University of Tartu

Faculty of Science and Technology

Institute of Technology

Ants Kärner

Development for Integrating KuupKulgur's On-Board Flight Computer

Bachelor's thesis (12 ECTS)
Computer Engineering

Supervisors:

MSc Tarvi Tepandi
MSc Ric Dengel

Tartu 2025

Abstract/Resümee

Development for Integrating KuupKulgur's On-Board Flight Computer

Interest in lunar exploration is increasing and there are many missions aiming for the Moon. KuupKulgur, a student project led by the Space Exploration Group at Tartu Observatory, is one such project, developing a standardised lunar rover platform to allow various third-party payloads operate on the Moon on a robotic platform.

In this thesis, an on-board flight computer to integrate into KuupKulgur's internal electronics stack was designed. To accompany this, firmware developed during a previous work was adapted to be future-proof for space-grade hardware, following which various hardware interfaces were implemented more reliably and some from scratch. Furthermore, the implemented interfaces were also tested to verify their functionality.

CERCS: T120 Systems engineering, computer technology; T170 Electronics; T320 Space technology;

Keywords: KuupKulgur, on-board computer, flight computer, CAN

Kuupkulguri pardaarvuti integreerimise arendus

Huvi Kuu avastamise vastu on kasvamas ja käimas on palju missioone, mille eesmärgiks on Kuule jõuda. KuupKulgur, üks Tartu Observatooriumi kosmose avastamise rühma poolt juhitud tudengiprojekt, üritab samuti Kuule jõuda ning selle projekti eesmärgiks on disainida standardiseeritud kuukulguri platform, millega saaks erinevate osapoolte lasti Kuule viia.

Selle bakalaureusetööga disainiti KuupKulguri olemasolevasse elektroonikasse integreeritav pardaarvuti. Sellega koos muudeti eelmises selleteemalises töös loodud püsivara tulevikukindlamaks, mis toimiks ka kosmoseklassi riistvara peal. Lisaks implementeeriti mitmete riistvaraliidestega suhtlus püsivaras töökindlamalt. Püsivaras implementeeritud riistvaraliidestega tehti ka teste, et olla kindlad nende toimivuses.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T170 Elektroonika; T320 Kosmosetehnoloogia;

Märksõnad: KuupKulgur, pardaarvuti, lennuarvuti, CAN

Contents

List of Figures

List of Tables

Acronyms

ASF4 Advanced Software Framework 4. An older software framework for Microchip microcontrollers, not actively developed [1]. 11, 12, 15

CAN Controller Area Network. 6, 9, 10, 12, 13, 15–22, 24–26, 33, 34, 37, 38

CAN FD Controller Area Network Flexible Data-Rate. An extension of the CAN standard with faster data rates. 5, 13, 21

CAN XL Controller Area Network Extended Data-Field Length. 13

COTS Commercial Off-the-Shelf. 12

CRC Cyclic redundancy check. 13, 19, 21

CS Chip Select. Pins used by SPI to select a device to communicate to. 16, 17, 20

FOG Fiber Optic Gyroscope. 14

GPIO General Purpose Input Output. Pins on a microcontroller used as simple digital inputs and outputs. 10, 16, 17

HAL Hardware Abstraction Layer. 11

HRG Hemispherical Resonant Gyroscope. 14

I2C Inter-Integrated Circuit. A two wire interface for connecting multiple chips. 5, 15–21, 24–26

IDE Integrated Development Environment. 11, 18

IMU Inertial Measurement Unit. A sensor usually consisting of an accelerometer and gyroscope to measure velocity and orientation [2]. 10, 13–19, 22, 24–26, 33

LEO Low Earth Orbit. 12

LQFP low-profile quad flat package. 15

MCC MPLAB® Code Configurator. 11, 18

MEMS Micro Electro-Mechanical Systems. 13, 14

PCB Printed circuit board. The electronics board, which contains all the components and connections. 5, 9, 12, 16–18, 22–26

RLG Ring Laser Gyroscope. 14

RTOS Real-Time Operating System. 11

SMD Surface-mount device. A electronics chip with pads on one side of the PCB and not stretching through the PCB. 16

SPI Serial Peripheral Interface. 5, 10, 15–22, 24–26

SWD Serial Wire Debug. 16, 18, 33

TFBGA Thin-profile fine-pitch ball grid array. 15

UART Universal asynchronous receiver-transmitter. 5, 6, 15, 16, 18–20, 22, 25, 34–38

USART Universal synchronous/asynchronous receiver-transmitter. An interface capable of synchronous and asynchronous serial communication. Can be used as an UART or SPI. 16, 17, 33

USB Universal Serial Bus. 18, 19

1 Introduction

Interest in lunar exploration has been increasing lately. The Artemis campaign led by NASA is attempting to establish long-term presence on the Moon and bring humans there again [3, 4] while supporting multiple private companies develop technologies and already send different preparatory lunar research missions [5]. Of those, Firefly Aerospace has successfully finished its Blue Ghost Mission 1 [6], becoming the first commercial company to achieve a successful Moon landing [7]. There are also different European based missions, such as the Argonaut and Moonlight by the European Space Agency, focused on creating the first European lunar lander and a satellite communication network around the Moon. These both are part of the foundation, which makes future missions possible [8, 9]

KuupKulgur is a student project by the Space Exploration Group at Tartu Observatory, focused on developing Estonia's first lunar rover. The goal is to create a standardised platform that allows payloads created by different parties, such as other universities' research projects and private companies, to be sent to the Moon without great costs going to the redevelopment of a rover platform. During the development, it is also used as a demonstrator platform for testing and developing payloads [10, 11]. The goal is to reach the Moon by the turn of the decade [12].

1.1 Problem Statement

Currently, the main computing element of KuupKulgur has to be directly connected to every subsystem on the rover and must manage all the smaller tasks related to it. This complicates its purpose and creates a greater point of failure. To fix this, plans were made to create an onboard computer to oversee smaller tasks and help different subsystems communicate between themselves independently. This would take some of the burden off of the main computing element and allow faults to be less catastrophic and more easily recoverable.

Last year, Artur Eksi successfully defended his thesis[13] "Development of a Prototype Flight Computer for KuupKulgur". His work resulted in a prototype printed circuit board (PCB), which could be used to test the usability of the selected microcontroller and its peripheral interfaces. It was also a starting point for firmware development. However, the board was not designed in a format matching the template for KuupKulgur's internal electronics stack, which was specified later, and was missing the main internal data bus connections. The firmware didn't implement Controller Area Network (CAN) support, which has been selected to be the main internal communication interface on KuupKulgur, and was also written in an older framework, which isn't compatible with a similar radiation-hardened alternative microcontroller.

1.2 Objectives

The main objective of this thesis is to advance the previous work by continuing the development of the firmware and to create a new PCB to integrate into the electronics stack of KuupKulgur. The steps to achieve that are as follows;

1. to create a new revision of the flight computer usable on-board of the KuupKulgur rover based of the microcontroller ATSAMV71Q21B, with the current prototype board[13] as reference;
2. to verify the connections of the CAN and Serial Peripheral Interface (SPI) interfaces as well as the General Purpose Input/Output (GPIO) pins via the main bus;
3. to port the existing firmware to a framework supported by the radiation hardened microcontroller alternative SAMRH71;
4. to add support for driver-level communication with the Inrtial Measurement Unit (IMU) in the firmware;
5. to test the functionality and performance of all of the interfaces implemented in firmware;
6. to integrate the developed flight computer with KuupKulgur's internal electronics stack.

2 Background

2.1 Embedded firmware frameworks

There are multiple ways of writing firmware for microcontrollers. One way is to write purely in a base programming language and modify necessary registers for peripheral interactions. This is time consuming and becomes quickly complicated, so larger embedded firmware projects are often written in a framework, which provides abstraction for peripherals, usually called the Hardware Abstraction Layer (HAL) [14]. The desire to make a single microcontroller responsible for multiple tasks also makes writing firmware more complicated and it is hard to ensure consistent timing. To mitigate that, a Real-Time Operating System (RTOS) is often used. An RTOS provides the ability to define tasks, which the RTOS executes and switches between rapidly according to a scheduling algorithm to make it seem they are running concurrently. It also gives protections and mechanisms to transfer information between tasks [15].

Advanced Software Framework 4 (ASF4) is a framework by Microchip Technology for their microcontroller product lines, mainly for SAM and older AVR families of microcontrollers. Peripherals for the devices can be configured using their web-based Atmel START tool and it supports development in multiple Integrated Development Environments (IDEs), including Atmel Studio 7. As of now, the framework is maintained for older projects, support for new microcontrollers isn't added [1].

A more recent and actively maintained framework by Microchip Technology, which is recommended for new projects, is Harmony v3. It supports all of Microchip Technology's 32-bit MIPS, PIC32A and Arm Cortex-based microcontrollers with the configured drivers and libraries for them generated by the MPLAB® Code Configurator (MCC). These include Peripheral libraries, which is their equivalent for HAL, drivers built on top of creating another layer of abstraction as well as multiple middleware libraries for networking, graphics and other more complex tasks. Some of these libraries require an RTOS and while multiple are supported, the most common and default selection is FreeRTOS [16, 17].

FreeRTOS[18] is a small RTOS, which implements the core functionality necessary for many projects of different levels of difficulty on microcontrollers. It implements real-time scheduling, communication between tasks and different synchronisation and timing functionalities. Additional features can be added with add-on components or developed separately. The project is open source with a permissive license allowing it to be used for free in any kind of project [19, 20].

2.2 KuupKulgur rover

KuupKulgur is the first lunar rover being developed in Estonia. Its initial design is a robotic platform on top of which is a 2U-sized payload [10, 11]. The "U" size comes from The CubeSat program, which standardises dimensions for smaller satellites. One unit (1U) is a cube with

side lengths of 10 cm and the 2U KuupKulgur measures 20x10x10 cm [21]. This cuboid payload can contain different scientific instruments that different projects or companies require and the platform, which is also meant to be used as a payload demonstrator, contains everything necessary for a lunar mission, including subsystems for the rover's mobility and communications for example. [10].

The existing prototype on-board flight computer features the ATSAMV71Q21B microcontroller as the core. It was chosen as a much cheaper alternative during development to the radiation-hardened SAMRH71 as both are from the same product line and have similar core functionality and necessary interfaces [22]. The on-board flight computer could be adapted to the SAMRH71 microcontroller for an actual lunar mission, with the firmware needing few changes. The prototype on-board flight computer also featured a CAN transceiver and convenient headers for all the hardware interfaces, which were predicted to be used in the rover, so they could be tested and basic firmware written [13].

While developing the basic firmware for the prototype PCB, a new communication protocol was created using FreeRTOS. Regretfully, the rest of the firmware directly using microcontroller specific functionality was written using the ASF4 framework. This poses a problem, because the SAMRH71 microcontroller is not supported by it [13]. The manufacturer recommends to move to Harmony v3 for compatibility with newer microcontrollers [1]. The communication protocol implementation, which is not tied to specific hardware, can be easily ported to the new framework.

2.3 On-board computers of space equipment

On-board computers play a central role in the operation of satellites and other space equipment. They manage different subsystems and often facilitate the internal communication between multiple subsystems with their most important component being the microcontroller [23]. For Low Earth Orbit (LEO) satellites, they have become much easier to develop, as the space industry is more accepting of risk and the usage of common Commercial Off-the-Shelf (COTS) components. Several easily accessible microcontrollers have been shown to tolerate the radiation levels in LEO, making the final on-board computers using those much cheaper [24].

Going further away from LEO, to geostationary orbits or beyond, there are multiple hazards to electronics to consider. Van Allen radiation belts are regions around the Earth, where charged particles are trapped by the Earth's magnetic field [25]. The belts are split into two and vary significantly over separate regions of the Earth. There is a region called the South Atlantic anomaly, where the belts are much closer to the Earth [26]. This region is the main radiation hazard for LEO satellites. Still, going through the belts, which is necessary for travel to greater distances than LEO, is much more hazardous as the outer layer has higher energy particles [25]. After passing the belts and escaping Earth's geomagnetic influence, unpredictable solar-particle radiation events and cosmic rays are prevalent [26], which can damage electronics and create malfunctions [24].

The harsher conditions farther from LEO create a need for durable components, so radiation-hardened microcontrollers are used on on-board computers for missions there [24]. That's not to say that COTS components can't handle space beyond LEO. Ingenuity, the Mars helicopter, which showed that flight was possible there [27], was based off of a cell phone processor [28]. Still, it exceeded expectations of a 30 days mission and lasted for just under 3 years [27] and its mission ended because of mechanical failure, not because of the electronics [29]. This shows that COTS components are more than capable, but radiation-hardened components should still be preferred on on-board computers for further reliability [24].

2.4 CAN

CAN is a protocol, that defines the physical interface and frame format of a serial bus [30]. Its development started in 1983 at Bosch and first presented at SAE congress in Detroit in 1986 [31]. Since then, different specifications have been created and currently there are three generations of CAN called classic CAN, Controller Area Network Flexible Data-Rate (CAN FD) and Controller Area Network Extended Data-Field Length (CAN XL). All of their physical parameters and frame format are defined in ISO 11898 standards [32]. The last generation CAN XL is less common due to it being standardised in just 2024 [31].

CAN bus uses two wires terminated by 120 Ohm resistors at each end to connect many devices [30]. It uses recessive state, where both wires are at the same voltage, and dominant state, where they are driven apart from each other, to encode bits [33], with recessive meaning 1 and dominant 0 [30]. To connects the two wires to different hosts, usually separate transceivers are used[32], which allow different features to be included and transceivers with necessary performance selected [34]. Bits sent on the bus are also stuffed, which means if the bus state remains the same for 5 bits, the next bit contains an additional bit of opposite value [30].

A classical CAN frame consists of a start bit, an 11 or 29-bit identifier, 3 control or reserved bits, 4 bits for data length, up to 8 bytes of data, 16-bit Cyclic redundancy check (CRC), 2 bits for acknowledgement and 10 bits in total for ending the frame and spacing. Control bits contain remote transmission request bit to ask for data, identifier extension bit, to use 29-bit extended identifiers, and a reserved bit set to 0 in classical CAN. The data rates are uniform and up to 1 Mbit/s [30].

CAN FD frames are similar, but support separate data rates for arbitration, which contains the identifier and some of the control bits, and data transfer with the rest of the control bits, data and most of the CRC [33]. There are also changes to control bits with the previously reserved bit being changed to 1 to indicate CAN FD, an additional reserved 0 bit for use in the future, a bit to indicate rate change and another for error. The possible data size was increased to 64 bits with decimal values 9-15 used to indicate data sizes from 12 to 64 with predefined meanings. The CRC is extended to 17 bits for up to 16 bytes of data and to 21 bits for more than 16 bytes of data. There are also fixed stuff bits in the CRC and a stuff bit counter before it. Data transfer speeds can be up to 8 Mbit/s [30].

CAN has been used in space for more than two decades. An example is SMART-1 in 2003 [35]. SMART-1 was the first mission to the moon by the European Space Agency. It was a test for communication and miniaturisation [36]. It also successfully used CAN to connect different systems and instruments [37]. It has also been embraced by microcontroller manufacturers with Microchip Technology having multiple radiation tolerant and radiation hardened microcontrollers meant for space exploration feature the CAN and CAN FD interfaces [22].

2.5 IMU

An IMU is a sensor, which combines multiple sensors used for inertial navigation [38]. There are multiple configurations of sensors, which can be included. An accelerometer and a gyroscope are the core of an IMU, providing information about linear acceleration and angular velocity. A magnetometer is often added to give a better idea about orientation [2]. Auxiliary sensors like temperature sensor can be present to better calibrate the device and improve accuracy [38].

For accelerometers, the only two notable technologies are mechanical and Micro Electro-Mechanical Systems (MEMS) accelerometers, there is much more variance for gyroscopes [38].

Most common are Ring Laser Gyroscope (RLG), Fiber Optic Gyroscope (FOG) and MEMS gyroscopes [39] with Hemispherical Resonant Gyroscope (HRG) being used mainly in space [40]. MEMS technology is more recent [2] and has taken over after with the majority of devices on the market being MEMS-based [38]. The reasons are its low cost and power needs, but most importantly its size, with its small size allowing IMUs to be used more widely [2].

An IMU is useful as a complementary input to navigation for rovers and can be used to detect abnormal situations [41]. They have been used in satellites and rovers for decades. MEMS has still posed a challenge with it being less precise [40], but they have been combined and used successfully even in situations with higher precision requirements [42].

3 Methodology

3.1 Requirements

Development was in two parts, a new hardware revision and extending the functionality of the existing firmware after porting it. Before the start of development, some requirements were set.

3.1.1 New hardware revision

The main requirements for the new hardware revision were set as following:

- the board must be compatible with KuupKulgur's main bus and chassis;
- the board must connect to all of the CAN and SPI interfaces on the bus;
- the board must bring out Universal asynchronous receiver-transmitter (UART) and Inter-Integrated Circuit (I2C) interfaces outside the main bus for further testing;
- the board must contain an IMU;
- the board must use the ATSAMV71Q21 as its microcontroller.

3.1.2 Firmware

The firmware developed must match the following requirements:

- the firmware must be ported to MPLAB Harmony v3 from ASF4;
- as much code should be reused as possible;
- UART and other interfaces' performance must be tested and should be improved if possible;
- CAN interface communication must be implemented.

3.2 Component selection

The most important component of the flight computer, chosen during the previous work, is the microcontroller ATSAMV71Q21 [13]. It is a 32-bit microcontroller based on the ARM Cortex-M7, which includes all of the necessary interfaces [43]. It was selected as the closest alternative to the radiation-hardened SAMRH71 [44]. The chip comes in two packages: 144 pin low-profile quad flat package (LQFP) and 144 pin thin-profile fine-pitch ball grid array (TFBGA)

[43]. The former was used on the PCB, because it was already present at Tartu Observatory, room wasn't an issue with the board and it is much easier to solder it correctly.

The CAN transceiver was replaced and the new chip selected was the TCAN3404-Q1 [45]. Previously, TJA1462AT was used [13], but that was decided against, because it would be the only device, that uses 5V on the board. Instead, TCAN3404-Q1 also works on 3.3V, which most of the other chips use. It was decided to try the new chip out and verify, if it works. It would simplify designing future boards if 5V needs to be routed less. Both of the chips also share a pin compatible mechanically identical footprint, which is named SO8 for TJA1462AT and SOIC-8 for TCAN3404-Q1. Only the power supply and additional functionality pin are different, which can be made to work using two solderable jumpers, so both chips could be used on the board [45, 46].

The IMU selected is IAM-20680HP. It was selected due to being actively manufactured, it survives to 10000g acceleration and it is automotive grade. It also needs little supplementary components to work and is meant to be used in driving conditions, which a rover will encounter [47].

The supplementary passives used were surface-mount device (SMD) components in imperial 0603 size. That size is easy to solder, there was much room on the board and Tartu Observatory had all of them present on site.

The main bus headers decided before were TW-20-09-F-D-305-SM-A and CLT-120-02-F-D-A-TR. All of the other headers were standard 2.54 mm connectors present on site at the Tartu Observatory.

The full bill of materials is visible in appendix A.

3.3 PCB development

The new hardware revision's schematic and layout were developed using KiCad. KiCad is a free cross platform and open source software, which allows further development without being tied to student licensing agreements or increasing costs [48]. It is also widely used at Tartu Observatory and all of the KuupKulgur project's electronics are designed in it. All of the designs were uploaded to an internal GitLab repository.

The PCB was designed on top of the main stack template agreed on before the start of the design. The template visible in figure 3.1 is a 96x96 mm PCB with rounded corners and 4 mounting holes in the corners. The main bus consists of two SMD headers on each side of the board allowing many PCBs to be stacked on top of each other. The height is determined by the main bus headers and all the components were put on top of the board to give the PCB below it as much room as possible.

The data side of the main bus includes two CAN interfaces, two SPI interfaces with two Chip Select (CS) pins each, 4 GPIO pins controllable from the flight computer microcontroller and many reserved pins. The main bus also carries 3.3V, 5V and 12V power. The main voltage considered was 3.3V, because it is used by most of the chips, and 5V was also included in the design as an alternative for the CAN transceivers.

On the board, besides the main bus connectors and main chips, there were also additional pin headers included. Some are necessary in some form on the final design as well, like the Serial Wire Debug (SWD) header, which is needed to program the microcontroller. There are also other headers included to ease testing. Ground connections were brought out, as well as Universal synchronous/asynchronous receiver-transmitter (USART), mostly to use a subset of it's features as an UART interface, and I2C interfaces, which aren't on the main bus, but might still be wanted to be tested or to just ease communications with the microcontroller during

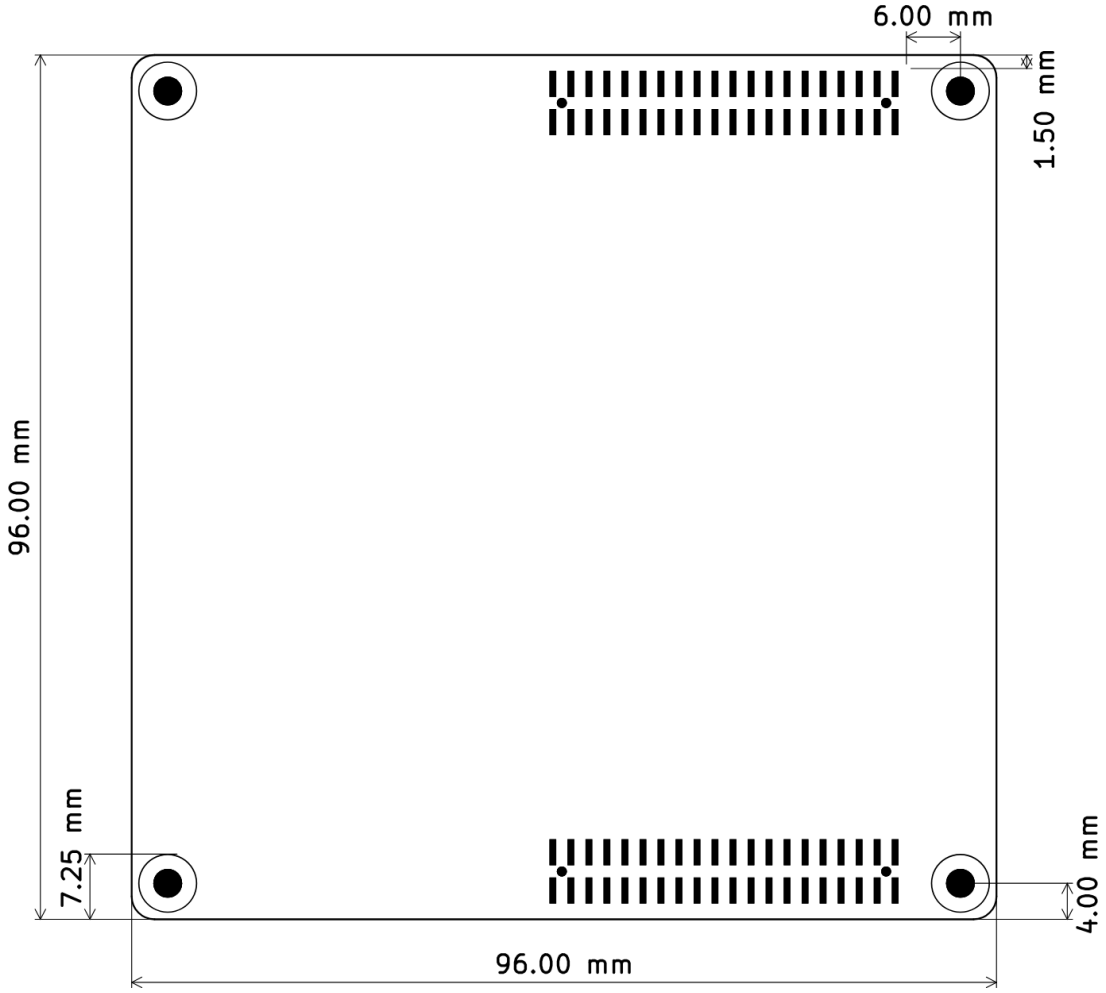

Figure 3.1: PCB template measurements with main bus connectors

testing.

On the schematic, the power and programming side were left the same as in the previous prototype. The only change was discarding the screw terminal previously used for power and connecting power traces to the main bus instead. Additional headers for USART2 and I2C0 were left the same as well, because it would make migrating code to the new board easier and they weren't in the way of other components. All of the pins not used for power or any data interface were left unconnected.

For the interfaces and data pins connected to the main bus instead of separate headers, there were some changes. SPI0 pins were left the same but two of the CS pins were discarded, as on the main bus, there are only two per SPI interface, instead of four on the previous prototype. SPI1 interface was added with previously undedicated pins. MCAN1 used the same data pins, but standby pin was moved to a more convenient location and shutdown pin was added as new functionality of the new chip. MCAN0 was added with previously undedicated pins as well as USART1 to be used for SPI communications with the IMU. The 20-pin GPIO header connected to a variety of pins of the microcontroller was discarded, as it was unused during the author's initial testing and firmware development, and 4 previously unused pins were used as GPIO pins for the main bus.

On the PCB, the microcontroller was positioned to be in the centre. It was the most logical position as there wasn't a lack of room, it allowed for other components to be placed around it

and easily routed. The IMU was placed to the side of the microcontroller, on the central axis between the main bus headers. This is also the central axis of the whole rover and allows data from the IMU be used with less transformations and give a better idea of the movement of the rover.

To simplify testing in the future, multiple notes were left on the PCB silkscreen. All of the headers other than the main bus and erase headers were named according to the interfaces they represented and all the pins were named according to their functionality on said interfaces. The erase header with the ERASE and 3.3V pins is meant to be shorted to erase the microcontroller flash [43] and didn't require more specific notes. The solderable jumpers for CAN chips' supply voltages and additional functionality pin were also labelled and described, which jumper selection means what.

The PCB was created with four layers with the first, second and fourth layers' empty space filled with a ground plane and the third layer's with a 3.3V power plane. The third layer was not used for any traces to allow better power delivery. All of the data traces were routed on the other layers while trying to ensure as short ground connections as possible by avoiding routing traces on the same path on all layers and placing a considerable amount of vias next to traces. 5V power was routed on the top layer to allow best heat dissipation. All of the high speed data traces of the SPI, UART, CAN and I2C interfaces were length matched for better performance and less errors during high speed communication.

3.4 Firmware development

3.4.1 Environment

All of the firmware development was done in MPLAB X. MPLAB X is a multi platform IDE meant for development on Microchip Technology's microcontrollers. It integrates the MCC used for including MPLAB Harmony v3 drivers and configuring peripheral drivers. The IDE has many powerful debugging functionalities, which are easy to use, and has seamless code uploading with many different programmers [49]. It also integrates the XC32 compiler, which is meant for Microchip Technology's 32 bit microcontrollers [16].

An alternative development environment based on the Visual Studio Code IDE [50] was also tested at the start of development to not be completely locked in Microchip Technology's products, but that turned out futile. MCC can be used as a standalone software, but it doesn't generate the build files [51]. Microchip Technology is working on Visual Studio Code extensions, one of which is connected to the MCC, which would hopefully generate build files and allow the external IDE to be used, but during the start of development, it wasn't finished and accessible [52]. MPLAB X could be used to generate build files for all the existing files and then development could be done in an another IDE, but that would become complicated and MPLAB X worked well enough.

To program the microcontroller, ATMEL-ICE was used. ATMEL-ICE is a hardware programmer and debugger by Microchip Technology, meant for AVR and ARM Cortex-M microcontroller development and supporting multiple programming interfaces. It integrates well with MPLAB X IDE and supports the SWD programming and debugging interface [53]. To connect the programmer to the flight computer board, a custom cable to connect the SWD header on the board to the 10 pin SAM port on ATMEL-ICE was used. The programmer was mainly used, because it was already available at Tartu Observatory and it supports all the necessary functionality.

For manual communication with the microcontroller, a simple Universal Serial Bus (USB)

to UART converter was used with HTerm software. It is a simple but powerful cross platform tool for connecting to different serial interfaces [54]. To communicate manually over CAN, a Canable 2.0 clone was used with the Cangaroo software [55].

3.4.2 Testing tool

To simplify the testing of interfaces, a Python testing tool was created during previous work. This testing tool included performance checks for simple back and forth UART packet sending and simple tasks to verify, that SPI and I2C are working [13].

During the development of the this thesis, the tool was extended. Reliability and missing packet detection was improved and more convenient checking of packet receiving, without having to wait for minutes just in case everything wasn't received, was implemented. In depth CAN testing was added.

Previously, only more thorough UART performance test was implemented. This consisted of the testing tool generating and sending packets through the UART interface to the on-board computer. The on-board computer routed the packets and sent them back to the testing tool through the same UART interface. The loopback sending time was measured to get the speed of the transfer and reliability was checked by checking if the received packets matched the sent ones.

The CAN performance test extended this. After reaching the on-board computer, the packets are instead routed out of one of the CAN transceivers. The destination can be another on-board computer, which routes the packets and sends them back to the CAN transceiver it came from, or another CAN transceiver on the same device. The latter was chosen for testing during this work to more clearly describe a packet stream and the on-board computer's capabilities, not the limitations of sending packets back and forth over a single link. Then the packets routed and sent back to the testing tool over the UART interface. CAN communication through a Canable 2.0 was also attempted to be implemented in the testing tool, but the device was unresponsive during the tries and focus was turned elsewhere.

SPI and I2C tests were less modified, only to account for the specific devices used.

The USB to UART converter used during testing was a cheap BTE22-11 module bought from Aliexpress, which was based on the CH9340s chip [56]. The device used to test I2C communications was a module based on the MPU6050 IMU[57] and SPI was tested with the KuupKulgur's V2 motor control board containing the ATmega324PB[58] microcontroller.

3.4.3 Communication protocol

During the development of the previous prototype, detailed in [13], a communication protocol was defined to allow easier and more uniform routing of data between different interfaces. Several existing protocols were considered, but they had different drawbacks, so a new one was created. The protocol packet as seen in figure 3.2 consists of a 4 byte header, a 1-250 byte data portion and a 2 byte tail, with the maximum length being 256 bytes and fitting in an 8-bit unsigned integer. The header consists of type and ID fields, which are used to recognize and order packets and their goals, the destination and origin addresses and the data portion length in bytes. Addresses can be interfaces, where the packet came from or is headed to, but can also be separate tasks. The data portion consists of user data, it is always at least 1 byte long and no longer than 250 bytes. The tail consists of 16-bit CRC value to check for package integrity.

This packet form is good for simple UART communication, which is what was tested in the previous thesis [13], but in the defined form, it won't work with SPI, I2C and CAN. SPI and I2C are also separate use-cases, because the bus' communication is initiated by the flight computer

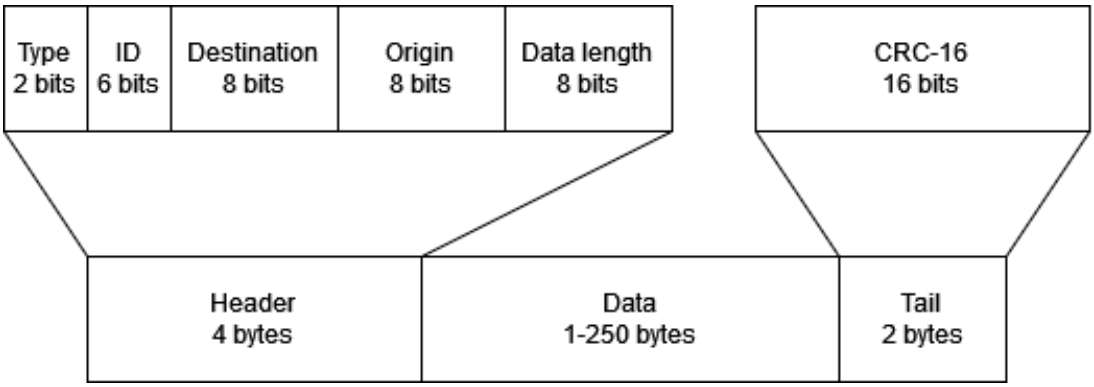

Figure 3.2: Communication protocol packet as defined during the development of the previous prototype and sent directly over UART[13]

and devices on either of the buses can't randomly send data to the flight computer. CAN is more similar to UART in that sense with communication being able to be started outside of the flight computer, but it has a clearly defined packet structure and limited packet size, which must be made to work with our communication protocol.

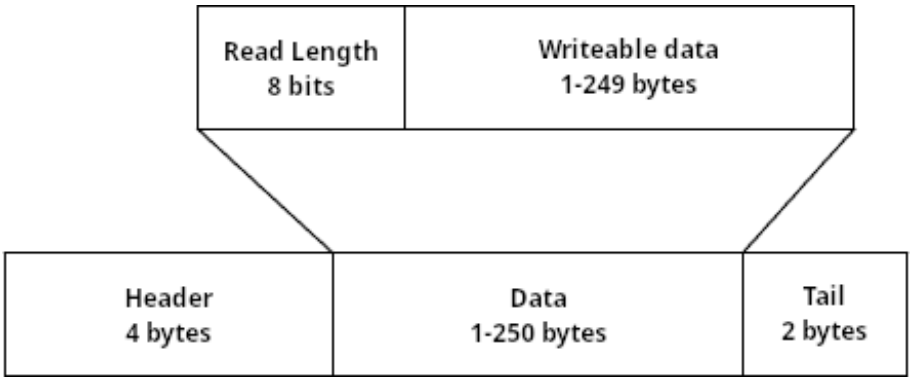

Figure 3.3: Communication protocol packet content for SPI communication

SPI can be easily made compatible, because it has a limited number of possible destinations, one for every CS pin on the on-board computer, which can each be assigned addresses. Although SPI communication is full-duplex, meaning data can be received and transmit at the same time, only consecutive writing and reading was implemented as the most common use case. On figure 3.3, the proposed command format is described, with the first data byte being reserved for read length and the remaining up to 249 bytes for writing. Writeable data is first fully sent out and then read length number of bytes are read in. The writeable length can be figured out from the data length field in the header.

I2C is a bit tougher to make work, because it can have random 7-bit and in newer rarer forms 10-bit addresses. It was decided to limit I2C to a single routable address for the interface on the flight computer and keep the rest of the details in the data portion of the packet. On figure 3.4, it can be seen that the first byte of the data field of a packet to the I2C communication interface contains the 7-bit I2C address and the bit identifying reading with 1 and writing with 0. When writing, then the rest of the 249 bytes can be filled with as many bytes to be written as necessary. When reading, then only one byte is necessary after the address, saying how many bytes to read.

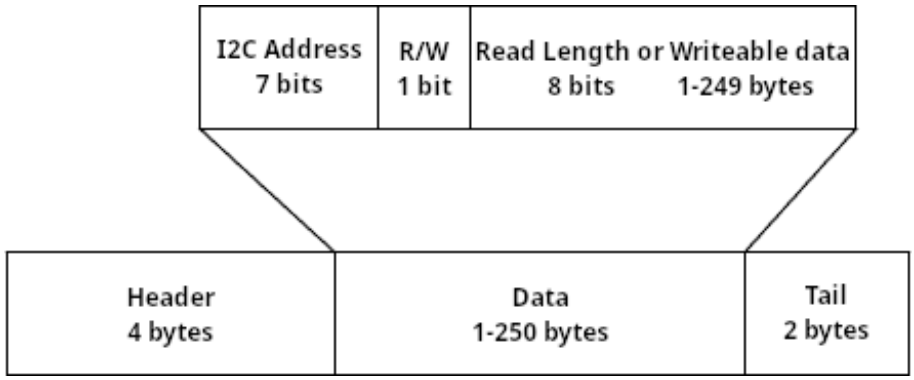

Figure 3.4: Communication protocol packet content for I2C communication

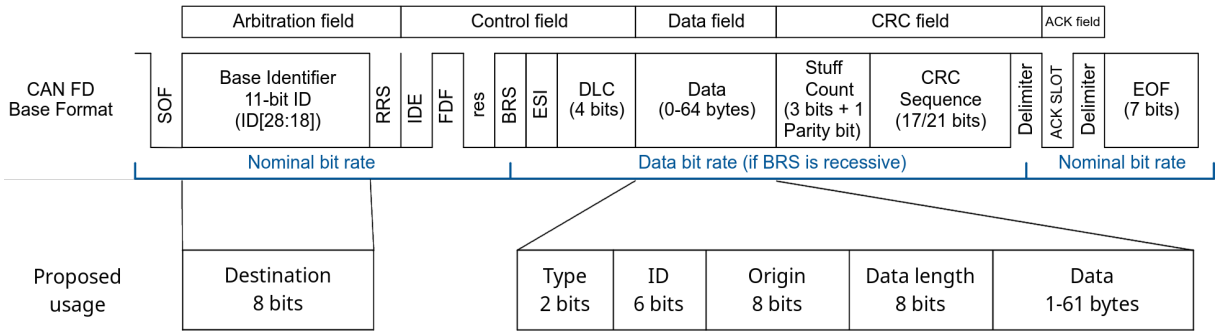

Figure 3.5: CAN FD frame[59] adapted to the communication protocol

CAN is capable of being randomly received, so its implementation must be more universal, not just wrapping requests in the communication protocol, as was done for SPI and I2C. The proposed format, visible in figure 3.5, contains the destination address in the CAN ID field, so the receiving device can identify itself. The other 3 bytes from the header were pushed to the CAN FD data field, leaving 61 bytes for user data. The tail was discarded for packets, which fit into the 61 bytes, because CAN has CRC checks built in.

As the communication protocol allows for packets of up to 250 bytes, they must be split into multiple CAN frames when the length is over 61 bytes and reassembled at the receiving side or task. The split frames can be detected when the data length field is greater than the actual length of data and the reassembling can be done using the ID field, which is incremented every split frame, in addition to the origin and destination addresses.

For those multi frame packets, it was decided to leave the CRC-16 at the end to check for integrity after reassembling. 250 bytes is just over four 61 byte frames, which means leaving it out wouldn't decrease the frame count and save that much performance, when considering the confidence it gives in finding faults. 4 frames of 61 bytes of user data is up to 244 bytes and 242 bytes when considering the CRC. Shortening the defined protocol length was considered, but it was decided against, and 242 bytes can be considered a recommended limit for packets transmitted over CAN, although the full 250 bytes is supported as well, with the fifth frame having the last 8 bytes.

3.4.4 Firmware architecture and additions

The architecture was conceived during the previous prototype's development and the overarching idea didn't change much. The main tasks are a routing task, an interface sending task and receiving tasks for all capable interfaces, of which previously, only UART was present. All of the communication between the tasks was handled through custom implemented ringbuffers, with every separate part of the firmware having input and output buffers [13].

During the development of this work, three new tasks were implemented, with two being duplicates. They were CAN receiving task and tasks for managing IMU initialisation and communications. As there were two IMUs tested, one on the PCB itself and one on another board connected over the main bus, the two IMU tasks were meant to be included when necessary.

The IMU tasks are intermediary optional tasks. They can automatically initialize the IMUs and can continuously ask current state from them. They can be queried for data as needed by any addressable destination. They are optional, because the IMUs are routable and any device can send requests over CAN or UART, but the tasks make it easier and the other side doesn't have to implement complicated SPI commands and parse the data to a readable format.

On the routing and buffer side, direct buffer to buffer copy was implemented without the need for intermediary arrays and indexing errors were fixed. This improved performance and UART communication reached 100% reliability during the same tests where problems were identified during the previous work. Performance and reliability was also impacted by adding multiple fail-safes to the UART receiving task.

4 Analysis and Discussion

4.1 Hardware

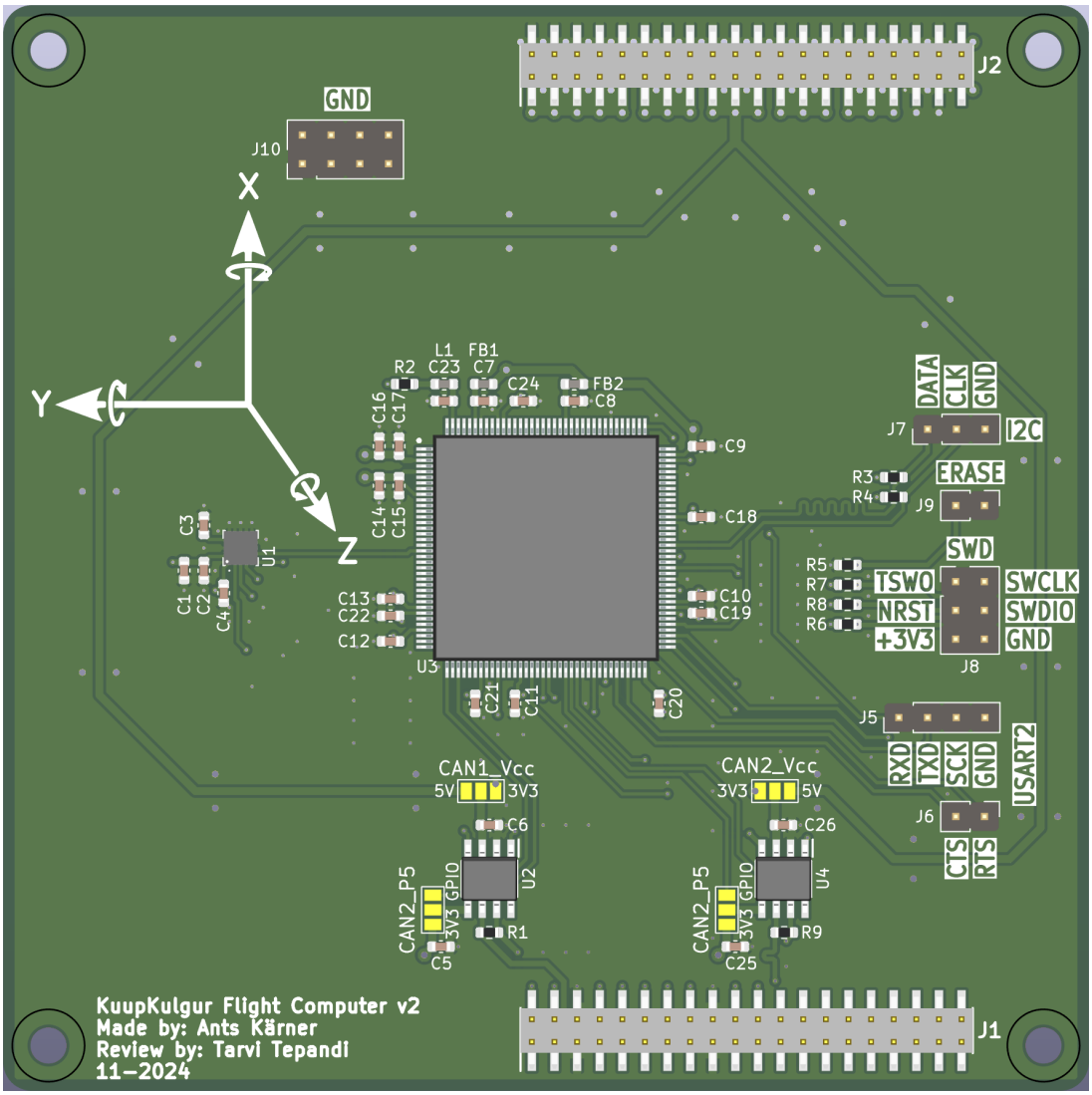

Figure 4.1: Ortographic 3D view of the finished PCB from Kicad

The new hardware revision visible in figure 4.1, designed and produced during this work, fulfils the requirements set in section 3.1.1. The PCB successfully fits in the electronics stack, as can be seen from figure 4.2, and includes all the other headers necessary for developing and testing. In the future, after sufficient firmware development and integration with the other

already present software, it can be included in the electronics stack and put in the rover's chassis to fulfil its purpose.

The developed PCB features multiple silkscreen notes to make firmware development and testing easier. The most notable of them is the image depicting IMU axis directions visible in figure 4.1.

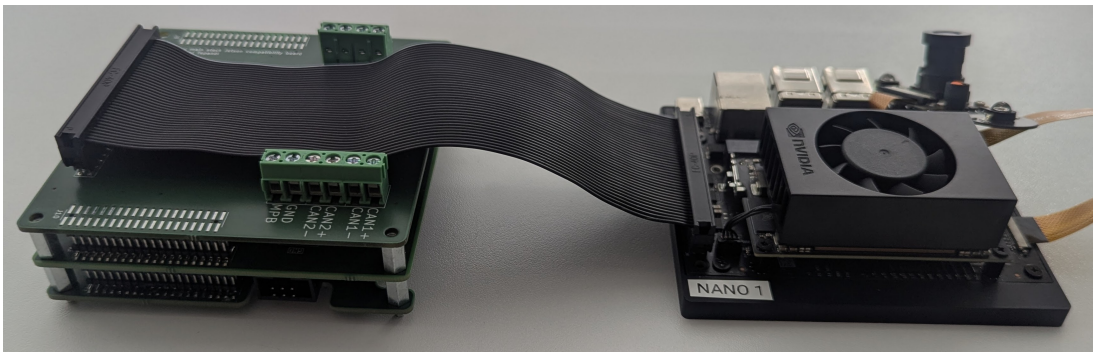

Figure 4.2: KuupKulgur's electronics stack with the flight computer in the middle of the left PCB stack

While testing the firmware after producing the PCB, it was found, that some other pins were slightly overlooked while setting the requirements and during the design and only consciously added to the main bus headers, which are not accessible using common jumper cables and made it harder to test. These include 3.3V power pins and the CAN pins. Fortunately, the 3.3V pin was present in the erase header, and the CAN pins were accessible via a separate cable[60] with the connector matching the main bus. This made it a slight annoyance and not an inherent problem with the design.

During the firmware development, it was also verified, that the newly chosen TCAN3404-Q1 CAN transceivers worked well and could be used with other transceivers, which are not powered by 3.3V.

4.2 Firmware

The written firmware successfully achieves the set out firmware requirements in section 3.1.2. It was successfully ported to MPLAB Harmony v3 and reused as much code as possible, fulfilling the first two requirements. The new development environment created will work with the radiation-hardened alternative SAMRH71 and can be used with the final hardware, which will be sent to the Moon.

CAN support, which was the fourth firmware requirement, was added and partly integrated into the existing communication protocol. The current implementation has some flaws. The received frames are not merged after a packet with more than 61 bytes of data was split, leaving the task to the destination. This is left as future work, if it is determined as necessary. Another observation to note is, that there are some problems with data cashing using the XC32 compiler and CAN started working only, when data cashing is disabled, even if the memory buffer is flagged to not be cacheable. This problem has been described in other systems as well and is not specific to this project [61, 62]. It should be investigated further, if some other configuration would allow general data caching to be enabled.

Support for I2C and SPI packets was specified and implemented. The implementation left room for improvements, when new information about the needs for the interface is determined.

For SPI, basic communications were implemented without support for receiving and transmitting at the same time. This was left for future work, as the proposed protocol command format can support it.

While adding I2C support, it was decided to limit the routable addresses to one for the existing interface on the flight computer even though there can be multiple devices connected to the it. These devices can be selected between with information in the data field. Multiple routable addresses was out of scope for this work, but the communication protocol can support this, when implemented in the future.

The testing of the hardware interfaces, which was the third firmware requirement, was conducted more deeply for UART and CAN interfaces and the test results given by the testing tool are presented in appendix B. The tables B.1 to B.4 there are directly comparable to the results reached in the previous work[13]. These show marginal improvements in speed in longer constant packet length tests and much greater improvements in shorter random packet length tests. Irrecoverable failure was also mitigated and for all of these tests, 100% transmission reliability was reached without impacting speed, fixing the few missing packets from the previous work.

Results from table B.7 also show, that transmissions could be sped up by increasing the baud rate of UART and disabling other unnecessary tasks, with only 1 in 10 tests showing decreased reliability. This unfortunately isn't the case for longer tests with a lower baud rate when including the CAN frame receiving tasks as well. Table B.5 shows reliability for up to 1000 packets transmitted consequentially, but table B.6 shows, that it isn't the case for consequential transfers of 10000 packets, where reliability drops. This is likely the case because packet routing can't keep up and the interface buffers are filled, so packets must be discarded, as error messages indicate. Routing itself isn't the problem, because the tests in table B.7 show, that it can keep up. The cause is a scheduling issue, with more tasks unnecessarily taking up more processing time and could be fixed by transitioning to a more interrupt and message driven task scheduling and not relying on making tasks sleep, when there isn't anything to do, to give other tasks time to run.

CAN tests tell a similar story. CAN by itself doesn't slow transfers down by much, with the purely UART transfers in table B.4 having an average speed of 4205 useful bytes per second and the comparable CAN transfers in table B.8 having an average speed of 4096 useful bytes per second going through the various interfaces and back to the testing tool. The longer 10000 packet transfers described in table B.6 for UART and B.9 for CAN show the CAN transfers missing 203 more packets on average, which confirms the routing being overwhelmed due to having to route a single packet twice for CAN transfers, instead of once for UART.

I2C and SPI interfaces were also shown to work with example devices, but further testing is necessary, when there is a clearer testable usage for the interfaces.

One of the initial goals was driver-level communication with the IMUs, which was implemented following the recommendations in the datasheet[47], but unable to be tested. Communication with the IMU on the flight computer PCB couldn't be established, which needs further investigation to verify, if the problem was in firmware or the chip was damaged during soldering. The external IMU did respond to requests, but they turned out to be incorrect and no valid responses could be received. This needs a similar investigation.

5 Conclusion and Future Work

In this thesis, an on-board flight computer was developed to integrate into KuupKulgur's electronics stack. A new hardware revision was designed based on the previous work[13], the firmware was ported to a new, more future proof framework and its performance and reliability was improved while additional interfaces and tasks were implemented.

The new PCB and its accompanying firmware achieves the requirements set at the start of development. After some further firmware development and integration with the existing software, the PCB is ready to be included in KuupKulgur's electronics stack. The created development environment is suitable for development continued development and is transferable to the radiation-hardened microcontroller alternative further down the line of KuupKulgur's mission. The existing firmware is verified to work as a packet router and is extensible with other tasks. All of the required hardware interfaces are implemented in the firmware.

During this thesis, multiple ways to improve further were found as follows:

- optimization of the firmware through interrupt and message driven task scheduling to improve routing performance;
- analysis of the issues with communications involving the IMUs;
- further investigation into the problems with CAN communication while enabling data caching with the XC32 compiler;
- implementation of CAN frame recombination;
- more I2C and SPI performance testing with actual use cases and workloads;
- complete integration into KuupKulgur's internal electronics stack.

Acknowledgements

I would like to thank my thesis supervisors, Tarvi Tepandi and Ric Dengel, for their patience and advice during this work.

I would also like to thank my partner and friends for their support, which was necessary to write this thesis.

/ signed digitally /

Bibliography

[1] Microchip. *Advanced Software Framework (ASF) for SAM Devices*. URL: https://www.microchip.com/en-us/tools-resources/develop/libraries/advanced-software-framework (visited on 12/04/2025).

[2] Norhafizan Ahmad, Raja Ariffin Raja Ghazilla and Nazirah M. Khairi. "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications". In: *International Journal of Signal Processing Systems* (1st Jan. 2013). URL: https://www.academia.edu/84030886/Reviews_on_Various_Inertial_Measurement_Unit_IMU_Sensor_Applications (visited on 13/04/2025).

[3] NASA. *Artemis feature*. Running Time: 274 Section: Artemis. URL: https://www.nasa.gov/feature/artemis/ (visited on 13/05/2025).

[4] NASA. *Artemis*. Section: Artemis. URL: https://www.nasa.gov/humans-in-space/artemis/ (visited on 13/05/2025).

[5] NASA. *Commercial Lunar Payload Services*. Running Time: 115 Section: Commercial Lunar Payload Services (CLPS). URL: https://www.nasa.gov/commercial-lunar-payload-services/ (visited on 13/05/2025).

[6] Firefly Aerospace. *Blue Ghost Mission 1*. URL: https://fireflyspace.com/missions/blue-ghost-mission-1/ (visited on 13/05/2025).

[7] Risa Schnautz. *Blue Ghost Mission 1: Live Updates*. Firefly Aerospace. 18th Mar. 2025. URL: https://fireflyspace.com/news/blue-ghost-mission-1-live-updates/ (visited on 13/05/2025).

[8] European Space Agency. *Argonaut: a first European lunar lander*. 30th Jan. 2025. URL: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/Argonaut_a_first_European_lunar_lander (visited on 20/05/2025).

[9] European Space Agency. *ESA Moonlight*. URL: https://connectivity.esa.int/esa-moonlight (visited on 20/05/2025).

[10] *KuupKulgur – Tartu Observatory Space Exploration Group*. URL: https://tospexgroup.space/projects/kuupkulgur/ (visited on 12/04/2025).

[11] KuupKulgur. *KuupKulgur*. URL: https://kuupkulgur.space/ (visited on 08/05/2025).

[12] Marja-Liisa Plats. *Teadlased ja tudengid hakkavad arendama esimest Eesti kuukulgurit*. Tartu Ülikool. 14th Mar. 2023. URL: https://ut.ee/et/sisu/teadlased-ja-tudengid-hakkavad-arendama-esimest-eesti-kuukulgurit (visited on 12/04/2025).

[13] Artur Eksi. “Development of a Prototype Flight Computer for KuupKulgur”. BSc thesis. Tartu Ülikool, 2024. URL: https://hdl.handle.net/10062/107676 (visited on 12/04/2025).

[14] STMicroelectronics. *UM1725 - Description of STM32F4 HAL and low-layer drivers - Rev 8March 2023*. Mar. 2023. URL: https://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf (visited on 10/05/2025).

[15] FreeRTOS™. *RTOS Fundamentals*. May 2025. URL: https://freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals (visited on 10/05/2025).

[16] Microchip. *MPLAB® Harmony v3*. URL: https://www.microchip.com/en-us/tools-resources/configure/mplab-harmony (visited on 10/05/2025).

[17] Microchip. *Introduction to MPLAB X IDE and Harmony v3 for Atmel Studio and ASF Users*. 2020. URL: https://ww1.microchip.com/downloads/en/Appnotes/Introduction_to_MPLAB_X_IDE_and_Harmonyv3_for_%20Atmel_Studio_and_ASF_%20Users_DS00003346A.pdf (visited on 10/05/2025).

[18] FreeRTOS™. *FreeRTOS™*. URL: https://freertos.org (visited on 10/05/2025).

[19] FreeRTOS™. *What is FreeRTOS?* URL: https://freertos.org/Why-FreeRTOS/What-is-FreeRTOS (visited on 10/05/2025).

[20] FreeRTOS™. *Why use FreeRTOS?* URL: https://freertos.org/Why-FreeRTOS/Why-FreeRTOS (visited on 10/05/2025).

[21] Alicia Johnstone. *CubeSat Design Specification (1U – 12U)*. San Luis Obispo, CA, Feb. 2022. URL: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/62193b7fc9e72e0053f00910/1645820809779/CDS+REV14_1+2022-02-09.pdf (visited on 12/04/2025).

[22] Eric Tinlot. “Microchip ARM CAN Solutions for Space”. CAN in Space workshop 2019, 13th June 2019. URL: https://indico.esa.int/event/276/contributions/4554/ (visited on 11/05/2025).

[23] Anusha N et al. “Studies on the Functionality of On-Board Computer in 1U CubeSat”. In: *2023 International Conference on Circuit Power and Computing Technologies (ICCPCT)*. 2023 International Conference on Circuit Power and Computing Technologies (ICCPCT). Aug. 2023, pp. 571–576. DOI: 10.1109/ICCPCT58313.2023.10245606. URL: https://ieeexplore.ieee.org/document/10245606 (visited on 12/04/2025).

[24] Angela Cratere et al. “On-Board Computer for CubeSats: State-of-the-Art and Future Trends”. In: *IEEE Access* 12 (2024), pp. 99537–99569. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3428388. URL: https://ieeexplore.ieee.org/document/10597570 (visited on 12/04/2025).

[25] Tsoline Mikaelian. *Spacecraft Charging and Hazards to Electronics in Space*. May 2001. DOI: 10.48550/arXiv.0906.3884. arXiv: 0906.3884[physics]. URL: http://arxiv.org/abs/0906.3884 (visited on 14/05/2025).

[26] Richard S. Johnston et al. *Biomedical Results of Apollo*. NTRS Author Affiliations: Johnson Space Center, BioTechnology, Inc., National Aeronautics and Space Administration NTRS Report/Patent Number: LC-75-600030 NTRS Document ID: 19760005580 NTRS Research Center: Headquarters (HQ). 1st Jan. 1975. URL: https://ntrs.nasa.gov/citations/19760005580 (visited on 14/05/2025).

[27] Lauren Lindsey. *Ingenuity Mars Helicopter*. NASA Science. Running Time: 137 Section: Ingenuity (Helicopter). 8th Apr. 2024. URL: https://science.nasa.gov/mission/mars-2020-perseverance/ingenuity-mars-helicopter/ (visited on 14/05/2025).

[28] Evan Ackerman. "How NASA Designed a Helicopter That Could Fly Autonomously on Mars". In: *IEEE Spectrum* (17th Feb. 2021). URL: https://spectrum.ieee.org/nasa-designed-perseverance-helicopter-rover-fly-autonomously-mars (visited on 14/05/2025).

[29] Abbey A. Donaldson. *After Three Years on Mars, NASA's Ingenuity Helicopter Mission Ends*. NASA. 25th Jan. 2024. URL: https://www.nasa.gov/news-release/after-three-years-on-mars-nasas-ingenuity-helicopter-mission-ends/ (visited on 14/05/2025).

[30] CSS Electronics. *CAN Bus - The Ultimate Guide*. 21st June 2023. URL: https://www.csselectronics.com/pages/can-bus-ultimate-guide (visited on 11/05/2025).

[31] CAN in Automation. *History of CAN technology*. URL: https://can-cia.org/can-knowledge/history-of-can-technology (visited on 11/05/2025).

[32] CAN in Automation. *CAN knowledge*. URL: https://can-cia.org/can-knowledge (visited on 11/05/2025).

[33] Holger Zeltwanger. "CAN FD: Status of international standardization and lessons learned from the first applications". CAN in Space workshop 2017, 15th June 2017. URL: https://indico.esa.int/event/162/contributions/1208/ (visited on 11/05/2025).

[34] CAN in Automation. *Physical layer options*. URL: https://can-cia.org/can-knowledge/physical-layer-options (visited on 18/05/2025).

[35] Dejan Gacnik. "New Space era with Controller Area Network bus". CAN in Space workshop 2019, 13th June 2019. URL: https://indico.esa.int/event/276/contributions/4545/ (visited on 11/05/2025).

[36] Alicia Cermak. *SMART-1*. Section: SMART-1 (Small Missions for Advanced Research in Technology 1). 22nd Dec. 2017. URL: https://science.nasa.gov/mission/smart-1/ (visited on 12/05/2025).

[37] Olivier Notebaert. "Can In Space - a little history". CAN in Space workshop 2017, 14th June 2017. URL: https://indico.esa.int/event/162/contributions/1188/ (visited on 11/05/2025).

[38] Krystian Borodacz, Cezary Szczepański and Stanisław Popowski. "Review and selection of commercially available IMU for a short time inertial navigation". In: *Aircraft Engineering and Aerospace Technology* 94.1 (26th July 2021). Publisher: Emerald Publishing Limited, pp. 45–59. ISSN: 1748-8842. DOI: 10.1108/AEAT-12-2020-0308. URL: https://www.emerald.com/insight/content/doi/10.1108/aeat-12-2020-0308/full/html (visited on 12/05/2025).

[39] SBG Systems. *IMU - Inertial Measurement Unit*. URL: https://www.sbg-systems.com/glossary/inertial-measurement-unit-imu-sensor/ (visited on 13/04/2025).

[40] Neil M Barbour. "Inertial navigation sensors". In: *NATO RTO Lecture Series, RTO-EN-SET-116, Low-Cost Navigation Sensors and Integration Technology* (2010). Publisher: Citeseer. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9dba30cad95662bceb6c0fce6e6c8bc283742e9a (visited on 12/05/2025).

[41] Felix Rehrmann et al. "A Miniaturised Space Qualified MEMS IMU for Rover Navigation - Requirements and Testing of a Proof of Concept Hardware Demonstrator". In: ASTRA 2011. Noordwijk, The Netherlands, 12th Apr. 2011. URL: https://www.researchgate.net/publication/277331920_A_Miniaturised_Space_Qualified_MEMS_IMU_for_Rover_Navigation_-_Requirements_and_Testing_of_a_Proof_of_Concept_Hardware_Demonstrator (visited on 12/05/2025).

[42] Wanliang Zhao et al. "Navigation Grade MEMS IMU for A Satellite". In: *Micromachines* 12.2 (2021). ISSN: 2072-666X. DOI: 10.3390/mi12020151. URL: https://www.mdpi.com/2072-666X/12/2/151.

[43] Microchip. *32-bit Arm Cortex-M7 MCUs with FPU, Audio and Graphics Interfaces, High-Speed USB, Ethernet, and Advanced Analog SAM E70/S70/V70/V71*. 2023. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/DataSheets/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527.pdf (visited on 13/04/2025).

[44] Microchip. *Rad-Hard 32-bit Arm® Cortex®-M7 Microcontroller for Aerospace Applications SAMRH71*. 2024. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/AERO/ProductDocuments/DataSheets/60001593J.pdf (visited on 13/04/2025).

[45] Texas Instruments. *TCAN340x-Q1 3.3V Automotive CAN FD Transceivers with Standby Mode and ±58V bus Standoff*. June 2024. URL: https://www.ti.com/lit/ds/symlink/tcan3404-q1.pdf (visited on 13/04/2025).

[46] NXP Semiconductors. *TJA1462 CAN FD signal improvement transceiver with Standby mode*. 12th Feb. 2025. URL: https://www.nxp.com/docs/en/data-sheet/TJA1462.pdf (visited on 13/04/2025).

[47] TDK InvenSense. *High Performance Automotive 6-Axis MotionTracking Device IAM-20680HP*. 10th Dec. 2020. URL: https://invensense.tdk.com/wp-content/uploads/2021/11/DS-000409-IAM-20680HP-v1.2-Typ.pdf (visited on 13/04/2025).

[48] *KiCad EDA*. URL: https://www.kicad.org/ (visited on 13/04/2025).

[49] Microchip. *MPLAB® XC32 Compiler*. URL: https://www.microchip.com/en-us/tools-resources/develop/mplab-xc-compilers/xc32 (visited on 02/05/2025).

[50] Microsoft. *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (visited on 02/05/2025).

[51] Microchip. *MPLAB® Extensions for VS Code®*. URL: https://www.microchip.com/en-us/tools-resources/develop/mplab-extensions-vs-code (visited on 02/05/2025).

[52] Microchip. *MPLAB® X IDE*. URL: https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide (visited on 02/05/2025).

[53] Microchip. *ATATMEL-ICE*. URL: https://www.microchip.com/en-us/development-tool/atatmel-ice (visited on 02/05/2025).

[54] Tobias Hammer. *HTerm*. der-hammer. URL: https://www.der-hammer.info/pages/terminal.html (visited on 02/05/2025).

[55] normaldotcom. *normaldotcom/cangaroo*. URL: https://github.com/normaldotcom/cangaroo (visited on 02/05/2025).

[56] NanjingQinhengMicroelectronics. *USB to UART Bridge Controller CH9340*. URL: https://www.wch-ic.com/products/CH9340.html (visited on 18/05/2025).

[57] TDK InvenSense. *MPU-6050*. URL: https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/ (visited on 18/05/2025).

[58] Microchip. *ATmega324PB*. URL: https://www.microchip.com/en-us/product/atmega324pb (visited on 18/05/2025).

[59] Infineon Technologies AG. *AN220278 CAN FD usage in TRAVEO™ T2G family*. Traveo Documentation. 20th June 2024. URL: https://documentation.infineon.com/traveo/docs/yar1680597282748_2 (visited on 20/05/2025).

[60] Samtec. *TCSD-20-S-10.00-01-N - IDC Socket Cable Assemblies, 2.00mm Pitch*. URL: https://www.samtec.com/products/tcsd-20-s-10.00-01-n (visited on 18/05/2025).

[61] dsherman26. *Harmony 3 and mcan*. Microchip Community Forum. 18th Jan. 2022. URL: https://forum.microchip.com/s/topic/a5C3l000000McnPEAS/t379186 (visited on 18/05/2025).

[62] g.oberhammer. *SAM V71 Xplained Ultra MCAN not working*. Microchip Community Forum. 8th Apr. 2022. URL: https://forum.microchip.com/s/topic/a5C3l000000Md42EAC/t380217 (visited on 18/05/2025).

A Bill of Materials

Component description	Value or part number	Quantity
Capacitor	0.01uF	1
Capacitor	0.1uF	20
Capacitor	0.47uF	1
Capacitor	1uF	1
Capacitor	2.2uF	1
Capacitor	4.7uF	2
Resistor	2.2Ohm	1
Resistor	120Ohm	2
Resistor	10kOhm	3
Resistor	33kOhm	3
Inductor	10uH	1
Ferrite bead	470Ohm	2
USART 2 main header	1x4 header	1
USART 2 side header, Erase header	1x2 header	2
SWD header	2x3 header	1
Ground header	2x4 header	1
Main bus top header	TW-20-09-F-D-305-SM-A	2
Main bus bottom header	CLT-120-02-F-D-A-TR	2
IMU	IAM-20680HP	1
CAN transceiver	TCAN3404DRQ1	2
Microcontroller	ATSAMV71Q21B-AAB	1

Table A.1: Bill of materials

B Test Results

This appendix contains the UART and CAN interface test results given by the testing tool. All the tests except the tests in table B.7 were conducted with a UART baud rate of 57600 Bd and consequently, there weren't failures needing a microcontroller reset. The tests in table B.7 were conducted with a baud rate of 912600 Bd and without the CAN tasks enabled in firmware, which necessitated a reset after flashing new firmware.

The tables were deliberately given in the same format as in the previous work[13] on the flight computer. This made tables B.1 to B.4 directly comparable with the previous work's results to show improvements.

Tables B.1 to B.7 contain groups of 10 UART loopback tests each with various parameters. Tables B.8 and B.9 contain groups of 10 CAN loopback tests each with various parameters.

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	50	50	50	2.206	56.6	12800	12500
2	50	50	50	2.206	56.3	12800	12500
3	50	50	50	2.211	56.1	12800	12500
4	50	50	50	2.211	56.4	12800	12500
5	50	50	50	2.211	56.1	12800	12500
6	50	50	50	2.206	56.5	12800	12500
7	50	50	50	2.206	56.7	12800	12500
8	50	50	50	2.206	57.0	12800	12500
9	50	50	50	2.211	57.0	12800	12500
10	50	50	50	2.206	56.4	12800	12500

Table B.1: UART loopback test, 50 packets, 250 bytes of data per packet, 57600 Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	50	50	50	0.215	18.5	1205	905
2	50	50	50	0.209	17.5	1161	861
3	50	50	50	0.189	18.2	1115	815
4	50	50	50	0.206	7.9	1146	846
5	50	50	50	0.197	18.9	1113	813
6	50	50	50	0.195	9.2	1119	819
7	50	50	50	0.207	18.2	1168	868
8	50	50	50	0.206	17.8	1212	912
9	50	50	50	0.218	7.5	1248	948
10	50	50	50	0.205	18.1	1146	846

Table B.2: UART loopback test, 50 packets, 1-32 bytes of data per packet, 57600 Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	100	100	100	0.393	7.8	2187	1587
2	100	100	100	0.390	6.6	2220	1620
3	100	100	100	0.387	18.1	2201	1601
4	100	100	100	0.400	13.4	2310	1710
5	100	100	100	0.402	18.0	2264	1664
6	100	100	100	0.403	6.9	2283	1683
7	100	100	100	0.379	16.7	2214	1614
8	100	100	100	0.389	18.0	2269	1669
9	100	100	100	0.340	15.8	2256	1656
10	100	100	100	0.403	17.9	2320	1720

Table B.3: UART loopback test, 100 packets, 1-32 bytes of data per packet, 57600 Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	200	200	200	0.814	18.4	4639	3439
2	200	200	200	0.772	10.0	4361	3161
3	200	200	200	0.791	18.0	4611	3411
4	200	200	200	0.780	16.9	4556	3356
5	200	200	200	0.817	18.6	4677	3477
6	200	200	200	0.824	15.9	4613	3413
7	200	200	200	0.756	6.2	4294	3094
8	200	200	200	0.797	14.5	4593	3393
9	200	200	200	0.764	17.6	4450	3250
10	200	200	200	0.769	9.9	4358	3158

Table B.4: UART loopback test, 200 packets, 1-32 bytes of data per packet, 57600 Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	1000	1000	1000	22.307	35.2	128942	122942
2	1000	1000	1000	22.803	32.9	131714	125714
3	1000	1000	1000	22.205	5.5	128258	122258
4	1000	1000	1000	22.692	12.9	131109	125109
5	1000	1000	1000	22.781	10.5	131598	125598
6	1000	1000	1000	22.780	50.9	131789	125789
7	1000	1000	1000	22.953	6.9	132497	126497
8	1000	1000	1000	23.632	49.5	136720	130720
9	1000	1000	1000	22.034	50.3	127472	121472
10	1000	1000	1000	23.006	50.9	133071	127071

Table B.5: UART loopback test, 1000 packets, 1-250 bytes of data per packet, 57600 Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	10000	9702	9702	64.380	19.9	370596	310596
2	10000	9738	9738	64.311	11.0	369055	309055
3	10000	9716	9716	64.648	22.5	369800	309800
4	10000	9724	9724	65.037	18.4	373341	313341
5	10000	9730	9730	64.609	20.6	370805	310805
6	10000	9696	9696	64.496	11.8	369757	309757
7	10000	9697	9697	64.354	19.9	369702	309702
8	10000	9693	9693	64.587	22.4	370574	310574
9	10000	9729	9729	65.040	9.6	372283	312283
10	10000	9743	9743	64.678	22.0	372068	312068

Table B.6: UART loopback test, 10000 packets, 1-61 bytes of data per packet, 57600 Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	100000	100000	100000	28.486	2.2	2246461	1646461
2	100000	100000	100000	28.529	2.2	2248690	1648690
3	100000	100000	100000	28.524	3.5	2249538	1649538
4	100000	100000	100000	28.574	2.2	2248067	1648067
5	100000	99909	87522	28.652	2.3	2254402	1654402
6	100000	100000	100000	28.583	2.4	2248912	1648912
7	100000	100000	100000	28.611	2.1	2251732	1651732
8	100000	100000	100000	28.647	2.4	2248694	1648694
9	100000	100000	100000	28.705	2.2	2250192	1650192
10	100000	100000	100000	28.554	2.2	2249060	1649060

Table B.7: UART loopback test without CAN, 100000 packets, 1-32 bytes of data per packet, 912600 Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	200	200	200	0.769	18.7	4347	3147
2	200	200	200	0.819	18.7	4480	3280
3	200	200	200	0.833	19.2	4700	3500
4	200	200	200	0.846	18.5	4749	3549
5	200	200	200	0.862	6.5	4704	3504
6	200	200	200	0.816	18.6	4536	3336
7	200	200	200	0.860	8.5	4522	3322
8	200	200	200	0.779	19.8	4383	3183
9	200	200	200	0.839	22.1	4745	3545
10	200	200	200	0.788	19.6	4468	3268

Table B.8: CAN loopback test, 200 packets, 1-32 bytes of data per packet, 57600 UART Bd

Test iteration	Packets sent	Packets received	Correct packets	Receive time (s)	Receive delay (ms)	Total data sent (byte)	Useful data sent (byte)
1	10000	9556	9533	64.683	25.5	371733	311733
2	10000	9495	9469	64.428	13.9	369324	309324
3	10000	9461	9394	64.086	22.2	368620	308620
4	10000	9595	9594	64.965	9.1	371938	311938
5	10000	9535	9517	64.335	11.2	369967	309967
6	10000	9565	9553	63.932	10.1	367146	307146
7	10000	9593	9577	64.847	23.7	371312	311312
8	10000	9588	9574	64.918	23.6	372098	312098
9	10000	9455	9414	64.466	22.7	370817	310817
10	10000	9532	9517	64.399	25.7	369969	309969

Table B.9: CAN loopback test, 10000 packets, 1-61 bytes of data per packet, 57600 UART Bd

Non-exclusive licence to reproduce thesis and make thesis public

I, Ants Kärner,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   "Development for Integrating KuupKulgur's On-Board Flight Computer",

   supervised by Tarvi Tepandi and Ric Dengel;

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Ants Kärner
20.05.2025