

Emanuela Tuong Vi Thi Tran

Testing and further development on the Sensor Data Acquisition Board for an nRF52840 SLAM Robot

Master's thesis in Cybernetics and Robotics

Supervisor: Tor Onshus

June 2023

Emanuela Tuong Vi Thi Tran

Testing and further development on the Sensor Data Acquisition Board for an nRF52840 SLAM Robot

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Onshus
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics



Preface

This master's thesis forms the foundation for evaluation in the 30-credit course TTK4900 - Engineering Cybernetics, Master's Thesis. It concludes a two-year master's degree program for the Department of Electronic Systems at the Norwegian University of Science and Technology (NTNU). This thesis continues a specialisation project and directly continues the previous work.

During my five years at NTNU, developing and implementing the Sensor Data Acquisition Board, the first embedded system on a large scale that I have undertaken has been a valuable and rewarding experience. This thesis has allowed me to test, debug, design, review, and create a new system, further enhancing my knowledge and skills in the field. Within the embedded systems field, my background knowledge has primarily leaned toward the hardware aspect. However, through this master's thesis, I have been able to delve into connecting hardware to software.

I want to thank my supervisor, Tor Onshus, for his unwavering availability and guidance throughout this thesis. Additionally, I thank the personnel at the mechanical workshop in the Department of Engineering Cybernetics for their assistance, equipment lending, and discussions that have helped me overcome various challenges encountered during my work.

I want to thank the students involved in the SLAM project and the students with whom I have shared the study hall. Their explanations and knowledge sharing have significantly shaped my perspective on embedded systems from a cybernetics and robotics standpoint. Their contributions and my background knowledge from the Department of Electronics Systems have made it possible to realise this master's thesis.

Lastly, I thank my family and friends for their encouragement and support throughout my five years at NTNU. Their presence and encouragement have been invaluable in my journey, and I am deeply grateful for their continuous support.

Problem Statement

In this master's thesis, a sensor data acquisition board will be developed for maze-mapping robots. The embedded system is based on Nordic Semiconductor's nRF52840 System on a Chip (SoC) and is in its prototype phase. This project involves further development and includes hardware testing, hardware implementation, software analysis, and software design. In addition to developing an embedded system for the robot's sensor tower, a cleanup and structuring of the whole project will be done with the cooperation of this year's students on the Simultaneous Localization and Mapping (SLAM) robot project.

The specific tasks outlined for the project are as follows:

- Hardware assembling and testing of the Sensor Data Acquisition Board.
- Consider a new revision of the Sensor Data Acquisition Board.
- Software analysis of the robot with a focus on the software concerning the robots servo tower.
- Software design for the robot's Infrared (IR) sensor tower.
- Project and code improvements on the SLAM robot project.

Software implementation of SLAM is not within this project's scope, nor is the production of a new revision of a Printed Circuit Board (PCB).

Summary and Conclusion

The purpose of this master thesis has been to further develop a new system, called Sensor Data Acquisition Board, such that it can be integrated on the nRF52840DK SLAM robots. In the autumn of 2022, the first prototype was designed. It features two nRF52840 SoCs where one is designated to run the task concerning the IR-sensor tower, while the second is designated to run SLAM algorithms. Design faults were discovered after the first design, but it was not concluded the need for a second revision. The objective of this thesis has been to test this new system further, review the need for a second revision of the printed circuit board, and start the software development of the system. If critical design faults were to be discovered, a second revision of the system must be started.

Extensive hardware testing has been done on the first prototype of the Sensor Data Acquisition Board. In addition to the known minor faults in the design, critical design faults were discovered. The layout design for the nRF52840 SoC was done incorrectly, such that specific pins were not accessible. The design fault concerns the two nRF52840 that are available on the board. This led to circuit faults for several critical circuit modules such as Serial Peripheral Interface (SPI), Pulse Width Modulation (PWM), Inter-Integrated Circuit (I2C) and Universal Asynchronous Receiver-Transmitter (UART). After identifying the faults in the initial design, a second revision was developed to rectify the issues.

The second revision of the system has now corrected the layout around the nRF52840. With the use of μ Vias, all pins not accessible by routing on the top layer are available on other layers. The second revision has also addressed the minor faults that were known earlier and the ones discovered during testing. A priority list was made to distribute the time used effectively.

To begin the software development, an analysis of the current software, which is relevant to this thesis, was conducted on the SLAM robots. The analysis resulted in a requirement and specifications list for the new system, and this thesis has formulated a design proposal.

In parallel, a collaborative side project aimed at improving and organizing the SLAM robot project was initiated with the involvement of the four students on the project. This thesis presents the enhancements, modifications, and contributions made by the author of this thesis to the overall robot project, addressing its poor maintainability.

This thesis concludes the first phase of a new embedded design, establishing the groundwork for the second revision of the Sensor Data Acquisition Board and initiating the software development for the new system. Integrating the new system with the current robot poses challenges since the robot project does not have the modularity needed for integrating new systems.

Sammendrag og Konklusjon

Hensikten med denne masteroppgaven har vært å videreutvikle et nytt system, kalt Sensor Data Acquisition Board, slik at det kan integreres på nRF52840DK SLAM-robotene. Høsten 2022 ble den første prototypen designet. Den har to nRF52840 SoCs der den ene er utpekt til å kjøre oppgaven vedrørende IR-sensortårnet, mens den andre er utpekt til å kjøre SLAM-algoritmer. Designfeil ble oppdaget etter det første designet, men det ble ikke konkludert med behov for en ny revisjon. Målet med denne oppgaven har vært å teste det nye systemet videre, vurdere behovet for en ny revisjon av kretskortet, og starte programvareutviklingen av systemet. Dersom kritiske designfeil skulle oppdages, må en ny revisjon av systemet startes.

Omfattende maskinvaretesting har blitt utført på den første prototypen av Sensor Data Acquisition Board. I tillegg til de kjente mindre feilene i konstruksjonen, ble det oppdaget kritiske konstruksjonsfeil. Layoutdesignet for nRF52840 SoC ble gjort feil, slik at spesifikke pinner ikke var tilgjengelige. Designfeilen gjelder de to nRF52840 som er tilgjengelig på kretskortet. Dette førte til kretsfeil for flere kritiske kretsmoduler som SPI, PWM, I2C og UART. Etter å ha identifisert feilene i det opprinnelige designet, ble det utviklet en ny revisjon for å rette opp problemene.

Den andre revisjonen av systemet har nå korrigert oppsettet rundt nRF52840. Ved bruk av μ Vias er alle pinner som ikke er tilgjengelige ved ruting på topplaget tilgjengelig på andre lag. Den andre revisjonen har også adressert de mindre feilene som var kjent tidligere og de som ble oppdaget under testing. Det ble laget en prioriteringsliste for å fordele tidsbruken effektivt.

For å starte programvareutviklingen ble det utført en analyse av gjeldende programvare, som er relevant for denne oppgaven, på SLAM-robotene. Analysen resulterte i en krav- og spesifikasjonsliste for det nye systemet, og denne oppgaven har formulert et designforslag.

Parallelt ble det satt i gang et samarbeidende sideprosjekt med sikte på å forbedre og organisere SLAM robotprosjektet med involvering av de fire studentene i prosjektet. Denne oppgaven presenterer forbedringene, modifikasjonene og bidragene fra forfatteren av denne oppgaven til det overordnede robotprosjektet, og tar for seg dets dårlige vedlikeholdsmuligheter.

Denne oppgaven avslutter den første fasen av et nytt innebygd design, og etablerer grunnlaget for den andre revisjonen av Sensor Data Acquisition Board og starter programvareutviklingen for det nye systemet. Integrering av det nye systemet med den nåværende roboten byr på utfordringer siden robotprosjektet ikke har den modulariteten som trengs for å integrere nye systemer.

Contents

Preface	i
Problem Statement	ii
Summary and Conclusion	iii
Sammendrag og Konklusjon	iv
List of Figures	xiii
List of Tables	xiv
List of Listings	xv
Acronyms	xv
1 Introduction	1
1.1 Motivation	1
1.2 Structure of the report	2
2 Background	3
2.1 SLAM robot project	3

2.2	Components and tools used in the project	5
2.2.1	nRF52840DK	5
2.2.2	L298N Motor driver	5
2.2.3	DC Motors with rotary encoders	5
2.2.4	IR-sensor tower	6
2.2.5	ICM-20948 Inertial Measurement Unit	6
2.2.6	Custom peripheral shield	6
2.2.7	J-link	6
2.2.8	SEGGER Embedded studio	7
2.2.9	nRF5 SDK	7
2.3	Robot application	8
2.4	Sensor Data Acquisition Board	11
2.4.1	Specifications of the custom PCB	14
2.5	Testing from previous work	15
3	Theory	16
3.1	Serial Peripheral Interface	16
3.1.1	SPI modes	18
3.2	PCB Fundamentals	20
3.2.1	Defining the layer stack	20
3.2.2	Blind, buried and micro via definition	21
3.2.3	PCB design: schematic	22
3.2.4	PCB design: layout	23
3.2.5	PCB design rules	24
3.3	Real-time operating systems	24
3.4	Code quality: variables	25
3.4.1	Naming conventions	25
3.4.2	Informal naming conventions	26

3.4.3	Creating short names that are readable	28
4	Hardware testing	29
4.1	Test plan	29
4.2	Soldering and programming the nRF52840	31
4.2.1	Development environment	32
4.3	Testing SPI communication	34
4.3.1	SPI example code provided by nRF5 SDK	35
4.3.2	Testing SPI with example codes	36
4.3.3	Debugging SPI communication	38
4.4	Additional errors discovered during hardware testing	43
5	Second revision of the Sensor Data Acquisition Board	45
5.1	Schematic	46
5.1.1	Add correct footprint for the P24 connector	46
5.1.2	Changed pin matching on motor driver connector	46
5.1.3	Added 2 LEDs for testing	47
5.1.4	Added informative notes	48
5.1.5	Name changes for net names	48
5.1.6	Changes for modules and descriptive names	49
5.1.7	Set to 1-based indexing	49
5.1.8	Rearrangement of modules	50
5.2	Layout	50
5.2.1	Rerouting	52
5.2.2	Correcting the pads for the nRF52840 SOC	54
5.2.3	Silk layer changes	55
6	Software analysis and design	57
6.1	Brief analysis of the robot code	58

6.1.1	Analysing the task <i>vMainSensorTowerTask</i>	58
6.1.2	Deciding between continuing development or rewriting the software	59
6.2	Requirements and specifications for the new system	60
6.2.1	Requirements	61
6.2.2	Specifications	62
6.3	Proposal of design	62
6.4	Creating a new software project	62
6.4.1	Testing the software project	63
7	Improving and structuring of the robot project	64
7.1	GitHub and GitHub Organizations	66
7.2	Structuring and sorting the code project: robot-code	71
7.2.1	Modified SDK	73
7.2.2	The final structure of the robot code	73
7.3	Documentation of the project (wiki)	75
7.4	Name conventions	76
8	Results	77
8.1	Results: Hardware testing	77
8.2	Results: The second revision of the PCB	79
8.3	Results: Software design	79
8.4	Results: Improving and structuring the robot project	80
9	Discussion	81
9.1	Hardware testing	81
9.2	The second revision of the PCB	82
9.3	Software design	83
9.4	Improving and structuring the robot project	83

10 Future work	85
10.1 Future work for the system Sensor Data Acquisition Board	85
10.2 Future work for the robot project	87
10.2.1 Documenting the work on the "wiki"	87
10.2.2 Name conventions	87
10.2.3 The use of GitHub	87
10.2.4 Modularisation	88
Bibliography	89
Appendices	91
A Schematic revision 1: main sheet	91
B Schematic revision 1: power sheet	92
C Schematic revision 1: servo_tower sheet	93
D Schematic revision 1: SLAM sheet	94
E Schematic revision 2: main sheet	95
F Schematic revision 2: power sheet	96
G Schematic revision 2: sensor_tower sheet	97
H Schematic revision 2: SLAM sheet	98
I Tutorial on creating a new software project method 1	99
J Tutorial on creating a new software project method 2	102
K Documentation (wiki)	105

List of Figures

2.1	Robot top view [1]	4
2.2	Robot bot view[1]	4
2.3	Robot side view[1]	4
2.4	Connecting a computer and the target Central Processing Unit (CPU) using regular J-Link[2]	6
2.5	Connecting a computer and the target CPU using a Jlink OB[2]	7
2.6	Interactions between the tasks concerning the robots behaviour[3, p.13]	9
2.7	Interface between mqttsn_task and tasks concerning the robot behaviour. [3, p.16]	10
2.8	System structure over Sensor Data Acquisition Board [4, p.14]	11
2.9	Custom board placed on top of the SLAM-robot: back view[4, p.37] . . .	13
2.10	Custom board placed on top of the SLAM-robot: side view [4, p.38] . . .	13
2.11	PCB front [4, p.36]	14
2.12	Test plan for hardware testing[4, p.31]	15
3.1	SPI configuration with master and slave	17
3.2	SPI transmission[5]	17
3.3	SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge[6].	19

3.4	SPI Mode 1, CPOL = 0, CPHA = 1: CLK idle state = low, data sampled on the falling edge and shifted on the rising edge [6].	19
3.5	SPI Mode 2, CPOL = 1, CPHA = 0: CLK idle state = high, data sampled on the rising edge and shifted on the falling edge [6].	19
3.6	SPI Mode 3, CPOL = 1, CPHA = 1: CLK idle state = high, data sampled on the falling edge and shifted on the rising edge [6]	20
3.7	3D view of the layer stack up on a PCB with visible vias[7]	21
3.8	The three types of vias that can be created: blind (1), buried (2) and thru-hole[7]	21
4.1	Updated test plan for hardware testing	30
4.2	Soldering of nRF52840 circuit	32
4.3	Connection between computer and Sensor Data Acquisition Board	32
4.4	Setup for debugging/programming with two nRF52840DK	33
4.5	Setup for debugging/progamming the sensor data acquisition board with a nRF52840DK	34
4.6	SPI connection between two nRF52840DKs	36
4.7	Resoldered pins on the Sensor Data Acquisition Board	39
4.8	Capturing SPI communication with a oscilloscope: two nRF52840DK . .	40
4.9	Capturing SPI communication with oscilloscope: Sensor Data Acquisition as master	40
4.10	Capturing SPI communication with oscilloscope: Sensor Data Acquisition as slave	41
4.11	Capturing attempt to set SPI pins in high state	42
4.12	SPI traces close to 5 V buck regulator	43
4.13	Close up of the layout around the nRF52840	43
4.14	Layout revision 1: Connector for motor driver	44
4.15	Different pin counting	44
4.16	Motor driver connector occupies more space	44
5.1	Schematic revision 1: Connector for motor driver	47

5.2	Schematic revision 2: Connector for motor driver	47
5.3	Schematic revision 2: The available LEDs on the PCB	48
5.4	Schematic 2nd revision: Informative note in the main sheet	48
5.5	Layout revision 2: Top layer	50
5.6	Layout revision 2: Layer 1	51
5.7	Layout revision 2: Layer 2	51
5.8	Layout Revision 2: Bot layer	52
5.9	PCB Layout revision 2: Layout around the PCB's Light Emitting Diodes (LEDs) and connector P24	52
5.10	Layout revision 2: Connector for motor driver	53
5.11	Layout revision 2: New placement for connectors concerning motor driver and IR sensors	53
5.12	Layout revision2: Routing on bot layer under the 5 V buck regulator	53
5.13	Altium Designer: Stakcup - Impedance - Via types	54
5.14	Altium Designer: Layer stack manager - Via types	54
5.15	Altium Designer: Layer stack manager - Properties of a μ via	55
5.16	Layout revision 2: nRF52840 with the use of μ Vias	55
5.17	Layout revision 2: Power circuit in 3D	56
5.18	Layout revision 2: The available LEDs on the PCB	56
5.19	Layout revision 2: Connectors for motor driver and IR sensors	56
6.1	Simplified diagram of the interactions concerning the task <i>vMainSensor-TowerTask</i>	58
6.2	Data transmission between development kit and the sensor data acquisition board, ideal scenario	59
7.1	GitHub Organization: SLAMRobotProject	68
7.2	Folder structure of the SLAM Project	68
7.3	Repositories of the GitHub Organization: SLAMRobotProject	70
7.4	README-file from the SLAM project	71

7.5	Path to the code concerning the robots nRF52840DK	72
7.6	Properties of the folder nrf5sdk_thread	72
7.7	New folder structure for the code concerning the robots nRF52840DK . .	74
7.8	Path to the code concerning the nRF52830DK robot	75
8.1	Testplan with results	78
8.2	Layout revision 2: 3D version with components	79
10.1	Jumpers/zero-ohm resistor are marked with a cross	86

List of Tables

2.1	Task overview of the software	8
3.1	SPI-mode configurations	18
3.2	Naming conventions for C	27
4.1	Equipment used for soldering	31
4.2	Component list for soldering the circuit around the nRF52840	31
4.3	Equipment used for testing	34
4.4	SPI pins on the nRF52840 SOC serving the IR-sensor tower	37
5.1	Net name changes	49
5.2	Module name changes	49
5.3	Descriptive name changes	49
7.1	New names for the different code projects	69
7.2	Size different for the code project after the clean up	71

Listings

4.1	Pin configuration for SPI	37
4.2	Setting SPI pin in high state	41

Acronyms

CAN	Controller Area Network
CPU	Central Processing Unit
DC	Direct Current
DRC	Design Rule Check
ESD	Electrostatic Discharge
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDI	High Density Interconnect
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IR	Infrared
JTAG	Joint Test Action Group
LED	Light Emitting Diode
NTNU	Norwegian University of Science and Technology
OLED	Organic Light-Emitting Diode
PCB	Printed Circuit Board
PWM	Pulse Width Modulation

RF	Radio Frequency
RTOS	Real Time Operating System
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
SMD	Surface Mount Device
SoC	System on a Chip
SPI	Serial Peripheral Interface
SWD	Serial Wire Debug
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
ZIP	Zoological Information Processing

Introduction

The SLAM robot project, arranged by the Department of Engineering Cybernetics, is an ongoing project dating back to its start in 2004. It involves using small, autonomous vehicles to map an unknown area. These robots, which are based on the nRF52840 SoC, are responsible for navigating to specific coordinates provided by the server without relying on external localization technology. They also collect and transmit measurements of their surroundings, including distances to nearby objects. By combining these measurements from multiple robots, the central server can build a detailed map of the area, which enables the robots to explore and map new environments independently.

1.1 Motivation

It is desired to separate the software on the development kit and have desired nRF52840 SoC for specific tasks. As the system has become more complex over time, dividing it into smaller parts will make it easier to iterate, debug, and further develop the system without encountering significant setbacks.

With additional hardware, the robot can access more than one SoC to perform its software. The first step towards dividing the software is to have one SoC running the software concerning the IR-sensor tower and a second SoC running the SLAM algorithms.

1.2 Structure of the report

- chapter 2 introduces the background.
- chapter 3 covers the relevant theory.
- chapter 4 covers the hardware testing of the Sensor Data Acquisition Board.
- chapter 5 covers the second revision of the printed circuit board.
- chapter 6 presents a software analysis and a proposal of the software design for the new system.
- chapter 7 covers the project and code base improvements, a collaborative work with the student on the SLAM Robot Project. The work done by the author of this thesis will be presented and discussed.
- chapter 8 presents the results.
- chapter 9 covers the discussion of the method and results.
- chapter 10 presents the suggested future work.

Chapter 2

Background

This section presents the background information for this master's thesis and provides an overview of the relevant previous work. It introduces the SLAM Robot Project, including a system overview and the supporting components and tools used in the project. Furthermore, this chapter will present the previous work on developing the new system Sensor Data Acquisition Board.

2.1 SLAM robot project

The SLAM robot project is a student project supervised by Professor Tor Onshus from the Department of Engineering Cybernetics. The students working on the project have been fifth-year students writing their project and master's thesis. Throughout the project and master's thesis, various areas have been covered. These include the construction of hardware-focused robots, developing low-level embedded software to create drivers for interfacing with hardware components, high-level embedded software for control, navigation, and communication, and implementing various server-side logic functionalities.

As briefly described in the introduction, chapter 1, the goal of the SLAM-project is to make multiple low-end robots cooperate with a local server to map out an unknown indoor area. The server application enables remote control of the robots to investigate their surrounding environment while also receiving updates on the robot's location and information about their observed local surroundings. The robots currently in use are the ones constructed by Jølgård (2020)[8], and the recent software is based on the work done by Frestad(2022) [3], which continues the work from Andersen's master thesis(2022) [1]. Figure 2.1, 2.2, and 2.3 illustrates the robot from different perspectives providing descriptions of the components in use as well as their dimensions.

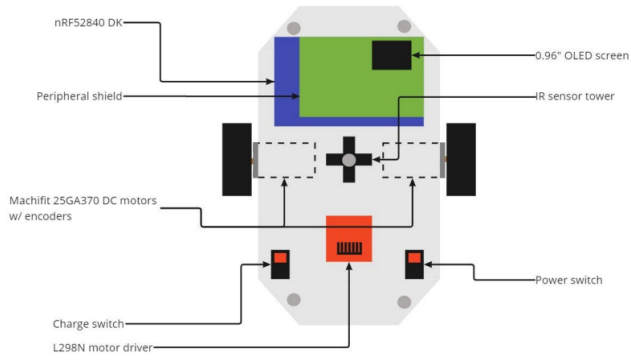


Figure 2.1: Robot top view [1]

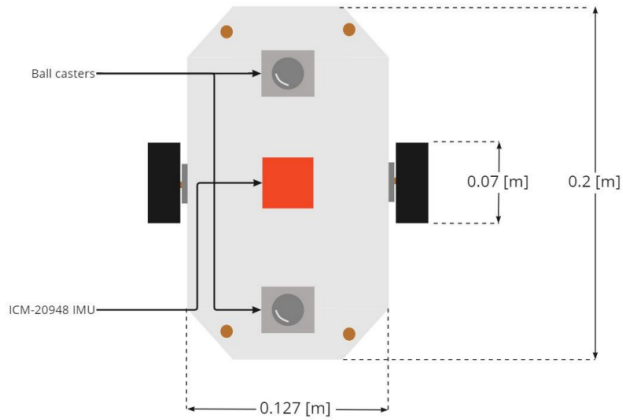


Figure 2.2: Robot bot view[1]

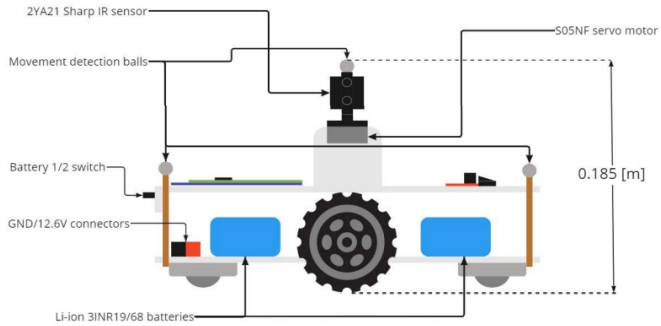


Figure 2.3: Robot side view[1]

In the previous semester (Autumn 2022), students working on the SLAM-project indicated that considerable attention should be given to enhancing code quality, software architecture, and documentation. Knowledge about the project has in the past been shared through Zoological Information Processing (ZIP)-files, resulting in the accumulation of multiple ZIP-files dating as far back as 2004. As an alternative approach to sharing software within the project organization, H. Frestad suggested in his project thesis [3] the use of GitHub as it offers valuable features for cooperative software development.

For this year's master students, four students are working on the SLAM-project with their specific areas: Marte Nordbotten Ruud-Olsen, Kristian Forsdahl, Magnus Isdal Kolbeinsen and the author of this thesis Emanuela Tuong Vi Thi Tran. The students have been encouraged to collaborate and organize the SLAM-robot project in a way that promotes a more conducive work environment, in addition to working on their respective areas of responsibility.

2.2 Components and tools used in the project

2.2.1 nRF52840DK

The SLAM-robots are all based on Nordic Semiconductor's development kit nRF52840DK board PCA10056 [9]. This development kit is utilized to develop embedded applications tailored for the nRF52840 SoC, which embeds a powerful 32-bit ARM Cortex-M4 CPU. It features an onboard Segger J-Link debugger, which allows for easy debugging and programming of the nRF52840 SoC. Further explanation on the J-link debugger can be found in section 2.2.7.

2.2.2 L298N Motor driver

The L298 motor driver, based on the L298 Dual H-Bridge Motor Driver Integrated Circuit [10], is a bi-directional motor driver. It is specifically designed for robotic applications and offers seamless integration with a microcontroller, requiring only a few control lines per motor. As per the datasheet, it can control two motors in both directions, each capable of handling up to 2 A of current.

2.2.3 DC Motors with rotary encoders

The robot uses two Machifit 12 V Direct Current (DC) motors[11], which are rated up to 100 rpm. These motors are equipped with built-in quadrature encoders, enabling wheel angle and speed measurement.

2.2.4 IR-sensor tower

The IR-sensor tower consists of four Sharp 2YA21 infrared sensors[12]. These sensors are mounted on an S05NF servo motor[13] and arranged in a radial configuration, enabling the entire set of IR-sensors to rotate. The sensor tower has a maximum rotation angle of 90°. Each of the IR-sensors has a valid measurement range of 0.1 to 0.8 meters.

2.2.5 ICM-20948 Inertial Measurement Unit

The Inertial Measurement Unit (IMU) used in the robot is the ICM-20948[14]. It comprises a 3-axis MEMS-based gyroscope, accelerometer, and compass. The IMU is positioned beneath the robot's chassis and is equipped with a digital motion processor. This processor performs basic filtering of sensor measurements and manages power consumption.

2.2.6 Custom peripheral shield

The nRF52840 shield, developed by Jølsgard[15], is specifically designed to be mounted on top of the nRF52840DK. Its primary function is to facilitate the connection of various peripherals to the General Purpose Input/Output (GPIO) headers of the development kit.

2.2.7 J-link

J-Link [16] is a family of hardware debugging tools developed by SEGGER Microcontroller. J-Link supports various target interfaces, including Joint Test Action Group (JTAG), Serial Wire Debug (SWD), and SPI, and it can be connected to a host computer via Universal Serial Bus (USB), Ethernet, or RS232 interface. J-Link provides high-speed flash programming and debugging capabilities and is suitable for professional and industrial use. Figure 2.4 illustrates how the J-link connects a computer with a target CPU.

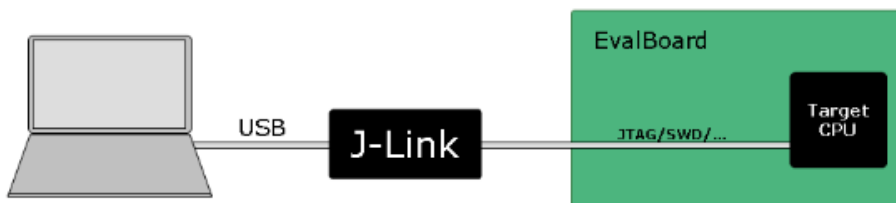


Figure 2.4: Connecting a computer and the target CPU using regular J-Link[2]

J-Link comes in various models, each offering different capabilities and features. J-Link OB [2] is a model that serves as an onboard debug interface that is incorporated into the target system. It is designed to work with microcontrollers with a built-in debug interface, like the ARM Cortex-M microcontrollers. Figure 2.5 illustrates how the computer is connected to the target CPU with the J-link OB.



Figure 2.5: Connecting a computer and the target CPU using a Jlink OB[2]

2.2.8 SEGGER Embedded studio

The project utilizes SEGGER Embedded Studio[17] as its Integrated Development Environment (IDE) to create and test software on the robot. The IDE is designed for developing embedded software on various microcontroller platforms. It provides a complete development environment, including an editor, debugger, and project management tools for various microcontrollers.

2.2.9 nRF5 SDK

The nRF5 Software Development Kit (SDK) [18] is a collection of software development tools, libraries, and example code provided by Nordic Semiconductor. The SDK is designed to help developers quickly and easily build applications based on Nordic Semiconductor's nRF5 series of SoC devices.

The provided example code demonstrates how to use the various features and capabilities of the nRF5 series devices. This code can be used as a starting point for developing custom applications. It can help developers get up to speed quickly with the various features and capabilities of the nRF5 series devices.

2.3 Robot application

The existing software application, developed for several years by multiple students, comprises a collection of seven FreeRTOS Tasks. An overview of these tasks can be found in Table 2.1. During the code implementation process, the SEGGER Embedded Studio ARM version 3.34a was used as the IDE and the supporting SDK version that was used where nRF5 SDK Thread and Zigbee version 4.1.0.

Task name	Description
MainPoseEstimateTask	Estimate position of the robot
MainPoseControllerTask	Controlling the robots behaviour
MainSpeedControllerTask	Control speed of the motors on the robot
MainSensorTowerTask	Scanning the environment around the robot
mapping_task	Extracting line segments from IR-sensor data
mqttsn_task	Execution of the MQTT-SN client
thread_stack_task	Initializing the thread interface

Table 2.1: Task overview of the software

mqttsn_task and thread_stack_task are tasks concerning mqtt and thread, while the remaining tasks concern the robot's behaviour. The interactions between the tasks concerning the robot's behaviour are illustrated in Figure 2.6.

Figure 2.7 shows the interface between the task concerning the robot behaviour and the MQTTSN task. The interface with thread_stack_task is simplified as the details do not affect the overall understanding of the robot application.

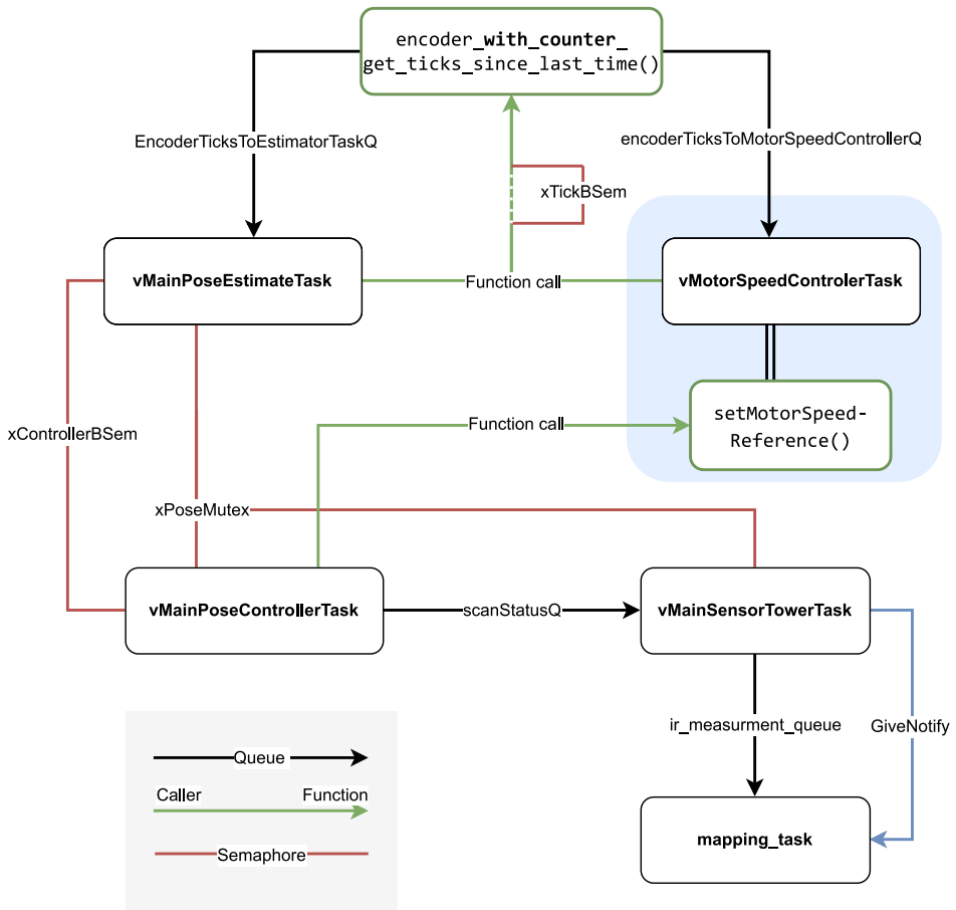


Figure 2.6: Interactions between the tasks concerning the robots behaviour[3, p.13]

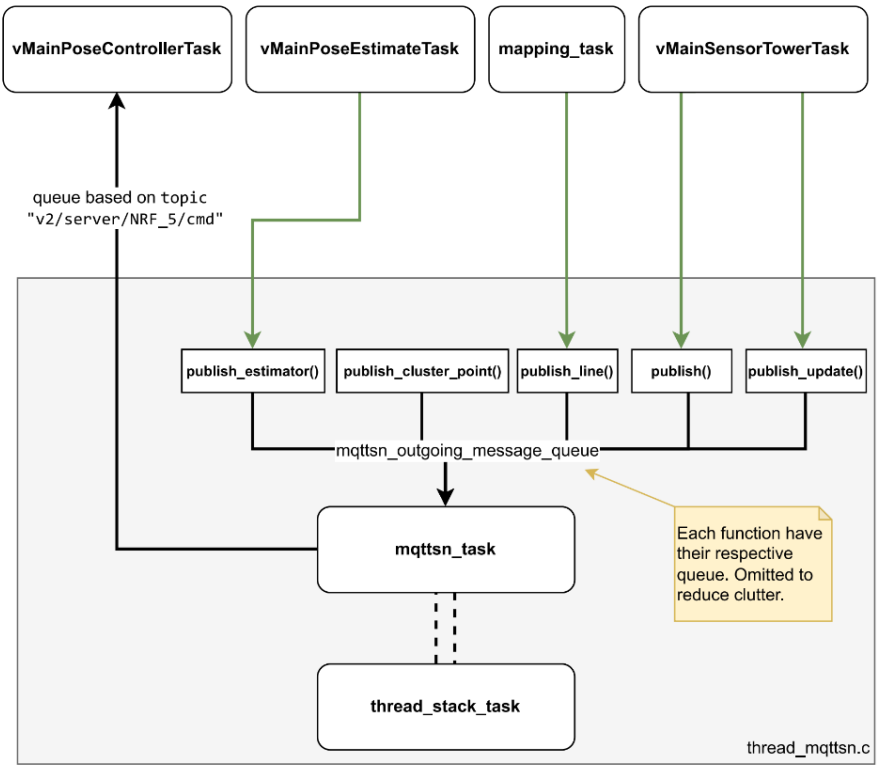


Figure 2.7: Interface between mqttsn_task and tasks concerning the robot behaviour. [3, p.16]

2.4 Sensor Data Acquisition Board

The Sensor Data Acquisition Board [4], which utilizes two nRF52840 SoCs, is a new system under development. The purpose of the new system is to collect data from a IR-sensor tower (equipped with four IR-sensors) and an IMU. It also includes a dedicated nRF52840 SoC specifically designed for SLAM algorithms. The schematics for the PCB can be found in appendices A, B, C and D.

This new system is currently in a prototype phase and the custom PCB still needs to be integrated into the robots. Further hardware testing, software development and integration remain.

Figure 2.8 visually represents the system's structure. Signals concerning the encoder, motor driver and Organic Light-Emitting Diode (OLED) will pass through the PCB and can be accessed by connectors dedicated to a nRF52840DK. The rest of the signals will pass through one of the SoCs on the board. The acquired data can either be sent directly to the SoC on the board or to another device.

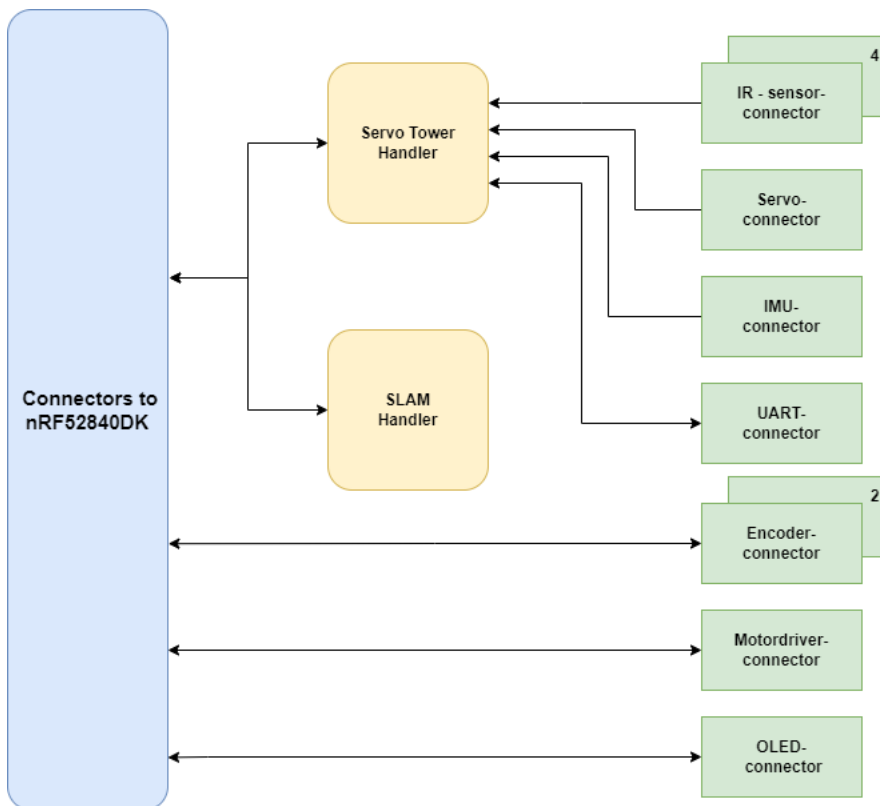


Figure 2.8: System structure over Sensor Data Acquisition Board [4, p.14]

The communication protocol used is SPI, where both SoCs on the board can serve as either:

- master - slave
- slave - master
- slave - slave (requires an additional device to serve as the master)

The following list is the description on the remaining future work described in the project thesis *Sensor Data Acquisition Board for nRF52840 maze mapping robots* [4]:

1. Most of the hardware on the system is done and software development can be further developed. Additional hardware testing is still needed but can now be tested simultaneously with new software.
2. It is important to divide the testing into different phases. First, the main goal is to have running software on one of the SoC on the board.
3. The next phase would be testing the PCB with the development kit and testing for communication. Testing the SPI communication between both SoC on the board might be better and easier as is almost little to nothing of software to handle. After figuring out the SPI setup for both SoCs, then testing with the development kit might be easier.
4. The last phase is testing the board with the robot and seeing how the software on the board cooperates with the rest of the robot.
5. During hardware testing, if more faults are found, a new revision of the PCB should be considered. Here are some fixes that should be considered if a new revision is necessary:
 - Change descriptive titles in the schematic for both SoCs.
 - Add + - signs in the silkscreen for the power connector.
 - The connector P24 must be changed to be the same pitch as the connector below on the development kit. It is also possible to physically modify the connector to connect the used pinouts to the development kit, but these are short-term modifications.
 - Consider using the 0402 packages around the nRF52840 if it is known that these components can be soldered at a manufacturer. This is to make the use of Radio Frequency (RF)-technology available for future development.
 - Consider using a solder oven than hand soldering. A stencil might be necessary to be ordered.
 - More describing text around connectors, such as around the stackable headers.

Figure 2.9 and Figure 2.10 show the custom PCB attached on top of the robots development kit from different angles.

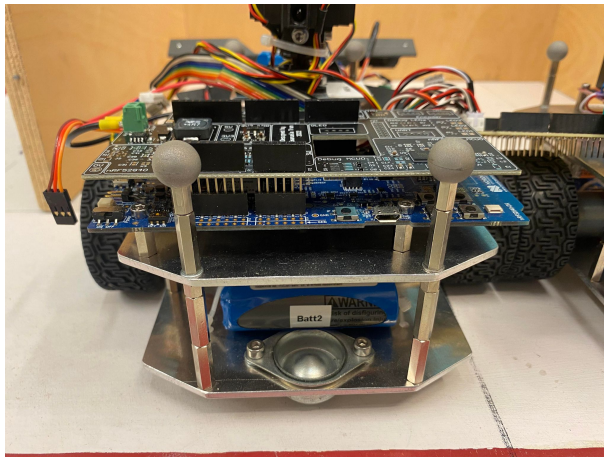


Figure 2.9: Custom board placed on top of the SLAM-robot: back view[4, p.37]

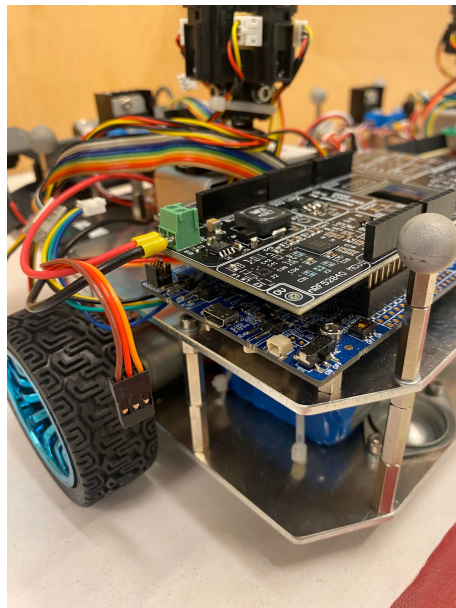


Figure 2.10: Custom board placed on top of the SLAM-robot: side view [4, p.38]

2.4.1 Specifications of the custom PCB

Listed below are the specifications of the custom PCB:

- Board size: 130mm x 67 mm.
- Four layer PCB : top, ground, power and bottom.
- 1 Oz copper thickness
- All components on the PCB are placed on one side.
- The PCB must be powered with 12 V DC to function.
- Can be put on top of an nRF52840DK and deliver power to the development kit.
- PCB can deliver both 5 V and 3.3 V
- Two nRF52840 SoCs with debug connector
- Total of 11 connectors for peripherals such as servo, motor driver, IR-sensors, IMU, encoders, OLED and UART
- Test point for 0 V, 5 V and 3.3 V
- Debug LEDs for 5 V and 3.3 V
- Indication LEDs for the robot
- Board cutout for accessing debug connectors from the nRF52840DK

Figure 2.11 illustrates the front side of the custom PCB.

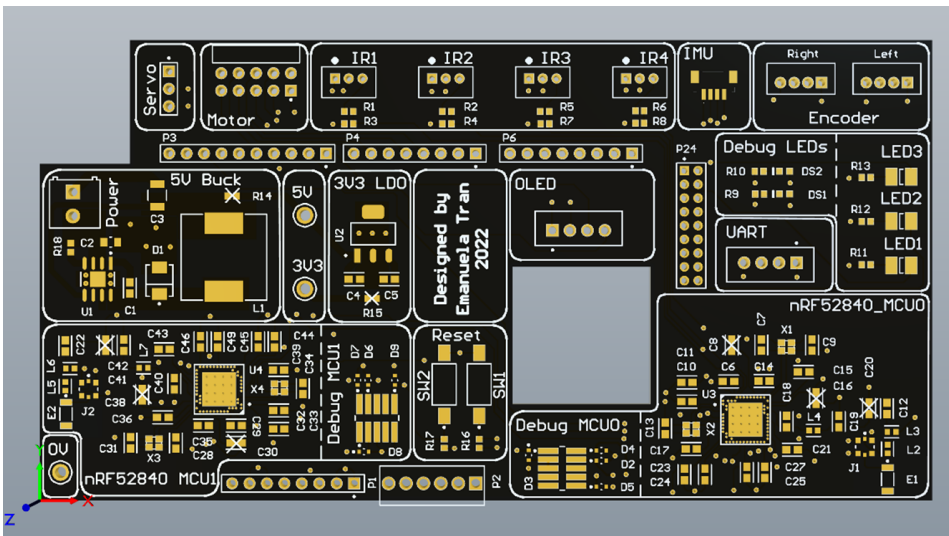


Figure 2.11: PCB front [4, p.36]

2.5 Testing from previous work

The custom board has yet to undergo complete soldering and testing. However, according to the project thesis *Sensor data acquisition board for nRF52840 maze mapping robots*[4], previous testing indicates that both regulators on the PCB are functioning correctly. Figure 2.12 shows the test plan that was made for the PCB.

What	How	Equipment	Comments	Pass/Fail
5 V buck regulator	Solder on components for the 5V buck regulator. Deliver 12 V to input. Measure stable 5 V from output. Solder on 5V jumper.	Soldering Iron, microscope, solder, flux, multimeter		
3v3 linear regulator	Solder on components for the 3v3 regulator. Deliver 12 to input. Measure stable 3.3V from output. Solder on 3V3 jumper.	Soldering Iron, microscope, solder, flux, multimeter		
MCU0	Solder on components for MCU0. Check for shortcircuit around the MCU. Connect to MCU0 with debugger.	Soldering Iron, microscope, solder, flux, multimeter, nRF debugger		
Program MCU0	Turn on one LED, set high from MCU0	computer, nRF debugger		
MCU1	Solder on components for MCU1. Check for shortcircuit around the MC1. Connect to MCU1 with debugger.	Soldering Iron, microscope, solder, flux, multimeter, nRF debugger		
Program MCU1	Turn on one LED, set high from MCU1	computer, nRF debugger		

Figure 2.12: Test plan for hardware testing[4, p.31]

Chapter 3

Theory

Within this chapter, the following sections will present the reader with relevant theories for this master thesis. These sections will cover the topics such as the communication protocol SPI, fundamental aspects of PCB-design, Real Time Operating System (RTOS), and code quality considerations.

3.1 Serial Peripheral Interface

SPI is a synchronous, full duplex master-slave-based interface. The data from the master or slave is synchronized on the rising or falling clock edge[6]. The physical connection between a master and a slave device consists of 4 wires; SCLK, CS, MOSI and MISO. The 4-wire SPI devices have four signals:

- System clock or clock (SCLK, CLK)
- Chip select (CS)
- Master out, slave in (MOSI)
- Slave out, master in (MISO)

In addition to the four wires, common ground is needed. Figure 3.1 illustrates the SPI communication between a master and a slave device.

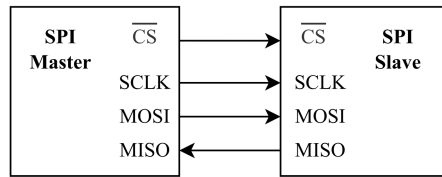


Figure 3.1: SPI configuration with master and slave

With the SPI interface, it is possible to have one master device and one or multiple slave devices. The chip-select signal from the master device is used to select the slave device. The signal is usually an active low signal and is pulled high to disconnect the slave device from the SPI bus. When multiple slave devices are used, an individual chip-select signal for each slave is required from the master.

Data transmitted between the master and slave device is synchronized to the clock generated by the master device and the data lines are called MOSI and MISO. MOSI transmits data from the master to the slave and MISO transmits data from the slave to the master.

Both masters and slaves contain a serial shift register. The master begins a byte transfer by writing the data to its SPI shift register. As the register transmits the byte to the slave on the MOSI signal line, the slave transfers the contents of its shift register back to the master on the MISO signal line [5]. In this way, the contents of the two shift registers are exchanged. Both a write and a read operation are performed with the slave simultaneously. Figure 3.2 illustrates the SPI transmission between a master and a slave with its SPI shift register.

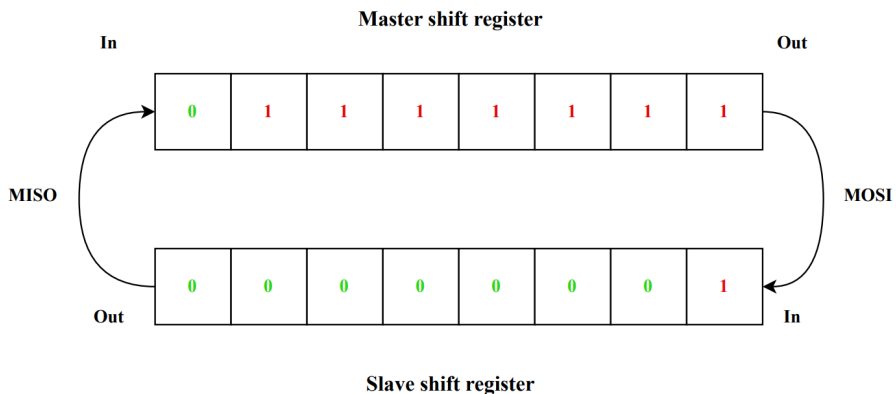


Figure 3.2: SPI transmission[5]

3.1.1 SPI modes

The communication protocol has four modes which differ by what logic level signals the start of a transaction and on which clock edge the data should be sampled. The master device can select the clock polarity and clock phase.

The CPOL bit sets the polarity of the clock signal during the idle state. The idle state is defined as the period when CS is high and transitions to low at the start of the transmission and when CS is low and transitioning to high at the end of the transmission.

The CPHA bit selects the clock phase. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data. The master device must select the clock polarity and clock phase, as per the requirement of the slave device. The four modes are shown in table Table 3.1.

SPI mode	CPOL	CPHA	Clock polarity in idle state	Clock phase used to sample and/or shift the data
SPI0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge.
SPI1	0	1	Logic low	Data samples on the falling edge and shifted out on the rising edge.
SPI2	1	0	Logic high	Data sampled on the rising edge and shifted out on the falling edge.
SPI3	1	1	Logic high	Data sampled on the falling edge and shifted out on the rising edge.

Table 3.1: SPI-mode configurations

An example of SPI communication with the four SPI modes can be seen in Figure 3.3 through Figure 3.6. The data is shown on the MOSI and MISO lines in the example. The dotted green line indicates the start and end of transmission, the sampling edge is indicated in orange, and the shifting edge is indicated in blue.

Figure 3.3 shows the timing diagram for SPI mode 0. In this mode, the clock polarity is 0, which indicates that the idle state of the clock signal is low. The clock phase is 0, which indicates that the data is sampled on the rising edge (shown by the orange dotted line), and the data is shifted on the falling edge (shown by the dotted blue line) of the clock signal.

Figure 3.4 shows the timing diagram for SPI Mode 1. In this mode, the clock polarity is 0, which indicates that the idle state of the clock signal is low. The clock phase in this mode is 1, indicating that the data is sampled on the falling edge and shifted on the rising edge of the clock signal.

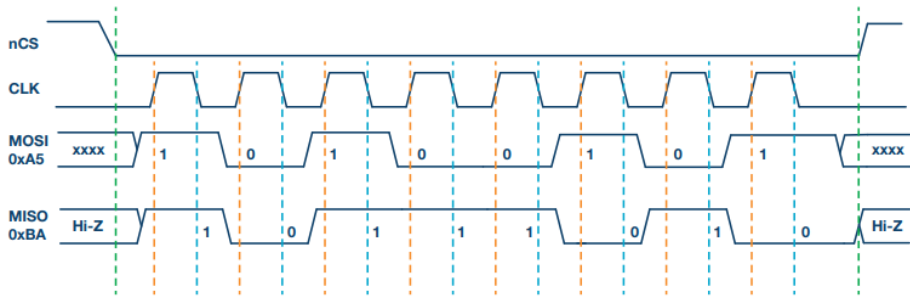


Figure 3.3: SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge[6].

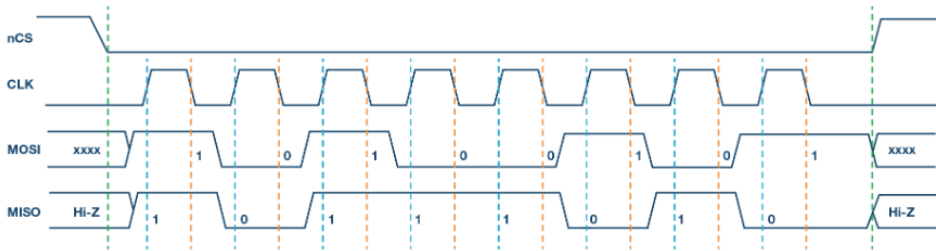


Figure 3.4: SPI Mode 1, CPOL = 0, CPHA = 1: CLK idle state = low, data sampled on the falling edge and shifted on the rising edge [6].

Figure 3.5 shows the timing diagram for SPI Mode 2. In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 0, which indicates that the data is sampled on the rising edge and the data is shifted on the falling edge of the clock signal.

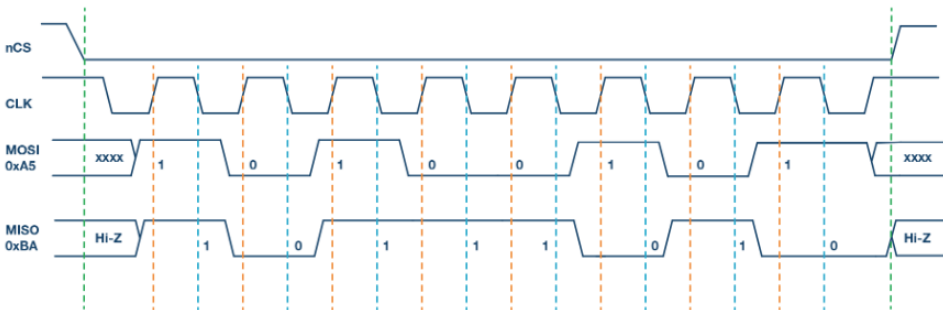


Figure 3.5: SPI Mode 2, CPOL = 1, CPHA = 0: CLK idle state = high, data sampled on the rising edge and shifted on the falling edge [6].

Figure 3.6 shows the timing diagram for SPI Mode 3. In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 1, indicating that the data is sampled on the falling edge and shifted on the rising edge of the clock signal.

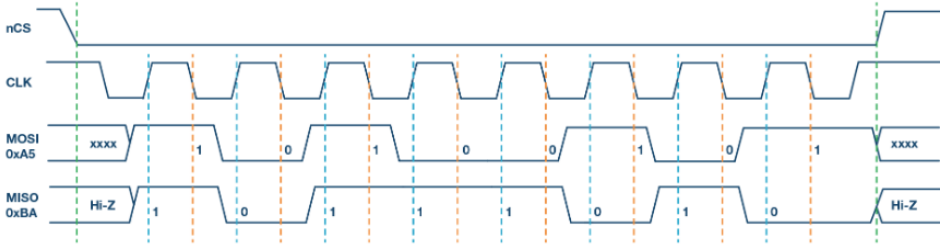


Figure 3.6: SPI Mode 3, CPOL = 1, CPHA = 1: CLK idle state = high, data sampled on the falling edge and shifted on the rising edge [6]

3.2 PCB Fundamentals

PCBs are an essential part of modern electronics. They are used to connect and mount electronic components to create functional circuits. PCBs are epoxy-bonded fiberglass sheets plated with copper. The copper plating is etched away, leaving tracks (traces) that form the interconnections of the circuit [5].

To ensure effective circuit operation, it is essential to appropriately design the conductive pathways on a PCB. Typically, engineers use specialized software to create a schematic representation of the circuit, which is then converted into a physical layout of the PCB. In order to guarantee the circuit's proper operation, the layout design is crucial. After the design is completed, components can be mounted on the PCB using techniques such as through-hole mounting or surface mounting. Once the components are mounted, the PCB undergoes testing to ensure that it functions properly and meets the design specifications.

This section will cover the fundamentals a PCB, emphasizing the elements related to the second revision of the custom PCB for this master's thesis.

3.2.1 Defining the layer stack

The PCB is designed and formed as a stack of layers. PCBs can either be single-sided (one layer), double-sided (two layers), or 4-layered, 6-layered, 8-layers, or more. The layer stack defines how the layers are arranged in the vertical direction.

3.2.2 Blind, buried and micro via definition

Vias are used to span or connect between the copper layers. If the via passes from the top surface of the board to the bottom surface, it is called a through hole via. It is possible to create vias that span other layers by creating the via at a specific point during fabrication. These types of vias fall into these groups: blind, buried, and micro vias (μ Vias). Figure 3.7 illustrates a 3D view of the layer stack up on a printed circuit board where the vias are visible.

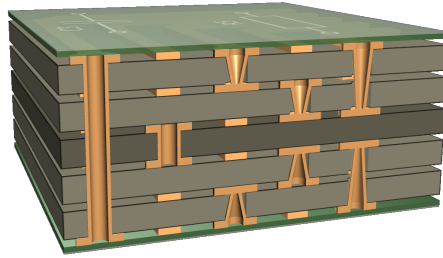


Figure 3.7: 3D view of the layer stack up on a PCB with visible vias[7]

The possible layers a via can span depend on the fabrication technology used to fabricate the board. Figure 3.8 shows a six-layer board with the layer names on the left side. The figure illustrates the three types of vias: blind, buried and through hole.

Blind vias - via spans from a surface layer to an inner layer.

Buried vias - via spans from one internal layer to another internal layer.

Through hole vias - vias spans through the whole PCB.

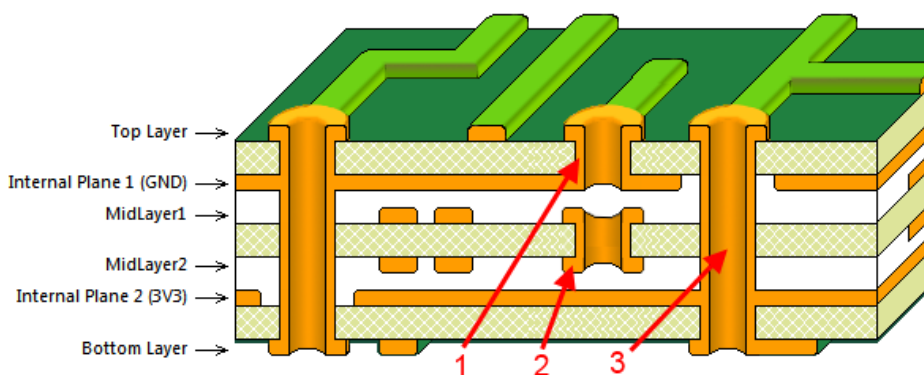


Figure 3.8: The three types of vias that can be created: blind (1), buried (2) and thru-hole[7]

The fourth type of via that can be created is called micro via or μ Via. They are used as the interconnects between layers in High Density Interconnect (HDI) designs. The μ Vias accommodate the high input/output(I/O) density of advanced components packaged and board designs.

3.2.3 PCB design: schematic

A schematic is utilized as a graphical representation of a circuit design to specify the electrical connections and components of the circuit.

Creating a clear schematic provides many benefits, especially for systems, projects, and teams. Clear schematics provide a concise and standardized way to communicate the details of a circuit design, making it easier for different team members to understand and work with the design. In addition to improved communication, a well-designed schematic reduces the risk of errors during the assembly process, which can save time and money by minimizing the rework and troubleshooting. Listed below are some other benefits of having a clear and neat schematic:

- **Easier debugging and modification** - Can help isolate circuit design issues, making it easier to debug and modify as needed.
- **Better documentation** - This can be helpful documentation for the design, making it easier to review the design later or share it with others needing to work with the circuit.
- **Improved quality control** - Can help ensure that the final product meets the design specifications and requirements, improving the overall quality of the circuit
- **Faster development** - Can help to speed up the development process by reducing the time required for design, prototyping, and testing.

Overall, creating a clear schematic is an essential aspect of circuit design and can provide significant benefits in terms of communication, quality, control and efficiency.

Guidelines for developing clear schematic:

- Start with a clear understanding of the circuit requirements and specifications before beginning the schematic design. This will ensure that the final schematic meets the desired functionality and performance.
- To make the circuit easier to follow, it is important to have a clear and consistent layout for the schematic. Organizing it logically will also help with readability.
- Use standard symbols and conventions, which will help to make the schematic more universally understandable to others.

- Label all components, connections, and nodes clearly to minimize confusion and errors during the assembly process.
- Use colors and shading to differentiate between different circuit parts, such as power supplies, input/output, and signal paths.
- Consider using hierarchical design techniques, which allow for the separation of the circuit into smaller, more manageable blocks. This approach can make it easier to debug and modify the circuit later on.

Developing a good schematic involves careful planning, attention to detail, and a focus on clear and consistent design practices. By following these guidelines, engineers can create schematics that are easy to read, understand, and build.

3.2.4 PCB design: layout

A PCB layout refers to the physical arrangement and placement of electronic components, traces and other elements on a PCB. Layout design involves designing the specific arrangement and interconnection of components and traces to ensure proper functionality and optimal performance of the electronic circuits.

The PCB layout is crucial in determining the electrical and mechanical integrity of the circuit design. It considers factors such as component placement, routing of electrical connections (traces), power and ground planes, signal integrity, thermal management, and manufacturability.

Guidelines for developing good layout

- **Component placement:** Carefully place components on the PCB, considering factors such as functionality, signal integrity, thermal management, and ease of assembly. Group related components together and allocate sufficient space between them.
- **Signal integrity:** Pay attention to signal paths and minimize trace length to reduce signal degradation. Avoid crossing signal traces, and use proper ground and power planes.
- **Routing:** Rout traces in a logical and organized manner. Use appropriate trace width and layer stack up to handle current carrying capacity and signal integrity requirements.
- **Design for manufacturability:** Ensure ease of PCB fabrication and assembly. Consider manufacturing constraints such as minimum trace widths, minimum clearances, and component footprints recommended by the manufacturer.

- **Testability:** Incorporating features and test points in the design to facilitate testing, inspection, and debugging during the manufacturing and assembly stages.
- **Documentation:** Maintain clear and detailed documentation of the PCB layout, including component placement, trace routing, layer stack up, and design constraints. This aids in future revisions, troubleshooting, and collaboration with other team members.

It is worth noting that PCB design can be complex and it is often helpful to use specialized PCB design software that provides Design Rule Check (DRC). DRCs are explained in section 3.2.5.

3.2.5 PCB design rules

Most PCB editor uses the concept of Design Rules to define the requirements of a design. These rules form an "instruction set" for the PCB editor to follow. The design rules can include minimum trace widths, minimum clearance between traces, minimum drill size, and other design specifications.

DRCs, in most PCB editors, can run a batch test at any time to generate DRC reports. The purpose of the DRC is to identify potential design errors or violations of design rules before the PCB is manufactured to avoid costly rework of manufacturing issues.

3.3 Real-time operating systems

A RTOS is an operating system designed for applications that require timing and deterministic behaviour. These applications often involve real-time control of physical systems, such as industrial automation, aerospace and automotive systems.

An RTOS provides services that allow applications to run predictably and efficiently on the underlying hardware. Listed below are some key features of an RTOS:

- Task scheduling
- Interrupt handling
- Inter-process communications
- Memory management
- Timing and synchronization

The advantage of an RTOS is its ability to provide deterministic behaviour, which means that the timing and execution of tasks can be guaranteed. Even the slightest delay or missed deadline can lead to system failure or safety risks in real-time systems. Therefore, it is crucial to prioritize timeliness and accuracy.

3.4 Code quality: variables

Good variable names are a vital element of program readability. Names should be as specific as possible. Names that are vague or general enough to be used for multiple purposes are usually bad names. Regardless of the kind of project one works on, one should adopt a variable naming convention. The convention one chooses to adopt depends on the size of the program and the number of people involved in working on it. In this section, guidelines on naming conventions will be introduced.

3.4.1 Naming conventions

Naming conventions play a considerable part in code quality as they structure the code. Any conventions at all are often better than no conventions. The power of naming conventions does not come from the specific convention chosen but rather the fact that a convention exists [19].

Conventions offer several benefits:

- **Transferring knowledge across the project:** The similarity in names gives a more accessible and more confident understanding of what unknown variables should do.
- **Learning code more quickly on a new project:** Rather than learning that several person types of code look like this and that, one can work with a more consistent set of code.
- **Reducing the amount of name proliferation:** The same thing can be called without name conventions by two different names. For example, one might call total points both *pointTotal* and *totalPoints*. This might not be confusing for the one writing the code, but it can be enormously confusing to a new programmer who reads it later.
- **Emphasize relationships among related items:** Names like *address*, *phone*, and *name* does not indicate that the variables are related. Suppose that one decides that employee-data variables should begin with an *Employee* prefix, *employeeAddress*, *employeePhone* and *employeeName* leave no doubt that the variables are related.

Listed below are the cases where conventions are worthwhile:

- When multiple programmers are working on a project.
- When one plans to turn a program over to another programmer for modifications and maintenance.
- When other programmers are reviewing own code
- Then the program will be long-lived enough that one might put it aside for a few weeks or months before working on it again.
- When one has a lot of unusual terms that are common on a project and want to have a standard term or abbreviations to use in coding.

3.4.2 Informal naming conventions

The language used to program the nRF52840DK on the robots is C. The guidelines for informal naming conventions will focus on the programming language C.

Several naming conventions apply specifically to the C programming language, such as [19]:

- *c* and *ch* are character variables
- *i* and *j* are integer indexes
- *n* is a number of something
- *p* is a pointer
- *s* is a string
- Variables and routine names are in *all_lowercase*
- The underscore (`_`) is used as a separator: *letters_in_lowercase* is more readable than *lettersinlowercase*

The naming conventions for C, which were adapted from the earlier guidelines, are provided in Table 3.2. These specifications are not necessarily recommended but give an idea of what an informal naming convention includes.

Entity	Description
<i>TypeName</i>	Type definitions use mixed uppercase and lowercase with an initial capital letter.
<i>GlobalRoutineName()</i>	Public routines are in mixed uppercase and lowercase.
<i>f_FileRoutineName()</i>	Routines that are private to a single module(file) are prefixed with an <i>f_</i> .
<i>LocalVariable</i>	Local variables are in mixed uppercase and lowercase. The name should refer to whatever the variable represents.
<i>RoutineParameter</i>	Routine parameters are formatted the same as local variables.
<i>f_FileStaticVariable</i>	Module (file) variables are prefixed with an <i>f_</i> .
<i>G_Global_GlobalVariable</i>	Global variables are prefixed with a <i>G_</i> and a mnemonic of the module (file) that defines the variable in all uppercase—for example, <i>SCREEN_Dimensions</i> .
<i>LOCAL_CONSTANT</i>	Named constants that are private to a single routine or module (file) are in all uppercase—for example, <i>ROWS_MAX</i> .
<i>G_GLOBALCONSTANT</i>	Global named constants are in all uppercase and are prefixed with <i>G_</i> and a mnemonic of the module (file) that defines the name constant in all uppercase—for example, <i>G_SCREEN_ROWS_MAX</i> .
<i>LOCALMACRO()</i>	Macro definitions that are private to a single routing or module (file) are in all uppercase.
<i>G_GLOBAL_MACRO()</i>	Global macro definitions are in all uppercase and are prefixed with <i>G_</i> and a mnemonic of the module(file) that defines the macro in all uppercase—for example, <i>G_SCREEN_LOCATION()</i> .

Table 3.2: Naming conventions for C

3.4.3 Creating short names that are readable

It is essential to prioritize readability when choosing names for code, as it is read more often than it is written. Here are some general guidelines for abbreviations. Some of them contract others, so do not try to use them all at the same time[19]:

- Use standard abbreviations (the ones in common use, which are listed in a dictionary)
- Remove all non-leading vowels. (*computer* becomes *cmptr*, *screen* becomes *scrn*, and *integer* becomes *intgr*.)
- Remove articles: *and*, *or*, *the*, and so on.
- Use the first letter and the last letters of each word.
- Truncate consistently after the first, second, or third (whichever is appropriate) letter of each word
- Keep the first and last letter of each word.
- Use every significant word in the name, up to three words.
- To clarify, try removing any unnecessary suffixes such as *-ing*, *-ed*, etc.
- Keep the most noticeable sound in each syllable.
- Be sure not to change the meaning of the variable.
- Iterate through these techniques until every variable name is shortened to between 8 and 20 characters or the maximum number of characters allowed by the programming language used.

Chapter 4

Hardware testing

This chapter will explore the method used for further testing on the Sensor Data Acquisition Board. As this thesis is a direct continuation of E. Tran's project thesis [4], the testing will continue where it was left. Testing that involves software will be limited as the focus is testing hardware circuits.

4.1 Test plan

Based on the test plan and the conclusions drawn by E. Tran in her project thesis, the conclusion was drawn that both regulators on the custom PCB are operational and function as intended. However, verifying the condition of the PCB is a good practice based on the previous work and test results before proceeding with further testing.

A test was performed on the Sensor Data Acquisition Board using the test plan displayed in Figure 2.12. The latest test results are available in Figure 4.1. The 5 V buck and 3.3 V linear regulators have passed the tests successfully. Green indicates pass, while red indicates red. The test plan has also been updated to include more descriptions and further tests for testing communication.

Chapter 4. Hardware testing

What	How	Equipment	Comments	Pass/Fail
5 V buck regulator	Solder on components for the 5V buck regulator. Deliver 12 V to input. Measure stable 5 V from output. Solder on 5V jumper.	Soldering Iron, microscope, solder, flux, multimeter		
3v3 linear regulator	Solder on components for the 3v3 regulator. Deliver 12 to input. Measure stable 3.3V from output. Solder on 3V3 jumper.	Soldering Iron, microscope, solder, flux, multimeter		
MCU0	1. Solder on components necessary for MCU0 to function. <i>Optional : circuit concerning 2nd XTAL and RF.</i> 2. Check for shortcircuit around the MCU. 3. Connect to MCU0 with debugger.	Soldering Iron, microscope, solder, flux, multimeter, nRF debugger		
Program MCU0	Set a pin high and measure.	computer, usbA to microUSB cable, nRF52840DK(debugger), 10pin cable from debugger to debug connector, multimeter		
Communication (SPI)	Write code for SPI. Set MCU0 as master. Test SPI to a device.	Computer, Oscilloscope, usbA to microUSB cable, nRF52840DK (debugger), 10pin cable from debugger to debug connector		
MCU1	1. Solder on components necessary for MCU1 to function. <i>Optional : circuit concerning 2nd XTAL and RF.</i> 2. Check for shortcircuit around the MCU1. 3. Connect to MCU1 with debugger.	Soldering Iron, microscope, solder, flux, multimeter, nRF debugger		
Program MCU1	Set a pin high and measure.	computer, usbA to microUSB cable, nRF52840DK(debugger), 10pin cable from debugger to debug connector, multimeter		
Communication (SPI) between both SoC on board	Write code for SPI. Set MCU0 as master and MCU1 slave.	Computer, Oscilloscope, usbA to microUSB cable, nRF52840DK (debugger), 10pin cable from debugger to debug connector		

Figure 4.1: Updated test plan for hardware testing

4.2 Soldering and programming the nRF52840

After confirming that the board's regulators were operating correctly, the subsequent step in the testing process was to solder around the MCU0. Certain components surrounding the nRF52840 had already been soldered on from previous work. However, the focus was on soldering only the essential components to validate that the SoC was functioning as planned. As a result, the circuit associated with the antenna and additional crystal was not soldered on. The SoC do not need to rely on these two circuits to function or be programmed. Table 4.1 shows the equipment used for soldering.

Equipment	Function
Solder	Metal to connect components to the copper on the PCB
Solder iron	Melt the solder
Microscope	Solder Surface Mount Device (SMD) components as they are small
Tweezer	Hold components in place
Fume extractor	Extract the fumes that are dangerous and toxic while soldering
Flux	Prepare the metal surfaces for soldering by cleaning and removing any oxides and impurities
ESD - band	Disperse static electricity generated from a person safely to ground
Schematics and layout of the PCB	Reference when soldering components

Table 4.1: Equipment used for soldering

Table 4.2 lists the components that were soldered and Figure 4.2 shows the soldering done to the nRF52840_MCU0 circuit. The soldering took place at the Department of Engineering Cybernetics' mechanical workshop.

Component number	Type	Value
C6	Capacitor	1.0 μ F 10% X7R
C7	Capacitor	12 pf 2% NP0
C9	Capacitor	12 pf 2% NP0
C11	Capacitor	1.0 μ F 10% X7R
X1	Crystal	32 MHz
Debug MCU0	Header	10 pin

Table 4.2: Component list for soldering the circuit around the nRF52840

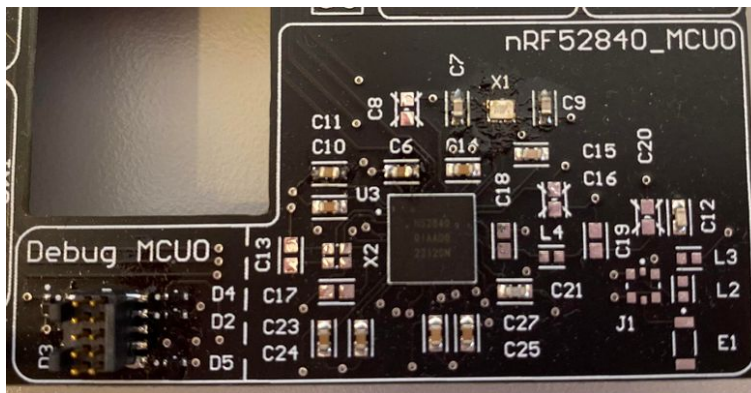


Figure 4.2: Soldering of nRF52840 circuit

4.2.1 Development environment

This section describes preparing and configuring the development environment to test the Sensor Data Acquisition Board. For testing the custom PCB the IDE used was SEGGER Embedded Studio. To flash the software onto the hardware, the nRF52840DK was used as it has an onboard debugger. Figure 4.3 illustrates the connection between a computer and the Sensor Data Acquisition Board.

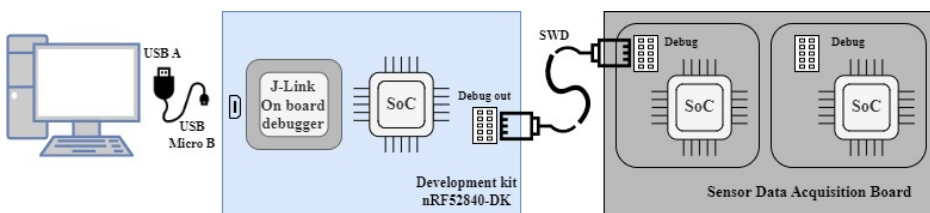


Figure 4.3: Connection between computer and Sensor Data Acquisition Board

To use the nRF52840DK as a debugger, the external SoC must be connected to the debug-out connector on the development kit. Once the external SoC is detected, the debugger will automatically switch the connection and ignore the onboard SoC. It is not possible to see this "switch" visually on some Graphical User Interface (GUI) and to assure that this setup was done correctly, it was first tested against two nRF52840DKs.

One of the development kits was used as the debugger and the other was used as the external device/SoC. The nRF52840DK has different switches on the development kit for different modes. As the nRF52840DK, used as the non-debugger, also has an onboard debugger, one must set the development kit in a mode where the onboard debugger is deactivated.

The development kit utilized as the debugger will now be referred to as nRF52840DK(1), while the one used as the external device will be referred to as nRF52840DK(2). Both development kit does not share the same power source.

Below are the settings for the two development kits:

- Powering the nRF52840DK(1) with μ USB from a computer.
- Powering the nRF52840DK(2) with μ USB from a computer.
- 10 pin cable from debug out on nRF52840DK(1) to debug in on nRF52840DK(2).
- SW8 is switched to ON for both development kits.
- SW6 is switched to DEFAULT for nRF52840DK(1).
- SW6 is switched on to nRF ONLY for nRF52840DK(2).
- SW9 is switched to VDD for both development kits.

Figure 4.4 shows the setup for two nRF52840DK.

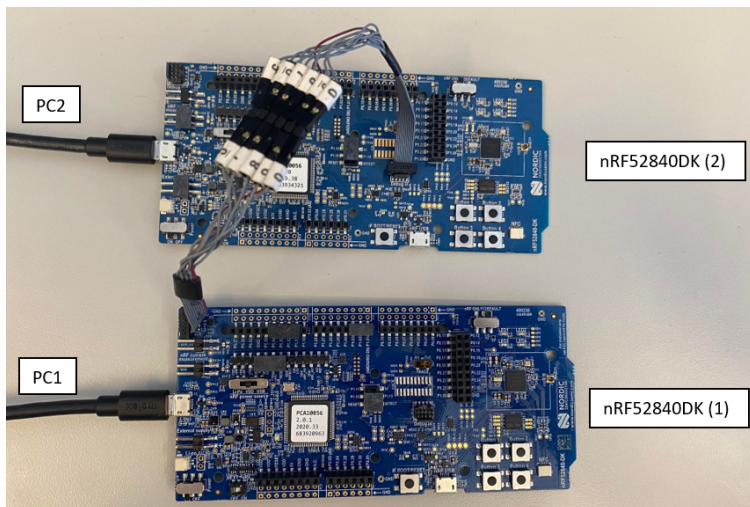


Figure 4.4: Setup for debugging/programming with two nRF52840DK

The nRF52840DK has four LEDs available on the board, and one of the LEDs on the nRF52840DK(2) was used for visual testing. The nRF52840DK(1) programmed a blinking LED on the nRF52840DK(2).

Once it was confirmed that the setup for utilizing the onboard debugger on the development kit was accurate, the next step involved testing with the custom PCB. Table 4.3

shows the equipment used when testing the PCB and Figure 4.5 shows the setup between the custom PCB and a nRF52840DK.

Equipment	Function
Power supply	Power source
Test lead set	Connect equipment to device
Wires	Connect the equipment to device
Multimeter	Electrical verification
Oscilloscope	Measure and visualize electrical signals
Electrostatic Discharge (ESD) - band	Prevent ESD damage
Computer	Writing code and debugging
nRF52840-DK	Development kit with debugger

Table 4.3: Equipment used for testing

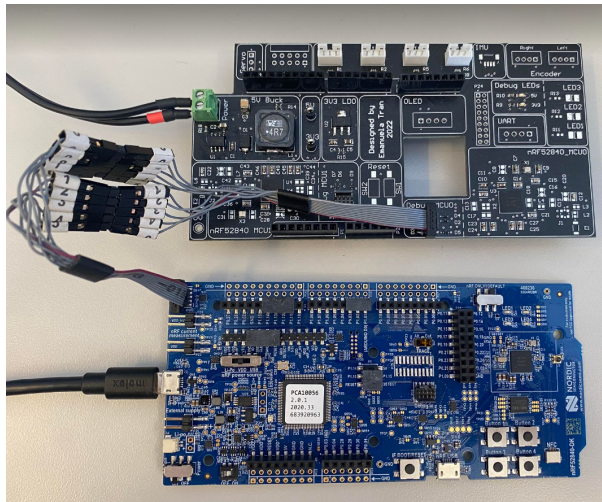


Figure 4.5: Setup for debugging/programming the sensor data acquisition board with a nRF52840DK

4.3 Testing SPI communication

To verify the functionality of SPI communication on the Sensor Data Acquisition board, testing was done with an nRF52840DK. Nordic Semiconductor’s nRF5 SDK provides example codes for running the nRF52840 as both a master and a slave device. The details of these example codes are explained in section 4.3.1.

By utilizing the provided example code, the process of verifying the SPI communication becomes straightforward and effortless. However, in the occasion where the test results

are not expected, further debugging and testing are required using measurement tools. This section will cover the method used for testing the SPI communication on the Sensor Data Acquisition Board.

4.3.1 SPI example code provided by nRF5 SDK

The source code and the project file for the SPI example codes can be found in the following folder of the nRF5 SDK:

```
<InstallFolder>\examples\peripheral
```

The example code used for the master device is called *spi*, and for the slave device, it is called *spis*.

SPI Master example code

The SPI Master Example [20] demonstrates using the SPIM peripheral. It uses the SPI master driver. The application executes SPI transactions every 200 ms and toggles the LED when the transfer is completed.

Testing the SPI Master example application by performing the following steps:

1. Compile and program the application.
2. Observe that the LED is toggled every 200 ms.

SPI Slave example code

The SPI Slave Example [21] demonstrates using the SPIS peripheral. In the main loop, the application prepares the SPIS for a transfer and waits until the transfer is completed. Every time a transfer is completed, the LED is toggled.

Hardware configurations must be done to enable the SPI slave device. The following configurations are listed below:

- Slave SCLK pin must be connected to Master SCLK.
- Slave CS pin must be connected to Master CS.
- Slave MOSI pin must be connected to Master MOSI.
- Slave MISO pin must be connected to Master MISO.

Figure 4.6 shows the physical SPI connection between two development kits.

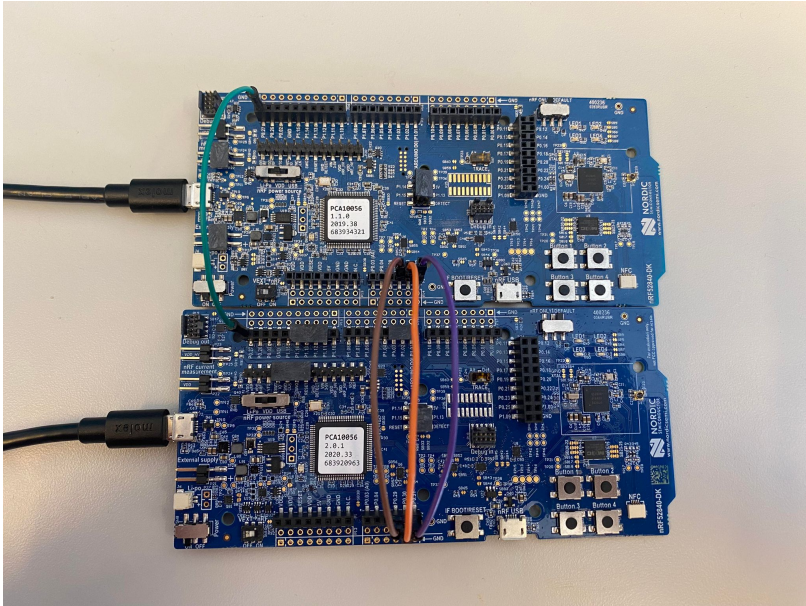


Figure 4.6: SPI connection between two nRF52840DKs

The default pin assignment for the SCLK, MOSI and MISO lines can be found in the file *sdk_config.h* whereas the CS pin can be found in the file *main.c*.

Testing the SPI Slave Example application by performing the following steps:

1. Compile and program the application.
2. Observe that no LED is toggled.
3. Connect the board to another board that runs the SPI Master Example.
4. Observe that the LED is toggled every 200 ms.

4.3.2 Testing SPI with example codes

For testing the SPI communication with the example codes provided by nRF5 SDK, two tests were conducted:

1. Test with two development kits (nRF52840DK) to verify both example codes.
2. Test with the Sensor Data Acquisition Board and a development kit (nRF52840DK).

Upon performing the first test, one could observe a toggling LED on the master device and that the slave device was toggling as well. Both devices were behaving as expected.

For the second test to achieve identical test results, it is essential for both devices to be equipped with LEDs. Unfortunately, the Sensor Data Acquisition Board does not have programmable LEDs, rendering it impossible to generate identical outcomes. However, SPI communication can still be verified by configuring the custom board as the master device and utilizing the development kit as the slave device. The slave device is dependent on the master device and will toggle its LED every time a transfer is completed.

The SPI pins available on the tested SoC are presented in Table 4.4. These pins can be located in the schematics provided in the appendix C.

SPI pin	Pin	Name	Software pin number
SPI_CS1	K2	P0.05	5
SPI_SCK	M2	P0.07	7
SPI_MOSI	N1	P0.08	8
SPI_MISO	P2	P1.08	40

Table 4.4: SPI pins on the nRF52840 SOC serving the IR-sensor tower

Due to variations in the SPI pins between the custom board and the development kit, the SPI Master example code cannot be utilized without making the necessary configurations. These configurations can be observed in Listing 4.1, where the SPI pins are defined to correspond with those on the SoC and are utilized as SPI pins in the code.

```

1 #define CHIP_SELECT_PIN 5
2 #define MOSI_PIN 8
3 #define MISO_PIN 40
4 #define SCK_PIN 7
5
6 spi_config.ss_pin = CHIP_SELECT_PIN;
7 spi_config.miso_pin = MISO_PIN;
8 spi_config.mosi_pin = MOSI_PIN;
9 spi_config.sck_pin = SCK_PIN;

```

Listing 4.1: Pin configuration for SPI

During the second test, it was noticed that the slave device either toggled at a rate faster than 200 ms or stopped toggling for specific periods. Depending solely on the LED on the slave device as a confirmation of SPI communication was no longer a reliable approach. Consequently, additional debugging of the SPI communication was deemed necessary.

4.3.3 Debugging SPI communication

A prioritised testing list had to be created to identify and isolate the cause of the observed behaviour during SPI communication testing. The list considers both hardware and software factors that could contribute to the issue.

1. Check the connections: Ensure that all connections between the SPI master and slave are correctly connected and have good solder joints.
2. Recreate the SPI communication with two development kits and use an oscilloscope to monitor the SPI signals.
3. Configure the custom PCB as the master and a development kit as the slave. Use an oscilloscope to monitor the SPI signals and compare the observations to those in step 2.
4. Same as step 3 but configure the custom PCB as the slave and development kit as the master.
5. Set each SPI pin of the custom board to a high state to confirm that the SPI lines are functioning properly.
6. In layout: review each SPI line in terms of design for signal integrity. List on things to look for when reviewing:
 - Minimize trace length: SPI traces should be as short as possible, and avoid routing them near noisy components such as voltage regulators or switching power supplies.
 - Match trace length: Ensure that the length of the traces for each SPI signals are matches to minimize skew.
 - Keep traces close together: Keep traces for the SPI signals close together to minimize crosstalk and noise.
 - Avoid using vias on SPI signals to reduce the impedance of the trace and minimize signal reflections.

STEP 1: Check the connections

During testing, to ensure the integrity of the wires, a replacement of new wires was carried out to eliminate the possibility of any broken connections. Figure 4.7 illustrates the resoldering of the SPI pins, ensuring a secure connection between the headers and the PCB.

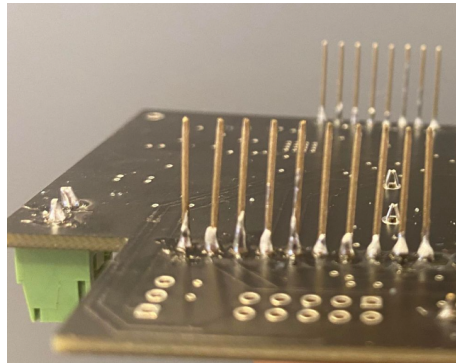


Figure 4.7: Resoldered pins on the Sensor Data Acquisition Board

STEP 2: Recreating the SPI communication with two development kits

An oscilloscope was used to capture the SPI communication. The oscilloscope used had both analogue and logic channels. During this test, one analogue channel and three logic channels were used.

Channel 1 measured the clock, while the remaining logic channels measured MOSI, MISO and the chip select. With the built-in logic analyzer, decoded the message sent with SPI communication was possible. Figure 4.8 shows the capturing of SPI communication between two nRF52840DK, and one can observe that the text string "Nordic" is sent from the master device and sent back from the slave device.

STEP 3: Sensor Data Acquisition as master

Configuring the Sensor Data Acquisition Board as master results in no data being transferred, as seen in Figure 4.9. Channel 1 displayed no indication of a clock signal among the channels observed. Out of the three logic channels, only Channel D1 were active.

STEP 4: Sensor Data Acquisition Board as slave

When configuring the Sensor Data Acquisition Board as the slave, data transfer is observed only from the master device, while no data is transmitted from the slave device. The captured SPI communication for this specific configuration can be observed in Figure 4.10. In the figure, one can observe the text string "Nordic" is sent from the master and none are sent from the slave.

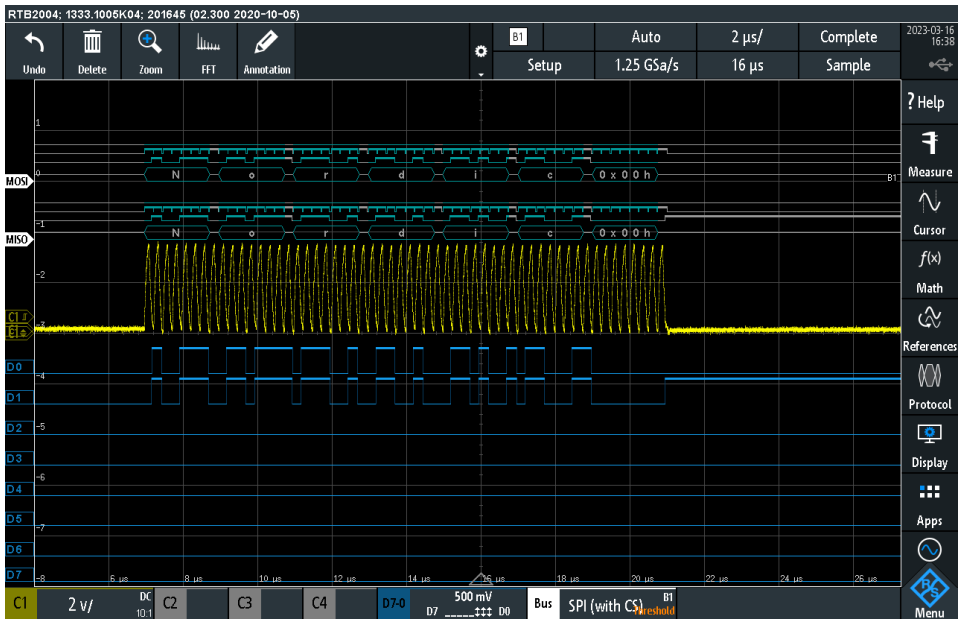


Figure 4.8: Capturing SPI communication with an oscilloscope: two nRF52840DK

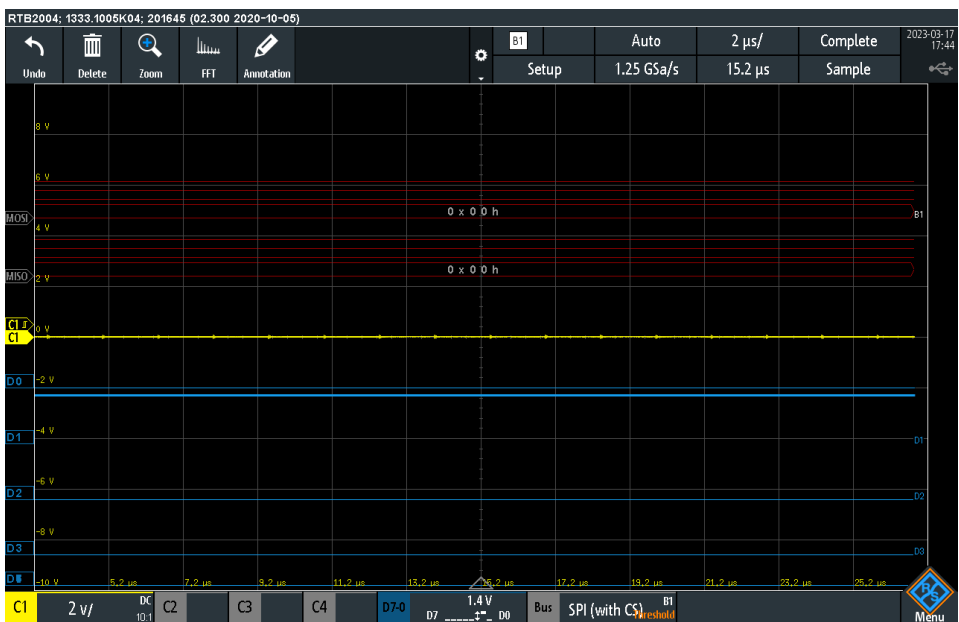


Figure 4.9: Capturing SPI communication with oscilloscope: Sensor Data Acquisition as master

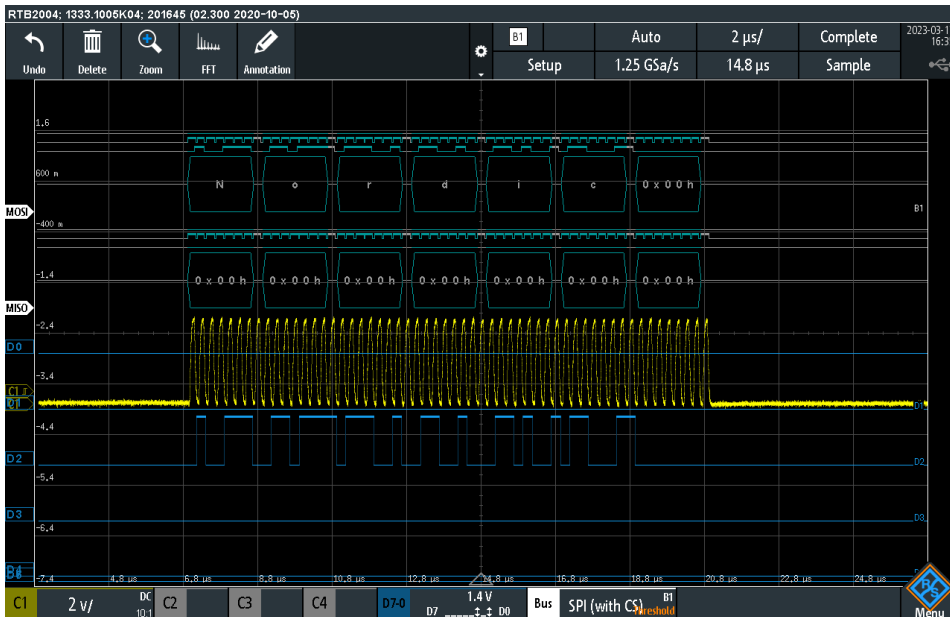


Figure 4.10: Capturing SPI communication with oscilloscope: Sensor Data Acquisition as slave

STEP 5: Setting each SPI pins in high state

All the SPI pins were set to a high state during this test, as seen in the Listing 4.2. Figure 4.11 shows the capturing of the attempt to set all SPI pins high where channel D1 is the only one in a high state. Here D0 is the clock signal, D1 is MISO, D2 is MOSI and D3 is the chip select.

```

1 #define CHIP_SELECT_PIN 5
2 #define MOSI_PIN 8
3 #define MISO_PIN 40
4 #define SCK_PIN 7
5
6 nrf_gpio_cfg_output(CHIP_SELECT_PIN);
7 nrf_gpio_cfg_output(MOSI_PIN);
8 nrf_gpio_cfg_output(MISO_PIN);
9 nrf_gpio_cfg_output(SCK_PIN);
10
11 nrf_gpio_set(CHIP_SELECT_PIN);
12 nrf_gpio_set(MOSI_PIN);
13 nrf_gpio_set(MISO_PIN);
14 nrf_gpio_set(SCK_PIN);

```

Listing 4.2: Setting SPI pin in high state



Figure 4.11: Capturing attempt to set SPI pins in high state

STEP 6: Layout review

List of observed things in the layout:

- SPI traces are kept close to each other
- Traces for MOSI, MISO and SCK are close to the 5 V buck voltage regulator. However, it is on the bottom layer and separated by a ground layer. Seen in Figure 4.12.
- Maximum two vias are used per trace for SPI data lines.
- The pins on the SoC that utilize layer 1 for tracing have been inaccurately defined, making these pins unusable on the SoC.

Figure 4.13 shows a close up on the nRF52840's footprint in both 2D and 3D layout mode. In 3D mode, one can observe that layer 1 (brown) pins are not visible.

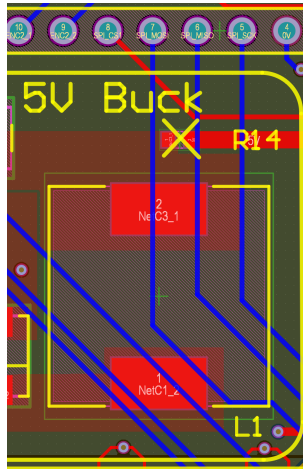
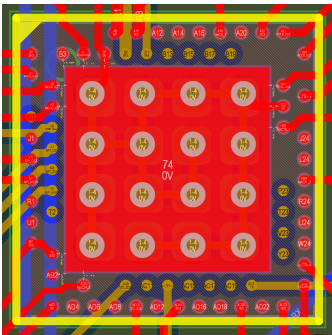
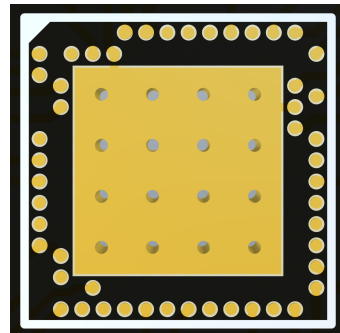


Figure 4.12: SPI traces close to 5 V buck regulator



(a) Close up of the nRF52840 2D layout mode



(b) Close up of the nRF52840 in 3D layout mode

Figure 4.13: Close up of the layout around the nRF52840

4.4 Additional errors discovered during hardware testing

While testing the custom board, it was noticed that there was an error in the pin-out configuration of the motor driver connector. Although the footprint size and pin count were accurate, the pin numbering was not as initially assumed. The selected connector had a continuous pin numbering scheme, which was expected to have an alternating pin count. Figure 4.14 shows the connector for the motor driver with continuous pin numbering, while Figure 4.15 illustrates the distinction between alternating and continuous pin numbering in a connector.

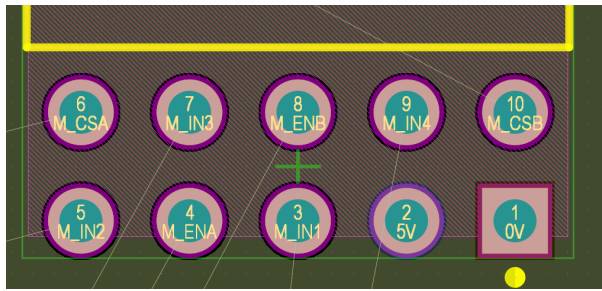


Figure 4.14: Layout revision 1: Connector for motor driver

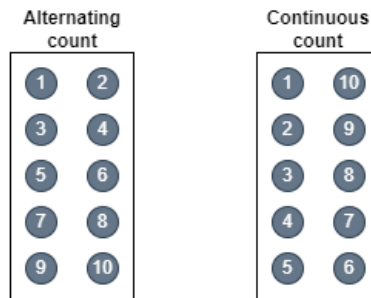


Figure 4.15: Different pin counting

Before soldering the connectors onto the custom board, all connectors and headers were placed. It was already known that the P24 connector needed to fit correctly due to an incorrect footprint selection from E.Tran’s project thesis [4]. However, during the testing phase of the remaining connectors, it was realized that the motor driver connector occupies more space than initially anticipated, as seen in Figure 4.16.

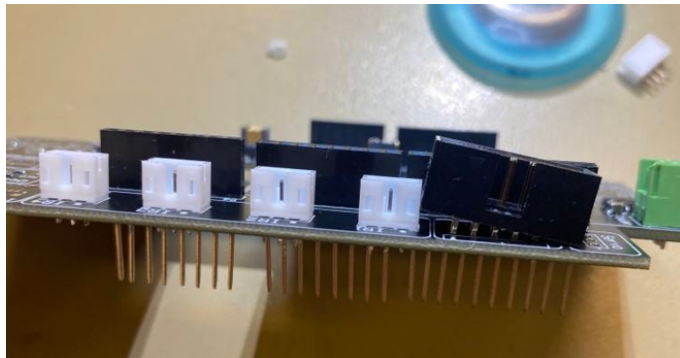


Figure 4.16: Motor driver connector occupies more space

Second revision of the Sensor Data Acquisition Board

The need for a second revision has been recognized to address the various issues and enhance the overall functionality of the design. This chapter will be divided into two sections where section 5.1 will cover the schematic changes while section 5.2 will cover the layout modifications. For the second revision of the PCB, the software tool utilized was Altium Designer, which was also used for the first revision. It is important to note that the second revision does not involve the production of the printed circuit board.

Refining the schematic and modifying layout can be time-consuming as it can take up to several hours if there are no recitations when it comes to time. The changes made to the second revision have therefore been categorized into a high, medium and low priority to distribute the time usage of each change.

Listed below are the different tasks, their prioritization and if the changes are made in the schematic or the layout:

Priority high - must be done to correct hardware failure:

- (Schematic) Add the correct footprint for connector P24.
- (Schematic) Change pin matching on motor driver connector.
- (Layout) Reroute the circuits cornering the connector P24.
- (Layout) Reroute the circuits concerning the motor driver connector.
- (Layout) Reroute the circuits concerning the IR connectors due to the direct effect of rerouting the circuit concerning the motor driver connector.

- (Layout) Correct the design fault concerning the pads for the two nRF52840 SoCs on the PCB and reroute the affected signals. The following signals are affected: *PWM, SPI_CS1, SPI_CS2, SPI_SCK, SPI_MISO, IR2, IR4, I2C_SDA, TX, nrf_reset*.

Priority medium - enhances the development of the system:

- (Schematic) Add two test LEDs.
- (Schematic) Add informative notes.
- (Schematic) Change net names.
- (Schematic) Change module and descriptive names.
- (Layout) Reroute the SPI signals were they are close to the buck regulator.
- (Layout) Adding silk layer with polarity symbols concerning the power connector.

Priority low - optional, will not affect overall functionality:

- (Schematic) Set to 1-based indexing.
- (Schematic) Rearrangement of modules.
- (Layout) Modify the silk layer to fit the connectors and components that have been modified or added.

5.1 Schematic

This section will cover the schematic changes for the second revision of the PCB. The schematic sheets for the second revision can be seen in the appendices from E to H.

5.1.1 Add correct footprint for the P24 connector

The P24 connector previously had a pitch of 2.00 mm, which has now been replaced with a connector with the correct pitch of 2.54 mm. The change does not affect the circuits on the schematic. The net names and pins remain the same.

5.1.2 Changed pin matching on motor driver connector

To resolve the issue with the motor driver connector, where the pin mapping was incorrect, this problem has been solved in the schematic by changing the placement of the net names. Figure 5.1 shows the schematic for the motor driver connector and Figure 5.2

shows the changes made for the second revision. A note in the second revision has been added for explanation.

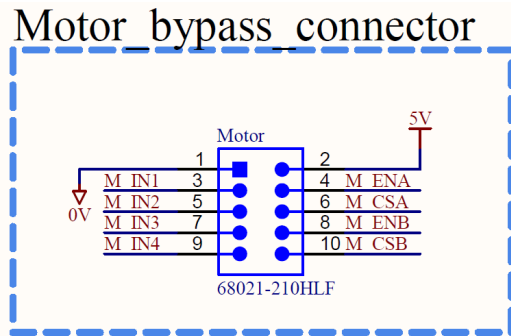


Figure 5.1: Schematic revision 1: Connector for motor driver

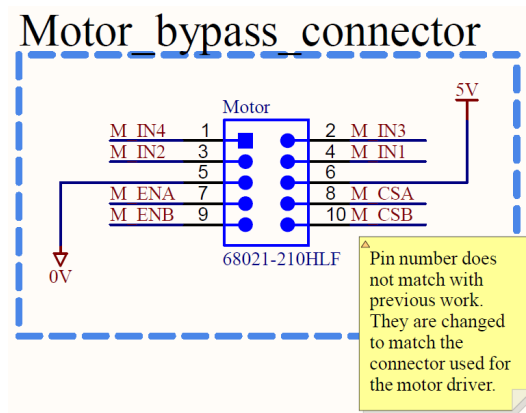


Figure 5.2: Schematic revision 2: Connector for motor driver

5.1.3 Added 2 LEDs for testing

Figure 5.3 shows a cut out of the schematic of the second revision where two tests LEDs have been added. In order to have a more clear schematic, all LEDs have been parted such that the two LEDs connected to the nRF52840DK and the LED connected to the board are separated. The LEDs under "LED_nRF52840DK" can only be accessed and used with another development board, while the two added LEDs can be controlled by the SoCs on the PCB. The purpose of the remaining LEDs is to indicate whether 5 V and 3.3 V can be accessed from the board.

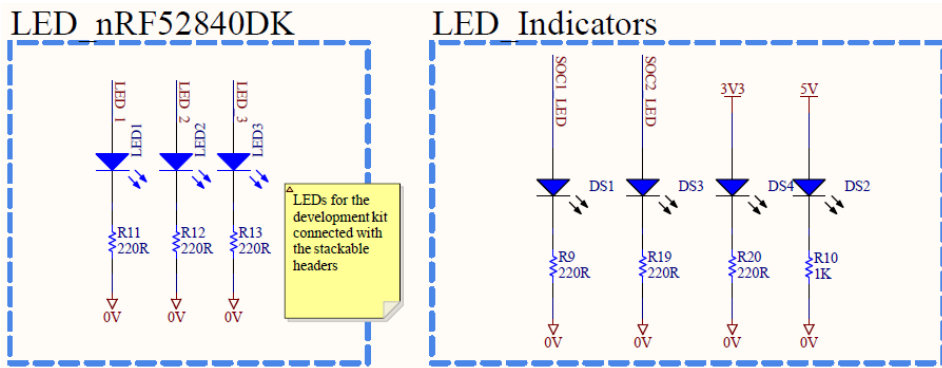


Figure 5.3: Schematic revision 2: The available LEDs on the PCB

5.1.4 Added informative notes

In order to provide additional information and aid user navigation, informative notes have been added to the main schematic. These notes serve to describe the overall system and assist users in understanding its various components. An example of an informative note can be observed in Figure 5.4, while all new notes can be found in the main schematic shown in the appendix E. These informative notes contribute to a more comprehensive and user-friendly documentation of the main schematic.

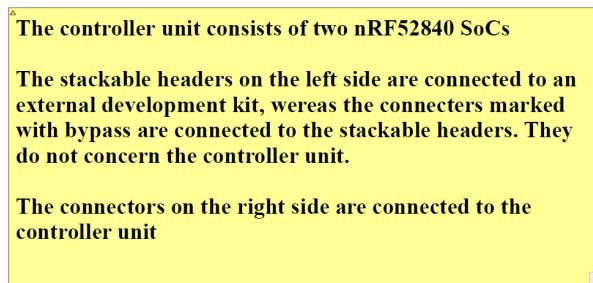


Figure 5.4: Schematic 2nd revision: Informative note in the main sheet

5.1.5 Name changes for net names

Table 5.1 shows the net names that have been changed. It is worth mentioning that the table does not present a list of all net names starting with MCU0 and MCU1. However, it emphasizes the modification of net names commencing with MCU0 or MCU1, which correspondingly have been substituted with SOC1 and SOC2.

Previous net name	New net name
MCU0_...	SOC1_...
MCU1_...	SOC2_...
ENC1_1	ENC_L_1
ENC1_2	ENC_L_2
ENC2_1	ENC_R_1
ENC2_2	ENC_R_2

Table 5.1: Net name changes

5.1.6 Changes for modules and descriptive names

In order to adopt a more informal and descriptive naming, the modules and descriptive names have been modified. Rather than using names like *Encoder 1* and *Encoder 2*, more descriptive names such as *Encoder Left* and *Encoder Right* have been implemented. This change aims to provide clearer and more informative labels for the respective modules. Table 5.2 provides an overview of the modifications made to the module names, whereas Table 5.3 displays the modifications made to the descriptive names.

Previous module name	New module name
Microcontroller unit	Controller unit
nRF52840_Servo_Tower	nRF52840_sensor_tower

Table 5.2: Module name changes

Previous descriptive name	New descriptive name
MCU_servo_tower	SOC_sensor_tower
Encoder 1	Encoder Left
Encoder 2	Encoder Right

Table 5.3: Descriptive name changes

5.1.7 Set to 1-based indexing

It was discovered that there was no consistency in using 0-based indexing or 1-base indexing. For consistency and to avoid confusion, 1-based indexing was chosen. The choice was decided based on the authors' preference.

5.1.8 Rearrangement of modules

The main schematic has been improved by reorganizing and modifying the modules for clarity. The schematic has been improved through careful rearrangement to provide a more coherent and easily understandable representation of the system. This revision ensures that the relationships between modules are better illustrated, making the schematic more user-friendly and facilitating efficient analysis and troubleshooting. The second revision of the main schematic can be seen in the appendix E.

5.2 Layout

This section will present the layout changes implemented in the second revision of the custom board. Figure 5.5, Figure 5.6, Figure 5.7, and Figure 5.8 shows the four layers of the second revision of the PCB. DRC was performed after the layout design to ensure proper design. Modifications and changes will be addressed in this section.

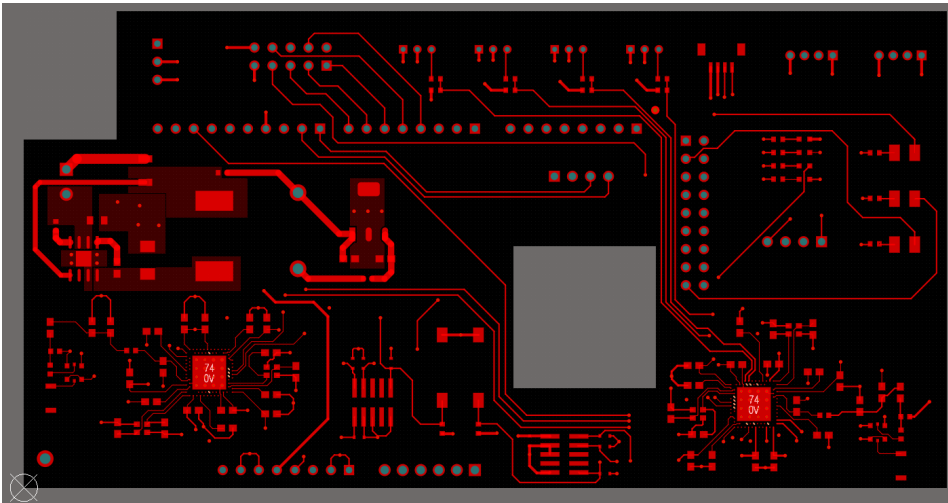


Figure 5.5: Layout revision 2: Top layer

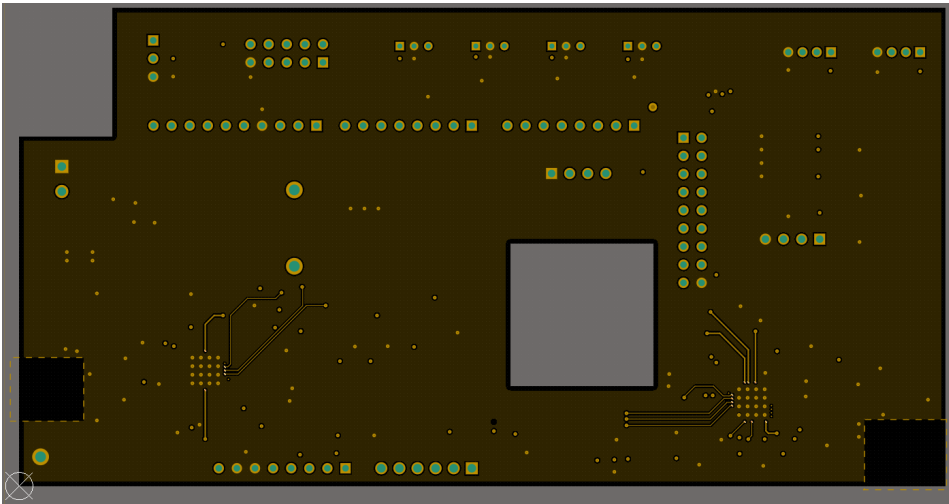


Figure 5.6: Layout revision 2: Layer 1

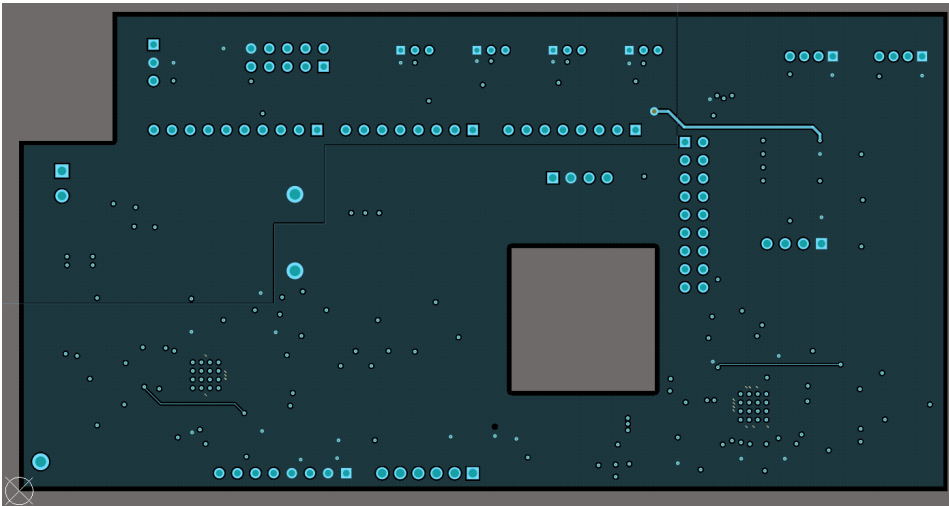


Figure 5.7: Layout revision 2: Layer 2

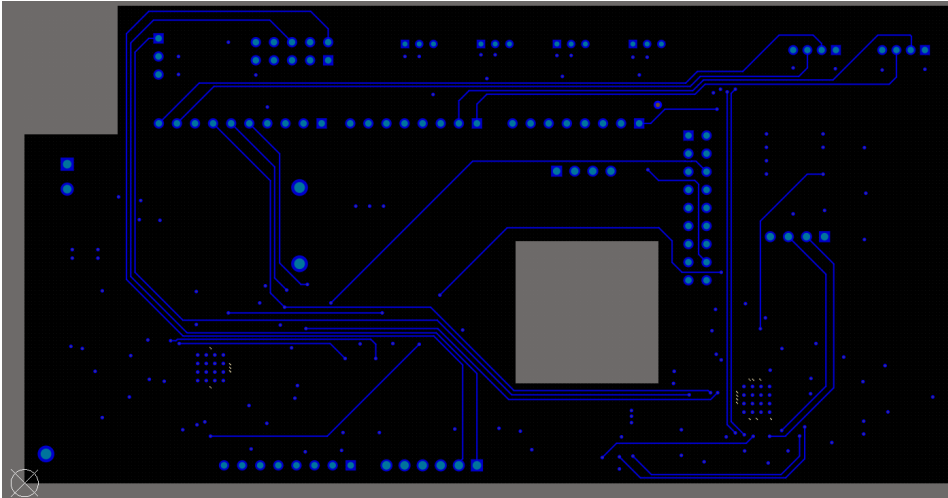


Figure 5.8: Layout Revision 2: Bot layer

5.2.1 Rerouting

Figure 5.9 shows the layout for the correct connector for P24 and the newly added LEDs: SOC1 and SOC2.

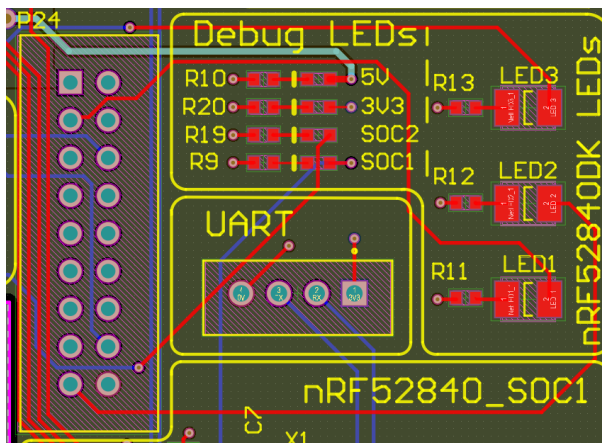


Figure 5.9: PCB Layout revision 2: Layout around the PCB's LEDs and connector P24

Figure 5.10 shows the exact pin mapping for the motor driver connector as in the first revision. Instead of changing the pin counting to alternating count, the different net names were shuffled to fit the motor connector. The placement of the net names was modified in the schematic, as described in section 5.1.2.

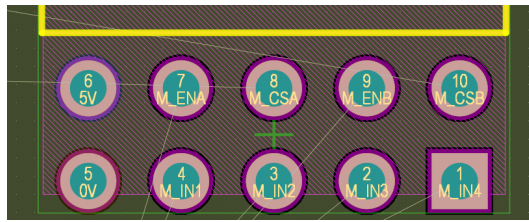


Figure 5.10: Layout revision 2: Connector for motor driver

Due to the size of the package of the motor driver connector, the IR-sensor connectors had to be moved. The IR-sensor connectors are moved closer to each other, giving the motor driver connector more space. The new connector layout can be seen in Figure 5.11.

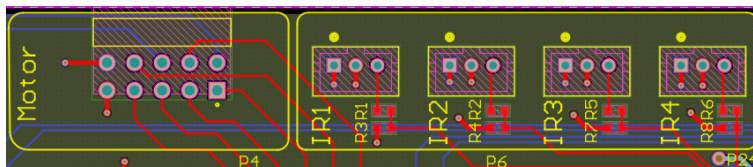


Figure 5.11: Layout revision 2: New placement for connectors concerning motor driver and IR sensors

To assure that the SPI are not affected by signal integrity, parts of the SPI lines have been rerouted to address the previous revision where the SPI lines were routed right beneath the components regarding the buck regulator. Figure 5.12 shows the new routing for the SPI lines where the traces avoid the components concerning the buck regulator.

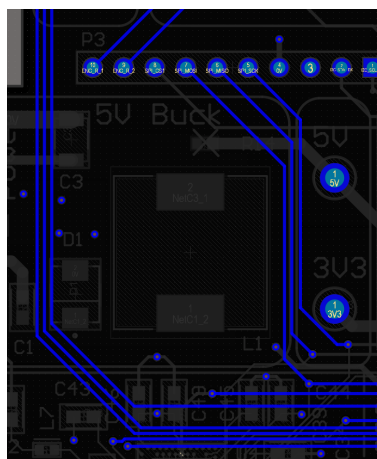


Figure 5.12: Layout revision2: Routing on bot layer under the 5 V buck regulator

5.2.2 Correcting the pads for the nRF52840 SOC

The datasheet of the nRF52840 does not specify how different layers are connected to the pads. Nordic Semiconductor does, however, have a document on guidelines for circuit boards for aQFN package[22], which is the package for the nRF52840 SoC that is used. In the guidelines for PCB land pattern design, it is specified that a four-layer board with micro vias are used. From the guild lines and the example picture which is provided in the datasheet for the SoC, all pads are set to be available on the top layer and to connect to other layers μ Vias are used.

The following list describes how to configure μ vias in Altium Designer:

1. Click on *Design*→ *Layer Stack Manager*
2. In the layer stack manager bar (see Figure 5.13) click on *Via types* (see Figure 5.14).
3. Add a via by clicking on *+add*
4. Go to Properties and choose the newly added via
5. Choose the first layer as the top layer and the last layer as layer 1. Check off μ via (see Figure 5.15).

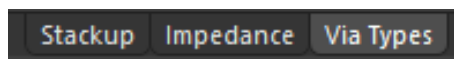


Figure 5.13: Altium Designer: Stackup - Impedance - Via types

#	Name	Type	Thickness	#	Thru 1:4	μ Via 1:2
	Top Overlay	Overlay				
	Top Solder	Solder Mask	0.01016mm			
1	Top Layer	Signal	0.035mm	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	Dielectric 2	Prepreg	0.07112mm			
2	Layer 1	Signal	0.035mm	2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	Dielectric 1	Core	0.889mm			
3	Layer 2	Signal	0.035mm	3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	Dielectric 3	Prepreg	0.07112mm			
4	Bottom Layer	Signal	0.035mm	4	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	Bottom Solder	Solder Mask	0.01016mm			
	Bottom Overlay	Overlay				

Figure 5.14: Altium Designer: Layer stack manager - Via types

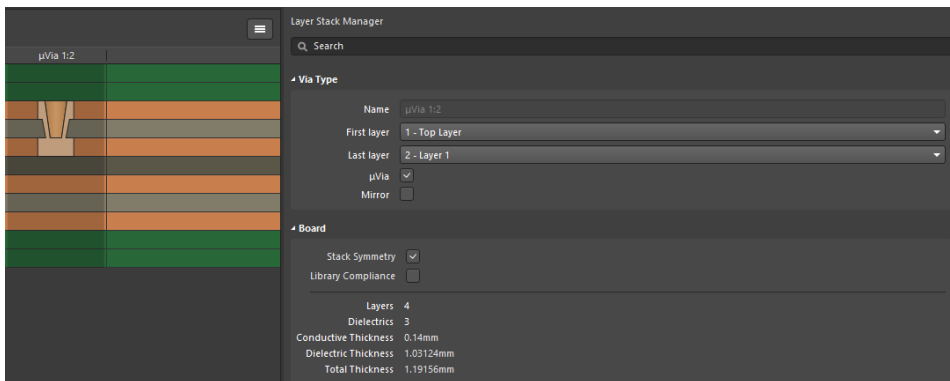
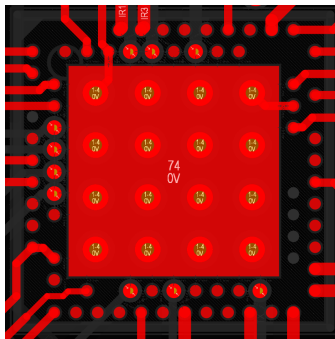
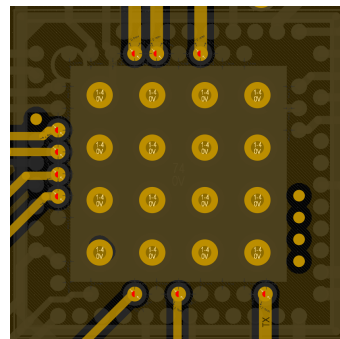


Figure 5.15: Altium Designer: Layer stack manager - Properties of a μ via

The used pads on the nRF52840 are set to be available on the top layer (in red) see Figure 5.16a, while the pads that use the layer 1 or power layer (in brown) to route have μ via attached to the pad as seen in Figure 5.16b. One can observe that the pads where a μ via is attached are half red and half brown. Note that this is done to both the SoCs on the PCB. The figures seen below illustrate the SoC designated for the IR-sensor tower and not the SLAM SoC.



(a) Top layer for the nRF52840



(b) Layer 1 for the nRF52840

Figure 5.16: Layout revision 2: nRF52840 with the use of μ Vias

5.2.3 Silk layer changes

Figure 5.17 shows the silk layer where the polarity symbols are added to the power connector. Visible polarity symbols are crucial because they provide essential information about a component's correct orientation and polarity, in this case, a power connector. If voltage with the wrong polarity is connected, it can cause several negative consequences, such as damaging the circuit, overheating or electric shocks.

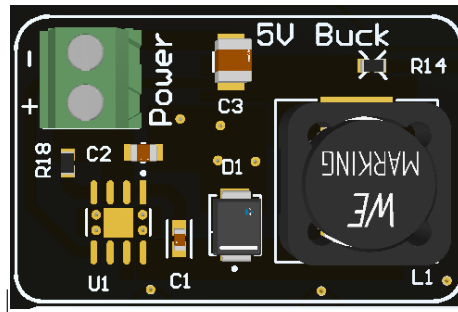


Figure 5.17: Layout revision 2: Power circuit in 3D

Figure 5.18 shows the silk layer around the LEDs available on the custom board. Observe how the Debug LEDs are separated from the LEDs that can be connected to an external nRF52840DK.

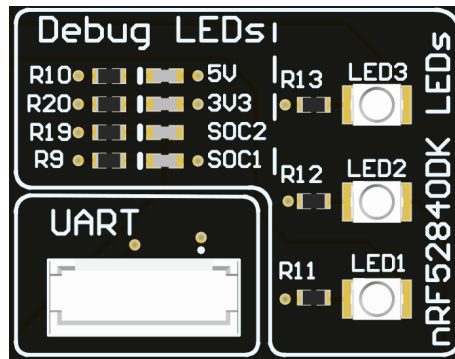


Figure 5.18: Layout revision 2: The available LEDs on the PCB

Figure 5.19 shows the silk layer around the motor driver and IR connectors.

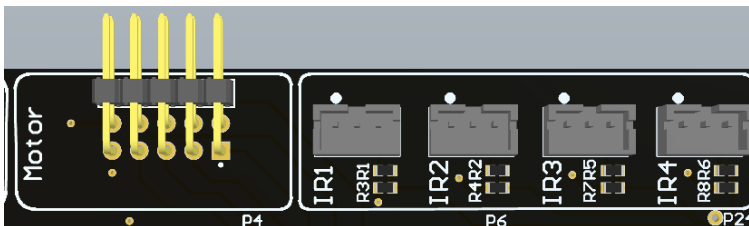


Figure 5.19: Layout revision 2: Connectors for motor driver and IR sensors

Chapter 6

Software analysis and design

This chapter focuses on the software design for the new system, particularly the development of new software for one of the two SoCs on the custom board. Before proceeding with the software design and implementation, a comprehensive analysis of the existing code is essential since certain portions are intended to be transferred to the new system. Assessing how introducing the new system will impact the current code base on the robots' nRF52840DK is crucial. Additionally, it is vital to identify the potential challenges that may arise during the integration process. The project can effectively navigate the integration by addressing these considerations, ensuring a seamless transition and successfully incorporating the new system.

The following topics will be covered in this chapter:

- A brief analysis of the code on the nRF52840DK
- Requirements and specifications for the new system
- Proposal of new design
- Creating a new software project

6.1 Brief analysis of the robot code

In the current robot code, running on the nRF52840DK, the task *vMainSensorTowerTask* is responsible for scanning the environment using the IR-sensors. This responsibility is intended to be delegated to one of the SoC on the new custom board. Before implementing this responsibility to a new system, it is necessary to analyse how the task *vMainSensorTowerTask* are integrated with the current code base. This is essential to determine the code's reusability and compatibility with the new system and the nRF52840DK.

6.1.1 Analysing the task *vMainSensorTowerTask*

A scanning sequence consists of a 90° rotation and results in one measurement for each degree of rotation. Each measurement contains sensor data from four IR-sensors and information on the current robot position. The current position is estimated in task *PoseEstimateTask* and shared through global variables. These global variables are protected by the mutex *xPoseMutex* [3]. Figure 6.1 illustrates a simplified overview of the interactions concerning the task *vMainSensorTowerTask*. Note that this figure does not include the remaining tasks and their interactions.

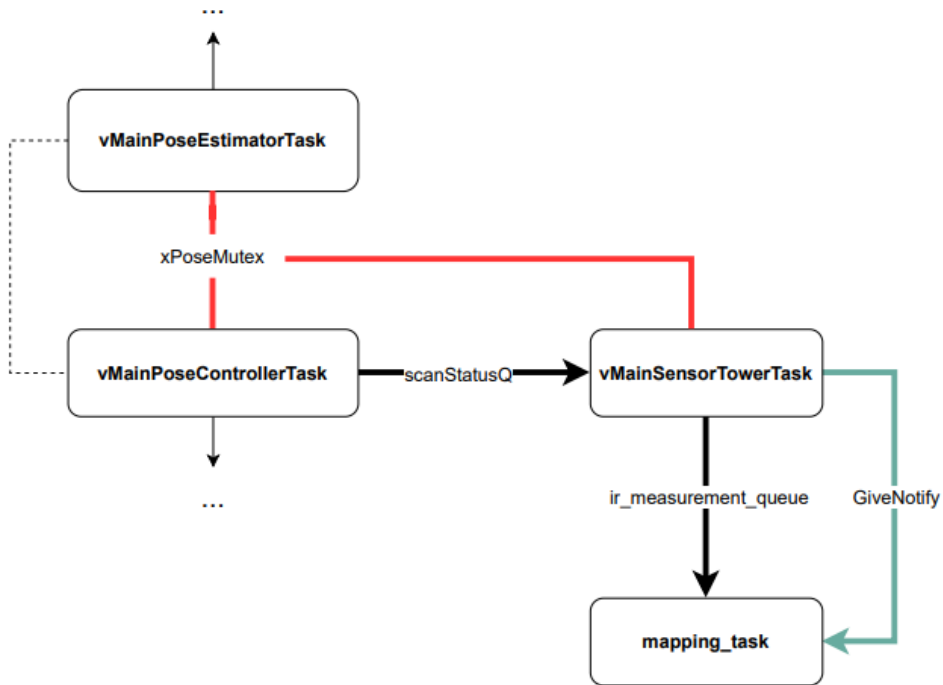


Figure 6.1: Simplified diagram of the interactions concerning the task *vMainSensorTowerTask*

Scanning begins when the robot is stationary and ends when it is in motion. With the queue *scanStatusQ*, the task *vMainPoseController* updates the task *vMainSensorTowerTask* with the latest robot movements.

The scanned data can be distributed to the server with two different methods. The first method involves sending each measurement to the task *mapping_task* for processing using the queue *ir_measurement_queue*. After 90 measurements, the task *vMainSensorTowerTask* will notify the task *mapping_task* that the scanning sequence is completed, and the task will send the processed data to the server. The second method sends each measurement to the server as they are generated. The server will then start processing the sent data and draw the map. This method is not described in Figure 6.1. The preferred method can be configured in the file *robot_config.h*.

Through analysis, it is revealed that the code has extensive dependencies and lacks modularity. The task *vMainSensorTowerTask* relies on three tasks for its operation, whereas in the ideal scenario with the new system, data transmission would occur between two devices. Upon a request from the robot's nRF52830DK, the custom board would send the processed, scanned data as seen in Figure 6.2.

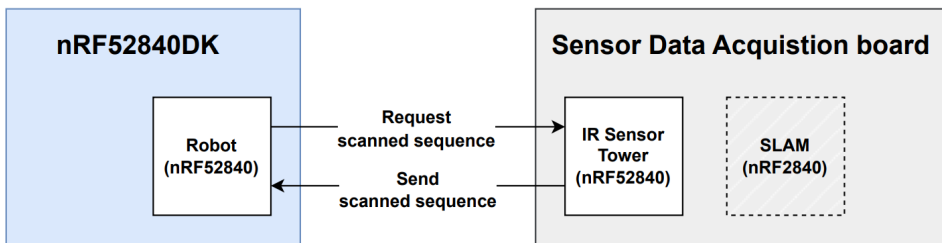


Figure 6.2: Data transmission between development kit and the sensor data acquisition board, ideal scenario

The current robot project lacks a framework defining the desired structure of the code base, making it challenging to design a new code project that can be seamlessly integrated into the existing robot project. It is proposed that the drivers for the *vMainSensorTowerTask* can be reused, although their use would need to be developed from scratch.

6.1.2 Deciding between continuing development or rewriting the software

Given that the new system is integrated, it should be considered whether the integration should be developed in alignment with the current code base or if a complete overhaul is required. In making this decision, one should evaluate the compatibility and scalability of the current code to new and future systems. Code complexity, maintainability, and potential conflicts must be analysed thoroughly.

The new system is designed to rely on SPI communication for collecting data sequences from sensor measurements. However, simply developing a SPI driver on the nRF52840DK does not resolve the integration challenges of incorporating the new system. Based on the analysis conducted in section 6.1.1, it is evident that the code base lacks modularity. Therefore, it is proposed to restructure and modularise parts of the robot's code base, considering the project's absence of a current software design framework.

Due to the absence of modularisation in the existing code base, various adverse effects arise, including:

- **Lack of reusability** : Non-modular code is often difficult to reuse. It can result in reinventing the wheel for similar functionalities across different projects or within the same project, leading to efficient use of development resources.
- **Poor maintainability**: Non-modular code tends to be more difficult to maintain. Making changes or fixing issues can be challenging as the codebase needs more precise separation and organization. This can result in longer debugging and troubleshooting times.
- **Scalability issues**: Non-modular code is less flexible and scalable. Integrating new features or making substantial changes to the system becomes more complicated and error-prone, potentially introducing more bugs and increasing the time required for the development.
- **Reduced collaboration**: Non-modular code can hinder collaboration among developers. Working on a monolithic codebase without clear boundaries and interfaces between components can lead to conflicts and difficulties in coordinating efforts among team members.
- **Testing challenges**: Testing non-modular code can be more complex. It may be difficult to isolate specific functionalities for testing, resulting in longer test cycles and increased chances of undetected bugs.
- **Limited flexibility**: Non-modular code is less adaptable to future changes. Introducing new technologies, frameworks, or architectures may be harder, limiting the system's ability to evolve and adapt to evolving requirements.

6.2 Requirements and specifications for the new system

Defining the systems requirements and specifications is crucial for several reasons:

- **Clarity of purpose**: Clearly defining the functionality helps ensure a shared understanding of what the software should achieve and establishes the purpose and goals of the project.

- **Scope management:** The process of defining requirements and specifications sets clear boundaries for the software project, outlining the specific elements that fall within the scope of work and identifying those excluded.
- **Quality assurance:** Specifications and requirements provide a basis for evaluating the quality and performance of the software system. They serve as criteria for testing and validation, enabling objective assessment of whether the software meets the desired standards and fulfils the specified requirements.

The system design requirement and specification are specifically focused on the new system involving the sensor tower. It is important to note that the system design and any software modifications related to the nRF52840DK are not part of this project's scope.

6.2.1 Requirements

Requirements refer to the descriptions of what a system or software should achieve or what characteristics it should have. The focus of requirements is to identify the needs and expectations of the system.

The objective of the IR-sensor tower is to perform a scanning sequence comprising 90 measurements. The initiation of a scanning sequence is triggered by request from the nR52840DK. Compared to the task *vMainSensorTowerTask* on the current software on the nRF52840DK, the transmitted data does not include information about the position of the robots. The position is not included as it is irrelevant for the IR-sensor tower, as its responsibility is to deliver scanned measurement sequences.

From the analysis covered in 6.1.1, the task *vMainSensorTowerTask* sends the data regarding the scanned sequence and information on the current robot position to the task *mapping_task*. Considering that the task *mapping_task* requires both the scanned data and information about the current robot position, it is recommended to address this internally within the nRF52840DK to modularize the operations of the SoC running the software for the IR-sensor tower.

The requirements of the software system are the following:

1. The initiation of a scanning sequence is triggered by request from the nR52840DK.
2. Perform a scanning sequence comprising 90 measurements.
3. Once the scanning sequence is completed, transmit the scanned sequence to the nRF52840DK.

6.2.2 Specifications

While requirements establish the system's objectives, specifications define the methods and approaches to achieve those goals. They specify the technical details, standards, interfaces, and protocols to be used in the development process.

The following specifications for the software system are:

- SPI communication protocol must be used for data transmission.
- An operating system is not needed for this application. However, if an operating system is employed, FreeRTOS is the preferred choice as it is the used RTOS in the project.

6.3 Proposal of design

This section will present the proposed design for the software that will run on one of the SOC of the custom board. Based on the requirements and specifications described in section 6.2, the following design recommendations are:

- Reuse the drivers: servo and IR-sensors from the original robot code base running on the nRF52840DK
- Write SPI drivers for communication. Either use the one provided by Nordic Semiconductor's SDK or develop own SPI drivers.
- Write the application that uses these drivers.

6.4 Creating a new software project

This thesis has explored two methods for developing a new software project intended to run on an nRF52840 SoC. The first method consists of creating a new blank project and manually downloading and including libraries as needed. The second method uses an example project with built-in libraries as a base.

Both methods were developed using the FreeRTOS operating system. Despite the uncertainty of whether the new system will require an operating system, it was still implemented as a precautionary measure. FreeRTOS was chosen due to its familiarity with the project. Implementing FreeRTOS does not affect software that does not use the operating system. The FreeRTOS package can be included in the file *main.c* if needed.

The IDE used to develop both methods was the SEGGER Embedded Studio. Detailed instructions for both methods can be found in the appendices I and J.

6.4.1 Testing the software project

Each method was tested with a nRF52840DK with these test cases:

1. Blink a LED every 500 ms. Does the project compile, build and run as intended?
2. Blink two LEDs with the use of two FreeRTOS-tasks. Does the project compile with the use of FreeRTOS?
3. Blink three LEDs using three FreeRTOS-tasks with significantly different priorities. Does the project compile, build and run FreeRTOS tasks as intended?

Method 2 was the only method that successfully passed the second and third tests, whereas both cleared the first test. When running the software created with method 1, more challenges were encountered, and more troubleshooting and debugging were required. On the other hand, running method 2 went seamlessly without issues.

Improving and structuring of the robot project

As described in chapter 2 the robot project has been under development since 2004 and consist of several project and master theses. The robot project was initially a group project. However, over the years, the work structure has changed the project to individual tasks where the common denominator is the robot. Over time, the progress of robot development has slowed down. Although new features and technologies are being added, there is a need to establish a cohesive structure to integrate them all. During the autumn of 2022, the student writing their thesis expressed that the project needed a standard structure. The students recommended that next year's students for the spring semester of 2023 take a step back, analyse the state of the project and structure the project rather than developing additional new features.

This chapter will discuss the collaborative efforts of four students who worked on the robot project in the spring of 2023. It will also explain the author's contribution and reasoning behind their work in this project. The collaborative work has naturally been distributed unevenly, given that the master's theses of these students have different focuses, with some being more reliant on the structural aspects. The four students have collaboratively written the following text, an introduction to the work contributed to the robot project.

“ The four students working on the SLAM Robot project this semester, Spring 2023, have made an attempt to improve the overall quality of the project, alongside our individual projects. The goal through this collaborative effort has been to ease the process of any future students, and to make the code easier to develop and maintain.

The project contains large amounts of code with little to no regulation of

code quality. Consequently, it has been regarded as hard to comprehend for newer students. This has in turn affected the progress of the SLAM-project over time. Ideally, new students should be starting on the project where the previous students left off, but this has not been the case. Students have reported that a lot of time has been spent in the initial stage of learning how the robots work, which could be prevented if necessary information was easy to find.

There is a large amount of information available regarding how the code works, and the reasoning behind certain features. However, there was no overview of which theses contained what information, and to get a full understanding of the current code you would have to read through a lot of thesis in order to find what is relevant. Additionally, it was up to each person to decide how to document their code and to write the code in any way they liked. This has caused a complete lack of structure, making it unnecessarily difficult to understand the code.

Upon completion of the project a student would submit their work as a zip file containing their entire project. As multiple students were working on different parts of the robot, this would result in multiple zip files of code being delivered separately, with necessarily no designated main project. The following semester, the new students would then be presented with the question as to which project to base their work with, leaving documentation and work from the other prior projects behind. As follows, the general project would miss out on useful features and bug fixes throughout the revisions.

Making use of the GitHub organization, one would open for version control and collaborative work on the projects, with the possibility of creating new repositories for experimental work, and merging these into the main project if they are decided to become a main feature. This way, new students will always have one place to look for their project material, and all documentation and code features will be preserved and accessible down the line.

Listed below are the work that has been done to improve the project and its workflow:

- Weekly meetings to update on workflow and the state of the project
- Set name conventions for the software running on the robots nRF52840DK.
 - Change the code to comply with the name conventions
 - Deleted unnecessary comments
- Changed the file structure of the project
 - Changed file names
 - Divided the project into parts
 - Changed the path of software running on the robots nRF52840DK.
- Created a GitHub organization
 - Added the projects code to GitHub

- Created git ignores
- Deleted unnecessary and unused files in the project
- Created repository name
- Added informal readMe files to each repository.
- Created a Wiki for documentation of the project
 - Added relevant project an master theses that has contributed the project
 - Added information of the project in parts such as: software, hardware, known bugs etc.
 - Added HowTos:
 - * How to start and end the project after a work period or semester
 - * How to create a new software project with FreeRTOS
 - * How to edit the Wiki

”

Ruud-Olsen, Forsdahl, Kolbeinsen, Tran

In the upcoming sections of this chapter, I will explain the reasoning and methods behind the tasks I have contributed to. Below is a detailed list of the specific tasks I have worked on:

- Created GitHub organization
- Restructuring the robot code on the nRF52840DK
- Code cleanup of the robot code on the nRF52840DK in collaboration with M.Olsen
- Pushed the IR-sensor tower code project to GitHub
- Written part of the documentation to the wiki
 - Robot HW
 - Robot SW - tutorials on how to create a software project
 - Sensor Data Acquisition Board
 - Added information about previous projects and master theses

7.1 GitHub and GitHub Organizations

Since the earlier students on the project have submitted their theses in the form of ZIP-files, the upcoming students working on the project will face difficulties in determining which one to base their project on. Furthermore, different ZIP-files often contain varying types of code that might be necessary for further development. With limited knowledge

of the project, students might arbitrarily choose a ZIP-file, potentially leading to the loss of important work done by other students that are located in other varies ZIP-files. Additionally, there are no guidelines indicating the level of collaboration among previous students or whether the project code has been merged effectively.

By the use of GitHub, these problems can be solved as GitHub enables collaborations on software development projects. GitHub is a web-based platform and version control system. It provides developers with a centralized hub to store, manage and track changes to their codebase. Listed below are some of the functionalities with the use of GitHub:

- **Version control:** GitHub utilizes Git, a distributed version control system, allowing developers to track changes, manage different versions of their code and collaborate seamlessly with others.
- **Repository hosting:** GitHub provides a platform to host repositories, which are collections of code and related files.
- **Collaborations:** GitHub enables seamless collaboration among developers. Multiple individuals can work on the same project, make changes to code, and merge their contributions using pull requests.
- **Branching and merging:** Developers can create separate branches to work on specific features or bug fixes and then merge them back into the main codebase.

In addition to delivering work in the form of ZIP-files, the project has made the decision to host all software on GitHub. This approach ensures that future students on the project will have clear access to the codebase on which they should base their projects. The intention is to encourage students to actively push and pull the most recent code changes, thereby keeping the entire project's code synchronized until the next iteration when new students join the project.

GitHub is great if the project consists of one code project. However, the robot project itself actually consists of several code projects. It was therefore decided to create a GitHub organization. GitHub organization are a feature provided by GitHub that allows individuals and teams to collaborate on software development projects. It provides a centralized platform to manage repositories, teams, and access permissions within an organization.

Currently, the robot project comprises a total of seven code projects. Each of these projects has its own repositories, which are all located in the Git organization. Figure 7.1 shows the organization created in GitHub, which is called SLAMRobotProject.

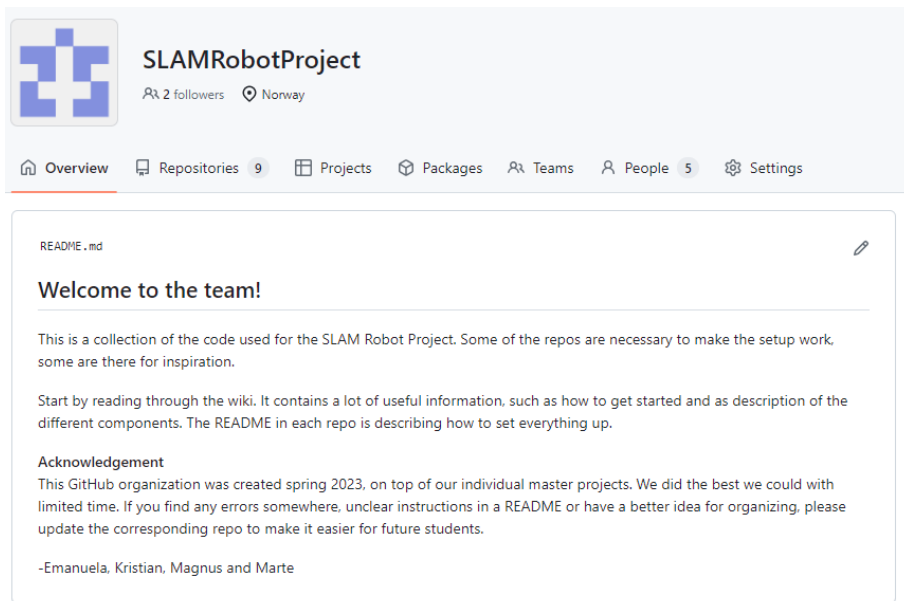


Figure 7.1: GitHub Organization: SLAMRobotProject

The seven code projects that are located in the GitHub organization are the most recent and updated code projects from the previous students on the SLAM robot project. Out of the seven code projects, four were previously located in one folder, whereas the remaining three were located in other ZIP-files belonging to other previous students on the project. The folder that consists of the four code projects is based on H.Frestad's work folder from his project thesis[3]. Figure 7.2 shows the original folder structure of the SLAM project.

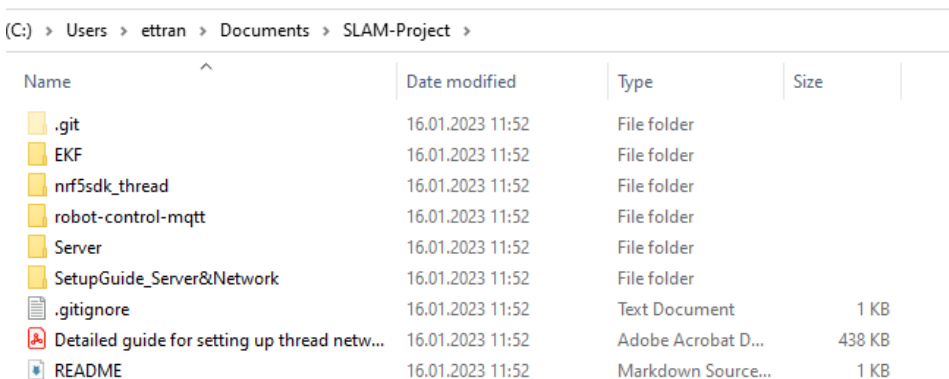


Figure 7.2: Folder structure of the SLAM Project

Without prior knowledge about the project, navigating and understanding the content of this project folder is not necessarily intuitive. This folder contains the following code projects:

- **EKF:** python code for implementing EKF SLAM on the robot
- **nrf5sdk_thread:** robot code on the nRF52840DK + nRF SDK
- **Server:** C++ sever
- **robot-control-mqtt:** communication with robots without the use of C++ server

In addition to the four code projects listed above, there are additional code projects located within ZIP-folders that belong to previous students. These are:

- **Sensor tower:** new system under development for the IR sensor tower on the robots
- **java server:** java server
- **matlab-robot-code:** MATLAB generated code for the robot

The current workflow lacks a guarantee of preserving all work and ensuring seamless functionality without the risk of introducing new bugs or failures. However, by actively utilizing GitHub organization, these challenges can be mitigated, and a more reliable process can be established. Figure 7.3 shows the current repositories available in the GitHub organization SLAMRobotProject.

In addition to the seven repositories, there are two repositories, one for the informative text when entering the organization, shown in Figure 7.1 and the second serving as the link for the documentation of the project. More about the documentation of the project will be covered in section 7.3.

The seven repositories contain the project codes that have been mentioned. However, the names of these code projects have been modified to provide more descriptive information, and each repository contains additional relevant details. Table 7.1 shows the new names that the code projects have been given.

Previous folder name	New repository name
EKF	EKF SLAM
nrf5sdk_thread	robot-code
server	cpp-server
robot-control-mqtt	python-robot-control
sensor data acquisition board	ir-sensor-tower
java server	java-server
matlab-robot-code	matlab-robot-code

Table 7.1: New names for the different code projects

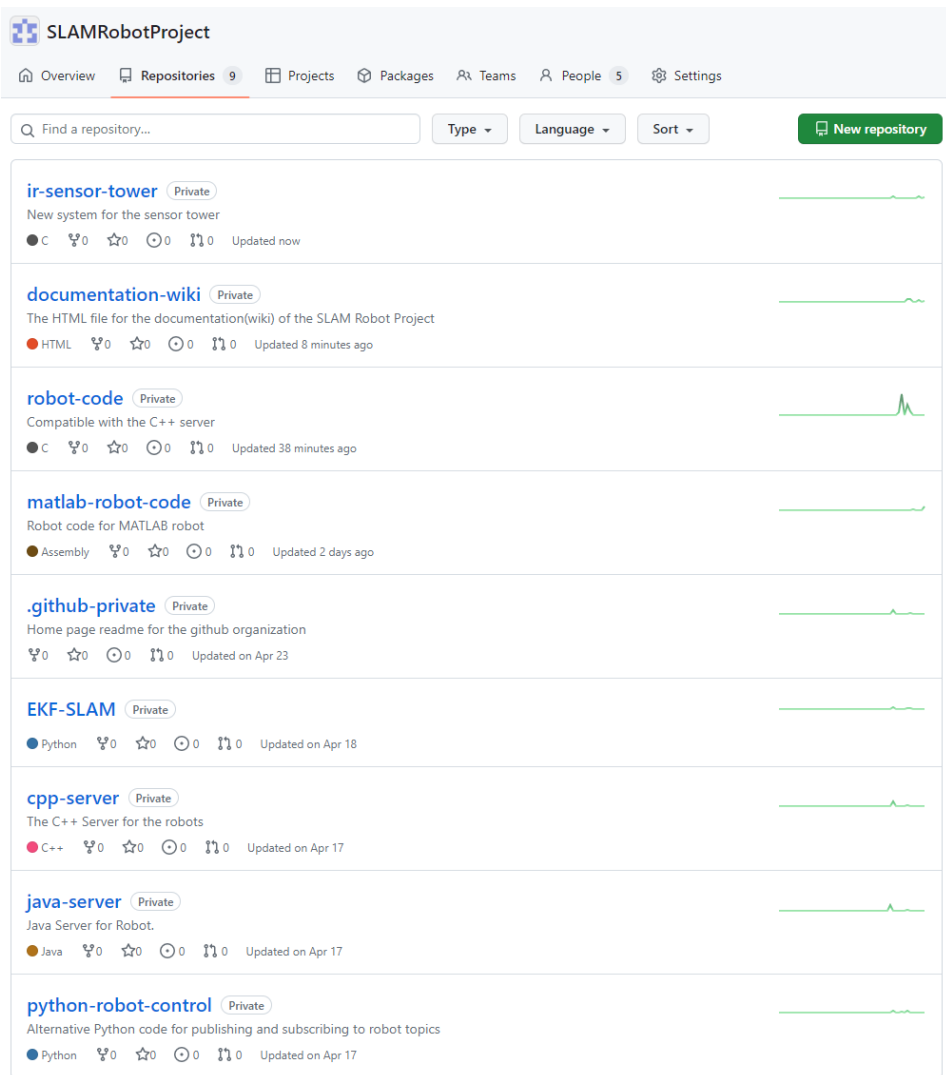


Figure 7.3: Repositories of the GitHub Organization: SLAMRobotProject

In addition to having all code on GitHub, unused files were deleted, and the paths of the folders were sorted out in collaboration with M. Ruud-Olsen. It is important to note that this effort is just the beginning of the process to improve the clarity of the project's codebase and is not yet considered complete. Table 7.2 shows the difference in the size of each code project after the cleanup.

Previous folder name	Current repository name	Previous size	Current size
nrf5sdk_thread	robot-code	1,33 GB	522 MB
robot-control-mqtt	python-robot-control	332 KB	18 KB
Server	cpp-server	82,7 MB	2.76 MB
EKD	EKF-SLAM	5,5 MB	5.5 MB

Table 7.2: Size different for the code project after the clean up

Section 7.2 will discuss the methodology used for structuring the code project robot-code. The code base of the robot code was extensive such that the workload had to be distributed. M. Ruud-Olsen was therefore assigned to structure the remaining code projects; *robot-control-mqtt*, *cpp-server* and *EKF-SLAM*, while the author of this thesis was assigned to structure the code project: *robot-code*.

7.2 Structuring and sorting the code project: robot-code

Previously, the robot code was referred to as *nrf5sdk_thread*, and its source file was located within an example folder in Nordic Semiconductors nRF5 Software Development Kit. The reasoning behind this approach was that dependencies are automatically fixed with this solution, as seen in Figure 7.4, which is a screenshot of the README-file located in the SLAM-project folder.

```
## Path to Segger project files:

**\nrf5sdk_thread\examples\thread\freertos_mqttsn\pca10056\blank\ses**
Yes, the project is located in the examples folder of a library... According to students that came before us, the reason is that dependencies is automatically fixed. Someone with the right knowledge should definitely try to fix this.
```

Figure 7.4: README-file from the SLAM project

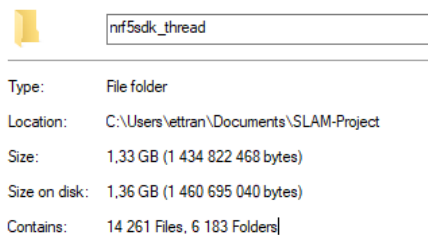
Given the information that dependencies are fixed and a brief look at the code, it was assumed and considered that the source code could be parted from the SDK since it only uses some of its libraries and drivers. The source code would then be the code base that should be changed while the SDK would be downloaded and used from Nordic Semiconductors website. That way, one will minimize the size of the project and only change the code that is needed.

Figure 7.5 shows the previous path and the location of the robot code, which runs on the robot's nRF52840DK. The properties, as shown in Figure 7.6, reveal that the folder comprises 14 261 files and 6183 folders. Navigating through this extensive folder structure can be extremely challenging without proper instructions, and even with knowledge of the project, it requires a total of seven mouse clicks before accessing the actual code.

:(C:) > Users > ettran > Documents > SLAM-Project > nrf5sdk_thread > examples > thread > freertos_mqttsn > pca10056 > blank > ses

Name	Date modified	Type	Size
.vs	16.01.2023 11:52	File folder	
.VSCodeCounter	16.01.2023 11:52	File folder	
Output	19.01.2023 14:29	File folder	
app.inc	16.01.2023 11:52	INC File	0 KB
ControllerTask	16.01.2023 11:52	C Source File	16 KB
ControllerTask	16.01.2023 11:52	C Header Source F...	1 KB
DBSCAN	16.01.2023 11:52	C Source File	10 KB
DBSCAN	16.01.2023 11:52	C Header Source F...	1 KB
defines	16.01.2023 11:52	C Header Source F...	2 KB
encoder	16.01.2023 11:52	C Source File	3 KB
encoder	16.01.2023 11:52	C Header Source F...	2 KB
encoder_with_counter	16.01.2023 11:52	C Source File	9 KB
encoder_with_counter	16.01.2023 11:52	C Header Source F...	2 KB
example_task	16.01.2023 11:52	C Source File	4 KB
example_task	16.01.2023 11:52	C Header Source F...	1 KB
ExtendedKalmanFilter	16.01.2023 11:52	C Source File	6 KB
ExtendedKalmanFilter	16.01.2023 11:52	C Header Source F...	1 KB
flash_placement	16.01.2023 11:52	XML Source File	5 KB

Figure 7.5: Path to the code concerning the robots nRF52840DK



nrf5sdk_thread	
Type:	File folder
Location:	C:\Users\ettran\Documents\SLAM-Project
Size:	1.33 GB (1 434 822 468 bytes)
Size on disk:	1.36 GB (1 460 695 040 bytes)
Contains:	14 261 Files, 6 183 Folders

Figure 7.6: Properties of the folder nrf5sdk_thread

In Andersen’s master thesis [1], he explains how he has implemented the MQTT-SN client for the nrf52840-robot which was based on two example projects: *Thread MQTT-SN* and *FreeRTOS CoAP*. The SDK that were used were the version *NRF5SDK for Thread and Zigbee version 4.1.0*. From these two example projects, the *freertos_mqttsn*-project was made.

Since the folder project for *freertos_mqttsn* has the same structure as any example project in Nordic’s SDK, and Andersen states that he used two example projects, it was assumed that one of the example projects Andersen mentioned were copied and integrated with the other to make the project folder *freertos_mqttsn*. Given that the assumptions are true, the project folder *freertos_mqttsn* can be kept and everything else can be deleted.

Even if the *freertos_mqtt* folder remains and most of the unused files are removed, there are still unnecessary files inside this folder. Inside the project folder *freertos_mqtt*, there are two PCA folders. PCA100xx indicates which board the software is made for. On the nRF52840DK, the board number indicated is PCA10056. Meaning the remaining PCA folders can be deleted.

7.2.1 Modified SDK

It was later discovered that the SDK used in this code project was modified and the refined code project was not compatible with the downloaded SDK version from Nordic Semiconductor. It is generally bad practice to modify standard libraries and should be avoided as it can cause different obstacles such as:

- **Compatibility:** Libraries are extensively tested. Modifying them can introduce compatibility issues, making the code incompatible with other systems or versions.
- **Stability:** Standard libraries are typically reliable and stable, as they have undergone rigorous testing and bug fixing. Modifying them may introduce bugs or errors, compromising the stability and reliability of the code.
- **Maintenance:** Modifying standard libraries makes it harder to maintain and update the code in the future. When a new version of the library is released, it becomes challenging to incorporate the updates and bug fixes if the code has been modified.
- **Code readability and collaboration:** Modifying standard libraries can make the code less readable and harder to understand for other developers. It can also hinder collaboration, as team members may have different versions of the modified libraries, leading to confusion and compatibility problems.

Due to time constraints, the modified SDK are still available and are the ones compatible with the robot code. The robot code has not been modified to fit one of the existing SDKs that Nordic Semiconductor's have available on their website.

7.2.2 The final structure of the robot code

After breaking down all the files and folders inside the project folder *nrf5sdk_tread*, one could separate the folder into two separate folders: the robot code and the modified SDK, which is compatible with the robot code. It was, however, decided to separate into two folders, but both folders share the same root folder.

It should be noted that restructuring and moving files and folders in the explorer will not change the path of the code project. This must be manually done in the IDE: SEGGER Embedded Studio. Every single file the robot code uses must be manually dragged and

dropped to ensure the right path. The current number of files the robot code uses is 298 files. To match the path of the new folder structure, this must be done in the IDE as well.

Figure 7.7 shows the new folder while Figure 7.8 shows the path for the robot code. Note that the code project folder is named `robot_code` with the symbol underscore, while the root folder is named `robot-code` with the symbol hyphen.

Initially, it was not necessary to have the robot code inside another `robot_code` folder, but the compatible SDK used had modifications. It was therefore decided to have the SDK inside this folder to ensure the right SDK were used when running the project.

In addition to restructuring the robot code files, names were changed to be more descriptive. All file names are written in `snake_case` and the file name that opens the software in the IDE, are changed from `thread_freertos_coap_server_pca10056m` to `robot_code`.

« (C:) » Users » ettran » Documents » robot-code »

Name	Date modified	Type	Size
.git	14.04.2023 16:07	File folder	
robot_code	14.04.2023 16:07	File folder	
SDK	28.03.2023 13:47	File folder	
SetupGuide	28.03.2023 13:47	File folder	
.gitignore	28.03.2023 13:47	Text Document	1 KB
license	28.03.2023 13:47	Text Document	1 KB
nRF_MDK_8_27_0_IAR_NordicLicense	28.03.2023 13:47	Windows Installer ...	2 340 KB
nRF_MDK_8_27_0_Keil4_NordicLicense	28.03.2023 13:47	Windows Installer ...	3 268 KB
README	28.03.2023 13:47	Markdown Source...	5 KB

Figure 7.7: New folder structure for the code concerning the robots nRF52840DK

:(C:) > Users > ettran > Documents > robot-code > robot_code >

Name	Date modified	Type	Size
.vscode	14.04.2023 16:07	File folder	
armgcc	28.03.2023 13:47	File folder	
config	28.03.2023 13:47	File folder	
iar	28.03.2023 13:47	File folder	
Output	28.03.2023 13:47	File folder	
src	14.04.2023 16:07	File folder	
app.inc	28.03.2023 13:47	INC File	0 KB
DBSCAN	28.03.2023 15:22	C Source File	9 KB
DBSCAN	14.04.2023 16:07	C Header Source F...	1 KB
defines	14.04.2023 16:07	C Header Source F...	2 KB
encoder	14.04.2023 16:07	C Source File	3 KB
encoder	14.04.2023 16:07	C Header Source F...	2 KB
encoder_with_counter	14.04.2023 16:07	C Source File	9 KB
encoder_with_counter	14.04.2023 16:07	C Header Source F...	2 KB
extended_kalman_filter	14.04.2023 16:07	C Source File	6 KB
extended_kalman_filter	14.04.2023 16:07	C Header Source F...	1 KB
flash_placement	28.03.2023 13:47	XML Source File	5 KB

Figure 7.8: Path to the code concerning the nRF52830DK robot

7.3 Documentation of the project (wiki)

Knowledge about the project has been passed through ZIP-files, and each year the number of ZIP-files has increased. In addition, there is currently no system that categorizes the different project and master's theses and usually, only the latest project/master's theses are used as the base knowledge for next year's students.

The documentation, called wiki, consists of a summary of all continuous systems and projects on the SLAM robot project. It serves as a "wikipendium" for the project. The previous students on the project expressed that it was difficult to read about the project by reading different projects and master theses without knowing which one gives an overall understanding of the project or focuses on the parts their interested in. It was therefore decided to have a page where one should list the relevant project and master theses and point out relevant topics they cover.

Additionally, the documentation covers several topics and information regarding the SLAM Robot project. All documentation is available for the GitHub organisation SLAM-Robot Project members but can be seen in the appendix K. I have contributed to the following documentation:

- Robot HW
- Robot SW

- Previous work
- Sensor Data Acquisition Board
- How to create new nRF software project method 1
- How to create new nRF software project method 2

7.4 Name conventions

Early on, the four students collectively agreed that implementing a naming convention would benefit the project. M. Ruud-Olsen had the main responsibility to propose a naming convention while the modifications on the code itself were done in collaboration. The workload was divided between two persons to minimize fault and review each other's changes.

Most of the names in the original code base have informal names. However, there is no clear distinction or differentiation on what is a task, function, queue etc. For example, the tasks *mapping_task* and *vNewMainPoseEstimatorTask* in the code base are both FreeRTOS tasks, but they follow different naming schemes. Similarly, *ir_measurement_queue*, *scanStatusQ* and *queue_display* are all queues, but here one can observe three different types of the naming scheme concerning queues. As described in section 3.4.1, having different naming schemes might not be confusing for the one writing the code, but it can be enormously confusing to a new programmer who reads it later.

The naming conventions for the robot code are the following:

- xName for semaphores and mutexes
- gName for global variables
- pName for pointers
- qName for queues
- Snake case (*snake_case*) for functions and structs
- CamelCase (*camelCase*) for name variables
- Definitions are in uppercase (UPPERCASE)

This naming convention has been adapted and implemented in the current robot code. The work was divided between M. Ruud-Olsen and the author of this thesis.

Results

This chapter will present the results regarding the hardware testing, the second revision of the PCB, the software design, and the improvement and structuring of the robot project.

8.1 Results: Hardware testing

The results from the hardware testing were recorded in the utilized test plan. Figure 8.1 shows the final test results. The tests that passed are highlighted in green and marked with **PASS**, while the tests that failed are highlighted in red and marked with **FAIL**. The other tests that were not tested are highlighted with grey and marked with **NOT TESTED**.

To comprehend the reasoning behind the failed test case *Communication (SPI)*, additional debugging were conducted. A debug list were established to debug the SPI issue. As described in the comments in the test plan, the MISO-line where the signal that was available when measuring SPI with an oscilloscope. It was later seen during a layout review of the PCB that the pads concerning the other lines were not connected to the SoC. Additionally all signal lines that were connected to the SoC with other layers that the top layer were affected.

During the testing phase, other faults were identified. Specifically, the motor driver had an incorrect pin count, and the measurements for the design space of the connector were inaccurately.

The test results from the test plan and additional physical review resulted in a second revision of the PCB to correct the faults discovered.

What	How	Equipment	Comments	Pass/Fail
5 V buck regulator	Solder on components for the 5V buck regulator. Deliver 12 V to input. Measure stable 5 V from output. Solder on 5V jumper.	Soldering Iron, microscope, solder, flux, multimeter		PASS
3v3 linear regulator	Solder on components for the 3v3 regulator. Deliver 12 to input. Measure stable 3.3V from output. Solder on 3V3 jumper.	Soldering Iron, microscope, solder, flux, multimeter		PASS
MCU0	1. Solder on components necessary for MCU0 to function. <i>Optional : circuit concerning 2nd XTAL and RF.</i> 2. Check for shortcircuit around the MCU. 3. Connect to MCU0 with debugger.	Soldering Iron, microscope, solder, flux, multimeter, nRF debugger	Did not solder the circuit concerning the 2nd crystal and RF circuit	PASS
Program MCU0	Set a pin high and measure.	computer, usbA to microUSB cable, nRF52840DK(debugger), 10pin cable from debugger to debug connector, multimeter	Set pin high for one of the IR pins and measured voltage(3.3 V) over the high pin. The pins that were not set as high measured 0V	PASS
Communication (SPI)	Write code for SPI. Set MCU0 as master. Test SPI to a device.	Computer, Oscilloscope, usbA to microUSB cable, nRF52840DK (debugger), 10pin cable from debugger to debug connector	Used a development kit type nRF52840DK for test support. Used SPI code from SDK exmple. MISO line are the only available signal that are measured with the oscilloscope. Reviewing the layout discover that theres a fault in the layout design, such that the signals : system clk, MOSI and CS are not connected to the pads on the SOC	FAIL
MCU1	1. Solder on components necessary for MCU1 to function. <i>Optional : circuit concerning 2nd XTAL and RF.</i> 2. Check for shortcircuit around the MCU1. 3. Connect to MCU1 with debugger.	Soldering Iron, microscope, solder, flux, multimeter, nRF debugger		NOT TESTED
Program MCU1	Set a pin high and measure.	computer, usbA to microUSB cable, nRF52840DK(debugger), 10pin cable from debugger to debug connector, multimeter		NOT TESTED
Communication (SPI) between both SoC on board	Write code for SPI. Set MCU0 as master and MCU1 slave.	Computer, Oscilloscope, usbA to microUSB cable, nRF52840DK (debugger), 10pin cable from debugger to debug connector		NOT TESTED

Figure 8.1: Testplan with results

8.2 Results: The second revision of the PCB

The modifications on the second revision were done according to the tasks described in chapter 5. Figure 8.2 shows the final design of the second revision of the Sensor Data Acquisition board. The second revision of the design closely resembles the first revision in terms of visual appearance. It retains its original design layout, with modules occupying the same positions. While some modules, such as motor driver, IR and LEDs have been resized and modified.

In the second revision, the silk layer maintains its emphasis on user-friendliness. Every component and module has a title. If there is only one component in a module the titles are shared. The schematics of the second revision can be found in the appendices E, F, G and H. The layout design of the four layers were presented in section 5.2 shown in Figure 5.5, Figure 5.6, Figure 5.7 and Figure 5.8.

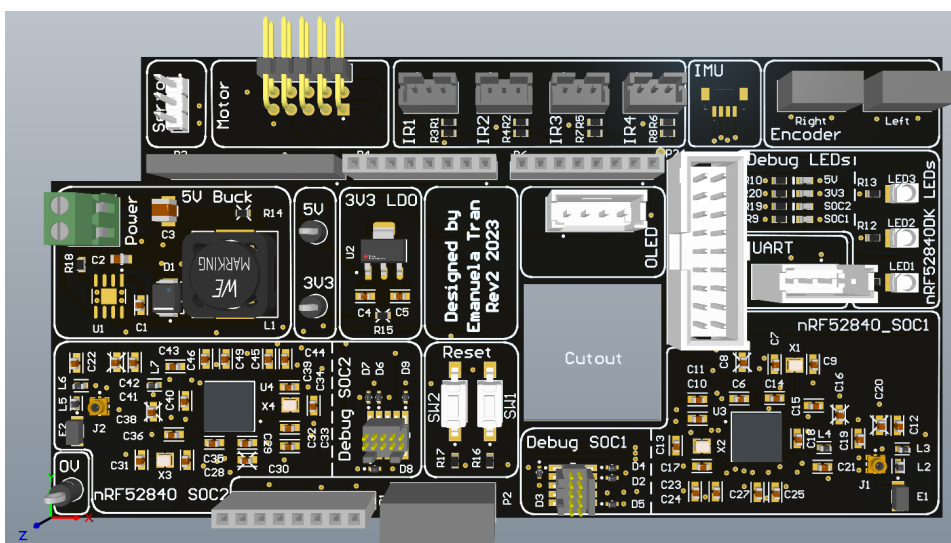


Figure 8.2: Layout revision 2: 3D version with components

8.3 Results: Software design

A FreeRTOS software project has been established to develop further software for the new system, where the proposed design was described in section 6.3. The software project is available on the project's GitHub organisation: *SLAMRobotProject*. The software project pushed to the GitHub organisation uses the method 2, which was described in section 6.4. FreeRTOS and all drivers provided by nRF5 SDK are available and can be included and used. The code project *ir-sensor-tower* in the GitHub repository have been used for testing the custom PCB, where the latest code show cases the test case for SPI.

8.4 Results: Improving and structuring the robot project

The work done by the author of this thesis to improve and structure the robot project are the following:

- A GitHub organisation has been established to host the latest code from the project.
- The latest code projects are pushed to the GitHub organisation are available.
- The code on the robot's nRF52840DK has been restructured such that all unnecessary files have been deleted and the compatible SDK are available in the root folder of the robot code.
- Naming conventions has been established and adapted with collaboration with M. Ruud-Olsen.
- Documentation on the overall project, the system overview of the Sensor Data Acquisition Board, tutorials and information regarding previous work has been written.

The details of the work done to improve and structure the robot project has been described in chapter 7.

Chapter 9

Discussion

This chapter will discuss the method and results around the topics hardware testing, the second revision of the custom PCB, software design, and improvement and structuring of the robot project.

9.1 Hardware testing

The test plan developed in E. Tran's project thesis was the foundation for the testing process. However, after carefully reviewing the test plan, several adjustments and modifications were made to align it with the current state of the hardware. Testing the SoC was challenging as I was not familiar with using a development kit to program a SoC. Due to this, I assumed that the testing would take longer than usual. Programming the SoC involved a lot of trial and error. The setup was complex and not necessarily intuitive. If something went wrong, it was not always clear whether it was a user error or a design flaw.

During the testing of SPI, it was challenging to determine what to test to identify the source of the error. Therefore, I paused the testing and created a debugging checklist outlining what I believed would be the most logical tests to conduct before concluding. This approach allowed me to eliminate and isolate the problem.

The decision to interrupt testing was made to allocate time for new hardware and software development. If I had more time, even though I knew that the pins on SOC2 were not correctly designed, I would have tested those that I knew might work to ensure that the circuits I designed functioned as expected. In addition to the test cases specified in the test plan, a visual inspection was conducted. In hindsight, I could have considered including it in the test plan.

From a personal perspective, the presence of a test plan and a place for documenting results and equipment requirements during testing was reassuring. It offered a clear roadmap for conducting tests with efficiency, enabling me to follow steps without any unnecessary time wastage.

9.2 The second revision of the PCB

The need for a new revision emerged due to the test results. By prioritizing the different tasks, it became easier to allocate time effectively. It is easy to become too detail-oriented and strive for perfection when making changes. The prioritization list was formulated based on how critical each modification was. If there was a time constraint, the least prioritized tasks that did not directly affect the circuit's functionality would be disregarded. However, it is essential to note that even low-priority tasks are still necessary and should be considered, as they contribute to the overall user experience.

The future work list, from E.Tran project thesis[4], mentioned in section 2.4 has also been considered when designing the second revision. The components regarding the nRF52840 SoCs have not been changed to package 0402, as the first revision of the board was not affected by component package 0603.

The connector's net names were rearranged to resolve the problem regarding the motor driver connector's pin mapping. This approach was the simplest and most effective. Another solution is looking for a connector with the correct pin mapping, but this was not done to avoid the risk of selecting the wrong connector or searching for a new one.

Additional notes were added to the schematics to provide more precise explanations for future students who will study the schematics for further system development or as inspiration for designing a new system. It can be more challenging to extract information solely from a master's thesis and study the schematics compared to having concise explanations directly in the schematics. The overall changes in the schematics offer a more explanatory and intuitive understanding of how the board functions and the purpose of various signals.

Regarding the layout changes, the main focus was only on making the necessary changes. It can be tempting to adjust everything during the layout process, and having a prioritization plan proved helpful. Occasionally, minor imperfections can be observed, but they are often not crucial faults that affect circuits and can be considered nitpicking. For example, the length of a trace may be shortened to enhance the aesthetics of the circuit, but it may not significantly impact performance. It is often required to reroute traces when new components are imported or reallocated. In Altium Designer, moving the components and the connection traces is possible. However, given the modified component placement, the traces were rerouted manually to ensure the circuit's accuracy.

9.3 Software design

While it is possible to begin a software project by simply diving into code writing, in this case, I recognised the significance of outlining specifications and requirements for the system. Planning a project is essential as it establishes a solid foundation for future work. It makes the implementation phase much easier and organised and facilitates smoother integration with other systems in the robot project. Since the new system's objective is to take over specific tasks performed on the robot's nRf52840DK, I acknowledged the importance of analysing how the task is currently executed. This involves comprehending the task's actual functionality and interactions with other tasks on the robot.

When developing a software project, two different approaches were explored. Without clearly understanding which method was the most correct or efficient, the initial option was to build the project from scratch, which might seem more intuitive. However, the first method posed challenges in debugging and ensuring the availability of all the desired packages.

On the other hand, method 2 relied on an example project. Following the provided guide resulted in a project not being integrated with the example projects that the robot code had previously relied on. I found method 1 to be more intuitive, but method 2 proved more straightforward to work with and made it easier to add drivers and libraries and run FreeRTOS.

Due to time constraints, further development on the Sensor Data Acquisition Board software project was not performed. This has been left for future work.

Both methods on how to create software project are presented so that future students can choose how they create their projects. However, the new students must decide which method to establish a standard for all new projects. This ensures maintainability and facilitates collaboration among students working on different projects, enabling them to support each other with code if needed.

As presented in section 6.2.2, FreeRTOS was chosen as the recommended RTOS for adaptation. This decision was based on the fact that FreeRTOS is the RTOS utilized in the robot project. However, it should be noted that this choice also ensures consistency across various systems and allows for knowledge sharing.

9.4 Improving and structuring the robot project

The efforts invested in improving and structuring the SLAM robot project have resulted in a more structured project. It is important to emphasise that this work is the first step towards making the project easier to start, understand, develop and document.

It is difficult to distribute the work in this project since no project management or leader is present. Generally, the individuals who take the initiative in a group usually do the

most work. Some tasks naturally fall on different individuals. Despite my specific focus on this project, I have contributed to tasks that may not directly relate to my area of expertise, but I have made an effort to assist where it was needed. An example of taking the initiative was when I suggested creating a GitHub organisation for the project. As a result, it naturally became my responsibility to take charge of that task.

Since I created a new software project from scratch, I was familiar with how the IDE and its settings worked, making it easier for me than for others to delve into cleaning up the robot code structure. However, it was more challenging to participate in the clean-up and implementation of the name conventions for the robot code.

Regarding the project's documentation, known as the wiki, I contributed what I believed necessary. However, upon reflection, the information I selected and wrote may not be the most relevant or comprehensible for most students participating in the project. The relevance and clarity of the documentation depend on the individual's background and area of focus. Looking back, I should have reached out to my fellow student on the project for their insights on my topics and how I could have better presented it. There may have been aspects that I overlooked or assumed were common knowledge but needed further explanation.

The effort to clean up the project has not directly impacted my work during this master's thesis, but it has provided me with insights into the structure and planning required for integrating new systems. It became even more apparent to me that integrating a new system would require a significant effort after working with and becoming familiar with the codebase of the robot code. Currently, it is not possible to efficiently integrate the new system I have been working on. It is unfortunately not sufficient to write a few drivers or lines of code; it essentially requires restructuring the entire robot code.

Chapter 10

Future work

This chapter will present the future work for this project and is divided into two parts. The first part will present the future work concerning the system Sensor Data Acquisition Board, while the second part will present the overall future work concerning the robot project.

10.1 Future work for the system Sensor Data Acquisition Board

Although the design of the second revision of the Sensor Data Acquisition Board is finished, reviewing the board before producing it is recommended. If all the components are accessible at the production site, it is recommended to order PCBs with pre-soldered components. However, in cases where specific components are unavailable, the mechanical workshop at the Department of Engineering Cybernetics offers a reflow oven as an alternative. This allows soldering components onto the PCB without the requirement for hand soldering. Unless the person performing the soldering is experienced and confident in soldering and desoldering the nRF52840 package, using the reflow oven for soldering purposes is not advisable. The footprint of the nRF52840 package presents challenges that require specialized skills for successful soldering and desoldering.

Additionally, further testing of the first revision of the board such be considered. Circuits that are more likely to work should be tested to ensure the correct circuit design before the second revision goes into production.

A test plan should be created for the second revision after its production. If all components are soldered at the production site, it is recommended to leave the unsoldering jumpers(zero-ohm resistor). Those components that should not be soldered are marked

with an X on the board as seen in Figure 10.1. That way, it is possible to test the 5 V and 3.3 V without harming other circuits if they are not soldered correctly.

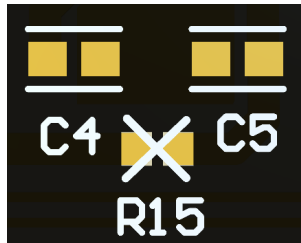


Figure 10.1: Jumpers/zero-ohm resistor are marked with a cross

The software development for the IR-sensor tower can begin as it is possible to accomplish this without producing a second version of the system. The nRF52840DK and the custom PCB utilize the same SoC. To enhance the efficiency of the software development and testing, it is recommend to create a test plan. This will ensure the coverage of all software features, facilitating smoother development and testing procedures.

Despite the development of the software for the new system, challenges may arise during the integration process with the robot. The robot code must also be modified so the new system can be integrated. It is not advisable for a new student who receives this system as a project and master's thesis to immediately undertake tasks such as producing a new revision, conducting extensive testing, developing code, and modifying the code on the robot.

The workload associated with these activities is significant, and it is essential to gradually approach the development of a new system to establish a solid foundation and ensure its success. Moreover, the current state of the robot code poses challenges when it comes to integrating SPI, which is essential for communication between the robot and new systems. Given the project's structure, it is anticipated that comprehending and modifying the current robot code will present significant challenges in addition to mastering a new system.

10.2 Future work for the robot project

The work described in chapter 9.4 highlight the work undertaken to improve and re-structure the robot project. However, further work is required to achieve a thoroughly structured project. This section will present the recommend future work regarding the robot project.

10.2.1 Documenting the work on the "wiki"

All efforts to make the " wiki " documentation will only be valuable if further documentation is done. It is recommended that the new students on the project continue and uses the documentation made. By actively updating the "wiki", it will ensure that information about the project will not be lost. Next year's students can decide whether to update it regularly or do this at the end of the semester.

10.2.2 Name conventions

The name conventions presented in section 7.4 should be continued when further developing the robot code. Other code projects that are written in the programming language C should implement this name convention. Creating new naming conventions for other software projects should be considered, as they are written in different languages.

10.2.3 The use of GitHub

As described in section 7.1, all known code bases had been pushed to the project's own Github Organization. While having the entire code base available on GitHub may seem organized, without proper rules and regulations for its usage, it can quickly devolve into chaos.

It is recommended that the new student taking over the project sit down and establish some rules and guidelines regarding the use of GitHub. Here are some rules that should be considered:

- **Branching strategy:** Define a branching strategy that outlines how branches should be created, named, and merged.
- **Code review:** Require code reviews before merging any changes into the main branch. This helps ensure code quality, catch errors, and share knowledge among team members.
- **Pull request guidelines:** Establish guidelines for creating pull requests, including a description of the changes, any required tests or documentation updates, and the reviewers who need to approve the changes.

- **Access control:** Define who can push changes directly to specific branches. Consider limiting direct push access to the main branch and requiring pull requests for other branches.
- **Commit messages:** Encourage descriptive and meaningful commit messages that clearly explain the purpose and impact of the changes.

10.2.4 Modularisation

Modularisation is crucial in large projects that involve multiple components and multiple users working on them. In this context, *components* are defined as both software components within a project and components in terms of applications in software. This implies that different systems have defined interfaces, allowing their functionality to be changed without affecting other systems. In this project, there is a lack of modularity. For instance, there are many dependencies among different components of the robot. This makes it difficult to understand and challenging to modify existing elements. Changes can potentially cause other systems to stop functioning correctly.

The following are the steps that should be taken to achieve modularity in the project:

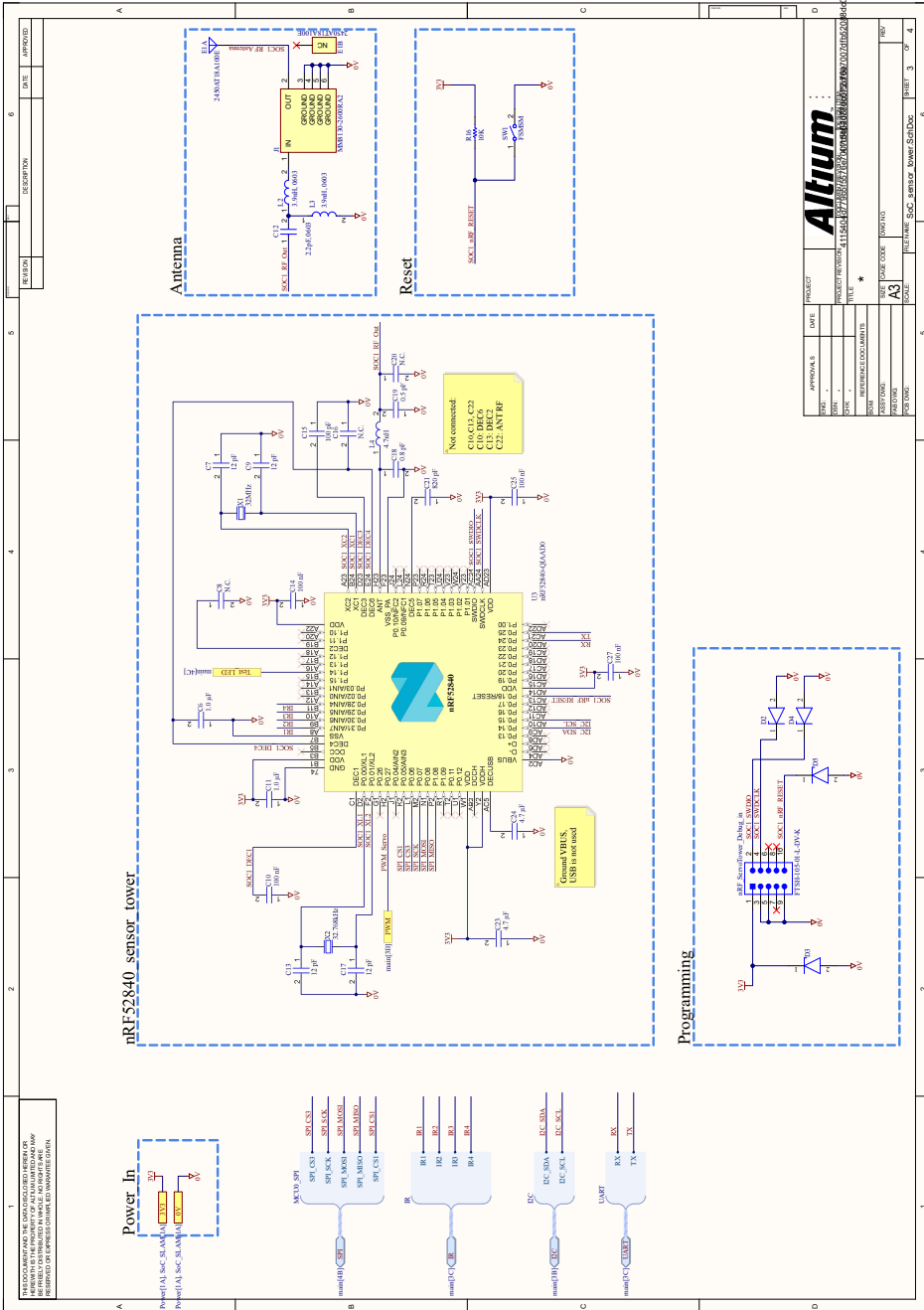
1. Establish a shared codebase on Github that contains low-level drivers used by all boards, which will standardize the usage of all components.
2. Each component within a system should be clearly divided. For example, all communication should be consolidated into a module with a well-defined interface. This makes it easier to add new components that can utilize the existing interface without modifying other components.
3. All systems should have clearly defined responsibilities. For instance, all external communication with the server can be delegated to a separate board with a clear interface that other boards/systems can utilize. This allows for easier integration of additional sensors or new systems in the future.
4. To achieve this, communication within the robot should also be modified. The suggestion to use the communication protocol Controller Area Network (CAN) were introduced in E.Trans project thesis [4].

Bibliography

- [1] T.Andersen. *Sparse IR sensor EKF-SLAM for MQTTSN/Thread connected robot*. NTNU Trondheim, 2022.
- [2] SEGGER Microcontroller. J-link ob, . URL <https://www.segger.com/products/debug-probes/j-link/models/j-link-ob/>.
- [3] H.Frestad. *The SLAM-Project*. NTNU Trondheim, 2022.
- [4] E.Tran. *Sensor data aquisition board for nRF52840 maze mapping robots*. NTNU Trondheim, 2022.
- [5] John Catsoulis. *Designing Embedded Hardware (2nd edition)*. O'Reilly.
- [6] Piyu Dhaker. Introduction to spi interface, 2018. URL <https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf>.
- [7] Altium Designer. Defining blind, buried micro vias in altium designer. URL <https://www.altium.com/documentation/altium-designer/blind-buried-micro-vias>.
- [8] E.Jølsgård. *Embedded nRF52 robot*. NTNU Trondheim, 2020.
- [9] Nordic Semiconductor. nrf52840dk, . URL <https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk>.
- [10] Handson Technology. L29n dual h-bridge motor driver. URL <http://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf>.
- [11] Machifit. Machifit 25ga370 dc 12v micro gear reduction encoder motor. URL https://www.banggood.com/Machifit-25GA370-DC-12V-Micro-Gear-Reduction-Encoder-Motor-with-Mounting-Bracket-and-Wheel-p-1532242.html?cur_warehouse=CN&ID=6157423.

-
- [12] Sharp Electronics. Sharp gp2ya21yk0f datasheet. URL https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf.
- [13] DGServo. Servo - generic metal gear (micro size). URL <https://www.sparkfun.com/products/14760>.
- [14] InvenSense. Icm-20948. URL https://www.elfadistelec.no/Web/Downloads/_m/an/SEN-15335_eng_man.pdf.
- [15] E.Jølsgard. *Shield for embedded nrf52840-DK robot*. NTNU Trondheim, 2020.
- [16] SEGGER Microcontroller. J-link, . URL <https://www.segger.com/products/debug-probes/j-link/>.
- [17] SEGGER Microcontroller. Segger embedded studio, . URL <https://www.segger.com/products/development-tools/embedded-studio/>.
- [18] Nordic Semiconductor. nrf5sdk documentation 17.1.0, 2023. URL https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_sdk%2Fstruct%2Fsdk_nrf5_latest.html&cp=9_1.
- [19] Steve McConnell. *Code Complete (2nd edition)*. Microsoft.
- [20] Nordic Semiconductor. nrf5sdk documentation, spi master example, 2021. URL https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsdk_nrf5_v17.1.0%2Fspi_master_example.html&cp=9_1_4_6_37.
- [21] Nordic Semiconductor. nrf5sdk documentation, spi slave example, 2021. URL https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsdk_nrf5_v17.1.0%2Fspi_slave_example.html&cp=9_1_4_6_39.
- [22] Nordic Semiconductor. Circuit board guideline for aqfn package, . URL https://infocenter.nordicsemi.com/pdf/nan_040.pdf.

G Schematic revision 2: sensor_tower sheet



I Tutorial on creating a new software project method

1

1. Start Embedded Studio.
2. (*optional*) install CPU support Package for your device family via Tools → Package Manager. Nordic Semiconductor nRF CPU Support Package were used here.
3. Create new project via File → New Project → Create the project in a new solution.
 - (a) Set a project name. If no CPU package is used then select "A C/C++ executable for a Cortex-M processor". Press Next, select your target device and keep pressing Next until your project is finished.
 - (b) If you are using a CPU support package select the "A C/C++ executable for..." from the corresponding package and finish your project as described in the point above.
4. Connect the nRF52840DK to a PC. Ensure that both the Jlink on the development board and the nRF52840 are powered, not just the nRF52840.
5. Give your newly created project a try by building it with F7 and executing it with F5.
6. Create a new folder /lib/FreeRTOS/FreeRTOS-Kernel in both the project explorer in Embedded Studio and on your hard drive in the project folder.
7. Download and unpack the FreeRTOS software to any location.
8. In the unpacked folder open folder /FreeRTOS and copy the folder /FreeRTOS/-Source to the /lib/FreeRTOS/FreeRTOS-Kernel in your ES project folder.
9. Add the same files to the Embedded Studio project explorer. The easiest way is to drag and drop the folder onto the /lib/FreeRTOS/FreeRTOS-Kernel folder.
10. Right click the new folder and select Setup. Check the box "Recurse into Subdirectories" and press OK.
11. This should add all FreeRTOS sources to your setup. However not all files are needed so the wrong files have to be removed again. To do this first convert the folder to a regular folder by right clicking it and select "Convert to regular folder".
12. You will need all .c source files from the /Source folder.
All include files from /Source/include and the folders /Source/portable/GCC and /Source/portable/MemManage.
13. All other folders and files can be safely removed by simply selecting them and pressing the DEL key our right click and delete.

-
14. Next make sure that in `/Source/portable/MemManage` you only have one `.c` file selected e.g. `heap_1.c`. Remove all other `.c` files, otherwise the project will later not build.
 15. In `/Source/portable/GCC` make sure that only the folder is included that is the architecture of your target platform. In our example it is a Cortex-M4 target device so only folder `/Source/portable/GCC/ARM_CM4F` stays. All other folders can be safely removed as before.
 16. Next you will need to create a `FreeRTOSConfig.h` file which will configure your FreeRTOS setup. For references see the FreeRTOS documentation or use one of the config headers from the many samples out there as reference. For example the one from the example project above. We recommend to place this file into your source folder where your `main.c` file is. In this case it is folder `/source` in the project folder.
 17. Next all include paths need to be set. You can add this in project options under `Project` → `Options` → `Preprocessor` → `User Include Directories`. If you are using the same folder structure as recommended the following three include paths must be set. If you are using another project structure adjust the paths accordingly.
 - (a) `$(ProjectDir)/source`
 - (b) `$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/include`
 - (c) `$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/portable/GCC/ARM_CM4F`
 18. Now edit your `main.c` to include `FreeRTOS.h` and `task.h` and add your FreeRTOS application code to the `main.c`.
 19. (*optional*) add your third party libraries, HALs, drivers etc. to your project by including their paths. In this project the nRF SDK folder is outside of this project folder. The paths for the nRf SDK therefore start with `../..`. Your paths should look something like this:
 - (a) `$(ProjectDir)/source`
 - (b) `$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/include`
 - (c) `$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/portable/GCC/ARM_CM4F`
 - (d) `../nRF5_SDK_17.1.0_ddde560/modules/nrfx`
 - (e) `../nRF5_SDK_17.1.0_ddde560/components/boards`
 - (f) `../nRF5_SDK_17.1.0_ddde560/components/libraries/util`
 - (g) `../nRF5_SDK_17.1.0_ddde560/modules/nrfx/hal`
 - (h) `../nRF5_SDK_17.1.0_ddde560/modules/nrfx/drivers`
 - (i) `../nRF5_SDK_17.1.0_ddde560/integration/nrfx/legacy`
 - (j) `../nRF5_SDK_17.1.0_ddde560/modules/nrfx/templates`
-

(k) `../nRF5_SDK_17.1.0_ddde560/modules/nrfx/templates/nRF52840`

20. Once all this is done your application should build now and you should be able to debug a FreeRTOS application in Embedded Studio.

J Tutorial on creating a new software project method 2

This section covers the second method on how to make a new project. This method are based on nRF SDK's example project.

Step 1: Download nRF5 SDK from Nordic Semiconductor

- <https://www.nordicsemi.com/Products/Development-software/nrf5-sdk/download>
- The version used for this project is 17.1.0

Step 2: Make a copy of the downloaded folder and rename to desired project name

1. In the copied folder go click into:
 - examples → peripheral → blinky_freertos → pca10056 → blank → ses
2. The new project will be the blinky_Freertos_pca10056 (type: SEGGER Embedded Studio ARM Projct file).
3. Build and run this project to check that you have a working FreeRTOS project.
4. Have the new project folder close to the original downloaded folder, preferably in the same folder. This is due to paths and project setup later on.

Step 3: Reorganize the structure of your project in the file explorer

Modifications will be made in the copied folder (new project folder) and not in the "nRF5_SDK_17.1.0_dde560" folder. Unused files/folder will be deleted.

1. In the ../examples/peripheral folder, move the "blinky_freertos" folder to the root folder
2. In the root folder, delete all folders except for the folder names "blinky_freertos"
3. Inside the "blinky_freertos" folder delete the following folders:
 - pca10040 and pca10100
4. In the ../blinky_freertos/pca10056/blank/ses folder, move all files/folders to the root folder.
5. In the ../blinky_freertos/pca10056/blank folder, delete the ses folder and move all remaining folders/folders to the root folder.

-
6. In the `../blink_freertos` folder, delete `pca10056` folder, `hex` folder and move the remaining folder/folders to the root folder.
 7. In the root folder, delete "blink_freertos"

Step 4 (Optional): Reorganize the structure of you project in SEGGER

You are free to choose whether you want to restructure the folder structure inside your project (SEGGER)

Right click on your project and add folders.

1. For this project two folders were created: `src` and `config`.
2. The application folder was deleted.
3. Add existing files into the project by right clicking on your folder.
4. The `src` folder contains `main.c`.
5. The `config` folder contains `sdk_config.h` and `FreeRTOSConfig.h`.

Step 5: Include paths and files

Open the new project in Segger. It is likely that the build process will encounter errors since all the original files have been removed. Therefore, it is essential to configure the correct paths for the new project so that it can locate the necessary files.

1. Right click on your project and select options.
2. On the top to the left select "Common" for private configurations.
3. Include and the right paths in Preprocessor → User Included Directories.

The existing paths are referencing the SDK files that have been deleted. The new project should utilize the SDK files from the original folder that was initially downloaded.

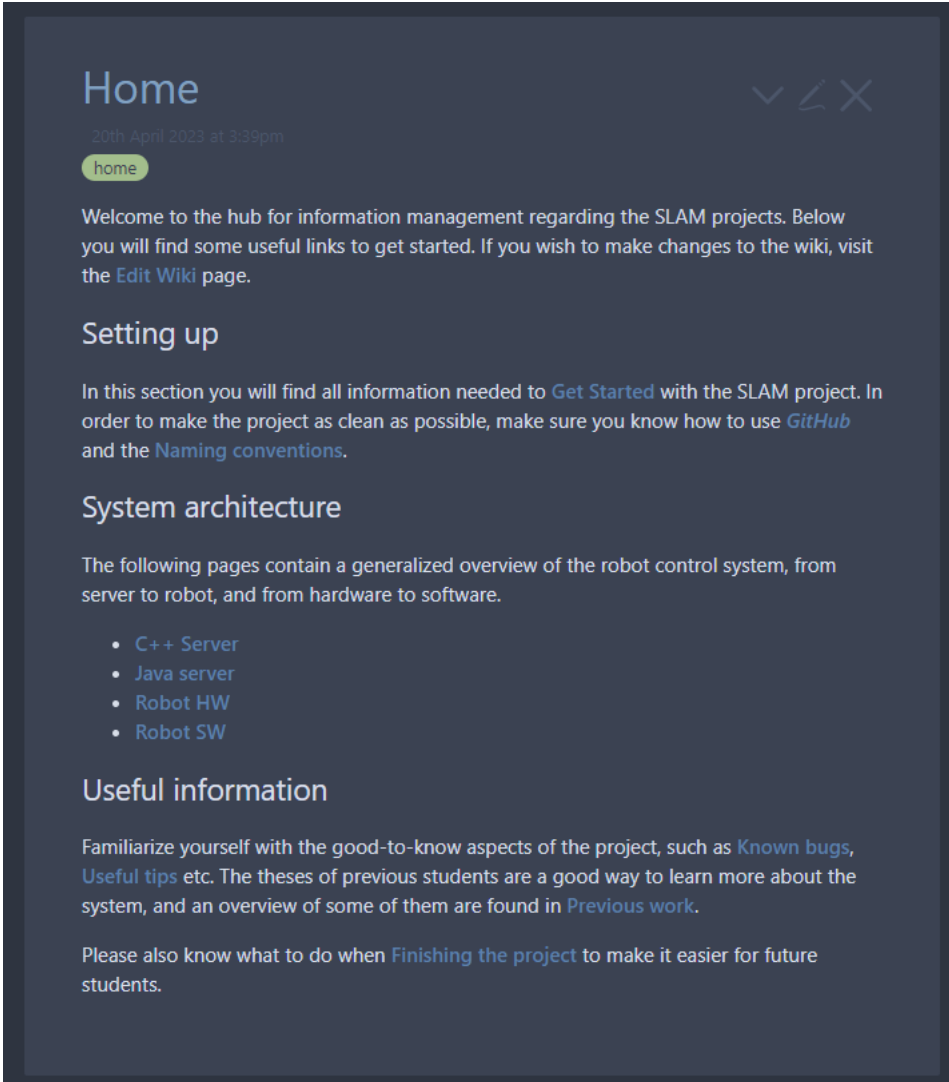
Here is an example of how the paths should appear if the root folder and the original SDK folder are located in the same directory: `../nRF5_SDK_17.1.0_ddde560/components`

After setting the correct paths, the necessary files must be included from the original `nRF5_SDK_17.1.0_ddde560` folder. The specific files that are required depends on the driver you wish to use in the project. Navigate to the original `nRF5_SDK_17.1.0_ddde560` folder and simply drag and drop the necessary files into the SEGGER project. This guide is specifically based on the `blink_freertos` example and includes the same files that are used in that project.

Compile and execute the code to verify if it functions similarly to the original `blinky_freertos` example project. If any issues arise, double-check the paths and ensure that the correct files have been included. The `sdk_config` used in this project are similar to the one used in the `blinky_freertos` example.

K Documentation (wiki)

Home



The screenshot shows a wiki home page with a dark blue background and white text. At the top left, the word "Home" is displayed in a large font, with a timestamp "20th April 2023 at 3:39pm" below it. To the right of the title are three icons: a checkmark, a pencil, and an 'X'. Below the title is a small green pill-shaped button with the word "home" inside. The main content area contains several sections: a welcome message, a "Setting up" section, a "System architecture" section with a bulleted list, and a "Useful information" section.

Home ✓ ✎ ✕

20th April 2023 at 3:39pm

home

Welcome to the hub for information management regarding the SLAM projects. Below you will find some useful links to get started. If you wish to make changes to the wiki, visit the [Edit Wiki](#) page.

Setting up

In this section you will find all information needed to [Get Started](#) with the SLAM project. In order to make the project as clean as possible, make sure you know how to use [GitHub](#) and the [Naming conventions](#).

System architecture

The following pages contain a generalized overview of the robot control system, from server to robot, and from hardware to software.

- [C++ Server](#)
- [Java server](#)
- [Robot HW](#)
- [Robot SW](#)

Useful information

Familiarize yourself with the good-to-know aspects of the project, such as [Known bugs](#), [Useful tips](#) etc. The theses of previous students are a good way to learn more about the system, and an overview of some of them are found in [Previous work](#).

Please also know what to do when [Finishing the project](#) to make it easier for future students.

Edit Wiki

Edit Wiki

13th April 2023 at 1:27pm

how-to

For making changes to the wiki, there are quite a few things to keep in mind. This page is intended to help you out by providing some information on how this wiki is structured, and some tips and tricks on how to navigate.

Making a new page

In order to make a new page, navigate to the «+» sign located beneath the title on the top right section of the screen. A blank page will appear on the left hand side. These pages are natively called «tiddlers», and will be referred to as such by the system. The new pages will be tagged as «untagged» by default, remember to remove this tag and provide a suitable tag, preferably from the existing selection to maintain a consistent overview, but in case noone suits the use properly, you can make your own. The idea is to keep it simple.

Formatting

When writing your tiddler, the toolbar on top of the text editor contain some helpful tools, but learning how to perform these things manually can also be time saving so here are some helpful tips:

- **Headers** In order to make headers, simply write an exclamation mark "!" at the start of the respective line, followed by a space and your header text. The amount of exclamation marks correspond to the header types, i.e. "!" corresponds to Header 1, "!!" to Header 2 etc.
- **Linebreak** To make a linebreak, simply hit enter two times. two linebreaks in the text editor results in a single linebreak in the output.

-
- **Fontstyles** To make text **bold**, enclose it in double apostrophes ('). For *italic* style, enclose it in double slashes (/). For underlined text, enclose it in double underscores (_). For ~~overlined~~ text, enclose it in double tildes (~).
 - **Images** To include images, firstly make sure the image is imported using the binder icon beneath the wiki title on the top right of the screen. If you can't find it there, you can find it in the Tools tab located in the same area. Make sure the image file has a simple name. Then, include it in the text by making double brackets, with "img" between the first and second bracket, followed by the filename inside the second bracket. Example [img [filename.png]] without the spacing.
 - **Bullet points** are made using the star symbol "*" followed by space and your text. Lists are not made automatically, so you will need to make a linebreak and follow up with a new bullet point manually.
 - **Links to other tiddlers** If you want to reference another tiddler, you can link it by simply encasing the title of the page in double brackets. For example, [[Home]] will create a link to [Home](#)

Saving the wiki

When you want to save your changes, make sure all the tiddlers, i.e. the "pages" you've made changes to, are saved by clicking the checkmark in the top right corner of each tiddler. To make the wiki a little less messy for the next person opening it, make sure to hit the Home button before the final save. This closes all open tiddlers except from the start page. Then, you can save the entire wiki by hitting the red circle icon on the top right of the screen, under the project title. Any unsaved tiddlers will lose their changes when you press this button. If you are working directly in the html-file, you are working offline and saving will produce a new html-file in your download folder for you to share with the project members. If you have uploaded the file to TiddlyHost, which is the online host service, the changes will be saved to your online project and you will need to download the updated project to an html file in order to share it.

Note: The system has no guarantees for your unsaved changes to be stored, and so if the browser you're working in goes idle or anything like that, the page may be refreshed and you might lose all your progress so **make sure you save often.**

Get Started

Get Started

15th May 2022 at 2:00pm

how-to

Before you start

Make sure you read through this wiki. It contains a lot of useful information and details on where to find more. There are also helpful insights from previous students that will hopefully decrease the time needed to understand the setup and code.

Familiarize yourself with the [GitHub](#) organization and its repositories. What repos are necessary for you depends on what version you are going to work on.

- C++ Server and robots
 - You will need to clone the repositories **cpp-server** and **robot-code**. A detailed guide on how to set up the server and how to flash code to the robots are found in their respective repos.
- Java server and robots
 - The server is available in **Java-Server**
 - What robot-code you want to base your project on depends on your task.
 - The code in **robot-code** is based on the work from Frestad(2022), and a naming convention was added and the file structure greatly improved spring 2023. The functionality for using the Java server was not removed, but the updated version was only tested with the C++ server. It is quite possible that this robot code will work with the Java server, but no guarantees can be made.
 - The code in **robot-code-java-server** is based on the work done by Andersen (2022), but it is without the work done by Frestad(2022) and the improvements done to **robot-code** during spring 2023. It is however guaranteed to be compatible with the Java server.

When you start your work, make sure that you work on a separate branch, and not directly into the main branch.

There are six robots available for this project, named NRF1 through NRF6. The robots have some differences in hardware. Make sure you choose the correct robot in *robot_config.h* in the robot code.

Charging the robots

- The orange covers on the wire are there to protect them. Remove these on the two free wires at the front of the robot.
- Connect the black charger plug to the black wire, and the red charger plug to the red wire.
- Flip the charger switch on the robot.
- Connect to power.

Naming Conventions

Naming conventions

14th April 2023 at 1:46pm

how-to

When beginning to work on this project, there is a lot of code to understand and read through. In addition, a lot of people have contributed to the current code with their own manner of programming. Consistent use of the naming conventions will improve the readability greatly. Note that these naming conventions, especially the delimitation, applies mainly to the robot code. The C++ server is written with snake case for everything. Whatever repo needs work done, please think through how to name things to make it easier for everyone.

Delimitation of words:

- Snake case (snake_case) is to be used for functions and structs.
 - The tasks should be prefixed with "task_". Eg. task_sensor_tower
- Camel Case (camelCase) is the default way to name variables.

To further differentiate between some key concepts an initial letter can be used on the variables:

- xName for semaphores and mutexes.
- gName is used for global variables.
- pName is used for pointers.
- qName is for queues (eg. qEncoderTicks)

Some general advice for best practice:

- Boolean variables should be named so it is easy to understand what its value means. Usually, this means including words as is, was, has and should, for example should_update and is_available. Avoid using not as that causes non-intuitive results of true and false.
- Header files ought to include
 - `#ifndef EXAMPLE_H` and `#define EXAMPLE_H` at the top of the file
 - `#endif /*EXAMPLE_H*/` at the end of the file
- `#define` constants should be capitalized
 - `#define BUFFER_LENGTH 6`

C++ Server

C++ Server

15th May 2023 at 2:03pm

architecture

The C++ server is a multithreaded process that can be run in Visual Studio 2019. Frestad (2022) contains a more detailed description of details of the different threads. Information about MQTT, MQTT-SN and the communication between the robots and the server can be found in Andersen (2020). An updated guide for how to set up the server correctly can be found in the repo **cpp-server**.

main.cpp sets the MQTT address to the Raspberry Pi. It also creates a couple of call-back functions, and is responsible for starting the five threads responsible for respectively the GUI, MQTT, robot inbox, robot outbox and simulation. The necessary data for each robot is found in the robot class in *robot.cpp*. It is *robots.cpp* that handles the updating of the different robots as new updates arrive from the inbox. *robots.cpp* also calculates the new target destination.

When running the program, a GUI window appears. Some of the important parts of this window are the possibility to choose between simulation and physical robots, register robots, start exploration of the area, and choose manual input. Note that switching between simulation and real driving also requires one line of code to be changed. As an alternative to manual input, one can also set target position by right clicking on the screen. The IR readings from the robots are displayed in the GUI as well, with possible obstacles displayed as black areas.

The simulation is handled by a separate thread. When the simulation is running, it reacts to inputs from the GUI window in the same manner a robot would. By letting the simulated robot explore, it will eventually reconstruct the predefined map it is in.

Incoming messages from the robots are registered by call-backs in the MQTT thread in *matt_handler.cpp*. If the message is from a simulation, the message is passed on to the simulation thread. Otherwise, it is passed on to the robot inbox thread, via a channel called *slam_ch*, which again forwards it to *robots.cpp* for further updating. The robot outbox thread is responsible for sending the correct new position to the correct robot and passing it on to the MQTT thread.

Choosing "Search Grid" after creating a connection to a robot will lead to the server choosing a target point and sending it to the robot. When the robot has moved, the server will add a new random path to its path. When "Enable Manual Drive" is checked, the server will send whatever value is in the Manual Input GUI slots with even intervals, roughly every second. Note that it does not have any concept of whether you finished writing your target points, and will potentially send an unfinished position, before sending the correct one.

When using the manual inputs, be aware of the coordinate frames used. The targets are written on the form (x,y) with millimetres as the unit. The robot's internal coordinate frame has the positive x-axis going straight ahead, and the positive y-axis to the left. In addition, somewhere along the line this input is multiplied by two. As a result, setting an input $(0,-300)$ will take the robot 600mm directly to the right of its starting point.

Java Server

Java server

24th April 2023 at 4:35am

architecture

The Java server is a server application that provides real time features to control the robots. By using BLE communication through nRF51 Dongle it is able to communicate with multiple robots and extract data from them, and ultimately send position commands to them. The server also contains a GUI with a map drawn from sensor data, and a simulator that can be used for testing.

It is based on a older implementation in MATLAB which was first introduced in 2005 by Syvertsen. The Java server was built in 2016 by Andersen and Rødseth. More details can be found in their report.

In order to use the Java server to control the robots, it needs to be flashed with the Java server robot code. A nRF 51 dongle with the correct hex file is also needed.

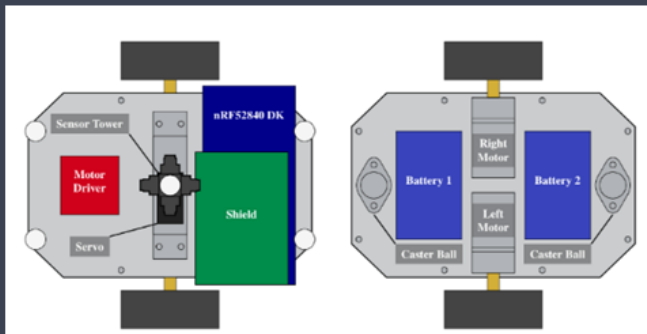
A user manual to the Java server can be found in the **SSNAR - cart/manuls** folder in the java-server repository. Master thesis' reports working with and alongside the java server can be found in the **reports/** folder in the repository.

Robot HW

Robot HW

29th May 2023 at 4:24pm

architecture



The figure above is an illustration of the robot's hardware. The left side of the figure shows the view above the robot and the right side shows the bottom side of the robot.

nRF52840DK Shield

The nRF52840 shield was custom made by Jølsgard and is attached on the top of the nRF52840 DK. The purpose of the shield is to connect all peripherals to the GPIO headers of the development kit.

nRF52840DK

The nRF52840-DK development kit from Nordic Semiconductor serves as the primary computing module for the robot. The nRF52840 System on a Chip on the development kit supports Bluetooth low energy, Bluetooth mesh, NFC, Matter, Thread, and Zigbee. It is compatible with Arduino UNO Revision 3, making it easy to use third-party shields. The development kit comes equipped with an on-board SEGGER J-Link debugger that allows for programming and debugging of both the on-board SoC and external targets through the debug header. The nRF52840 DK can be powered through a USB connection, but it also has the capability to be powered by a wide range of external sources within the 1.7 to 5 volt supply range. In addition to USB, it has a CR2032 battery holder and a Li-Po battery connector, giving it the flexibility to be powered by various sources depending on the needs of the project.

Sensor tower

The sensor tower is made up of four 2YA21 Sharp infrared sensors. These sensors are mounted on a S05NF servo motor and arranged radially with respect to each other, allowing the set of IR sensors to rotate. The sensor tower has a maximum rotation angle of 90 degree and each of the IR sensors have a valid measurement range of 0.1 to 0.8 m.

Motor driver

The L298 motor driver is a bi-directional motor driver based on the L298 Dual H-Bridge Motor Driver Integrated Circuit. According to its data sheet, it is ideal for robotic applications and can be easily connected to a microcontroller, requiring only a few control lines per motor. It is capable of controlling two motors at up to 2 A each in both directions.

DC motors + encoders

The robot uses two 12V DC-motors from Machifit. Which are rated up to 100 rpm. Each motor has built-in quadrature encoders for measuring wheel angle and speed.

IMU

The ICM-20948 Measurement Unit is made up of a 3-axis MEMS-based gyroscope, accelerometer, and compass. It is located underneath the robot's chassis and is equipped with a digital motion processor that is used for simple filtering of sensor measurements and power management. The IMU is not illustrated on the figure above.

Sensor data acquisition board

In addition to the existing hardware on the robot, there is an ongoing project that works with splitting up the hardware on the robot.

This project is under development and a custom printed circuit board has been created. Read more about the Sensor data Acquisition board here: [Sensor Data Acquisition Board](#)

Robot SW

Robot SW

29th May 2023 at 4:40pm

architecture

Tutorials on how to create new software project for a new system

The following are tutorials on how to create a new software project with *FreeRTOS*. Method 1 creates a project from scratch by including libraries. The 2nd method uses a nRF SDK example project as its base. The 2nd method is preferred as it is easier when troubleshooting.

[How to create new nrf software project method 1](#)

[How to create new nrf software project method 2](#)

Known Bugs

Known bugs

14th April 2023 at 2:08pm

bugs faq good-to-know

These are some known bugs. The robots might suddenly decide to behave unexpectedly, in this case flash the same code again and try again.

- C++ server
 - The server only sends a new target if the robot has moved. "If the robot is stuck against a wall and consequently does not move, this bug causes server to never send new targets." (*robot_outbox_thread()* in *robot_outbox_handler.cpp*)
 - When using manual input, the server will send the input to the robot roughly once every second. It is not aware of whether you are currently writing, and as a result it might send a target (0, -1) as you are trying to write (0,-100). The next time the correct target will be sent.
- Java server
 - Robots connecting to the server may occasionally disconnect. The reason for this is unknown.
- Hardware
 - NRF3: Only one of the battery packs works.
 - NRF3: The robot cannot start its program by pushing the power button. It needs a kickstart with USB on the short side of the NRF52840 and can maintain power by pushing the power button.
 - The robot will reset its program if a problem occurs during runtime. An example would be a SEGFAULT (running out of memory, out of bounds in an array or any similar errors).

Useful Tips

Useful tips

15th May 2023 at 2:06pm

faq

good-to-know

Debugging

Segger has a very useful debugging tool. Nordic extension on VS code is another way to debug the robot through USB.

If an error happens during runtime that causes the robot code to crash, it will not notify you in any way. If a motor input is set before it crashed, this input will continue, and the robot will keep moving as it did before it crashed.

The IR tower is supposed to start rotating shortly after turning the robot on. If this does not happen, it is likely due to an issue with the gateway (Raspberry Pi). Make sure that the dongle is properly attached and try to reset it by removing the power source.

Other general tips include:

- Test the different hardware components if you get unexpected behavior repeatedly. They have a tendency to not work as expected.
- Turn everything off and on again, including the Raspberry Pi used.
- Try the code on a different robot to determine if it is a hardware or software error.

Software used

There are a lot of useful software used. Some have their own quirks that is useful to be aware of.

- Segger
 - The file system is weird. How the files are organized in the file explorer does not necessarily match the one inside the emProject file. If you add a file to the correct directory, it will not automatically appear in SEGGER. You will have to manually add them there as well.
- *OptiTrack* Motive
 - The axes used by Motive is non intuitive, and might lead to confusion when exporting the data. The floor is the x-z-plane, while the y-axis describes the height.

Corrupted NRF52840

If the NRF52840 becomes corrupted, you would need to bootload the device and pass it a new firmware file. If the NRF52840 does not show up in device manager as JLINK, this may very well be the case.

Useful posts about this issue:

<https://devzone.nordicsemi.com/f/nordic-q-a/73999/nrf52840-unseen-under-device-manager-how-to-restore> <https://devzone.nordicsemi.com/f/nordic-q-a/41192/where-to-download-j-link-ob-firmware-j-link-ob-sam3u128-v2-nordicsemi-170213-bin>

Previous Work

Previous work

29th May 2023 at 3:37pm

good-to-know

In this section you can find a summary of some of the previous thesis on the project. Note that this list is not complete.

Master theses

- Berg 2023
- Kolbeinsen 2023
- Ruud-Olsen 2023
 - Getting the robot home without server input
 - Some basic obstacle avoidance
- Tran 2023
 - Hardware development
 - Hardware testing
 - Software development
 - nRF52840
- Andersen 2022
 - MQTT and MQTTSN
 - EKF SLAM (not implemented on the robots)
- Mullins 2020
 - Online SLAM

Project theses

- Tran 2022
 - Printed circuit board design
 - Sensor Data Acquisition Board
 - nRF52840
- Frestad 2022
 - Software structure
- Jølsgård 2020
 - nRF custom peripheral shield
- Korsnes 2018
 - Hardware and software development
 - Low level embedded

Finish the Project

Finishing the project

14th April 2023 at 1:55pm

untagged

If your work is finished and works as intended, merge it into the main branch of the repo so it becomes available to future students. If there are several students working the same repo, this might require some work.

If your work did not result in a finished feature, or the merging was too much work, leave your project as a branch on the repo. Name the branch something that clearly describes the feature it includes. Add a readme in the branch that describes it further, and add your name so future students can refer to your report for further information. Please remove any unnecessary comments and add explanations where needed. Delete all the other branches you have worked on.

Please add a brief summary of the work done to [Previous work](#).

Sensor Data Acquisition Board

Sensor Data Acquisition Board

29th May 2023 at 4:35pm

sensor tower SLAM under development

The sensor data acquisition board, which utilizes two nRF52840 SoCs, is a new system under development. The purpose of the new system is to collect data from a IR-sensor tower (equipped with 4 IR sensors) and an IMU. It also includes a dedicated nRF52840 SoC specifically designed for SLAM algorithms.

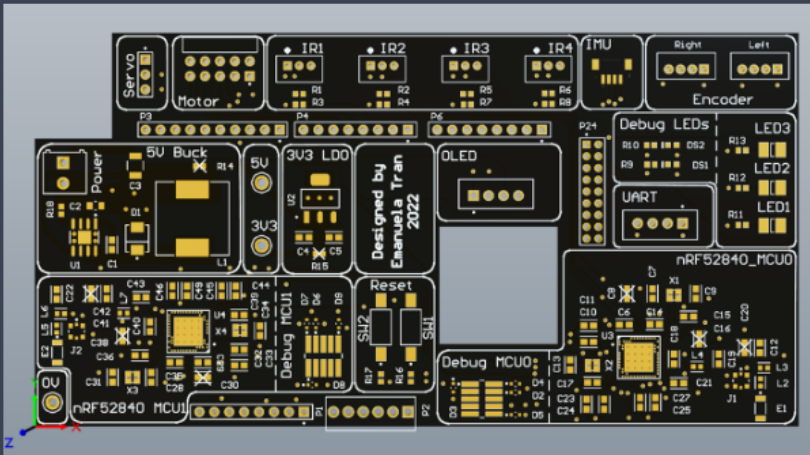
This new system is currently in a prototype phase and the custom PCB is not yet integrated on the robots. Further hardware testing, software development and integration still remains

The figure below provides a visual representation of the system's structure. Signals concerning encoder, motor driver and OLED will pass through the PCB and can be accessed by connectors dedicated to a nRF52840-DK. The rest of the signals will pass through one of the SoCs on the board. The acquired data can either be sent directly to the SoC on the board or to another device

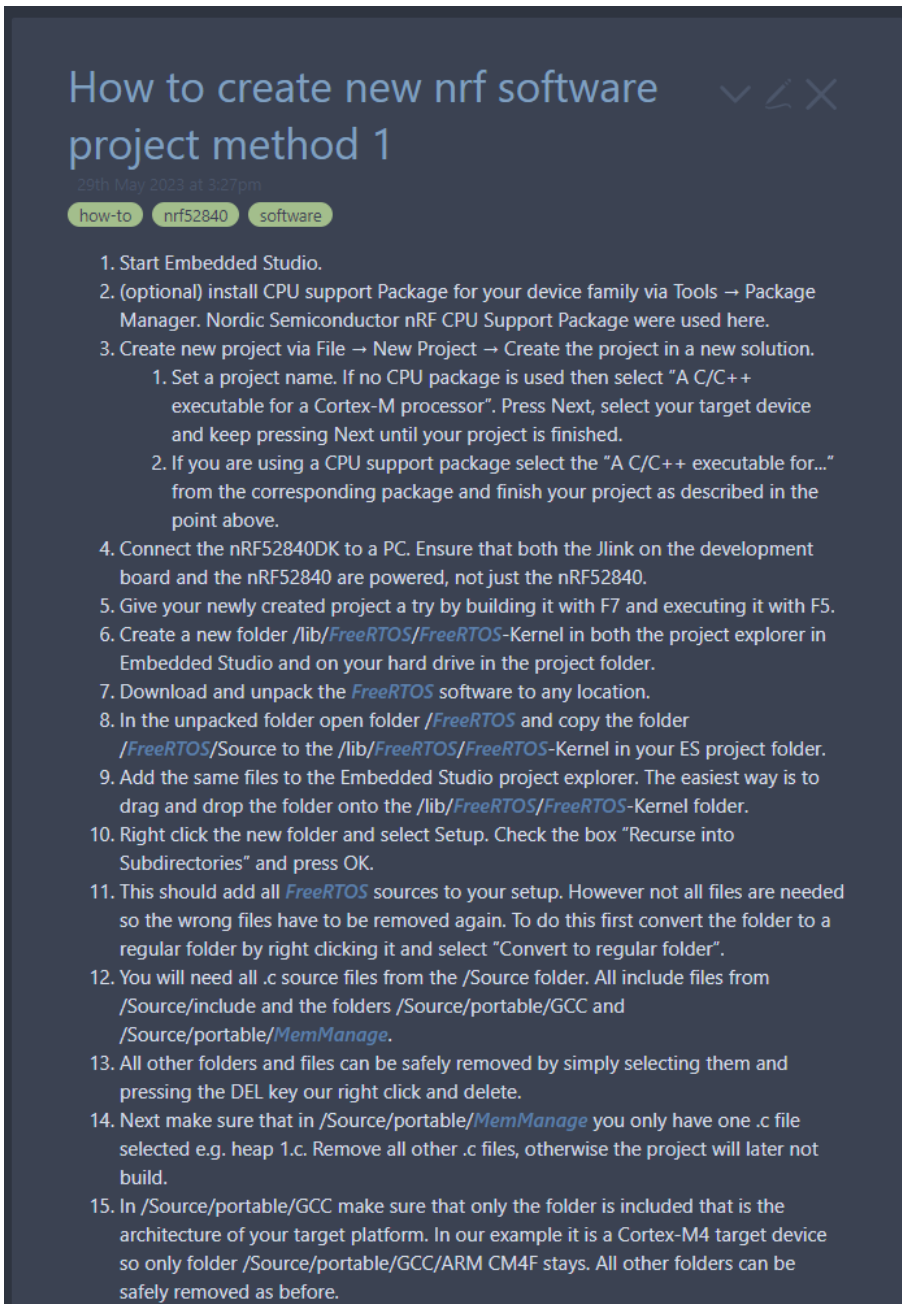
```
graph LR; Connectors[Connectors to nRF52840DK]; ServoTower[Servo Tower Handler]; SLAM[SLAM Handler]; IR[IR - sensor-connector]; Servo[Servo-connector]; IMU[IMU-connector]; UART[UART-connector]; Encoder[Encoder-connector]; Motor[Motordriver-connector]; OLED[OLED-connector]; Connectors <--> ServoTower; Connectors <--> SLAM; ServoTower <--> IR; ServoTower <--> Servo; ServoTower <--> IMU; SLAM <--> UART; SLAM <--> Encoder; SLAM <--> Motor; Connectors <--> Encoder; Connectors <--> Motor; Connectors <--> OLED;
```

Listed below are specifications of the custom PCB:

- Board size: 130mm x 67 mm.
- Four layer PCB : top, ground, power and bottom.
- 1 Oz copper thickness
- All components on the PCB are placed on one side.
- The PCB must be powered with 12 V DC to function.
- Can be put on top of a nRF52840DK and deliver power to the development kit.
- PCB can deliver both 5 V and 3.3 V
- Two nRF52840 *SoCs* with debug connector
- Total of 11 connectors for peripherals such as servo, motor driver, IR-sensors, IMU, encoders, OLED and UART
- Test point for 0 V, 5 V and 3.3 V
- Debug LEDs for 5 V and 3.3 V
- Indication LEDs for the robot
- Board cutout for accessing debug connectors from the nRF52840-DK



How to create new nrf software project method 1



The image shows a screenshot of a documentation page with a dark blue background. At the top, the title "How to create new nrf software project method 1" is displayed in a light blue font. To the right of the title are three small icons: a downward arrow, a leftward arrow, and an 'X'. Below the title, the date "29th May 2023 at 3:27pm" is shown. There are three tags: "how-to", "nrf52840", and "software". The main content is a numbered list of 15 steps for creating a new nRF software project in Embedded Studio.

How to create new nrf software project method 1

29th May 2023 at 3:27pm

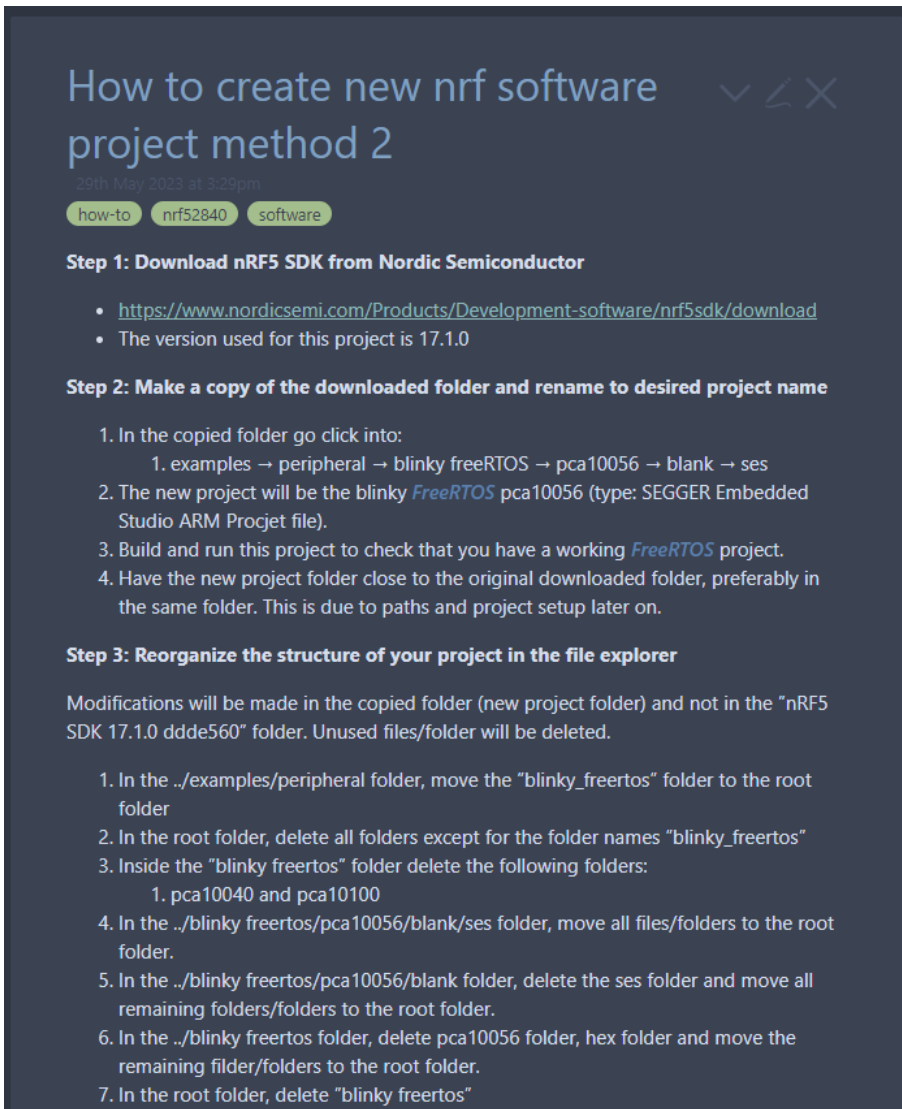
how-to nrf52840 software

1. Start Embedded Studio.
2. (optional) install CPU support Package for your device family via Tools → Package Manager. Nordic Semiconductor nRF CPU Support Package were used here.
3. Create new project via File → New Project → Create the project in a new solution.
 1. Set a project name. If no CPU package is used then select "A C/C++ executable for a Cortex-M processor". Press Next, select your target device and keep pressing Next until your project is finished.
 2. If you are using a CPU support package select the "A C/C++ executable for..." from the corresponding package and finish your project as described in the point above.
4. Connect the nRF52840DK to a PC. Ensure that both the Jlink on the development board and the nRF52840 are powered, not just the nRF52840.
5. Give your newly created project a try by building it with F7 and executing it with F5.
6. Create a new folder `/lib/FreeRTOS/FreeRTOS-Kernel` in both the project explorer in Embedded Studio and on your hard drive in the project folder.
7. Download and unpack the *FreeRTOS* software to any location.
8. In the unpacked folder open folder `/FreeRTOS` and copy the folder `/FreeRTOS/Source` to the `/lib/FreeRTOS/FreeRTOS-Kernel` in your ES project folder.
9. Add the same files to the Embedded Studio project explorer. The easiest way is to drag and drop the folder onto the `/lib/FreeRTOS/FreeRTOS-Kernel` folder.
10. Right click the new folder and select Setup. Check the box "Recurse into Subdirectories" and press OK.
11. This should add all *FreeRTOS* sources to your setup. However not all files are needed so the wrong files have to be removed again. To do this first convert the folder to a regular folder by right clicking it and select "Convert to regular folder".
12. You will need all .c source files from the `/Source` folder. All include files from `/Source/include` and the folders `/Source/portable/GCC` and `/Source/portable/MemManage`.
13. All other folders and files can be safely removed by simply selecting them and pressing the DEL key our right click and delete.
14. Next make sure that in `/Source/portable/MemManage` you only have one .c file selected e.g. heap 1.c. Remove all other .c files, otherwise the project will later not build.
15. In `/Source/portable/GCC` make sure that only the folder is included that is the architecture of your target platform. In our example it is a Cortex-M4 target device so only folder `/Source/portable/GCC/ARM CM4F` stays. All other folders can be safely removed as before.

-
16. Next you will need to create a *FreeRTOSConfig.h* file which will configure your *FreeRTOS* setup. For references see the *FreeRTOS* documentation or use one of the config headers from the many samples out there as reference. For example the one from the example project above. We recommend to place this file into your source folder where your main.c file is. In this case it is folder /source in the project folder.
 17. Next all include paths need to be set. You can add this in project options under Project → Options → Preprocessor → User Include Directories. If you are using the same folder structure as recommended the following three include paths must be set. If you are using another project structure adjust the paths accordingly.
 1. $\$(ProjectDir)/source$
 2. $\$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/include$
 3. $\$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/portable/GCC/ARM_CM4F$
 18. Now edit your main.c to include *FreeRTOS.h* and *task.h* and add your *FreeRTOS* application code to the main.c.
 19. (optional) add your third party libraries, HALs, drivers etc. to your project by including their paths. In this project the nRF SDK folder is outside of this project folder. The paths for the nRF SDK therefore start with "../". Your paths should look something like this:
 1. $\$(ProjectDir)/source$
 2. $\$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/include$
 3. $\$(ProjectDir)/lib/FreeRTOS/FreeRTOS-Kernel/Source/portable/GCC/ARM_CM4F$
 4. ../nRF5 SDK 17.1.0 dde560/modules/nrfx
 5. ../nRF5 SDK 17.1.0 dde560/components/boards
 6. ../nRF5 SDK 17.1.0 dde560/components/libraries/util
 7. ../nRF5 SDK 17.1.0 dde560/modules/nrfx/hal
 8. ../nRF5 SDK 17.1.0 dde560/modules/nrfx/drivers
 9. ../nRF5 SDK 17.1.0 dde560/integration/nrfx/legacy
 10. ../nRF5 SDK 17.1.0 dde560/modules/nrfx/templates
 11. ../nRF5 SDK 17.1.0 dde560/modules/nrfx/templates/nRF52840

Once all this is done your application should build now and you should be able to debug a *FreeRTOS* application in Embedded Studio.

How to create new nrf software project method 2



The screenshot shows a dark-themed web page with the following content:

How to create new nrf software project method 2

29th May 2023 at 3:29pm

how-to nrf52840 software

Step 1: Download nRF5 SDK from Nordic Semiconductor

- <https://www.nordicsemi.com/Products/Development-software/nrf5sdk/download>
- The version used for this project is 17.1.0

Step 2: Make a copy of the downloaded folder and rename to desired project name

1. In the copied folder go click into:
 1. examples → peripheral → blinky freeRTOS → pca10056 → blank → ses
2. The new project will be the blinky *FreeRTOS* pca10056 (type: SEGGER Embedded Studio ARM Projct file).
3. Build and run this project to check that you have a working *FreeRTOS* project.
4. Have the new project folder close to the original downloaded folder, preferably in the same folder. This is due to paths and project setup later on.

Step 3: Reorganize the structure of your project in the file explorer

Modifications will be made in the copied folder (new project folder) and not in the "nRF5 SDK 17.1.0 dde560" folder. Unused files/folder will be deleted.

1. In the ../examples/peripheral folder, move the "blinky_freertos" folder to the root folder
2. In the root folder, delete all folders except for the folder names "blinky_freertos"
3. Inside the "blinky_freertos" folder delete the following folders:
 1. pca10040 and pca10100
4. In the ../blinky_freertos/pca10056/blank/ses folder, move all files/folders to the root folder.
5. In the ../blinky_freertos/pca10056/blank folder, delete the ses folder and move all remaining folders/folders to the root folder.
6. In the ../blinky_freertos folder, delete pca10056 folder, hex folder and move the remaining filder/folders to the root folder.
7. In the root folder, delete "blinky_freertos"

Step 4 (Optional): Reorganize the structure of your project in SEGGER You are free to choose whether you want to restructure the folder structure inside your project (SEGGER)

Right click on your project and add folders. # For this project two folders were created: src and config.

1. The application folder was deleted.
2. Add existing files into the project by right clicking on your folder.
3. The src folder contains main.c.
4. The config folder contains sdk config.h and *FreeRTOSConfig.h*.

Step 5: Include paths and files Open the new project in Segger. It is likely that the build process will encounter errors since all the original files have been removed. Therefore, it is essential to configure the correct paths for the new project so that it can locate the necessary files.

1. Right click on your project and select options.
2. On the top to the left select "Common" for private configurations.
3. Include and the right paths in Preprocessor → User Included Directories.

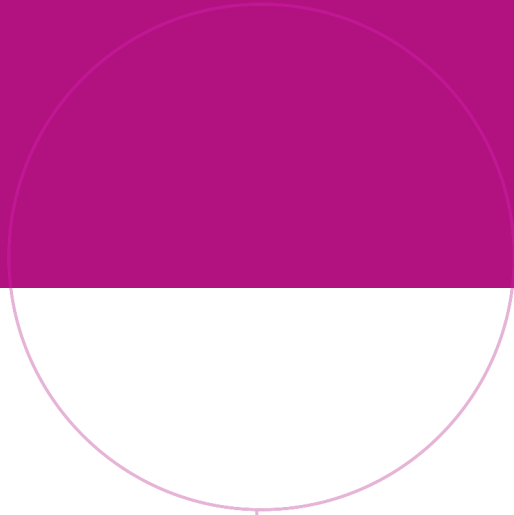
The existing paths are referencing the SDK files that have been deleted. The new project should utilize the SDK files from the original folder that was initially downloaded.

Here is an example of how the paths should appear if the root folder and the original SDK folder are located in the same directory:

```
../nRF5_SDK_17.1.0_ddde560/components
```

After setting the correct paths, the necessary files must be included from the original nRF5 SDK 17.1.0 ddde560 folder. The specific files that are required depends on the driver you wish to use in the project. Navigate to the original nRF5 SDK 17.1.0 ddde560 folder and simply drag and drop the necessary files into the SEGGER project. This guide is specifically based on the *blinky_freertos* example and includes the same files that are used in that project.

Compile and execute the code to verify if it functions similarly to the original *blinky_freertos* example project. If any issues arise, double-check the paths and ensure that the correct files have been included. The *sdk_config* used in this project are similar to the one used in the *blinky_freertos* example.



Norwegian University of
Science and Technology