Tartu University

Faculty of Science and Technology

Institute of Technology

Artur Eksi

Development of a Prototype Flight Computer for KuupKulgur

Bachelor's thesis (12 ECTS)
Computer Engineering

Supervisors:

MSc Ric Dengel
MSc Tarvi Tepandi

Tartu 2024

Abstract/Resümee

Development of a Prototype Flight Computer for KuupKulgur

With mankind's recent resurgence in interest in the Moon, a number of lunar rovers are set to be deployed, among these are a number of student rovers. In this light, Tartu Observatory is developing a microrover of its own, KuupKulgur. In this body of work, a prototype on-board flight computer is developed for the lunar rover KuupKulgur. This involved the selection of a radiation-hardened microprocessor, the creation of a development board, and the development of firmware and interface testing tools. Additionally, tests were performed on the interfaces.

CERCS: T120 Systems engineering, computer technology; T170 Electronics; T320 Space technology; [1]

Keywords: KuupKulgur, lunar rover, on board computer, flight computer

Kulguri lennuarvuti väljatöötamine KuupKulgurile

Lähiajal on inimkonna huvi Kuu vastu taaselavnenud, sellega kaasnevalt on plaanitud Kuule saata mitmeid kulgureid, nende hulgas ka tudengikulgureid. Hetkel arenduses olevate tudengikulgurite seas on ka Tartu Observatooriumi oma mikrokulgur - KuupKulgur. Selles töö raames arendati Kuupkulguri lennuarvuti prototüüp. See hõlmab kiirguskarastatud mikroprotsessori valimist, arendusplaadi loomist, manusvara ja liideste testimis-tööriista arendamist ning liideste testimist.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T170 Elektroonika; T320 Kosmosetehnoloogia; [1]

Märksõnad: KuupKulgur, kuukulgur, pardaarvuti, lennuarvuti

Contents

List of Figures

List of Tables

Abbreviations

ASF4 Advanced Software Framework 4. 16, 17

ASIC Application-Specific Integrated Circuit. 12

CADRE Cooperative Autonomous Distributed Robotic Exploration. 11

CAN Controller Area Network. 11, 12, 14, 15, 20–23

COTS Commercial Off the Shelf. 12, 21

CSP Cubestat Space Protocol. 11, 19

ESA European Space Agency. 12

FD Flexible Data-Rate. 12, 14, 15

FOSS Free and Open-Source Software. 15, 18

FPGA Field-programmable Gate Array. 12

GPIO General-Purpose Input/Output. 15

IC Integrated Circuit. 20

ICP Internal Communication Protocol. 11, 19

IDE Integrated Development Environment. 17, 18

I^{2}C Inter-Integrated Circuit. 14, 20–23

LFBGA Low Profile Fine-Pitch Ball Grid Array. 15

LQFP Low Profile Quad Flat Package. 15

MAPP Mobile Autonomous Prospecting Platform. 11

MCU Microcontroller Unit. 13, 15, 16, 18, 21–23

MIT Massachusetts Institute of Technology. 11

NASA National Aeronautics and Space Administration. 11

PCB Printed Circuit Board. 5, 15, 16

RAM Random-Access Memory. 12

RFID Radio-Frequency IDentification. 20

RTOS Real-Time Operating System. 18

SMD Surface-Mount Device. 15

SPARC-V8 Scalable Processor ARChitecture version 8. 12

SPI Serial Peripheral Interface. 14, 20–23

SWD Serial Wire Debug. 16, 18

TTL Transistor-Transistor Logic. 20

UART Universal Asynchronous Receiver-Transmitter. 6, 14, 18, 20–23, 32, 33

USB Universal Serial Bus. 20

VHDL Very High Speed Integrated Circuit (VHSIC) Hardware Description Language. 12

VHSIC Very High Speed Integrated Circuit. 8, 12

VIPER Volatiles Investigating Polar Exploration Rover. 11

1 Introduction

With mankind's recent resurgence in interest in visiting the moon for the first time in over 50 years, and setting up a long-term presence [2], a large number of rovers are intended to be deployed alongside for scientific research [3–8]. Among the rovers made by space agencies and commercial providers, there is a rising amount of small student rovers intended to fulfil different scientific studies [7, 8].

In that light, after the development of two CubeSats, Estcube 1 and 2 [9], Tartu Observatory has decided to start the development of a microrover of their own named KuupKulgur. It aims to offer a standardized rover for lunar mission payloads so that those wishing to send their scientific instruments to the Moon do not need to design the whole rover but just the payload [10,11]. The planned rover consists of a main chassis that contains all of the electronics required for the operation of the rover and a 2 CubeSat unit [12], or 20x10x10 cm sized payload located on top of the rover as seen in Figure 1.1.

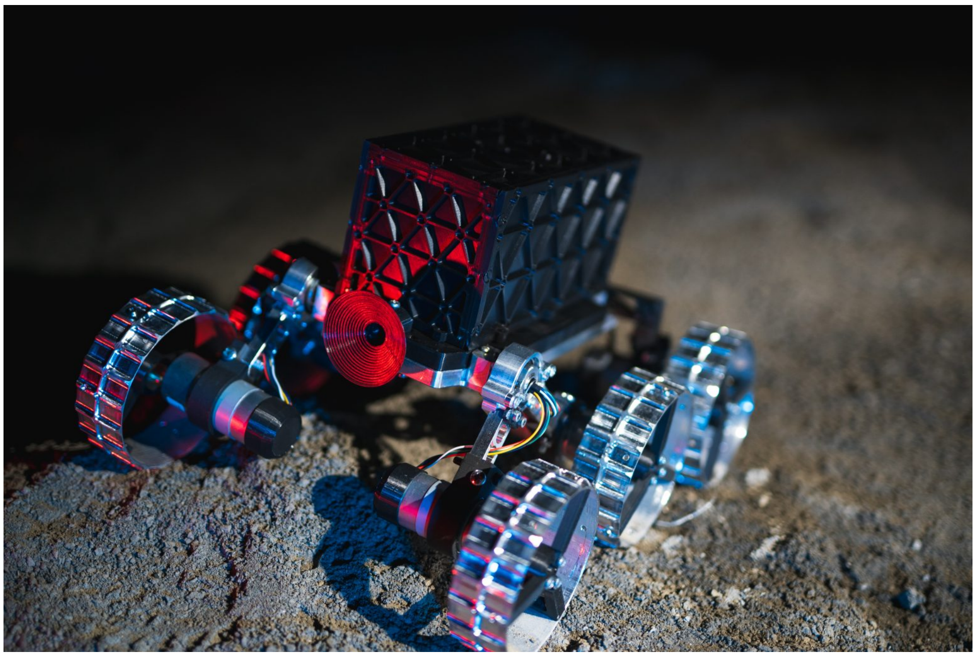

Figure 1.1: Current version of the KuupKulgur Payload Demonstrator Model [13]

1.1 Problem Statement

To control the chassis' electronics and allow for communication between it and the payload, a flight computer and supporting firmware are required.

1.2 Objectives

The aim of this thesis was to create a prototype flight computer for KuupKulgur. This required the completion of several steps:

1. picking an appropriate microcontroller,
2. designing the development board,
3. setting up and developing the baseline firmware.

2 State of the Art

2.1 Modern Lunar Rovers

While the two active NASA rovers on Mars: Curiosity and Perseverance, are the size of cars, most of the planned and launched modern lunar rovers are comparatively small ranging from the size of a large personal computer, to small enough to fit in a hand [14]. These include commercial platforms offered by private companies such as Mobile Autonomous Prospecting Platform (MAPP) by Lunar Outpost, and CuberRover by Astrobotic; National Aeronautics and Space Administration (NASA) projects Cooperative Autonomous Distributed Robotic Exploration (CADRE) and Volatiles Investigating Polar Exploration Rover (VIPER); and student rovers like Iris by Carnegie Mellon University, and AstroAnt by Massachusetts Institute of Technology (MIT) [3–8].

Of these, the Iris rover and CubeRover platform are most similar in size and function to the KuupKulgur rover, but as these rovers are still in development, and with restricted available information, only limited comparisons with KuupKulgur can be made.

2.2 Communication Protocols

To facilitate communication between different parts of a spacecraft, a communication protocol is often used on top of a physical interface. Due to the specific requirements of a project, a custom protocol may be needed, as was the case with Estcube 2 and its Internal Communication Protocol 2 (ICP-2) [15,16]. There have also been attempts at standardisation in the field through the creation of a protocol that can be used on many projects, such as the Cubestat Space Protocol (CSP), which is not physical interface specific [17]. In contrast, there are also protocols directly tied to a physical interface, such as Controller Area Network (CAN).

2.3 Rover Reliability in Space

As a single component or subsystem failure could cause rover failure, it is important to minimize the chances of critical failures, as on-site human intervention is unfeasible.

In firmware, this comes down to minimizing the chances of a critical error in the program causing an irrecoverable situation. While not too difficult on small projects this becomes significantly harder as the scope and amount of contributors increases. To combat this, rules beyond the recommended style and coding standards can be employed such as ”The Power of Ten – Rules for Developing Safety Critical Code,” consisting of 10 generalised rules to improve the verifiability, legibility, and reliability of safety-critical code [18]. Additionally, to prevent issues created during runtime from transmissions over noisy interfaces, cyclic redundancy checks can be used to verify the integrity of messages, and to repair corrupted messages [19].

In hardware, this comes down to combating the problems caused by the conditions of space, such as difficulties dissipating heat, outgassing, and radiation-induced faults in semiconductors. Of these, the reduction of radiation-induced faults such as lattice displacement, ionizing effects, and single-event upsets is the most critical for a microcontroller as it requires bottom-up design considerations [20].

A common approach to this has been to use more resilient manufacturing processes and conduct thorough verification. Methods of improving fault tolerance through the use of multiple reconfigurable cores with software verification have also been explored [21], but due to the added software complexity, and less predictable nature, it is not considered for the KuupKulgur.

While the usage of Commercial Off the Shelf (COTS) components was sufficient in a previous Tartu Observatory project, Estcube 1 [9], it is unlikely to be sufficient for KuupKulgur. Estcube 1 was deployed to low earth orbit where the conditions are more favourable than on a lunar mission, as the Earth's magnetosphere and atmosphere reduce the amount of radiation present significantly [22].

2.4 Microcontroller

As only a small subsection of COTS microcontrollers and microprocessors have been characterized or hardened against radiation-induced faults [23], it was decided to focus on radiation-hardened space-grade products. This has led to a minimal selection of options for projects such as KuupKulgur as many radiation-hardened processors are not publicly available [24]. The two commonly available processor core families among radiation-hardened microprocessors and -controllers are LEON and ARM Cortex.

2.4.1 LEON

The LEON family of soft processors based on the Scalable Processor ARChitecture version 8 (SPARC-V8) instruction set was initially developed by the European Space Agency (ESA) and later by Frontgrade Gaisler [25]. It is defined in Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) and can be implemented in an Application-Specific Integrated Circuit (ASIC) or Field-programmable Gate Array (FPGA) [26, 27]. While it is possible to implement a radiation-hardened LEON microprocessor on a radiation-hardened FPGA for use in the project using the fault-tolerant variants, it'd be impractical as the architectures are also available as ASICs.

In the author's opinion, out of the fault-tolerant LEON processors available [28–34], the most fitting model for the project would've been the GR716B microcontroller by Frontgrade Gaisler. All of the considered models are highly radiation-resistant but the GR716B stood out as it supports CAN Flexible Data-Rate (FD), which offers better support for large transfers over the older 2.0 standard [35], has on-chip Random-Access Memory (RAM) and is a single core processor, leading to less complexity than a multi-core processor. However, it still requires external memory components, is in active development, and has not been released yet, lacks development frameworks and would lead to a more complex development. As such it was ruled out of becoming the flight computer's processor.

2.4.2 ARM Cortex

The ARM Cortex family of microprocessor cores based on the ARM instruction set is developed by Arm Limited and licensed to manufacturers, and its partners [36].

Out of the companies that offer radiation-hardened microcontrollers based on ARM Cortex cores, Vorago [37] and Microchip [38] offer options designed around the more powerful Cortex-M4 and Cortex-M7 cores [39].

Vorago's product lineup includes four radiation-hardened microprocessors based around the Cortex-M4 core, of which two support the required interfaces [40]. Out of the two functionally similar Microcontroller Unit (MCU)s, the VA41630 has additional non-volatile memory at the cost of lower resistance against radiation compared to VA41620 [41]. Microchip offers two radiation-hardened microprocessors, of which the SAMRH71 is the newer model with several improvements compared to the older SAMRH707 [42, 43].

Out of the two ARM Cortex options, the Microchip SAMRH71 and the Vorago VA41630, the SAMRH71 was chosen as it has a wider community around its developer tools, preexisting drivers, lower price, a better selection of interfaces and newer core design. However it does have inferior radiation characteristics to both the GR716B and VA41630. [30, 41, 43–45].

3 Methodology

3.1 Requirements and Constraints

As this piece of work started in the early stages of KuupKulgur, the hardware needed to be easily accessible and affordable for the prototype rover, and didn't have to comply with space-grade standards. However, development was carried out in a way that it would be possible to directly apply as much work as possible from the prototype to the final space-grade rover to reduce time and resource waste.

3.1.1 Interfaces

The required interfaces are:

1. CAN, preferably CAN-FD,
2. Inter-Integrated Circuit (I²C),
3. Serial Peripheral Interface (SPI),
4. Universal Asynchronous Receiver-Transmitter (UART).

3.1.2 Microcontroller

The primary requirements of the microcontroller are:

1. the existence of a radiation-hardened counterpart,
2. driver support for interfaces mentioned above.

3.1.3 Development Board

The primary requirements of the development board are:

1. it has to fit into the existing prototype chassis,
2. all previously mentioned interfaces have to be exposed through connectors,
3. it has to be structured and documented sufficiently to allow for continued development by others,
4. usage and testing should be made convenient with silkscreen labels and the ability of the board to be connected with other subsystems as a tabletop test platform.

3.1.4 Firmware

The primary requirements of the firmware are:

1. it has to be structured and documented sufficiently to allow for continued development by others,
2. it needs to function as a packet router to allow communication between various subsystems,
3. the reliability and speed of transfer needs to be evaluable.

3.2 Component Selection

For the non-radiation-hardened alternative MCU for SAMRH71, it was decided to use the ATSAMV71Q21; although not pin-compatible due to the differences in interfaces, it was the closest match offered by Microchip, and had all of the required interfaces [46]. As this microcontroller came in two packages, 144-lead Low Profile Quad Flat Package (LQFP), and 144-ball Low Profile Fine-Pitch Ball Grid Array (LFBGA), the former was selected as room on the board was plentiful, and the board was intended to be hand-soldered and -verified [47].

To allow for the use of the CAN bus, an external CAN transceiver was required, for this TJA1462AT in the SO8 package was selected as it supported Controller Area Network Flexible Data-Rate (FD), data rates up to 8 megabits/s, and 3.3V logic level [48].

Supporting passive components were all selected in the Surface-Mount Device (SMD) 0603 package (imperial) for its relative ease of soldering, and availability in both component retailers and onsite at Tartu Observatory [49].

All General-Purpose Input/Output (GPIO) and interface connectors were chosen to be generic 2.54mm pitch pin headers as it is the *de-facto* standard for prototyping and development boards [50]. This also allows for the flexibility of replacing it with locking connectors of the same pitch to allow for reliable use inside the prototype chassis. The power connectors were chosen to be generic screw terminals in stock at the Tartu Observatory electronics lab.

3.3 Development Board

The schematic and Printed Circuit Board (PCB) for the development board were developed using KiCad [51], a cross-platform Free and Open-Source Software (FOSS) electronics design automation suite, to make further modifications and developments of the board not reliant on paid licenses or platforms. All of the project files were added to Tartu Observatory's internal GitLab repository.

The board's dimensions were determined by the mechanical constraints of the chassis and the existing electronics stack, which led to a generously sized 96mm by 96mm rounded square shape with four fixed location mounting holes as seen in Figure 3.1. As space on the board was plentiful, and this version was intended to be used in the prototype rover, the design process followed general PCB design practices and no precautions were taken to make it suitable for use in outer space.

This led to a simple 4-layered PCB with the outermost layers intended for signals, and the middle two for ground plane and power traces respectively. The MCU was located in the centre, with the interfaces being situated on the right side of the board, and the GPIO, power connectors

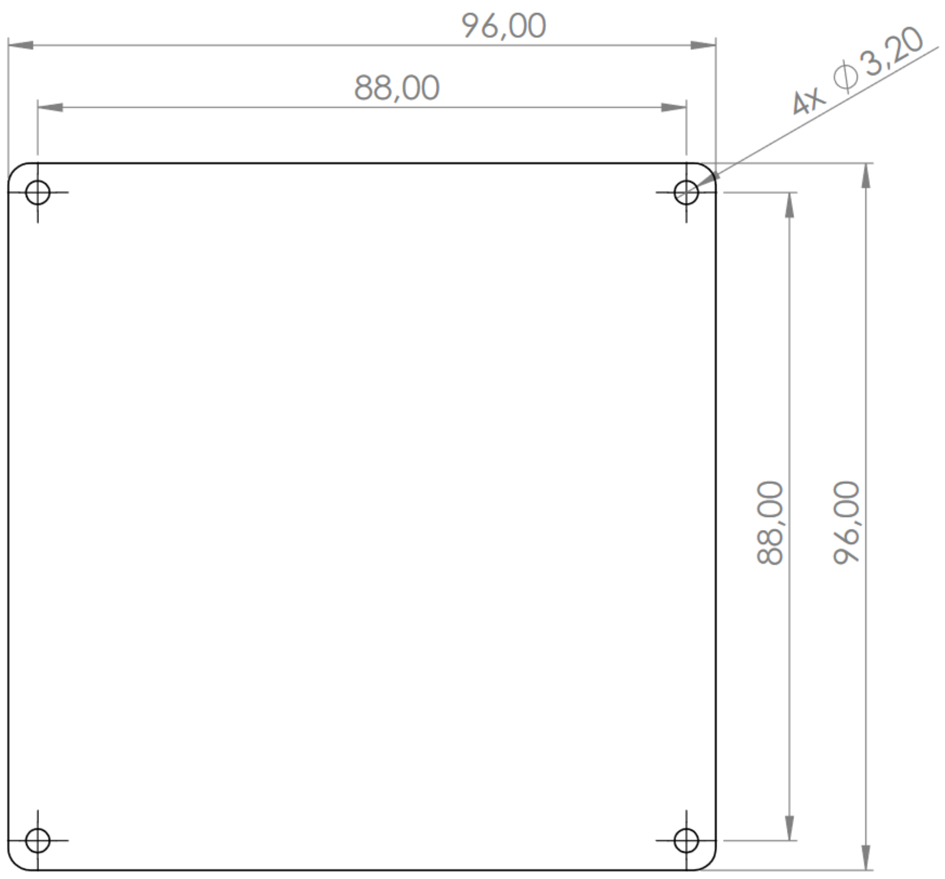

Figure 3.1: PCB constraints for the flight computer

and extra ground connectors being located on the left side. A 3-dimensional rendering of the flight computer's development board can be seen in Figure 3.2.

For extra convenience while prototyping and testing, four square test points connected to the ground plane were added near interface headers to connect the oscilloscope or signal analyzer grounding crocodile clips directly to the board's edge. To further improve the user experience, components and the pinouts of connectors were labelled. For the purposes of debugging and programming, a manual reset button and Serial Wire Debug (SWD) header are provided alongside a flash erase header that can be shorted to clear firmware that has disabled the ability to program the MCU.

Out of the three voltages used on the board, 3.3 and 5 volts are supplied by the two screw terminals, and the third voltage, 1.8V, for the MCU's core, is created by the MCU's internal voltage regulator. As only the CAN transceiver requires 5V, the rest of the board can function with only a 3.3V supply [48].

3.4 Firmware

3.4.1 Development Environment

The processor used was compatible with the current framework offered by Microchip - Harmony 3, and the older framework initially developed by Atmel - Advanced Software Framework

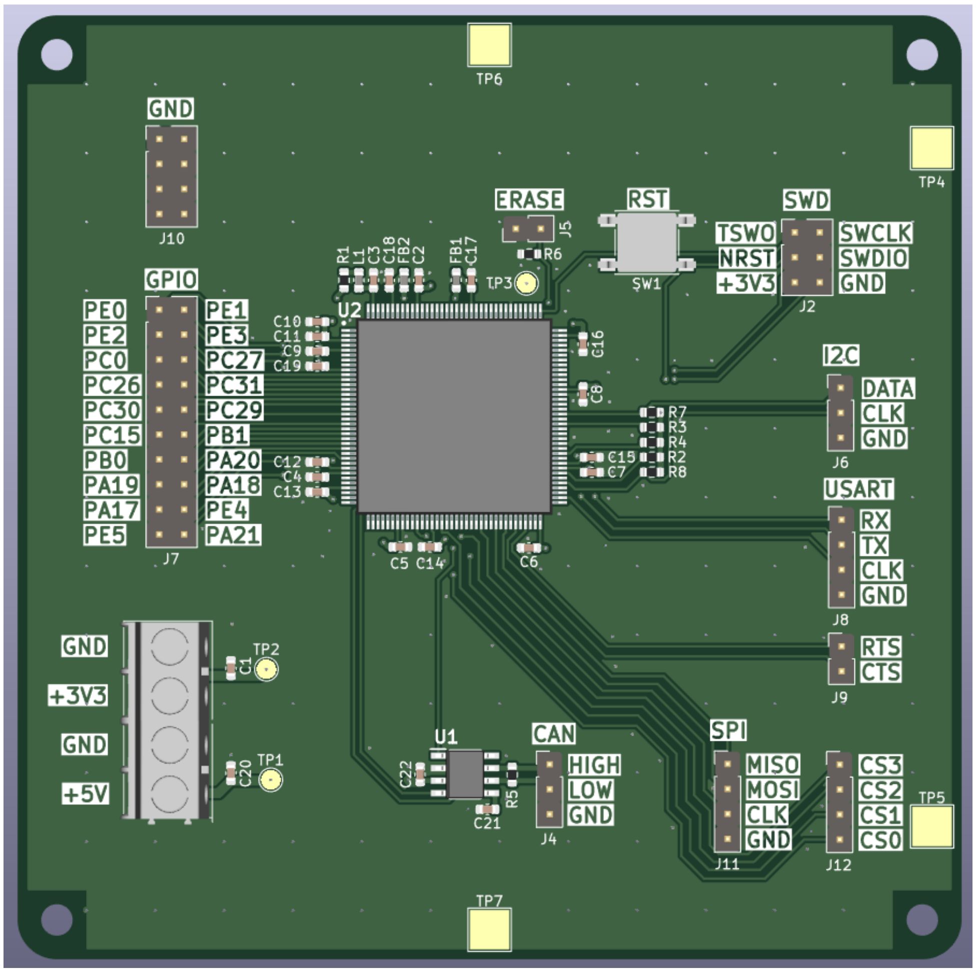

Figure 3.2: 3-dimensional view of the flight computer in KiCad

4 (ASF4) [52, 53]. It was decided that for a project of this scope, register-level programming wouldn't be viable, and that a framework would be used. Out of the two options ASF4 was used as there had been constant issues with the initialisation of the base project on several different attempts using Harmony 3. In addition, it was decided to use the C programming language for the project as it is already widely used in the field and the framework is based around it.

With ASF4, it was possible to use Atmel START - a web-based project configuring tool, which was used to set up and generate the basis of the project [54]. The project's base included interface drivers, FreeRTOS [55], and clock and pin configurations. Alongside necessary components, it also included examples and documentation, which greatly aided in the interfaces' initial setup.

The base project generated with Atmel START can be natively imported into two of Microchip's Integrated Development Environments (IDE), Microchip Studio [56] and MPLAB® X IDE [57], which include the necessary compiler and support for Microchip's programmers and debuggers. As the author was more experienced with Microchip Studio and the Tartu Observatory had an Atmel-ICE debugger available, which is supported by the IDE, the project was developed using Microchip Studio and the microprocessor programmed using the Atmel-ICE debugger [58]. The Microchip Studio also supports programmers not made by Microchip, however they are restricted to programming and do not allow for debugging inside of the IDE.

As Microchip Studio isn't open-source software, or available on Linux, and the Atmel-

ICE debugger could be discontinued, alternative options were considered for all parts of the development process. For the code development, any IDE or text editor could be used and manually compiled using the project makefile generated with Atmel START and Arm GNU Toolchain [59,60]. For the debugging and programming of the microcontroller, OpenOCD [61], a FOSS debugger software, could possibly be used in conjunction with one of the supported SWD-compatible debuggers and programmers [62]. However, these options were not tested.

Because the number of tasks meant to run in parallel on the MCU was unknown, it was decided that an operating system would be required for flexibility. The two primary types of operating systems to pick from were general-purpose operating systems, which are intended for human interfacing applications, and Real-Time Operating Systems (RTOS), which provide a deterministic execution pattern. As this application requires predictability to ensure stable and continued operation, it was decided that an RTOS optimised for microcontrollers was to be used [63]. Multiple options were considered for the RTOS to be used; however, since FreeRTOS 10.0.0 is already supported and available on Atmel START, it was chosen [54].

To allow for the continued development of the firmware, all of the files associated with the firmware and testing tool were stored on Tartu Observatory's internal GitLab and documented, and a short guide was added on how to set up the necessary software used in the project.

3.4.2 Firmware Architecture

The flight computer is intended to serve as the central control hub for the rover that connects several modules and sensors together. To facilitate this firmware and a communication protocol were developed.

The method implemented is a balance between memory efficiency and security against a task stalling all other internal communications with a large transfer, such as an image. It is built up of three tasks, of which the interface and UART receiving tasks are used for external interfaces while the routing task routes packets internally, as seen in Figure 3.3. The single-core processor's time is split between the tasks using time slice scheduling, a scheduling algorithm that switches the task being processed every system tick, in this case, every millisecond.

For a task or function to transfer data between external interfaces or other tasks it uses an outgoing buffer to send out packets while receiving packets from other locations into its incoming buffer. The buffers were separated to allow for a simultaneous two-sided internal connection while enforcing a system of one reader and one writer per buffer, which inherently helps avoid access conflicts. If a task only needs one-way communication, then it only needs one of the buffers; for example, the UART task only has an outgoing buffer as it is strictly for receiving data from an outside interface and relaying it to the wider system.

For the buffers, it was decided to use ring buffers as it's designed with principles of asynchronous 'first-in, first-out' reading and writing with a seamless wraparound at the end of the buffer [64]. Existing solutions, such as the FreeRTOS queue, were considered. However, these lacked some necessary functionality, such as variable item size and peeking, which would allow to read data without removing it from the buffer [65]. For that reason, a custom variant was implemented. While it is intended to be used as a part of a single-reader, single-writer system, it supports multiple readers and writers through the use of locks, also known as mutexes, which only allow one of the requesters to access the resource at a time.

The routing of packets between the buffers is handled by the routing task, which retrieves packets from outgoing buffers and inserts them into the incoming buffer tied to the destination address. In addition, the task was designed to allow, with further development, for the verification of packet integrity and subsequent requests for a re-transmission of the faulty packet.

To aid in the monitoring and debugging of the flight computer, a custom logging system was implemented to send to an external device. A log can be created from anywhere (besides the logging system itself, which could cause an infinite loop), after which it gets placed into a logging ring buffer and then routed to the specified logging interface. It features four log importance levels so that less relevant logs can be disabled during regular operation.

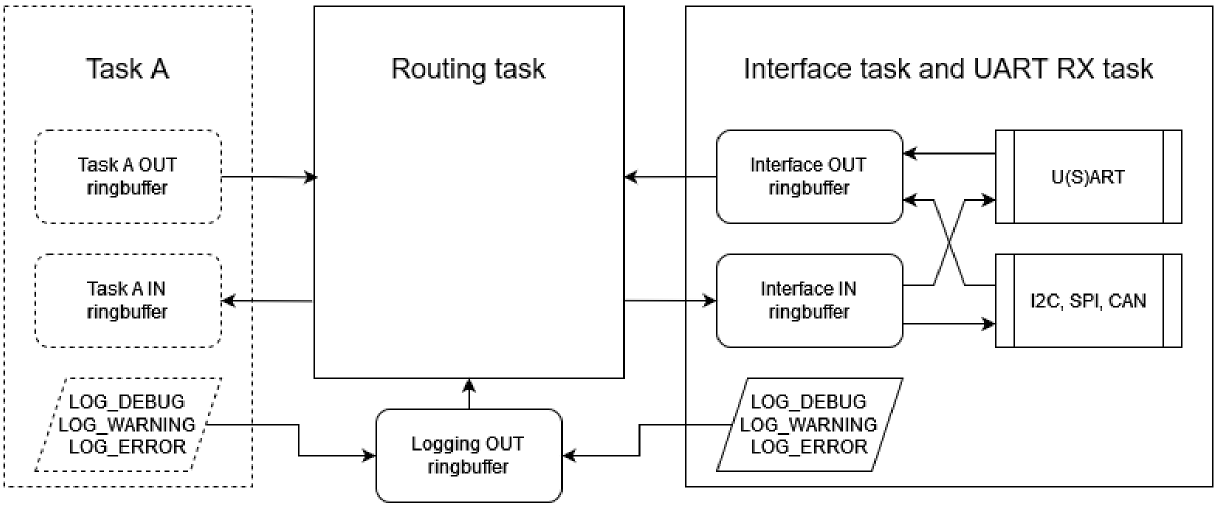

Figure 3.3: Packet routing task architecture with a hypothetical task A

3.4.3 Communication Protocol

The communication protocol is used in the routing of packets between interfaces and tasks in the flight computer and is largely inspired by the ICP-2 used in Estcube 2 and CSP as the intended applications are similar [15, 16]. However, neither of them could be directly used as the ICP-2 requires two additional data lines as a part of it and the CSP uses fixed size buffer elements, severely limiting the maximum size of a packet [15, 66]. The full description of the protocol is available in addendum A.

The protocol is intended to be a balance between low memory requirements, as memory is limited, and low packet overhead, as most packets are projected to be short while retaining the ability to transmit large quantities of data efficiently, such as images from a camera. To achieve this several design choices were implemented:

1. The maximum length of packets was set to 256 bytes; this led to the packet length field being a single byte long and set the minimum buffer size of a receiver to a reasonable amount.

2. Packet types were limited to 4 and the length of the ID to 6 bits to keep the protocol simple and reduce protocol overhead.

3. Packet origin and destination addresses were limited to 1 byte each to reduce the protocol overhead.

Additionally, methods for verifying packet integrity and message types for data re-transmission requests were added to allow for better recovery from errors.

3.4.4 UART communicator tool

To facilitate the receiving of logs from the device and testing interface speed and reliability, a UART communicator script was developed using Python 3 and Universal Serial Bus (USB)-Transistor-Transistor Logic (TTL) converter module connecting the flight computer and a personal computer [67]. It features two methods of operation: a listen mode and a testing mode. The listening mode allows for the display of logs and data transmitted from the flight computer over UART. The testing mode allows for the functionality verification of I²C and SPI connected devices and for the speed and reliability testing of the UART interface with varying packet sizes and counts.

3.4.5 Interface Tests

To verify the functionality of the interfaces, several tests were conducted with devices available at the time and do not reflect the peripherals that are to be used on those interfaces.

The CAN interface was tested earlier in the development using the CAN driver example with an external CAN-SPI converter and an Arduino Nano.

The I²C, SPI and UART interfaces were tested using the tester mode of the UART communicator script run on a separate computer connected to the flight computer over UART using a USB-TTL converter module [67].

The UART interface was tested using a series of loopback packets whose origin and destination are both the UART interface. The sent packets were recorded and then compared against the received packets. Several batches of tests were conducted varying the amount of packets sent and the size of the packets.

The SPI interface was tested using the RC522 Radio-Frequency IDentification (RFID) reader module based on the MFRC522 IC connected to the SPI bus [68]. The test involved sending two command packets over UART to the SPI test device to first write a value to one of the registers and then read it. The intended result involves the flight computer returning a reply-type packet with the value written to the register.

The I²C interface was tested using the 3-axis accelerometer and gyroscope MPU-6050 module [70] connected to the I²C bus. The test involved sending a command packet over UART to the I²C interface to read from one of the device's registers with a set value [71]. The intended result involves the flight computer returning a reply-type packet with the set value.

4 The Results

Several options were considered for the radiation-hardened MCU and SAMRH71 was selected with ATSAMV71Q21 as its COTS counterpart. Additional supporting components were selected and developed into a development board compliant with the dimensional requirements of the mission (3.1).

For the development of the firmware, a development environment was established and documented. In addition, to support further development of the project, documentation was added to both the hardware design files and firmware source files. Firmware was developed for the development board, and an accompanying communication protocol was defined for packet routing and to allow access to the required interfaces.

To verify the functionality of the required interfaces a testing tool was developed and used. The tests on the UART interface showed expected results in some test batches as seen in tables B.1 and B.2 in appendix B, which confirms the basic functionality of the interface and routing system. The tests on the I²C, SPI and CAN interfaces showed the expected results, as described in the methodology, on several separate iterations which confirmed the basic functionality of the interfaces.

5 Analysis and Discussion

5.1 Hardware

The selected MCU fulfils the criteria as described in the requirements and constraints section of the methodology (3.1). Given the availability, environmental and computational constraints, the selected radiation-hardened MCU is the best we can realistically get and initial firmware estimates show that it is sufficient.

The designed development board fulfils the criteria as described in the requirements and serves as a good starting point for the development of additional firmware and the next flight computer hardware revisions.

5.2 Firmware

The firmware fulfils the criteria as described in the requirements and has been documented and structured to serve as a good starting point for additional development. The development environment selected has supported the development of the firmware well; however, the choice has been partially short-sighted as the Atmel START tool is officially depreciated and, due to its online nature, can disappear without notice. Additionally, the SAMRH71 MCU isn't supported by it or the Microchip Studio, meaning that to use the radiation-hardened MCU, the project would need to be ported over to the Harmony 3 framework.

The communication protocol has been sufficient as proof of concept. However, as not all intended functionality has been implemented and tested, it requires additional development.

5.3 Interfaces

The test results from the communicator tool's UART test mode show that packet size does not seem to have a significant influence on the reliability of transmission, but larger packets do improve the overall data transmission rates, as seen in tables B.1 and B.2. Additionally, a significant drop in reliability was detected with an increased amount of packets, resulting in an irrecoverable failure as seen in appendix B, tables B.3 and B.4. Whether this is connected inherently to the communication protocol, architectural or hardware issues is currently unknown and requires further testing.

While the I²C, SPI and CAN interfaces were proven to work to some degree, the tests conducted were very shallow and didn't test all available functionality. Therefore the full functionality can't be confirmed.

6 Conclusion and Future Works

In this thesis, the prototype flight computer for KuupKulgur was developed. This entailed the selection of a MCU, the design of a development board and the development of firmware.

The selected hardware and designed development board fulfils the requirements and will serve as the basis for future revisions. The established development environment for the firmware will be sufficient for further development in the near future but will need to be changed to account for the radiation-hardened MCU. The developed firmware has been proven to work as a packet router and will serve as the basis for additional functionality in the future. The required interfaces were proven to function; however, the reliability of the UART interfaces was shown to vary.

A number of possible future improvements were identified during the thesis:

- the optimisation of the routing system through the addition of a direct copy functionality between ring buffers to reduce the amount of unnecessary copying,
- the further development of the communication standard and its implementation including adding the verification of packet integrity and a method of requesting re-transmissions,
- additional tests for the CAN, I²C and SPI interfaces to verify full functionality and the improvement of the reliability of the UART interface,
- the addition of an internal communication bus connector, established shortly after the design of the development board, for direct connections with other subsystems.

Acknowledgements

I would like to thank my thesis supervisors, Tarvi Tepandi and Ric Dengel, for offering support and advice throughout the project.

Additionally, I would like to thank my British friends who helped with grammar.

/ signed digitally /

Bibliography

[1] Common European Research Classification Scheme (CERCS) https://wiki.ut.ee/download/attachments/16581162/Common%20European %20Research%20Classification%20Scheme.pdf 19.05.2024, 14:06 (UTC)

[2] NASA, Artemis https://www.nasa.gov/humans-in-space/artemis/ 18.05.2024, 13:25 (UTC)

[3] NASA, ”VIPER Volatiles Investigating Polar Exploration Rover” https://science.nasa.gov/mission/viper/ 05.05.2024, 12:41 (UTC).

[4] NASA Jet Propulsion Laboratory, ”CADRE” https://www.jpl.nasa.gov/missions/cadre 05.05.2024, 12:42 (UTC).

[5] K. Cowing, ”Lunar Outpost Delivers First Flight Model Rover in Record Time” https://spaceref.com/newspace-and-tech/lunar-outpost-delivers-first-flight-model-rover-in-record-time/ 05.05.2024, 12:47 (UTC).

[6] Astrobotic, ”16 Years, 11 Contracts, 1 Customizable CubeRover®” https://www.astrobotic.com/lunar-delivery/rovers/our-history/ 05.05.2024, 12:27 (UTC).

[7] Carnegie Mellon University, https://irislunarrover.space/, 05.05.2024, 12:27 (UTC).

[8] Massachusetts Institute of Technology, ”AstroAnt Payload” https://www.tothemoon.mit.edu/astroant 05.05.2024, 13:02 (UTC).

[9] eoPortal, ”ESTCube-1 & -2 (Estonian Student Satellite-1 & -2)” https://www.eoportal.org/satellite-missions/estcube-1 13.05.2024, 14:42 (UTC)

[10] KuupKulgur homepage https://kuupkulgur.space/ 18.05.2024, 12:32 (UTC)

[11] M. L. Plats, ”Teadlased ja tudengid hakkavad arendama esimest Eesti kuukulgurit” https://ut.ee/et/sisu/teadlased-ja-tudengid-hakkavad-arendama-esimest-eesti-kuukulgurit, 2023, 18.05.2024, 13:32 (UTC)

[12] NASA, ”What are SmallSats and CubeSats?” https://www.nasa.gov/what-are-smallsats-and-cubesats/, 18.05.2024, 19:02 (UTC)

[13] S. Quazi, ”KuupKulgur, The Estonian Lunar Rover” https://tospexgroup.space/projects/kuupkulgur/, 19.05.2024, 14:18 (UTC)

[14] N. T. Tillmann, M. Wall, ”Perseverance rover: Everything you need to know” https://www.space.com/perseverance-rover-mars-2020-mission, 05.05.2024, 12:27 (UTC).

[15] S. Tammesoo ”Suhtlusprotokoll ESTCube-2 alamsüsteemide vaheliseks suhtluseks”, 2015, 16–21, HDL:10062/50465

[16] L. E. Lindmaa, ”Interface development for ESTCube-2 cameras and plasma brake”, 2023, 10, HDL:10062/93428

[17] The Cubesat Space Protocol homepage https://libcsp.github.io/libcsp/index.html 17.05.2024, 20:36 (UTC)

[18] M. Hinchey, ”The Power of Ten—Rules for Developing Safety Critical Code”, *Software Technology: 10 Years of Innovation in IEEE Computer*, 2018, 192-200, DOI:10.1002/9781119174240.ch10

[19] P. Koopman, T. Chakravarty, ”Cyclic redundancy code (CRC) polynomial selection for embedded networks” *International Conference on Dependable Systems and Networks*, 2004, Florence, Italy, 145-154, DOI:10.1109/DSN.2004.1311885

[20] H. Bokil, ”COTS Semiconductor Components for the New Space Industry”, *2020 4th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, Penang, Malaysia, 2020, 1-4, DOI:10.1109/EDTM47692.2020.9117834.

[21] C. M. Fuchs, N. M. Murillo, A. Plaat, E. van der Kouwe, D. Harsono, T. P. Stefanov, ”Fault-Tolerant Nanosatellite Computing on a Budget” *2018 18th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Goteborg, Sweden, 2018, 1-8, DOI:10.1109/RADECS45761.2018.9328685

[22] G. Reitz, ”Characteristic of the radiation field in low earth orbit and in deep space” *Zeitschrift für Medizinische Physik Volume 18, Issue 4*, 2008, 233-243, DOI:10.1016/j.zemedi.2008.06.015

[23] D. Sinclair, J. Dyer, ”Radiation Effects and COTS Parts in SmallSats” *27th Annual AIAA/USU Conference on Small Satellites*, Logan, Utah, United States of America, 2013, https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=2934&context=smallsat

[24] Bae Systems, ”Radiation-hardened electronics product guide” file:///C:/Users/eksia/Downloads/Rad+Hardened+Short+Form_product+guide_WEB-2.pdf 17.05.2024, 19:37 (UTC)

[25] Frontgrade Gaisler, https://www.gaisler.com/ 14.05.2024, 16:36 (UTC)

[26] P. Clarke, ”European Space Agency launches free Sparc-like core” https://www.eetimes.com/european-space-agency-launches-free-sparc-like-core/ 26.04.2024, 14:19 (UTC).

[27] Frontgrade Gaisler, ”Processors” https://www.gaisler.com/index.php/products/processors 26.04.2024, 14:32 (UTC).

[28] Microchip, ”Rad-Hard 32 bit SPARC V8 Processor AT697F” https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/Product Documents/DataSheets/doc7703.pdf 27.04.2024, 16:26 (UTC).

[29] Frontgrade Gaisler, ”GR716A - LEON3FT Microcontroller” https://www.gaisler.com/index.php/products/components/gr716/gr716a 14.05.2024, 16:37 (UTC)

[30] Frontgrade Gaisler, ”GR716B - LEON3FT Microcontroller” https://www.gaisler.com/index.php/products/components/gr716/gr716b 14.05.2024, 16:39 (UTC)

[31] Frontgrade Gaisler, ”GR712RC Dual-Core LEON3FT SPARC V8 Processor” https://www.gaisler.com/index.php/products/components/gr712rc 14.05.2024, 16:39 (UTC)

[32] Frontgrade Gaisler, ”GR740 Quad-Core LEON4FT SPARC V8 Processor” https://www.gaisler.com/index.php/products/components/gr740 14.05.2024, 16:40 (UTC)

[33] Frontgrade, Microprocessor UT699 product description https://www.frontgrade.com/product/ut699 14.05.2024, 16:48 (UTC)

[34] Frontgrade, Microprocessor UT700 product description https://www.frontgrade.com/product/ut700 14.05.2024, 16:49 (UTC)

[35] K. Lennartsson, ”Comparing CAN FD with Classical CAN” https://kvaser.com/wp-content/uploads/2016/10/comparing-can-fd-with-classical-can.pdf 17.05.2024, 20:24 (UTC)

[36] Arm limited, ”The Future is Built on arm” https://www.arm.com/company, 01.05.2024, 20:41 (UTC).

[37] Vorago Technologies, https://www.voragotech.com/ 12.05.2024, 20:19 (UTC)

[38] Microchip, https://www.microchip.com/ 12.05.2024, 20:20 (UTC)

[39] Microchip, ”Differences Between Arm® Cortex® Families” https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/32bit-mcu/sam/arm-cortex-differences/ 14.05.2024, 15:19 (UTC)

[40] Vorago Technologies, ”Arm® Cortex®-M4” https://www.voragotech.com/arm-cortex-m4-family, 01.05.2024, 18:22 (UTC).

[41] Vorago Technologies, ”Radiation Hardened VA416X0 32-Bit Arm® Cortex®-M4 (with FPU) microcontroller manufactured with HARDSIL® technology offering best in class radiation performance and latch-up immunity.” Datasheet requested from the manufacturer, 01.05.2024, 19:09 (UTC).

[42] Microchip, ”Rad-Hard Arm® Cortex®-M7 Microcontroller SAMRH707” https://www.microchip.com/en-us/product/SAMRH707, 01.05.2024, 18:43 (UTC).

[43] Microchip, ”SAMRH71 Rad-Hard Arm® Cortex®-M7 Microprocessor” https://www.microchip.com/en-us/product/SAMRH71, 01.05.2024, 18:44 (UTC).

[44] Mouser Electronics, ”VA41630-PQ176F0PBA” https://mou.sr/3QuYrdl 07.05.2024, 13:21 (UTC).

[45] Octopart, ”Microchip SAMRH71F20E-HFB-HP” https://octopart.com/samrh71f20e-hfb-hp-microchip-148990774?r=sp 15.05.2024, 14:11 (UTC)

[46] 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 datasheet https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/Product Documents/DataSheets/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527.pdf 25.04.2024, 13:57 (UTC).

[47] U Choice Manufacturing Inc, ”QFP and BGA Assembly” https://uchoice.ca/qfp-and-bga-assembly/ 25.04.2024, 14:32 (UTC).

[48] NXP, TJA1462AT CAN FD signal improvement transceiver with Standby mode datasheet https://www.nxp.com/docs/en/data-sheet/TJA1462.pdf 25.04.2024, 13:31 (UTC).

[49] I. Poole, ”SMT / SMD Components & packages, sizes, dimensions, details” https://www.electronics-notes.com/articles/electronic_components/surface-mount-technology-smd-smt/packages.php 25.04.2024, 14:09 (UTC).

[50] Gadgettronix, ”Common types of connectors in electronics” https://www.gadgetronicx.com/common-types-connectors-electronics/ 25.04.2024, 15:09 (UTC).

[51] KiCad, https://www.kicad.org/ 25.04.2024, 15:13 (UTC).

[52] Microchip, ”MPLAB® Harmony v3” https://www.microchip.com/en-us/tools-resources/configure/mplab-harmony 26.04.2024, 7:48 (UTC).

[53] Microchip, ”Introduction to ASF4” https://onlinedocs.microchip.com/pr/GUID-2A8AADED-413E-4021-AF0C-D99E61B8160D-en-US-4/index.html 26.04.2024, 7:46 (UTC).

[54] Microchip, Atmel START https://start.atmel.com/ 26.04.2024, 7:43 (UTC).

[55] FreeRTOS, https://www.freertos.org/ 11.05.2024, 11:20 (UTC)

[56] Microchip, ”Microchip Studio for AVR® and SAM Devices” https://www.microchip.com/en-us/tools-resources/develop/microchip-studio 14.05.2024, 23:34 (UTC)

[57] Microchip, ”MPLAB® X IDE” https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide 14.05.2024, 23:35 (UTC)

[58] Microchip Developer Help, ”Programmers and Debuggers” https://developerhelp.microchip.com/xwiki/bin/view/software-tools/programmers-and-debuggers/ 14.05.2024, 22:37 (UTC)

[59] Microchip, ”GCC Compilers for AVR® and Arm®-Based MCUs and MPUs” https://www.microchip.com/en-us/tools-resources/develop/microchip-studio/gcc-compilers 14.05.2024, 23:05 (UTC)

[60] Arm limited, ”Arm GNU Toolchain” https://developer.arm.com/Tools%20and%20 Software/GNU%20Toolchain 14.05.2024, 23:12 (UTC)

[61] OpenOCD, https://openocd.org/ 14.05.2024, 23:19 (UTC)

[62] OpenOCD, ”Open On-Chip Debugger: OpenOCD User’s Guide” https://openocd.org/doc-release/pdf/openocd.pdf 14.05.2024, 23:25 (UTC)

[63] FreeRTOS, ”What is An RTOS?” https://www.freertos.org/about-RTOS.html, 01.05.2024, 14:19 (UTC).

[64] C. Dobson, ”How To Implement A Simple Circular Buffer In C” https://medium.com/@charlesdobson/how-to-implement-a-simple-circular-buffer-in-c-34b7e945d30e, 01.05.2024, 15:58 (UTC).

[65] FreeRTOS, ”xQueueCreate” https://www.freertos.org/a00116.html, 01.05.2024, 16:02 (UTC).

[66] ”The basics of CSP” https://libcsp.github.io/libcsp/basic.html 17.05.2024, 20:36 (UTC)

[67] Oomipood, ”USB-TTL konverter moodul 6-pin CP2102” https://www.oomipood.ee/product/oky3411_usb_ttl_konverter_moodul_6_pin_cp2102 17.05.2024, 10:29 (UTC)

[68] Robolabor, ”RFID module RC522” https://robolabor.ee/en/wireless-modules/909-rfid-module-rc522.html 17.05.2024, 09:18 (UTC)

[69] NXP, ”MFRC522 Standard performance MIFARE and NTAG frontend” https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf 17.05.2024, 09:24 (UTC)

[70] Tinytronics, ”MPU-6050 Accelerometer and Gyroscope 3-Axis Module 3.3V-5V” https://www.tinytronics.nl/en/sensors/acceleration-rotation/mpu-6050-accelerometer-and-gyroscope-3-axis-module-3.3v-5v 17.05.2024, 10:01 (UTC)

[71] InvenSense Inc., ”MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2” https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf 17.05.2024, 10:03 (UTC)

A Communication Protocol

A packet consists of a 4-byte header, 1 to 250-byte data and a 2-byte tail section A.1.

The header section contains all of the data necessary to route and process the packet:

- The type field of the packet header is used to differentiate between the four types of messages:
    1. Command - This type of message is the first message of an exchange from the starting party, it determines the ID to be used throughout the rest of the exchange.
    2. Reply - This type of message is a response to a command message and contains the requested data or an error code.
    3. Data - This type of message is used when the data to be transmitted does not fit into one message. The command or reply message contains the first chunk of data and every subsequent chunk is delivered in a data message with the ID incremented by one from the previous.
    4. Request - This type of message is used to request a re-transmission of a packet that is lost in transmission or fails an integrity check.
- The ID field is primarily used to associate requests and replies, and to ensure data order in data fields but also to allow for requested re-transmissions. It starts from 0 and loops back to it after reaching 63.
- The destination and origin fields are used to store an 8-bit address of the recipient and sender device or 'port'.
- The data length field is used to indicate the length of the following data section of the payload.

The data section contains the payload of the packet and is at least one byte long but no more than 250; if the data being transmitted is more than 250 bytes long, then it is split into several separate messages.

The tail section reserves 2 bytes for error detection and correction such as a 16-bit cyclic redundancy check.

This leads to a total protocol overhead of 6 bytes per packet - a relatively small amount of extra bytes for the most common small transmissions while providing a 250 to 6 useful data to transmitted data ratio for large transfers.

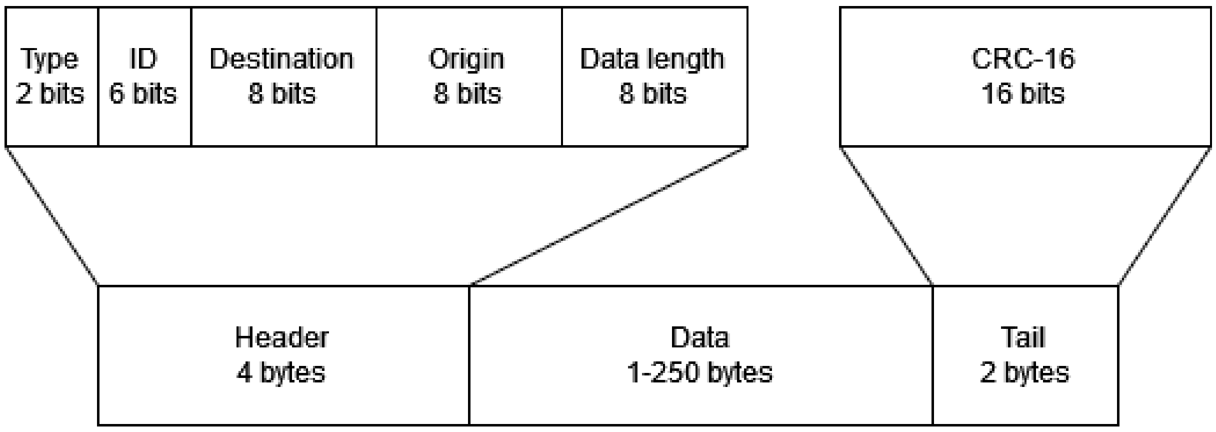

Figure A.1: Communication protocol packet

B Test Results

This appendix contains the UART interface test results from the UART communicator. All of the tests in a group were conducted subsequently at a baud rate of 57600 Bd. The system was only reset after a group of tests if the group had ended in an irrecoverable failure.

Table B.1 consists of a group of 10 tests, each of which had 50 packets of 250 bytes of data to test the performance of maximum-size packets.

Tables B.2, B.3, and B.4 all consist of groups of 10 tests each, each of which contained 50, 100 and 200 packets respectively. Each packet's data section was selected randomly to contain between 1 and 32 bytes.

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	2.306	49	12800	12500
2	50	50	50	2.267	49	12800	12500
3	50	50	50	2.293	48	12800	12500
4	50	50	50	2.328	49	12800	12500
5	50	50	50	2.334	49	12800	12500
6	50	50	50	2.338	50	12800	12500
7	50	50	50	2.284	49	12800	12500
8	50	50	50	2.318	48	12800	12500
9	50	50	50	2.309	49	12800	12500
10	50	50	50	2.241	48	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	49	48	2.035	10	1108	808
2	50	50	48	1.785	9	1136	836
3	50	50	49	1.389	9	1087	787
4	50	50	49	1.853	5	1050	750
5	50	50	49	1.440	8	1124	824
6	50	50	49	1.508	11	1104	804
7	50	50	49	1.627	10	990	690
8	50	50	49	1.405	9	1115	815
9	50	50	49	1.324	4	1266	966
10	50	50	49	1.246	10	1100	800

Table B.2: UART loopback test, 50 packets, 1 to 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	98	1.797	10	2297	1697
2	100	100	98	1.545	10	2110	1510
3	100	100	98	1.687	9	2271	1671
4	100	99	51	1.293	3	2297	1697
5	100	100	0	1.473	4	2274	1674
6	100	98	0	2.001	4	2146	1546
7	100	100	0	0.961	5	2274	1674
8	100	100	0	1.360	5	2170	1570
9	100	100	0	1.407	4	2293	1693
10	100	100	0	1.883	4	2309	1709

Table B.3: UART loopback test, 100 packets, 1 to 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	199	146	2.975	6	4453	3253
2	200	200	0	2.250	4	4656	3456
3	200	200	0	2.845	5	4783	3583
4	200	196	0	3.006	4	4433	3233
5	200	200	0	2.900	5	4616	3416
6	200	183	0	1.786	4	4478	3278
7	200	0	0	0.000	-	4456	3256
8	200	0	0	0.000	-	4642	3442
9	200	0	0	0.000	-	4394	3194
10	200	0	0	0.000	-	4379	3179

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

Non-exclusive licence to reproduce thesis and make thesis public