List of Figures

1.1	Template SLAM robot.	2
2.1	Simplified SoC overview, showing only the relevant subsystems.	9
2.2	Platform memory capacities compared.	12
2.3	Flash and memory layout on the SoC with SoftDevice.	13
2.4	Components and peripheral usage of current control system.	17
2.5	Components and peripheral usage of new control system.	19
2.6	ATmega control system, emphasis on wiring.	22
2.7	Comparison between previous and new power connectors.	23
2.8	JST connectors for LIDAR and IR sensors.	23
2.9	Power distribution scheme.	24
2.10	LDO Power Supply Rejection Ratio as a function of frequency.	27
2.11	Primary components of a PCB in four-layer configuration.	28
2.12	Steps involved in generating an executable for target.	29
2.13	Final process of transferring the program to the target.	30
3.1	Flowchart illustrating the complete PCB development cycle.	34
3.2	Parts constituting an AD component.	34
3.3	Thermal relief on left side vs no thermal relief on right side.	37
3.4	Different solutions for solving the via wicking issue.	37
3.5	DC-DC regulator with surrounding circuitry.	39
3.6	ESR requirements for LDO regulator output capacitor.	40
3.7	LDO regulator with surrounding circuitry.	41
3.8	Voltage divider structure	42
3.9	Bi-directional level shifter circuit.	43

3.10	PCB layer stackup legend.	45
3.11	Planned board positioning of sections and components.	46
3.12	Board set up and ready for layout design.	48
3.13	PCB layout with primary components placed.	49
3.14	PCB layout with primary and passive components.	50
3.15	PCB layout with components and tracks.	51
3.16	Internal power layer split into two sections.	52
3.17	Final PCB layout with ground plane.	52
4.1	Flowchart illustrating the complete PCB development cycle.	57
4.2	Visual inspection of the received PCB.	59
4.3	Solder paste and flux application, and the final SoC alignment.	60
4.4	Difference between good and bad QFN soldering joints.	61
4.5	Stencil that aids in dispensing solder paste correctly.	62
4.6	All components of a JST connector	63
4.7	Probing method for voltage regulators.	64
4.8	Oscilloscope measurements of voltage regulator output.	66
5.1	The connection between PC and control system.	68
5.2	Monitor mode debugging principle.	70
5.3	Components of the display driver.	73
5.4	Mapping between display buffer and pixels on the display.	74
5.5	Gatekeeper task structure.	80
5.6	Architecture of template application	82
5.7	Flow chart describing an IR scan cycle.	83
6.1	Assembled control system.	86
6.2	Status of driver and assembly of control system components.	88
6.3	Tracealyzer trace and profiler view.	90
6.4	Tracealyzer object history and communication flow view	91

Glossary and list of abbreviations

AD Altium Designer, electronics design automation software.

API Application Programming Interface

Application Developer Person using drivers and underlying facilities to develop the actual SLAM application.

AM Arduino Mega which some of the robots are based on. The Arduino Mega is based on the ATmega2560 microcontroller.

BLE Bluetooth Low Energy

BOM Bill of Materials

DC-DC Buck down voltage regulator.

DMA Direct Memory Access, enables peripherals to write/read memory without processor intervention.

DSP Digital Signal Processing.

FPU Floating Point Unit, hardware processing unit dedicated to floating point operations.

GPIO General Purpose Input/Output.

GPIOE General Purpose Input/Output with Task and Events.

GUI Graphical User Interface

I2C Inter-Integrated Circuit, serial multi-drop bus for communication.

IC Integrated Circuit

IMU Inertial Measurement Unit

JST Japan Solderless Terminal, series of different electrical connectors.

JTAG Joint Test Action Group, programming and debugging interface.

LDO Low Drop-out, type of voltage regulator.

MCB Motor Control Board, dedicated board to control the motors on the robots.

MLCC Multi-layer Ceramic Capacitor

MMD Monitor Mode Debugging, non-intrusive debugging technique.

NTNU Norwegian University of Technology and Science

PCB Printed Circuit Board

PPI Programmable Peripheral Interconnect, feature allowing for autonomous peripheral operations.

QFN Quad Flat No-leads, a type of IC package with pads instead of pins.

RTOS Real-time Operating System. E.g., FreeRTOS.

RTT Real-Time Transfer, protocol for real-time transfer.

SDK Software Development Kit, referring to Nordic Semiconductor's nRF SDK unless stated otherwise.

SES Segger Embedded Studio, an integrated development environment.

SLAM Simultaneous Localization and Mapping

SoC System-on-Chip, refers to the nRF52832 SoC in this thesis.

SPI Serial Peripheral Interface, serial bus for communication.

SWD Serial Wire Debug, programming and debugging interface.

SWO Serial Wire Output, trace data drain for streaming data.

UART Universal Asynchronous Receiver Transmitter, hardware for serial communication.

Chapter 1

Introduction

This chapter begins with a brief description of the ongoing SLAM project and then provides a summary of previous work done on the SLAM robots. This chapter also presents the motivation for the control system that is to be developed. The scope and objectives are defined, and the structure of the thesis is presented.

1.1 Background

The purpose of this section is to make the reader familiar with the SLAM project and the definition of the control system that is to be developed.

1.1.1 Simultaneous Localization and Mapping

This thesis is part of a series on simultaneous localization and mapping (SLAM) related projects dating back to its start in 2004 at NTNU. SLAM is the problem of mapping an unknown area using a robot. The problem is fundamentally a dual—that of creating a map, and that of knowing the robot’s position. While this problem has been successfully solved on a theoretical level, it is its implementation that remains the primary challenge. SLAM is a cornerstone in autonomous robotics operations, and as such it has never been more relevant with the recent advent of autonomous vehicles. As the SLAM problem itself will not be addressed directly as part of this thesis (ref. problem statement), the reader

should consult material such as Durrant-Whyte and Bailey's SLAM introduction for a more in-depth background.[1] [2]

1.1.2 Template Robot

At the outset of the project, the author was assigned one of the current SLAM robots. During the design and development of the new control system, parts of this robot and its control system have served as a starting point and reference. A picture of the robot can be seen in Figure 1.1.

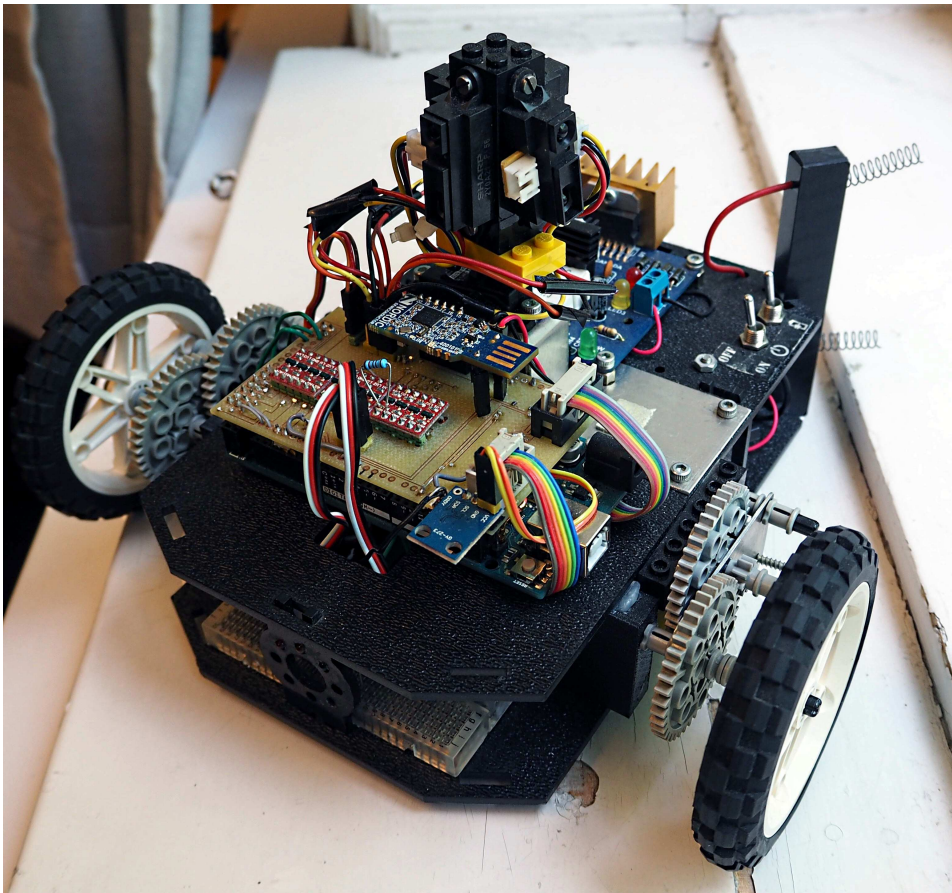


Figure 1.1: Template SLAM robot.

The robot consists of the following main subsystems:

- Control system

- Sensor tower: IR-sensors mounted atop a servo motor
- Motors: dedicated motor control board and one motor for each wheel
- Wheels with encoders
- 12V Li-Ion battery

There is no exact definition as to which components encompass the control system. The term will herein refer to those components that will be integrated within the new PCB that will be developed. For the template robot, this includes the stack of components and devices seen in Figure 1.1 and the IMU located below. In some sense, a more fitting term for this could be an integrated control system, as it will include components that are not performing any control logic. This selection of components is also the result of more practical consideration, that of: "which components are easily integrated into a PCB?".



Given the previous definition of the control system, a brief overview of the robot's functioning follows: In the current SLAM project, the actual mapping is performed on a central PC that runs a server communicating with one or more robots. The server orders a robot to move to a given location. The robot will then execute the movement order using the motors for the actual movement, and a combination of wheel encoders, IMU, and magnetometer to estimate its position. Either during, or after arrival at the requested location, a scanning sequence begins. The scanning sequence consists of moving the angle of the servo controlling the sensor tower in fixed increments, and for each increment, the IR sensor performs a distance measurement. This data is transferred back to the server that generates a map. This procedure repeats until the environment is fully mapped.

1.2 Previous Work

In order to both motivate and bring context to this project, a quick review of previous work done on the robots will be presented. In the timeline given in Table 1.1, it can be seen that the project spans many years, with many different students participating in the development. The information in the timeline is obtained from the project summaries given in Homestad's and Ese's theses [3] [4], as well as the original theses cited in the timeline itself.

The first robot built in 2004 was based on an ATmega32 in combination with LEGO Mindstorms RCX. The Mindstorms are Lego's robotics kit. The RCX was used as a

Table 1.1: Timeline highlighting all major control system revisions and redesigns.

2004	•	ATmega32 (Skjelten [5])
2008	•	NXT + ATmega48 (Bakken [6])
2015	•	EV3 (Stüper [7])
2016	• 	ATmega1284 (Ese [4])
2016	• 	ATmega2560 (Andersen and Rødseth [8])

dedicated motor controller. The project focused on assessing the performance of different sensors in SLAM applications.

During the summer of 2006, Lego released Mindstorms NXT, featuring more connectivity options and a more powerful built-in microcontroller. In 2008, a new robot was developed to take advantage of the NXT. The primary goal was to create a more stable and unified robot, as the NXT would be capable of handling more of the control functions itself.

A third-generation Mindstorms robotics kit, the EV3, was announced in early 2013. A new SLAM robot based on this kit was developed in 2015. Recently a project to port FreeRTOS to the EV3 platform was performed. [9]

In 2016 two new robots were developed, both based on ATmega microcontrollers. These are both running FreeRTOS. One of them has been designed around the ATmega 1284 microcontroller, while the other uses the Arduino Mega (AM). The AM platform is essentially an ATmega2560 on a PCB with rows of headers for prototyping. Also, it runs a special bootloader, making it possible to flash program the microcontroller directly via USB.

In parallel with this project there have been three other ongoing projects as part of the SLAM project:

- Transition from IR to LIDAR sensor. ATmega (AM)-based robot used.
- Improved robot navigation. ATmega (AM)-based robot used.
- More algorithmic work locally on the robot. ATmega1284-based robot used.

Since these are the robots currently used, and likely to be used for future projects as well, the specifications derived in Chapter 2 will be based mainly on the comparison of the new SoC with the AM and the ATmega1284 control system.

1.3 Motivation

Feedback from those who are developing on the current control systems is concerned with the lack of proper debugging facilities. For some robots, the only available aids in the debug process is performing print statements to an external computer, and the encoding of information via a set of three status LEDs. With the introduction of a complete RTOS on recent robots, progress has been severely curbed by the lack of any real-time supporting debug systems. The complexity of the system has increased exponentially, while the debug facilities have been at a standstill.

The current control system setup consists of an intermediate PCB to connect different modules together. These components could have been organized into a single PCB instead. While this separation is not in itself a major issue, the implementation seen in the template robot is. The system is fraught with frayed wiring and fragile connections prone to breaking at some point. This means that the application developer has to inspect and potentially perform measurements on hardware if the robot malfunctions. This distrust in the hardware is far from optimal.

A peculiar aspect of the current robots is that the control systems are based on older 8-bit architecture microcontrollers, while the Bluetooth communication dongle mounted on it contains a modern and powerful 32-bit ARM Cortex processor with significantly larger program and data memories dedicated for user applications.

By developing a new control system around a modern system with advanced debug functionality, the desire is to significantly accelerate the development of the SLAM application. It would seem as if more time is spent debugging, than on the SLAM challenge at this point. The expected increase in computing power will also aid and enable the current research into more autonomous robots.

1.4 Objective and Scope

A new control system will be developed with the current robots and the SLAM application in mind. This will involve system design, schematic capture, PCB layout development and embedded software development. The actual implementation of any SLAM algorithm or functionality is not within the scope of this thesis.

The RTOS template project implemented will serve as a starting point for the implementation of a SLAM application. It will include drivers and a minimal RTOS set-up using basic

synchronization mechanisms. Relevant debug facilities should also be implemented.

1.5 Outline

This thesis is outlined with two primary considerations in mind—reconstruction and understanding. The former should be evident by the thesis’ structure, while the latter will be more of an implicit manifestation throughout. The intention is not only to prepare the reader for reconstruction, but also provide sufficient understanding to enable the reader to modify and improve upon the system.

Each chapter starts with a brief introduction describing the chapter’s topic and outline.

Chapter 2 describes the system design phase of the project, where requirements and improvements are discussed and selected. In addition, this chapter serves as a theoretical primer where relevant.

Chapter 3 covers the electrical schematics capture and the design of a PCB layout.

Chapter 4 covers the PCB assembly and precision soldering techniques used. It also details electrical verification of the power supply.

Chapter 5 starts out with detailing the set-up of a development environment with RTOS aware debugging facilities. Following this the driver development is discussed, before a template RTOS project is set up using the drivers developed.

Chapter 6 presents the final status of the new control system.

Chapter 7 presents a project discussion and suggestions for further work.

Chapter 2

Specifications and Theory

This chapter starts out with an assessment of the SoC, comparing it with the current ATmega platforms with regard to central features such as peripherals, program memory, RAM, compute performance, and debugging facilities supported. This assessment will establish the SoC's capabilities, especially for SLAM applications.

The previous system configuration is reviewed, and a new one set up. This configuration involves the allocation of the SoC's resources such that all SLAM requirements are satisfied. New functionality is introduced and selected. A two-stage power distribution scheme is presented to power the new system. A brief introduction to basic concepts in the field of PCB design is given. The final section covers the new development environment that will be set up.

This section will also serve as a theoretical primer on the most central topics. This chapter is therefore intentionally detailed in some of its descriptions.

2.1 System-on-Chip

The nRF52832-QFAA SoC will be used for the new control system. This SoC is the next generation of the nRF51 used on the current robot's Bluetooth dongle. This section will introduce relevant features of this new SoC and assess its performance relative to the currently used ATmegs for the SLAM robots.

Figure 2.1 shows a simplified block schematic of the SoC adapted from a more detailed system overview that can be found in the datasheet. The primary components shown are:

- Cortex M4 CPU with floating point coprocessor (FPU)
- Program storage (Flash) and RAM (SRAM)
- Peripherals with DMA blocks if supported
- Bus system for interconnecting the different components
- Processor instruction cache (I-Cache)

In the following subsections, all of these components except the bus system will be discussed. Also, the debug facilities will be reviewed.

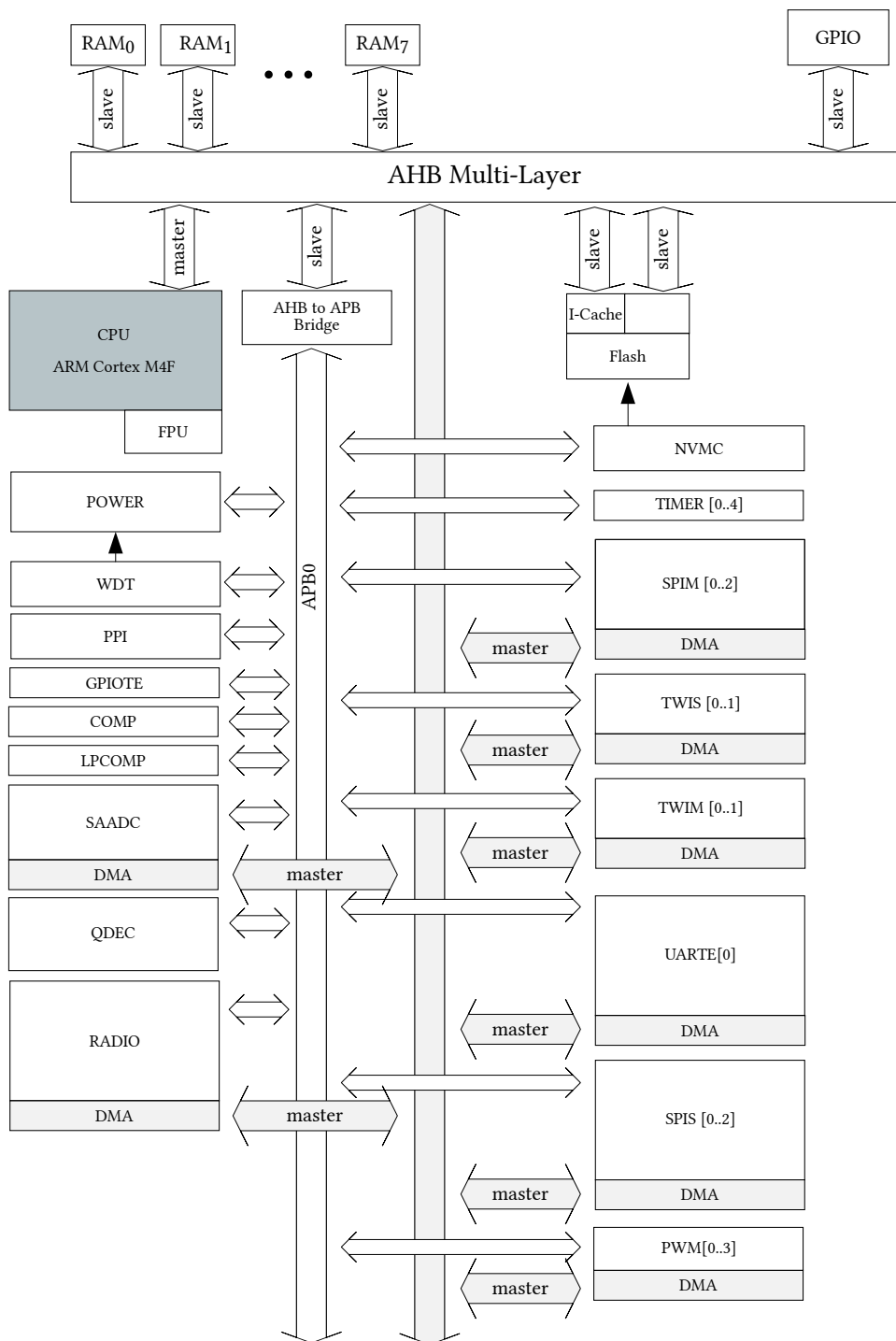


Figure 2.1: Simplified SoC overview, showing only the relevant subsystems.

2.1.1 Peripherals

Peripherals are devices that allow the microcontroller to interact with its surroundings, either via input, output or both. In the SLAM project, they are used for controlling motors and the sensor tower, and for reading sensor data from devices such as the IMU and wheel encoders.

It is essential that the SoC has peripherals that support all the different components of the SLAM robot, and that it has enough of these. Referring to Figure 2.1, a summary of the most central peripherals will now be described. It is worth noting that in contrast to the ATmegs, all peripherals except the SAADC can be configured to any physical pin as long as the pin is not dedicated to any special functionality (e.g., ground, power or antenna).

- The General Purpose Input/Output (**GPIO**) peripheral allows for configuring pins as either input or output. For example a switch connected to the SoC, where the GPIO peripheral can be used to read the status of the button on the pin it is connected to. The GPIO Tasks and Events (**GPITE**) is an extension enabling tasks and events on GPIO pins. Tasks and events are features that will be described at the end of this subsection.
- Two-Wire Interface (**TWI**) in slave (**TWIS**) or master (**TWIM**) configuration is a two-wire master/slave multidrop bus. It can be configured to conform to the Inter-integrated Circuit (I²C) standard.
- Serial Peripheral Interface (**SPI**) is a serial interface. In addition to the clock, there are two data lines and a slave select necessary for multi-drop configurations. The benefit of SPI over I2C is a more lightweight protocol and typically higher transfer rates.
- The timer (**TIMER**) peripheral supplies timers. These timers can be configured to operate in either timer or counter mode.
- The Pulse-width-modulator (**PWM**) peripheral uses a TIMER instance and a GPIO pin to output pulse-width-modulated signal. This type of signal is used for controlling the motors and the servo on the robot.
- The Successive Approximation Analog-to-Digital Converter (**SAADC**) converts analog input voltage to a digital representation. For the robot this peripheral is used for monitoring the battery voltage and perform readings from the IR sensors that output a voltage as function of measured distance.

The SoC incorporates two additional new features that complement the whole peripheral system. These are the Programmable Peripheral Interconnect (**PPI**) and Direct Memory Access (**DMA**). The former allows for supporting peripherals to set up events (e.g., ADC conversion complete) and tasks (e.g., GPIOTE sets pin activating LED), and then connect these tasks and events. The latter allows supporting peripheral systems to access memory directly via read or write operations. These features allows for basic and recurring routines in the peripheral system to operate without the processor intervening. Not relying on the processor also makes for a less complex real-time system, making timing analysis techniques more viable.

A comparison of the number of peripherals relevant to the SLAM application can be seen in Table 2.1. Keep in mind that this table does not take into consideration the different system's implementation of the peripherals, apart from mentioning resolution where relevant. For more details, the individual datasheets should be consulted.

Table 2.1: Comparison of number of peripherals available on the different platforms.

	SoC	ATmega2560	ATmega1284
Pins	34*	85	32
Timers	4‡ (32 bit)	2 (8 bit) + 4 (16 bit)	2 (8 bit) + 2 (16 bit)
PWM†	3x4 (15 bit)	1x4 (8 bit) + 1x12 (2–16 bit)	1x8 (8 bit)
ADC†	1x8 (12 bit)	1x16 (10 bit)	1x8 (10 bit)
UART	1	4	2
SPI	3	1	1
I2C	2	1	1

*All pins configurable with all peripherals except ADC.

†Hardware peripherals × Channels.

‡TIMER0 is reserved for the SoftDevice, thus only 4 timers are available.

2.1.2 Storage

Both the SoC and microcontrollers contain two types of memory, a program memory (flash memory) and a RAM (SRAM). The former is used for storing the program code, while the latter for storing run-time data such as variables. Table 2.2 summarizes the capacities for the different platforms.

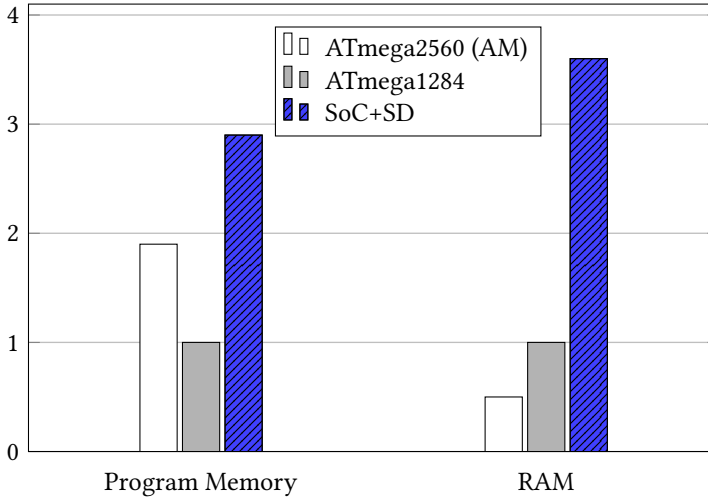
Table 2.2: Memory capacities for the different platforms.

Platform	ATmega1284	ATmega2560	AM*	SoC	SoC*	Unit
Program memory	128	256	248	512	372	KiB
RAM	16.384	8	8	64	59.12	KiB

*Bootloader and SD accounted for.

A direct comparison based on capacities given in the datasheets of the different systems will not be an adequate comparison. The ATmega control systems offload the Bluetooth operations to the dongle, meaning that these systems will have minimal storage overhead for handling this communication. The SoC has to manage this communication itself, and will thus need to hold a full BLE stack of drivers onboard (known as the SoftDevice). This stack requires a significant amount of program memory, and some RAM as well. The AM contains a bootloader, which occupies some program storage.

The memory capacities for the different platforms are compared in Figure 2.2, where the bar chart has been normalized around the ATmega1284. Highlighted in blue is the SoC with SoftDevice, which represents the configuration that will be used for the control system. For the ATmega2560, the Arduino bootloader has been accounted for in the comparison.

**Figure 2.2:** Memory capacities available to the application developer on the different platforms.

The program memory and RAM layout of the SoC are shown in Figure 2.3. The values

APP_CODE_BASE and APP_RAM_BASE parameters specifies the flash storage and memory requirements, respectively. The RAM requirements depends on the Bluetooth application, e.g., the number of services and characteristics.

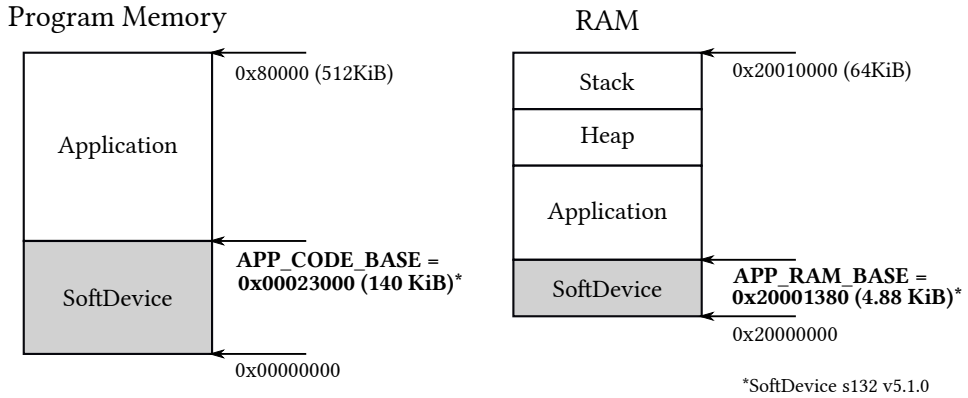


Figure 2.3: Flash and memory layout on the SoC with SoftDevice.

2.1.3 Compute Performance

There is ongoing work to move more of the computational workload from a central server to the robots themselves. As preliminary results have shown, the compute performance of the 8-bit ATmegs is likely to be a limiting factor. [10] It is therefore of interest to assess the compute performance of the SoC.

A brief comparison between the ATmegs currently used, and the processor part of the SoC will be presented here. The processor part of the SoC is an ARM Cortex M4F. It is important to stress that while both are RISC architectures, there are significant architectural differences between the 8-bit AVR architecture the ATmegs are based on, and the 32-bit Cortex. This architectural difference means that basing the comparison solely on the performance metrics given in the datasheets, such as clock frequency, is likely an inadequate basis for a comparison.

The ATmegs are based on the AVR architecture, while the Cortex on ARMv7E-M. Both the AVR and ARMv7E-M instruction sets are considered RISC, where each architecture features many instructions able to complete in a single clock cycle. [11] [12] The SoC operates with a clock frequency of 64MHz, while the ATmegs operates at 16MHz. The Cortex is a 32-bit architecture, while the AVR is an 8-bit architecture. Much simplified, this means that the SoC perform its operations on words consisting of 32 bits, while the

ATmegs will be limited to 8 bits. This is not to say that the ATmegs do not support wider data types, but this will incur the use of four more instructions unless the compiler manages to optimize it.

Consider the unsigned multiplication instruction MUL which both architectures implement. Both can execute this instruction in one cycle (assuming no wait states), but in the same cycle the Cortex can operate on operands in the range of $2^{32} - 1 = 4294967295$ while the 8-bit architecture of the AVR will be limited to $2^8 - 1 = 255$. Due to the differences in clock frequency the cycle times are also different. For the Cortex a clock cycle takes $T_{M4} = \frac{1}{64 \cdot 10^6} = 16\text{ns}$, while for the AVR, $T_{AVR} = \frac{1}{16 \cdot 10^6} = 63\text{ns}$.

CoreMark is a benchmark designed for embedded systems. It is written in C and consists of several algorithms aimed for assessing embedded processors pipeline, memory and integer operations performance. The SoC's Cortex-M4F implementation has a CoreMark score of 215 according to its datasheet. This is compared with a score of 4.25 for the ATmega2560 when operating with a clock frequency of 8MHz.¹ Unofficial results at EEMBC is used for the ATmega, which means that this score may be unreliable.

A major difference between the Cortex and the ATmegs is the presence of a dedicated floating point coprocessor (FPU) on the Cortex. This unit can handle single-precision floating-point number operations directly in hardware. It also introduces several new DSP instructions such as the Fast-Fourier-transform algorithm as well as some single instruction multiple data (SIMD) instructions. In the ATmega microcontrollers, floating point operations have to be implemented in software, as there is no hardware support for these operations. It is thus generally best to avoid such instructions if possible. It was mentioned in Eikeland's thesis that better support for floating point operations would be useful when moving more of the algorithmic work from the central server to the individual SLAM robots. [10]

The author would like to stress that he is not in any way attempting to dismiss the AVR architecture as being inferior or obsolete. Instead, the issue at hand here is its intended application. For high volume productions of simpler control systems, the ATmega could be more than sufficient at a lower price point with reduced system complexity. For the SLAM project, especially considering that more of the algorithmic work is now being moved to the robot, the simplicity of the ATmega seems to become more of a limiting, than enabling factor.

¹Unable to obtain a URL: navigate to eembc.org, Benchmarks → Processors → CoreMark → Scores → Search, select *Atmel* as vendor, and the ATmega2560 should appear in the filtered list.

2.1.4 Debugging

A concern that has been raised on several occasions is the lack of proper debugging facilities in the SLAM project. The ATmega2560 control systems has generally been limited to printing via UART. The ATmega1284 based robot uses the JTAG interface of the microcontroller, which means that single stepping and halting is available. While this is an improvement over printing via the UART peripheral, there are two major limitations to this debugging as well: they lack any RTOS awareness, and they can break timing sensitive systems due to their invasive nature. The goal of the debug system on the new control system will thus be to provide real-time aware and non-invasive debugging facilities.

The nRF52 incorporates several debugging features, which are briefly summarized below:

In addition to halt debugging, the Cortex supports a second mode called Monitor Mode Debugging (**MMD**). This is a special debug mode where debug events triggers a debug exception handler. The implication of this is that the exception handler will execute at a given priority, such that higher level priority exceptions will still be able to run. This less intrusive debugging mode will not interfere with critical time-sensitive operations, such as Bluetooth or motor control.

Serial Wire Debug (**SWD**) interface, Serial Wire Output (**SWO**) and Serial Wire Viewer (**SWV**) are different but interacting technologies. SWD defines a two-wire interface for debugging and flashing program memory. The SWO is an additional pin, which when combined with SWD enables SWV operations. SWV can then be used for real-time trace and system analysis without the need to halt the processor to extract the information. The type of information can be event counters and notifications, timestamps and CPU cycle information. For the control system it is used for transferring trace data to an external PC running software to perform RTOS analysis.

Flash Patch and Breakpoint unit (**FPB**) enables hardware breakpoints. Hardware breakpoints are as their name implies integrated into hardware (special registers), and as such have a smaller overhead associated with them compared with software breakpoints. The downside is that there is usually a very limited amount of these registers, meaning that only a few hardware breakpoints (e.g., 6) can be set versus infinitely many software breakpoints.

Data Watchpoint and Trace unit (**DWT**) enables data watchpoints and different tracing functions. Data watchpoints are breakpoints that trigger when specific addresses are accessed.

Instrumentation Trace Macrocell (**ITM**) enables communication via the SWO pin discussed in the first bullet point. An application can write directly to the ITM, which then encapsulates these messages and attach timestamps such that external debugging software could as an example analyze events generated over time. The `printf()` function may be redirected to the ITM allowing for a real-time friendly debug print functionality as the ITM output does not cause much delay such as the traditional UART peripheral does.

Embedded Trace Macrocell (**ETM**) provides full instruction traces, at the expensive of additional pins that have to be used to support the potentially much higher transfer rates required.

For the control system, most of the functionality mentioned above will be available, except for the ETM. The reason for omitting the ETM is twofold, first of all, it will require more GPIO pins than the control system can spare. Secondly, it will need much more expensive debugging hardware (e.g., J-trace) than what is currently available. A strategy for leveraging these debugging features of the SoC is presented in Section 2.6.

2.1.5 Summary

This section has presented the relevant peripherals available on the SoC, and described PPI and DMA, two complementing technologies enhancing the complete peripheral system. The storage capacities of the Soc has been detailed, and a thorough comparison with the ATmegs has been presented, where bootloaders and SoftDevice has been accounted for. An attempt to assess and compare the computational performance is presented. A brief overview of the different debug features has been presented. In conclusion the SoC seems to be a very promising candidate to replace the ATmegs; on paper it outperforms the ATmegs in almost all aspects.

The new SoC with the Cortex-M4F is a complex architecture. Fortunately, official SDKs are available, which should make development on the platform easier regardless of the underlying architecture. For more information on developing on the SoC, and its architecture, the online resources Nordic Infocenter and Nordic Devzone are invaluable resources.

2.2 System Configuration

This section deals with the challenge of allocating the SoC's resources in a manner such that both new functionality and interoperability with current robots are ensured. The new

features—getting slightly ahead of oneself—will also have to be considered here. As these will be covered individually later in this chapter, it is sufficient to take note of their pin and peripheral requirements for now. While the different components that form a complete SLAM robot were briefly mentioned in Section 1.1, a more comprehensive review will now be given.

As a minimum, the new control system must be able to interface with the sensors and actuators currently used in the different robot configurations. Figure 2.4 is an overview of the components associated with the current control systems, where the dashed lines indicate that the surveying sensor can be either a set of 4 IR sensors or a single LIDAR. The LIDAR has recently been introduced to the project to replace the currently used IR sensors. The robots are likely to transition to LIDAR-only in the future, but as the SLAM project is presently in this transition, the new control system will have to support both.

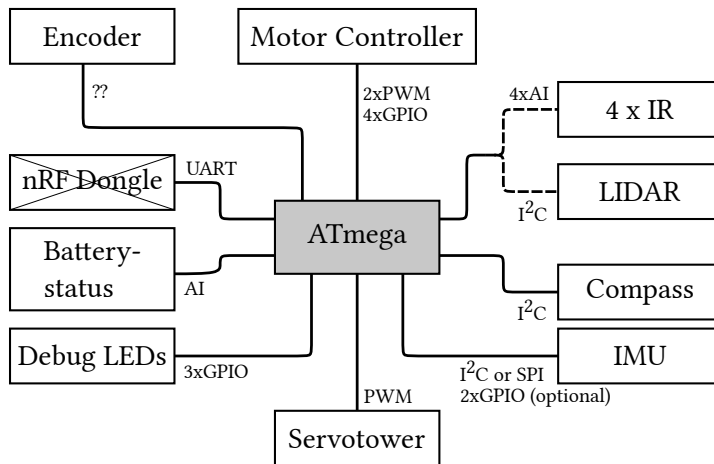


Figure 2.4: Components and peripheral usage of current control system.

Figure 2.4 lists the peripheral requirements for each of the components, such as ADC for the IR sensors and PWM for servo control. In Table 2.3 all components part of the current and new SLAM control system are listed. Their peripheral requirements, and how many physical pins these requirements translate into are given.

An immediate problem looking at Table 2.3 is the sum of physical pins required, 38-40 in total. This is more pins than the 32 available on the SoC for peripherals. As a first measure the debug LEDs are removed from the new system. With the introduction of a display and much more advanced debug facilities, these are unlikely to be necessary.

The multi-drop feature of the I2C bus lends itself to the pin saving effort. While I2C may

Table 2.3: SLAM robot peripheral and pin requirements.

Component	Peripheral	Physical Pins	On-board
Motor Controller	2xPWM* 4xGPIO	6	No
IRs	4xADC*	4	No
LIDAR	I2C	2	No
Encoders	2xGPIO	2	No
Battery Status	ADC	1	–
Debug LEDs	3xGPIO	3	No
Sensortower	PWM	1	No
IMU†	SPI + 4xGPIO	4 + 4	Yes
	I2C + 4xGPIO	2 + 4	
Reset Button	1xGPIO	1	Yes
Magnetometer	I2C + 1xGPIO	3	Yes
Display	SPI + 2xGPIO	5‡	Yes
microSD	SPI	4	Yes

*Shared channel is acceptable.

†GPIOs for IMU are interrupt pairs for acc. and gyro, i.e. optional.

‡MISO is not applicable for display, it only receives data.

handle more than 100 slave devices on a single bus, only the onboard components will be set up on a shared I2C bus. Considering that the address and pull-up configuration of external devices are unknown, it is safer to dedicate a new I2C bus for these devices. If the pin shortage was more acute, adding these external devices to the same bus could have been a viable option.

As both the display and microSD slot have each their chip select, the MOSI line can be shared. As tempting as it may seem, even if they had separate data lines, merely pulling the chip select active with a resistor is not an option. For SPI communication the chip select is often part of the protocol definition. With the configuration and changes so far, the pin usage has dropped to 34 pins.

The final change is reducing the number of interrupts from the IMU. It comes with four interrupt lines, two for the integrated accelerometer, and two for the integrated gyroscopes. According to the datasheet both the accelerometer and gyroscope are equipped with several

programmable interrupt engines (e.g., configuration of latching and timeout values), but all interrupt events can be mapped to either of the two associated interrupt pins. For efficiently obtaining sensor readings from the IMU, it is assumed that it is sufficient with one interrupt line for each of the two sensors. With this final cutback, the 32 pins on the SoC is now sufficient.

In case more pins would be necessary at a later point, there are still more cutbacks possible. The external I2C bus can be connected to the internal, saving two pins. A jumper could be introduced such that it could be selected in hardware whether the LIDAR or IRs should be used. This jumper would save two or four more pins.

The final set-up can be seen in Figure 2.5.

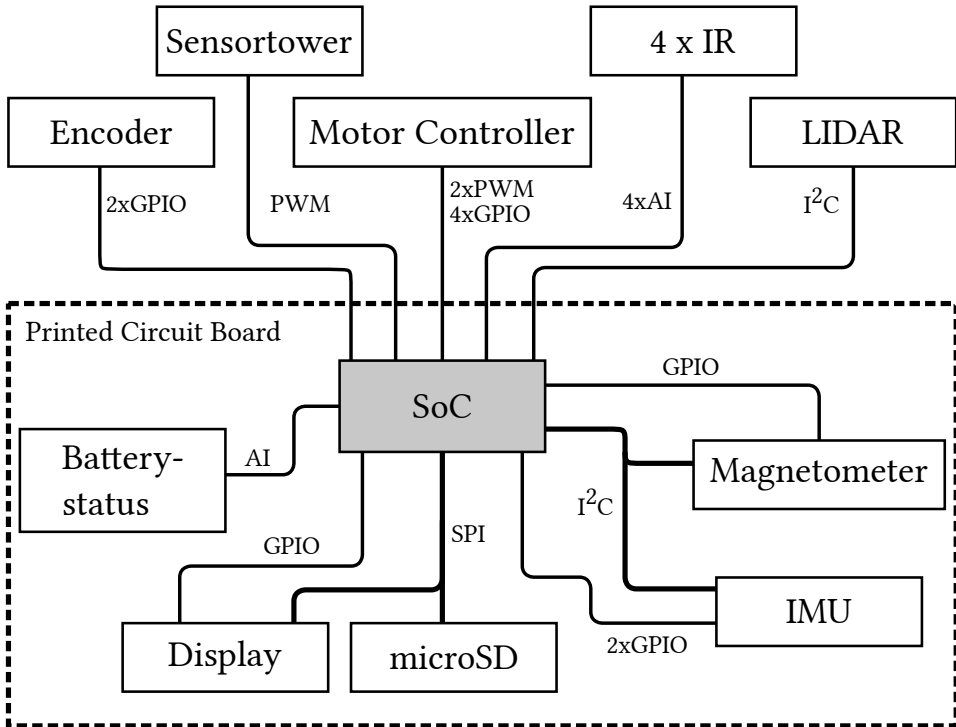


Figure 2.5: Components and peripheral usage of new control system.

2.3 New Functionality

While ensuring compatibility with existing robots in the design of the new control system is important, it is also an excellent opportunity to introduce new features and functionality. Since the components making up the new features will not be removable, and also shared by all robots, thought has to be given to a feature's applicability in current and future SLAM projects.

2.3.1 Display

With the aforementioned applicability in mind, the first and most obvious feature will be a display. While those robots using the default LEGO controllers are already equipped with LCDs, none of the custom ATmega based designs are. Instead, they are limited to a set of three LEDs. An LCD will facilitate easier and more informative debugging and logging during live SLAM runs.

The display selected is a monochrome 1.3" 128×64 OLED. It measures 35mm by 35mm. Despite its small form factor, a compact typeface will allow for several lines of information. It comes packaged in a breakout board, and will therefore be mounted and soldered to the main PCB via a header row. It may be necessary with a fixture to support it. Fortunately, the resulting standoff-height also means that the surface area below the display will be available for routing and low-profile SMT components.

While many displays come with a parallel interface, this display communicates over SPI.

2.3.2 Portable Storage

Among the most significant upgrades to the new control system is the new RTOS trace functionality. It will be detailed at the end of the section, but the gist of it is logging of all events and actions in the operating system. One mode of operation is where the trace system continually stores the event data into RAM, and it can then be uploaded to a PC for analysis afterward. The problem is that these traces may consume large amounts of RAM depending on how long the tracing runs, which means that the limited RAM available on the SoC can severely limit the trace duration.

By implementing a microSD slot, the trace data can be stored on a microSD-card. Unlike the RAM that has less than a MiB of storage available, microSD cards come in capacities of many gigabytes. They are also highly portable, making it easy to transfer the data

for analysis on a PC. Also, as part of the SoC's SDK is a library for using SD cards. This library supports a file system with individual file management.

The addition of portable storage means that traces can be run for hours, or even days, without having to stop. Tracing over larger time spans is important to resolve *heisenbugs*, which are bugs often associated with real-time systems. These are timing sensitive bugs that rarely manifests into errors (e.g., only every thousandth time).

The SLAM application also involves a lot of sensor data. Since there is ongoing work to move more of the SLAM algorithms over to the robots themselves, it could be of interest to log data locally on the robot and dedicate the BLE transfers to mapping data only. Or in some cases, the robot could generate larger maps entirely autonomously, and store all log data as well as generated maps locally.

2.3.3 Connectors and Test Points

An additional—and highly desired—feature of the new control system will be the usage of proper connectors and receptacles for most external components.

The picture of the current AVR robot in Figure 2.6 highlights an issue with the contemporary design: much of the wiring is done in a fragile style, making the whole robot prone to disconnects and shorts. The poor electrical wiring is especially unfortunate considering there is no solder mask on the junction PCB that is mounted and connected on the top of the AM.

The first set of new connectors will be power connectors connecting the battery to the control system. The wires from the battery are 0.75mm^2 in cross section. The JST VH series is one of the few wire-to-board connectors from JST that can directly crimp onto wires with a cross section this large. The VH series is also designed for power supplies, making it a good match. A comparison between the old battery connectors and the new can be seen in Figure 2.7.

The IR-sensors wires are already connected using 3-pin JST PH connectors on the component side. The same PH series 3-pin connectors are chosen for the control system board to match the IR sensor component. The LIDAR also uses a JST connector, but a 6-pin GH series type. Again, the control system board will match this setup exactly. Drawings of both connectors can be seen in Figure 2.8.

Amsen suggests in his thesis that for further work the Bluetooth dongle should be properly connected. He also specifically suggested improvements to the connectors for the battery

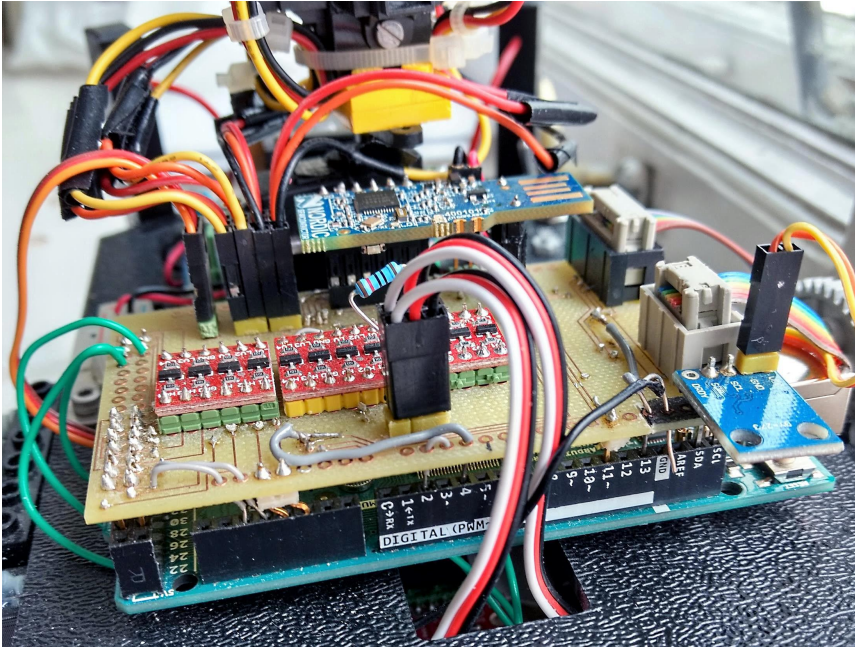
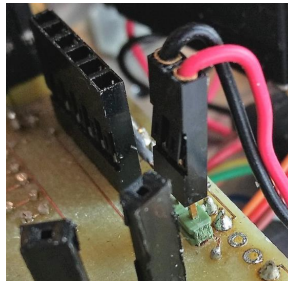


Figure 2.6: Picture of control system of current ATmega robot, with emphasis on wiring issues.

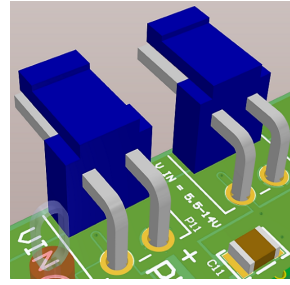
and sensor tower. [13] The Bluetooth dongle is no longer an issue, as it is part of the SoC integrated into the PCB. All IRs on the sensor-tower and battery connectors will be upgraded with proper connectors as was outlined in this section.

Using proper connections makes for a more robust control system and robot, but it can also make it more difficult to probe and perform electrical measurements. With the previous control systems using headers and open traces, it was easy to probe with a multimeter or oscilloscope, as there are many places with exposed copper. Alternatively one could add a jumper wire to one of the headers for debugging purposes. Opting for more robust connectors, this is one of the trade-offs: there will be fewer spots suitable for probing, and the insertion of intermediate jumpers may not be a feasible option.

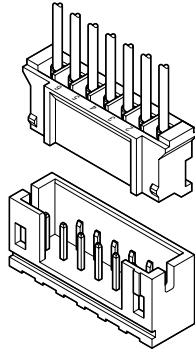
Dedicated test-points will be set up to facilitate debugging using multimeters and oscilloscope. These are through-hole components consisting of two parts: one conducting pin that is connected to a track on the PCB, and one conducting metal loop for probing. The loop structure makes it possible to use clip-on probes. These test points are then added to central lines of the PCB. These test points can be seen both in the 3D-render of the PCB in Appendix A, and the final picture of the PCB in Figure 6.1 in Chapter 6. The black test-point is connected to GND, the red ones to 3V3 and 5V, and the blue ones to the I2C



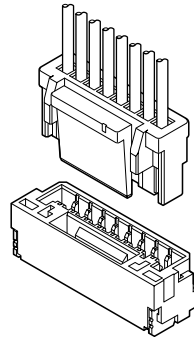
(a) Previous connectors.



(b) New connectors.

Figure 2.7: Comparison between previous and new power connectors.

(a) PH series.



(b) GH series.

Figure 2.8: JST connectors for LIDAR and IR sensors. Drawings from datasheet.

serial buses.

2.4 Power Supply

All components on the robot operate at either 3.3V or 5V, while the battery has a nominal supply voltage of 11.1V and a peak cut-off (after charge) voltage of maximum 13.05V. This introduces the need for a two-stage voltage reduction scheme, as has been illustrated in the power distribution in Figure 2.9.

The voltage regulation on the control system will consist of two stages. The first converts the battery input voltage down to 5V, and the second regulator brings this 5V down to 3.3V. An implication with this architecture is that the first stage not only has a greater

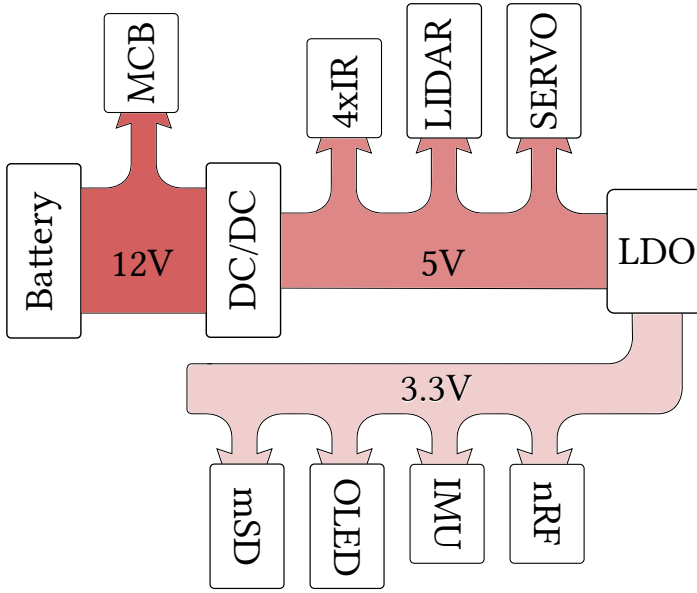


Figure 2.9: Power distribution scheme.

drop in voltage across it, but it also has to supply more current. This gives a significantly increase in power throughput compared to the second stage regulator. Therefore the first stage will consists of a buck-down (hereafter referred to as DC-DC) converter, and the second stage a simpler low dropout (LDO) regulator.

Before selecting a voltage regulator, current consumption must be estimated. In Table 2.4, a rough current consumption estimate is presented. Knowing the voltage down regulation levels and estimated maximum current consumption, a suitable DC-DC and LDO regulator can be selected for the two stages.

2.4.1 Component Selection

For the first stage a Murata OKL-T/1-W12 series DC-DC module is chosen. Besides fulfilling the requirements on voltage regulation and current supply, it has several other benefits. First and foremost it comes as a complete module containing both the inductor and switcher, which means that no time has to be spent designing the actual buck converter. Furthermore it has built-in soft-start, short circuit and over current protection, and features on/off control as well. The module's primary specifications are given in Table 2.5, where it can also be seen that it comes in an inspectable land grid array package, which will aid

Table 2.4: Control system current consumption estimates.

	Component	Current Consumption
3.3V	nRF	20mA
	IMU	15mA
	OLED	60mA
	microSD	80mA*
	LEDs	16mA
	SUBTOTAL	211mA
5V	$4 \times IR$	120mA†
	LIDAR	135mA†
	Servo	500mA‡
	TOTAL	846mA

*Toshiba High Speed M102 microSD card specifications.

† For robot configurations, mutual exclusive.

‡ Highly dependent on torque. Stalling, i.e., worse case assumed here.

the soldering process.

For the second stage a Texas Instruments LM3940 LDO regulator is selected. It is designed for an input voltage between 4.5V and 5.5V, and supplies a fixed output voltage of 3.3V. Primary specifications are given in Table 2.6, where the small-outline-transistor-223 (SOT-223) package it comes in can be seen as well. Since the voltage down regulation and current draw is less for the second stage, using an LDO regulator is considered acceptable.

Table 2.5: DC-DC module used for the first stage.


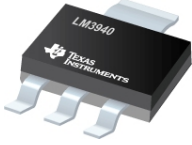
	Model	Murata OKL-T/1-W12
	V_{in}	2.9 – 14V
	V_{out}	0.9 – 5.5V
	I_{max}	1A
	Features	Short circuit protection, On/off control
	Package	Custom LGA (inspectable)

Table 2.6: LDO used for the second stage.

Model	Texas Instruments LM3940
V_{in}	4.5 – 5.5V
V_{out}	3.3V
I_{max}	1A
Features	Short circuit protection, thermal protection
Package	SOT-223

2.4.2 Performance

An inherent attribute of DC-DCs operating on the buck-down principle is ripple on the output voltage. Since the working principle of a DC-DC converter involves high frequency switching, there will be a ripple component on the output voltage. This ripple component is usually measured by its peak-to-peak value, and will here be referred to as the $V_{P_{kPk}}$ voltage.

The manufacturer has included several measurements for different operating conditions in the datasheet, where the closest match for the control system is the down regulation from 12V to 3.3V at 1A. The peak-to-peak ripple voltage is listed to be $V_{P_{kPk}} = 27\text{mV}$. It should be mentioned that the manufacturer performed the ripple tests with an output capacitor of $C_o = 10\mu\text{F}$, while in the control system a capacitor with capacitance of $C_o = 22\mu\text{F}$ will be used. Considering that the voltage down regulation is lower in the control system ($11\text{V} \rightarrow 5\text{V}$ opposed to $12\text{V} \rightarrow 3.3\text{V}$ in data sheet) and the output capacitor is larger, it is likely that the actual peak-to-peak voltage will be significantly lower than the listed 27mV.

All LDO regulators have some degree of ripple rejection. This means that they will to some extent reject the ripple on the input voltage, such that this ripple voltage is attenuated on the regulated side.[14] The details of this rejection property will not be covered here, it is suffice to say that it is a frequency dependent factor that is usually expressed as a dB ratio, known as:

$$\text{PSRR} = 20 \log \frac{V_{pkpk_input}}{V_{pkpk_output}} \quad (2.1)$$

In Figure 2.10 the PSRR chart of the LM3940 LDO regulator is reproduced with the frequency of the output of the DC-DC converter indicated. Unfortunately this seem to be a resonant frequency for the LDO regulator, but it should still yield at least a 25dB

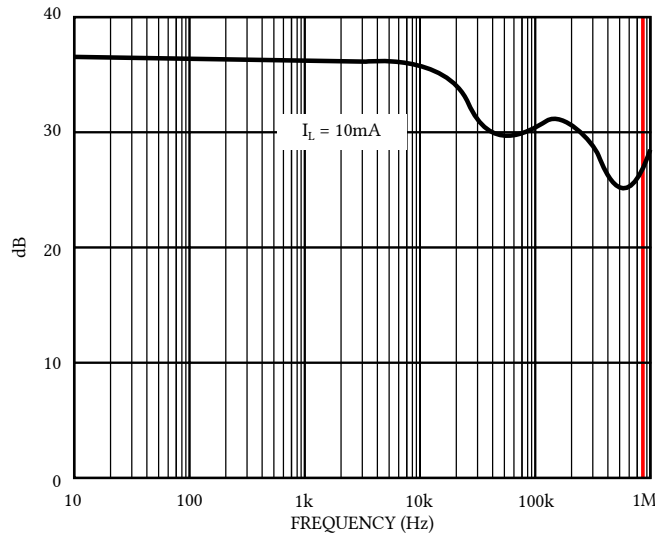


Figure 2.10: LDO Power Supply Rejection Ratio as a function of frequency.

rejection of the input ripple. The following calculation then gives the output (attenuated) ripple voltage

$$V_{ppk_output} = \frac{V_{ppk_input}}{10^{\frac{25}{20}}} \approx 1.52\text{mV} \quad (2.2)$$

According to a Nordic Semiconductor representative, the SoC supply voltage can have a maximum ripple of 100mV.² Referring to the preceding calculations, this should not be an issue. Since the calculations may deviate significantly from real-world performance, proper testing and verification of the voltage regulation system will be performed when operational.

2.5 PCB Fundamentals

This section will cover some printed circuit board (PCB) fundamentals. There are a few basic concepts and associated terminology that the reader has to be familiar prior to any PCB development.

Apart from the two external signal layers, a PCB can have any number of internal layers as well. This is referred to a PCB's board stack-up configuration. Among the simplest is

²<https://devzone.nordicsemi.com/f/nordic-q-a/15064/power-supply-requirements-for-nrf52832>

the usage of one or two layers, i.e., bottom and top layer. Another common configuration is a four-layer board stack-up, two internal, plus the top and bottom. Other than the signal layers there is the solder-mask and silkscreen. The solder-masks facilitates easier soldering as only copper pads that are to be soldered on are exposed. The silkscreen is an annotation layer that usually contains the text with component designators and outlines on the PCBs.

In the 4-layer configuration it is common to dedicate one of the internal layers as a pure ground plane, and the other as a dedicated power plane. This design ensures low impedance paths to ground. No signals are sent across the internal layers. This configuration can be seen in Figure 2.11.

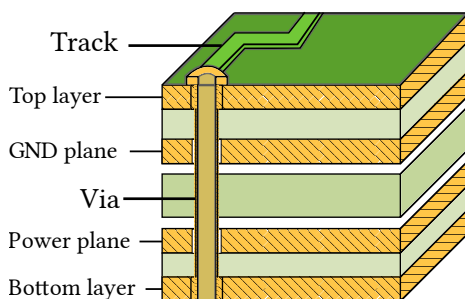


Figure 2.11: Primary components of a PCB in four-layer configuration.

When components are placed on the PCB, tracks are placed in order to connect them. Tracks are copper pathways on the PCB. Since the PCB consists of multiple layers, a mechanism for inter-layer connections is needed. The solution is vias. Vias are holes that are drilled, and chemically plated with copper. There are many types of vias, e.g., buried vias where the drill cavity is internal, or blind vias, that extend from the surface to a set distance into the board. The most common is through-hole vias, where a hole is drilled all the way through and is then copper plated. Those layers that should not be connected to a through-hole via removes copper where this via passes through them. An example is given in Figure 2.11.

The components are supplied in two categories of packages—either surface mount or through-hole mount. Surface mount components may have pins extending slightly from the package, or may only have a set of pads. Through-hole components come with pins that extend through the whole board and are soldered on the opposite side of where they are inserted. The trend is for more smaller surface mount packages, reducing board space requirements and exhibiting better noise and interference properties.[15]

2.6 Development Environment

The development environment will in this thesis refer to the environment used for software development, device programming and debugging. First, the underlying development process is described, a process that will be common to any configuration of development environment. Following this, software and tools are selected to achieve a modern and powerful development environment.

2.6.1 Build Process

To get any code to run on the SoC, the code has to be preprocessed, compiled, linked and flashed into the program memory of the SoC. This is a rather intricate multi-step process. Fortunately, there are tools for each of these steps, and a collection of tools to perform all of these tasks is referred to as a toolchain.

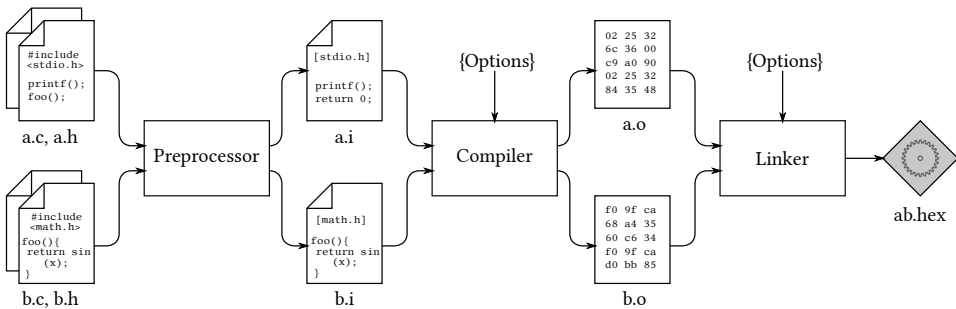


Figure 2.12: Steps involved in generating an executable for target.

Referring to Figure 2.12, the first step is to feed the source code to the preprocessor. The preprocessor will expand all the `#includes`, and replace `#define` macros throughout the source files. Following this, the compiler translates the source to the target architecture assembly language, which in turn is fed to an assembler that emits the final machine code that the target processor can execute.³

As seen in Figure 2.12, the code has been split into two sets of files. This way, only the modified set has to be recompiled. The compiler will then output separate machine code files for each of the sets. The linker will go through the interactions between the files, and link the separate files into one common executable. The final result is the `.hex` executable.

³The compilation and assembly are two different processes, but are not distinguished here for simplicity.

The final step is shown in Figure 2.13. Here the previously compiled and linked *.hex*-file is written to the program memory of the target. For this last step, specialized programming hardware is necessary, usually designed for a specific family of target platforms.

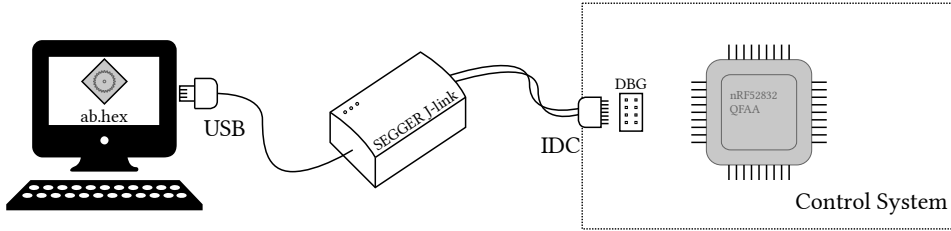


Figure 2.13: Final process of transferring the program to the target.

Knowing this process and the tools involved will not only aid the developer in understanding error message during compilation. Due to the existence of free and open software making up the toolchain, it ensures that the developer will always be able to fall back on these tools should the licensing terms or availability of an IDE change with time.

2.6.2 IDE and Flash Programmer

An integrated development environment usually comes with a text editor, toolchain, debug support and possibly version control features as well. In this thesis, the choice of IDE has fallen on Segger Embedded Studio (SES), which has all of these features. Among the benefits of using this IDE are the following:

- Cross-platform, supporting Windows, Linux and even macOS.
- Free for non-commercial use, and a commercial license free of charge when used with Nordic Semiconductor products.
- Nordic Semiconductor's SDK comes bundled with examples containing fully configured SES project files.

The equipment for flashing new programs to the SoC, and perform debugging will be the nRF52 development kit. This development kit is equipped with an onboard J-link debug probe, which can be used to program external devices. Compatibility with SES should be ensured, as it is the same company that has developed both. As several development kits are already available as part of the SLAM project, this means that there will be no additional expenses associated with debug or program flashing hardware.

2.6.3 Debugging Facilities

The topic of debugging was briefly mentioned in Section 2.1, where the debug support of the ARM Cortex processor part of the SoC was listed. In this section, the strategy for implementing and leveraging some of these supported mechanisms will be described.

The company behind SES, Segger, is also the developer of the hardware used for flashing and debugging the control system. The result of this is that three debug facilities should be available immediately as part of SES: traditional halt-mode debugging and monitor mode debugging (MMD).

Real-Time Transfer (RTT) is a technology for high-speed communication with a Cortex-M debuggee while maintaining the target's real-time performance. It uses the SWD described previously, in combination with ARM's *semihosting* functionality which is a mechanism that makes it possible for the Cortex to use the host PC's I/O facilities. In other words, it will mainly be used for setting up a virtual terminal on the host PC. It has support for several channels, which could, e.g., be set up for standard output and error output. As will be seen in Chapter 5, Nordic Semiconductor's logging library part of the SoC's SDK can easily be configured to use this RTT technology as its back-end.

The trace features of the SoC will be leveraged using Percepio's Tracealyzer. Tracealyzer is a software package dedicated to RTOS trace visualization and analysis and will give complete insight into scheduling, synchronization mechanisms and many different types of operating system events. While the software package originally costs about 1700USD, the company supplies licenses free of charge for educational purposes.

Chapter 3

Schematics and Layout

Both the electrical schematics and the PCB layout are made in Altium Designer, an electronic design automation software designed for PCB development. The chapter starts with an brief introduction to AD, before covering details regarding the component library. Following this the schematic capture is described, which forms the foundation for the four-layer PCB layout design detailed in the subsequent section.

The complete development cycle can be seen in Figure 3.1, where the stages relevant for this chapter are highlighted.

3.1 Altium Designer

Altium Designer (AD) is a complete software package for library management, schematic capture, PCB layout, component design, bill-of-material, and many more advanced features such as electrical simulations. All of the electrical schematics, and the PCB layout designed in this chapter, are made within the integrated AD environment.

The workflow used in this project is to first develop and set up a components library with all components used in the control system. A detailed guide for the development of such

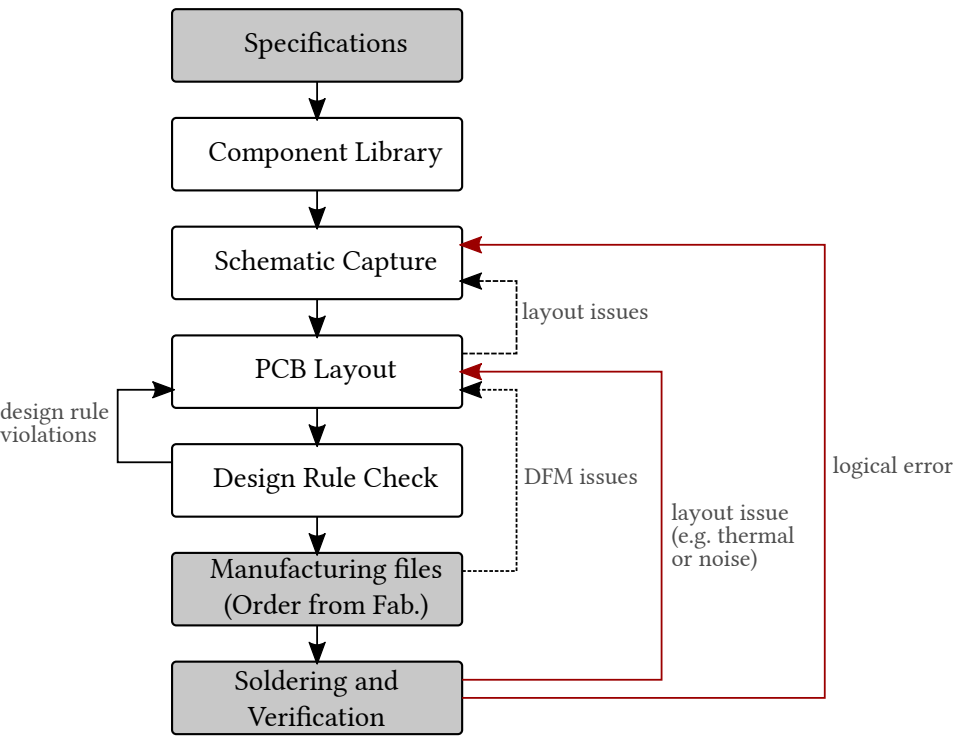


Figure 3.1: Flowchart illustrating the complete PCB development cycle.

a component library is outlined in the media attachment, as described in Appendix A. Figure 3.2 illustrates the different parts that composes a component.

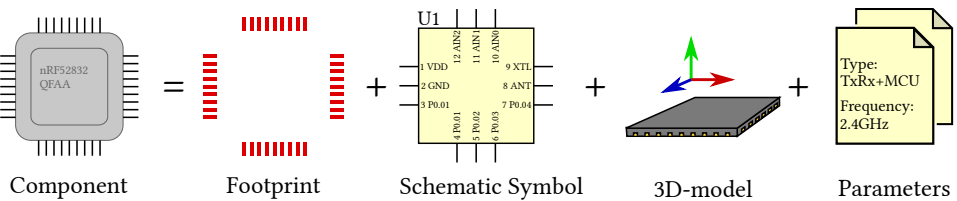


Figure 3.2: Parts constituting an AD component.

The component library is then used when developing the schematics in the schematics environment of AD. When the schematics are done, work continues in the PCB editor environment. In the PCB editor environment the board layer stackup and dimensions are set. The components and net list are imported from the schematics, and the complete PCB layout is made. The net list associate the different pins of the components with each-other, such that AD can guide the designer when making connections between pins. In this PCB

environment the footprint and 3D-model of the component is used.

In reality the workflow can be far more complex. For example, operations can be performed across the footprint editor affecting components in the PCB environment, and *vice versa*. A whole book could be written on the usage AD, something which the countless pages of bundled documentation proves. Since the topic of this thesis is not AD itself, the reader is encouraged to refer to one of the many great resources available on the Internet for guides on its usage. A good starting point would be Altium's own hands-on-tutorial. [16]

The balance between writing for reproduction and brevity has been a concern for this chapter. A compromise is selected, where only the function used in AD is specified, and the reader can then consult the official documentation for details on its usage. A great feature of Altium is that basically all functions are available via a set of context aware keyboard shortcuts (i.e., changes based on which environment is currently active in AD). This feature will be taken advantage of when referring to the functions in this chapter. These keyboard combinations make for a compact form, for which an example comparison can be seen here:

Tools → Convert → Explode Component to Free Primitives

is instead expressed in the more succinct form:

t → v → c

The key combination given will then be valid for the environment currently discussed in that section. E.g., in the schematics capture section it will be the schematics editor environment, while in the PCB section it is the PCB editor environment.

3.2 Component Library

A library containing all the components used in the control system is set up in AD. This section briefly the details of some component requiring special considerations. The component definition was given in the previous section, and is summarized in Figure 3.2. As stated in that section, a guide to creating components efficiently is part of the media attachment, described in Appendix A.

3.2.1 Low Drop-out Regulator Heatsinking

The LDO's datasheet specifies that heatsinking may be necessary, depending on the amount of power it has to dissipate, which is a function of voltage drop and current. The current flow was estimated to be $\approx 211\text{mA}$ for the 3.3V regulator in Chapter 2. The total amount of power dissipated is given by the formula in Eq. 3.1, where I_G is the ground current. The ground current is assumed equal to the quiescent current, which is current consumed for operating the LDO. [17] It is estimated to be $I_G = I_Q \approx 35\text{mA}$ based on values given in the datasheet.

This results in $P_D = 533.7\text{mW}$ being dissipated. Next Eq. 3.2 is used to calculate the maximum allowable temperature rise $T_R(max)$. Here $T_J(max)$ is the maximum allowable junction temperature, which is 125°C for commercial grade parts, and therefore also assumed valid for the control system. $T_A(max)$ is the maximum ambient temperature, which is assumed to be 25°C for the SLAM robots. This results in $T_R(max) = 100^\circ\text{C}$.

The final formula given in Eq. 3.3 calculates the maximum allowable junction-to-ambient thermal resistance, i.e. the environments resistance to conduct heat away from the LDO. This resulting maximum resistance is $R_{\theta(JA)} \approx 188^\circ\text{C/W}$. As this value is greater than 59.3°C/W , the SOT-223 package will not require a heatsink in the current set-up according to the datasheet.

$$P_D = (V_{IN} - V_{OUT})I_{OUT} + V_{IN}I_G \quad (3.1)$$

$$T_R(max) = T_J(max) - T_A(max) \quad (3.2)$$

$$R_{\theta(JA)} = T_R(max)/P_D \quad (3.3)$$

The thermal reliefs on the LDO's ground pads are omitted. This effectively makes the PCB function as a heatsink for the LDO, ensuring good thermal performance and enables support for larger currents. The difference between thermal relief connections, and no thermal relief connection is shown in Figure 3.3.

Thermal relief connections are useful to ease the soldering process, as less heat will have to be applied during the soldering process. This means that hand-soldering the LDO might be challenging, but using a reflow oven should still work well, as in this oven everything

is heated to the reflow temperature.

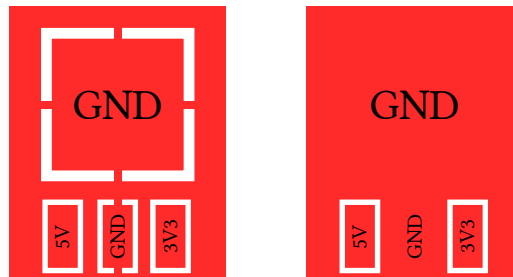


Figure 3.3: Thermal relief on left side vs no thermal relief on right side.

3.2.2 System-on-Chip Vias

The SoC comes in a quad flat no-leads (QFN) package, with a ground pad in center. To ensure both proper thermal conductivity and grounding, there must be an array of vias on the PCB where this pad is soldered. These vias can be problematic during soldering though.

During the soldering process, when sufficient heat is applied, the solder paste may exhibit capillary action. If a significant amount of solder paste is drawn into the vias, voids in the solder joint occurs. Even smaller voids will degrade the radio frequency performance of the chip. Ideally, these vias should be plugged (i.e. filled), but this makes fabrication expensive. Instead solder mask will cover the vias on the bottom side (tenting), reducing the solder paste wicking. While tenting on the top side would be more effective in reducing voids, this may cause issues with solder paste dispensing. [18] [19] Figure 3.4 illustrates the different solutions.

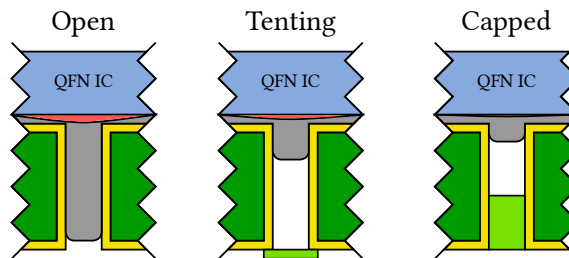


Figure 3.4: Different solutions for solving the via wicking issue.

3.3 Schematic Capture

With the component library set up, the schematic capture can begin. In terms of AD usage, there are essentially two commands that are necessary in the process of schematic capture, these are listed below. Other than this, careful reading of all datasheets is necessary, especially the application sections as these usually includes suggested circuitry and PCB layout.

- `p → p` places a component from library.
- `t → a → a` brings up the annotation tool.

3.3.1 Power Supply

The DC-DC regulator selected requires three external components for proper functioning, as per the data sheet. These are two filtering capacitors, one on the input side and the second on the regulated side. The manufacturer's recommendations are followed, a 22μF 25V capacitor on the input and a 22μF capacitor on the output. Only during testing and verification of the voltage regulation the optimal values for the capacitors may be found. The final external component is a trim resistor for configuring the output voltage. The resistor value is calculated according to the formula:

$$R_{TRIM}(k\Omega) = \frac{10}{\frac{V_{out}}{0.895} - 1} = 2.18k\Omega \quad (3.4)$$

The DC-DC regulator and the external components can be seen in Figure 3.5. Test points for GND, 12V and 5V can also be seen in the schematic. The battery is connected to the P1, PWR connector, and filtered through shunt capacitor C11 before entering the DC-DC regulator. At the regulated 5V output, a second shunt capacitor, C12 is located. The DC-DC regulator has been designed such that the trim resistor value aligns with standard values for 5V, which means that this resistor is available as a chip resistor with the recommended $\pm 0.5\%$ accuracy and a temperature coefficient of $\pm 100ppm/^{\circ}C$.

As it features On/Off control, a switch for power control will be connected to the relevant pin. This will be the main power switch of the control system, shutting down the supply to all circuits except an additional 12V connector.

The LDO performs the final voltage regulation step, converting the 5V from the DC-DC regulator down to 3.3V. According to the application notes in the data sheet it requires an

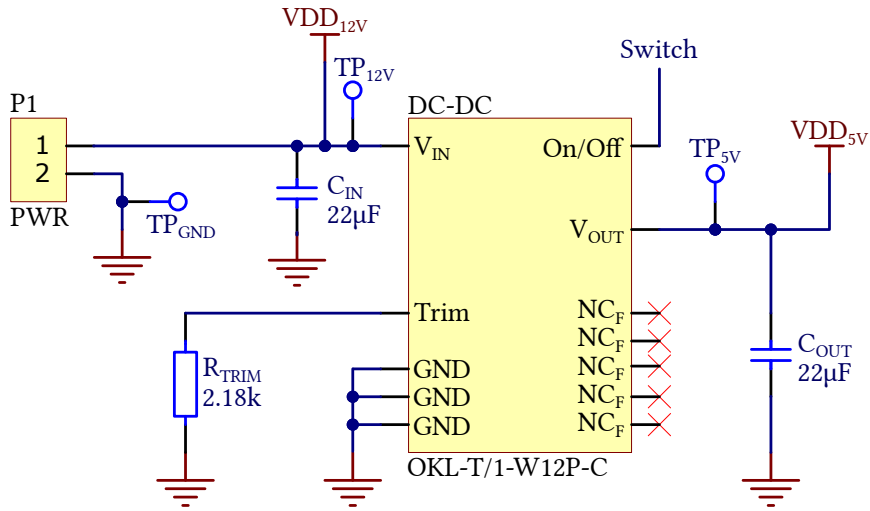


Figure 3.5: DC-DC regulator with surrounding circuitry.

input capacitor if battery power is to be used, and an output capacitor with low equivalent series resistance (ESR) over a large temperature span is critical for regulator stability. Figure 3.6 is taken from the data sheet, where the graph indicates acceptable range of ESR values for different output currents.

It is not unlikely that the control system will consume less than 200mA from the LDO regulator, and it will therefore be necessary to maintain an ESR within the narrowest region of the graph, $\approx 0.1\Omega$ to $\approx 0.8\Omega$. The capacitor selected is a tantalum capacitor with an ESR rating of 500m Ω . These capacitors features improved temperature characteristics compared with the cheaper aluminum electrolytic capacitors, at the cost of a couple of NOK.

This selection reflects a general philosophy used throughout the project: as only a very few of these PCBs are to be developed—as compared to, say several thousand—performance is weighted far more than price when selecting components.

The LDO regulator and the external components can be seen in Figure 3.7. A test point is connected to the output voltage, V_{DD} , and a power indicator LED is connected at the output.

The company that had developed the LDO, Texas Instruments, was contacted in order to get some feedback on the current set-up of the voltage regulator. It was there mentioned that it could be problematic feeding the LDO from a DC-DC regulator due to poor ripple rejection in LDOs in general.

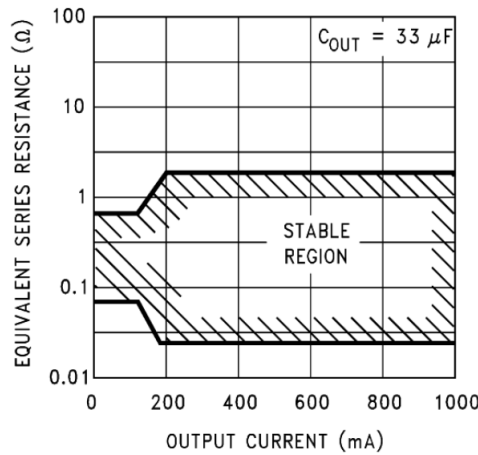


Figure 3.6: ESR requirements for LDO regulator output capacitor.

3.3.2 System-on-Chip

The SoC can be set up in several different configurations, as outlined in the reference circuitry part of the datasheet. There is the optional external low-frequency clock, as well as the option to set the SoC up using an internal DC-DC converter or an internal LDO regulator. In addition, some components from the reference circuitry are in a sense optional, and will be omitted for the control system.

While the 32MHz external crystal is required for the functioning of the radio in the SoC, the 32.768kHz suggested in the reference schematics is optional. The SoC actually has a corresponding crystal integrated, but this internal oscillator requires continuous calibration, and as a result the system will draw more current. The savings with the external LF oscillator is in the order of a few μA , which in the grand schemes of things in the SLAM system is negligible.¹ For this reason this circuitry is omitted in this control system design. This means that there are three fewer components to care about. More importantly this frees two pins, which also support AI.

Setting up the SoC using either an internal DC-DC converter or an internal LDO regulator is again a compromise between components and power savings. While these are both internal voltage regulators, they require some external components to operate. The DC-DC converter will require three extra components, a capacitor and two coils. By using the internal LDO regulator only a single capacitor is needed. Implementing efficient coils

¹<https://devzone.nordicsemi.com/f/nordic-q-a/19/what-s-the-benefit-of-having-an-external-32-khz-crystal>

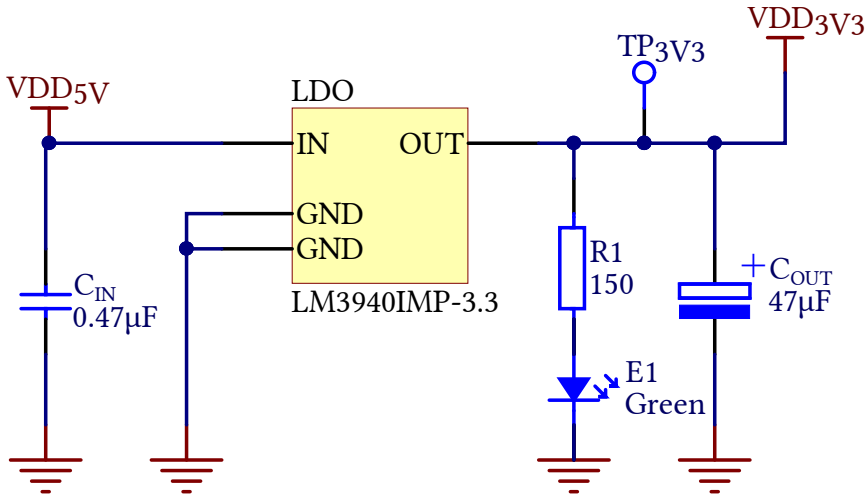


Figure 3.7: LDO regulator with surrounding circuitry.

in silicone and having them as part of ICs is difficult[20], and it is reasonable to believe that implementing the capacitors in silicone would waste a lot of space. As with the low-frequency clock, the internal LDO is opted for as it results in fewer components. The current draw from the SoC is unlikely to be problematic.

The antenna section of the circuitry is copied directly from the reference circuitry. The topic of antenna performance is complicated, and as per Nordic Semiconductor's recommendation their reference schematics should be used without modification. It is difficult to properly perform simulations to assess antenna performance in advance. Instead the suggested approach is to use a vector network analyzer and perform tuning by length of the PCB antenna. This process and more details regarding the design of the antenna is covered in a set of white papers. [21] [22]

As the SoC is to be programmed and debugged using the SWD interface, a two-row 1.24mm polarized semi-shrugged header is connected to the appropriate debug pins on SoC. By also exposing the reset-line via the header it will be possible for the external debugger to perform hardware resets of the SoC. In addition to the SWD interface, an optional pin, SWO, is connected to the programming header. This pin is part of the Cortex-M debug interface, and it enables real-time output from the CPU, meaning that instead of using a full peripheral module such as UART, this can be used. It also helps to ensure that the real time capabilities of the system is maintained when performing debugging as well.

Schematics with SoC and surrounding circuitry can be found as part of the complete electrical schematics in in Appendix B.

3.3.3 Analog-to-Digital Conversion

In order to monitor the battery voltage and obtain the IR-sensor measurements, the SoC's ADC must be used to convert the voltages to a digital representation. Internally, multiple channels are connected to a single ADC, which performs scan cycles converting the voltages on the different channels. These channels are then connected to a set of pins on the SoC, and only these can be used for ADC operations. The conversion process involves a sample-and-hold circuit, where successive approximation is used to estimate the voltage. The set-up is illustrated in Figure 3.8

During the PCB layout process, connections were re-arranged with regard to ease of placing tracks, and not on the AI support of the SoC's pins. As a consequence battery monitoring is broken. It has been corrected in the second revision, as described in Section 3.5.

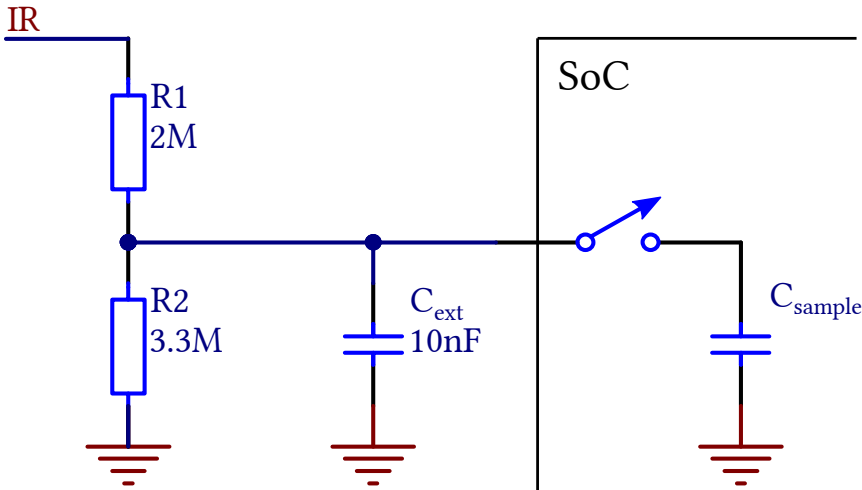


Figure 3.8: The voltage dividers and internal structure of the SoC's ADC sample-and-hold circuit.

The ADC will be configured in single-ended mode using VDD as reference. This results in a supported input voltage range of 0 – 3.3V. As an example, the IR-sensor maximum output voltage is 5.03V. A voltage divider with a capacitor on the input is set up in order to match the output voltage of the IR-sensor with the input range of the ADC. The resistor values chosen are 2M Ω and 3.3M Ω .

Selecting large resistors reduce the leakage current, with the side effect of causing a significant voltage drop in case the load draws any current. This latter issue is mitigated by the capacitor, which acts as an accumulator, supplying the sample-and-hold circuit internally on the SoC. This setup can be seen in Figure 3.8. By using the 10nF capacitor, the minimum ADC acquisition time of 3 μ s can be used.² With five ADC channels active (battery, four IR-sensors), this will allow for the following maximum sample rate:

$$f_s < \frac{1}{5(t_{acq} + t_{conv})} = \frac{1}{5(3\text{ms} + 2\text{ms})} = 40\text{kHz} \quad (3.5)$$

3.3.4 Bus Devices

There will in total be three separate bus systems connected to the SoC. One dedicated I2C bus for the external LIDAR device. A shared I2C bus for the onboard IMU and magnetometer. An SPI bus semi-shared between the microSD slot and the display.

The LIDAR communicates using transistor-transistor logic (TTL) levels, that is 5V, while the SoC only supports communication over CMOS logic levels of 3.3V. To enable communication between the two devices, a bi-directional level shifter will be set up. Based on application notes on bi-directional level shifters from NXP Semiconductors and Philips Semiconductors the circuit in Figure 3.9a is set up. [23] [24]

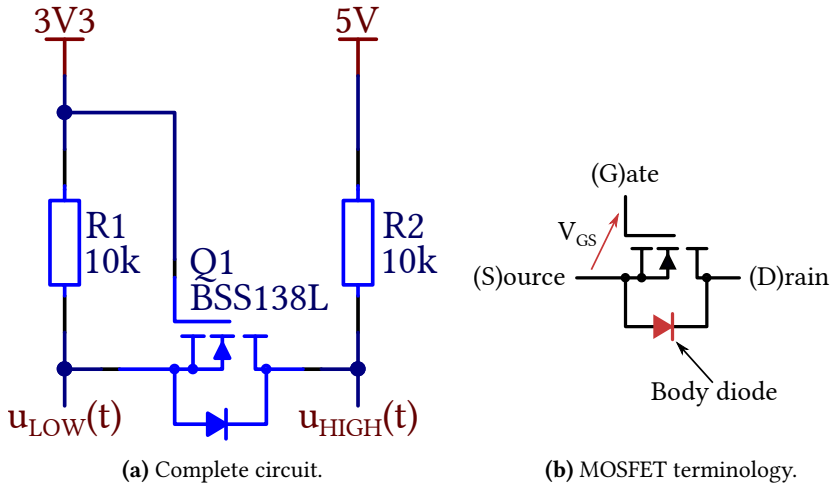


Figure 3.9: Bi-directional level shifter circuit.

²<https://devzone.nordicsemi.com/b/blog/posts/measuring-lithium-battery-voltage-with-nrf52>

As described in the application notes, there are three different scenarios that have to be accounted for in order to verify that this will function as a bi-directional level shifter:

1. Neither device pulls down the bus line: the MOSFET stays closed, and the two pull-up resistors causes HIGH level on both sides.
2. 3V pulls down: gate-source voltage reaches the threshold voltage which opens the MOSFET, causing both sides to be pulled LOW.
3. 5V pulls down: body diode of MOSFET starts conducting, the voltage on the low level side starts drooping, until it opens the MOSFET which now causes both sides to go LOW.

The electrical characteristics required for the power MOSFET is detailed in the application notes, where the most important factors are to ensure that the breakdown voltage is rated with a minimum value of 0.1V and a maximum of 2V, and the switching delay $t_{on}t_{off} \leq 50\text{ns}$.

3.4 Printed Circuit Board

This section covers the process of realizing the previously defined schematics into a four-layer PCB design. It will involve components positioning, and the actual copper tracks and vias creating the connections between the components. It will also describe some of the more subtle aspects in an attempt to adhere to industry best practices and design for manufacturability.

While presented in a linear fashion here, the actual design process is more of an iterative process. Before positioning a component, one has to think carefully ahead to ensure that the choice will not result in conflicts later on. The design of a PCB layout is not a hard science with a definitively single correct solution. During the layout for the control system, David L. Jones's introduction to PCB design has served as the primary source for guidelines and best practices. [25] More detailed guidelines about radio frequency design can be found in Texas Instrument's application notes. [26] [27]

3.4.1 Dimensions and Layers

Board dimensions may not exceed 100mm in either direction, as this will significantly increase the production cost. The AM has a width of 101.6mm, so selecting a width of

100mm seems natural for the new control system. The height of the AM card is 53.34mm, so an initial height of presumably ample 60mm is used for the control system.

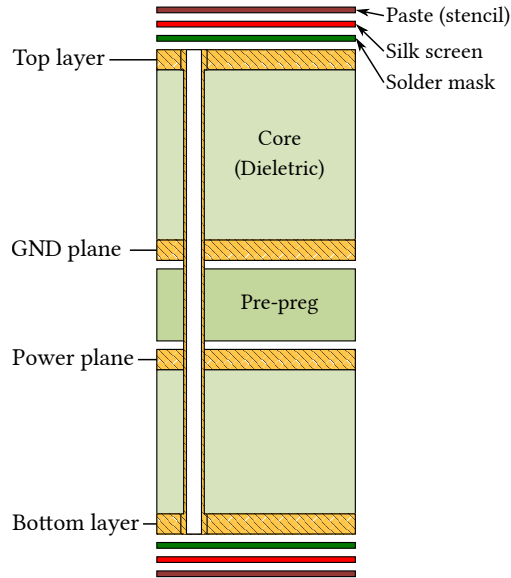


Figure 3.10: PCB layer stackup legend.

A 4-layer board stackup is opted for, primarily due to two main concerns: ease of track routing and good grounding and shielding with regard to EMC noise. The setup can be seen in Figure 3.10. While it is stated that a 2-layer board stackup should be able to achieve comparable performance with regard to EMI, considering the author's lack of experience in the field the increase in price was deemed worth it in order to give some leeway in the design details. [26] [27] This will also create for a PCB that is easier to extend and modify in the future.

An attempt is made to position components into functional groups. For the control system the main groups are the voltage regulation circuitry, SoC with antenna and crystal oscillators, and connectors for sensors and actuators. Figure 3.11 shows a sketch illustrating this planned layout of functional groups. The voltage regulator group and SoC group has been separated as much as possible. An attempt has been made to isolate the magnetometer at the bottom right, and the IMU has been position in center in order to limit offset calculations.

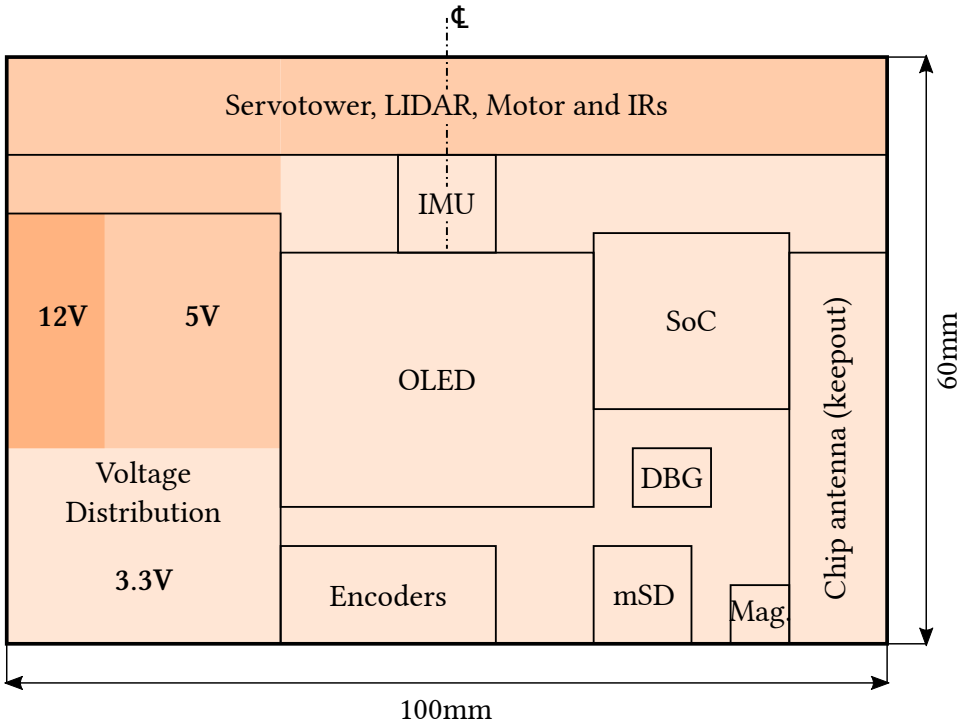


Figure 3.11: Planned board positioning of sections and components.

3.4.2 Initial Configuration

Based on the manufacturer's capabilities³, a rule-set is configured in AD $d \rightarrow r$. This way AD can perform a verification of the design, automatically ensuring that all tolerances are met, e.g. track widths and via diameters.

Next the board layer stack-up is configured from within the *Layer Stack Manager*, $d \rightarrow k$. The layer configuration in AD can be seen in Table 3.1. The dedicated layer type of *Power* has not been used, as this locks the fabrication output to be inverted for the power planes. Many of the low-cost fabricators are unable to handle inverted designs, and for this reason the default signal-layer type is used for the power planes as well.

The board dimensions are set up from within board planning mode $v \rightarrow 1$. The board shape is redefined $d \rightarrow r$. A coarse grid $g \rightarrow 2.5\text{mm}$ is set up, and the with the origin in the left lower corner, the board outline is drawn using $d \rightarrow r$ such that it measures

³JLCPCB is used as manufacturer for the control system. Their capabilities list is available at <https://jlcpcb.com/capabilities/Capabilities>

Table 3.1: Layer stack configuration in AD.

Layer Number	Layer Name	Type	Orientation
1	Top Overlay	Overlay	Top
	Top Solder	Solder Mask/Coverlay	
	Component Side	Signal	
	Dielectric 1	Dielectric	
2	Ground	Signal	Not Allowed
	Dielectric 2	Dielectric	
3	Power	Signal	Not Allowed
	Dielectric 4	Dielectric	
4	Solder Side	Signal	Bottom
	Bottom Solder	Solder Mask/Coverlay	
	Bottom Overlay	Overlay	

100mm in the x dimension and 60mm in the y dimension. The board outline is drawn on the keep-out layer, which will be the layer specifying the board dimensions in the fabrication data. This is generated from the previously defined board shape using `d → s → p`. The width is set to match the manufacturer's requirement, and the keep-out layer is selected as target layer.

Finalizing the initial configuration, a grid used for positioning components is set up. It will be measured in mils, which is the unit for one thousandth of an inch. Imperial units are –unfortunately– still the most common system used in PCB design. The component positioning strategy used, is to first select the imperial grid `q`, and then start with the most coarse grid `g → 100mil` and cycle downwards until component positioning is feasible. Consistently aligning components with a coarse grid aids in the development of a systematic and tidy layout. For this strategy to work, a fixed origin is positioned at the PCB's lower left corner.

This covers the initial configuration. The result is an empty *canvas* similar to what is shown in Figure 3.12, with the layer stack-up configured as in Table 3.1. All manufacturer's rules are also implemented at this point.

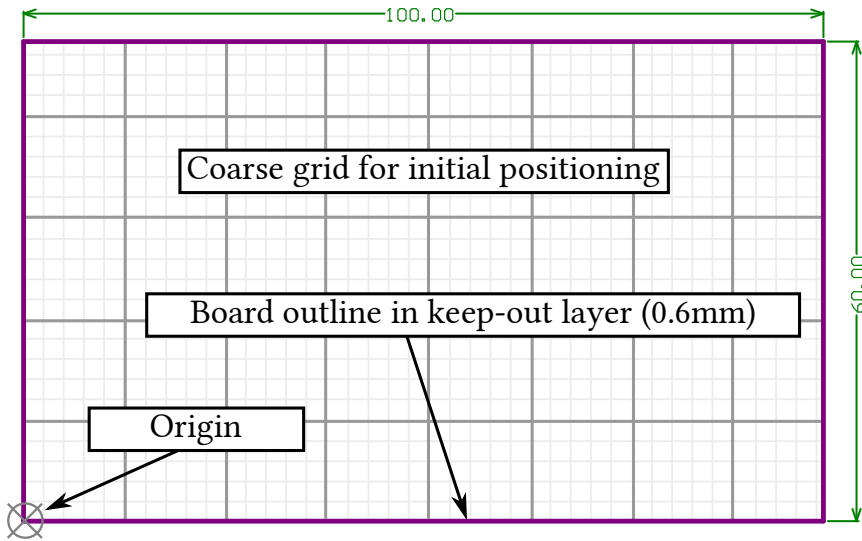


Figure 3.12: Board set up and ready for layout design.

3.4.3 Design Process

With the initial configurations set up, the schematics design with all its components, nets and labels are imported `d → i`. All components are then positioned based on the sketch in Figure 3.11. The final components positioning is shown in Figure 3.13. Deviations in component positions are mainly due to priority given to ease of routing tracks. This process of components positioning is part of an iterative process, mainly due to track routing challenges, but for brevity it is here described as a linear process.

While it was stated that component positioning was based on logical grouping, there are other aspects to consider as well. The voltage regulation circuitry with the DC-DC converter (U4) has been placed on the left side of the board, while the SoC (U1) with antenna has been placed on the right side. DC-DC converters are known to be a source of electromagnetic interference, and the SoC with antenna is likely to be the part of the circuit susceptible to this noise. This problem is then mitigated by positioning these two systems as far apart as feasible.

Another component susceptible to noise in general is the magnetometer. As can be seen in Figure 3.16 and Figure 3.17, all four layers have been cleared within a given distance of the magnetometer. The noise emission from the display may be problematic, in which case moving it to the lower right into the antenna section could be a viable solution.

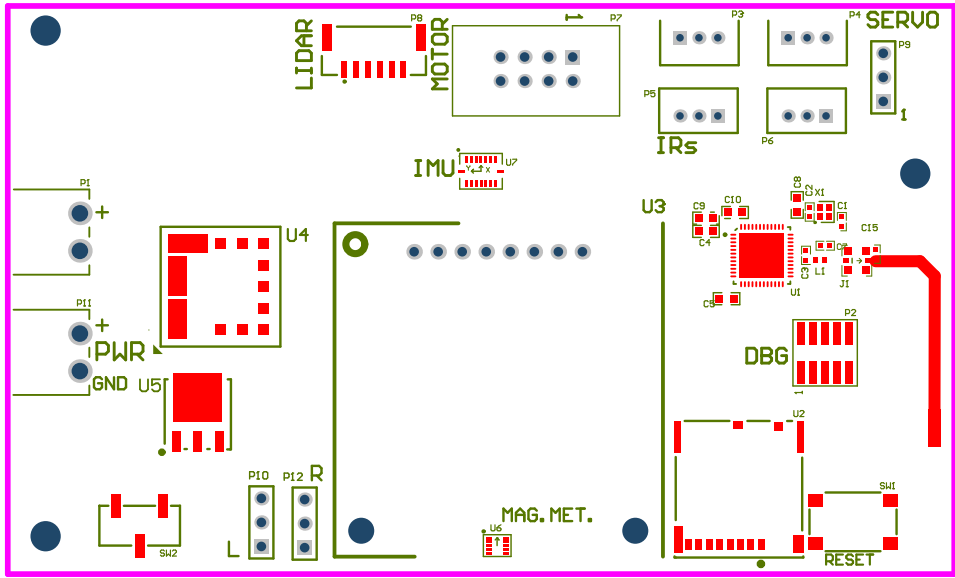


Figure 3.13: PCB layout with primary components placed.

With all primary components positioned, the passives such as resistors and capacitors are positioned. The majority of passives are decoupling capacitors and are thus placed in the vicinity of the integrated circuits. Primarily components no less than 0603 have been selected to ease the assembly process. The SoC circuitry is copied directly from the reference design without modification, and it includes components for antenna impedance matching. The antenna is extended a couple of millimeters in length to allow for antenna tuning later on. In Figure 3.14, the layout with passives positioned can be seen.

Tracks are laid out on the top and bottom layer to create the electrical connections between components. Tracks are never routed along the the internal layers, they strictly pass through, or connect, using vias. This is especially important for the GND layer, as it is undesirable to have this layer split due to noise considerations. The final top layer with tracks can be seen in Figure 3.15. Via stitching is added to minimize impedance caused by long ground return loops.

All bends part of a track are 45° . While the electrical performance aspect to this is disputed, it is broadly accepted as making for a more tidy design. [25] [28] Track widths are also variable throughout the whole design. The primary factor when selecting track width is the current carrying capabilities. A clear sign of this consideration can be seen by comparing the track widths in the voltage regulation section with those surrounding the

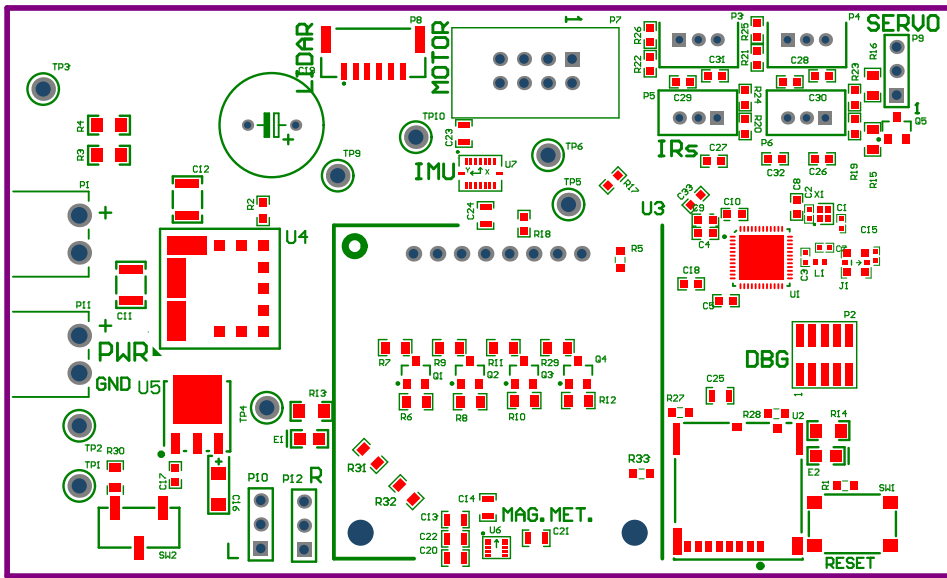


Figure 3.14: PCB layout with primary and passive components.

SoC. Due to the fine pitch of the SoC's pads, some of the tracks actually narrows/widens midway in this area.

All tracks and vias connected to passives are kept as symmetric as feasible. This way the design will be less prone to the phenomena of tombstoning, where thermal asymmetry causes one side of the chip to raise during reflow soldering. The pad with the least copper heats faster, and as a result reflowing will occur earlier at this pad, raising the chip like a tombstone.

The internal power plane is divided into a 3.3V section and a 5V section. This power plane segmentation can be seen in Figure 3.16. Internally on the PCB all components are 3.3V-compatible, but many the sensors and actuators are 5V components, as that matches the voltage of the ATmega systems. The narrow 5V peninsula is there to supply the level shifters discussed in Section 3.3.

The final step of the design is to perform design rule checks in AD `t → d`. In this process, AD will ensure that all aspects of the PCB design is fully supported by the manufacturer's capabilities. It uses the rules that were set up initially to perform this test, so it is important that the rule-set has been set up correctly according to the manufacturer's specifications.

The final PCB layout for the top layer can be seen in Figure 3.17. A ground fill (copper pour) is added to the top layer. This *may* improve noise and performance by creating

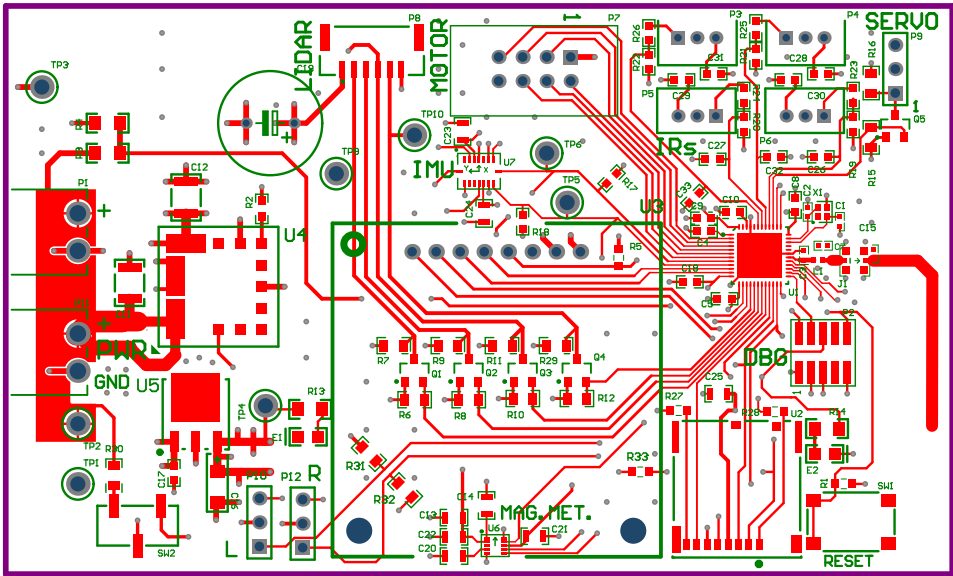


Figure 3.15: PCB layout with components and tracks.

shorter paths to ground, but reading on different forums this seems to be a disputed claim. As the Nordic Semiconductor's reference design also features a top ground layer, mimicking this set-up as closely as possible is desirable. The top ground fill does impact the antenna performance.

As with the magnetometer, a keep-out section has been introduced on all layers for the antenna section. This way antenna signals are not attenuated by copper on any of the layers, which should increase receive sensitivity and transmit power.

A 3D-model and fabrication output can be seen in Appendix B.

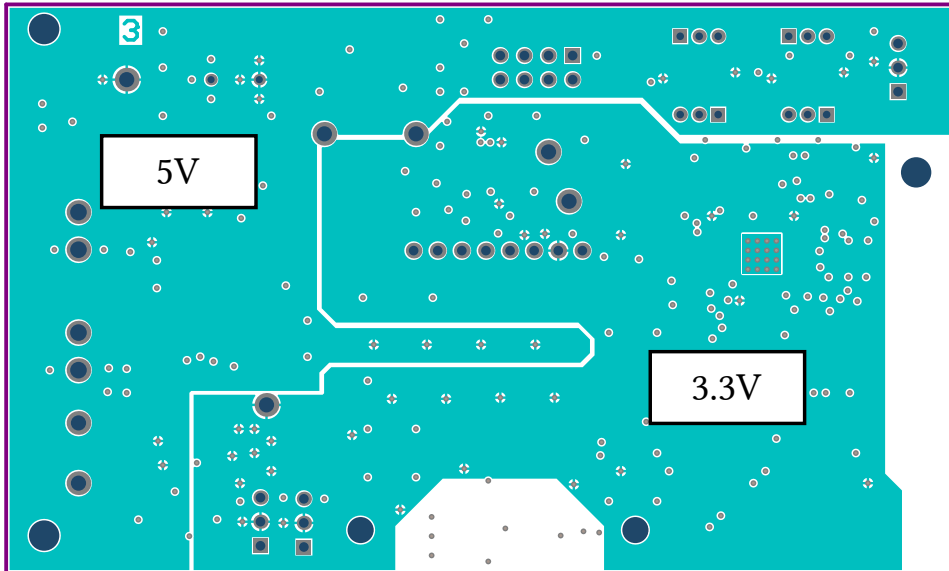


Figure 3.16: Internal power layer has been split into one 3.3V segment, and one 5V segment.

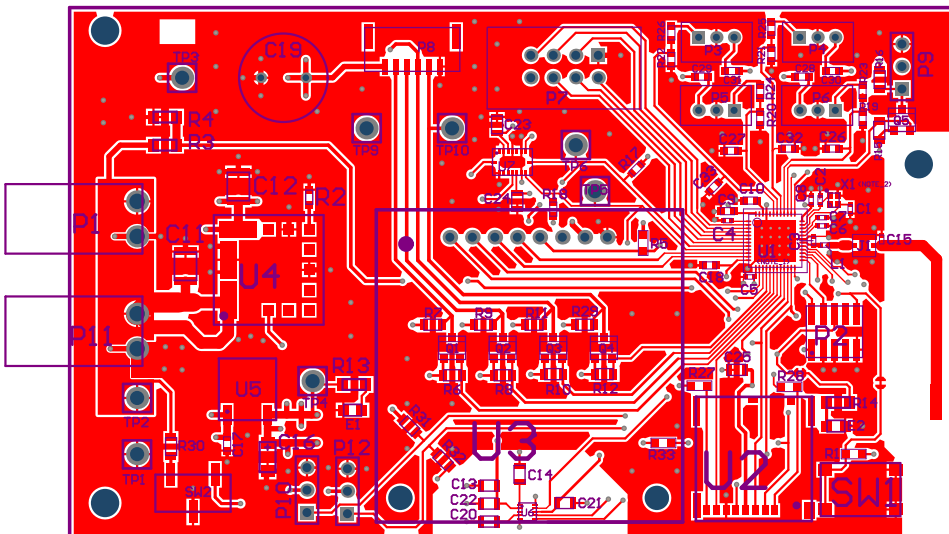


Figure 3.17: Final PCB layout with top ground plane and components mechanical outline.

3.5 Second Revision

A second revision of the schematics and layout has been made. This second revision includes a set of corrections and improvements based on experiences made with the first revision. This section also includes some suggestions that has not been included, but should be considered before manufacturing of a new revision.

3.5.1 Corrections and Improvements

The following corrections have been made in the second revision:

- **Reset-button:** The reset-button footprint was incorrect. The result is a short to ground when the button is soldered on, resulting in constantly repeating resets.
- **IR decoupling capacitors:** The decoupling capacitors ended up in series with the supply voltage, not in parallel as they should have been. They must be shorted on the first revision. On the second revision this has been fixed.
- **Battery monitor:** During the PCB layout design, pin assignment was changed to facilitate tracks, but not with regard to actual pin functionality. An ADC enabled pin must be used to measure the battery voltage, but this was not maintained during the rearrangement. This has been fixed in the second edition.

The following improvements have been made in the second revision:

- **Sensor-tower servo:** The servo header has been moved away from the antenna section, and has been connected to the 680 μ F capacitor that was previously dedicated to the LIDAR. The large capacitance of this capacitor is mainly due to limitations in USB power supplies used on Arduinos, usually rated for 500mA. As such, this sharing of the capacitor should be acceptable.
- **QFN footprint:** based on feedback from the staff at the electronics and prototype-laboratory at NTNU, the center vias in the QFN footprint has been removed. Also bottom tenting of vias has been added, which had been forgotten in the first revision.
- **Via tenting:** several top via tents have been removed, such that they can easily be probed with a multimeter. This is especially useful when for example checking ICs for soldering short circuits.

3.5.2 Ease of Manufacturing

The author has inquired Nordic Semiconductor about positioning and dimensioning of decoupling capacitors and inductor part of the antenna impedance balancing components. According to them the strict requirements given in the reference layout guidelines are there to ensure optimal range, and to ensure that there will be no issues during conformance testing of the product.

Since it is not problematic with a slight loss of range, and conformance testing for licensing is not relevant for the control system, more leeway is given for chip dimensions and positioning. In Nordic Semiconductor's response it was suggested that replacing all 0402 chip components with 0603 or bigger components is unlikely to cause any issues. The spacing may also be increased in order to facilitate easier soldering. Implementing these changes are recommended if an assembly line is not setup at the manufacturer for the development of the next revision.

3.5.3 Inrush Current

While it has not been a problem during testing of the first revision, inrush current could become an issue with the amount of capacitance in the system. Inrush current is a high initial current that occurs during power-on as the capacitors charge up. The DC-DC converter includes a soft-start mechanism that can automatically reduce the slew rate in case of large inrush current. Too large of an inrush current can potentially damage PCB traces and connectors. If this built-in soft-start should be insufficient in handling the inrush current on the final system, a load-switch should be implemented. An alternative is to simply put a resistor in series with the capacitor, but this will reduce system efficiency. [29] [30]

Chapter 4

Hardware

In this chapter, the process of generating the manufacturing files for the PCB layout is described. The production of the PCB itself is out-sourced to an external manufacturer, but the soldering and assembly of components onto the PCB are performed as part of the thesis work. Parts of this process, involving precision- and reflow-soldering, is detailed in this chapter. For brevity, only the most complex solder job is described in detail. The final section describes the electrical testing and verification of the power supply.

The whole development process can be seen in Figure 4.1, where the stages relevant for this chapter have been highlighted.

4.1 Equipment

Table 4.1 below lists tools and equipment for precision soldering, crimping of connectors, and equipment for electrical testing and verification.

4.2 PCB Fabrication

This section primarily details the process of exporting the PCB design from AD to a format parseable by manufacturers. While the design rule check performed in AD should suffice, the exported PCB design is submitted for further design-for-manufacturing (DFM) analysis.

Table 4.1: Equipment used during soldering, assembly and verification.

Equipment	Model/Make	Function
Basics		
Precision pliers	ATG 27/08	Manual pick-and-place
Microscope	MS-OPT-2280	Tiny components
Cleaning agent	Isopropyl Alcohol	Cleaning of pads
Cotton swabs	Lint free	Cleaning of pads
Flux	Kingbo RMA-218	Improve wetting properties
Solder wire	60/40 Rosin core 0.5mm	Precision hand soldering
Solder paste	Lodestar L309050	Reflow soldering
Soldering Iron		
Soldering station	Metcal MX-5200 (MX-PS5200)	Temperature control
Soldering iron	MX-H2-UF	Precision hand soldering
Solder iron tip	Metcal STTC Chisel and Bevel	Precision hand soldering
Reflowing Soldering		
Hot-air rework station	Metcal HCT-900-21	Component level reflow
Reflow oven	SEF 548.04 G	Board level reflow
Heating jig	Metcal PCT-100	Pre-heating of PCB
Testing & Verification		
Power supply	Oltronix B202	Stable power supply
Multimeter	Fluke 87V	Electrical verification
Oscilloscope	Agilent MSO-X 2024A	Time domain signal analysis
Connectors		
Unofficial crimper	N/A	Crimp JST connectors
Official JST crimper	YRS-1590	Crimp JST connectors

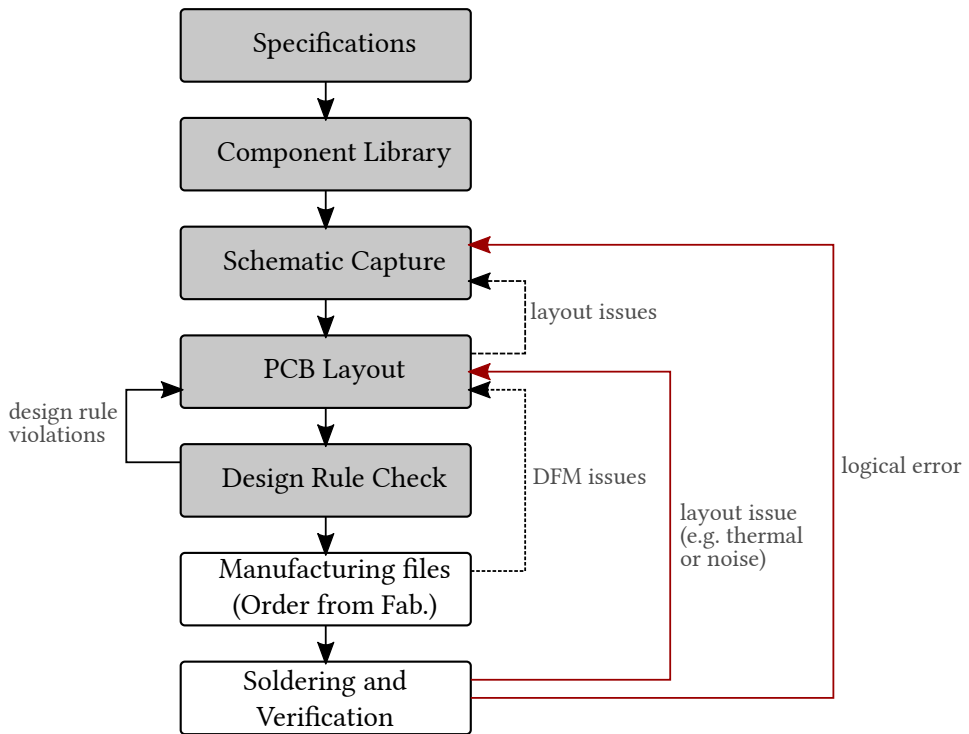


Figure 4.1: Flowchart illustrating the complete PCB development cycle.

This DFM analysis is automated and may catch issues caused by misunderstanding rules or incomplete rule sets given by the manufacturer. The section ends with a short visual inspection of the received PCB.

First, the fabrication outputs are generated. These are translations of the PCB design into a common format that the manufacturer can parse and use during fabrication. The most common format used for this purpose is the vector Gerber format. There will be a Gerber file for each of the four layers in the PCB. Drill holes are exported to the IPC-NC-349 (NC Drill) format. To export the design to these formats in AD enter `f → f` and select either *Gerber Files* or *NC Drill Files*.

The layers that are exported (plotted) are in relation to Figure 3.10: GTO (Top Overlay) Silkscreen; GTS (Top Solder) Solder mask; GTL (Component Side) Top layer; G1 (GND) GND plane; G2 (Power) Power plane; GBL (Solder side) Bottom layer; GBS (Bottom solder) Bottom solder mask; GBO (Bottom Overlay) Bottom silkscreen, GKO (Keep-Out layer) Board outline. A simple text file describing how the Gerber files relate to the board layer stack-up is included in the fabrication files.

If the internal layers are set to power planes, something which is the default for the generated board stack-up in AD, these planes will be exported as negatives in the Gerber format. Not all manufacturers can parse and use inverted Gerber files, and there is no option in AD to invert back. If non-negatives are needed, the best option is to change the internal layers from Power layer to Signal layer, and redraw them.

The exported Gerber files are submitted to FreeDFM (<http://my4pcb.com>) which is an automated DFM service that can verify the manufacturability of the design. A report is generated indicating two categories of errors: "Potential Show Stoppers" and "Problems Automatically Fixed." The former is reviewed thoroughly before sending off to production, while the latter is likely to be handled by the manufacturer automatically.¹ An example of an error during the first run in this project can be seen below. This error should have been caught in the DFM check of AD, but the rule configurations in AD were incorrectly set up.

Insufficient Annular Ring: Requirements: a minimum of .005" annular ring for vias, a minimum of .007" for component holes. Resolution: all layout packages provide this as a DFM check.

Finally, the order is submitted to JCLPCB² with the default manufacturing configuration. It is worth noting that the default green colored solder-mask can be beneficial for visually distinguishing the traces and features on the circuit board.

The final PCB can be seen in Figure 4.2. A visual inspection highlights some interesting features of the manufacturing capabilities of JCLPCB:

1. There is solder mask between the fine pitch pads for the SoC. These solder mask slivers are beyond their listed capabilities, which is impressive.
2. The silk layer annotations for the passive components are smaller than the minimum recommendations given by the manufacturer, yet still they are for the most part legible.
3. The layer stack indicator is slightly broken, but this is due to a design error introduced in the process of changing the inner power layers to AD signal layers. This error is not visible in the figure.

¹The manufacturers will frequently make small design changes to improve manufacturability. In other cases, they may contact the customer, something which will incur a delay to the production.

²JCLPCB (Shenzhen JIALICHUANG Electronic Technology Development Co., Ltd.), available at <https://jlcpcb.com>

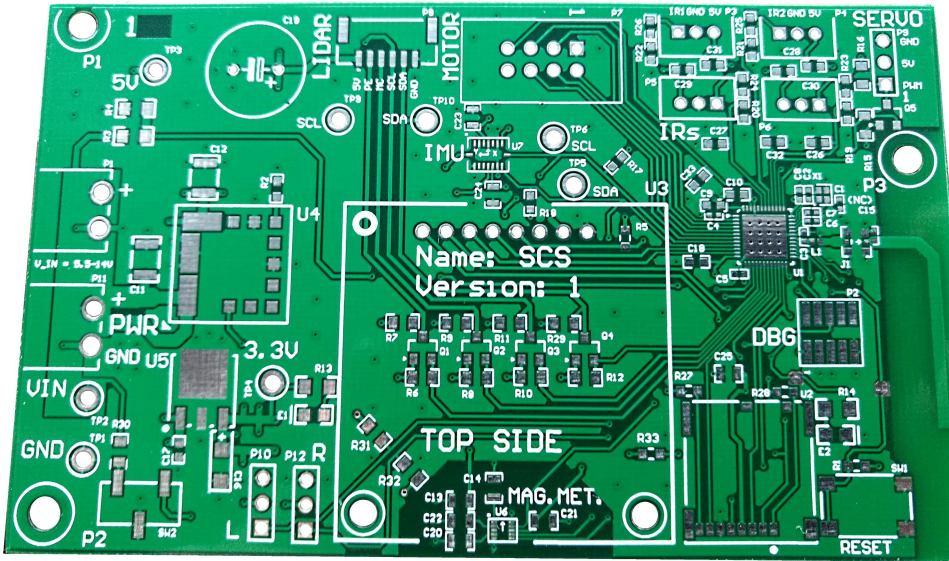


Figure 4.2: Visual inspection of the received PCB.

4.3 System-on-Chip

The most challenging part of the soldering process is to get the SoC soldered correctly. The fine pitch (0.4mm) of the QFN package makes it prone to both solder-bridges and pad misalignment. While a technique for fixing exterior bridges is presented here, there is no guaranteed fix for any hidden bridge below the chip itself. Making matters worse is the surrounding array of passives in miniature 0402 packages. This section presents a procedure using techniques that have proven to work adequately.

To solder the SoC a combination of hot air flow and usage of traditional solder iron is used. Initially, both the pads on the PCB and the SoC are cleaned using lint-free cotton swabs soaked in isopropyl alcohol. A thin line of flux is applied along the pads on the PCB, and a small amount onto the ground pad. A solder paste dispenser is used to dispense a thin line of solder paste on top of the flux. Applying too much solder paste can be problematic. Along the pads, it may cause solder bridges, while for the ground pad it may end up lifting the SoC such that its pads and the PCB's pads end up too far apart. The picture in Figure 4.3a shows a suggested amount of flux and solder paste.

With arms resting firmly on the tabletop, the SoC is positioned to the best of one's ability using a set of precision pliers and a microscope. The initial placement is not critical, and readjustments of a couple of millimeters is not itself problematic. The SoC is moved until

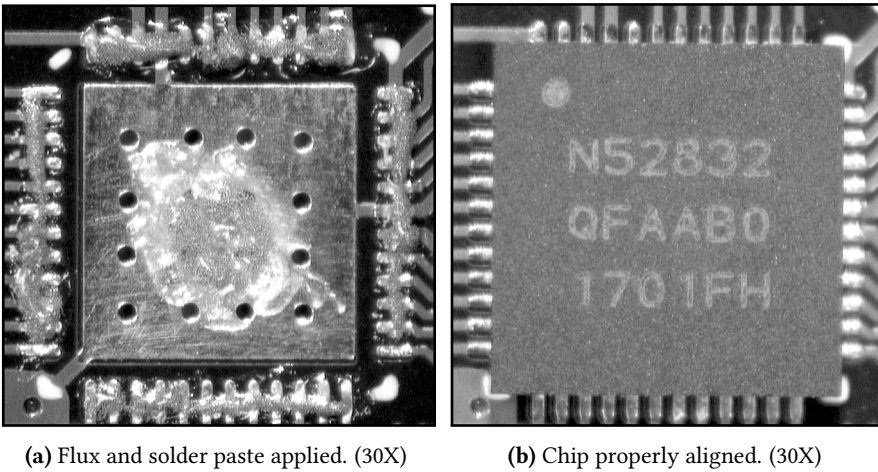


Figure 4.3: Solder paste and flux application, and the final SoC alignment.

it is aligned with the silkscreen. While the SoC will align with pads during the reflow process, if the initial alignment is poor, the pads may be completely shifted by a row of pads. There is no easy way to fix such misalignment, apart from removing the SoC and repeating this whole process. Proper alignment is shown in Figure 4.3b.

A pre-heating station and a hot-air rework station is used to perform the reflow soldering. The pre-heating station is given a minute to pre-heat the PCB. Then more heat is gradually applied using a hot-air gun until the solder paste reflows. Applying too much heat, too quickly, may damage the SoC internally. The solder joint at the ground pad below is not of concern, as the whole board will be sent through a reflow oven at a later stage, where proper reflow is assured.

Correct alignment and solder joints are verified using a microscope. In case of misalignment, it is possible to apply generous amounts of flux around the SoC, reflow as in the previous step, and attempt to carefully push it into position. This realignment process is difficult and may result in pads flaking off the PCB. Solder-bridges, or lack of visible solder, are fixed using a soldering iron with a fine beveled tip. For solder bridges, flux is applied, and a clean soldering iron is swept over. This is repeated until the bridge is gone. For pads lacking visible solder, apply flux, and apply a minute amount of solder to a clean solder iron, and sweep the soldering iron over the row. When this is done, flux residuals may be removed using isopropyl alcohol. A solder bridge can be seen in Figure 4.4a, while good solder joints can be seen in Figure 4.4b.

Before soldering the chip components surrounding the SoC, a final verification is per-

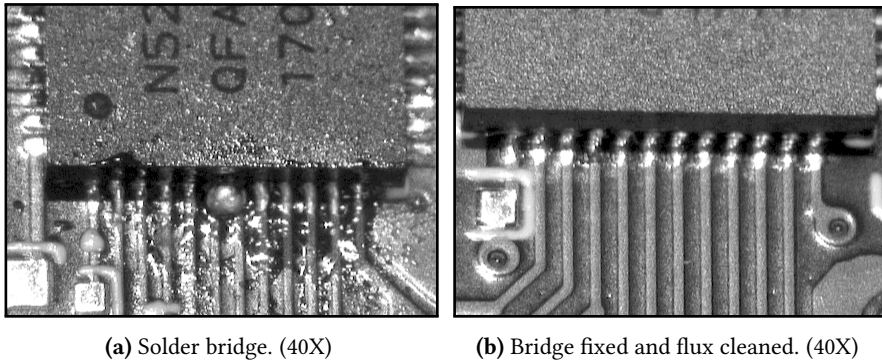


Figure 4.4: Difference between good and bad QFN soldering joints.

formed to verify that there are no shorts between any of the pins. The design is opened in AD, where pairs of adjacent pins and attached tracks are highlighted. Then a multimeter is used on the corresponding pads or pins to ensure that there is no continuity between adjacent pairs of pins. There is one exception, and that is between pin 30 and 31. There is a balun inside the SoC that has DC short to ground, so 2.8Ω is expected between this pair. When satisfied with the result, flux residues are cleaned off using isopropyl alcohol.

The passives surrounding the SoC are soldered by dispensing solder paste on all the pads. Following this the chips are positioned using a microscope and a set of very fine tweezers. The reflow temperature graph for the solder paste is found in its datasheet. The temperature profile is set up in the reflow oven, and the reflow process is performed. More components can be soldered in this process. The SoC can sustain at least six reflow processes, while the Murata DC-DC module is only capable of sustaining between two and three such processes.³ The performance of chip capacitors is degraded for every pass in the reflow oven, so it is advisable to keep the number of passes to a minimum.

For the second revision, the use of stencil should be considered. The stencil can be especially useful for the IMU which does not have any of its pads running along the side such as is the case with the SoC. Stencils have been designed and ordered for the control system. While these stencils are designed for the first revision, it should still be possible to use it for the most critical components such as the IMU and SoC, as these have not changed position. The stencil can be seen in Figure 4.5, where the SoC area is on the right side.

³This information was obtained by contacting both manufacturers directly.

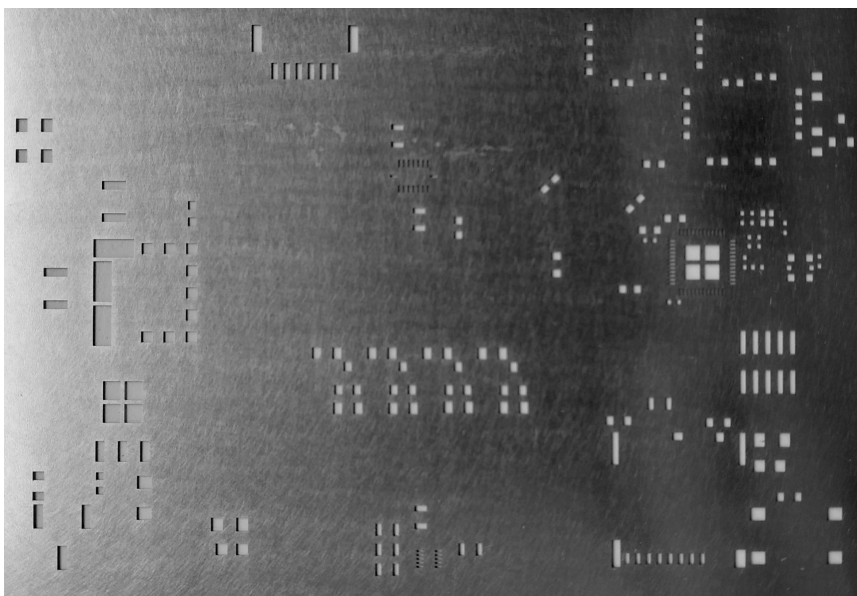


Figure 4.5: Stencil that aids in dispensing solder paste correctly.

4.4 Remaining Components

With the SoC soldered successfully, there are two more components that may prove challenging to solder. That is the magnetometer and the IMU. The magnetometer comes in a package similar to the SoC, so the same procedure detailed in Section 4.3 should be applicable. The IMU is more difficult, as the QFN pads do not extend to the outside of the package at all. For this package, the best option is likely to set up and use the stencil.

For the remaining components all of the following methods should be applicable:

- Solder using a soldering iron: Difficult for smaller chip packages, and more prone to oblique orientation of components; allows for a very delimited area of heating.
- Hot air gun: Easy to solder; not possible to follow the recommended temperature curve for the solder paste; allows for a somewhat delimited area of heating.
- Reflow oven: Can solder the whole PCB in one pass; can be configured to follow the temperature curve for the solder paste; no delimitation on area of heating.

Apart from the soldering process, the different JST connectors also have to be crimped. All JST connectors consist of three components: the contact itself, that is crimped onto the wire; the housing in which the wire with the crimped contact is locked into; a

locking receptacle header mounted onto the PCB. As an example Figure 4.6 illustrates all components for the VH-series JST connector.

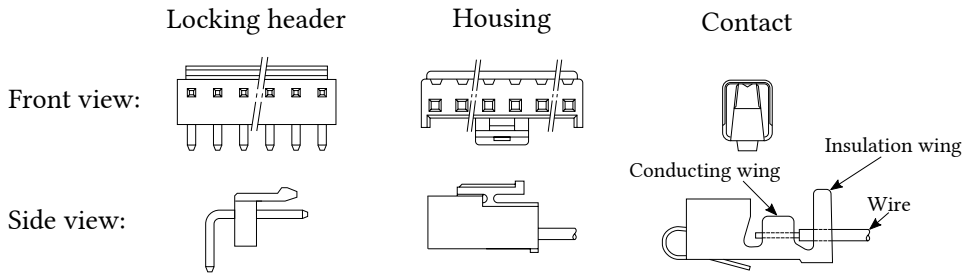


Figure 4.6: All components of a JST connector, using the VH-series as an example.

For the JST connectors, there are official crimping tools available, but they are costly. In the following, a method using unofficial crimping tools will be described. The VH series connector is used in the example, but the same approach applies to all series. The smaller series are more difficult to get right, and some of the generic crimping tools may be too big for the these.

A cable with cross section of $0.326\text{--}1.31\text{mm}^2$ (AWG 22–16) should be selected, and about 5mm of insulation stripped. Referring to the contact in Figure 4.6, the following should now be ensured:

1. insulation crimping wings are only in contact with wire insulation
2. conductor crimping wings are only in contact with conducting wire
3. no wire enters the locking mechanism section; this affects locking force

Next, a pair of flat nose pliers is used to slightly bend the two set of wings inwards. Then the insulation wings are inserted into the crimping tool, and the crimping is performed gradually while observing that the insulation is not penetrated. Then the conductor wings are crimped onto the wire. Afterward, the nose pliers are used to ensure that the wings are properly crimped.

The final step is optional, and somewhat controversial for Japanese *Solderless* Terminals—solder the conduction wing. Generous amounts of flux are applied to the conducting area. Then some solder is put on the soldering iron, before moving it in contact with the fluxed area for no more than a second. The flux residues are properly cleaned off to avoid corrosion. The cable with the contact is now ready to be pushed into the housing, where it will click lock when fully inserted.

4.5 Power Supply Verification

In this section the verification and testing of the power supply is described.

Correctly measuring the ripple voltage has been performed using a set-up loosely based on the one described in Analog Devices' AN-1144. [31] The probing method used can be seen in Figure 4.7, where all probing is done directly on the output capacitors of the voltage regulators. A ground clip is used instead of the more common ground cables, to reduce parasitic effects as much as possible. This detail is further described in the application note.

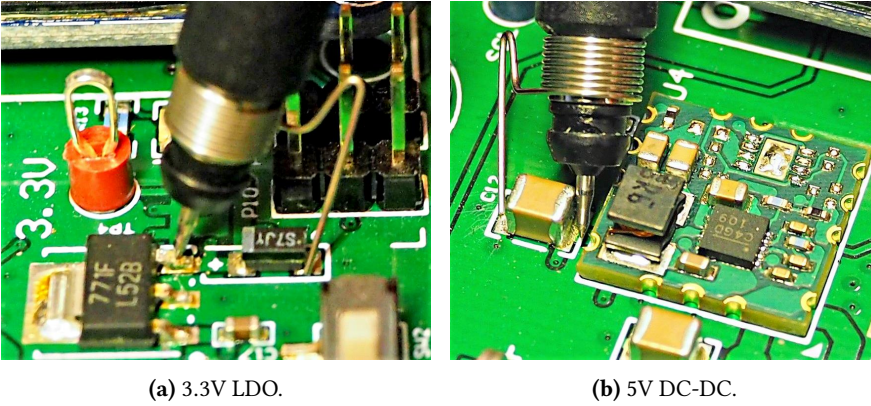


Figure 4.7: Probing method for voltage regulators.

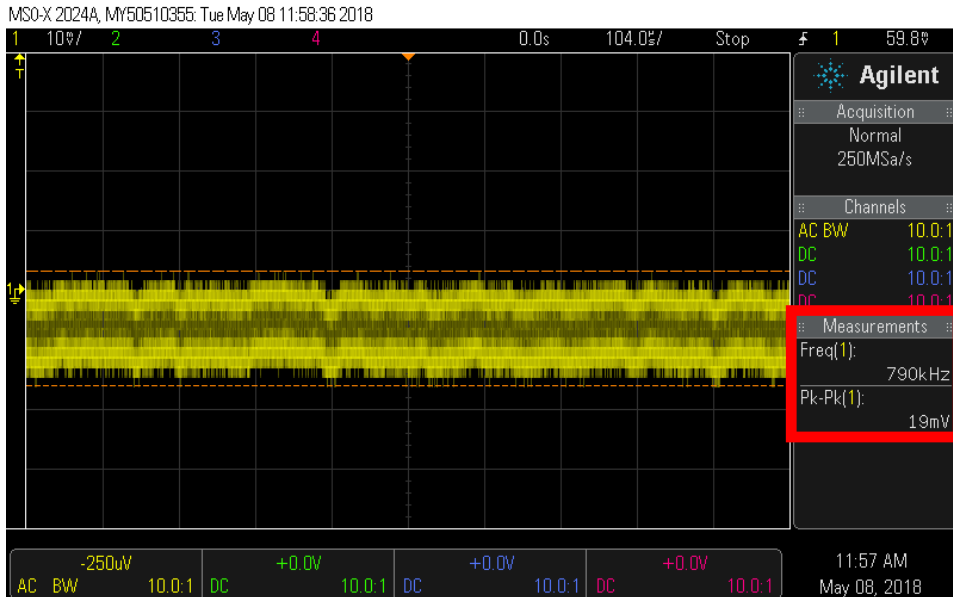
The oscilloscope is configured to AC coupling to filter out the stationary DC component that is not of interest, which also allows for greater sensitivity to display the AC component of the signal.[32] The system is running with the SoC performing Bluetooth actions, and the display performing simple operations. These operations should result in a current draw of about 60mA. The final voltage ripple results can be seen in Figure 4.8.

Now referring to the oscilloscope readings shown in Figure 4.8, the peak-to-peak voltage at the output of the the DC-DC converter shown in Figure 4.8a is $V_{pkpk_DCDC} \approx 19\text{mV}$. In Section 2.4 it was mentioned that the datasheet specified an output ripple of 27mV. Due to the differences between the test set-up specified in the datasheet, and the control system, it was argued that it was reasonable to expect the ripple voltage to be significantly lower than specified in the datasheet. This does indeed seem to be the case.

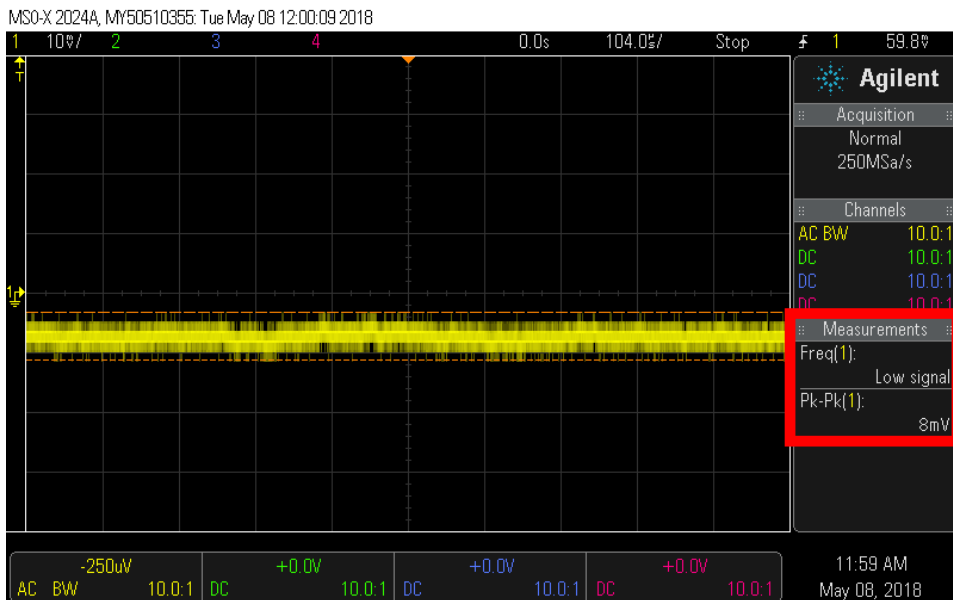
The peak-to-peak voltage at the output of the LDO regulator shown in Figure 4.8b is $V_{pkpk_LDO} \approx 8\text{mV}$. In Section 2.4 an estimate was calculated by accounting for the LDO's

PSRR. While the LDO certainly does attenuate the output ripple, it does not come close to the earlier estimate of $V_{pkpk_LDO_est} = 1.52\text{mV}$. A contributing factor to this discrepancy could be attributed to the specific set-up, which Texas Instruments warned could be problematic. The calculations used were also very simple, giving little or no consideration to the surrounding circuitry. Also, the PSRR graph in the datasheet listed the load current as 10mA.

While far from achieving the desired estimate of 1.52mV, the output ripple at 8mV from the LDO is well within the SoC's requirement of $V_{pkpk_LDO} < 100\text{mV}$.



(a) Output voltage ripple from 5V from DC/DC converter.



(b) Output voltage ripple from 3.3V LDO regulator.

Figure 4.8: Oscilloscope measurements of voltage regulator output.

Chapter 5

Software

This chapter details the software aspects of the new control system, encompassing the set-up of a development environment, driver development, and the development of a RTOS template application.

5.1 Development Environment

This section describes the set-up and configuration of a complete development environment. The physical set-up between a host computer and the control system is described. Nordic Semiconductor's software development kit (SDK) is set-up such that it is compatible with the control system. The RTOS tracing software Tracealyzer is implemented.

5.1.1 Host to Target Interface

The complete connection between host and SoC can be seen in Figure 5.1. For debugging and programming the nRF development kit will be used. This kit has an onboard J-Link that can be used for both the debugging and programming of an external device.

By default, the onboard J-Link is connected to the nRF SoC on the development kit. Once the control system is connected to the debug header of the development kit, the J-link automatically switches over to the external debug interface.

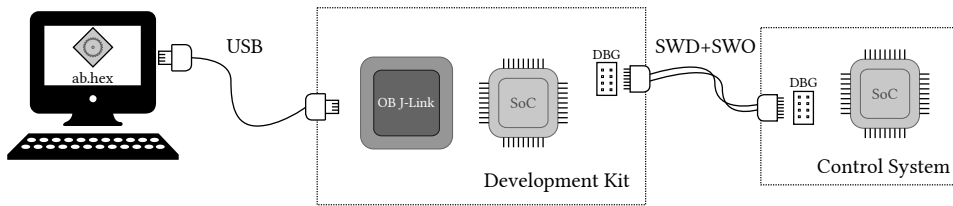


Figure 5.1: The connection between PC and control system.

5.1.2 Setting up a Minimal Example

The approach used for developing on the SoC is to use one of the example projects bundled with the SDK as a starting point. This way the project configuration is mostly set up correctly at the outset, meaning that less time has to be spent configuring the toolchain and setting up include directories. As described in Section 2.6, the IDE Segger Embedded Studio (SES) has been chosen, for which project files already exist in the SDK. For all Bluetooth projects, there are large amounts of boilerplate code. When using one of the examples as a starting point, all this code will already have been implemented correctly. Still, as the examples are designed to run on the development kit, some configuration will have to be done to make them compatible with the control system.

The example project used with the control system is the FreeRTOS example project, `ble_app_hrs_freertos_slam`. It has been set up with a FreeRTOS version 10.0.0 port, combined with Bluetooth functionality interfacing via a FreeRTOS task. It should be ideally suited as a starting point for the control system. The heap size is statically defined in the `FreeRTOSConfig.h`-file, and is by default set to a minimum only enough for a single task. Both the heap and stack sizes should be adjusted according to the application implemented, as shown in Listing 5.1. If the system runs out of heap memory, this can easily be caught and handled. Stack overflow is a more problematic issue, requiring more planning and techniques to handle correctly.^{1 2}

Listing 5.1: `FreeRTOSConfig.h`: Increase heap size for more tasks.

```
// Increasing heap to allow for spawning more tasks
#define configTOTAL_HEAP_SIZE ( 16384 )
// Estimate stack size based on usage with uxTaskGetStackHighWaterMark()
#define configMINIMAL_STACK_SIZE ?
```

¹Stack size: <https://www.freertos.org/FAQMem.html#StackSize>

²Stack overflow: <https://www.freertos.org/Stacks-and-stack-overflow-checking.html>

Unlike the development kit, the external low-frequency clock has been omitted in the control system. Part of the SDK configuration is the specification of low-frequency clock source. Either using the CMSIS Configurator³ or editing the `sdk_config.h` file directly, the low-frequency clock source have to be specified as the internal RC oscillator. The correct configuration is given in Listing 5.2. As can be seen from the listing, the RC oscillator requires periodic calibration, especially in case of fluctuating temperature.

Listing 5.2: `sdk_config.h`: SDK low-frequency clock source configuration.

```
// Legacy drivers should use internal RC oscillator as LF clock source
#define CLOCK_CONFIG_LF_SRC 0
// New drivers should use internal RC oscillator as LF clock source
#define NRFX_CLOCK_CONFIG_LF_SRC 0
// SoftDevice should use internal RC oscillator as LF clock source
#define NRF_SDH_CLOCK_LF_SRC 0
// SoftDevice calibration timer interval
#define NRF_SDH_CLOCK_LF_RC_CTIV 16
// SoftDevice calibration timer interval under constant temperature
#define NRF_SDH_CLOCK_LF_RC_TEMP_CTIV 4
```

Pin 11 on the SoC is by default set up with NFC functionality, and for this reason cannot be used as GPIO, as is required on the control system. The project will compile without error if it is used as a GPIO, but its output will remain HIGH at all time. This default behavior was discovered after hours of debugging using an oscilloscope. The solution is to add the symbol `CONFIG_NFCT_PINS_AS_GPIOS` in the list of preprocessor definitions in SES.

At this point the FreeRTOS example runs correctly on the control system.

5.1.3 Debugging

Already at this point, there are debugging facilities more powerful than those available on the ATmega-based control systems. Halt mode debugging is now available, meaning that single-stepping can be executed, and breakpoints added either in the assembly or code editor view. A shortcoming of this debugging quickly becomes apparent; if the application is stalled for too long, the time-sensitive Bluetooth subsystem crashes the SoC resulting in a hard-fault.

³A GUI tool supporting the CMSIS configuration format used in the `sdk_config.h` SDK configuration file.

Monitor mode debugging (MMD) is a less intrusive form of halt-mode debugging. Instead of completely halting the processor, interrupts handlers with a higher priority than the MMD are allowed to continue in the background. This way the Bluetooth system and motor control that have been assigned high priorities can continue to execute in the background during debugging. The concept is illustrated in Figure 5.2. The MMD libraries have been implemented in the SLAM project, where MMD is enabled by adding `SetMonModeDebugging = 1` to *Project Options* → *Debug* → *J-Link* → *Additional J-Link Options*, and calling `NVIC_SetPriority(DebugMonitor_IRQn, 7ul)` at the very start of the `main()`-function.



Figure 5.2: Monitor mode debugging principle.

As with the current ATmega control systems, debug printing is available. Instead of dedicating a UART to transmit the text, the RTT technology described in Chapter 2 is used. With the SDK, it is already implemented and combined with a complete library for logging. Its configuration and example usage is given in Listing 5.3. Debug printing can be useful for notifying about events, e.g., "robot scan complete." These can then be seen either directly within SES via the *Debug Terminal*, or via one of the dedicated RTT tools bundled with the J-link software package.

Listing 5.3: `sdk_config.h, main.c`: Enabling and using RTT.

```
// Configure RTT as back end for the nRF logging library
#define NRF_LOG_BACKEND_RTT_ENABLED 1
// Example function call to perform logging over RTT
NRF_LOG_INFO("Scan cycle %d completed \n", n_scan_cycles);
```

As long as the nRF logging library is redirected to RTT it will have superior performance compared with traditional UART printing. RTT was designed to enable high-speed transfers, with minimal impact on real-time performance.⁴

The final—and by far the most sophisticated—debugging tool will be Tracealyzer. This is software designed for FreeRTOS tracing and introduces RTOS aware debugging. The software consists of two components: a trace recorder library running on the target, and

⁴<https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/>

a visualization and analysis tool running on a host computer. The tracing library can operate in one of two modes: either in snapshot mode or in streaming mode. For the former, trace data is stored in RAM and transferred to host PC for post-mortem analysis. The latter continuously streams trace data to the external PC, allowing for live analysis during run-time. The streaming mode will require debugging hardware connected to the control systems at all time. The limited memory region holding the trace snapshot can be regularly be dumped to an SD-card, which is necessary for tracing prolonged SLAM sessions.

Tracealyzer is set up by running the installer after a free educational license has been obtained from Percepio⁵. The tracing library for the target is located at *Install_Dir/FreeRTOS/*. The three implementation files (.c), configuration files (.h) and include files (.h) are copied to appropriate folders into the example project. Listing 5.4 lists all the configuration necessary to set the trace library up for snapshot mode tracing, while Listing 5.5 lists all the configurations necessary to enable event tracing in FreeRTOS.

Listing 5.4: *trcConfig.h* Trace library target configuration.

```
#define TRC_CFG_HARDWARE_PORT TRC_HARDWARE_PORT_ARM_Cortex_M
#define TRC_CFG_RECORDER_MODE TRC_RECORDER_MODE_SNAPSHOT
#define TRC_CFG_FREERTOS_VERSION TRC_FREERTOS_VERSION_10_0_0
```

Listing 5.5: *FreeRTOSConfig.h*: Enabling trace in FreeRTOS.

```
#define configUSE_TRACE_FACILITY 1
#if ( configUSE_TRACE_FACILITY == 1 )
#include "trcRecorder.h"
#endif
```

To begin tracing on the target the function `vTraceEnable(startOption)` is first called with the parameter `TRC_INIT` to initialize the library, and then a second time with `TRC_START` to begin the actual tracing. It is important that tracing is started before any FreeRTOS function calls. There is a caveat with the current set-up though. As both debug printing, and the trace library uses RTT, they fail when both are in use. This caveat should not be much of a limitation considering that the user event support in Tracealyzer should eliminate the need for traditional printing. For convenience, this is solved automatically in the project by using the preprocessor. Usage format can be seen in Listing 5.6. The return value from `xTraceGetLastError()` should be checked before proceeding.

⁵<https://percepio.com/licensing/>

Listing 5.6: main.c: RTT interlock and Tracealyzer usage.

```
#if !NRF_LOG_ENABLED || !NRF_LOG_BACKEND_RTT_ENABLED
    vTraceEnable(TRC_INIT);
    vTraceEnable(TRC_START);
    error_msg = xTraceGetLastError();
#endif
// No FreeRTOS function calls prior to this line!
```

The trace data on the target is read out by supplying the software running on the host PC with the address of the data in the target's RAM. This address is obtained by pausing the target, and reading the global variable RecorderDataPtr. Before importing the data, Tracealyzer running on the PC has to be configured to use SWD and not JTAG. From *Settings* → *J-Link Settings*, set the debug interface to SWD, and the J-Link speed to 4000kHz. The trace data is then imported and analyzed by pressing the "Read Snapshot Trace" button from the right-hand panel within Tracealyzer.

At this point, all the debugging facilities are set up. These facilities include traditional halt-mode debugging, the less intrusive monitor mode debugging, real-time printing and RTOS aware trace functionality. These should all serve as a solid foundation for further application development.

5.2 Driver Development

This section details the development of drivers for the different devices on the SLAM robot and control system. The drivers include documentation in the form of comments conforming to the Doxygen standard. This means that PDF, LaTeX or HTML based documentation for the drivers can be generated.

All drivers have been successfully tested on relevant hardware, where they have performed as specified in this section. All drivers are available in the media attachment as described in Appendix A. They have all been implemented in the template FreeRTOS application set up later in this chapter.

5.2.1 Display

The display driver performs initialization of the OLED and provides a clean interface with drawing operations where all protocol and algorithms have been abstracted away.

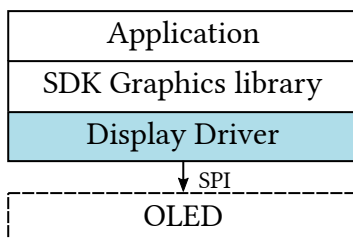


Figure 5.3: Components of the display driver.

As part of the SDK, there already exists a graphics library for drawing objects and printing text. This library is device agnostic and is only concerned with the algorithms for drawing pixels in the correct positions. To implement this drawing library, the display driver developed here has to supply the library with a set of basic functions, given in Table 5.1. These functions are registered with the library via function pointers. The structure of the display stack is illustrated in Figure 5.3

Table 5.1: OLED driver implementing basic functionality required by the graphics library.

Function	Description
<code>oled_init()</code>	Initialize the OLED.
<code>oled_uninit()</code>	Deinitialize the OLED.
<code>oled_draw_pixel(x,y)</code>	Draw pixel at x,y in display buffer.
<code>oled_draw_rectangle(x,y,w,h)</code>	Draw rectangle at x, y in display buffer.
<code>oled_display()</code>	Update OLED with display buffer content.
<code>oled_clear()</code>	Clear the display.

These are callback functions for the graphics library.

The OLED driver handles all communication with the display over the SPI bus. As this bus is shared with the microSD slot, care has to be taken when initializing the underlying SPI driver. If the bus were not shared, it would be safe to initialize the driver once in the `oled_init()`. Since the bus is shared it will instead check if the bus has already been initialized, and if so, deinitialize it, and reinitialize it. The function sending new display data to the OLED, `oled_display()` will have to perform the same procedure. While a bit cumbersome, this does ensure that the SPI driver is never double-initialized, double-deinitialized or otherwise left in a broken state. The SPI configuration is given in Listing 5.7, and the initialization in Listing 5.8.

Listing 5.7: oled_driver.c: Part of the SPI configuration used.

```
static nrf_drv_spi_config_t const spi_config = { [...] // snippet
    .miso_pin = NRF_DRV_SPI_PIN_NOT_USED,
    .frequency = NRF_DRV_SPI_FREQ_1M,
    .mode = NRF_DRV_SPI_MODE_0,
    .bit_order = NRF_DRV_SPI_BIT_ORDER_MSB_FIRST };
```

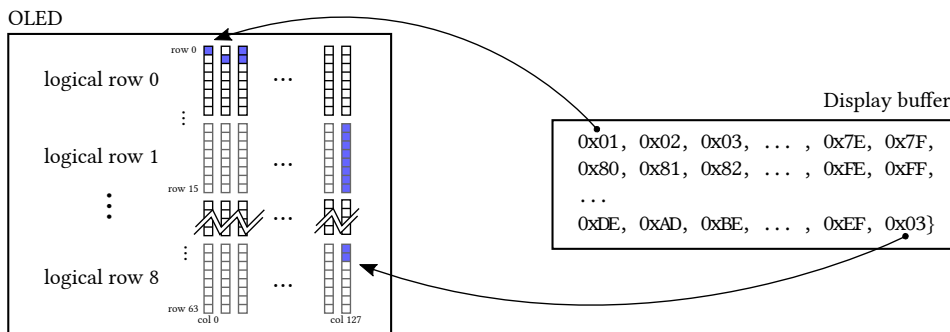
Listing 5.8: oled_driver.c: SPI initialization routine.

```
err_code = nrf_spi_mngr_init(&m_nrf_spi_mngr, &spi_config);
if (err_code == NRFX_ERROR_INVALID_STATE) {
    nrf_spi_mngr_uninit(&m_nrf_spi_mngr);
    nrf_spi_mngr_init(&m_nrf_spi_mngr, &spi_config);}

```

The initialization function performs the initial configuration of the OLED over the SPI bus. Among other configurations, it sets the display height, pixel mapping configuration, power supply settings and display contrast. Details about all the configurations are available in the OLED's datasheet. In the end, it performs a reset of the OLED, ensuring that it is in a valid state with the new configuration after the initialization routine has finished.

The SDK graphics library requires two primitive drawing operations to perform the more complex ones: drawing a single pixel and drawing a rectangle. The display driver maintains a display buffer on the SoC, where these drawing operations are performed locally. The `oled_display()` function is used to transfer this buffer to the OLED. The rectangle drawing function `oled_draw_rectangle()` simply uses the `oled_draw_pixel()` function to draw a rectangle to this buffer. The challenge is to implement the `oled_draw_pixel()`-function, such that the correct pixel in the display buffer is set.

**Figure 5.4:** Mapping between display buffer and pixels on the display.

The display buffer is defined as static `uint8_t display_buffer[64*128/8]`, and maps to the display as illustrated in Figure 5.4. Each byte’s binary encoding corresponds to the pixel configuration for a single column in a logical row. E.g., from the figure the first byte’s binary encoding is `00000001`, and thus the first pixel in the first column of the first logical row is turned on. In this manner, the first 128 bytes of the display buffer describes the first logical row, and the consecutive bytes the remaining logical rows. The function for setting a single pixel is designed around the bit twiddling given in Listing 5.9.⁶

Listing 5.9: `oled_driver.c`: Setting or resetting a single pixel.

```
if (set_pixel)
    display_buffer[x + (y/8)*OLED_WIDTH] |= (1 << (y&7));
else
    display[x + (y/8)*OLED_WIDTH] &= ~(1 << (y&7));
```

The functions defined in Table 5.1 are associated with the graphics library via an `nrf_lcd_t` struct, which is passed as parameter to the graphics library when it is initialized using `nrf_gfx_init(nrf_lcd_t)`. At this point all the drawing operations in Table 5.2 are available to the application developer.

Table 5.2: SDK graphics library drawing operations.

Function	Drawing Operation, and Associated Options
<code>nrf_gfx_point_draw(p)</code>	Pixel
<code>nrf_gfx_line_draw(l)</code>	Line, thickness
<code>nrf_gfx_circle_draw(c)</code>	Circle, fill
<code>nrf_gfx_rect_draw(r)</code>	Rectangle, thickness, fill
<code>nrf_gfx_bmp565_draw bmp)</code>	Draw image from .BMP-file
<code>nrf_gfx_print(t)</code>	Text

5.2.2 microSD

The microSD driver is essentially a restructuring of the FatFs example provided as part of the SDK. FatFs is a File Allocation Table (FAT) filesystem library for interacting with devices conforming to the FAT specifications. By using a filesystem on the microSD card,

⁶Since the buffer is byte-addressable, not bit-addressable, the bitwise operators are used to leave the remaining bits in the byte intact.

files and folders can be used to contain and organize all types of logging and writing to the microSD card. It also makes it very easy to transfer the data to a PC, as all PC operating systems have native support for the FAT filesystem. For example, a task can write its log-data to `a.txt`, while another task writes its log data to `b.txt`.

The driver interface consists of a single function, as given in Table 5.3. If the file does not exist, it will be created and the content written to the newly created file. If the file does exist, the content is appended to the already-existing file.

Table 5.3: microSD-card driver interface.

Function	Description
<code>microsd_write(<i>f</i>,<i>c</i>)</code>	Write content <i>c</i> to file <i>f</i> . If <i>f</i> exists, <i>c</i> is appended to <i>f</i> .

The write function will initialize and re-mount the file-system on each invocation. This makes the function stateless, making it easy to implement as part of a RTOS project safely. The downside is the performance penalty associated with the initializations and mounting operations.

As the microSD shares SPI bus with the display, double initialization or deinitialization must be avoided. This is avoided by modifying the code part of the microSD card driver that performs the SPI bus initialization (`app_sdcard.c`), in the same manner as was done with the display driver.

5.2.3 Motors

The motor driver provides individual control of each of the two motors on the robot. This control is achieved by controlling the H-bridge on the motor controller board. The driver interface is given in Table 5.4. The driver uses the PWM library part of the SDK, and the GPIOTE driver to configure the type of operation (e.g., forward, backward or brake).

The PWM library consumes one timer peripheral, and provides two channels. The duty cycle can be configured individually for each of the two channels; only the PWM frequency has to be identical. The result is full individual control of each of the two motors. Each motor can thus be set off, stopped (brakes), turn forward or backward.

This driver supplies all the operations available from the motor controller board, for each of the motors. It is expected that for the SLAM application, either a new layer is built on

Table 5.4: Motor driver interface.

Function	Description
<code>motor_dir_forward(p)</code>	Run left or right motor forward with $p\%$ power.
<code>motor_forward(p)</code>	Run both motors forward with $p\%$ power.
<code>motor_dir_backward(p)</code>	Run left or right motor backward with $p\%$ power.
<code>motor_backward(p)</code>	Run both motors backward with $p\%$ power.
<code>motor_dir_stop()</code>	Stop supplying power to left or right motor.
<code>motor_stop()</code>	Stop supplying power to both motors.
<code>motor_dir_brake()</code>	Actively brake left or right motor.
<code>motor_brake()</code>	Actively brake both motors.

top of this driver, or it is refit for the new application. It would seem natural to include a closed loop speed controller, using the encoder feedback.

5.2.4 Encoders

The encoder driver exposes two different models of interacting with an encoder. The driver interface can be seen in Table 5.5.

Table 5.5: Encoder initialization functions.

Function	Description
<code>encoder_init_int(side, cb_fn)</code>	Interrupt driven encoder initialization.
<code>encoder_init_ppi(side, timer)</code>	PPI and counter driven encoder initialization.

In the first model, the user supplies the driver with a callback function. Encoder actions on that side are then set up to trigger interrupts on the GPIOTE system. These interrupts will, through a trampoline callback function in the driver, call the user-supplied callback function. This corresponds to the set-up that is used in the current robots, except that the application developer supplies a normal callback function, and does not have to consider the underlying interrupting structure. This model requires more time from the processor, but it does not consume additional peripherals, which the PPI-based method described below does.

The second model uses the PPI functionality to perform the counting autonomously in the background, without using any of the processor's time. This offloading of the processor is

accomplished by setting up a PPI channel between the event generated in the GPIOTE subsystem on the encoder pin, and a counting task on the supplied timer peripheral. As the application developer provides the timer peripheral, the encoder count can be captured and read on one's convenience. The drawback is the consumption of possibly two timer peripherals.

The encoders mounted on the robot are not quadrature encoders, which means that it is not possible to establish direction based on the output signals they generate. This puts the burden of maintaining directionality on the application developer.

5.2.5 IR

The functions making up the driver interface are given in Table 5.6. The initialization function sets up the ADC with resolution r and sets up for DMA transfer of results. It sets up the correct ADC reference voltage and scaling to match that of the IR's output. It also performs an initial calibration of the ADC peripheral. There is a static configuration, `N_SAMPLES` that specifies how many samples to average before returning. The default has been set to four samples.

The IR sensors output a voltage as a function of measured distance. The voltage-distance relationship is given in the data sheet, and a look-up-table (LTU) with calibrated values are used on the current Arduino robots. For the driver developed in this thesis, only the functioning of the ADC system has been of interest, and thus no lookup table has been implemented.

Table 5.6: IR driver interface functions.

Function	Description
<code>ir_init(resolution)</code>	Initializes and configures ADC with DMA.
<code>ir_read(sensor, cb)</code>	Perform reading and transfer with DMA.
<code>ir_read_blocking(sensor)</code>	Perform reading, and return value when done.
<code>ir_calibrate()</code>	Performs calibration.

5.2.6 Sensor Tower Servo

This driver allows for control of the sensor tower angle. The driver interface is given in Table 5.7.

As with the motor driver, a PWM peripheral will be used for controlling the servo. But here PWM is used to encode a signal, and not as a technique for limiting the power supplied as with the motor.⁷

Table 5.7: Servo driver interface functions.

Function	Description
<code>servo_init(resolution)</code>	Initialize PWM on the servo pin.
<code>servo_rotate(θ)</code>	Rotate to angle θ .

The driver has not been calibrated to ensure that the servo moves into the correct positions, but it has verified that the servo moves when changing PWM value. Another issue is the use of the PWM library. It has very coarse resolution on the PWM, resulting in large increments in servo position. A solution that was tested was to use the PWM drivers instead. That showed significant improvements, but the downside is that this driver has been designed for controlling LEDs, and is therefore dead set on using sequences and other unnecessary features making simple modifications to duty cycle overly complicated.

5.2.7 Magnetometer

The purpose of the drivers for the magnetometer is to output its direction, in the range of $\pm 180^\circ$. The heading is calculated using an algorithm taken from SparkFun's library for the same device. In order to use the `mag_heading()`-function, the magnetometer first has to be calibrated. This is done by calling `mag_calibrate()`, and then within a given time-frame rotate the control system 360° . Table 5.8 lists the driver's interface.

Table 5.8: Magnetometer driver interface functions.

Function	Description
<code>mag_init(os)</code>	Initialize magnetometer with oversampling <code>os</code> .
<code>mag_calibrate()</code>	Perform calibration routine.
<code>mag_heading()</code>	Returns the magnetometer heading, $\pm 180^\circ$

⁷It would be technically more correct to refer to this as pulse position modulation (PPM), but such pedantry is more likely to confuse than anything else.

5.3 Template Application

A template FreeRTOS application is set up in this section, based on the example application from the SDK that was configured to work with the new control system in Section 5.1. The primary purpose of this template application is to integrate all the drivers into a single RTOS. In doing so, the opportunity is also used to showcase some of the synchronization mechanisms part of FreeRTOS.

5.3.1 Gatekeepers

There are primarily two shared resources in the control system: the display and the microSD slot. All task should be able to use any of these resources in a thread-safe manner. One method for solving this issue is to set up mutual exclusion on the resources, using a mutex. This approach introduces mutex-related issues such as deadlocks and priority inversion.

Instead thread-safe set-up for the display is set up according to the gatekeeper task principle. This involves a dedicated task managing the resource, resulting in a synchronization scheme not prone to deadlocks or priority inversion. [33] This approach will be used to ensure safe concurrent access to the display and microSD card. The implementation for the display is described here, while the implementation for the microSD card is omitted as it is principally identical.

A complete overview of the system can be seen in Figure 5.5. In the lower right the actual display driver is situated, which was detailed in Section 5.2. Neither the display driver nor this graphics library needs to be thread-safe, as only a single thread is ever allowed to interact with them: the Display Task.

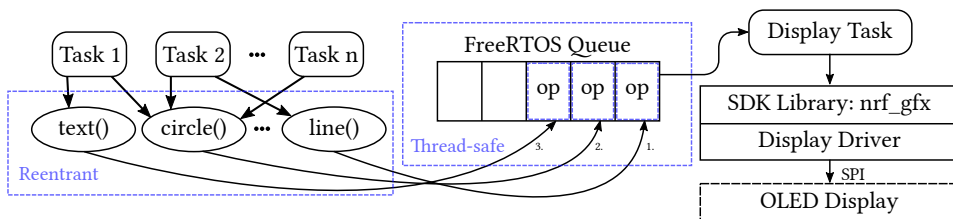


Figure 5.5: Overview of system architecture with gatekeeper task for the shared display resource.

Associated with the Display Task is a FreeRTOS queue, which the Display Task will wait on. Such a FreeRTOS queue is inherently thread-safe, and is one of many synchronization

mechanisms provided by the RTOS. All display drawing operations in the system are enqueued into this queue, ensuring that there will be no collision issues using the display driver and graphics library.

The gatekeeping Display Task constructs a new queue with five slots for elements containing display operations. It then enters a loop that starts out by waiting on the queue. Once there is an element (display operation) in the queue, the element is removed from the queue and handled by one of the several clauses in a switch statement. Pseudo-code part of the Display Task is given in Listing 5.10.

Listing 5.10: `task_display.c`: Display Task structure.

```
display_queue = xQueueCreate(5, sizeof(display_operation_t));

for ( ;; )
    xQueueReceive(display_queue, &display_operation, portMAX_DELAY);

    switch (display_operation) { ... } // perform drawing operation
```

The drawing operations combined with the enqueueing operations are somewhat complex. In order to hide this from the application developer, a thin abstraction layer consisting of helper functions is set up. An important property of these helper functions is that they only ever manipulate thread-local (stack) variables or variables in registers, and never heap-allocated data. Such functions are reentrant, which means that they are thread-safe. [33] For example, the `display_point()` helper function creates and configures a new point draw operation, creates and configures a point display element and finally enqueues the whole operation on the Display Task's operation queue.

The same approach with a gatekeeping task is used for the microSD resource. A new queue is created that stores write operations instead of drawing operations, and a dedicated microSD Task is set up to handle all the operations enqueued.

5.3.2 System Architecture

FreeRTOS is set up with tasks for each of the drivers created, in addition to the preexisting Bluetooth task that regularly polls the `SoftDevice` to check for new Bluetooth events that need to be handled. As the example is based on the Bluetooth heart-rate monitor example, this application is left intact and is fully functional when connected to over Bluetooth

with an Android smartphone running the nRF toolbox.⁸ The architecture is illustrated in Figure 5.6.

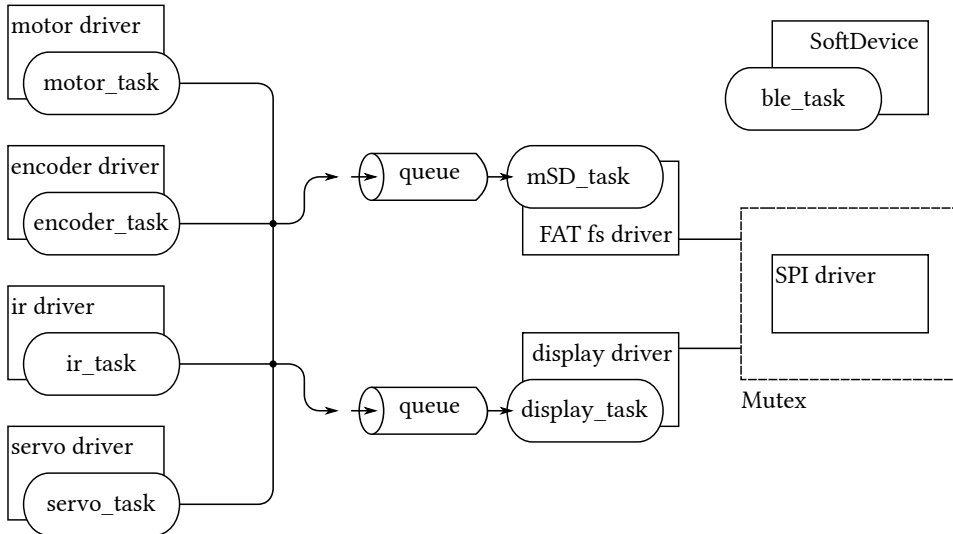


Figure 5.6: Architecture of template application with tasks, drivers and synchronization mechanisms.

As can be seen in Figure 5.6, any of the tasks on the left may enqueue item into either of the two queues. These queues belong to the gatekeeper tasks previously set up. `mSD_task` for writing to a portable microSD card, and `display_task` for display operations

As both the microSD slot and OLED share a common SPI bus, a mutex has been set up to synchronize access to the bus. Note that the SPI drivers supplied in the SDK had to be modified to accommodate the sharing of the SPI bus between two slaves.

5.3.3 Autonomous Scan Cycle

A suggested solution where PPI and DMA are used to make the SLAM scan cycle autonomous is now presented. As described in Chapter 1, this is the cycle where distance measurements are taken to be used to generate a map. Due to time constraints, it has not been implemented.

The flowchart in Figure 5.7 illustrates the scan cycle process, where it can be seen that the processor only has to perform the initiation. All events and tasks part of the PPI

⁸The nRF toolbox is an application for Android from Nordic Semiconductor that can interact with the SDK examples. For this example, that includes generated heart-rate and battery-status.

subsystem are indicated, as it is these that ensure the scan cycle's progress.

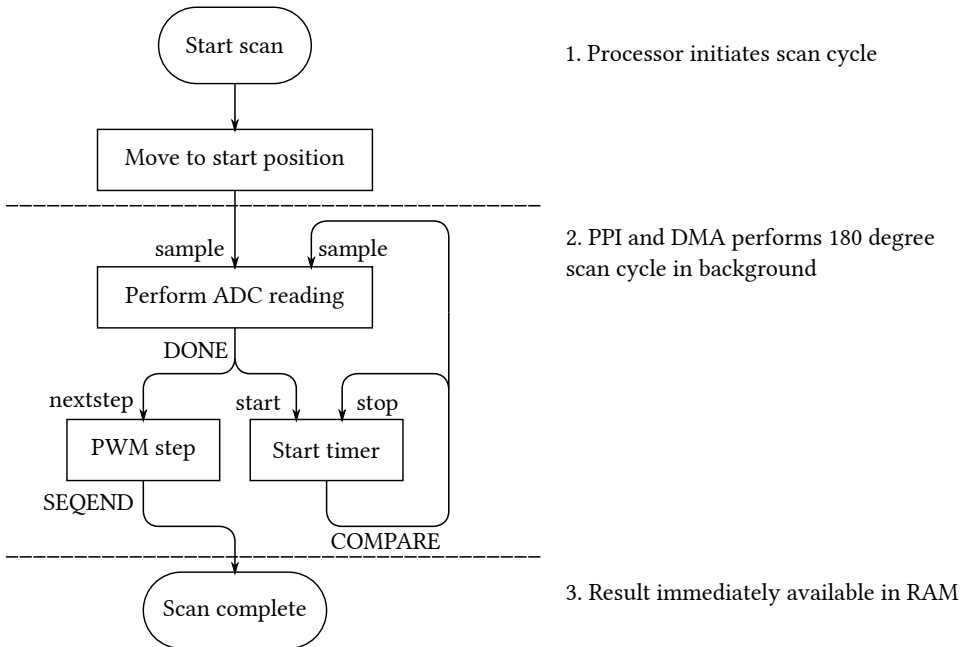


Figure 5.7: Flow chart describing an IR scan cycle. Event names in all upper case, tasks in all lower case.

The PWM peripheral features a DMA decoder, which means that a duty-cycle sequence can be put into memory, and this decoder can step through the sequence. Thus a simple sequence of n increments is set up, where n effectively specifies the scan cycle radial resolution.

Unfortunately, the PWM peripheral event system seems to lack an event indicating that it has progressed to the next step in a sequence. Instead, there is only an event for sequence completed. To ensure synchronization between the rotation and IR readings a more pragmatic solution is opted for, using a timer. If the PPI initialization is chosen for the encoders, this means there are no more TIMER peripherals available. Fortunately a slightly lower-resolution timer running on the low-frequency clock is available from the RTC (Real Time Counter) peripheral.

The timer must run for at least so long such that the servo and IR-sensor have reached steady state before initiating an ADC conversion. As the ADC triggers an event when it has finished the conversion, a second timer will not be necessary for the ADC conversion.

Peripherals for both the IR sensor and sensor-tower servo are now capable of performing

their tasks. What remains is to orchestrate the whole scan cycle using PPI, DMA and the RTC peripheral as illustrated in Figure 5.7. There are three parts to setting up PPI between peripherals. First, a channel has to be allocated. Then a peripheral event, and one or more peripheral tasks are registered with the allocated PPI channel. The final step is then to activate this channel.

Chapter 6

Results

This chapter presents the final state of the control system. The first section details the hardware aspects, such as soldering and electrical functionality and performance. The second section details the software aspects, including the status of the development environment, drivers for the different systems as well as the template application developed.

6.1 Hardware

The first revision of the control system has been successfully built, with only the soldering of the IMU remaining. The power supply circuit performs well, producing a stable supply with a peak-to-peak voltage ripple of about 8mV. Bluetooth connectivity works and has been verified using an Android smart-phone to connect to example applications from the SDK. A picture of the final assembled control system can be seen in Figure 6.1. For the SoC and magnetometer, continuity testing has been performed on all pins, ensuring that there are no shorts due to, e.g., solder bridges.

A second revision of the schematics and PCB layout has been designed. Besides some general improvements, this second revision fixes the broken battery status circuitry.

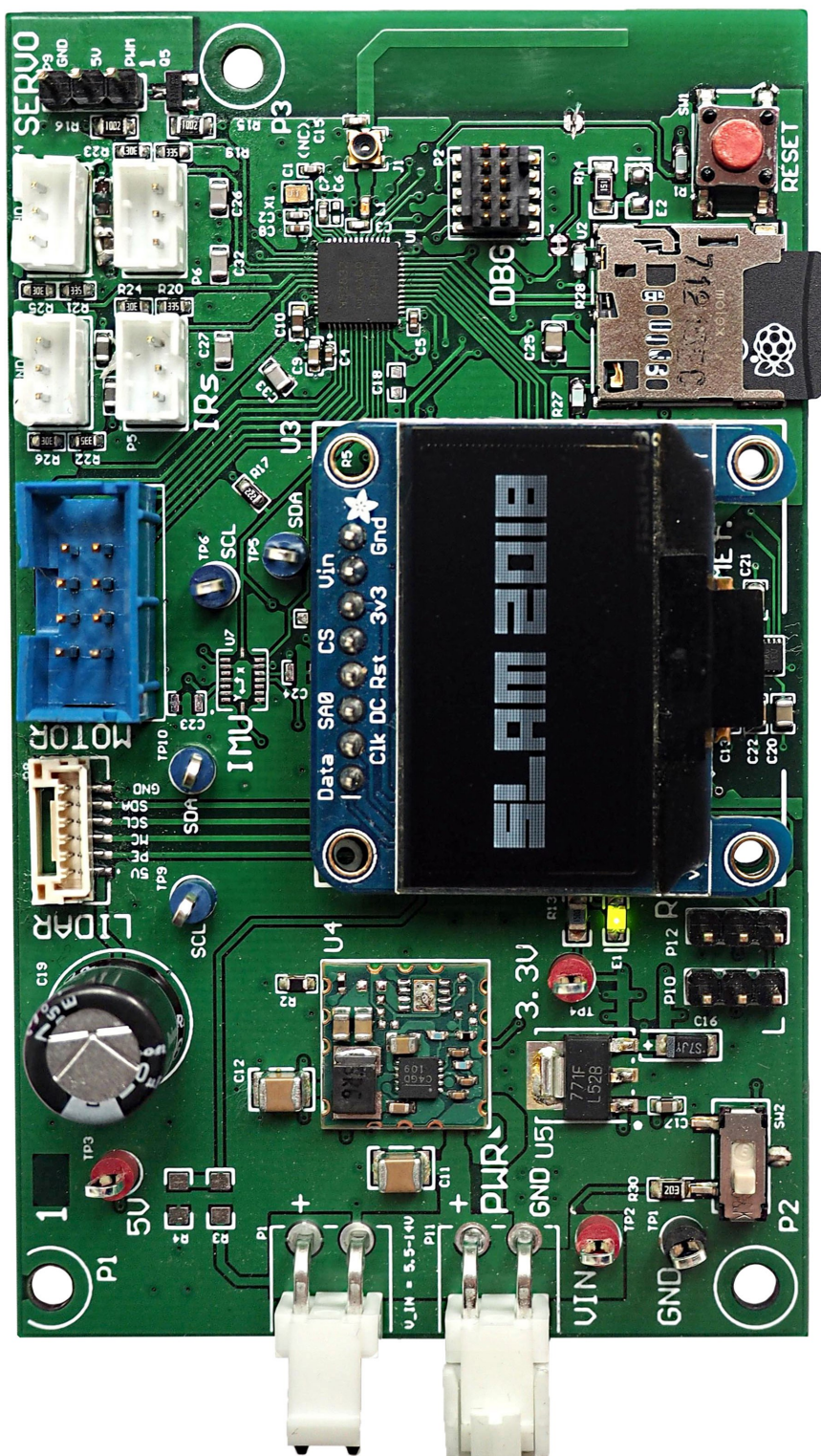


Figure 6.1: Assembled control system.

6.2 Software

Drivers have been developed for all systems except the LIDAR, Bluetooth¹ and IMU. The status is summarized in Figure 6.2. Red indicates that the system is malfunctioning, gray indicates that the system has not been mounted or soldered, yellow indicates that the system has been mounted or soldered, while green indicates that it is up and running with drivers.

6.2.1 Development Environment

For debugging much of the basic functionality simply worked without modifications. This functionality includes single-stepping and real-time printing functionality via RTT. Monitor mode debugging has been enabled in the template application, albeit in a trial version, where a trial pop-up appears.

As detailed later in this section, Percepio's Tracealyzer has been successfully implemented in a template application developed.

6.2.2 Drivers

The following drivers have been made:

- **Display** drivers have been developed and set up a graphics library part of the SDK.
- **IR-sensors** drivers have been set up, where reading works via ADC, but the translation to distance via, e.g., a lookup-table has not been implemented. This driver uses DMA.
- **Encoder** drivers have been set allowing for two different usage patterns: interrupt-driven or using PPI.
- **Magnetometer** drivers have been developed, including calibration routine and outputting of $\pm 180^\circ$.
- **Servo** drivers for the sensor tower has been set up.
- **microSD** has been set up with FAT file system support using FatFs.
- **Motor** drivers have been developed for direction, speed and brake control.

¹Bluetooth is operational, but the SLAM protocol has not been implemented.

The following drivers are missing:

- **IMU** has not been mounted. Drivers have been published by Bosch, and they including a porting guide.²
- **Lidar** drivers have not been developed. The device can output distance measurements directly over I2C, so only a very minimal layer to abstract away the I2C operations should be necessary. I2C is already up and running with the magnetometer.
- **Bluetooth** works, but the SLAM protocol has not been implemented.
- **Battery status** is broken due to fault in PCB layout. No amount of drivers can fix this.

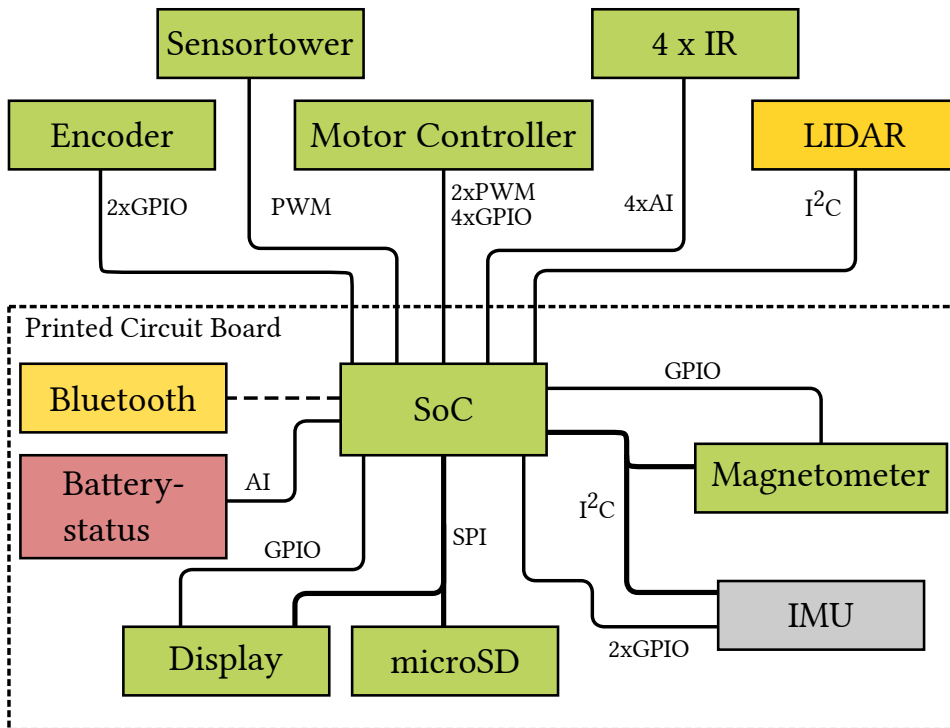


Figure 6.2: Status of driver and assembly of control system components.

²https://github.com/BoschSensortec/BMA2x2_driver

6.2.3 Template Application

A template application based on Nordic Semiconductor's FreeRTOS port has been developed and set up. It features tasks for each of the drivers created. A gatekeeper synchronization mechanism is set up for display and microSD operations. As the display and microSD share a common bus, a mutex is set up to protect the bus.

Tracealyzer trace library has been successfully implemented in the template application, enabling real-time operating system aware trace functionality. An example trace from the template application can be seen in Figure 6.3 and Figure 6.4.

Figure 6.3 shows the trace view in the upper pane and the processor profiler in the bottom pane. The trace view is vertical timeline clearly showing the scheduling action. Also, it highlights all system events. In the figure the Display Task "DSP" in dark blue can be seen blocking on the display-operations queue (red rectangle). The processor profiler has been moved to the start-up sequence, where it can be seen it does not take long until the microSD Task "SD" in yellow and Display Task "DSP" in dark blue occupy the processor for most of the time.

Figure 6.4 shows the object history view in the top pane and the communication flow view in the bottom pane. The object here is the mutex ensuring mutual access to the SPI bus.³ Both gatekeeper tasks, microSD "SD" in yellow and Display Task "DIS" in dark blue, can be seen taking and releasing the mutex. The communication flow visualizes inter-task communication flow. In the figure the User Task "USE" in orange requests a drawing operation by enqueueing it. The Display Task "DIS" in dark blue receives the operation via this queue, which will attempt to lock the mutex to safely perform the drawing on the OLED over the SPI bus. Simultaneously it logs this operation by enqueueing it onto the microSD Task's queue. The Motor Task "MOT" in bright green requests logging to the microSD card when the motor changes direction.

³In FreeRTOS lingo, a mutex is a type of semaphore.

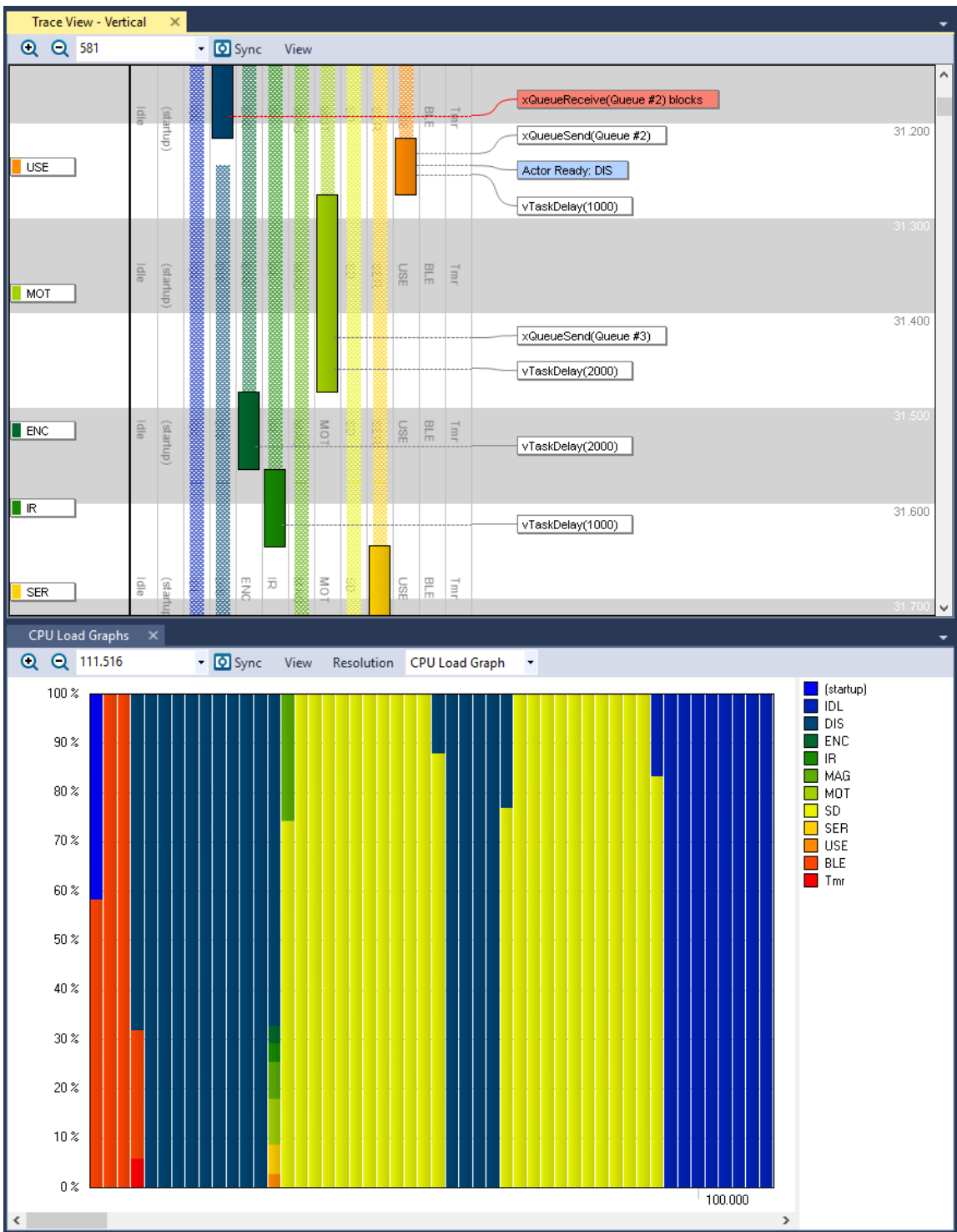


Figure 6.3: Tracealyzer used on the template application. Trace view and processor profiler shown.

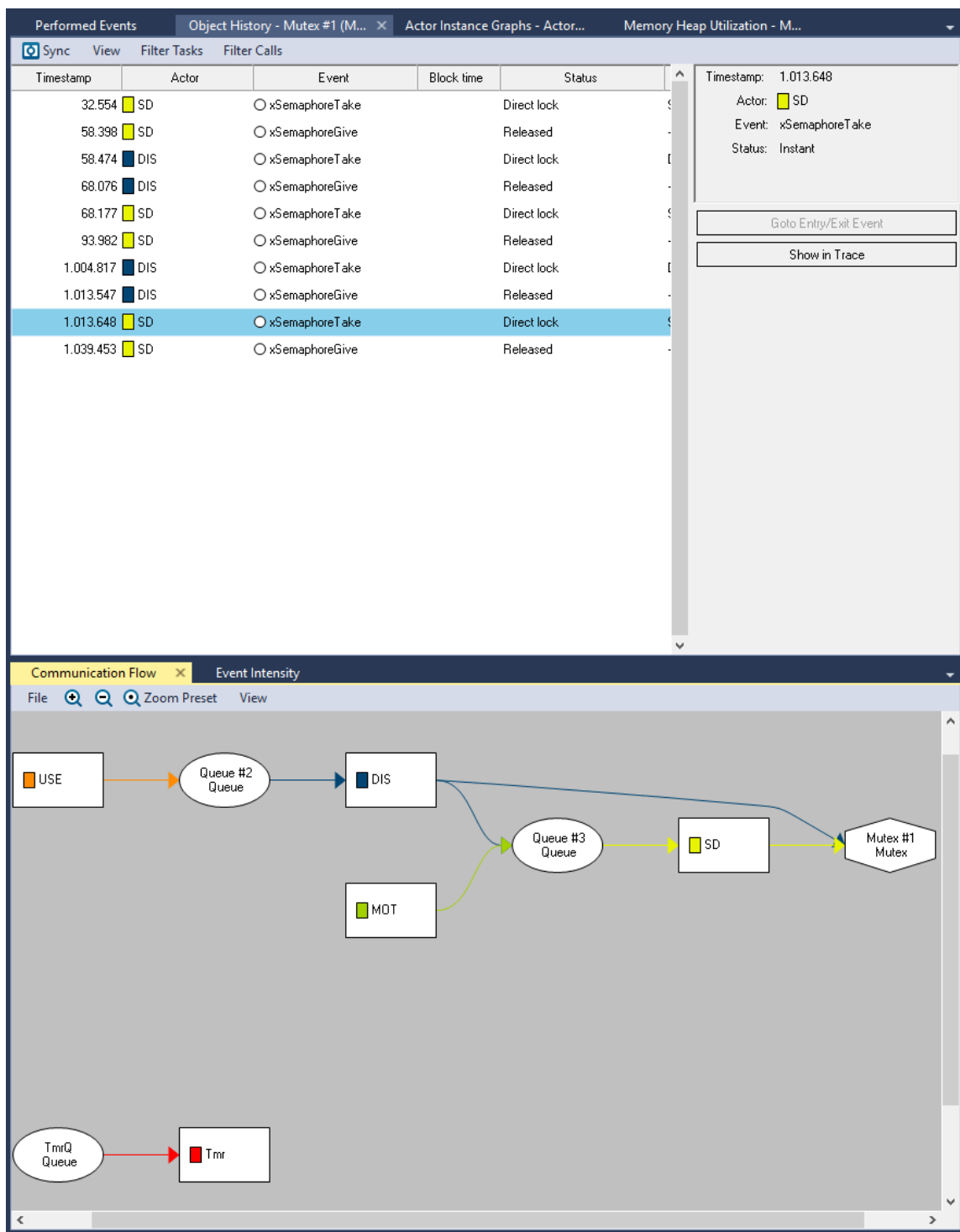


Figure 6.4: Tracealyzer used on the template application. Object history and communication flow.

Discussion and Further Work

This chapter starts off with a section discussing the problem statement, and the solution developed and presented in this thesis. Experiences made during the execution are discussed, and whether the approaches that have been taken during the development were found to be appropriate. It finalizes in a statement regarding this project's relevance and contribution to the overall ongoing SLAM project.

The final section details further work such as preparation for robot implementation, and a strategy for the development of a second edition. A starting point is given for the development of the SLAM application itself, where references are made to current implementations on the AVR robots and the project and mater theses available.

7.1 Discussion

The goal of this project has been to develop a new integrated control system for the SLAM robots using the nRF52 SoC. The purpose of the migration is manifold. Some of the primary reasons were to enhance robot computing power, simplify and unify the on-robot electrical set-up and introduce proper RTOS aware debug capabilities. Also, the opportunity was used to add new features such as an onboard display and a microSD slot for extended portable flash storage.

A trait of the current robots was the disunite structure of the control system, and the components and circuit boards part of it. This all attributes to a distrust in the electrical

system, where the application developer had to consider the hardware aspects when bugs occur, perform visual verification and possibly start probing with an oscilloscope. Students that had worked on the robot voiced concerns about the stability of the robot, where it would work one day, just to malfunction the following day running the same exact software. Alluding this to hardware does not seem unreasonable.

The merging of the SoC and all components into a single unified circuit board certainly goes a long way to alleviate the issues with the previous design. As long as the soldering is done properly—which is verifiable for the majority of components—it should now be sufficient to ensure that the relevant systems operate correctly once. If a bug occurs now, the application developer can spend time debugging the application, instead of worrying about the hardware.

It is worth noting that the introduction of proper connectors, as opposed to standard headers, does raise some concerns as to ease-of-prototyping with new hardware. The dissuasive pricing and limited availability of the crimping equipment for the JST connectors mean that more complicated unofficial methods will have to be used to create the connectors. Proper connectors should be considered a matter of trade-off between ease of prototyping and hardware reliability. Since the project has been ongoing for a long time, the assumption has been made that hardware selection has stabilized, and therefore secure connections have been prioritized.

While it was rather easy to see the improvements in storage capacity using the new SoC, a single unambiguous measure of computing power across the AVR and SoC architectures proved to be somewhat tricky to obtain. However, the significant difference in clock frequency combined with the Cortex-M4F's 32-bit architecture with DSP and FPU capabilities goes in favor of the SoC.

The cost associated with this upgrade to a 32-bit platform is an increase in the architectural complexity. The Cortex pipelines are deeper, and the number of special registers and instructions have increased compared to the ATmegs. But the development methodology seems to change with the 32-bit platforms. As a general rule, there will be drivers that the application developer interacts with, meaning that tinkering with low-level registers is no longer necessary. In the author's experience, this works so well that it is easier to develop on these platforms than on the AVRs, where the datasheet continually have to be referred to find out precisely which registers that need to be configured. The abstractions are there, and they seem to work well.

It is the real-time aspects of the new control system that are some of the most impres-

sive. While the availability of zero-copy mechanisms via DMA and direct peripheral-to-peripheral interaction via the PPI system is beneficial, it is in the domain of debugging the most significant real-time improvements are made. The limited debugging facilities that have been available this far in the SLAM project has been a major curb for the application developers. With some of the current robots limited to printing via UART peripheral and a set of three LEDs, the debugging of an RTOS system has at times been like searching for a needle in a haystack.

With the introduction of Monitor Mode debugging, single-stepping can be performed while motor control, BLE communication, and other higher priority tasks still run. RTOS aware debugging is provided in the form of trace features and dedicated software for analyzing the trace data. The software offers features such as visualization of thread scheduling, usage patterns on synchronization mechanisms such as timers, semaphores, and queues. These features could potentially be one of the most significant upgrades to the whole SLAM project since its inception. While the other new features and improvements are important, the new debugging functionality is likely to be the most appealing upgrade for the application developers.

The new features introduced do come off as appealing at first. Especially the onboard display was a feature highly desired by those working on the current robots. This desire had likely arisen as a direct consequence of the limited debugging options available, limited to printing over serial or encoding custom errors via a set of three LEDs. With the powerful debugging facilities introduced in the new control system, the author is left questioning the value of the display. It consumes a significant amount of board space and requires several pins, of which there is already a shortage. Probably the most interesting feature of the display—leveling it above a gimmicky feature—is the ability to read out detailed debug information during SLAM operations, where it could be cumbersome to have cables attached. Concerns have been raised as to how this is handled in current robots, where such debug information is transmitted via the BLE dongle, something which has shown to be a somewhat unreliable solution.

The microSD support is essential for logging RTOS trace data during live SLAM operations, where a system with many events can generate megabytes of trace data in a limited amount of time. It also introduces the possibility for stable (as opposed to BLE) logging of vast amounts of data (e.g., sensor, events, log), and the storage of bitmaps used for the display.

Some work remains before the new control system can be implemented on any of the SLAM robots. But even so, the control system in its current state looks very promising, and it seems to be able to create for a much more powerful platform for future develop-

ment of SLAM applications. It is more computationally powerful, more robust and most importantly of all: it features modern, sophisticated debugging facilities. The developers will now have full insight into the running RTOS, which removes a lot of the guesswork involved in the current debugging. Hopefully, this will accelerate development on future projects, ensuring that the developers can spend more time on the SLAM challenge, and less on debugging and fixing the hardware they are using.

7.2 Further Work

This section lists remaining and suggested future work on the new control system. The first two subsections are concerned with the finalizing of the control system for robot implementation, while the last subsection moves on to the actual SLAM application.

7.2.1 Completing the Control System

It is possible to integrate the control system developed in this thesis on the current robots, but thought should be given to whether it would be a better idea to set up manufacturing of the second revision as outlined below.

BLE is functional, but the protocol used between the PC running the Java server and the robots is not implemented. Note that part of the protocol design is to ensure reliable UART communication between the ATmega and the nRF dongle, and is therefore not necessary on the new control system. For implementing the protocol, good starting points would be Lien's thesis where it was initially developed, and Iversen's project thesis where it was ported to a new platform.[34] [35]

The only remaining component to be mounted and developed drivers for—neglecting the broken battery voltage measurement system—is the IMU. Bosch has published drivers for the IMU that include guidelines for porting the drivers to new platforms.¹ These drivers should serve as a good starting point for getting the IMU up and running.

While the control system is already running a template application with FreeRTOS, in its current state it serves as a proof-of-concept. It showcases the usage of central platform features such as synchronization mechanisms, PPI, DMA and the functioning of peripherals using the drivers developed.

¹Drivers and porting guides available at https://github.com/BoschSensortec/BMA2x2_driver for the accelerometer, and https://github.com/BoschSensortec/BMG160_driver for the gyroscope.

7.2.2 Second Revision

Complete schematics and layout have been made for a second revision of the control system, correcting some issues and introducing some improvements to the design. Having discovered what works, and what works less well in the first revision, it could be a worthwhile endeavor to set up a production line where not only the PCB is manufactured, but also the component assembly and soldering are performed at the factory. This way more effort can be spent on the development of the actual SLAM application, and less on the hardware manufacturing aspects that have been covered in this thesis.

Alternatively—depending on the outcome of the project implementing a LIDAR that has been ongoing this semester—a third design revision with LIDAR-support only could be developed. Removing IR support will free up four pins with ADC support on the SoC, and frees part of the PCB real estate.

7.2.3 SLAM Application

When the control system has been integrated with a robot, the development of the actual SLAM application can begin. This development has never been within the scope of this thesis. A good place to start is reviewing the source code currently used on the AVR robots, as these are running the same RTOS. Besides the source code, all previous reports in the SLAM project should be available, where the most central ones have been listed in the introduction in Chapter 1

The mechanical state of the SLAM robot that was assigned in this project was not impressive. For the implementation on a robot, it is strongly suggested that a new robot is developed based on one of the more recent designs part of the SLAM project. In addition, the encoders should be replaced with quadrature encoders, such that wheel direction can correctly be accounted for.

Bibliography

- [1] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part i,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, Jun. 2006, ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022. [Online]. Available: <http://ieeexplore.ieee.org/document/1638022/>.
- [2] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): Part II,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, Sep. 2006, ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1678144. [Online]. Available: <http://ieeexplore.ieee.org/document/1678144/>.
- [3] T. K. Homestad, *Fjernstyring av Legorobot*. 2013. [Online]. Available: <https://brage.bibsys.no/xmlui/handle/11250/260903>.
- [4] E. Ese, *Sanntidsprogrammering på samarbeidende mobil-robotar*. 2016. [Online]. Available: <https://brage.bibsys.no/xmlui/handle/11250/2403570>.
- [5] H. Skjelten, *Fjernnavigasjon av LEGO-robot*. 2004.
- [6] J. M. H. Bakken, *Bygge og programmere ny Legorobot*. 2008.
- [7] J. Stüper, *LEGO Mindstorms EV3 robot*. 2015.
- [8] T. E. S. Andersen and M. G. Rødseth, *System for Self-Navigating Autonomous Robots*. 2016. [Online]. Available: <https://brage.bibsys.no/xmlui/handle/11250/2403559>.
- [9] K. Z. Helders, *Real-Time System Implementation on Autonomous Lego-Robot*. 2017. [Online]. Available: <https://brage.bibsys.no/xmlui/handle/11250/2451636>.
- [10] G. H. Eikeland, *Implementation of Mapping and Navigation on an Autonomous Robot*. 2018.

-
- [11] Atmel, *AVR Instruction Set Manual*. 2016. [Online]. Available: <http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf>.
- [12] M. McDermott, "The ARM instruction set architecture," 2008, [Online]. Available: http://users.ece.utexas.edu/~valvano/EE345M/Arm_EE382N_4.pdf.
- [13] J. Ø. Amsen, *Improving Navigation and Mapping with Arduino robot*. 2007. [Online]. Available: <https://brage.bibsys.no/xmlui/handle/11250/2451639>.
- [14] J. C. Teel, "Understanding power supply ripple rejection in linear regulators," *Texas Instrument Analog Design Journal*, 2005. [Online]. Available: <http://www.ti.com/lit/an/slyt202/slyt202.pdf>.
- [15] Catsoulis, John, *Designing embedded hardware*, 2nd ed. Sebastopol, CA: O'Reilly, 2005, 377 pp., ISBN: 978-0-596-00755-3.
- [16] P. Loughhead, *Altium Tutorial - Getting Started with PCB Design*. 2016. [Online]. Available: <https://www.altium.com/solution/pcb-layout-tutorial>.
- [17] B. S. Lee, *Understanding the Terms and Definitions of LDO Voltage Regulators*. 1999. [Online]. Available: <http://www.ti.com/lit/an/slva079/slva079.pdf>.
- [18] A. Syed and W. Kang, *Board level assembly and reliability considerations for QFN type packages*. Amkor Technology Inc., 2003. [Online]. Available: http://www.solder.net/images/qfn_assembly_reliability.pdf.
- [19] NXP Semiconductors, *Appl. Note 1902 Assembly guidelines for QFN (quad flat no-lead) and SON (small outline no-lead) packages*. 2018. [Online]. Available: <https://www.nxp.com/docs/en/application-note/AN1902.pdf>.
- [20] C. L. Chua, D. K. Fork, K. Van Schuylenbergh, and J.-P. Lu, "High q rf coils on silicon integrated circuits," in *MEMS Components and Applications for Industry, Automobiles, Aerospace, and Communication II*, International Society for Optics and Photonics, vol. 4981, 2003, pp. 150–156.
- [21] Nordic Semiconductor, *White Paper nWP-017 Antenna Tuning*. 2012. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nwp_017.pdf.
- [22] Nordic Semiconductor, *Quarter lambda printed monopole antenna for 2.45GHz*. 2005. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nwp_008.pdf.
- [23] NXP Semiconductors, *Appl. Note 10441 Level shifting techniques in I2C-bus design*. 2007. [Online]. Available: <https://www.nxp.com/docs/en/application-note/AN10441.pdf>.
- [24] Philips Semiconductors, *Appl. Note 97055 Bi-directional level shifter for I2C-bus and other systems*. 1997.

-
- [25] D. L. Jones, *PCB Design Tutorial*. 2004. [Online]. Available: <http://alternatezone.com/electronics/files/PCBDesignTutorialRevA.pdf>.
- [26] Jain, Suyash, *Appl. Note 098 Layout Review Techniques for Low Power RF Designs*. Texas Instruments, 2012. [Online]. Available: <http://www.ti.com/lit/an/swra367a/swra367a.pdf>.
- [27] Texas Instruments, *Appl. Note SZZA009 PCB Design Guidelines For Reduced EMI*. 1999. [Online]. Available: <http://www.ti.com/lit/an/szza009/szza009.pdf>.
- [28] T. C. Lun, *Designing for Board Level Electromagnetic Compatibility*. 2005. [Online]. Available: <https://www.nxp.com/docs/en/application-note/AN2321.pdf>.
- [29] A. Kaknevicus and A. Hoover, *Managing Inrush Current*. 2015. [Online]. Available: <http://www.ti.com/lit/an/slva670a/slva670a.pdf>.
- [30] A. Kaknevicus, *Integrated Load Switches versus Discrete MOSFETs*. 2015. [Online]. Available: <http://www.ti.com/lit/an/slva716/slva716.pdf>.
- [31] S. Limjoco, Aldrick, *Appl. Note 1144 Measuring Output Ripple and Switching Transients in Switching Regulators*. 2014. [Online]. Available: <https://bit.ly/2FERhvT>.
- [32] Agilent Technologies, *Agilent InfiniiVision 2000 X-Series Oscilloscopes*, Third edition. Malaysia: Agilent Technologies, 2011. [Online]. Available: http://www.brown.edu/Departments/Engineering/Courses/En163/2000_series_users_guide.pdf.
- [33] Barry, Richard, *Mastering the FreeRTOS Real Time Kernel*, 161204th ed. Real Time Engineers Ltd., 2016, 371 pp. [Online]. Available: https://www.freertos.org/Documentation/RTOS_book.html.
- [34] K. Lien, *Embedded utvikling på en fjernstyrt kartleggingsrobot*. 2017. [Online]. Available: <https://brage.bibsys.no/xmlui/handle/11250/2451065>.
- [35] B. B. Iversen, *Integrating a Raspberry Pi into the LEGO-robot project*. 2017.

Appendix A

Media Attachment

This appendix lists the structure and content of the companion media attachment.

A.1 Schematics and PCB Layout

Two revisions are included in the media attachment. The first revision of the control system is included as it reflects the design that has been built in this thesis. The second revision of the design includes improvements and fixes based on experiences made with the first revision. All files are located in the folder pcb, and is structured as given in Table A.1.

Table A.1: Description of the content in the pcb folder.

pcb	AD project files.
├ 3d_models	3D models common to both revisions.
├ datasheets	Component datasheets.
├ nrf_ref	nRF reference designs.
├ └ antenna	Antenna section.
├ └ dk	Development kit.
├ rev1	Design revision 1.
├ └ doc	Schematics, fabrication outputs and BOM.
└ rev2	Design revision 2.

A.2 Software

The template project with all drivers have been included. This project is based on Nordic Semiconductor’s SDK version 15.0.0, which can be downloaded directly from their website.

There was a bug in the 15.0.0 version of the SDK, which has been fixed by modifying one of the drivers Nordic Semiconductor supplies in the SDK. This file has been included in this media attachment, and should replace the original file when transferring to a new SDK. All files are located in the folder `sw`, and is structured as given in Table A.2.

Table A.2: Description of the content in the `sw` folder.

<code>sw</code>	Software folder.
├	<code>config</code> Tracealyzer and FreeRTOS config files.
├	<code>drivers</code> All drivers developed.
├	<code>pca10040</code> Segger Embedded Studio project files.
├	<code>TraceRecorder</code> Tracealyzer source files.
└	<code>include</code> Tracealyzer include files.

In order to build the project, download Nordic Semiconductor’s SDK version 15.0.0, and extract the content. Then move the `slam_application` folder into `SDK_ROOT/examples/ble_peripheral/`. The fixed library file `nrfx_ppi.c` should replace `SDK_ROOT/modules/nrfx/drivers/src/nrfx_ppi.c`.¹

A.3 Tutorials

Two tutorials are part of the media attachment. They can be found in the folder `tutorials`. The two tutorials are:

- **component_creation.pdf**: Guide to creating new components in Altium Designer.
- **ses_import.pdf**: Guide to implementing additional SDK libraries and drivers into a Segger Embedded Studio project.

¹A list over known issues in version 15.x.0 of the SDK is available at <https://devzone.nordicsemi.com/f/nordic-q-a/34155/what-are-sdk-15-x-0-known-issues>

Appendix **B**

Schematics and PCB Layout

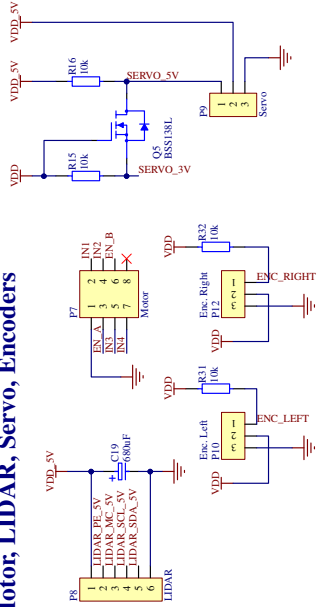
This Appendix contains electrical schematics, mechanical drawings and PCB layouts and a layer stack legend for the control system developed in this thesis. It is important to stress that this is the first revision. A second revision—as detailed in the thesis—has been made, and should be used for the next production of the control system. The schematics and drawings for the first revision are included as they reflect the system that was realized and built in this thesis. All design data for the second revision is available in the media attachment, as described in Appendix A.

Note that the electrical schematic is designed for A3 paper, but has been scaled down to B5 in order to match the page setup of the thesis. It is completely in vector format though, so on a PC it can easily be scaled up. Regardless, all the documentation is available in its original form in the media attachment.

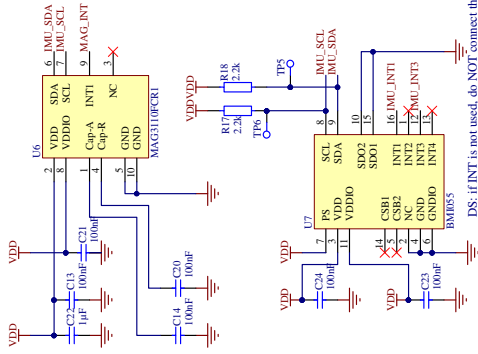
The following documents are included:

1. **Electrical Schematics**
2. **Fabrication Drawings**
3. **Layer Stack Legend**
4. **PCB 3D-Render**

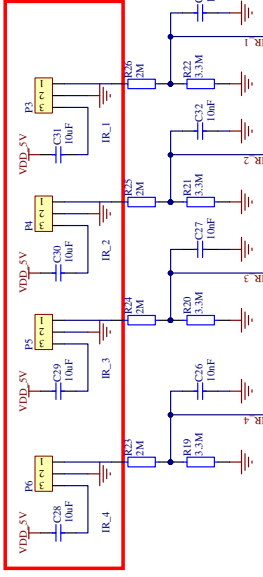
Motor, LIDAR, Servo, Encoders



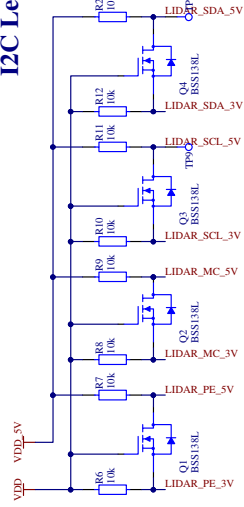
Compass, Acc, Gyro



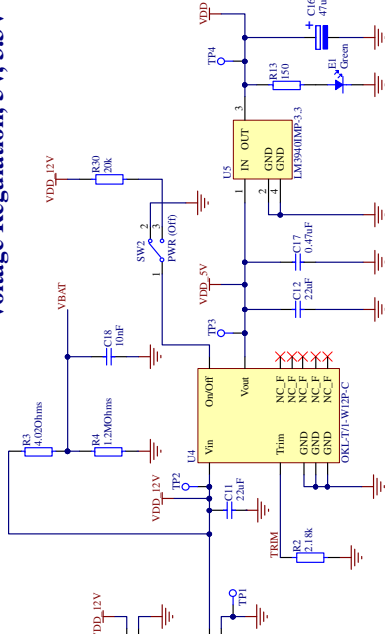
IR Connectors



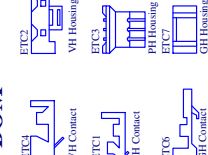
I2C Level Shifters



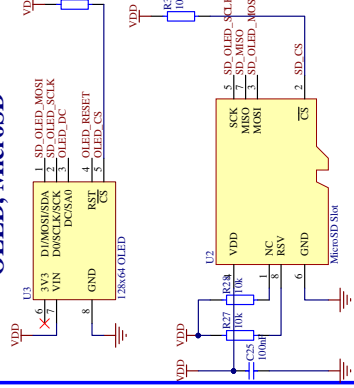
Voltage Regulation, 5V, 3.3V



BOM



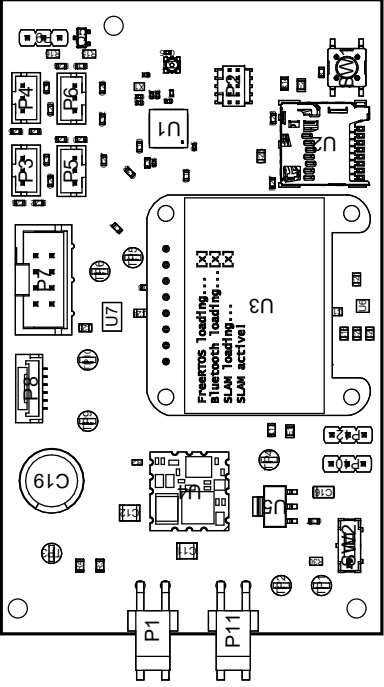
OLED, MicroSD



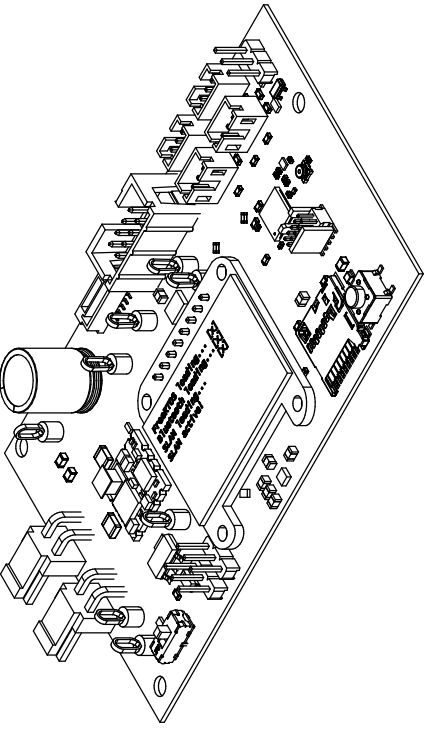
SLAM Control System (SCS)

Size	Number	Revision	I
A3	USE REVISION 2 FOR FUTURE DESIGNS!		
Date:	5/11/2018	Sheet 1 of	8
File:	C:\Users\jcs\Documents	Drawn By:	John Korman

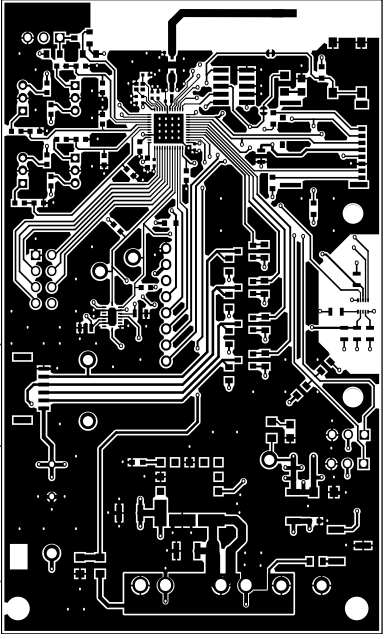
View from Top side (Scale 1)



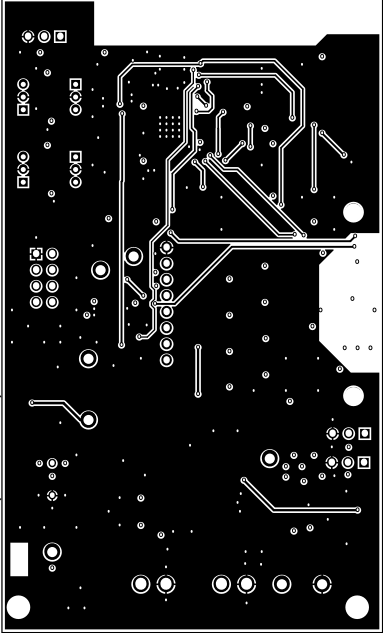
View from Front side (Scale 1)



Component Side (Scale 1:1)



Solder Side (Scale 1:1)



Layer Stack Legend

Material	Layer	Thickness	Dielectric Material	Type	Gerber
	Top Paste			Paste Mask	GTP
	Top Overlay			Legend	GTO
Surface Material	Top Solder	0.01mm	Solder Resist	Solder Mask	GTS
Copper	Component Side	0.04mm		Signal	GTL
Core		0.32mm	FR-4	Dielectric	
Copper	GND	0.04mm		Signal	G1
Prepreg		0.13mm		Dielectric	
Copper	NEW_POWER	0.04mm		Signal	G2
Core		0.25mm		Dielectric	
Copper	Solder Side	0.04mm		Signal	GBL
Surface Material	Bottom Solder	0.01mm	Solder Resist	Solder Mask	GBS
	Bottom Overlay			Legend	GBO
	Bottom Paste			Paste Mask	GBP

Total thickness: 0.86mm

