

Design and Development of the SMILE SXI Radiation Shutter Control System

A thesis by
Ove Lylund

For the degree of
Master of Science in Physics



Department of Physics and Technology

University of Bergen

June 2018

Abstract

This thesis concerns the design, implementation, testing and verification of the radiation shutter control system, an embedded, microcontroller-based system, to be used in the Solar Wind Magnetosphere Ionosphere Link Explorer (SMILE) satellite. SMILE is a scientific space mission for exploring the Earth's radiation belt, ionosphere, and magnetosphere. It involves several institutes where European Space Agency (ESA) and the Chinese Academy of Science (CAS) are the primary contributors. The SMILE satellite is expected to launch towards the end of 2021 and will follow a highly elliptical orbit, reaching almost 128 000 km over the northern hemisphere.

SMILE carries four instruments, including the Soft X-ray Imager (SXI). The SXI features a sensitive soft X-ray detector, built for detecting low energy electrons. The orbit traverses the Earth's radiation belts when the spacecraft is near perigee. The radiation belt is a highly radiative area that the SXI detector cannot handle, and therefore, must be shielded.

The University of Bergen is responsible for the development of the radiation shutter, which consists of the Radiation Shutter Electronics (RSE), the Radiation Shutter Mechanism (RSM), and the control system provided by this thesis. The radiation shutter will significantly reduce the SXI detector's exposure to radiation by covering its focal plane during the crossing of the Earth's radiation belts. In addition, if necessary, it will shield while performing spacecraft manoeuvres where the sensor is prone to stray light.

This thesis describes the implementation of a well-designed, generic and modular control system which accomplish all operations required to protect the detectors. The system holds a custom interrupt driven scheduler which guarantees execution of both soft and hard real-time tasks within the timing constraints. These tasks include communication with the SXIs' control unit, collecting HK-readouts from sensors, and safely operate the shutter through a stepper motor. Furthermore, a flexible stepper driver software is developed with emphasis on predictable stepper motor operations, efficiency, and in-flight adjustable features.

The system has been methodically tested through software analysis and hardware measurements. The tests confirmed that all tasks were completed without any timing violations and the driver software successfully drives the stepper motor.

Acknowledgements

I have worked on this thesis at the Department of Physics and Technology at the University of Bergen throughout autumn 2017 to spring 2018. During the project, I have worked with several individuals that have been vital contributors to this thesis achievement. First and foremost, I want to express my utmost gratitude to my two supervisors *Professor Kjetil Ullaland* and *Senior engineer Shiming Yang*. They have guided me through this project and provided me with invaluable knowledge that has been greatly appreciated. Furthermore, I would like to thank the SMILE staff at UB that have influenced this thesis directly or indirectly: *Kjellmar Oksavik, Georgi Genov, Torstein Frantzen, Thomas Poulianitis and Maja Rostad*. I will also express my gratitude to my fellow students' that have made my time at UB very enjoyable and in particular *Magnus Ersdal* for his constructive suggestions. Finally, I wish to thank my family and friends for their help and encouragement throughout my education.

Contents

ABSTRACT	III
ACKNOWLEDGEMENTS.....	V
ABBREVIATIONS.....	IX
1 INTRODUCTION.....	1
1.1 BACKGROUND AND MOTIVATION	1
1.2 ABOUT THIS THESIS.....	1
1.2.1 <i>Thesis Overview</i>	2
1.3 SOFT X-RAY IMAGER INSTRUMENT	3
1.3.1 <i>SXI Electronics Box</i>	4
2 RADIATION SHUTTER OVERVIEW	5
2.1 OBJECTIVE.....	5
2.2 SYSTEM REQUIREMENTS	6
2.3 IMPLEMENTATION.....	7
2.3.1 <i>Radiation Shutter Mechanism</i>	8
2.3.2 <i>Radiation Shutter Electronics</i>	11
2.4 E-BOX COMMUNICATION TOPOLOGY.....	15
2.4.1 <i>RMAP over RBDP Communication Protocol</i>	16
2.4.2 <i>RoR Read and Write Operations</i>	18
2.4.3 <i>RoR Timing Constraints</i>	20
3 STEPPER MOTOR AND DRIVER	21
3.1 STEPPER MOTOR.....	21
3.1.1 <i>Stepper Motor Drawbacks</i>	22
3.2 TYPES OF STEPPER MOTORS	22
3.2.1 <i>Variable-Reluctance Stepper Motor</i>	22
3.2.2 <i>Permanent Magnet Stepper Motor</i>	23
3.2.3 <i>Hybrid Stepper Motor</i>	24
3.3 STEPPER MOTORS IN SPACE APPLICATIONS.....	25
3.3.1 <i>Lubrication</i>	25
3.4 DRIVER HARDWARE	26
3.4.1 <i>H-Bridge</i>	27
3.4.2 <i>Current Control Circuit</i>	27
3.4.3 <i>Voltage Regulator</i>	31
4 RSE DEVELOPMENT	33
4.1 EMBEDDED SYSTEM	33
4.1.1 <i>Interrupts</i>	33
4.1.2 <i>Real-Time Systems</i>	35
4.1.3 <i>Task Scheduling</i>	36
4.2 SW DEVELOPMENT STRATEGY	37
4.3 RSE SOFTWARE DESIGN AND IMPLEMENTATION.....	38
4.3.1 <i>Initialisation Routines</i>	39
4.4 SCHEDULER	39

4.4.1	<i>Idle Task</i>	40
4.5	PERIODIC DISPATCHER.....	40
4.6	SPORADIC DISPATCHER.....	41
4.6.1	<i>Open/Close Shutter Stop at End</i>	42
4.6.2	<i>Open/Close Shutter Max no of Steps</i>	43
4.6.3	<i>Emergency Close Stop at End</i>	44
4.6.4	<i>Arm/Activate HDRM</i>	44
4.7	COMMUNICATION TASK.....	45
4.7.1	<i>Character Level</i>	46
4.7.2	<i>Packet Level</i>	47
4.7.3	<i>RMAP Level</i>	47
4.7.4	<i>Application Level</i>	48
4.7.5	<i>RMAP Response Level</i>	49
4.7.6	<i>Transmit Data</i>	50
4.8	RSE REGISTER OVERVIEW.....	50
4.8.1	<i>Status Registers</i>	51
4.8.2	<i>Control Registers</i>	54
4.9	STEPPER DRIVER SOFTWARE.....	55
4.9.1	<i>Stepper Motor Operation Mode</i>	56
4.9.2	<i>Driver Software Design</i>	57
4.10	HARDWARE MODULES.....	60
4.10.1	<i>USART 0</i>	60
4.10.2	<i>Timers</i>	61
5	TEST AND DEVELOPMENT	65
5.1	RSE MODIFICATIONS FOR TESTING.....	65
5.2	DPU SIMULATOR.....	66
5.3	DRIVER TEST.....	68
5.3.1	<i>Step-Sequence Test</i>	68
5.3.2	<i>Driver Parameters Test</i>	69
5.4	TASK EXECUTION TEST.....	73
5.5	MOTOR MAGNETIC FIELD TEST.....	76
5.6	TEST EVALUATION AND KEY FINDINGS.....	77
5.6.1	<i>Driver and Motor Evaluation</i>	77
5.6.2	<i>Tasks Handling Evaluation</i>	79
6	OUTLOOK AND CONCLUSION	81
6.1	FUTURE WORK.....	81
6.2	FPGA IMPLEMENTATION.....	81
6.3	CONCLUSION.....	82
APPENDIX A	SOFTWARE REQUIREMENT SPECIFICATION	83
APPENDIX B	TEST REPORT FOR RSE MOTOR CONTROL	89
APPENDIX C	SOFTWARE ORGANISING	101
APPENDIX D	RSE PIN MAP	103
7	REFERENCES	105

Abbreviations

SMILE	Solar Wind Magnetosphere - Link Explorer	P-MOS	P-type Metal Oxide-Semiconductor
ESA	European Space Agency	N-MOS	N-type Metal Oxide-Semiconductor
CAS	Chinese Academy of Science	RoR	RMAP over RBDP
SXI	Soft X-ray Imager	M-LVDS	Multipoint Low-Voltage - Differential Signalling
UB	University of Bergen	RBDP	Regular Byte Stream DAQ - Protocol
RSE	Radiation Shutter Electronics	HK-readouts	Housekeeping readouts
RSM	Radiation Shutter Mechanism	RMAP	Remote Memory Access - Protocol
SWCX	Solar Wind Charge Exchange	UART	Universal Asynchronous Receiver/Transmitter
SRS	Software Requirement Specification	MSB	Most Significant Bit
DPU	Data Process Unit	LSB	Least Significant Bit
CPU	Central Processing Unit	CRC	Cyclic Redundancy Checksum
CCD	Charge-Coupled Device	VR	Variable-Reluctance
E-Box	Electronics Box	PM	Permanent Magnet
PSU	Power Supply Unit	PWM	Pulse-Width Modulation
HDRM	Hold Down Release - Mechanism	ISR	Interrupt Service Routine
SoC	System on a Chip	WTD	Watchdog Timer
IO	Input-Output	RTOS	Real-Time Operating System
OCD	On-Chip Debugging	MISRA	Motor Industry Software Reliability Association
TID	Total-Ionising Dose	FIFO	First In First Out
SEE	Single-Event Effects	CTC	Clear on Compare Match
IC	Integrated Circuit	F-PWM	Fast Pulse Width Modulation
LET	Linear Energy Transfer	OCR	Output Compare Register
SEU	Single Event Upset	GUI	Graphical User Interface
SEL	Single Event Latch-up	WCET	Worst Case Execution Time
FPGA	Field Programmable Gate Array	TBD	To Be Decided
ASIC	Application-Specific Integrated Circuit	TBC	To Be Concluded

1 Introduction

1.1 Background and Motivation

The Sun continuously emits a stream of high-energy particles, known as solar wind. The Earth's magnetosphere acts as a protective shield that deflects these charged particles. Occasionally, the interaction causes severe disturbances on Earth, generally called space weather, which can affect performance and reliability of both ground and space technologies. As our world becomes more dependent on sophisticated technology in space, the understanding of space weather is not only important for scientific interest, but also for society. [1]

X-rays are emitted when the solar wind's highly charged ions, such as oxygen, collide with the natural hydrogen atoms in the outer magnetosphere [2]. This process is called Solar Wind Charge Exchange (SWCX) and can by observation, address fundamental aspects of space weather.

SMILE is the first space mission that studies these effects using 2D imaging technologies. This technology will provide soft X-ray images of the magnetic cusps, magnetosheath, and ultraviolet images of the aurora [1]. Furthermore, it will utilise in-situ solar wind plasma and magnetic field measurements. This mission will significantly increase our knowledge of the interaction between the solar wind and the magnetosphere. It will also provide further information regarding specific processes in the ionosphere (e.g., the aurora).

1.2 About this Thesis

This thesis primary objective is to start the development of the radiation shutter control unit, which is referred to as the radiation shutter electronics (RSE). The control system must meet the overall system requirements, described in Section 2.2. Consequently, a significant amount of effort was put into creating software requirements and design, before the code was implemented and tested. The RSE Software Requirement Specifications (SRS) is presented in Appendix A, and chapter 4 offers a detailed description of design and implementation. The tests and verification are further explained in Chapter 5.

A substantial amount of work was put into the communication, which ensured that the RSE followed the RoR communication protocol, used by the SXI instrument's control cards. This protocol provided the Data Process Unit (DPU) with a memory mapped interface where the RSE's registers could be accessed through memory read and memory write operations.

At the start of this project, the University of Bergen (UB) had already designed and created a stepper driver prototype which should be used to drive the stepper motor. However, the driver was not tested, and a driver software was not developed. Therefore, this thesis must also provide a well-constructed driver software and test the driver functionality. Due to the driver construction, the driver operation required control of several different signals as well as precise timing. Consequently, a great deal of effort was put into the driver software development where most of the driver is operated by hardware timers, which significantly decreased the use of the limited Central Processing Unit (CPU) resources.

Furthermore, a substantial amount of effort was invested in testing and verification. The control systems main modules were tested individually (e.g., task handling, communication, stepper driver and stepper motor). Finally, the overall functionality were tested. In addition, a great deal of effort was put into the development of custom software, written in Python. This program simulated the RSEs' master node (DPU), and thereby allowed to test and operate the RSE realistically. It also provides a user-friendly GUI where all the RSE functionalist can be initiated with ease.

The RSE software is commented and structured in logical separate files. In addition, the code is provided with version control, where each commit contains a short description of the changes and stored in UBs' git repository.

1.2.1 Thesis Overview

Chapter 2: Radiation Shutter Overview. This chapter starts by introducing the radiation shutter objectives, and the system requirements. Followed by an overview of the radiation shutter implementation and the communication protocol utilised by the SXI instrument's control modules. The communication protocol is a fundamental aspect of the system, and must therefore be understood before the RSE control system implementation makes sense.

Chapter 3: Stepper Motor. This chapter describes the operating principle of a stepper motor, why a stepper motor was selected, and special considerations for stepper motors used in space applications. This chapter also offers a detailed description of the stepper driver. An understanding of the driver hardware is required to comprehend the driver software implementation and trade-offs.

Chapter 4: RSE Development. This chapter offers a discussion about embedded systems, an overview of the RSE software and the development strategy. Furthermore, it provides a comprehensive description of the RSE software design and implementation.

Chapter 5: Test and Development. This chapter offers a brief description of the ground support equipment, developed for enhancement of testing, verification, and operation of the RSE. The chapter also provides a detailed description of the tests performed to verify the functionality of the control system, stepper driver, and the stepper driver.

Chapter 6: Outlook and Conclusion. This chapter gives some suggestions for future RSE development and an alternative solution for the RSE. Finally, the thesis conclusion is presented.

Appendix A, B, C, and D: These chapters provide supplementary information: software requirements specification, test rapport for RSE motor control, software organising, and RSE pin map.

1.3 Soft X-ray Imager Instrument

The SMILE SXI instrument features a wide-field lobster-eye telescope, which guides the photons through an array of squared tubes before appearing on the detector. The detector consists of two highly sensitive Charge-Coupled Devices (CCDs) measuring energy between 0.2 keV and 5 keV. The CCDs absorb the SWCX soft X-ray emission and convert the energy to electrical signals. A baffle system encircles the optics and the CCDs to prevent contamination from stray light. In addition, the radiation shutter created by the UB provides a controlled shielding mechanism that covers the CCDs when the CCDs are prone to radiation. Figure 1 illustrates the SMILE SXI instrument by means of a CAD model (left) and a schematic diagram (right). The information provided by the instrument can further be used to create a 2D image illustrating the response of the magnetosphere to the interplanetary magnetic field. [1]

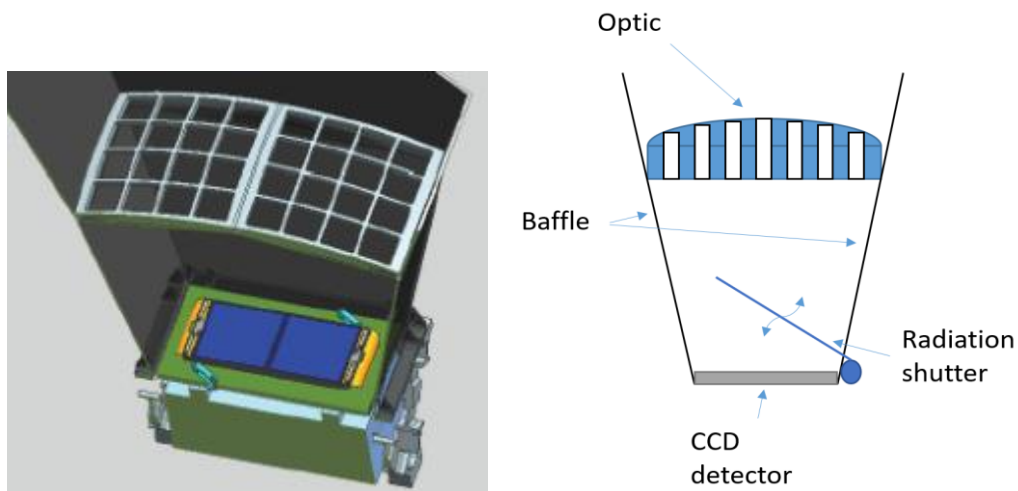


Figure 1. CAD model and schematic illustration of the SMILE SXI instrument [3].

1.3.1 SXI Electronics Box

The Electronics Box (E-box) contains all electronics cards required to operate the SXI instrument: the Power Supply Unit (PSU) providing power to all SXI components; the Data Process Unit (DPU), which is the E-box central control unit; and the RSE controlling the radiation shutter. The E-box is constructed as a cold redundant system (i.e., two identical cards for each unit), therefore, if one card fail, the other card will be enabled, and the system will operate as normal. This redundancy limits unpredictable breakdowns caused by failure parts and will significantly increase the larger system reliability. In addition, the E-box provides a shared communication bus used by the cards, as further explained in Section 2.4. Although the SMILE satellite has several other DPUs and PSUs and general components, this thesis only refers to the SXI instrument's related components. [3]

2 Radiation Shutter Overview

2.1 Objective

The highly sensitive CCD detectors are designed to detect soft X-rays with energy around 1 keV. The satellite traverses the Earth's radiation belts during parts of the orbit. This is a high radiation environment where particles such as protons can exceed energy of 10 MeV, for more information about radiation belts, see for example [4]. This energy level is several orders higher than the CCD's tolerance and will, therefore, degrade or destroy the CCDs. As a result, the radiation shutter is designed with the primary objective to protect the CCDs during the crossing of the radiation belts. Figure 2 shows a simplified illustration of the satellite's orbit where the red field around the Earth represents the radiation belts. The green vertical arrow illustrates the low radiate part of the orbit, and the red arrow indicates where the CCDs requires protection. However, the satellite may also perform space manoeuvres, e.g., in case other instruments on the satellite, like the magnetometer, requires calibration. This means that the CCDs may be exposed to stray light from the Sun or reflections from the Moon or the Earth. Hence, a secondary objective is to protect the CCDs during these events as well.

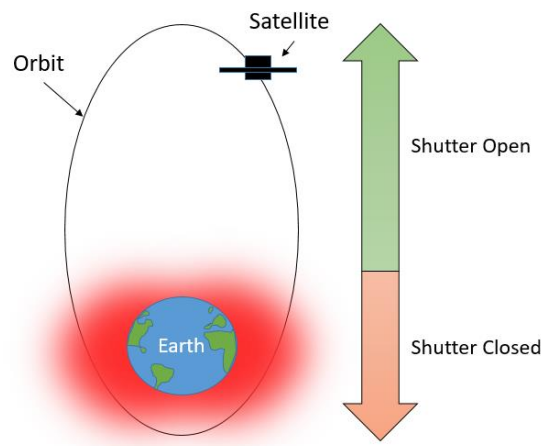


Figure 2. The SMILE satellite's orbit around the Earth. *The radiation belts are indicated in red, where the red arrow indicates the part of the orbit that requires CCD protection (i.e., the shutter must be closed). The green arrow represents the low radiate part of the orbit that the SXI instrument operates unaffected by the radiation (i.e., the shutter should normally be open).*

The third objective is that a radiation source can be mounted onto the shutter's inner surface. By closing the shutter, the radiation source will provide a known reference point for the SXI instrument, and thereby enable in-flight calibration possibility. Note, that this option has not yet been decided and is not within the scope of UB's work [3]. This alternative will therefore not be further discussed in this thesis.

2.2 System Requirements

The radiation shutter must provide a dynamic blockade which completely shields the CCDs from the radiation environment. This requirement imposes the need for a shutter which is mechanically operated, constructed of sufficient material and sited on top of the CCDs. On the other hand, the radiation shutter components, including the shutter, must be prevented from interfering with the detectors focal plane when the SXI instrument is active.

A baffle system is placed around the detectors to reduce exposure from stray light. However, this also limits the size of the radiation shutter solution, because the baffles define the outer barrier. As a result, the structure and components must have a small footprint, and the shutter must lie alongside the baffle wall when it is open. Therefore, the shutter must be able to move from a closed position, covering the CCDs, to an open position at an angle of 120° , which corresponds to the baffle angle. In addition, the radiation shutter must be designed for tolerating temperatures from -100°C to $+10^\circ\text{C}$ and should survive three years of unattended operation [3]. This imposes the need for a comprehensive and compact design which can withstand the mission's life cycle.

To reduce the possibility of system failure, the radiation shutter must be implemented as a redundant system. In case one system fails, the other system should be introduced and operate the shutter as normal. This means that two independent and identical radiation shutter systems must be implemented. However, some components cannot be redundant, for example, the structure and the shutter. This is because these components will obstruct the SXI instrument if they fail. Therefore, redundancy would be pointless.

The E-box, which contains the SXI instrument's PSU and DPU, must also hold the Radiation Shutter Electronics (RSE). Because the E-box provides a shared communication bus, where the DPU is the bus master, the RSE must be implemented as a bus slave. This configuration means that the slaves, including the RSE, must follow the E-box communication protocol that provides the DPU with a dependable control interface. To prevent bus contamination, the slaves must only act on commands from the DPU. Hence, the RSE does not determine when the CCDs need protection; this is the DPU's responsibility. However, the RSE must comprise safety features which automatically protects the CCDs in case of DPU or communication fails. This safety future is required because the RSE on its own, has no knowledge of the orbit or the radiation environment. Therefore, if communication is corrupt, the RSE must assume the worst-case scenario, meaning that the shutter must be closed and remain closed until communication is re-established.

The RSE must hold a reliable control system which enables the DPU to collect HK-readouts, configured RSE settings and perform operations, such as open the shutter. These demands require that the RSE contains a separate control unit, sensors, and actuators. In addition, the system must guarantee predictable operation of the shutter, even under extreme conditions that may cause the shutter to be cold weld onto the structure surface. These circumstances impose the need for an adequate shutter activator which provides enough torque.

Furthermore, the shutter must be restrained from moving during launch and until the satellite is successfully deployed in orbit. In addition, the RSE must process sensor readings and ensure that components are not exposed to excessive stress. For example, if the shutter activator temperature is critically high, the ongoing shutter operation must be suspended. However, protecting the CCDs must always be a higher priority than the radiation shutter wellbeing. Therefore, the RSE must provide emergency features that enable shutter operations even if it may degrade the radiation shutter components.

2.3 Implementation

The radiation shutter is based on a simple principle, where the shutter is mechanically connected to a stepper motor. This principle enables the shutter to be accurately and safely controlled. Furthermore, the radiation shutter is split into two separate redundant systems as shown in Figure 3. The RSM contains the mechanical components, and the RSE contains the electrical components required to operate the radiation shutter.

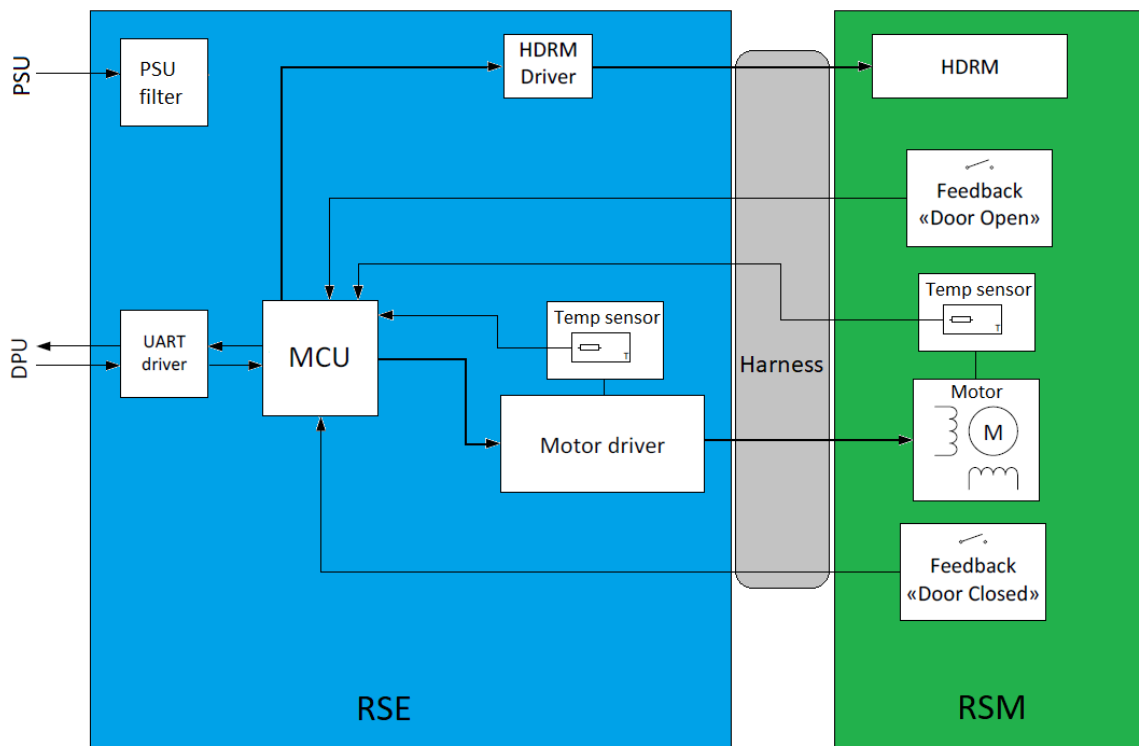


Figure 3. Schematic diagram of the radiation shutter [5]. *The radiation shutter electronics are located within the E-box, and the radiation shutter mechanics is sited on top of the CCDs.*

2.3.1 Radiation Shutter Mechanism

The RSM will be mounted on top of the CCDs inside the SXI assembly tube. Figure 4 illustrates the RSM assembly where the different elements are labelled from 1 to 9. Most of the structure consists of aluminium, but some parts are made of lead loaded bronze, stainless steel and other materials. The RSM components operated by the control system are further explained in the sections that follow. [6]

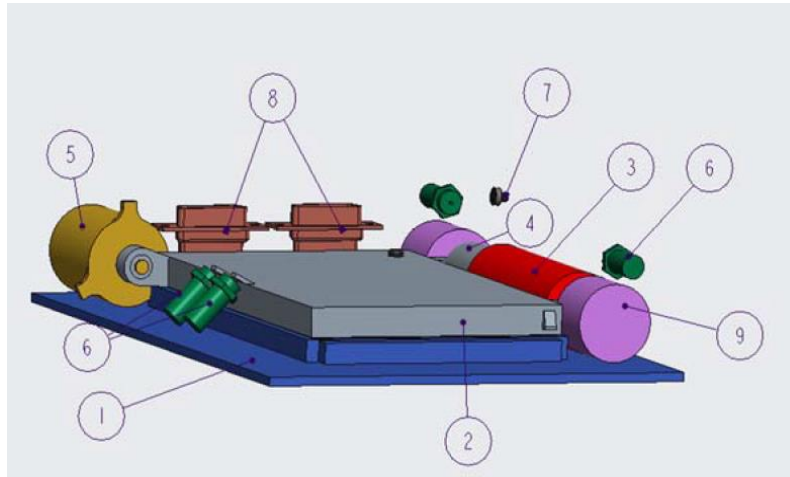


Figure 4. Components of the Radiation Shutter Mechanism (RSM) [6]. (1) Frame holds the RSM components. (2) Shutter which is constructed of 10 mm thick solid aluminium. (3) Stepper motor operates the shutter. (4) Coupling compensates for mounting misalignments. (5) Hold down release mechanism (HDRM) holds the shutter in the closed position during launch. (6) End switches sense the open and closed positions. (7) End stop act as a damper for the shutter, (8) Harness connectors between RSM and Radiation Shutter Electronics (RSE). (9) Bearings for the stepper motor.

Stepper Motor and Gearbox

The shutter is operated by a space classified, two-phase stepper motor, equipped with an internal gearbox, as shown in Figure 5. The motor is constructed of high-quality materials that limit outgassing and enable operation during extreme temperatures and hard vacuum conditions. [6]



Figure 5. Phytron phySPACE stepper motor. Mounted on the frame used during development. Motor height, approximately 8 cm.

Two thermocouples integrated into the motor windings provides temperature surveillance from -270°C to 1370°C [7]. In contrast to the radiation shutter redundant system, the RSM does not contain two separate motor elements. However, the motor's most critical components are doubled, i.e., a double set of windings and a double set of temperature sensors [5]. The gearbox contains a three-stage, planetary gear system. The planetary gears provide excellent gear reduction without compromising the weight, size, and robustness [8]. Table 1 presents an overview of the stepper motor and gearbox specifications.

Table 1. Stepper motor and gearbox specifications. [7]

Parameters	Value/identification
Motor type	Phytron phySPACE 25-2-200-0.6
Number of coil phases	2
Steps per rotation	200
Nominal motor current	0.6 A
Windings resistance	3.25 Ω
Windings inductance	1.1 mH
Max operation voltage	70 V
Thermocouple	Type K
Holding torque at nominal motor current (without gearbox) ¹	12 mNm
Detent torque (without gearbox) ²	2 mNm
Combined mass of motor and gear	0.18 kg
Bearings	Ball bearings with dry lubrication
Gear type	VGPL 22
Gear ratio	1:196
Gear rated torque	1.5 Nm
Gear efficiency at full load	80%

Hold Down Release Mechanism

The spacecraft will be exposed to shock and vibrations during launch and while travelling through the atmosphere. The holding force generated by the stepper motor is insufficient to oppose these conditions. Because of this, an additional mechanism called the Hold Down Release Mechanism (HDRM) is implemented to restrain the shutter from moving. The HDRM is an actuator controlling a pin which is inserted in the shutter. The pin is released when the spacecraft is successfully deployed, thereby enabling the shutter to be moved freely by the stepper motor. Figure 6 illustrates the HDRM when the shutter is in the locked position. [6]

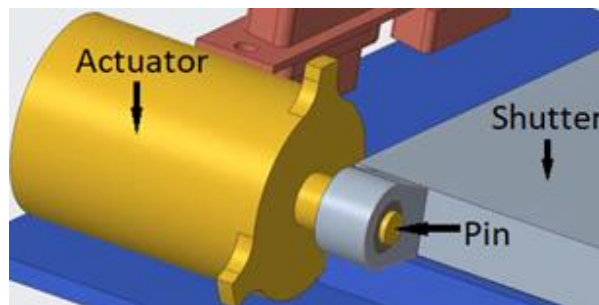


Figure 6. HDRM launch lock [6]. *The actuator operates a pin which is inserted in the shutter. This restrains the shutter from moving during launch.*

¹ Holding torque is the amount of torque required to move the rotor while the coils are energized.

² Detent torque is the amount of torque required to move the rotor without energized coils.

End Switches

The end switches are used to sense the shutter end positions (open or closed) and are activated upon contact. A double set of switches is placed at each end position, providing redundancy in case of switch failure. [6]

2.3.2 Radiation Shutter Electronics

The RSE is constructed as a System on a Chip (SoC), containing the microcontroller and the driver hardware necessary to operate the stepper motor. Two identical RSE cards will be mounted inside the E-box, connected to their corresponding redundant PSU and DPU. This provides two independent systems, enabling one to be used for backup. [5]

Prototyping

As a consequence of the project's early stage, requirements regarding the RSE control unit were limited. However, it was suggested that an AVR microcontroller should be used. Therefore, the RSE prototype was established using ATMEL STK600, which is a highly customisable development platform.

STK600 supports programming of all 8-bit and 32-bit AVR microcontrollers, where it is connected on top of the STK600 with a separate socket card. All Input-Output (IO) ports are routed to different headers, which are organised in logical groups and identified by their port names. This system significantly increases the speed of development because the microcontroller can easily be changed without requiring other hardware modifications, only simple software adjustments. In addition, the card includes hardware features such as pushbuttons and LEDs that can be connected to the preferred IO ports. The only drawback was that the platform did not support On-Chip Debugging (OCD). OCD provides the possibility of single stepping through the code implemented on the chip, thereby enabling a more natural method for following the program flow and detecting bugs. Because embedded systems frequently use interrupts, the program flow is hard to predict without OCD abilities. However, this problem was solved by introducing a separate debugger, AVR Dragon. For more information about the STK600, see [9].

AVR Dragon, a development tool compatible with both 8-bit and 32-bit AVR microcontrollers, was selected in this project because of its ability to enable OCD. The AVR Dragon allows for 3 hardware breakpoints and 32 software breakpoints, enabling efficient bug tracking. For more material regarding the AVR Dragon debugger, see [10].

The development software Atmel Studio 7 was a natural choice because it is designed explicitly for programming AVR microcontrollers. This program combined with the debugger produced a powerful tool that enabled fast programming and OCD. In addition, it enabled the ability to view the processor status, registers, communication, and analogue interfaces directly from a sub window, all of which were essential for confirming register settings and task execution. For additional information about Atmel Studio 7, see [11]. Figure 7 presents the RSE development setup, where the key elements are identified.

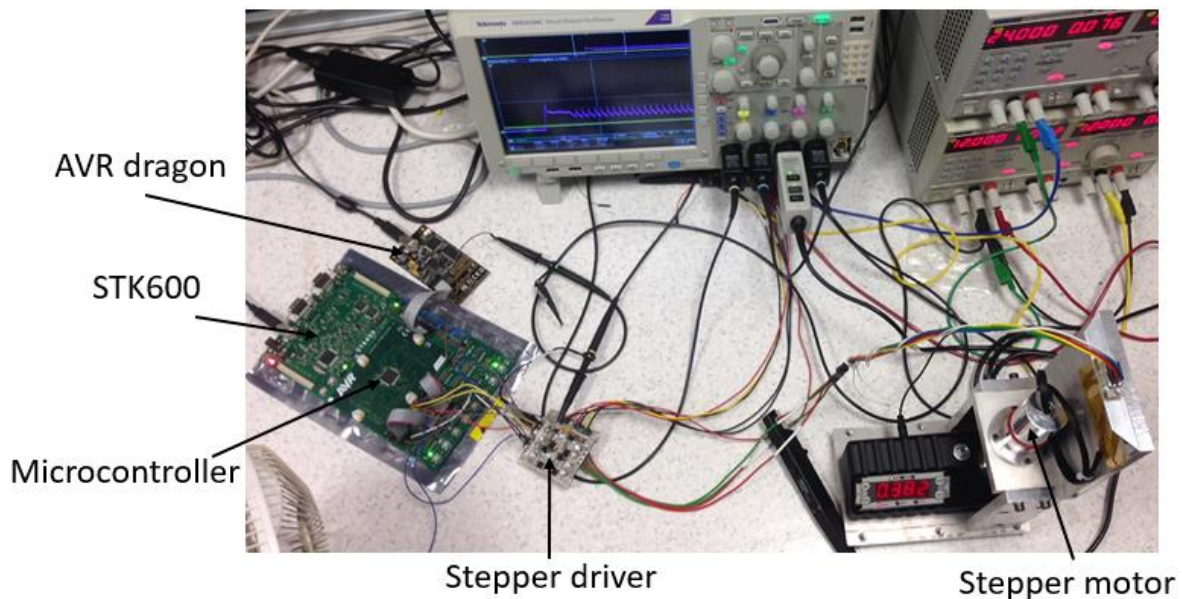


Figure 7. RSE prototyping setup at the University of Bergen. *Two power supply units provide power to the stepper driver and motor. The oscilloscope measure motor coil current, and the torque meter sited below the stepper motor measure motor torque.*

Radiation Tolerance

Space applications require that the electronics are evaluated and qualified in terms of radiation tolerance. Because the protective shield provided by the atmosphere is absent at the mission's altitude, the electronics are exposed to radiative surroundings. High-energy particles such as ions, protons, and electrons can affect electronic performance or worse, destroy vital components. In general, radiation effects concerning electronics are split into two classes: Total-Ionising Dose (TID) and Single-Event Effects (SEE).

TID is a gradual effect where charged particles or photons ionise the medium as it passes through. The energy is accumulated over time, slowly degrading the silicon dioxide (SiO_2) which is the main material used in Integrated Circuits (IC). This effect results in reduced performance and unpredictable characteristics. In contrast to SEE, TID can be reduced by using sufficient shielding. The ability of electronics to tolerate TID is measured in krads, a unit describing absorbed dose per seconds.

SEE occurs when a single, high-energy particle interacts with the IC. SEE tolerance is usually rated by the maximum tolerable Linear Energy Transfer³ (LET). Single-Event Upset (SEU) and Single-Event Latch-up (SEL) are two common forms of SEEs. SEU occurs when charge deposited by a particle exceeds the transistor's threshold voltage and thereby changes the transistor state. In general, SEU is a significant problem for ICs registers where a bit or more can be flipped. However, the circuit's ability to withstand SEUs can be significantly increased by for example creating a redundant memory system.

SEL occurs when a high energy particle activates a parasitic PnP transistor between the N-type Metal Oxide-Semiconductor (N-MOS) transistor, the bulk, and the P-type Metal Oxide-Semiconductor (P-MOS) transistor. This parasitic effect creates a low impedance path between the power rail and ground. The low impedance path permits high current to flow through the junction, potentially resulting in internal junction breakdown. For additional information about total-ionising dose and single-event effects, see for example [12].

Microcontroller

The radiation shutter control unit needs to be highly reliable, low powered, with a small footprint and an ability to withstand the radiative environment. The control unit's tasks include operating the motor, monitoring the sensors, and communicate with the DPU. Because these tasks can be accomplished without parallel processing, a microcontroller was selected. Another factor when selecting the control unit is cost. In general, microcontrollers are inexpensive compared to other solutions such as Field Programmable Gate Arrays (FPGAs) or an Application-Specific Integrated Circuits (ASICs).

The current market for space-qualified microcontrollers is limited. However, Microchip⁴ has released the ATmegaS128, a space market version of the standard ATmega128 microcontroller. ATmegaS128 is a low power AVR 8-bit microcontroller designed to prevent SEL with a LET below 62.5 MeV.cm²/mg, and a TID tolerance of 30 krad. Table 2 offers a brief overview of the ATmegaS128 specifications. For more information about the ATmegaS128 see [13].

Except for the radiation tolerance, both these microcontrollers (ATmegaS128 and ATmega128) are equivalent and thereby enables compatible software implementation. Consequently, it was decided to use the non-space graded, inexpensive ATmega128 in the development phase and uploaded the software onto the ATmegaS128 when the software is tested and verified.

³ LET is a measurement describing the energy deposit by a particle per unit length, with the SI unit MeVcm²/mg.

⁴ Microchip has acquired Atmel in 2016.

Table 2. ATmegaS128 specification summary [13]

Microcontroller	ATmegaS128
CPU	8-bit AVR RISC architecture
EEPROM	4 kBytes
Flash	128 kBytes
SRAM	4 kBytes
PWM channels	6 units
Timers 8-bit	2 units
Timers 16-bit	2 units
Speed grade	0-8 MHz
USART modules	2 units
ADC	1 unit
Voltage range	3V to 3.6V
SEL tolerance	62.5 MeV.cm ² /mg
TID tolerance	30 krad
ESD tolerance	2000V HBM / 750V CDM
Temperature range	-55°C to +125°C

Stepper Driver

The stepper driver is an electrical circuit that acts as a buffer between the microcontroller and the stepper motor. It provides a simplified interface where the microcontroller can control the relatively complex and high current demanding motor operation. The present market for space-qualified stepper drivers is narrow, so the few available drivers are costly. However, a stepper driver is a relatively simple circuit with few components. Therefore, the stepper driver will be designed and built by UB where all components can be selected with an emphasis on radiation tolerance. Besides the reduced cost, another advantage of this plan is that the transistors can be placed as separate components. By isolating each transistor, the circuit is essentially SEL proof because a latch-up requires two transistors (one P-MOS and one N-MOS) situated in the same bulk. [12]

The driver design is still in an early phase; however, the UB has developed a prototype with the following features:

- In-flight adjustable current limit.
- In-flight adjustable power saving features.
- SEL proof transistors, and
- Built-in temperature sensors.

2.4 E-box Communication Topology

A general-purpose master/slave communication protocol called RMAP over RBDP (RoR) provides the DPU with a simple memory-mapped interface. The E-box cards are connected via a Multipoint Low-Voltage Differential Signalling (M-LVDS)⁵ bus, as shown in Figure 8. The DPU is the bus master, and the additional cards are slaves; this means that only the DPU can initiate bus transactions. The following subsections briefly describe the higher level of the RoR communication protocol. For additional information regarding the RoR communication protocol see [14] and [15].

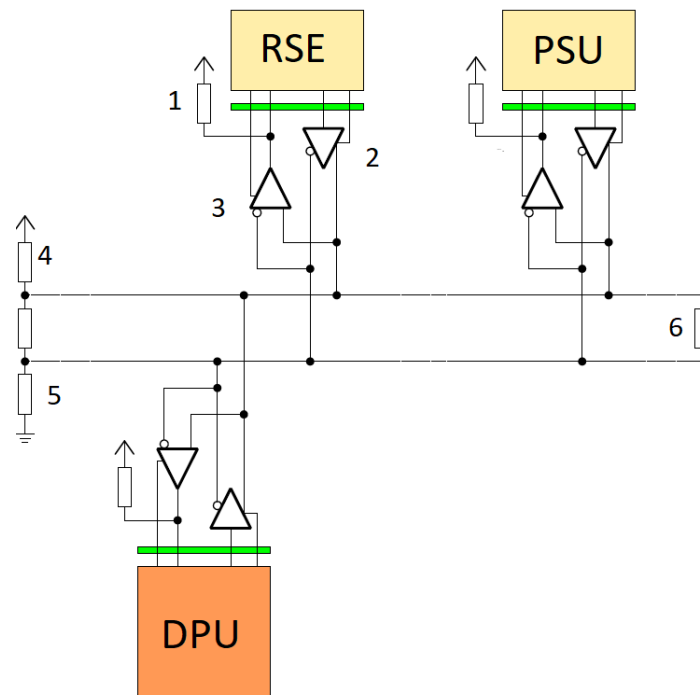


Figure 8. M-LVDS bus interface [14]. (1) Pull-up resistor ensures that the line is high if it is not driven. (2) Transmitter. (3) Receiver. (4, 5) Fail-safe biasing prevents undesirable oscillations. (6) Termination resistor, prevents signal reflections.

⁵ The bus follows the TIA/EIA-889 Interface standard.

2.4.1 RMAP over RBDP Communication Protocol

RoR exploits the fundamental aspects defined by the Regular Byte Stream DAQ Protocol (RBDP), a protocol designed for collecting HK-readouts. However, RBDP does not establish the interface required to fulfil the slaves' responsibilities. Consequently, an additional protocol called the Remote Memory Access Protocol (RMAP) is implemented on top of the RBDP, thus the name RoR. The merged protocol makes the slaves' registers accessible both via memory read and memory write operations. Figure 9 illustrates the RoR protocol stack, where the M-LVDS bus topology defines signal level. The other levels are described in the following sections.



Figure 9. RoR protocol stack. *The RoR communication protocol which the E-box modules must follow, including the RSE, contains five hierarchical levels.*

Character Level

Character level defines the smallest unit of transmitted data, identified as a character. The Universal Asynchronous Receiver/Transmitter (UART) arranges the characters, configured in UART 9O1⁶ mode as illustrated in Figure 10. In addition, character level provides bit error verifications, such as parity check. In case of a bit error, the character is discarded on the character level. The UART transmission speed has not yet been decided, but a baud rate of 200 kbps has been suggested.

⁶ UART 9O1 characterises the serial data properties; 9 data bits, odd parity, and one stop bit.

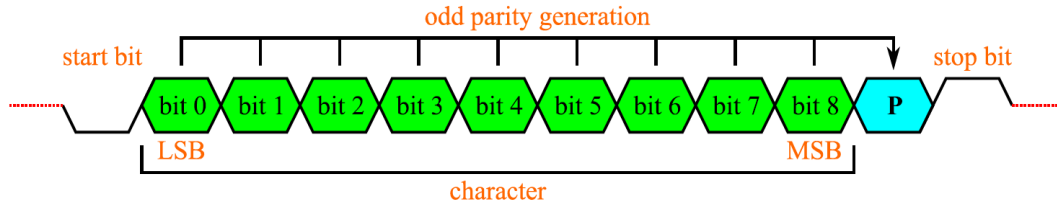


Figure 10. RoR UART configuration [14]. A character contains 9-bit (green) and is transmitted along with a start bit, odd parity bit (blue) and a stop bit. Start bit and stop bit signal the start and the end of a transmission, respectively. The parity bit provides the receiving node with odd parity detection possibilities.

Packet Level

Packet level sorts a concrete set of characters into two individual packets: query packets and response packets. These packets provide a higher level of information, where the DPU generates query packets and the slaves generate response packets. The query packets hold a single character, identified by the Most Significant Bit (MSB) set logically low. A total of five different query packets are defined, as shown in Table 3.

Table 3. Query packets structure

Query name	Header		Data	
	Bit [8] (MSB)	Bit [7-5] (Query id)	Bit [4]	Bit [3-0]
Invocation	0	"000"	Slave address bit [4]	Slave address bit [3-0]
Instruction	0	"001"	R/nW	Register address bit [8-5]
Reg-address	0	"010"	Register address bit [4]	Register address bit [3-0]
Write data H	0	"011"	Reserved	Register data to write bit [7-4]
Write data L	0	"100"	Reserved	Register data to write bit [3-0]

During normal operations, every query sent by the master is answered by a corresponding response packet, generated by the target slave. The only exception is when the target slave receives a bit error. RoR defines five unique response packets, each of which contains three characters, and are distinguished from the query packets by the MSB set logically high, as shown in Table 4. The first character named "confirmation" identifies the responded query type and indicates whether the slave's application-level authorised the query or not. The second character named "remark" contains acquired register content, no data, or a remark identifying the error. The third character named "CRC" contains a Cyclic Redundancy Checksum⁷ (CRC) generated by the two first characters. The CRC character ensures that the master detects errors introduced during transmission.

⁷ RMAP CRC polynomial: $g(x) = x^8 + x^2 + x + 1$.

Table 4. Response packets structure

Response name	Character	Header		Data		
		Bit [8] (MSB)	Bit [7-5]	Bit [4-2]	Bit [1]	Bit [0]
Invocation	Confirmation	1	ID = "000"	Reserved	Auto timeout	Query reject
	Remark	1	E.g., Cause of rejection			
	CRC	1	CRC			
Instruction	Confirmation	1	ID = "001"	Reserved	Auto timeout	Query reject
	Remark	1	E.g., Cause of rejection			
	CRC	1	CRC			
Reg-address	Confirmation	1	ID= "010"	Reserved	Auto timeout	Query reject
	Remark	1	E.g., Cause of rejection or register address			
	CRC	1	CRC			
Write data H	Confirmation	1	ID = "011"	Reserved	Auto timeout	Query reject
	Remark	1	E.g., Cause of rejection			
	CRC	1	CRC			
Write data L	Confirmation	1	ID = "100"	Reserved	Auto timeout	Query reject
	Remark	1	E.g., Cause of rejection			
	CRC	1	CRC			

Application Level

Application level describes the behaviour of the software and is individual for each slave.

2.4.2 RoR Read and Write Operations

As discussed in Section 2.4, a RoR bus transactions involve two requests: memory write, or memory read. Both operations follow a non-posted procedure, meaning that the source must wait for an acknowledgement or a reply to be received, which in this case is a response packet produced by the slaves. This procedure is obviously slower than a posted transaction, where the source can send a continuous stream of requests. However, the E-box slaves manage vital systems for instants, the radiation shutter. Therefore, it is beneficial to detect failures as soon as possible. In addition, the slaves do not require a more efficient transaction topology as there are few slaves and limited bus transactions.

All RoR transactions start with an invocation query that establishes communication with the acquired slave, as shown in Figure 11. A memory read operation is identical to a write transaction, except that the performed cycles will end at the reg-address query (i.e., write data H and write data L queries are not issued). Note that once a RoR transaction completes, the communication between the DPU and the target slave terminates. Hence, an invocation query must be sent to re-establish the connection. [5]

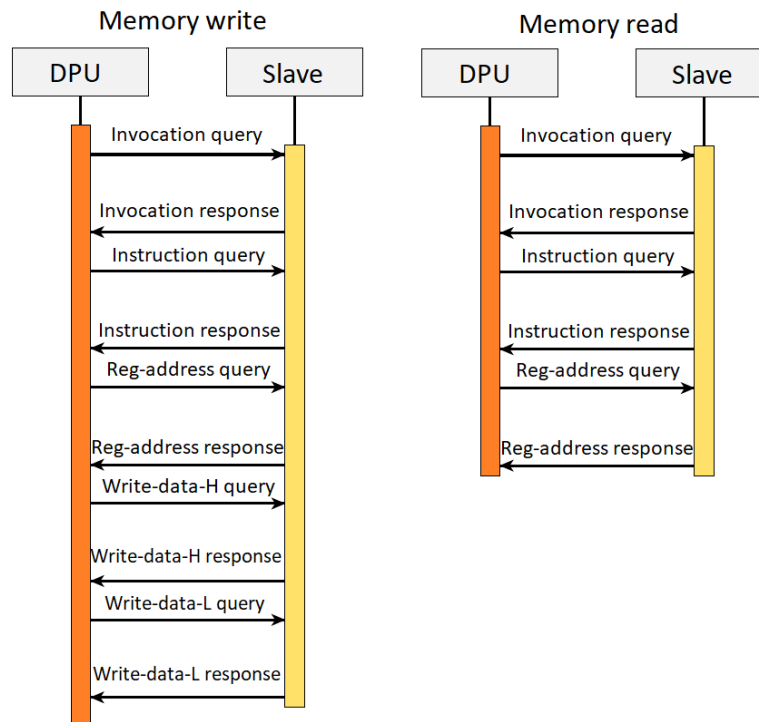


Figure 11. RoR bus transaction with memory write (left) and memory read (right) operations [14]. Assuming normal operation, every query transmitted by the DPU is answered by a response packet, generated by the slaves.

Reject Query

A query that is intentionally rejected by the slave will result in a response packet where the “confirmation character” signifies the rejected query, and the “remark character” identifies the cause of the rejection. Note that the slave will be re-initialized and an “invocation query” must be sent to re-establish communication.

Sequence Error and Query Authorisation

A particular case of an error occurs if the query is authorised, but the sequence is wrong. This sequence error occurs if the slave expects a “non-invocation query” but receives an “invocation query”. This will result in a response packet where the “confirmation character” indicates that the query is authorised, but the “remark character” contains the “sequence error id”. The previously received data will be discarded, and the next expected query is an “instruction query”. In other words, an “invocation query” will always be authorised, regardless of the sequence.

2.4.3 RoR Timing Constraints

The RoR communication protocol defines three timing constraints, which are essential for preventing buss contamination. Because the protocol does not specify a transmission speed, the timing is represented by baud delays, where 1 baud delay = $\frac{1}{\text{Baud rate [bps]}}$.

- **Slave slack:** After a received query, the slaves must wait for at least 1 baud delay before a response message can be initiated.
- **Master slack:** After a complete bus transaction, the master must wait for at least 1 baud delay before a new bus transaction can be initiated.
- **Response timeout delay:** After a received query, the slaves must start to drive the bus within 24 baud delays. If a response is not generated within 24 baud delays, the master can start a new transaction.

For the RSE, these requirements mean that the control system must be designed such that a response packet is generated between 1 baud delay and 24 baud delays, after a received query.

3 Stepper Motor and Driver

This chapter begins by introducing some general aspects regarding stepper motors and, discussing the challenges regarding stepper motor in space applications. These sections are based on materials from [16], [17], [18], [19], and [20]. Finally, a detailed description of the RSE driver hardware is presented.

3.1 Stepper Motor

A stepper motor is an electromechanical device that generates discrete mechanical movement when digital pulses are applied. By stimulating the motor in a specific sequence, the rotor rotates in discrete steps where the angular speed is proportional to the input signal frequency. The sequence order determines the rotating direction. If the load does not exceed the motor torque capability, the position is identified by tracking the steps/pulses. Because of this, the stepper motor archives precise positioning without requiring a closed loop regulating system. This feature eliminates the need for expensive sensing devices and feedback and often results in simpler and more robust system. Besides the ability to be controlled accurately without a closed loop system, several other advantages make a stepper motor well-suited for this project.

- **Reliability:**

Stepper motors do not use contact brushes to transfer energy to the rotor such that it rotates. Hence, it is only mechanically connected via the output shaft and bearings. This advantage makes the motor's reliability only dependent on the lifetime of the bearings.

- **Responsiveness:**

The motor has full torque at a standstill, which gives an excellent response to starting/stopping/reversing. The quick response is beneficial if the load (e.g., the shutter) is stuck. By increasing the current supplied to maximum, the torque generated will work as a hammer, forcing the load to move.

- **Holding force:**

Unlike a conventional DC motor, a stepper motor generates equal force regardless of whether the motor is moving or at a standstill, assuming energised coils. In addition, stepper motors generate some force even if the coils are not energized. These static features eliminate the need for separate components to hold the load at a fixed position.

- **Controllability:**

Because the angular speed is proportional to the pulse frequency, the rate can easily be controlled digitally. This feature of the motor enables full control at low and high speeds.

3.1.1 Stepper Motor Drawbacks

Due to the discrete behaviour, the stepper motor is prone to resonance. The inertia of the rotating load and rotor will overshoot the rotor a small amount each step before settling to a stable position. Figure 12 illustrates the single step response for a stepper motor where the oscillations are present. If the motor is poorly controlled the oscillations can become severe and potentially result in sudden torque loss and skipped steps. Due to the open loop system, the controller does not recognise step loss. Because of this problem, the radiation shutter has two end-switches placed at the shutter open and closed position.

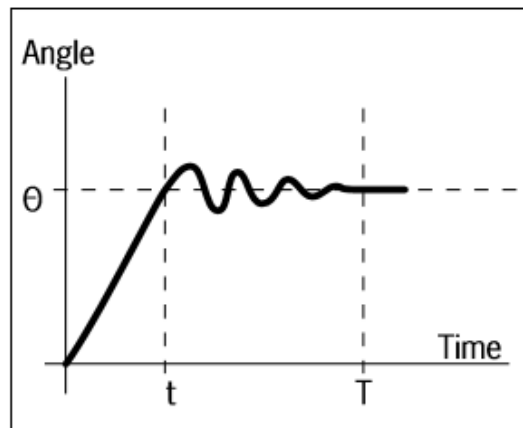


Figure 12. Stepper motor “step response” Step angle vs time [19]. *The rotor exceeds the desired angle “ θ ” at time “ t ” and oscillates until time “ T .”*

Because the stepper motor’s torque characteristics significantly decrease at high speeds, the stepper motors are best suited for applications that operate at speeds below 1000 rpm.

3.2 Types of Stepper Motors

Stepper motors can be divided into three categories, Variable-Reluctance (VR), Permanent Magnet (PM) and hybrid stepper motor. This section discusses each type, including advantages and limitations, and then concludes by selecting the one that best suits this project.

3.2.1 Variable-Reluctance Stepper Motor

Figure 13 illustrates a cross-section of a VR-stepper motor. The rotor consists of a soft magnetic-toothed iron core, where the teeth positions are offset from the stators. The offset between the stators and the rotor defines the stepping angle. The stators create diagonal pole pairs as illustrated in Figure 13 where the pole pairs are denoted by letter A–D and A’–D’. When energising the stator windings, the rotor seeks the path of least magnetic reluctance, which is equivalent to the least resistive path in terms of electrical circuits. This will align the energised pole pair with the adjacent rotor’s teeth. By stimulating the stators in a specific sequence, the rotor rotates.

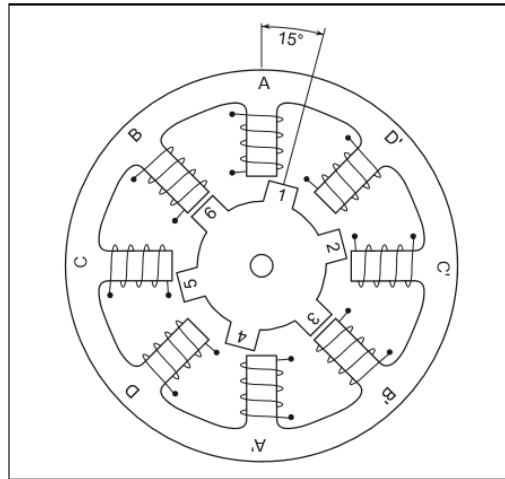


Figure 13. Cross-section of a Variable-reluctance stepper motor [19]. *The toothed rotor is sited in the middle, and the stators are placed around the rotor with an offset of 15° angle.*

Because the VR stepper motor has the most straightforward and smallest design among these three types, it is also the cheapest and most robust. All stepper motors will have a reduction in torque regarding speed, but the other types, especially the PM-stepper motor, have steeper torque drop-off curves than the VR-stepper motor. This makes the VR-stepper motor preferred in applications that require a wide range of speed (e.g., washing machine).

However, the VR stepper generates noise due to the lack of controllability. In addition, the VR-stepper motor does not have a permanent magnet rotor which increases the magnetic field. Consequently, the motor generates less torque and does not generate detent torque which means it cannot hold a load in a fixed position when the coils are not energised.

3.2.2 Permanent Magnet Stepper Motor

A cross-section of a simplified PM stepper motor is illustrated in Figure 14. In contrast to the VR stepper motor, the PM motor has a magnetised rotor without teeth. The magnetic poles are situated in straight lines parallel to the shaft. When the coils are energised, they create a magnetic flux pattern that interacts with the PM rotor. The rotor will seek the path that aligns with the magnetic field generated by the windings. If the windings are energised in a specific sequence, the rotor will follow the discrete rotating field.

The primary advantage of the PM-motor is the presence of the permanently magnetised rotors, which generate detent torque and increase magnetic flux, resulting in improved torque characteristics. In addition, it is also possible to increase the resolution by dividing the steps through software techniques.

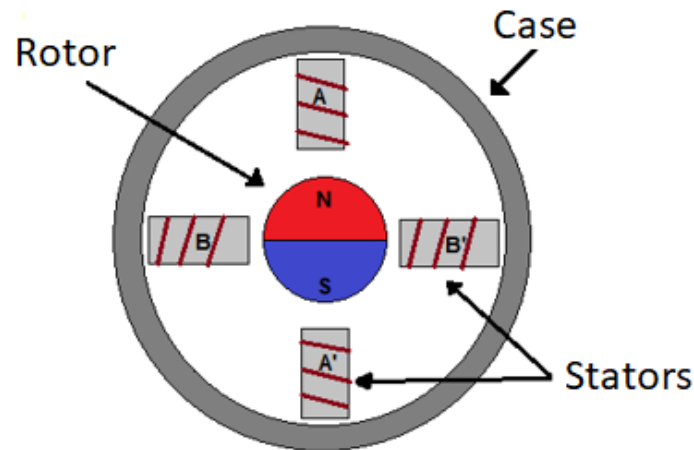


Figure 14. Cross-section of a permanent magnet stepper motor.

3.2.3 Hybrid Stepper Motor

The hybrid stepper motor is a combination of the VR and PM stepper motor. By using toothed stators and a toothed PM rotor, the hybrid motor gains the advantages of both motors. In Figure 15, a cross-section of a simplified hybrid stepper motor is illustrated. Unlike the PM motor, the rotor of a hybrid motor is axially magnetised, one end polarized south and the other end polarised north. The teeth of the stators and the rotor align in different configurations, which defines the step angle. The principle of rotation remains the same: by sequentially energizing the coils, a rotating magnetic field forces the rotor to rotate.

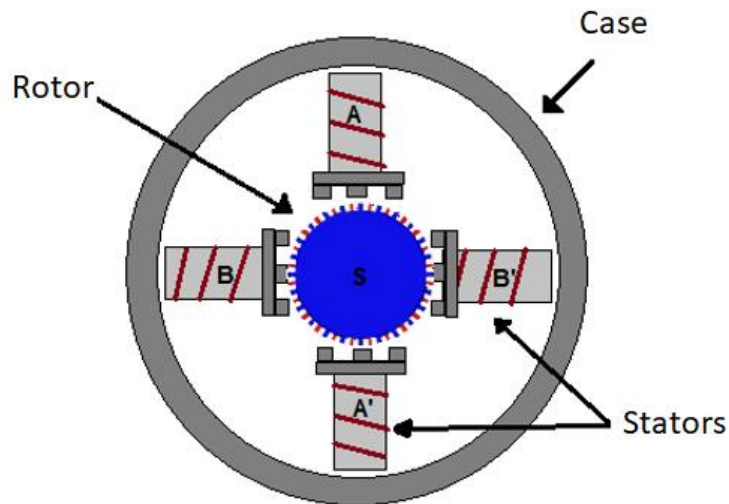


Figure 15. Cross-section of a simplified hybrid stepper motor.

By combining the advantages of the PM and VR motor, the hybrid stepper motor gains excellent torque characteristics and excellent stepping resolution due to software configurable stepping interval.

The SXI instrument relies on the stepper motors ability to always guarantee predictable operations of the radiation shutter. If the shutter gets stuck in either the open or closed position, the SXI sensor may be damaged due to radiation, or the sensor's field of view may be blocked. Consequently, the motor torque needs to be high enough to move the radiation shutter, even under extreme conditions. The weight of the motor is also an aspect while selecting a motor. The hybrid stepper motor has the highest performance-to-weight-ratio and is for that reason chosen in our project.

3.3 Stepper Motors in Space Applications

Even when the general requirements for a motor, such as torque, response, speed, and power consumption are met, some additional requirements must be considered when designing space applications. The motor needs to be highly reliable, operate unattended for several years, and able to withstand hard vacuum conditions and a wide temperature range. It also needs to be unaffected by radiation during operation and able to withstand vibration and shock during launch. The materials used should not be prone to outgassing. Furthermore, the motor should be highly efficient and have a compact, light design. These requirements limit the use of, bearings, lubrication, and material.

3.3.1 Lubrication

Due to the high vacuum, zero gravity, and radiative conditions, lubrication is one of the leading problems with moving parts in space applications. If the lubrication is not correctly done, the friction will be too high, extended wear and unpredictable performance will occur, and the mechanical surfaces could be welded together. These problems would lead to catastrophic failures that could potentially make the mission fail.

In ground-applications, regular lubrication, such as oil and grease, is used. Even the humidity and the organic contents of the atmosphere will lubricate the surfaces. In addition, the ambient pressure on Earth is relatively high so the fluid will not premature evaporate.

For the bearings to function correctly, lubrication needs to be present. Otherwise their lifetime will be significantly reduced. The lifetime of the bearing is also dependent on the operation speed, which limits the use of lubrication in very high-speed operations. However, the stepper motor used in this project operates at low speed and is therefore lubricated with dry lubrication.

3.4 Driver Hardware

A stepper motor turns by sequentially shifting current through the motor coils. This current routing principle is achieved by means of a stepper driver. Figure 16 illustrates the RSE driver circuit for one coil. It involves three main modules, H-bridge, current control circuitry, and the voltage regulator. Notably, the driver circuit is identical for both coils but controlled separately.

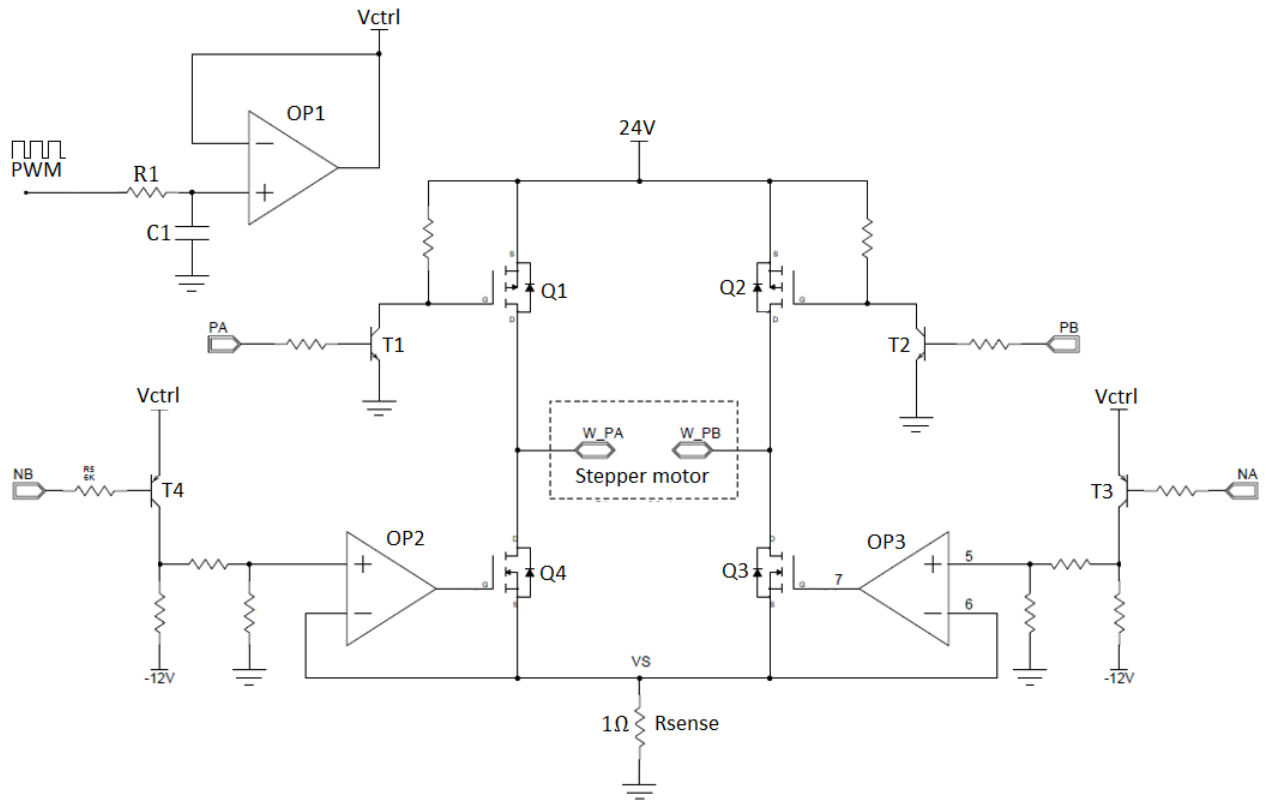


Figure 16. RSE Driver circuit for one coil [21]. The voltage regulator consists of a low pass filter ($R1$ and $C1$) and an operation amplifier $OP1$. The H-bridge contains four transistors, $Q1$, $Q2$, $Q3$, and $Q4$. The current control circuitry $OP2$, $OP3$, and their corresponding voltage dividers.

3.4.1 H-Bridge

In general, an H-bridge is a current routing network involving four transistors and the load: two pull-up transistors linked to the power rail, two pull-down transistors connected to ground, and the load situated in the middle. Current is routed through the load, and the stimulated transistors determine its direction. Figure 17 shows the H-bridge operation principle, where the green arrows represent the current flow. The H-bridge labelled “1” in Figure 17, enables current to flow through pull-up transistor Q1, the motor windings, and pull-down transistor Q3. The H-bridge labelled “2” reverses the motor current by activating pull-up transistor Q2 and pull-down transistor Q4. Note, that only one pull-up transistor and the pull-down transistor opposite to the load can be activated simultaneously. Otherwise, the H-bridge creates a short circuit that may destroy the transistors.

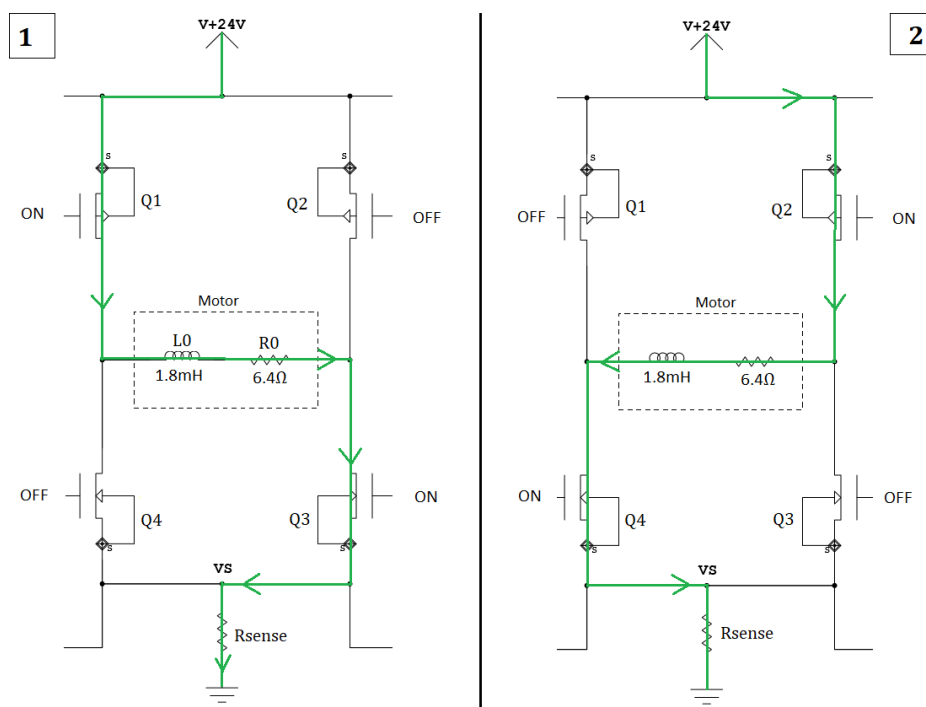


Figure 17. H-bridge operation principle [21]. The activated transistors lead current through the motor windings, as indicated by the arrows.

3.4.2 Current Control Circuit

From a fundamental perspective, a stepper motor consists of an inductor and a resistor, as modelled in Figure 18. By using Kirchhoff's law and Laplace transform, the current can be described in relation to frequency, resulting in Equation (1) or in the time domain, resulting in Equation (2). These equations show that the supply voltage and the motor impedance primarily determine the current.

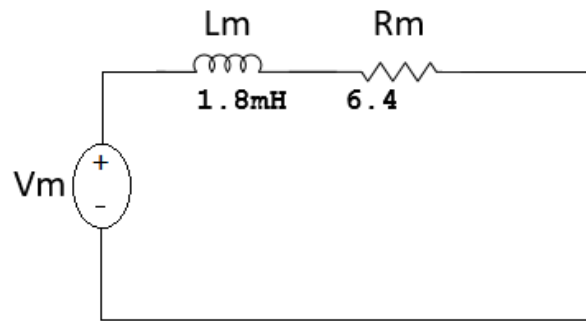


Figure 18. Resistor-inductor circuit model of a stepper motor. The following components are identified, motor voltage (V_m), motor inductance (L_m), and motor resistance (R_m). *The values are specified according to the stepper motor used in this project.*

$$I_M(s) = \frac{V_M(s)}{R_M + S \cdot L} \quad (1)$$

$$I_M(t) = \frac{V_M}{R_M} \cdot e^{-\frac{R_M \cdot t}{L_M}} \quad (2)$$

$$I_M = \frac{V_{DD}}{R_M} \quad (3)$$

L/R Drive

A general current limiting topology is to apply enough voltage so that the internal motor resistance determines the target current, so-called L/R drive, as represented by Equation (3). Because the motor torque is proportional to the current, short current rise time is preferred. According to Equation (2), a shorter current rise time requires, increasing the voltage supply or decreasing the inductance or resistance. Because decreasing the inductance or the resistance means changing the motor, our only reasonable option is to increase the voltage. However, the voltage also affects the current because it is only limited by the windings resistance. Therefore, an L/R-drive topology would be highly ineffective since the voltage is always at a minimum. [22]

L/nR Drive

One typical solution to the L/R-drive problem is to insert an external resistor in series with the motor and increase the supply voltage, a so-called L/nR drive. As the current builds up, most of the voltage occurs across the motor windings, resulting in short current rise time. However, when the current settles, the voltage is distributed between the low resistive windings and the higher resistive external resistor. Consequently, almost all power is dissipated by the external resistor which is highly inefficient as well as it may degrade the transistors due to the excessive heat. This drive topology is suited for applications that have unlimited current supply and excellent cooling capability. However, in space application that has strict requirements regarding efficiency and reduced cooling abilities, this would be a terrible solution. [18] [22]

Chop Drive

The optimal topology is a highly efficient driver that provides a short current rise time. This can be achieved by several methods. However, only the solution for the RSE driver is disused, a so-called chop drive.

The current control circuitry is a part of the driver shown in Figure 16. It involves one resistor named R_{sense} and two operational amplifiers $OP2$ and $OP3$ that control pulldown transistor $Q4$ and $Q3$ respectively. The transistors operate in a current source configuration, and the current results in a voltage drop over R_{sense} , which is fed back via the operational amplifiers [5]. The feedback acts as the current limiting operation, ensuring that the current never exceeds the target value. However, due to the significant power dissipation in the pull-down transistor when the current settles, an additional regulating solution is added. The regulation is controlled by the microcontroller and performs fast on/off switching of the pull-down transistors when the target current is reached (i.e., current chopping).

The chopping procedure significantly reduces the average current draw from the power supply. When the transistors are on, the only power loss is the saturation loss of the transistors, resistive loss of R_{sense} , and the winding resistance. However, when the transistors are switched off, the magnetic field around the windings collapses. The collapsing magnetic field induces a current that recirculates through a flyback diode, which results in no current draw from the power supply. Because the current decay rate is primarily determined by the relative slow windings time constant, this current recirculating topology is often referred to as slow decay.

Slow Decay

Figure 19 demonstrates the slow current decay principle for one coil and one current phase. During “on” time, the current is drawn from the power supply, enabling current build up. When the target current is reached, pull-down transistor $Q3$ switches off for a short period. Consequently, the windings magnetic field collapses and sets a positive voltage over the built-in flyback diode for transistor $Q2$ (“off time” in Figure 19). The forward biased diode enables the current to circulate through pull-up transistor $Q1$, the winding and the built-in flyback diode for transistor $Q2$. Due to the slow current decay, the current is approximately sustained without drawing current from the power supply.

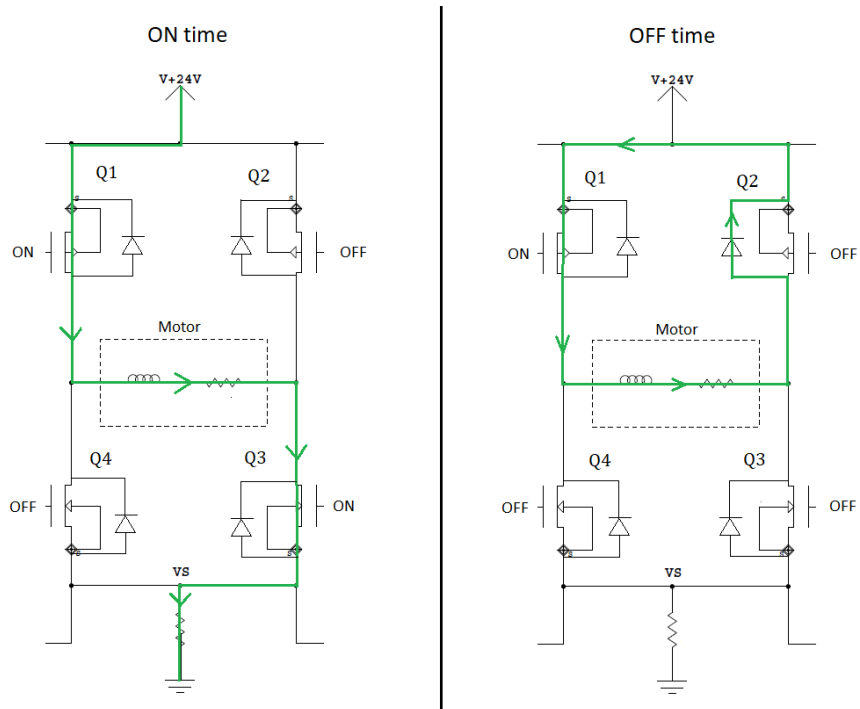


Figure 19. H-bridge Current flow during “on” time and “off” time [21]. *Arrows represent the current direction.*

Comparison of the Current Limiting Topologies

Figure 20 illustrates the simulated current step response for the different solutions when the target current is 0.78 A. For the L/R drive, 5 volts is applied, and only the motor resistance limits the current. For the L/nR-drive, an external resistor four times the motor-rated resistance is added, yielding a total resistance of 5 times the motor resistance. The voltage is increased five times such that nominal current remains the same. For the chop drive, the voltage is identical to that of L/nR drive, but no external resistor is added. When the nominal current is reached, the chopping operation is activated and holds the current at a nearly constant level. Figure 20 shows that each method eventually reaches the target value of 0.78 A, but the chop-drive has a significantly shorter current rise time.

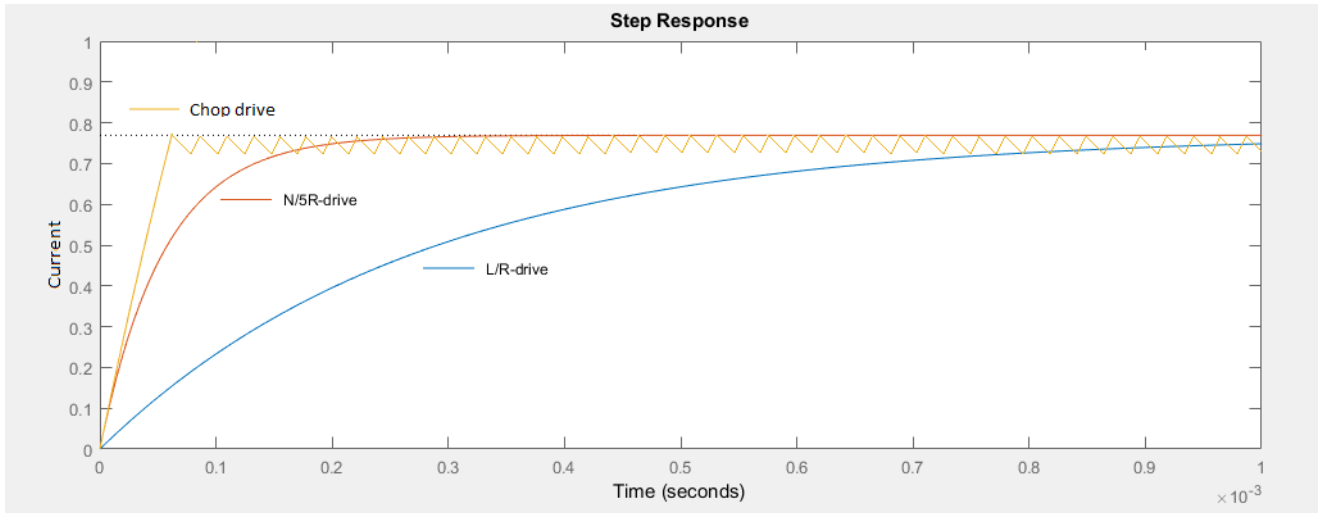


Figure 20. Current step response for L/R, L/nR and chop drives.

3.4.3 Voltage Regulator

The voltage regulator supplies the current control operation amplifiers *OP2* and *OP3* with a software configurable reference voltage signal, denoted V_{ctrl} in Figure 16. This signal is reduced by a factor of four before appearing at the amplifiers positive terminal. The voltage across the 1Ω resistor R_{sense} is fed back to the OP-amps negative terminal, resulting in a current limit defined by Equation (4).

$$I_{Motor} = \frac{V_{ctrl}}{4 \cdot R_{sense}} = \frac{V_{ctrl}}{4} \quad (4)$$

Figure 16 illustrates the voltage regulator that consist of the operation amplifier *OP1*, resistor *R1*, and capacitor *C1*. The output of *OP1* is direct feedback to the negative terminal. The voltage appearing on the positive terminal, reflects on to the output, without requiring a sufficient current source. This means that an output-pin from a microcontroller can be used as the source. By connecting the positive terminal across the low-pass filter's capacitor, and applying a high-frequency signal, the RC-time constant can be used to create an adjustable voltage, which is purely determined by the signal's duty cycle. Therefore, when the input frequency is much higher than the RC-time constant, V_{reg} can primarily be determined by Equation (5).

$$V_{reg} = V_{input} \cdot \frac{D_{cycle}}{100\%} \quad (5)$$

A simulation of the voltage regulator is presented in Figure 21, generated by the simulation tool Simulink. A Pulse-Width Modulated (PWM) digital single with a 3.3 V amplitude and a period of $32 \mu s$ is used as the input source. Figure 21 illustrates the output when the duty cycle of the input signal is 80% and 40%. The output eventually settles at the target values of 2.65 V and 1.32 V, respectively, with a small amount of ripple, which is a result of the high-frequency input signal.

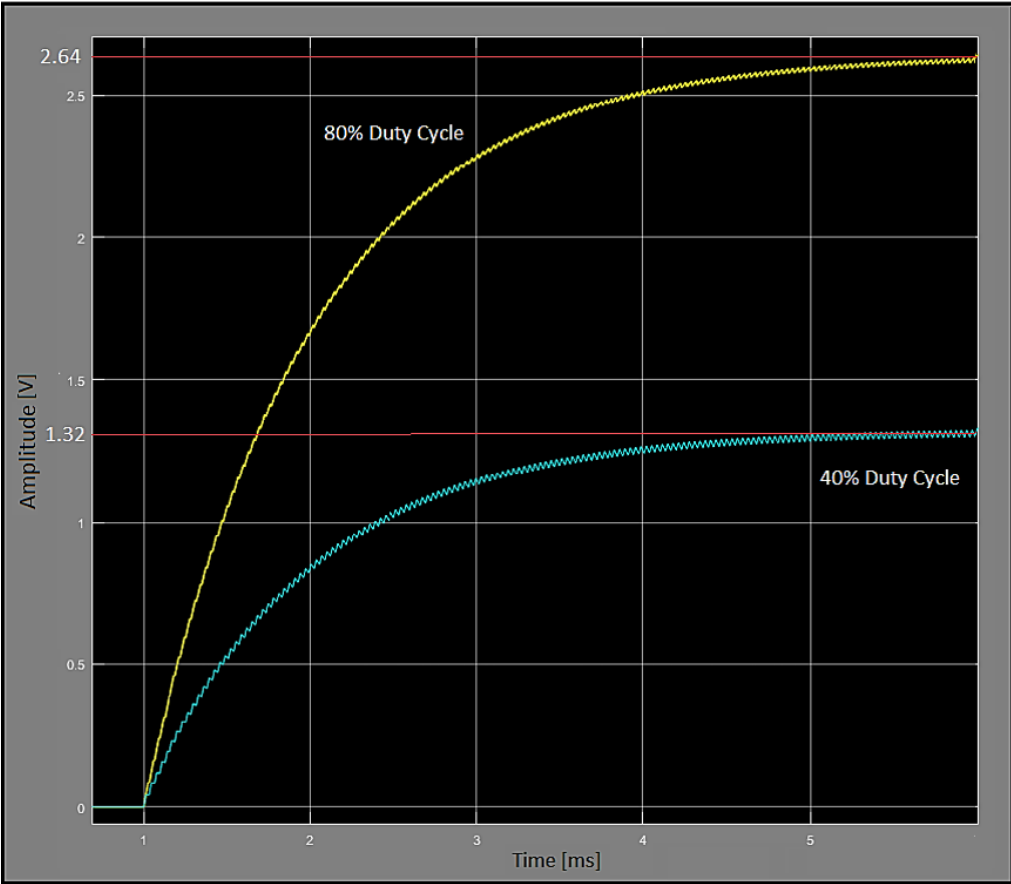


Figure 21. Voltage regulator output signals with different PWM input signals.

4 RSE Development

This chapter begins by introducing some central aspects regarding embedded system development, followed by a detailed description of the RSE control system design and implementation.

4.1 Embedded System

An embedded system is a single-purpose, computer-based unit, which often collaborates with a larger system, such as an automobile, or a satellite. Unlike a conventional computer, the software of an embedded system is highly dependent upon the hardware. Both the hardware and software are tailored towards a specific job, such as controlling the radiation shutter. Usually, embedded systems react to the surroundings through sensors and produce an output that may be vital for the greater system. As a result, strict requirements are imposed on these systems regarding timing, performance, and reliability. This section introduces three central aspects for embedded systems, interrupts, real-time system and task scheduling. For more information about embedded systems and the three central aspects, see for example: [23], [24], and [25].

4.1.1 Interrupts

Interrupts are one of the microcontroller's most powerful tools and essential for embedded systems. Interrupts are used when operations require asynchronous handling for example during interactions with the microcontroller's hardware. In essence, an interrupt is a signal to the CPU that requests execution of different code, corresponding to the provoked interrupt signal (generally called interrupt vector). The code is written within a particular handling routine, the so-called Interrupt Service Routine (ISR). Once an interrupt occurs, the main program is paused, the corresponding ISR is executed, and the main program is picked up where it left off. If interrupts are appropriately used, system performance and flexibility can increase significantly. For example, rather than continuously poll events, the events itself can cause interrupts and therefore not waste CPU resource. However, some precautions are required when dealing with interrupts. The list below describes three crucial safety measures that were emphasised during RSE development of this project.

- 1. ISR execution time:** Because an ISR occupies the CPU, the main program must wait until the CPU is available. A time-consuming ISR can affect tasks by means of timing jitter⁸, or even worse time-dependent tasks executed too late. A short ISR is especially crucial if interrupts occur frequently. An efficient solution for a short ISR is to temporarily store key parameters in the ISR and conduct the processing in the main program. However, response time will always be present before entering the ISR, and return time before the main program is resumed. For AVR microcontrollers, including the ATmegaS128, both the response and return times are four clock cycles, at a minimum [13]. This means that the main program is at a halt for at least eight clock cycles plus ISR execution time.
- 2. Volatile variable declaration:** According to the main program's awareness, the ISR is "invisible", meaning that variables updated within the ISR, may not be recognized by the main program. This is because the compiler can optimize multiple readings of a variable. However, by declaring variables shared between ISR and main program volatile, the optimization will not be present for that particular variable. Volatile declaration essentially tells the compiler that the variable may change outside the main program. Therefore, a volatile declaration forces the main program to read the variable each time it is specified in the code rather than using an old value which it may believe is up to date.
- 3. Corrupt variable:** An interrupt may occur while reading/writing a variable and thereby modify parts of the variable. For example, an eight-bit microcontroller, such as the ATmegaS128 can read/write one byte each clock cycle, two bytes need two clock cycles, and so on [13]. Corrupt variables are easily prevented by disabling interrupts before and enabling them after reading/writing a variable which requires more than one clock cycle to complete.

⁸ Task timing jitter is referred to as the deviation between periodic task executions.

4.1.2 Real-Time Systems

Most embedded systems, including the RSE, fall under the real-time system category. A real-time system requires, completion of processes, tasks or responses within a specific period. This is often referred to as timing constraints. Exceeded timing constraints can risk fatal consequences. In general, real-time systems are categorised as hard or soft, based on the response time and the consequences regarding a timing violation.

- **Hard real-time system:** An operation completed after the deadline will lead to severe failures, such as a human fatality or an expensive equipment breakdown. In addition, the response time of this type of system is tight, generally in milliseconds or less. A typical example of a real-time system is the anti-lock braking system (ABS) in a car. The ABS must guarantee a predictable and instantaneous response. Failing to do so may result in a car crash.
- **Soft real-time systems:** These systems only require operations to meet an average response time. Although a time constraint violation will have a significant impact on system performance, this type of system can tolerate it. Instead, only when time violations occur repeatedly, it will be considered as an error. The system responsiveness is typically in the order of seconds and generally less complex than that of a hard real-time system. A typical soft real-time system is, for example, a washing machine; a program starting a second or two later than usual is hardly recognisable.

As described in Section 2.4.3, the RoR communication protocol requires that a query is answered within 24 baud delays, which corresponds to 120 μs . In addition, the response cannot be initiated before 1 baud delay which corresponds to 5 μs . Failing to do so may prevent the RSE from receiving essential commands, potentially resulting in radiation-damaged CCDs or unintentionally blocked CCDs. Because both scenarios would have severe consequences, the RSE requires a hard real-time system. However, the RSE also performs other tasks, such as HK-readouts and motor control that only need an average response time. A soft real-time system would be sufficient for these tasks. Therefore, to accommodate both system types without increasing complexity, the RSE software is designed as a hybrid of hard and soft real-time systems.

4.1.3 Task Scheduling

As previously discussed in this chapter, most real-time systems manage several time-dependent tasks. The tasks must, therefore, be organised such that all operations are accomplished without compromising reliability and performance. Because embedded systems have limited resources, task management often requires multitasking. Thus, the tasks are handled by a software algorithm called a scheduler. During the last four decades, various scheduler topologies have been developed, ranging from sophisticated schedulers such as Real Time-Operating Systems (RTOS) to a simple scheduler that loops through a fixed set of tasks. Hard real-time systems is generally constructed as an RTOS because of its ability to suspend a task, execute higher priority tasks, and resume the suspended task later. This guarantees that the higher priority tasks are executed within the deadline. On the other hand, RTOS generally requires a complex scheduler algorithm, which often is hard to debug and occupies more memory than a simpler scheduler. In addition, the task swapping overhead increases significantly with RTOS, since it must spend time on deciding which task is going to be executed.

Because the RSE only handles one hard real-time task, the use of an RTOS just for that single purpose alone is excessive. Consequently, it was decided to design a custom, cooperative scheduler which loops through a set of tasks, for the RSE. Although this type of scheduler is perfect for soft real-time tasks, it may not fulfil the strict hard real-time constraints of the RSE. However, by exploiting the microcontroller's interrupts, the hard real-time constraints can also be met. As a result, the task which handles the communication (i.e., the hard real-time task) is located in an ISR, and the cooperative scheduler controls the tasks with soft constraints. This solution has some similarities to RTOS because interrupts can also suspend the ongoing task and execute the higher priority task. However, the difference is that this solution only works if the amount of hard real-time tasks is limited, because the microcontroller only has a few interrupt sources. Furthermore, an RTOS can return to any of the tasks after a higher priority task is executed. The cooperative schedule can only resume the suspended task after the interrupt driven tasks are complete. As a result, the RSE scheduler solution is not as flexible but, the reduced complexity, almost instantaneous task swapping, and minimal memory usage makes this an acceptable trade-off. The design and implementation of this scheduler are further explained in Section 4.4.

4.2 SW Development Strategy

An intuitive software development approach involves two simple steps, first, analyse the problem, and then start coding. This approach may give the impression of an efficient and reasonable method because time spent on documentation is not in the picture. Unfortunately, this technique is practically guaranteed to fail and definitely more time-consuming. Poorly planned programmes often suffer from bugs and complex structure. This makes debugging harder, resulting in “invisible” bugs or those, discovered late in the proses, might require re-constructing of the whole program.

A better approach that eliminates such unprofessional mistakes is planning and preparation before writing the code. A typical development strategy that follows these principles is the waterfall design strategy, which was used for the contents of this thesis. It involves four basic steps as shown in Figure 22, and requires that each step is completed before moving onto the next. First, overall requirements from the customer are analysed, and software requirements are produced. Second, developers develop a design plan that describes a detailed solution of the software construction and behaviour. Third, the design document is transformed into actual code. In contrast to the other steps, this is often the least time-consuming step because the problem solving and “error-free” design was prepared in advanced. Finally, the product is tested to verify it meets its requirements. For more information about the waterfall design strategy, see for example [26].

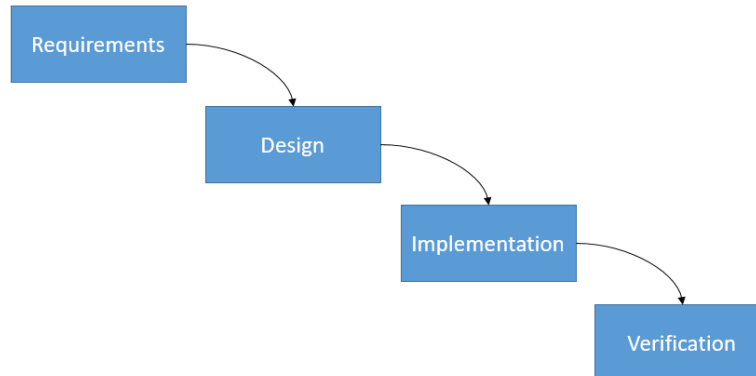


Figure 22. Waterfall design strategy. *Each step in the software development lifecycle is completed before moving to the next. This approach will enable errors to be found early in the development process, thereby reducing cost and time.*

Although the SMILE project has a 4-years duration and is currently in its early phase, this thesis was limited to only one year. Because the RSE requirements frequently change, it was not possible to follow the waterfall strategy exactly. Hence, some adjustments in the earlier “completed” stage were required. Furthermore, measurements such as stepper motor torque and driver performance needed to be confirmed before further design could be accomplished.

4.3 RSE Software Design and Implementation

The RSE software is implemented using the C programming language, due to its ability to accommodate both higher and lower level functionalities, such as hardware access through registers. In addition, Atmel Studio 7 provides an excellent C-compiler, designed for programming of AVR microcontrollers, such as the ATmegaS128. However, the C language can be prone to errors, even if the code is compiled error free. These errors can, for instance, be caused by poor syntax that is technically correct according to the C language. Because the RSE software must meet a high-quality standard, these errors must be prevented. Consequently, the RSE software is designed and implemented with emphasis on the MISRA-C standard. MISRA-C defines a set of C language programming rules which is developed by the Motor Industry Software Reliability Association (MISRA). By following these rules, the implemented code will be significantly more reliable. [27] [28] [11]

Figure 23 shows an overview of the RSE software. The boxes represent the key modules, and the arrows represent the execution flow. The scheduler contains three modules: idle, sporadic dispatcher and periodic dispatcher. These modules are sequentially executed in a loop configuration. Note that the communication module can suspend the ongoing scheduler tasks, as indicated by the lightning symbol.

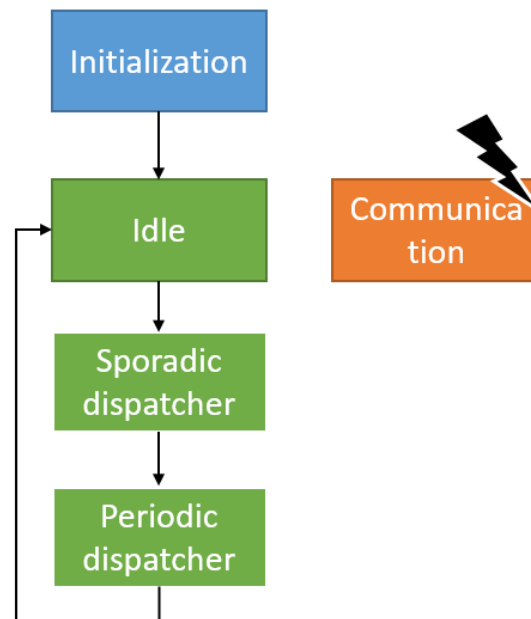


Figure 23. RSE software overview. First, the program completes the initialisation routine, then the scheduler is entered (green) and loops through three software modules, in a sequential manner.

4.3.1 Initialisation Routines

Several initialisation routines are executed when the RSE is started or restarted. These routines configure system settings and set register default values, including the following:

- **Initialise IO-ports:** configure port mode (input/output) and set default values.
- **Initialise register:** set register default values.
- **Initialise Timer0:** configure HW-timer-0, (controls the driver voltage regulator).
- **Initialise Timer1:** configure HW-timer-1, (controls scheduler and driver settling time).
- **Initialise Timer2:** configure HW-timer-2, (controls three driver chopping pins).
- **Initialise Timer3:** configure HW-timer-3, (controls one driver chopping pin).
- **Initialise UART0:** configure UART-0, (utilises serial communication).
- **Enable global interrupt:** enable all interrupt lines.

4.4 Scheduler

The scheduler is entered once all initialisation routines are completed. Timer1 synchronises the scheduler, ensuring that the scheduler loops in a constant interval. Within one cycle, two dispatcher functions are always executed, named “sporadic dispatcher” and “periodic dispatcher”. Both dispatchers contain several tasks that they are responsible for determining which tasks are given CPU access.

This cyclic scheduler topology requires that each task performed within a cycle must also complete before the period ends. Because of this, tasks that need long operation time are split into many subtasks. An example of such tasks is the motor-related procedure. It may require thousands of steps before the shutter reach its final destination. The step configuration (stimulating the H-bridge transistors) itself is a quick operation, in the order of microseconds. However, between each step, a delay must be present, typical in the order of milliseconds. The delay defines the stepper motor speed, resulting in a severe execution timer. Therefore, the scheduler interval time is chosen by the stepper motor speed, where the scheduler performs one step each cycle.

Due to the limited lubrication capabilities in space applications, the motor is restricted to operate below 100 rpm. Because the scheduler interval time is used as a reference time for the program, it is convenient to operate with a round number. Hence, the scheduler interval time of 2 ms was selected, which corresponds to a motor speed of 75 rpm. This time ensures that the delay between each step is preserved by the scheduler cycle. This also gives an impression of multitasking, since after the step is configured, the scheduler has plenty of available time to perform other tasks, such as HK-readouts. Because the scheduler runs in intervals of 2 ms, the worst-case response time will also be 2 ms and the average response time is 1 ms. Hence, this scheduler topology will be a responsive solution.

4.4.1 Idle Task

The idle task preserves the program's timing, by acquiring CPU attention while waiting for a cycle to end. This guarantees a fixed execution rate of queued tasks, where the first task in the queue notices the least amount of timing jitter, practically none. Therefore, the most time-dependent task is placed first in the queue. Figure 24 illustrates the task handling principle for two exemplified cycles. Each cycle lasts for precisely 2 ms, regardless of the number of tasks or tasks execution time, as maintained by the idle task. The idle task stays in a continuous loop and waits for a flag which is controlled by the ISR for Timer1. Every second ms, timer1 generates an interrupt which enables the ISR to set the flag. This procedure terminates the idle task, resets the flag and executes all tasks in the queue. Figure 24 also identifies the timing jitter present in task 3. However, the first task is guaranteed a permanent execution interval, unaffected by the other tasks execution time.

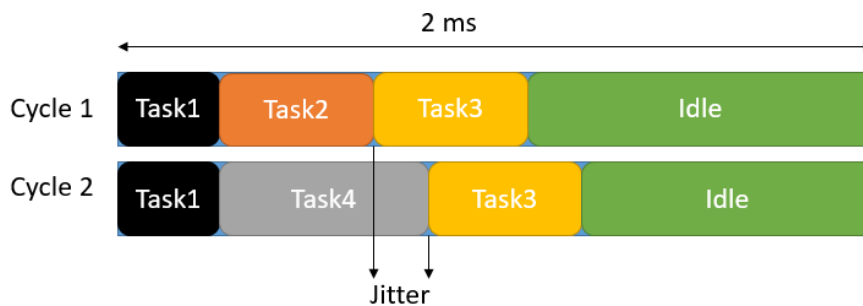


Figure 24. Scheduler task handling principle for two cycles. *The first task (Task1) has a fixed execution interval. The other tasks may be affected by timing jitter, as indicated for Task3.*

4.5 Periodic Dispatcher

The periodic dispatcher contains all tasks that require a fixed execution interval. The dispatcher includes the following HK readout tasks as listed below:

1. Measure motor temperature
2. Measure electronics temperature
3. Measure open shutter end switch
4. Measure closed shutter end switch
5. Measure HDRM arm time

The dispatcher determines the execution interval for these measurements. For example, while the motor is running, a temperature reading must frequently be measured; however, when the motor is inactive, a longer measurement interval is adequate. By utilising the known scheduler interval time (2 ms) and count number of iterations, the dispatcher is provided a reference time. For each cycle, the dispatcher compares the elapsed time with the task's defined interval time. A match between elapsed time and interval time results in a measurement. All measurements are directly stored in their corresponding status register, which is further described in Section 4.8.1. The periodic measurements' interval time can be configured in the periodic dispatcher's header file, as shown in Listing 1. Note that the unit is the number of scheduler cycle iterations. Thus, if the scheduler interval time is changed, these variables must also be changed.

```

1. //-----Measurement interval time-----//
2. #define motor_ON_temp_interval 500UL
3. #define motor_OFF_temp_interval 15000UL
4. #define motor_ON_electronics_temp_interval 500UL
5. #define motor_OFF_electronics_temp_interval 15000UL
6. #define motor_ON_shutter_status_interval 1
7. #define motor_OFF_shutter_status_interval 10000UL
8. #define HDRM_armed_time 500UL

```

Listing 1. Macros for adjusting the periodic measurements. *These parameters are located in the header file for the periodic dispatcher (periodic_dispatcher.h).*

In contrast to the “sporadic” tasks, “periodic” tasks are executed independent of the DPU and does not require precise timing. Because of this, “periodic” tasks are always put last in the queue and therefore may be affected by a timing jitter.

4.6 Sporadic Dispatcher

The sporadic dispatcher contains all tasks that interact with the stepper motor or the HDRM. These tasks can be initiated at any time, hence the name “sporadic”. The dispatcher includes the following tasks:

1. Open shutter stop at end
2. Close shutter stop at end
3. Open shutter max no of steps
4. Close shutter max no of steps
5. Emergency close stop at end
6. ARM HDRM
7. Activate HDRM

This dispatcher only executes its tasks, if the DPU has requested it. Unfortunately, the RoR protocol is only capable of performing memory read and write operations. Thus, the protocol does not specify how the master (DPU) should command the slaves (e.g., open the RSE shutter). Because of this, the RSE has an 8-bit register named `command register`, which the DPU can access. If the DPU updates the `command register` with one of the command identifiers shown in Table 5, the register will be read by the RSE, and the corresponding operation is performed. The DPU can also read the register for verification, and it will read the issued command identifier, as long as the operation is active. After a finishing operation, the register will be cleared to 0x00, which also is the default value. This procedure means that if the DPU continuously reads the status registers and the command register, the DPU can safely monitor any ongoing operation. The sporadic dispatcher is placed first in the queue to prevent irregular stepping interval. As previously discussed, the first task is unaffected by a timing jitter. Timing jitters can potentially introduce motor resonance and thereby increases mechanical stress.

Table 5. Command register overview [5].

RSE operation	Address	Command identifier
Open shutter Stop at end	0x80	0x01
Close shutter Stop at end	0x80	0x02
Open shutter Max no of steps	0x80	0x04
Close shutter Max no of steps	0x80	0x08
Emergency close Stop at end	0x80	0x10
Arm HDRM	0x80	0x40
Activate HDRM	0x80	0x42
Cancel command	0x80	0x80

4.6.1 Open/Close Shutter Stop at End

During normal operation will an *open shutter stop at end* command, run the motor in open direction until the “open” end end switch is activated. However, due to scenarios such as end switch malfunction or electronics/motor overheating, two more criteria are added. The functional structure is best described by a flowchart, as shown in Figure 25. The scheduler will run this procedure each cycle until the “open” end switch is activated, the motor/electronics overheat, or the number of performed steps equals the maximum stepping limit. The same procedure is performed in a *close shutter stop at end* operation, except that the motor runs in close direction and “closed” end switch must be activated. Notably, a cancel command operation terminates all ongoing operation.

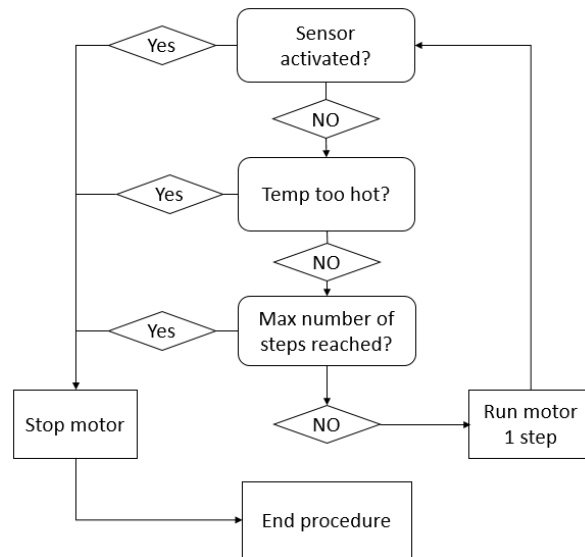


Figure 25. Software behaviour for an ongoing open/close shutter stop at end operation.

4.6.2 Open/Close Shutter Max no of Steps

The Open/close shutter max no of steps procedures are optional tasks and only used if the “open” end switch is stuck activated. As described in Section 3.1.1, the RSE stepper motor operates in an open loop configuration. Thus, the number of performed steps are not measured. However, the performed steps is represented by the programs iterations. For this reason, these two tasks are identical to the corresponding open shutter stop at end, and close shutter stop at end procedures, except the end switch criteria is excluded. Figure 26 describes the functional structure of these procedures.

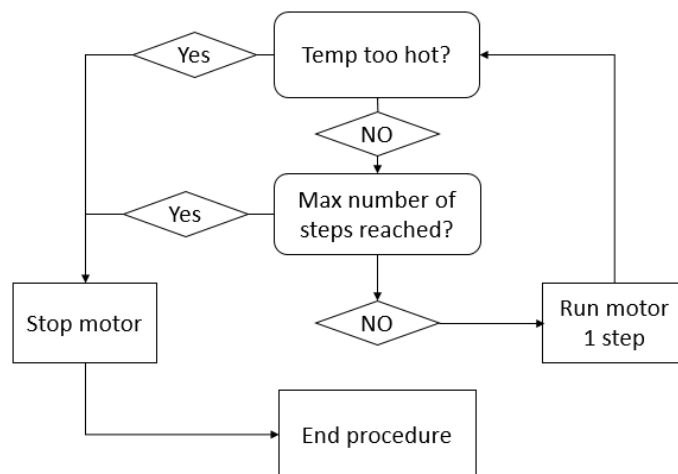


Figure 26. Software behaviour for an ongoing open/close shutter max no of steps operation.

4.6.3 Emergency Close Stop at End

As the name implies, `Emergency close stop at end` is a task used during an emergency. Such an emergency may be, if stray light from the Moon suddenly strikes the CCDs or the communication is broken. Either scenarios, the CCDs must be protected as fast as possible. Therefore, this procedure is identical to the `close shutter stop at end` operation, except for the increased motor speed and the removed temperature criteria. Figure 27 presents the functional structure of this emergency procedure.

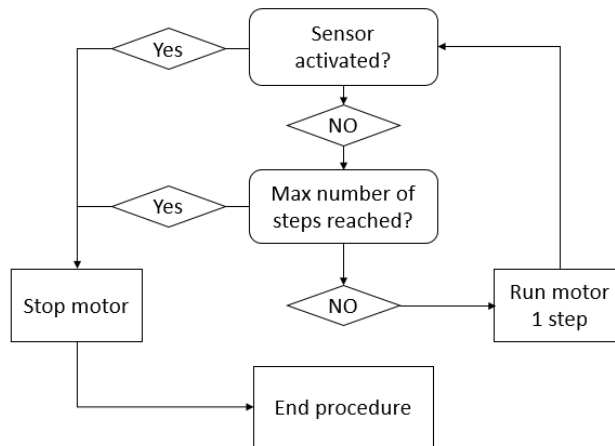


Figure 27. Software behaviour for an ongoing `emergency close stop at end` operation.

4.6.4 Arm/Activate HDRM

The `arm HDRM` procedure is an additional safety future that prevents undesirable activation of the HDRM. Figure 28 illustrates the `arm HDRM` procedure. During an issued `arm HDRM` command, a software implemented counter increments for each cycle until it reaches one second (i.e., 1000 scheduler cycles). The HDRM can only be activated within this time window. In addition, all unrelated “memory write” operations disarm the HDRM, so the process must be repeated. However, a successfully issued HDRM command triggers the actuator and releases the shutter.

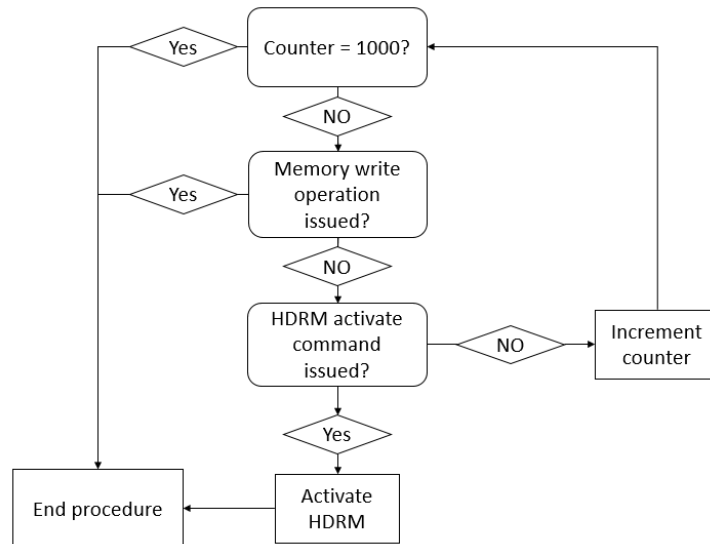


Figure 28. Software behaviour for an ongoing `arm` HDRM operation.

4.7 Communication Task

The communication task is responsible for interpreting received data and engaging the transmission procedure. Communication between the DPU and the RSE must follow strict timing requirements, which is considerably shorter than the scheduler time resolution of 2 ms. Therefore, the communication task cannot be controlled by the scheduler. As discussed in Section 4.1.3, this problem could be solved by implementing an RTOS where the communications task has the highest priority. However, this solution would significantly increase the system's complexity and therefore may make it more prone to error and not as responsive. Instead, the communication task was implemented using an ISR. The ISR executes after reception of a full data frame, known as the “receive complete ISR”, and follows an atomic behaviour, meaning that other tasks cannot interrupt the task. The communication task is kept as short as possible so that it does not interfere with the timing of other tasks. It is also structured and named like the RoR stack described in Section 2.4.1. This structure significantly increases readability which is beneficial for further development.

Figure 29 illustrates the communications task execution flow where received data is sorted and interpreted through several levels. Errors discovered at the character level, packet level or RMAP level will end the procedure eminently. This procedure is further explained in the sections that follow.

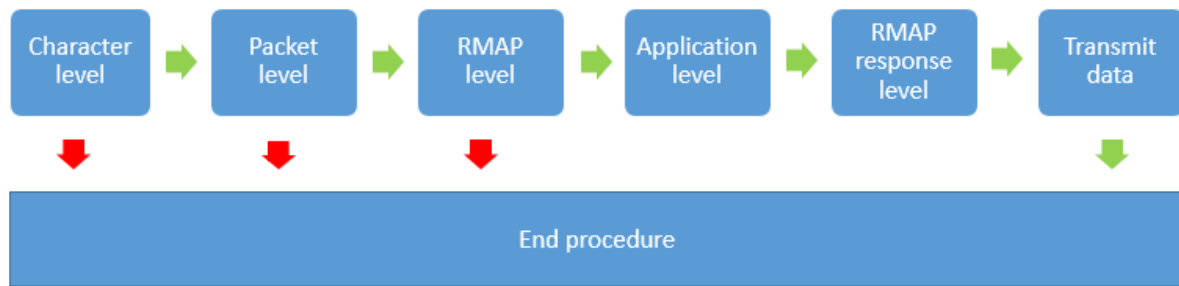


Figure 29. Communication task functionality overview.

A complete read and write transaction requires 3 and 5 characters, respectively. Each character contains information that needs to be stored until the necessary read/write operation is complete. This means that the characters information must be stored in variables that retain its value during the program lifecycle. A static variable declaration within a function could solve this, but due to increased readability, the variables are implemented globally in a struct, as shown in Listing 2.

```

1. struct RMAP{
2.     uint8_t R_nw;           // Read write indication
3.     uint16_t register_address; // Acquired RSE register
4.     uint8_t register_data_to_write; // New register content
5.     uint8_t sequence;      // Query sequence
6.     uint8_t reject_query;  // Query rejection
7.     uint8_t remark;        // RMAP error ID
8. };
9. struct RMAP RMAP_struct;
  
```

Listing 2. RMAP struct. *Contains the communication task variables that must retain its value. Located in RMAP.c.*

The RMAP struct is declared and defined in the communication tasks' source file, limiting the scope to that specific source file. This programming practice reduces the possibility that these variables are unintentionally modified by other functions. Variables that only require one communication task life cycle is passed through the different levels as function arguments. Hence, these variable does not consume memory when the communication task is finished.

4.7.1 Character Level

Character level utilizes the UART hardware module, where the serial signal is converted into characters. A complete character reception triggers the "receive complete interrupt" and only then is the communication task initiated. The task starts by verifying that received data does not contain bit errors. A bit error will end the procedure at the character level, resulting in a short ISR routine. If not, the sequence continues to the packet level. Table 6 shows the interface signal between the character level and the packet level.

Table 6. Character and packet level interface signal

Interface signal	Description
UART_raw_data	16-bit local variable, containing an error-free character, received by the UART

4.7.2 Packet Level

Packet level is responsible for detecting package type (response/query) and only continue to the next level if a query is received. Due to the shared bus topology, the RSE will read all data exchanged on the bus, including response packets sent from other slaves. Therefore, response packets must be discovered as soon as possible so that CPU is not unnecessarily occupied. Table 7 shows the interface signal between packet level and RMAP level.

Table 7. Packet level and RMAP level interface signal

Interface signal	Description
query_data	8-bit local variable containing query data

4.7.3 RMAP Level

At this point, the received data does not contain bit errors and is verified as being sent from the DPU (i.e., a query is received). The query is then sent through a sorting algorithm that identifies the query type and sequence. If both of these identifiers are correct, the content is sorted and stored in the corresponding struct variables, according to the query type. The sequence indicates the order of the received queries, where an invocation query is the first in the series. If a different query (other than an invocation query) is received, while the RSE expect an invocation query, the procedure is terminated. Table 8 shows the interface signals between the RMAP level and the application level. Table 9 describes the RMAP level interaction with the struct variables, depending on the query type.

Table 8. RMAP level and application level interface signals

Interface signal to application level	Description
query_type	8-bit local variable, identifying query type
sequence_error	8-bit local variable, signalling sequence error

Table 9. RMAP-level struct variable interactions

Query type	Struct variables	Description
Invocation	Sequence = 1	Next expected query is an instruction-query
Instruction	R_nw = query_data [bit 4]	Indicate read or write request
	Register_address [bit 8:5] = query_data [bit 3:0]	Transfer the lower 4 bits, containing RSE register address [8:5]
	Sequence = 2	Next expected query is a Reg-address-query
Reg-address	Register_address [bit 4:0] = query_data [bit 4:0]	Transfer the lower 5 bits, containing RSE register address [4:0]
	Sequence = 3 or 0	Next expected query is a write data H-query or invocation-query, depending on write or read request
Write data H	Register_data_to_write [bit 7:4] = query_data [bit 3:0]	Transfer the lower 4 bits, containing the RSE register data to write bit [7:4]
	Sequence = 4	Next expected query is an write data L-query
Write data L	Register_data_to_write [bit 3:0] = query_data [bit 3:0]	Transfer the lower 4 bits, containing the RSE register data to write bit [3:0]
	Sequence = 0	Close communication. Next expected query is an Invocation-query

4.7.4 Application Level

Application level determines if the query's instruction is authorised or rejected and handles the RSE register access. Only a query that is authorised results in a register write or register read operation (i.e., a rejected query is not granted register access). In addition, application level updates two of the RMAP struct variables, as shown in Table 10. Causes of query rejection and their corresponding remark ID is presented in Table 11. Notably, an authorised query also results in a remark ID. The process then proceeds to RMAP response level where the interface signal "query type" is passed as an argument.

Table 10. Application level struct variables interactions

Query type	Struct variables	Description
Invocation	reject_query [bit 0] =1/0	Invocation query is rejected/authorised
	remark = ID	Identifies cause of rejection or authorised
Instruction	reject_query [bit 1] =1/0	Instruction query is rejected/authorised
	remark = ID	Identifies cause of rejection or authorised
Reg-address	reject_query [bit 2] =1/0	Reg-address query is rejected/authorised
	remark = ID	Identifies cause of rejection or authorised
Write data H	reject_query [bit 3] =1/0	Write data H query is rejected/authorised
	remark = ID	Identifies cause of rejection or authorised
Write data L	reject_query [bit 4] =1/0	Write data L query is rejected/authorised
	remark = ID	Identifies cause of rejection or authorised

Table 11. Remark ID

Response	ID	Description
Authorised	0x00	The operation is correct and accepted by the RSE
Sequence error	0x01	RMAP protocol sequence error
Not writable register	0x02	The register is read-only
Wrong address	0x03	No such register
Not allowed	0x04	Not allowed since a command is already being performed
Shutter already closed	0x05	Not allowed since the shutter is already closed
Shutter already open	0x06	Not allowed since the shutter is already open
Motor temperature too high	0x07	Not allowed since the motor is too hot
Electronics temperature too high	0x08	Not allowed since electronics is too hot

4.7.5 RMAP Response Level

RMAP response level organises the response packets according to the RoR communication protocol described in 2.4.1. This means that RMAP response level also sends the two first characters through a CRC algorithm that generates the third character. In general, CRC calculations are time-consuming operations that may considerably increase the ISR execution time. Therefore, it is crucial to use an optimum solution where the execution time is at a minimum. AVR supports a wide range of libraries, among them the `crc16.h` that provides an optimised, inline function for calculating CRC. Therefore, this function was selected for this project because creating a custom algorithm will most likely not be as efficient. After a complete response packet, the characters are stored in the transmit First In First Out (FIFO), and the transmit procedure is initiated.

4.7.6 Transmit Data

The transmit procedure is the smallest and simplest procedure. Nevertheless, it is the most important part for keeping the ISR execution time at a minimum. The transmit procedure is responsible for sending a whole response packet, which contains three characters. Because the characters must be transmitted one at the time, the ISR routine must wait until the next character can be sent. This is an inefficient solution and often referred to as “polling operating”, where the CPU must continuously check if the UART transmit register is available. However, by utilising the UART transmit complete interrupt line, this problem is eliminated. Therefore, the transmit procedure is implemented in two separate ISR routines. The first character stored in the transmit FIFO is buffered onto the UART transmit register in the same ISR as the other levels. While the first character transmits, the CPU returns to the main program. Once the first character is sent, the “transmit complete interrupt” triggers. The CPU then returns to the transmit ISR, buffers a new character onto the UART transmit register and returns to the main program. This procedure continues until the transmit FIFO is empty (i.e., until all three characters are transmitted).

4.8 RSE Register Overview

The RSE registers serve as the interface between the DPU and the RSE. This interface means that all interaction between the DPU and the RSE, involves access registers through memory read or memory write operations. Because the communication protocol requires it, all registers are of 8-bit size. In addition, the ATmegaS128 uses an 8-bit architecture [13]. Thus, a register can be written or read within one clock cycle. The registers are further divided into three categories:

1. Status registers
2. Control registers
3. Command register

Unique addresses identify the registers, where each register categories are organised in a separate struct, of global scope. This does not affect performance, only increases readability. Listing 3 shows the control registers declaration. Not that these registers are declared “volatile”. As discussed in Section 4.1.1, volatile declaration informs the compiler that a variable may change outside the main environment. Because the registers are both accessed through the communications task ISR and the main program, the volatile declaration guarantees that the variable/register is up to date.

```

1. //-----Control registers-----//
2. struct control{
3.     //      TYPE      Reg name      Address
4.     volatile uint8_t Motor_current;    // 0x40
5.     volatile uint8_t Settling_time;    // 0x41
6.     volatile uint8_t Chop_D_cycle;    // 0x42
7.     volatile uint8_t Max_motor_temp;    // 0x43
8.     volatile uint8_t Max_electronics_temp; // 0x44
9.     volatile uint8_t Max_step_for_operation; // 0x45
10. };
11. extern struct control control_reg;

```

Listing 3. Control registers declaration. *Located in RSE_registers.h.*

The following sections describe the status registers and the control registers. The command register is previously discussed in Section 4.6.

4.8.1 Status Registers

The status registers contain HK readouts from sensors or other status variables (e.g., temperature and motor step iterations). The RSE regularly updates these registers, and they are always readable by the DPU. However, the status register does not allow DPU writing access, (i.e., from the DPU standpoint, a status register is read-only). Table 12 shows the status registers and their corresponding addresses. [5]

Table 12. Status register overview [5].

Register	Address
Software version	0x00
Motor temperature	0x01
Electronics temperature	0x02
Shutter status	0x03
HDRM status	0x04
Performed steps L	0x05
Performed steps H	0x06
Heartbeat count	0x07
Processor status	0x08

Software Version

The software version register contains the embedded software revision number. This register is only updated during patching.

Motor Temperature

This register contains the motor temperature value and is updated during a motor temperature measurement. The periodic dispatcher determines the measurement interval. The motor is inactive for most of the time, and thus the temperature should remain relatively stable. Because of this stability, the periodic dispatcher engages a measurement once every 30 seconds when the motor is inactive. By contrast, while the motor is active, temperature increases significantly, and therefore, a measurement is performed once every second.

Electronics Temperature

The electronics temperature register contains the driver circuit temperature value and is updated during an electronics temperature measurement. The measurement procedure is identical to that of a motor temperature measurement.

Shutter Status

The shutter status register indicates the central status of software and hardware. The register bits represent a single event where a high bit characterises a triggered event. Table 13 describes the shutter status registers structure.

Table 13. Shutter status register structure [5].

Event	bit	Purpose
Shutter is closed	0 (LSB)	The shutter is closed, closed end stop switch is active
Shutter is open	1	The shutter is open, open end stop switch is active
Close shutter in progress	2	A close shutter command has been issued and is being performed
Open shutter in progress	3	An open shutter command has been issued and is being performed
Emergency closure initiated	4	An emergency closure has been initiated due to a missing heartbeat signal from the DPU
Motor to hot	5	Motor exceeds maximum temperature range
Electronics to hot	6	Electronics exceeds maximum temperature range
-	7 (MSB)	Spare

HDRM Status

The HDRM status register indicates the status of HDRM related software. Therefore, no sensors measure the HDRM status, only events indicating performed HDRM tasks. The HDRM must be armed before the HDRM are permitted activation, as indicated by the `HDRM_armed` status bit. Table 14 shows the HDRM status events.

Table 14. HDRM status register description [5].

Event	Bit	Purpose
HDRM armed	0 (LSB)	HDRM is armed
HDRM activated	1	HDRM is activated
-	2	Spare
-	3	Spare
-	4	Spare
-	5	Spare
-	6	Spare
-	7 (MSB)	Spare

Performed Steps L and Performed Steps H

The `performed_steps` registers contain a 16-bit number that reflects the stepper motor iterations. `performed_steps L` contains the step number lower byte (bit [7-0]) and `performed_steps H` contains the high byte (bit [15-9]). `performed_steps L`, increments each step and an overflow increments `performed_steps H`, and is default `0x00` if the motor is off. By reading these registers, the DPU can track the shutter position.

Heartbeat Count

As discussed in Section 2.2, the RSE does not sense the radiated environment, only act on commands from the DPU. Therefore, if the communication between the DPU and the RSE fails, the CCDs must be covered. The heartbeat count register act as a software implemented watchdog timer. The DPU must access and read this register regularly, for example, every 30 seconds. Every read operation will increment its content, with wrap around. However, failing to read it regularly will initiate the automatic emergency closure of the radiation shutter. This functionality ensures that the radiation shutter is shut by default if the DPU is inactive. [5]

Processor Status

The processor status register indicates the main status of the software and hardware, as described in Table 15

Table 15. Processor status register, description [5].

Event	Bit	Purpose
Heartbeat missing	0 (LSB)	DCE did not receive heartbeat signals from the DPU
-	1	Spare
-	2	Spare
-	3	Spare
-	4	Spare
-	5	Spare
-	6	Spare
-	7 (MSB)	Spare

4.8.2 Control Registers

Contrary to the status registers, control registers are only updated by the DPU (i.e., from the RSE point of view, control registers are read-only). These registers configure the RSE motor operation parameters. Therefore, the DPU is prevented from updating these registers while the motor is active. Table 16 outlines the control registers, address and the register default values.

Table 16. Control registers [5].

Register	Address	Default value
Motor current	0x40	205
Settling time	0x41	105
Chop duty cycle	0x42	115
Max acceptable motor temperature	0x43	TBD
Max acceptable electronics temperature	0x44	TBD
Max steps for operation	0x45	TBD

Motor Current

This register contains the parameter that sets the motor current limit. The current limit is adjustable between 0 and 0.8 A, which corresponds to a register value range from 0 to 255. The register default value is 205, with a corresponding nominal motor current of 0.6 A.

Settling Time

This register contains the settling time parameter, which determines the delay before the current chopping operation starts. The delay is configurable from 1 to 255 μ s, which corresponds to a register value range from 1 to 255.

Chop Duty Cycle

The `chop duty cycle` register determines the duty cycle of the chop signal. The duty cycle is configurable from 0 to 100%, which corresponds to a register value from 0 to 255.

Max Acceptable Motor Temperature

This register determines the maximum acceptable motor temperature. The temperature limit range is not yet concluded. However, by adjusting this register to its maximum value, the temperature limit will be infinity. This is implemented because most of the motor-related procedures are suspended if the temperature is over the limit defined by this register. A malfunctioning end stop sensor could potentially read a high value and thereby prevent the motor from operating.

Max Acceptable Electronics Temperature

This register is identical to the `max acceptable motor temperature` register, except that it determines the electronics temperature limit.

Max Steps for Operation

This register determines the absolute number of steps the motor is allowed for one operation. Therefore, any ongoing operation will be aborted if the performed steps exceed the limit. However, if the `max steps for operation` register is set to the maximum value (255), the limit will be infinity, thereby not interfere with the ongoing operation.

4.9 Stepper Driver Software

At the time the driver software was designed, empirical test data concerning the driver hardware was not yet established. However, data from the driver simulations and the driver schematic was available. Regarding the stepper motor, only the following specifications were determined:

1. Hybrid stepper motor
2. 200 steps per rotation
3. Gear ratio: 1:196
4. Max motor speed: less than 100 rpm
5. Motor torque: 1.2 Nm or more

The software is therefore made to be highly generic, such that could easily be configured when the data become available.

4.9.1 Stepper Motor Operation Mode

In general, a stepper motor can operate in three distinctive modes: micro stepping, half step, and full step. These operation modes affect the motor's behaviour differently (e.g., motor smoothness, torque characteristics, and driver software complexity). Therefore, the operation mode must be carefully considered while designing the driver software.

The following four subsections briefly describe the operation modes. For the sake of simplicity, all examples are based on a motor with one pole pair on the rotor (i.e., four stators, which gives a step resolution of 90°). This is an extreme step angle, which is not used by any reasonable stepper motors. However, the operation principle remains the same.

Full Step

The full step operation can further be divided into two categories: one-phase full step and two-phase full step. Figure 30 illustrates the step sequence for the full step operation. The arrow represents the rotor pointing in the magnetic vector direction, generated by the energised stators coloured orange. The stators are labelled "A" and "B", identifying the pole pairs.

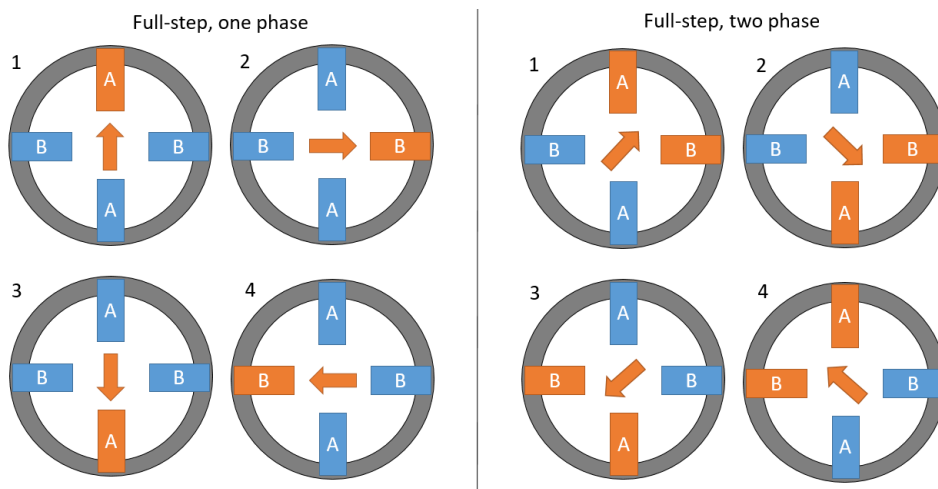


Figure 30. Step sequence for full step one-phase (left) and full step two-phase (right).

One-phase full step energises one pole pair per sequence. This creates a rotating magnetic field, which rotates the rotor in discrete steps at an angle of 0°, 90°, 180° and 270°. By utilising both phases (two-phase full step), the magnetic vector equals the vector sum of both coils (i.e., $\sqrt{2}$ multiplied by the magnitude of one magnetic vector). Thus, the rotating field is offset by an angle of 45° and provides approximately 30% more torque. However, twice as much power is needed resulting in poor efficiency. The full-step requires the least complicated software because only four electrical sequences are needed. [19]

Half Step

Half step operation exploits the magnetic field offset between one-phase and two-phase full step. By sequentially alternating between one phase and two phases, the resolution doubles. The increased resolution yields more control and smoother operation and therefore less mechanical stress. However, the one phase sequences produce less torque than the two-phase sequence. This can potentially create a performance issue, often referred to as torque ripple. If needed, a variable current source can significantly reduce torque ripple, where the current is increased by a factor of $\sqrt{2}$ for each one phase sequence. In addition, compared to the full step mode, the half-step mod requires a more complex software because eight electrical sequences are needed. [19]

Micro Step

In contrast to the other stepping modes, micro-stepping progressively transfer current from one phase to the other phase, thereby reducing the stepper motors' jerky movement. This gradually effect is achieved by inducing a sinusoidal voltage across the phases, rather than using a complementary binary signal which switches the phases entirely on or off. Therefore, micro-stepping significantly increases the step resolution and enhance smother operations. However, micro-stepping increases the drive software complexity. In addition, the torque characteristic will be reduced. [20]

4.9.2 Driver Software Design

Based on the stepper motor requirements, 200 steps per rotation yields a step resolution of 1.8° . Furthermore, the low gear ratio of 1:196 provides an output resolution of $\frac{1.8^\circ}{196} = 0.009^\circ$ per step, indicating that the advantage of micro-step is limited and that both full-step and half-step modes are adequate. However, the full-step operation may be harsher on the gearbox, due to the larger discrete steps. Therefore, the stepper motor will run in half-step mode to enhance smoother transaction between steps, thereby reducing mechanical stress. As described in Section 4.9.1, half-step operation will introduce some torque ripple. Although the driver hardware utilizes a variable current limit, the voltage regulator time constant is too slow and, therefore, not suitable for torque ripple compensation. Considering the stepper motor's torque margined which is more than adequate, the torque loss in one phase sequences are negligible.

Step Control

One electrical cycle consists of eight half steps, where the coil current is on, reverse, or off, as illustrated in Figure 31. The rotor sited in the middle, points in the magnetic vector direction, as maintained by the two current phases (I_1 and I_2 in Figure 31). For each step (marked by numbers), at least one of the current phases changes its direction or turns off. This creates a rotating magnetic field that rotates the rotor. Based on this information, the H-bridge transistor for the RSE driver circuit must be controlled according to the sequence diagram shown in Figure 32.

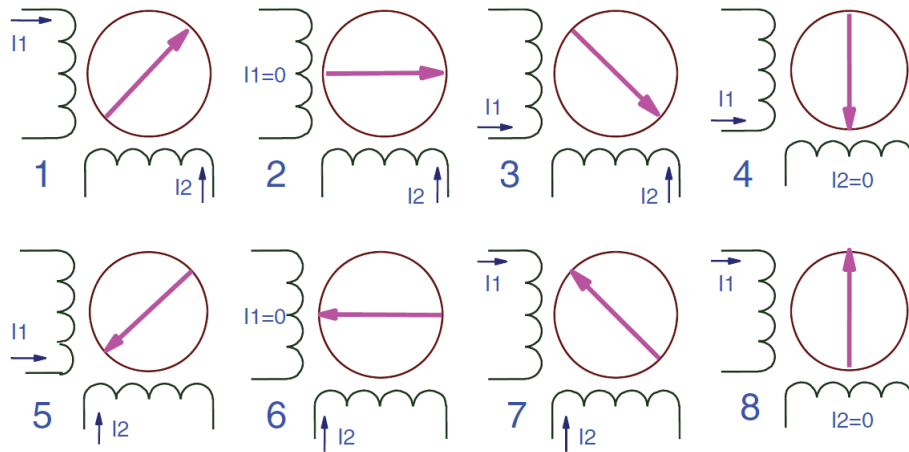


Figure 31. Coil current for half-step mod [22]. I_1 and I_2 represent current in coil 1, and 2, respectively and the sequence is indicated by numbering.

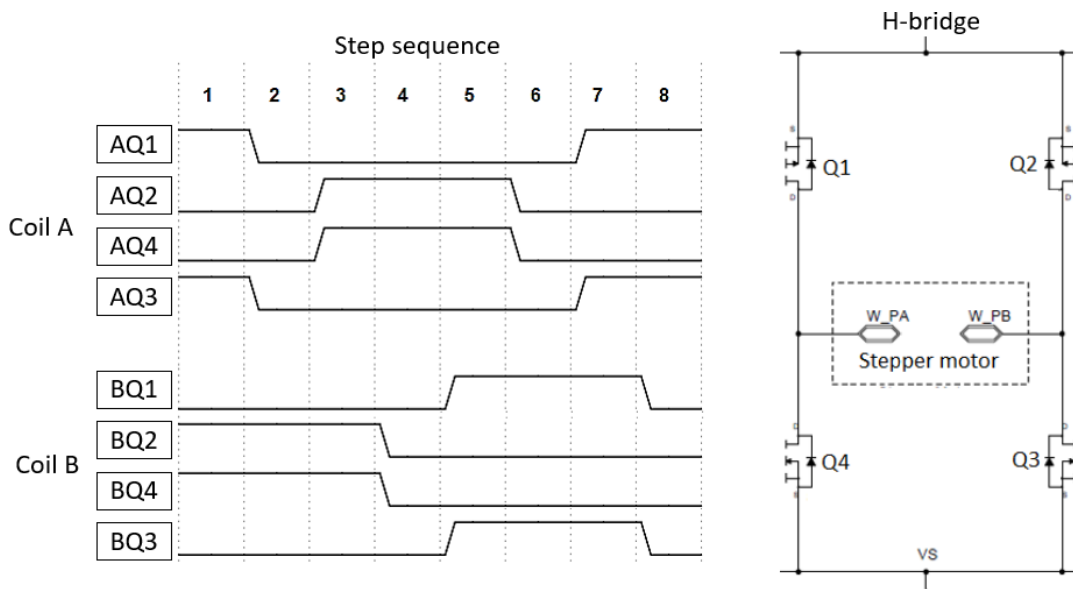


Figure 32. H-bridge steering signals (left) and driver circuit's H-bridge (right). Steering signal high and low indicates that the transistors are on and off, respectively, regardless of transistor type (N-MOS/P-MOS).

By sequentially repeating these eight cycles presented in Figure 32, the motor rotates continuously in one direction. The motor direction is easily changed by swapping the transistor's steering signals for coil A and coil B. Except for the starting sequence, the current is sustained in one direction for 3 out of 8 cycles before turning off, and changing course, hereby denoted as the “motor on period”.

It is crucial that the software exploits the driver circuit's potential fully. Because of this, the “motor on period” is divided into three phases: setup, fully on, and chop. Figure 33 illustrates the hypothetical motor current behaviour for a “motor on period”. This also applies to the starting sequence. However, for one of the coils, depending on motor direction, the “motor on period” lasts for only one cycle.

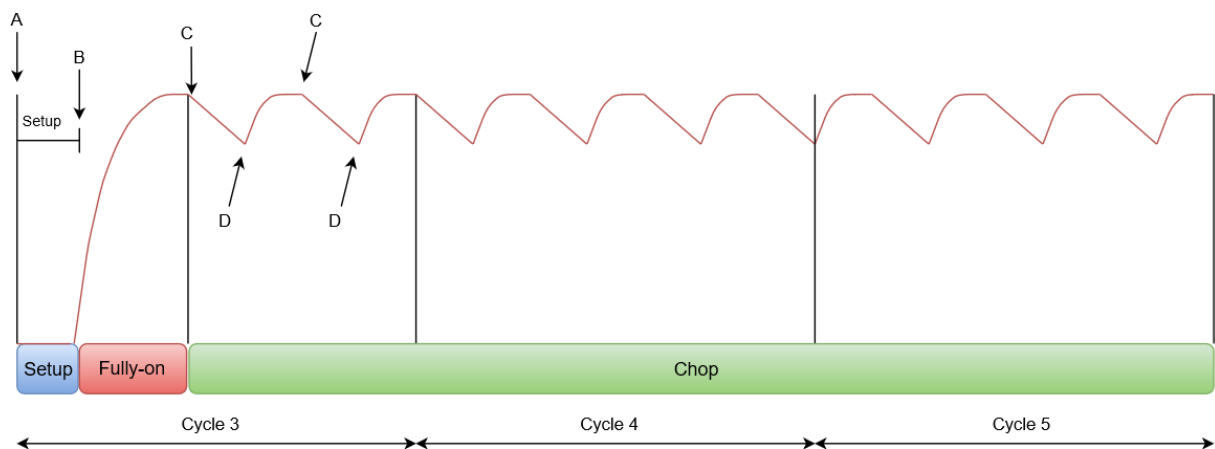


Figure 33. Motor on period. Red line represents coil current for coil A, in step sequence 3, 4, and 5.

The setup phase (Arrow A in Figure 33) configures the transistor states according to the specific cycle and enables/disables timers used by the fully on and chop phase. This is implemented as a simple state machine where the electrical cycle (1-8) is the argument. Because a total of eight transistors are connected to various output ports, and some transistors are active low, the readability can become messy. Consequently, each transistor state is acquired by using the “define” pre-processor directives as shown in Listing 4.

```

1. //----- define transistor states for coil A -----//
2. #define AQ1_OFF (PORTE |= (1<<PINE3)) // transistor AQ1 OFF
3. #define AQ2_OFF (PORTA &= ~(1<<PINA0)) // transistor AQ2 OFF
4. #define AQ3_OFF (PORTE |= (1<<PINE4)) // transistor AQ3 OFF
5. #define AQ4_OFF (PORTA &= ~(1<<PINA1)) // transistor AQ4 OFF

```

Listing 4. Macros for operating H-bridge transistors. *Macros for the other transistor are identical, except for different output ports. These macros are located in the stepper driver header file (stepper_driver.h).*

After the setup phase, pull-down transistor Q3 or Q4 (depending on the step sequence) is entirely on until time C. The fully on duration (time from B to C) is adjustable from 1 to 255 μ s and defined by control register `settling_time`. After reaching C, one of these transistors (Q4 or Q3) is turned off, resulting in a current decay determined by the motor coil inductance, before the appropriate transistor is turned on again at time D. The duty cycle of this chopping procedure is adjustable from 0 to 100% and determined by control register `chop_duty_cycle`. This utilises full control of both decay time and on time. Both the fully on phase and chop phase rely on precise timing and is therefore implemented using hardware timers which are further explained in Section 4.10.2.

4.10 Hardware Modules

The RSE software utilizes five separate hardware modules, where two modules generate interrupts. The various modules and their corresponding interrupt handling routines are listed in Table 17. These modules operate asynchronously to the CPU, meaning that they are separate circuits, where the modules registers are the only interface.

Table 17. RSE HW-modules

HW-module	Interrupt no	ISR
USART 0	19	ISR(USART0_RX_vect)
	21	ISR(USART0_TX_vect)
TIMER 0	-	-
TIMER 1	13	ISR(TIMER1_COMPA_vect)
	14	ISR(TIMER1_COMPB_vect)
TIMER 2	-	-
TIMER 3	-	-

4.10.1 USART 0

This module handles the low-level communication between the RSE and DPU. The UART initialisation routine configures the UART registers: `UBRR0`, `UCSR0C`, and `UCSR0B`.

The `UBRR0` is a 12-bit register used by the USART baud rate generator. The generated baud rate is determined by the clock frequency, the operation mode, and the `UBRR0` register value, as shown in Equation (6).

$$\text{Baud rate [bps]} = \frac{f_{osc}[\text{Hz}]}{8 \cdot (\text{UBRR0}[12\text{bit}] + 1)} \quad (6)$$

The `UBRR0` register can only hold a whole number, meaning that both the clock frequency and baud rate must be chosen relating to the `UBRR0` value. If this is not evaluated, an error given by Equation (7) will be present. The USART module tolerates a small amount of error because the signal sampling occurs in the middle of a bit. However, the baud rate error gives less room for distortions introduced during transmission.

$$\text{Baud rate error}[\%] = \frac{\text{actual baudrate}[\text{bps}]}{\text{desired baudrate}[\text{bps}]} \cdot 100 \quad (7)$$

The E-box modules baud rate has not yet been decided, but approximately *200 000 bps* was suggested. The system currently operates with an internal oscillator measured at 7.6 MHz. Because a baud rate of 237 500 bps provides the nearest error-free baud rate, it was chosen as the baud rate for this thesis. However, when the baud rate is decided, an external crystal can be selected based on the baud rate.

The `UCSR0C` register determines the operation mode and the data frame. This register is configured according to the communication protocol specifications (i.e., asynchronous operation mode, one start-bit, nine data-bit, odd parity and one stop-bit).

The `UCSR0C` register enables the hardware module and the acquired interrupts. Two interrupt lines are enabled: receive complete and transmit complete. A complete data reception will run the ISR (`USART0_RX_vect`), and a complete transmitted frame will run the ISR (`USART0_TX_vect`). For more information about AVR microcontrollers' UART see for example [13].

4.10.2 Timers

Timers are used when precise timing is required. The RSE utilises four timers, operating in Clear on Compare Match (CTC) mode or Fast Pulse Width Modulation (F-PWM) mode, explicitly. Because the ATmegaS128 does not have enough timers, two of the timers handles more than one process.

In CTC mode, the timer's resolution is modified, thereby changing the interrupt interval. This is achieved by manipulating the Output Compare Register (OCR), a register that defines the timer's top value. The timer continuously increments the counter register (TCNT) and resets the moment a compare match between TCNT and OCR occurs. Figure 34 demonstrates the timer's behaviour during different values of OCR. [13]

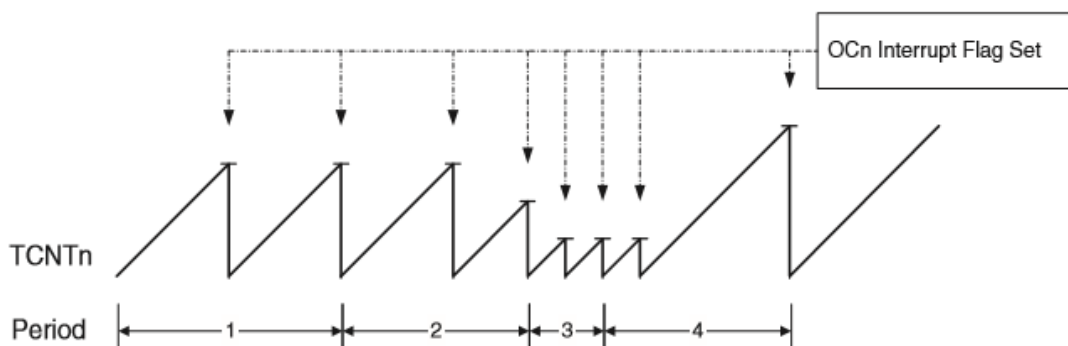


Figure 34. CTC mode, timing diagram [13]. *Adjusting the compare value (TCNT) enables the interrupt to be triggered at a specific time, rather than use the fixed overflow interrupt.*

The F-PWM mode and the CTC mode are relatively similar. However, a timer configured in F-PWM mode utilize direct pin control without using an ISR. This is a significant advantage because the pin is precisely controlled and does not require CPU attention. Figure 35 illustrates the F-PWM operation procedure. The OCR register controls the output compare pin number "n" (OCn). The counter counts from the bottom to top and toggle the pin, immediately, a compare match between TCNT and OCR occurs [13], thereby generating a high-frequency PWM signal (i.e., a signal with adjustable duty cycle).

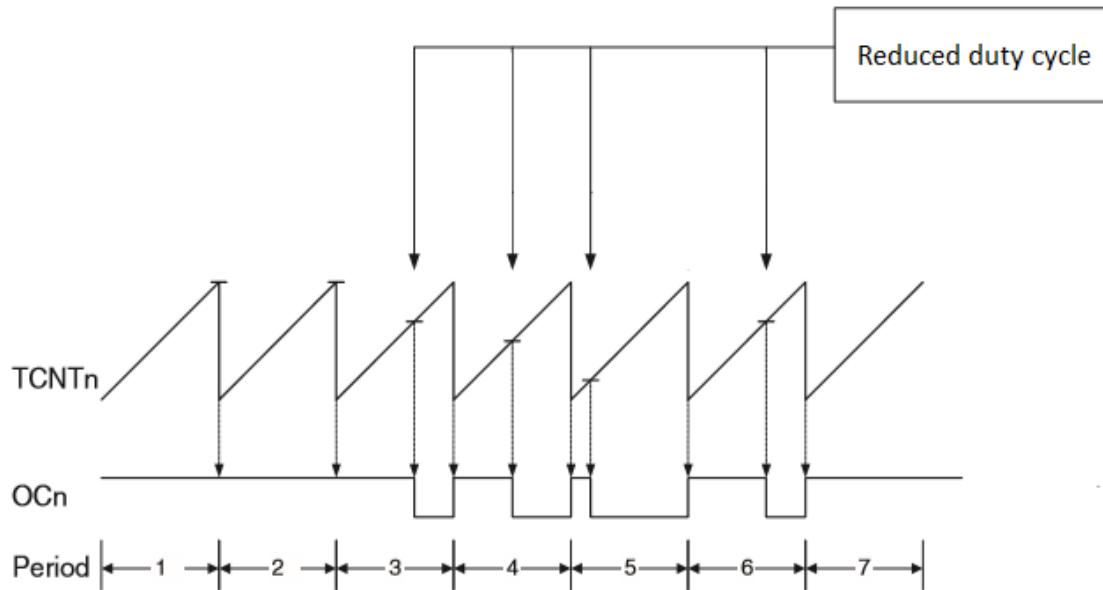


Figure 35. F-PWM mode, timing diagram [13]. *Period 1 and 2, the timers output-compare-value equals the timer's top value, i.e., the output pin (OCn) duty cycle is 100%. For the other periods, the output compare value is reduced, thus the duty cycle decrease.*

Timer0

This timer controls the stepper motor current limit. As described in Section 3.4.3, the current limit is proportional to the duty cycle of a high-frequency input signal. The signal could be controlled by the scheduler, where a task toggles an output port. However, this would be an inefficient and unreliable solution because the task would suffer from timing jitter and frequently require CPU resources. Instead, the procedure is hardware implemented by utilising timer0. Timer0 operates in F-PWM mode and thereby generates a high-frequency PWM signal, without requiring CPU resources. Control register `Motor current` determines the signal's duty cycle, and the base frequency is 32 kHz.

Timer1

Timer1 handles two operations, synchronisation of the scheduler and setting the motor current settling time. The timer operates in CTC mode where two different interrupt lines are accessible. The first interrupt line is in-flight configurable and determined by control register: `Settling-time`. The stepper driver software enables the interrupt line at the start of each “*motor on period*”, as described in Section 4.9.2. The second interrupt line controls the scheduler interval time. Because the scheduler maintains the program timing and several other operations uses the interval time as a reference time, the interrupt line is fixed. This means that the DPU cannot change the scheduler interval time; however, it is possible during patching.

Timer2 and Timer3

The chopping procedure requires a precise and high-frequency signal that performs fast switching of the H-bridge pull-down transistors. Consequently, this procedure is controlled by timer2 and timer3 which both runs in F-PWM mode. Timer3 handles both transistors for coil A and one transistor for coil B; Timer2 controls the second transistor for coil B. The PWM signal's base frequency was determined based on the motor time constant. A low frequency would allow current to exceed the current limiting circuit's threshold, resulting in severe power dissipation. On the other hand, an excessively high frequency would significantly increase the transistor's switching power consumption. A base frequency of 32 kHz is a decent compromise and was for this reason selected. The driver software controls both these timers, where control register `chop_duty_cycle` adjusts the PWM signal's duty cycle.

5 Test and Development

A system that simulates the DPU must exist to command and receive data from the RSE. The optimal solution would be to use the authentic DPU. Unfortunately, a DPU prototype was not available; the DPU was therefore simulated by using a computer and the program Real-Term. Real-Term is an open source terminal emulator that enables serial communication through the computer's COM ports. The emulator is fully configurable where a variety of different baud rates and data frames can be selected. In addition to the emulator, a USB-to-UART converter was connected between the RSE and the computer. This step was necessary because the computer's USB port generates 5 V signals while the RSE requires 3.3 V signals. This setup worked nicely at the beginning of the tests, where only a command or two was tested. However, as the RSE software evolved, several "memory read" and "memory write" commands were required to cover all the RSE functions. As described in Section 2.4.2, a "memory read" and "memory write" transaction requires three and five separate queries, respectively. Because the emulator did not support multiple transmissions, the query's payload must be manually arranged for each transmission, a time-consuming and inefficient method. Consequently, it was concluded that development of a custom terminal emulator, simulating the DPU would be faster in the long-term.

5.1 RSE Modifications for Testing

The RoR protocol defines a UART transmission to contain 9 data bits, which represents one character. This is an unusual configuration that the computer's COM ports do not support (8 data bits or less). Constructing an additional microcontroller that simulates the DPU would be significantly more time consuming and not as flexible as a program on a computer. Therefore, it was decided to remove the MSB from the UART configuration and instead, insert it at the beginning of the RSE communication task, as shown in Listing 5.

```

1. ISR(USART0_RX_vect){
2.     //----- RBDP character level begin -----//
3.     uint8_t status = UCSRA; // Read UART status register
4.     uint16_t raw_data;
5.     // if bit-error, discard the data
6.     if ( (status) & ((1<<UPE0)) )
7.     {
8.         uint16_t received_data_error=UDR0; // Don't use data, i.e. garbage
9.     }
10.    else // Error free data
11.    {
12.        raw_data = ((0<<8) | UDR0); // Manually insert the MSB and read UART data bit[7-0]
13.        //----- RBDP character level end -----//
14.        packet_level(raw_data); //Continue to packet level with a whole character (9bit)
15.    }
16. }

```

Listing 5. RSE communication task. *The MSB which indicates query or response packet is manually inserted in the code, enabling 8-data bit UART configuration. (Located in main.c)*

Listing 5 illustrates the communication task ISR, where the character’s MSB is set to ‘0’ (marked blue), indicating a Query and the remaining 8 data bits are provided by the UART. This allowed testing of both queries and response characters, which means it could simulate, other slaves and the DPU. However, the MSB must be manually changed in the code. Note that the UART control register C (UCSR0C) which defines the data bit size, must also be changed (i.e., to 8 data bits).

5.2 DPU Simulator

The DPU simulator is created in Python, which is a high-level, general-purpose programming language. Python has simpler syntax compared to the alternative programming languages: C, C++, and Java. It also supports a large variety of libraries and development tools, such as serial communication and Graphical User Interface (GUI). This enables complex programmes to be created in a relatively short time, so it was decided to use Python as the framework for constructing the DPU simulator. [29]

Because this simulator also is intended to be used for further development of the radiation shutter, the programme needs to be user-friendly and robust. Therefore, the programme is operated through a GUI, as shown in Figure 36. The GUI is organised in three separate windows: RSE register window, RSE command window, and RSE terminal.

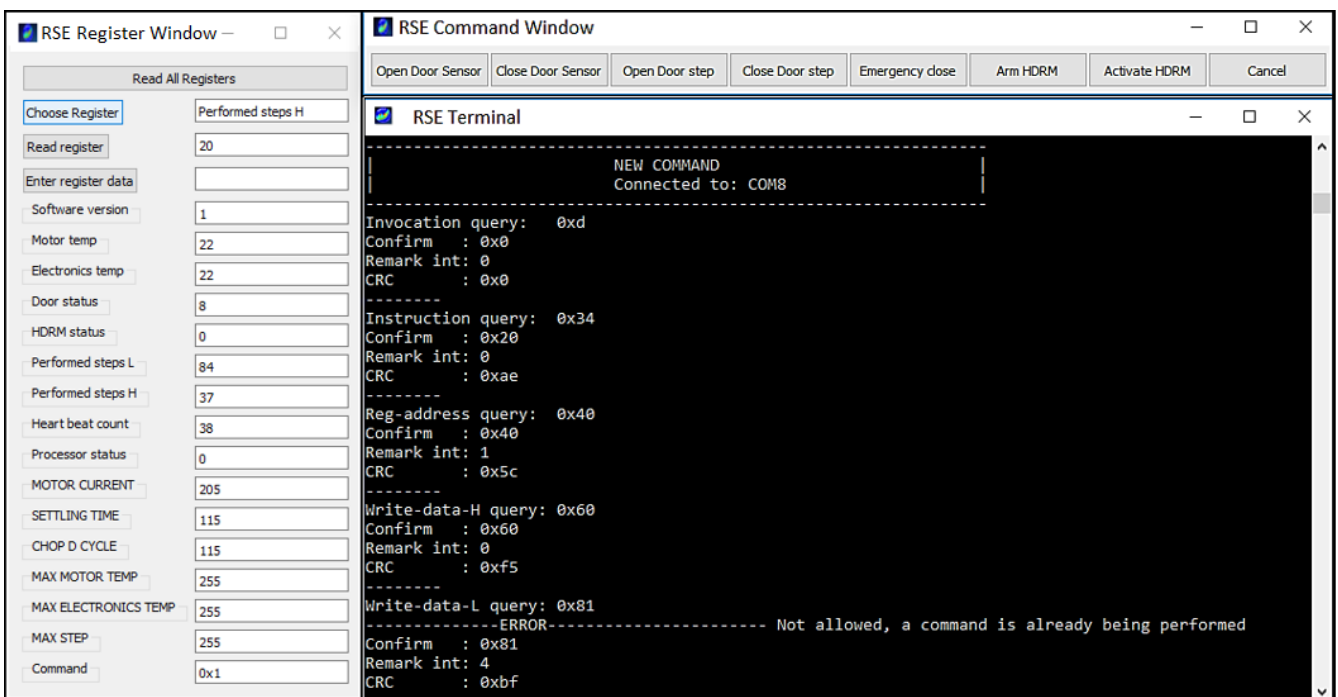


Figure 36. DPU simulator for operating and test the RSE. *From this program, all RSE functionalities can be acquired (read/write operations) thereby configure RSE settings, collect HK-readouts and activate motor/HDRM. In addition, the transactions can be observed from the RSE terminal.*

RSE Register Window

As the name implies, this window handles the RSE registers. The registers are organised as a list where the register content is displayed in a status bar, parallel to the corresponding register. Control registers are separated from status registers and the command register by their name written in capital letters. A memory read operation is achieved by selecting the acquired register from the “Choose Register” drop-down menu and clicking the “Read register” button. The program will automatically broadcast the required queries, interpret the received response packets, and place the register content in the correct register status bar. Because the RSE test procedure often required reading of multiple registers, an extra functionality was added. The functionality enables to read all registers by activating the “read all registers” button, and thus, significantly reduced the time of testing. The memory write procedure is identical to that of a memory read operation, except that the register content must be written in the “enter register data” menu.

RSE Command Window

As discussed in Section 4.6, a command is initiated by writing a specific value into the command register. Hence, a command can be started via the RSE register window. However, the command was frequently used and therefore required a more efficient solution. Consequently, the command window was introduced, which works as a shortcut for a pre-coded memory write operation. This window gave the ability to start commands through the buttons instead of the cumbersome, manual register write operations.

RSE Terminal

Before the system behaviour could be confirmed, the transactions between the DPU and RSE must be correct (i.e., follow the RoR communication protocol). This imposes the need for a predictable and organised solution where the transactions can be observed. To fill this need, the RSE terminal was created. The terminal serves as a monitor where a sorting algorithm translates the transactions and displays the data in a structured form. Because this type of sorting algorithm already was developed for the RSE, it was relatively easy to convert it into Python code. The queries are identified by their name, followed by their corresponding response packets, which contain three characters as specified by the RoR communication protocol, described in Section 2.4.1. In addition, a rejected query is identified by an error message. For example, in Figure 36, two open shutter commands have been started, which is an illegal operation according to the RSE applications level (see Section 4.7.4). The RSE terminal signals that an error occurred at the last query and identifies the cause of rejection.

5.3 Driver Test

The motor and the drive are two critical elements in the radiation shutter system. Therefore, it is crucial that both driver software and driver hardware collaborate so that the motor is accurately controlled, provides the required torque, and operates efficiently. In addition, the magnetic field generated by the motor needed to be measured and confirmed that it does not affect other instruments on the SMILE satellite. These requirements impose the need for a comprehensive test setup, which covers the following:

- Step sequence
- Driver parameters (chop duty cycle, V_{ctrl} duty-cycle, and settling-time)
- Motor torque and efficiency
- Motor magnetic field

5.3.1 Step-Sequence Test

Software Test

As discussed in Section 4.9.2, the step-operation is implemented as a state machine where the step sequence is the argument. The state machine was therefore examined and verified by using Atmel Studio's OCD tool. This enabled to single step through each state and observe the output ports register.

Hardware Test

The eight RSE output ports that operate the driver's H-bridge was connected to a logical-analyser. This provided the information required to verify that the signals were correct for each step sequence, as shown in Figure 37. Due to the H-bridge operation principle, a wrong stimulated transistor not only affects the motor behaviour but also may cause a short circuit. The step sequence was, therefore, systematically tested before the output ports were connected to the driver circuit. The test then proceeded to the circuit test. An oscilloscope measured the gate voltage on the H-bridge transistors, and the voltage was verified to ensure sufficiency to fully activate and deactivate the transistors.

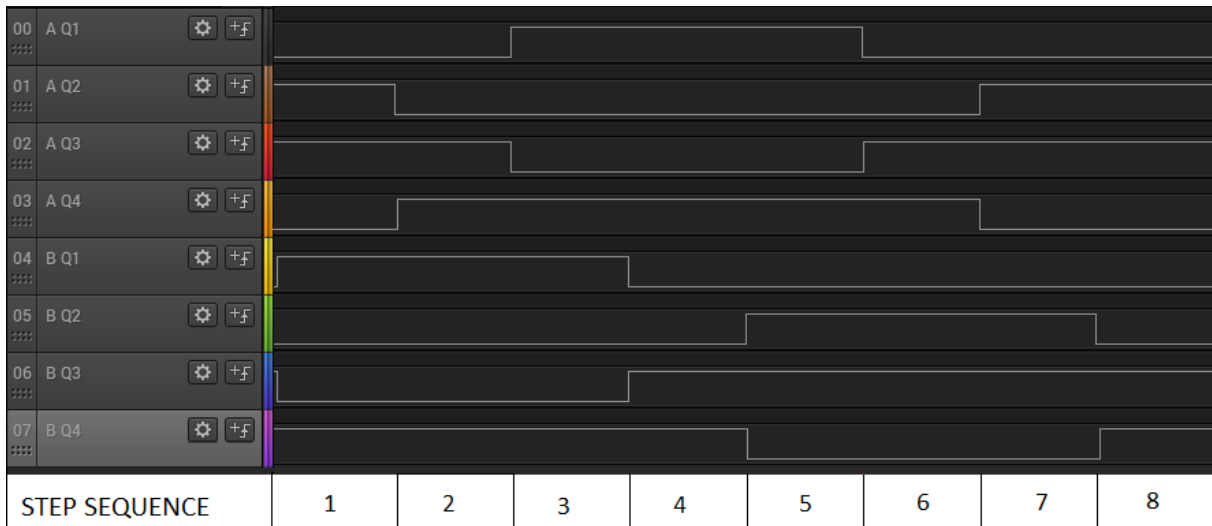


Figure 37. Step sequence sample. The transistor states are labelled according to the driver circuit diagram, where A and B identify the coil pairs. The numbering identifies the sequence.

5.3.2 Driver Parameters Test

Due to the construction of the motor driver, three parameters are adjustable: the duty cycle of V_{ctrl} signal, the duty cycle of the chop signal, and the settling time. Because hardware timers control these parameters, it is difficult to inspect the procedure through software analysis. Therefore, the functionality of these parameters was primarily hardware tested where the output signals were inspected by using a logical analyser.

V_{ctrl} Test

The functionality of the V_{ctrl} signal is much simpler than those of chop and settling time. It only handles one port and is not affected by the step sequence. Therefore, the behaviour was confirmed by incrementing control register “motor current” and inspecting the signal’s duty cycle.

Chop Test

The chop procedure was tested similarly to the step sequence. The eight RSE output ports that operate the driver’s H-bridge were sampled with different values of control register `chop_duty_cycle`, where one sample is shown in Figure 38. The test confirmed that the chopping procedure followed the correct sequence and that the register value controlled the duty cycle. However, when a chopping phase for one coil started, the ongoing pulse for the other coil was longer than the others, as highlighted in Figure 38. This is because Timer2 controls signals for both coil A and B. The timer did not permit, introducing of new signals in the middle of a count operation so the timer must complete the ongoing count or manually be reset.

Waiting for the timer to complete, would mean that the chopping procedure for the passive coil starts too late. This results in additional power dissipation because the current exceeds the current limit. The other alternative, manually resetting the timer, would introduce a longer pulse because the timer is reset before the pulse is set low. However, this would only mean that the current would decrease slightly more for that particular pulse.

Because the current rise time is significantly quicker than current decay time, as described in Section 3.4.2, the effect would be more noticeable if chopping starts too late. Therefore, the driver SW operates by manually resetting the timer at the start of each chopping phase.

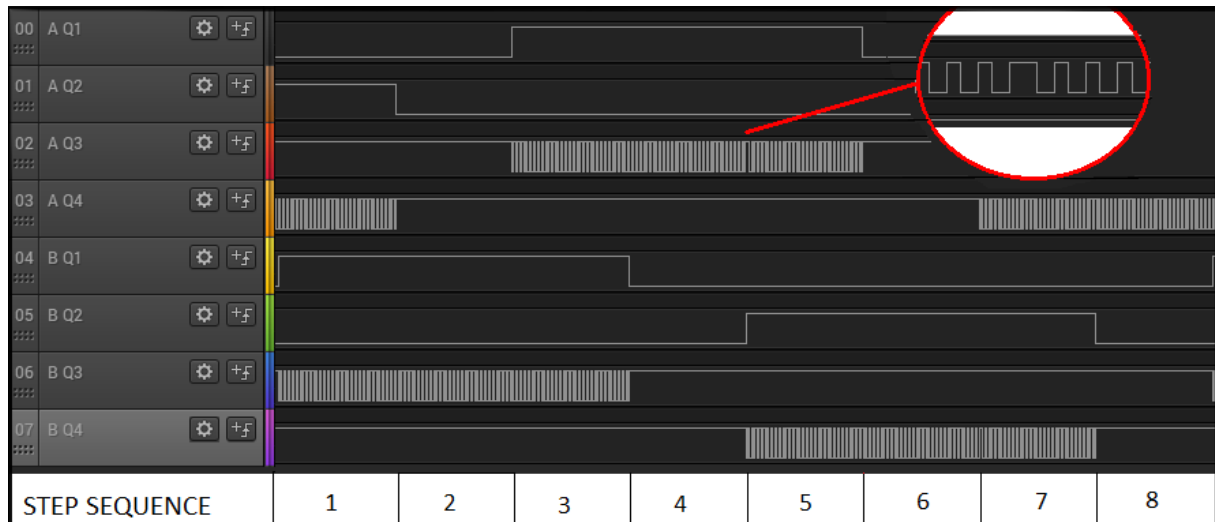


Figure 38. Chop sample. The figure illustrates the chopping procedure for an eight-step sequence where the steering signal for the pull-down transistors (Q3 and Q4) are switched fast on and off. The figure also indicates (red) the undesirable extended pulse, introduced by the timer.

Settling Time Test

This parameter determines how long the current is allowed to rise unaffected by the chopping procedure. This means that there must be a delay before the chopping starts, which is called the “fully on phase”. The settling time test procedure was identical to that of the chop producer, except that the control register `settling_time` was incremented. Figure 39 illustrates one sample of four steps where settling time is 250 μ s.

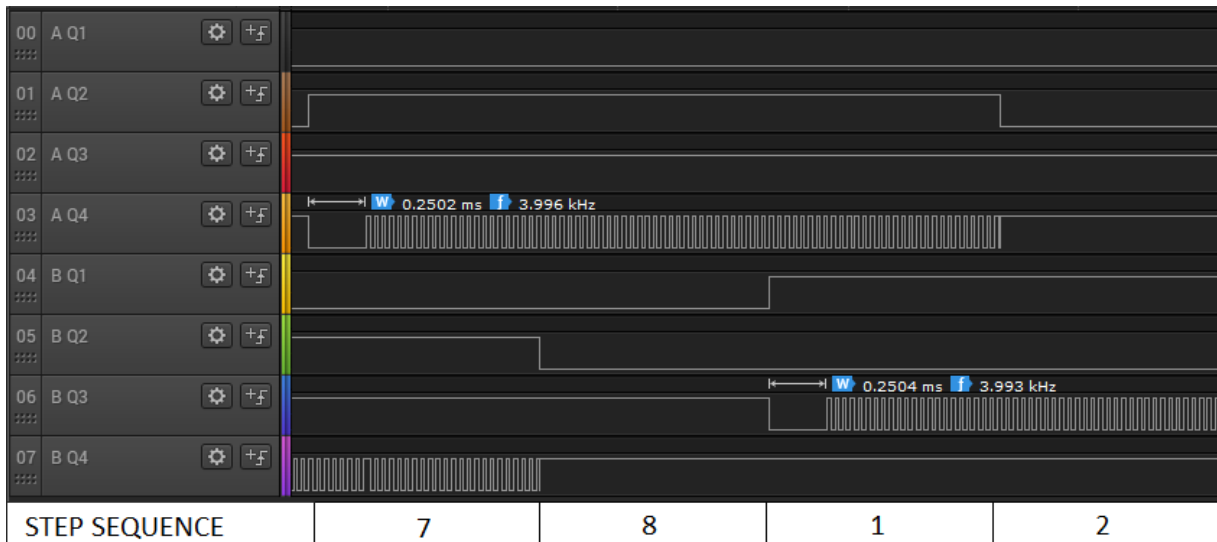


Figure 39. Settling time sample for 4 out of 8 step sequences. *Settling time register is adjusted to 250 μ s. Therefore, the chopping (fast on/off switching) procedure for pull-down transistor AQ4 and BQ3 starts after 250 μ s.*

Current Waveform Test

The overall functionality was verified by connecting the stepper motor to the driver and observing the motor current. As discussed in Section 4.9.2, a “motor on period” is divided into three phases: setup, fully on, and chop. Figure 40 shows one “motor on period” for coil A where the driver is configured by the following settings:

- V_{ctrl} duty cycle: 80% (current limit: 0.6 A)
- Settling time: 250 μ s
- Chop duty cycle: 45%

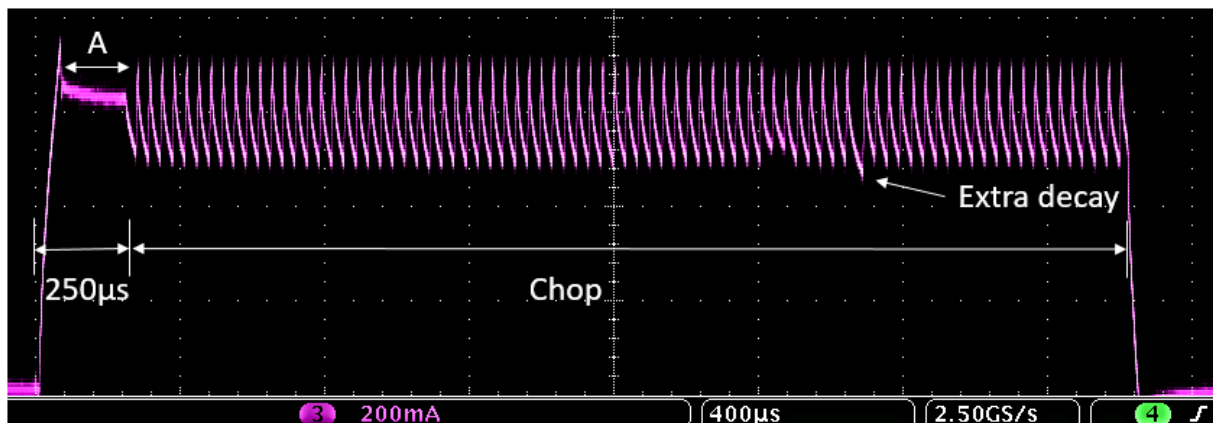


Figure 40. “Motor on period” for coil A. *Purple line represents the motor current.*

Figure 40 illustrates that the current change direction and increase until the current limit is reached. We see that the current is sustained (A in Figure 40) until the chopping procedure starts and holds the current at approximately 0.6 A. As previously discussed in Section 3.4.2, the most efficient setting is when the chopping procedures start as soon as the limit is reached. This enables current to rise quickly, but without the excessive power dissipated by the current limit circuitry, as indicated for period A in Figure 40. Therefore, a settling time of approximately 120 μs would be a more suitable setting for this particular test. Figure 40 also illustrates the undesirable current decay, introduced by the timer. However, this effect is hardly recognizable and, therefore, would not affect the torque characteristics.

System Test

The driver's parameters affect motor torque characteristics and power consumption. Therefore, it is vital to characterise the driver parameters effects and operation range, as well as to find the optimum settings for the driver software. A complete test which covers these demands is located in Appendix B, and a summary of the test is presented in this section.

The current limit worked as expected: both motor current and motor torque increased linearly as the V_{ctrl} duty cycle increased. However, the driver was non-functional below 25% duty cycle, which corresponds to a V_{ctrl} signal of 0.8 V and a current limit of 0.16 A, because the current limit circuit requires a V_{ctrl} signal above 0.8 V. The test confirmed that a settling time less than the current rise-time reduced the torque. Although a settling time longer than the current needed to reach the current limit did not affect the torque, the current draw increased considerably.

The chopping operation had a significant impact on both torque and current draw. However, a chop duty cycle of approximately 45% reduced the average current draw by 70% and only decreased the torque by roughly 5% from the maximum value. These results confirm that the chopping operation significantly increased the efficiency. The torque is fully adjustable from 0.1 Nm to 1.63 Nm and provides the required torque of 1.2 Nm at a current limit of 0.6 A. Table 18 shows the parameters operation range and the ideal parameter settings at normal operation.

Table 18. Driver parameters test result

Parameter	Optimal setting (0.6 A)	Operation range
V_{ctrl} duty cycle	80%	25–100% (0.16–0.8 A)
Chop duty cycle	45%	0%–55%
Settling time	120 μs	1–255 μs

5.4 Task Execution Test

Communication Timing Test

As discussed in Section 2.4.3, the RoR communication protocol defines three timing constraints. Therefore, it is crucial to ensure that the RSE does not violate any of these constraints. The transmission speed between the E-box cards was suggested to be around 200 kbs. Thus 1 baud delay corresponds to $5\ \mu\text{s}$, meaning that the RSE must generate a response message between $5\ \mu\text{s}$ (slave slack) and $120\ \mu\text{s}$ (response timeout) after a received query. The RoR protocol defines five different queries that contain separate instructions, so the delay before the RSE generates a response packet will vary. To ensure that the RSE operates within the timing constraints, the transactions were analysed by an oscilloscope. The scope sampled several transactions and generated a combined image, as shown in Figure 41. All samples are stacked on top of each other; together, the samples will cover an area that represents the time variation of the transactions. Figure 41 shows that after a received query, the RSE waits for at least $38\ \mu\text{s}$ and a maximum of $68\ \mu\text{s}$ before a response is generated. This confirms that the RSE operates within timing constraints and at a safe value away from the minimum and maximum limits.

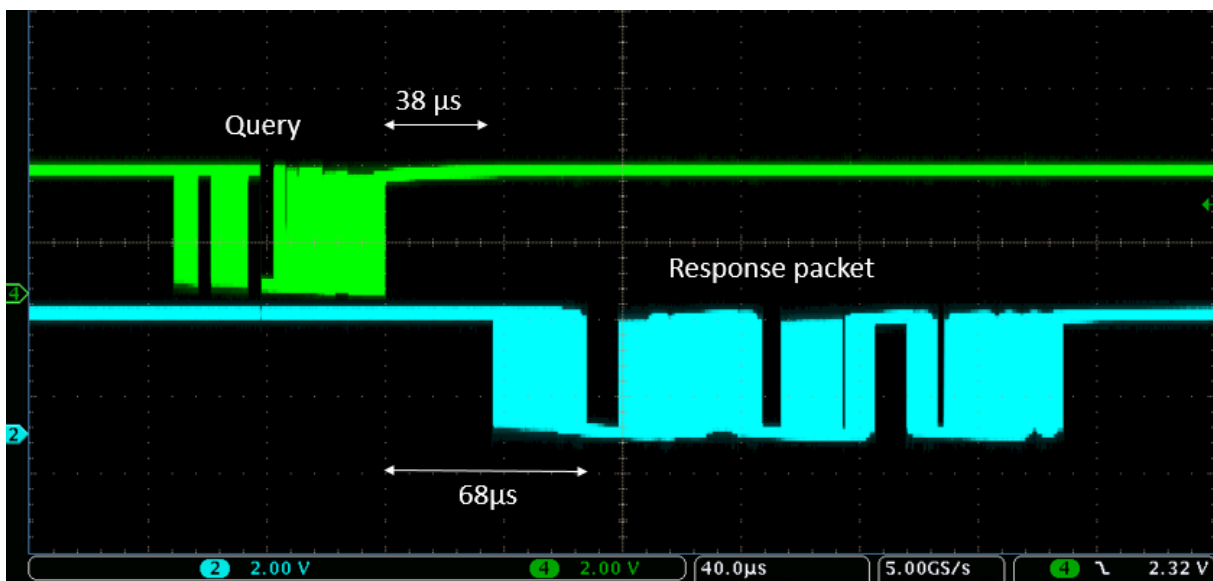


Figure 41. Transactions envelop samples. *Query packet (on top) contains one character, and the response packet (below) contains three characters. A query is answered by a response packet between $38\ \mu\text{s}$ and $68\ \mu\text{s}$.*

Scheduler Tasks Test

As discussed in Section 4.4, the scheduler requires that all tasks started within a cycle must complete before the cycle ends. One cycle lasts for two ms, which should be more than enough because the tasks perform relatively simple operations. However, the communication task is implemented as two ISRs, one for receiving data and one for transmitting data. Because the CPU only has one core, the tasks controlled by the scheduler can be affected, especially if the interrupt occurs frequently. By measuring the tasks' worst-case execution time (WCET), it is possible to calculate the WCET of a cycle and confirm that it is less than two ms.

The scheduler executes two task categories, periodic tasks, and sporadic tasks. All periodic tasks are permitted to execute once per cycle, and only one sporadic task is allowed. Therefore, the WCET for one cycle can be calculated by means of Equation (8).

$$WCET_{Cycle} = WCET_{Periodic} + WCET_{Sporadic} + WCET_{Rx} \cdot a + WCET_{Tx} \cdot b \quad (8)$$

- $WCET_{Periodic}$: Execution time of all periodic tasks
- $WCET_{Sporadic}$: Execution time of the most time-consuming sporadic task
- $WCET_{Rx}$: Execution time of the receive ISR
- $WCET_{Tx}$: Execution time of the transmit ISR
- a : Maximum amount of receive ISR, before all scheduler tasks, are complete
- b : Maximum amount of transmit ISR, before all scheduler tasks are complete

The tasks execution time was established by toggling an output port before and after each task and measuring the pulse length. The results are listed in Table 19.

Table 19. WCET results

Task	WCET
$WCET_{Fixed}$	32 μ s
$WCET_{Command}$	24 μ s
$WCET_{Rx}$	68 μ s
$WCET_{Tx}$	7 μ s

Figure 42 illustrates the CPU resources, assuming the DPU sends queries at the maximum rate. The light-green boxes represent available CPU resource, and red boxes represent CPU occupied by an ISR. Once a query is received, the communication task ISR is entered. As previously described, the communication task interprets the data and buffers the first response character into the UART transmit register. The UART operates asynchronously to the CPU; this means that while the UART transmits the data, the CPU is available for 60 μ s, as defined by Equation (9).

$$Character\ transmission\ time = 12\ bit \cdot 5\mu s = 60\mu s \quad (9)$$

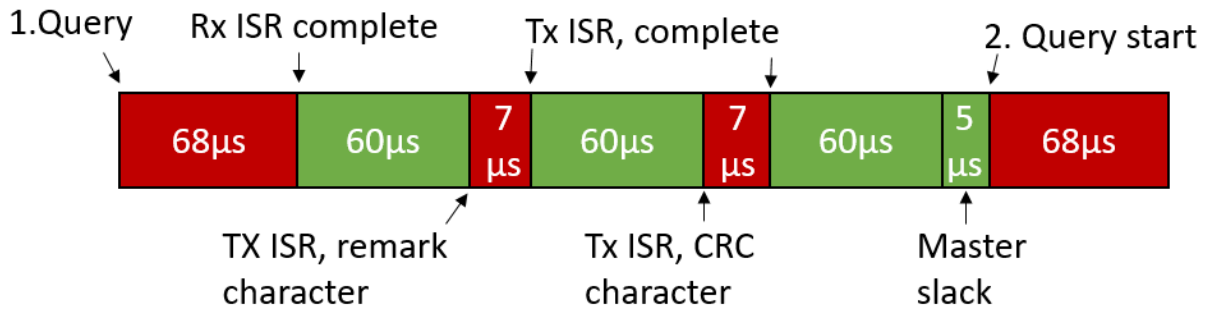


Figure 42. WCET CPU illustration. Dark red boxes represent CPU occupied by an ISR. Light green boxes represent available CPU.

Once the first character is transmitted, the transmit ISR is entered, and the next character is buffered into the UART transmit register. This is a short ISR and only occupies the CPU for 7µs. After the second character is transmitted, a new transmit ISR is entered, and the last character is buffered. The CPU is then available for 60 µs plus the master slack, which is one of the RoR timing constraints. This operation repeats until the cycle ends.

$$ISR_{Time} = \frac{CPU_{Occupied}}{CPU_{Available} + CPU_{Occupied}} = \frac{82\mu s}{185\mu s + 82\mu s} \cdot 100 \approx 30\% \quad (10)$$

From Equation (10), the CPU is at worst occupied by an ISR for 30% of the cycle, which means that the tasks have 1.4 ms available CPU time each cycle. Considering the short task execution time $WCET_{Periodic} + WCET_{Sporadic} = 56 \mu s$, the available CPU resource is more than adequate. Because the execution of the scheduler tasks adds up to 56 µs which are less than one character transmission time, the WCET for a cycle according to Equation (8) is $68 \mu s + 56 \mu s = 124 \mu s$.

To confirm that the calculations were correct, a test similar to the communication test was performed. An output port was toggled at the beginning of the first task and at the end of the last task. In addition, an output port was toggled by the scheduler timer. This setup provides two signals where one represents the execution time of all tasks, and the other represents the scheduler cycle. The oscilloscope repetitively sampled the output signal while several memory read and memory write operations were initiated, as fast as possible, ensuring that an interrupt occurred before the tasks were complete. Figure 43 illustrates the generated envelope image where the upper signal represents the scheduler time, and the lower signal represents the total task execution time.

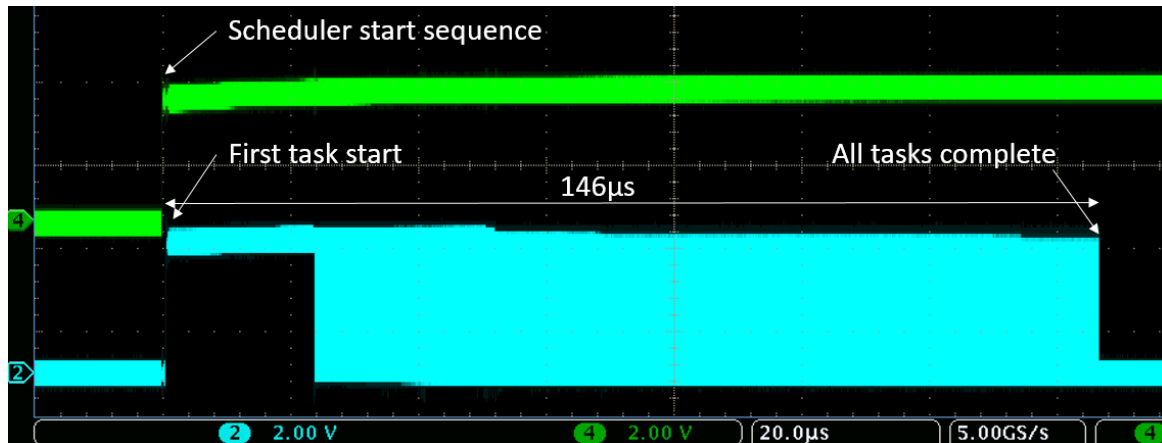


Figure 43. Scheduler tasks envelope sample. *The pulse on top represents one scheduler cycle (lasts for two ms) and the pulse below represents the WCET of all tasks in a cycle (146 μ s).*

Figure 43 illustrates that all tasks complete after 146 μ s, which is slightly more than the calculated. Considering the rough estimate method, the two results are relatively similar and confirm that the scheduler guarantee execution off all tasks well within the limit. Figure 43 also shows that the first task is not affected by timing jitter, denoted by how the rising edge of the pulse does not cover an area, meaning it is started at the same time each cycle. The first task is only affected by timing jitter if an interrupt occurs at that particular moment, which is unlikely.

5.5 Motor Magnetic Field Test

The SMILE satellite contains several components that may be affected by the magnetic field generated by the stepper motor. In particular, the magnetometer instrument which is designed for measurements of the Earth's magnetic field. Even though the stepper motor will be off for most of the time and only generate a rotating magnetic field for a short period, the stepper motor's permanent magnetized rotor will produce a static magnetic field. However, by characterising the magnetic field during both these events, the magnetometer's readings can be compensated.

Consequently, the motors magnetic field in the x, y, and z-axis was measured from distances between 5 and 25 cm. In addition, these measurements were repeated with mu-metal wrapped around the motor. Mu-metal is a soft ferromagnetic alloy of about 80% nickel, and the remaining 20% consists of iron and various other materials [30]. This composition has an ability to reduces magnetic field and will most likely encapsulate the motor to prevent magnetic field contamination.

Figure 44 shows two magnetic field measurements (x-axis) when the motor is active, with and without the mu-metal. These graphs confirm that the mu-metal significantly reduces the magnetic field exposure, especial at lower distances. The other measurements that are not included in this thesis showed similar results. In addition, the magnetic field measurements that were performed while the motor was off and on, was almost identical. For more information about the magnetic field measurements, see [31]

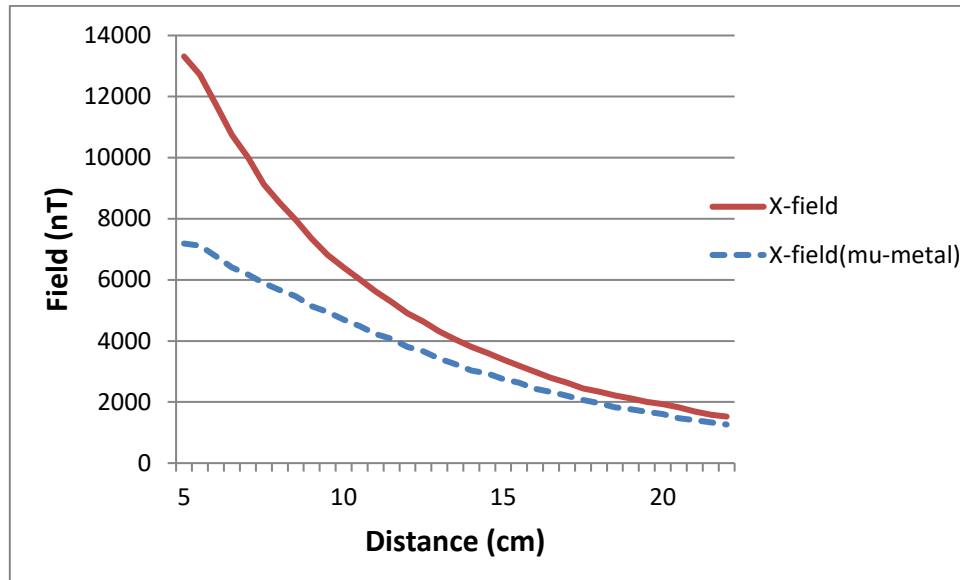


Figure 44. Stepper motor magnetic field measurement [31].

5.6 Test Evaluation and Key Findings

A user-friendly and efficient simulator program has been developed. This allowed to test the RSE functionalities and monitor the communication. It was also beneficial during motor torque and magnetic field measurements since the program provided a simple interface for operating the RSE. The program did not support the correct UART configuration. Consequently, the RSE required smaller modifications. However, the modifications did not affect the RSE operations and therefore did not compromise the tests integrity.

5.6.1 Driver and Motor Evaluation

The driver's key functionalities were analysed and tested separately. The test confirmed that H-bridge transistors were sufficient controlled and that a short circuit did not occur. The driver parameters worked as expected, the output signal was determined by their corresponding control registers. In addition, the settling time and chop procedure where activate at the correct sequence. However, the chop procedure which should provide a stable PWM signal had one extend pulse, each fully on period. The current waveform test showed that the extended pulse had a negligible effect on the current waveform. Hence, it will not affect the motor characteristics.

The system test showed that both driver hardware and software were sufficient to control the stepper motor efficiently. The test also characterized the parameters effect on motor torque and current draw. This provides the information required to find the optimal driver setting, which gives the highest torque without compromising the efficiency. In addition, the system test provides key findings which can increase the motor's performance and reliability. For example, by decreasing the settling-time parameter (i.e., chopping starts earlier), the current rise time will be longer, as shown in Figure 45. Consequently, each step will have slower acceleration and therefore not be as harsh on the gearbox.

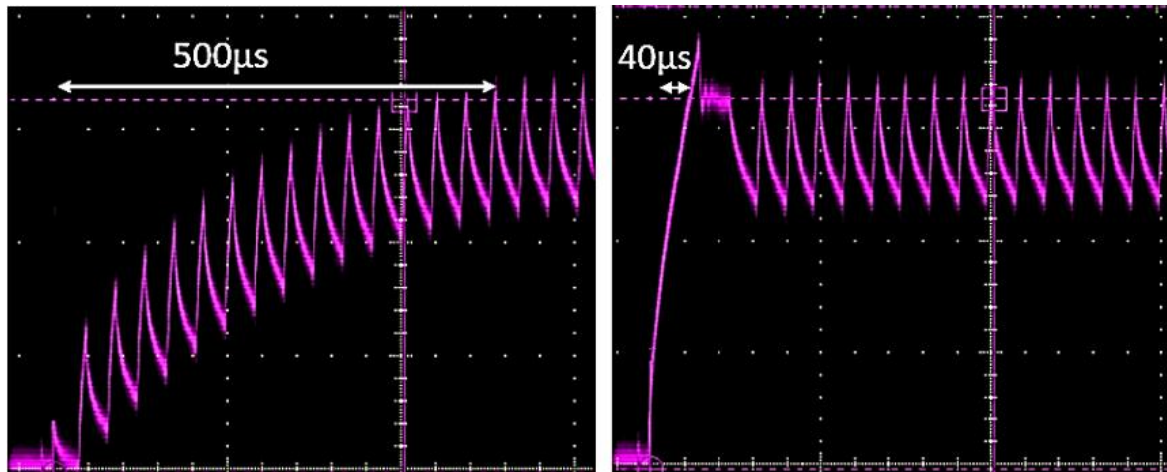


Figure 45. Driver settling time parameter effect. *The two images show the coil current at the start of a “motor on period”, i.e., the start of a step. Left: the current chopping starts immediately, resulting in long current rise time. Right: the current chopping starts after $\approx 100 \mu\text{s}$, resulting in shorter current rise time.*

Furthermore, the current limit can also be reduced, but this will not affect current rise time, only the current final value. However, if an operation requires higher torque than normal, e.g., in case of the shutter is stuck, can the driver parameters be adjusted accordingly:

- Current limit set to maximum (0.8 A)
- Settling time over 115 μs
- Chop duty cycle 45% or less

This will provide approximately 25% more torque than normal operations. However, the current draw and power consumption will increase significantly, especially if the chop duty cycle is less than 10%. Therefore, this setting must only be used for a short period, to prevent the motor and driver from overheating.

5.6.2 Tasks Handling Evaluation

The RSE control system is constructed as a hybrid of hard and soft real-time system. The task testes confirmed that system guaranteed execution of all task well within the deadline. The communication task which is the only hard real-time task could potentially affect the soft real-time tasks. This is because the communicating task is implemented as an ISR, rather than a pre-emptive task within an RTOS. The ISR could therefore frequently acquire CPU resources and prevent the scheduler from completing the soft-real-time operations. However, calculations showed that the scheduler had at least 70% available execution time for each cycle. Considering the relatively long scheduler interval time of 2 ms and the short WCET for a cycle, of 124 μ s, the CPU has plenty of available time. This was also confirmed with hardware measurements, where the communication task was continuously stimulated, and the task execution time was measured. However, the HW test showed that the scheduler task used about 15% more time than the calculated, which corresponds to only 22 μ s difference. Considering that calculated method did not account for the task swapping overhead, ISR response time and the modified UART frame, the results are almost identical.

6 Outlook and Conclusion

6.1 Future Work

Due to the projects early stage, some components and software requirement specifications (SRS) are not yet finalised (e.g., HDRM actuator, sensors and the periodic measurements' interval). However, the software is designed highly generic and modular, enabling simple software adjustments, adoption of new hardware components and software functionality. Therefore, all tasks are independent procedures. Those tasks that are not completed due to the projects early stage, will read dummy variables (simulating sensors) or activate dummy output ports (simulating actuators).

Because the RSE must be able to operate unattended during several years, the program must be prevented from a system halt. A feature that prevents this scenario is not yet implemented. Therefore, it is strongly suggested that for example, a Watchdog Timer (WDT) is implemented in the future. A WDT operates identically to a hardware timer, but rather than count in a loop; the WDT manually resets the system when a set/specific value is reached. For example, by implementing a task in the periodic dispatcher that zeroes the timer, a system reset will not occur. However, if the system is at a halt, the zeroing will not happen. Consequently, the system will be reinitialised by the watchdog and continue as normal.

6.2 FPGA Implementation

At the time of writing this thesis, requirements regarding components' radiation tolerance became more defined, as well as stricter. Therefore, the ATmegaS128 microcontroller's radiation tolerance is in the lower region within these requirements. Consequently, it has been suggested that a radiation hardened FPGA solution would be a safer alternative.

Because FPGAs and microcontrollers are two different platforms, the RSE implementation provided by this thesis will require some re-design. However, the system requirements will not change. In addition, most of the SRS and software design can easily be converted into operating on a hardware platform such as an FPGA. This is because the RSE software is highly modular, controlled through registers, and most of the execution is done in state machines (SM). Furthermore, most of the output signals steering the motor are already directly controlled by hardware timers. These design choices are very close to a hardware approach, and will enable a quick transition to an FPGA if required.

Figure 46 offers a simplified block diagram of a potential FPGA solution for the RSE. The individual blocks illustrate the main modules and the arrows illustrate their interconnections. Instead of using a scheduler, the registers can directly acquire and configure timers and state machines, thereby executing both periodic and sporadic tasks in hardware.

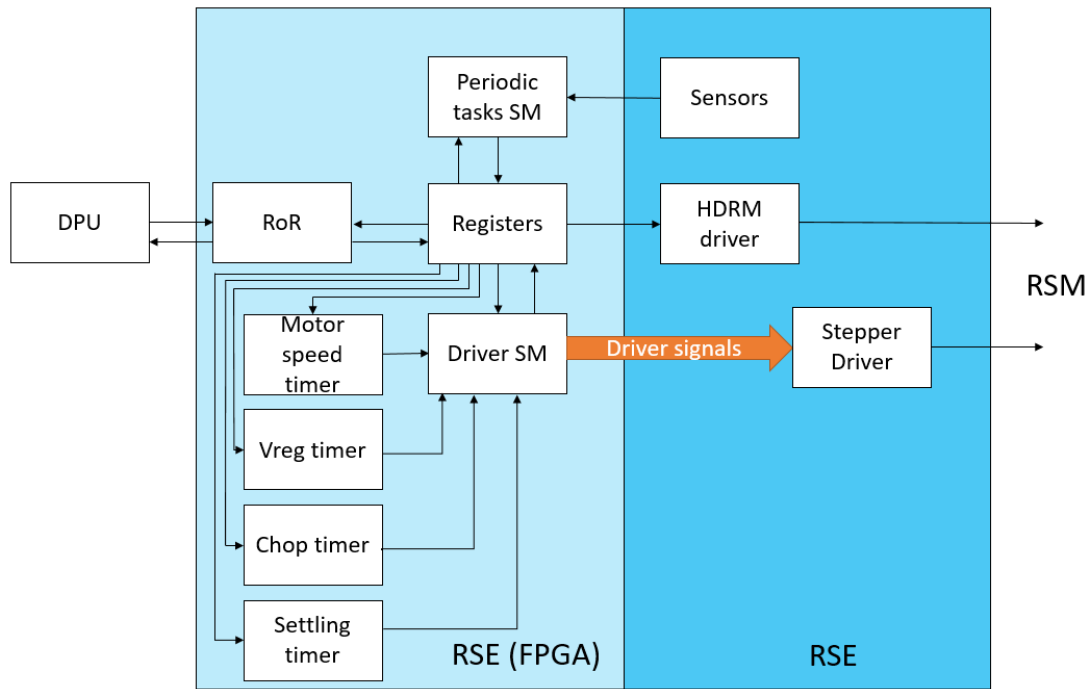


Figure 46. Potential FPGA solution for the RSE. *The RSE software is easily converted into a hardware solution, as represented by the boxes.*

6.3 Conclusion

This thesis has presented the design, implementation, and testing of a control system solution for the radiation shutter. The embedded software is implemented as a hybrid of hard and soft real-time system, without a real-time operating system. Instead, the system utilizes a custom interrupt driven scheduler that consume less memory, reduces system complexity and is more responsive. Furthermore, the system provides the SXI control unit (DPU), with a reliable memory mapped interface. This interface enables the DPU to collect HK-readout, configure RSE settings and safely operate the radiation shutter.

The radiation shutter main modules have been implemented, tested and verified. These main modules include the task handling procedures, communication, and the stepper driver software. In addition, the testing of the stepper driver hardware and the stepper motor confirmed that the system worked successfully.

The task handling tests showed that the custom interrupt driven scheduler guaranteed execution of both soft and hard real-time tasks, well within the timing constraints. Because a significant amount of effort was spent on design, the implementation and testing were vastly simplified. This decision was a major contributor to being able to develop an almost complete solution within the MSc project time frame of one year.

Appendix A Software Requirement Specification

This chapter presents the software requirements specification for the RSE. Due to the project's early stage, several key elements of the software are not yet decided or confirmed. This is indicated by: To Be Decided (TBD) or To Be Confirmed (TBC).

A.1 Start-Up

RSE-SRS-FN-0001 *Start-up routine* **Verification: T**

The start-up routine shall be executed automatically after powering on or after a reset. The start-up routine includes the following initialisation tasks:

- Initialise USART
- Initialize GPIO-ports
- Initialise timers
- Initialise register values
- Enable global interrupt

After a finished start-up routine, the main function shall transfer control to the scheduler.

RSE-SRS-FN-0002 *Initialise USART* **Verification: T / I**

Initialisation of USART shall configure the correct settings listed below:

- Set baud rate to TBD-01
- Enable transmit and receive
- Configure data frame: 1-start-bit, 9-data-bit, odd parity, 1-stop-bit TBC-01
- Enable RX complete interrupt
- Enable TX complete interrupt

RSE-SRS-FN-0003 *Initialise GPIO ports* **Verification: T / I**

Initialisation of GPIO ports shall set ports as input or output and set default values on output ports.

RSE-SRS-FN-0004 *Initialise timers* **Verification: T**

Initialisation of timers shall configure motor driver timers and the scheduler timer, with TBD-02 seconds intervals.

RSE-SRS-FN-0005 ***Initialise register values*** ***Verification: T***

Initialisation of register values shall set all registers' (status registers, control registers, and command register) default values TBD-03.

RSE-SRS-FN-0006 ***Enable global interrupt*** ***Verification: T***

After all interrupts are configured, global interrupt shall be enabled. None of the interrupts shall be triggered before this is done.

A.2 Scheduler

RSE-SRS-FN-0009 ***Scheduler*** ***Verification: T***

The scheduler shall run in a constant cycle of TBD-04 seconds. The scheduler shall guarantee the following:

- Perform periodic HK-readouts (e.g., temperature measurements)
- Interpret received data and generate a response within the RMAP over RBDP (RoR) protocol timing constraints (i.e., a response shall be generated between 1 baud delay and 24 baud delays, after a received query)
- Perform aperiodic tasks (e.g., open shutter, close shutter)

Periodic tasks:

Several tasks can execute within a cycle, including but not limited to:

- Measure motor temperature
- Measure electronics temperature
- Measure shutter status
- Measure HDRM status
- Measure processor status

Aperiodic tasks:

Only one task can execute within a cycle, including but not limited to:

- Open shutter stop at end
- Close shutter stop at end
- Open shutter max no of steps
- Close shutter max no of steps
- Emergency shutter stop at end
- Activate HDRM
- Cancel command

A.2.1 Periodic Tasks

<i>RSE-SRS-FN-0010</i>	<i>Measure motor temperature</i>	<i>Verification: T</i>
-------------------------------	---	-------------------------------

RSE shall measure the motor temperature and store the values in status register: `Motor temperature`. When the motor is inactive, a temperature reading shall be performed at intervals of 30 TBC-02 seconds. When the motor is activated, a temperature reading shall be performed at intervals of 1 TBC-03 seconds.

<i>RSE-SRS-FN-0011</i>	<i>Measure electronics temperature</i>	<i>Verification: T</i>
-------------------------------	---	-------------------------------

RSE shall measure the electronics temperature and store the values in status register: `Electronics temperature`. When the motor is inactive, a temperature reading shall be performed at intervals of 30 seconds TBD-05. When the motor is activated, a temperature reading shall be performed at intervals of 1second TBD-06.

<i>RSE-SRS-FN-0012</i>	<i>Measure shutter status</i>	<i>Verification: T</i>
-------------------------------	--------------------------------------	-------------------------------

RSE shall measure the shutter status sensors and store the value in status register: `Shutter status register`. The `shutter status register` shall also indicate if an emergency closure is initiated and if the motor or/and electronics temperatures are too hot to operate. When the motor is inactive, the `shutter status register` shall be updated in intervals of TBD-07 seconds. When the motor is activated, the `shutter status register` shall be updated in intervals of TBD-08 second.

<i>RSE-SRS-FN-0013</i>	<i>Update HDRM status</i>	<i>Verification: T</i>
-------------------------------	----------------------------------	-------------------------------

TBD-09

<i>RSE-SRS-FN-0014</i>	<i>Update Processor status</i>	<i>Verification: T</i>
-------------------------------	---------------------------------------	-------------------------------

TBD-10

A.2.2 Aperiodic Tasks

Aperiodic tasks shall be executed if the DPU has requested it. This involves reading the command register and performing the operation that corresponds to the command registers unique content.

RSE-SRS-FN-0015***Open shutter stop at end******Verification: T***

If the command register content indicates an `open shutter stop at end` operation, the RSE shall read the control registers, configure motor settings, and start the motor in the open direction. The motor shall run continuously until the open shutter sensor is activated, motor/electronics overheating, performed steps equals the maximum stepping limit, or the cancel command is initiated.

RSE-SRS-FN-0016***Close shutter stop at end******Verification: T***

If the command register content indicates a `close shutter stop at end` operation, the RSE shall read the control registers, configure motor settings and start the motor in the closed direction. The motor shall run continuously until the closed shutter sensor is activated, motor/electronics overheating, performed steps equals the maximum stepping limit or the cancel command is initiated.

RSE-SRS-FN-0017***Open shutter max no of steps******Verification: T***

If the command register content indicates an `Open shutter max no of steps` operation, the RSE shall read the control registers, configure motor settings and start the motor in open shutter direction. The motor shall run continuously until max no of steps is performed, motor/electronics overheat, or the cancel command is initiated. Notably, the sensor criteria is removed, which guarantees opening of the shutter even if the open sensor is stuck activated.

RSE-SRS-FN-0018***Close shutter max no of steps******Verification: T***

If the command register content indicates a `close shutter max no of steps` operation, the RSE shall read the control registers, configure motor settings and start the motor in open shutter direction. The motor shall run continuously until max no of steps is performed, motor/electronics overheat, or the cancel command is initiated. Notably, the sensor criteria is removed, which guarantees closing of the shutter even if the closed sensor is stuck activated.

RSE-SRS-FN-0019***Emergency close stop at end******Verification: T***

If the command register value indicates an `emergency close stop at end` operation, the RSE shall read the control registers, configure motor settings, and start the motor in close shutter direction. The motor shall run continuously as fast as possible until **only** the close shutter sensor is activated, or the cancel command is initiated. Notably, motor/electronics overheating criteria is omitted.

RSE-SRS-FN-0020***Activate HDRM******Verification: T***

The HDRM shall only be activated if the RSE receive two unique HDRM commands. The first command shall arm the HDRM and the second command shall activate the HDRM. The HDRM shall be armed for 1 second TBD-11; if an activate HDRM command is not received within this time or a command that updates any of the RSE registers are received, the HDRM shall be disarmed and the proses must be repeated. TBD-12

RSE-SRS-FN-0021***Cancel command******Verification: T***

If the command register value indicates a `Cancel command` operation, the RSE shall always cancel any ongoing operation.

A.3 Interpret Received Data

If data is received, the interpret data task shall be performed, by means of an interrupt routine. When the RSE has received a full data packet, the data packet shall be interpreted as following depending on its content:

- Read register request
- Write register request
- Undefined/ illegal request

The interpret data and response procedure shall follow the RoR protocol.

RSE-SRS-FN-0022***Read register request******Verification: T***

The RSE registers shall always be readable regardless of whether the motor is active or not; this includes the status register, the control registers, and the command register.

RSE-SRS-FN-0023***Write register request******Verification: T***

Only the control registers and the command register shall be writable; the status registers shall be read-only. All writing requests for the status register shall result in a rejected request and the remark character shall identify the error.

The control registers shall only be written if the motor is inactive; any writing request for the control registers while the motor is active, shall reject the request and the remark character shall identify the error. This shall also apply to the command registers, except if the data written to the command register identifies `cancel command`, the request shall be accepted.

A request for opening the shutter while the shutter is already open shall be rejected, and the remark character shall identify the error.

A request for closing the shutter while the shutter is already closed shall be rejected, and the remark character shall identify the error.

A request for closing/opening the shutter while the motor temperature is too hot shall be rejected, and the remark character shall identify the error.

A request for closing/opening the shutter while the electronics temperature is too hot shall be rejected, and the remark character shall identify the error.

RSE-SRS-FN-0023 **Undefined/ illegal request** **Verification: T**

Write or read request of a register that is undefined shall be rejected and, the remark character shall identify the error.

A.4 Error Messages

RSE-SRS-FN-0024 **RMAP transaction errors** **Verification: T**

After completion of a full read or write transaction, the remark character shall indicate the status of the operation as shown in Table 20. The detection of an illegal read/write operation, or a sequence error, shall also result in an aborted transaction with the remark character containing the corresponding ID listed in Table 20 , identifying the error.

Table 20: RSE application level error and status messages

Response	ID	Comment
Authorised	0x00	The operation is correct and accepted by the RSE
Sequence error	0x01	RMAP protocol sequence error
Not writable register	0x02	The register is read-only
Wrong address	0x03	No such register
Not allowed	0x04	Not allowed since a command is already being performed
Shutter already closed	0x05	Not allowed since the Shutter is already closed
Shutter already open	0x06	Not allowed since the Shutter is already open
Motor temperature too high	0x07	Not allowed since the motor is too hot
Electronics temperature too high	0x08	Not allowed since electronics is too hot

Appendix B Test Report for RSE Motor Control

This chapter presents the test report for RSE motor control.

Due to the construction of the motor driver, three parameters are adjustable, including the duty cycle of V_{ctrl} signal, the duty cycle of the chop signal and the settling time. Each of these parameters affects both the motor torque and the average power supply current draw. In these experiments, both the motor torque and the average power supply current draw with respect to the parameters is measured independently. The intent of these measurements is to find a compromise between torque and the average current draw and to confirm that key parameters, such as maximum torque and motor current, match their theoretical values.

B.1 Equipment

Table 21. Equipment list

Device	Name	Serial No
Torque meter	TruCheck Plus 3 N·m	43250
Power supply	TTI QL355TP	429733
Power supply	TTI QL355TP	314963
Oscilloscope	Tektronix MD04104C	MD04104C C002440
Stepper motor	VSS 25.200.0.6-HV-GPL-KTC	
Microcontroller	Atmega2560	
Motor driver	Custom	
Fan	Ide line LQ-7	

B.2 Experiment Overview

The equipment, listed in Table 21, is connected as illustrated in Figure 47 and remain unchanged during the whole experiment.

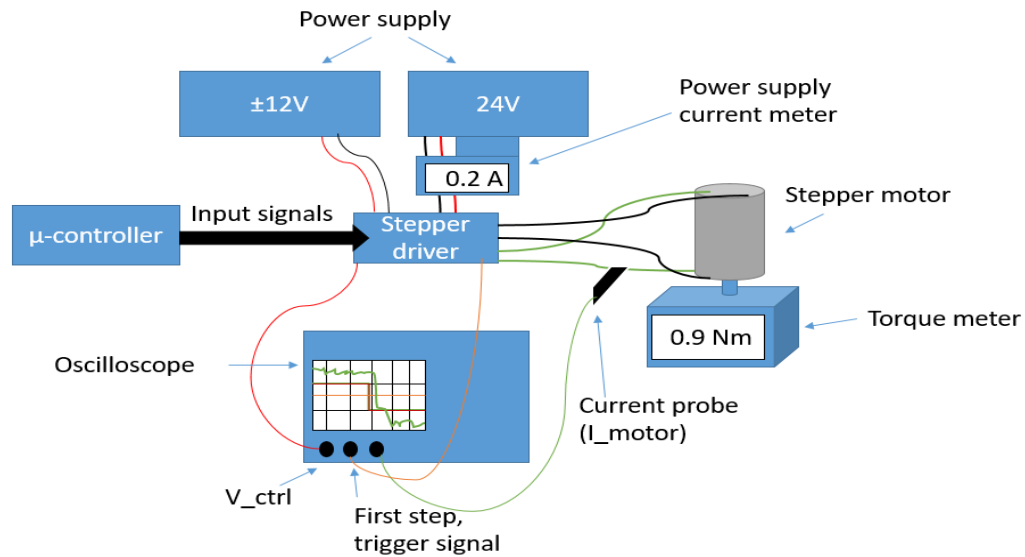


Figure 47. Test setup

The input signals from the microcontroller are connected to the driver: eight control signals for the transistors in the two H-bridges, one control signal for the voltage regulator (V_{ctrl}) and one for ground. The driver is supplied with two voltage sources $\pm 12\text{ V}$ for the operational amplifiers and 24 V for the H-bridges which is responsible for directing current in the motor. The outputs from the driver H-bridges are connected to the stepper motor, two wires for each coil pair. The stepper motor is mechanically connected to the torque measurement device which is bolted to a massive frame. The oscilloscope measured the current in one coil pair, the voltage regulator output voltage V_{ctrl} and a trigger signal for the first step in a sequence. The fan prevents the driver from overheating.

A total of three experiments were performed. In the first experiment, the effect of adjusting V_{ctrl} was measured. In the second experiment, the effect of adjusting chop duty cycle was investigated. In the third experiment, the effect of adjusting the settling time was examined.

B.3 Experiment 1, V_{ctrl} Duty Cycle Sweep

B.3.1 Theory, Experiment 1

The voltage regulator shall set the current limit for the stepper motor. This is done by adjusting the duty cycle of the voltage regulator's input signal which will result in a configurable voltage between 0 – 3.3 V. The current limit is given by Equation (11) and should be adjustable between 0 – 0.825 A

$$I_{Motor} = \frac{V_{ctrl}}{4 \cdot R_{sense}} = \frac{V_{ctrl}}{4} \quad (11)$$

B.3.2 Objective, Experiment 1

The goal of this experiment is to find the effect V_{ctrl} has on both the torque and average current draw. The linearity between the motor current and torque indicates whether the motor and driver work properly, and this is what we want to confirm by means of this experiment. The theoretical values of V_{ctrl} , I_{motor} and the motor torque should match the experimental values.

B.3.3 Method, Experiment 1

The duty cycle of V_{ctrl} is adjusted from 22 – 100 %. The range from 0 – 21% is skipped because this is below the driver operation range. A total of four torque readings are performed, two readings each direction, before the duty cycle is changed. In addition, the experimental values of V_{ctrl} , I_{motor} and power supply current draw are measured. All other driver parameters are constant during the whole experiment, as listed in Table 22. Table 23 shows the calculated values of V_{ctrl} and I_{motor} .

Table 22. Driver settings, experiment 1

Driver timer	Parameter value	Register value
Chop timer 8bit	50 [%]	128
Chop settling timer 16bit	140 [μ s]	140

Table 23. Calculated values for experiment 1

V_{ctrl} Duty Cycle [%]	V_{ctrl} timer register Value	V_{ctrl} [V]	I_{motor} [A]
22	56	0.7	0.182
23	59	0.8	0.190
24	61	0.8	0.198
25	64	0.8	0.206
30	77	1.0	0.248
35	89	1.2	0.289
40	102	1.3	0.330
45	115	1.5	0.371
50	128	1.7	0.413
55	140	1.8	0.454
60	153	2.0	0.495
65	166	2.1	0.536
70	179	2.3	0.578
75	191	2.5	0.619
80	204	2.6	0.660
85	217	2.8	0.701
90	230	3.0	0.743
95	242	3.1	0.784
100	255	3.3	0.825

B.3.4 Results for Experiment 1

Table 24 shows the experimental values and corresponding measurement uncertainties. The average value of the four readings (*AVG Torque*) is calculated for each torque measurement. We observe that the uncertainty from the measurements are much greater than the uncertainty from the torque meter, therefore the instrument uncertainty is omitted. The uncertainty for the torque measurements is therefore calculated by using the standard deviation of the four measurements, given by Equation (12).

$$S_{AVG Torque} = \sqrt{\frac{\sum \bar{x}_t^2}{(N-1)}} \quad (12)$$

The power supply current meter is specified to have an accuracy of $\pm 2\% + 0.005[A]$. Since the measurements for $I_{pwr-supply}$ are almost constant during each measurement, and the current is low, the major contributor to the measurement uncertainty is the fixed part, $\pm 0.005 A$. The uncertainty for the motor current measurements (I_{motor}) and voltage regulator output voltage (V_{ctrl}) is estimated by the smallest resolution the signal can be measured accurately without too much ripple. The estimated uncertainty for I_{motor} and V_{ctrl} is $\pm 0.02A$ and $\pm 0.05 V$ respectively.

Table 24. Experimental values for experiment 1

V_{ctrl} Duty Cycle [%]	AVG Torque [Nm]	$I_{Pwr-supply}$ [A]	I_{motor} [A]	V_{ctrl} [V]
22	0	0.034 ± 0.005	0 ± 0.03	0.75 ± 0.05
23	0	0.045 ± 0.005	0 ± 0.03	0.787 ± 0.05
24	0.12 ± 0.04	0.053 ± 0.005	0.150 ± 0.03	0.815 ± 0.05
25	0.23 ± 0.01	0.059 ± 0.005	0.159 ± 0.03	0.854 ± 0.05
30	0.35 ± 0.01	0.076 ± 0.005	0.200 ± 0.03	1.03 ± 0.05
35	0.44 ± 0.03	0.089 ± 0.005	0.260 ± 0.03	1.19 ± 0.05
40	0.51 ± 0.03	0.103 ± 0.005	0.310 ± 0.03	1.36 ± 0.05
45	0.62 ± 0.07	0.118 ± 0.005	0.350 ± 0.03	1.52 ± 0.05
50	0.67 ± 0.06	0.133 ± 0.005	0.400 ± 0.03	1.69 ± 0.05
55	0.80 ± 0.03	0.148 ± 0.005	0.430 ± 0.03	1.84 ± 0.05
60	0.90 ± 0.05	0.165 ± 0.005	0.495 ± 0.03	2.01 ± 0.05
65	1.00 ± 0.07	0.182 ± 0.005	0.520 ± 0.03	2.19 ± 0.05
70	1.11 ± 0.05	0.202 ± 0.005	0.560 ± 0.03	2.35 ± 0.05
75	1.23 ± 0.03	0.220 ± 0.005	0.600 ± 0.03	2.52 ± 0.05
80	1.32 ± 0.04	0.240 ± 0.005	0.640 ± 0.03	2.69 ± 0.05
85	1.42 ± 0.01	0.260 ± 0.005	0.690 ± 0.03	2.82 ± 0.05
90	1.51 ± 0.05	0.287 ± 0.005	0.710 ± 0.03	2.98 ± 0.05
95	1.58 ± 0.01	0.310 ± 0.005	0.755 ± 0.03	$3,14 \pm 0.05$
100	1.64 ± 0.02	0.332 ± 0.005	0.800 ± 0.03	$3,30 \pm 0.05$

B.3.5 Discussion for Experiment 1

The experimental results listed in Table 24 were used to graphically illustrate the effect of adjusting V_{ctrl} . Error bars represent the uncertainty for each measurement.

Figure 48 shows the relationship between I_{motor} and V_{ctrl} . The graph clearly illustrates that below 24% duty cycle, which corresponds to $V_{ctrl} \approx 0.8V$, the driver is non-functional. This is due to the threshold voltage required to turn on the driver PnP transistors, labelled T3 and T4 in Figure 16. On the other hand, I_{motor} is linear from 24-100% duty cycle. By comparing the calculated and experimental values of V_{ctrl} and I_{motor} listed in Table 23 and Table 24, respectively, the values are practically equal if we consider the uncertainty. This indicates that the current limiting circuitry functions correctly and the current limit can be adjusted between 0.15 – 0.80A.

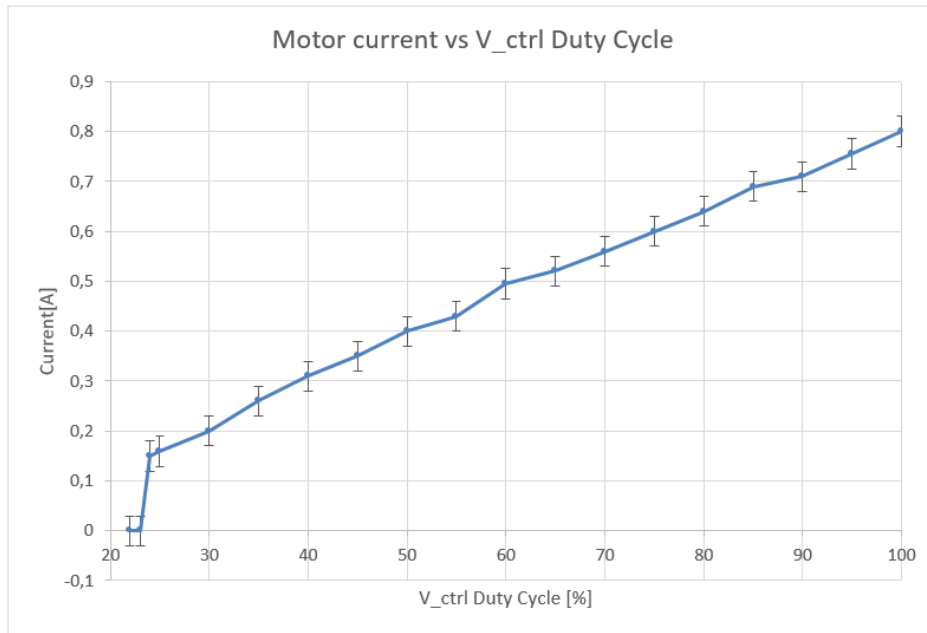


Figure 48. Motor current versus V_{ctrl} duty cycle

Figure 49 shows the relationship between torque and V_{ctrl} duty cycle. The graph indicates that the torque is linear between 25 – 100% duty cycle and not between 24 – 100% duty cycle as we would assume from Figure 48. The reason for this may be that the PnP transistors are at their edge of the threshold voltage when the V_{ctrl} duty cycle is in the low range, and therefore not fully turned on. Figure 50 shows the relationship between I_{motor} and the torque. The graph shows that I_{motor} and the torque are nearly linear, and the maximum torque is $1.64 \pm 0.02 Nm$.

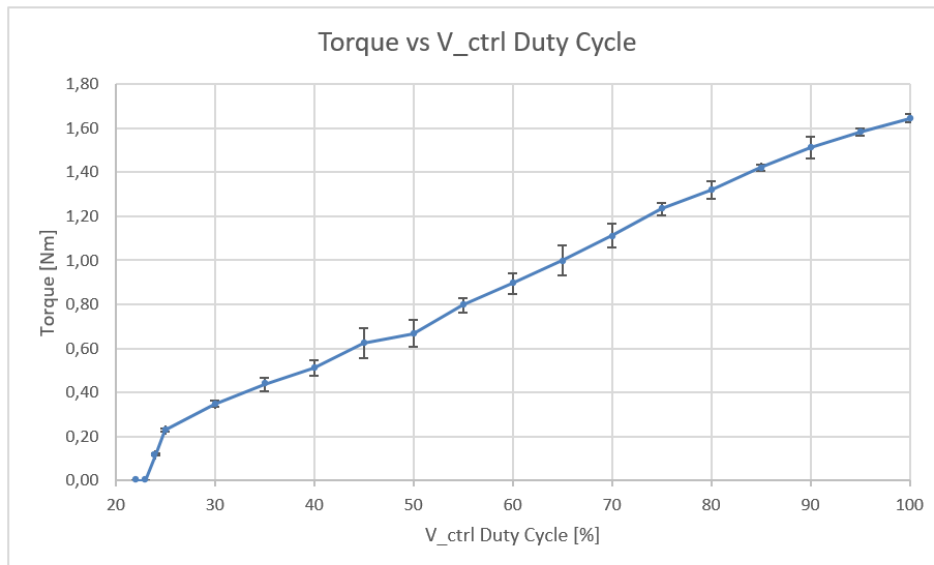


Figure 49. Torque versus V_{ctrl} duty cycle

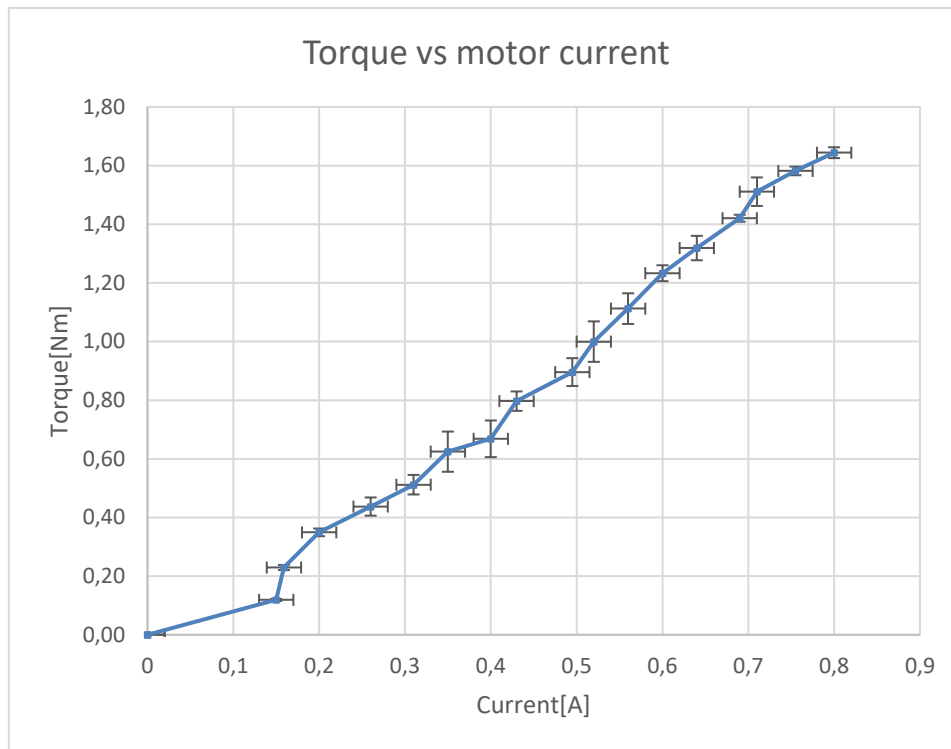


Figure 50. Torque versus V_{ctrl} duty cycle

According to the motor and gearing specifications, the theoretical motor torque at nominal current (0.6A) is given by Equation (13).

$$\begin{aligned} \textit{Theoretical}_{torque} &= \textit{Holding}_{torque} \cdot \textit{Ratio} \cdot \textit{Efficiency} \\ \textit{Theoretical}_{torque} &= 0.012Nm \cdot 196 \cdot 0.8 = 1.881 Nm \end{aligned} \quad (13)$$

Table 24 shows that a motor current of 0.6[A] corresponds to a torque of $1.23 \pm 0.03 Nm$, which is only 65% of the theoretical torque. This might be because of the stepper motor is running in half step mode. The half step mode gives better resolution and smoother operation, but the torque is only 70% every second step (the half step). Another factor could be the spring, which connects the motor to the torque measurement device. By taking these factors into account are the experimental value and theoretical value of the torque as expected.

B.4 Experiment 2, Chop Duty Cycle Sweep

B.4.1 Theory

The chopping operation takes advantages of the fact that the current in an inductor, i.e., the motor current, cannot change instantaneously. By switching the inductor current on and off fast enough, the inductor manages to keep the current at a nearly constant level, without continuously wasting too much power on a current limiting resistor. By using this chopping technique, the power consumption and heat generation should be drastically reduced. The on/off time is controlled by the duty cycle of the chopping signal, where lower percentage duty cycle means more on time and likely more power dissipation.

B.4.2 Experiment 2, Objective

In this experiment, the effect of chop duty cycle with respect to the torque characteristics and the average power dissipation is examined. The experiment should also give a reasonable assumption of the optimal chop duty cycle where there would be a compromise between high torque and low power dissipation.

B.4.3 Method for Experiment 2

In the following experiment, the duty cycle of chop signal is adjusted from 0 – 80%. The range from 80 – 100% is skipped because this is outside of the driver operation range. A total of four torque readings are performed, two readings each direction, before the duty cycle is incremented. In addition, the experimental value of the power supply current draw is measured. The nominal operating current for the motor is rated to 0.6A, therefore the current limit is set to 0.6A and kept constant during the whole experiment. The fixed driver parameters are listed in Table 25.

Table 25. Driver settings for experiment 2

Parameter	Parameter value	Register value
V_{ctrl}	80[%]	205
Settling time	140 [μs]	140

B.4.4 Results for Experiment 2

The experimental values and its corresponding measurement uncertainties are listed in Table 26. The values and the uncertainties are estimated by the same method as in experiment 1.

Table 26. Experimental values for experiment 2

Chop Duty Cycle [%]	AVG Torque [Nm]	$I_{Pwr-supply}$ [A]
0	1.44 ± 0.02	0.964 ± 0.005
2	1.431 ± 0.007	0.964 ± 0.005
5	1.430 ± 0.009	0.901 ± 0.005
10	1.42 ± 0.02	0.808 ± 0.005
15	1.41 ± 0.02	0.723 ± 0.005
20	1.40 ± 0.02	0.624 ± 0.005
25	1.39 ± 0.01	0.525 ± 0.005
30	1.386 ± 0.004	0.443 ± 0.005
35	1.381 ± 0.004	0.377 ± 0.005
40	1.36 ± 0.02	0.310 ± 0.005
45	1.360 ± 0.004	0.256 ± 0.005
50	1.350 ± 0.009	0.245 ± 0.005
55	1.276 ± 0.008	0.230 ± 0.005
60	0.67 ± 0.04	0.135 ± 0.005
62	0.44 ± 0.02	0.098 ± 0.005
65	0.162 ± 0.006	0.061 ± 0.005
75	0.147 ± 0.009	0.059 ± 0.005
80	0.14 ± 0.01	0.059 ± 0.005

B.4.5 Discussion for Experiment 2

Figure 51 shows the effect that chop duty cycle has on both the torque and the average current draw. The graph illustrates that beyond 50% duty cycle, the torque drops significantly. This is limited by the speed of operational amplifier, which needs a minimal time to reach the transistor gate threshold voltage. On the other hand, the torque is nearly stable for 0 – 50% duty cycle, with only a difference of 90 Nm which corresponding to about 5% less torque. As the duty cycle increases the power supply current drops considerably. Furthermore, a chop duty cycle between 45% and 50%, the torque is almost at its maximum while the current draw is reduced by roughly 75%. Since operating close to the edge (50%) could potentially create unstable behaviour, a duty cycle of approximately 45% would be ideal.

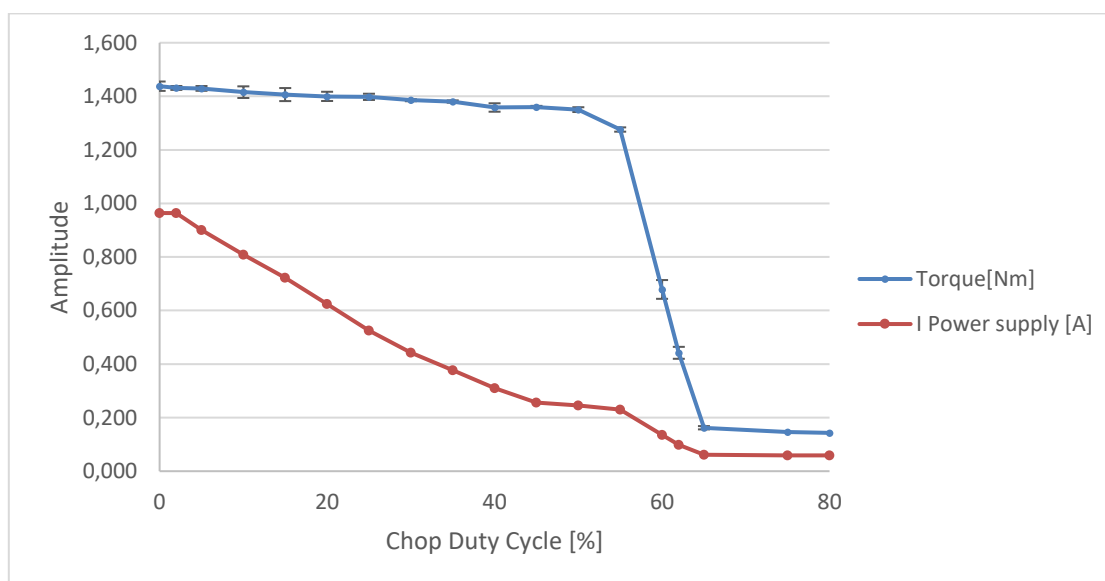


Figure 51. Chop duty cycle effect

B.5 Experiment 3, Settling Time Sweep

B.5.1 Theory, Experiment 3

The driver's settling time variable is an adjustable delay, which determined when the motor current starts to chop. If the delay is too short, the motor current rise time might be longer, which would affect the torque. On the other hand, if the delay is too long, unnecessary power is dissipated in the current limiting transistor.

B.5.2 Objective, Experiment 3

In this experiment, the effect of settling time with respect to the torque characteristics and the average power dissipation is examined. The experiment should also give a reasonable assumption of the optimal settling time where there would be a compromise between high torque and low power dissipation.

B.5.3 Method for Experiment 3

In the following experiment, the settling time is adjusted from 0 – 1000 μs . A total of four torque readings are performed, two readings each direction, the settling time is incremented in steps from 10 – 500 μs . In addition, the value of the power supply current draw is measured. Table 27 shows the fixed driver parameters.

Table 27. Driver settings for experiment 3

Parameter	Parameter value	Register value
V_{ctrl}	80[%]	205
Chop duty cycle	50[%]	128

B.5.4 Results for Experiment 3

The experimental values and its corresponding measurement uncertainties are listed in Table 28. The values and the uncertainties are estimated identical to that of experiment 1.

Table 28. Experimental values for experiment 3

Settling time [μs]	AVG Torque [Nm]	$I_{Pwr-supply}$ [A]
3	1.253 \pm 0.06	0.211 \pm 0.05
13	1.267 \pm 0.05	0.214 \pm 0.05
26	1.285 \pm 0.04	0.215 \pm 0.05
38	1.292 \pm 0.06	0.217 \pm 0.05
51	1.296 \pm 0.05	0.219 \pm 0.05
64	1.324 \pm 0.03	0.225 \pm 0.05
77	1.326 \pm 0.05	0.227 \pm 0.05
89	1.329 \pm 0.04	0.230 \pm 0.05
102	1.338 \pm 0.04	0.232 \pm 0.05
115	1.333 \pm 0.03	0.236 \pm 0.05
128	1.338 \pm 0.04	0.238 \pm 0.05
140	1.325 \pm 0.04	0.241 \pm 0.05
153	1.335 \pm 0.04	0.243 \pm 0.05
166	1.328 \pm 0.04	0.247 \pm 0.05
179	1.335 \pm 0.03	0.251 \pm 0.05
191	1.336 \pm 0.03	0.254 \pm 0.05
250	1.335 \pm 0.03	0.269 \pm 0.05
300	1.337 \pm 0.04	0.282 \pm 0.05
500	1.342 \pm 0.05	0.333 \pm 0.05
1000	1.347 \pm 0.06	0.586 \pm 0.05

B.5.5 Discussion for Experiment 3

Figure 52 illustrates both the torque and the power supply current draw with respect to settling time. The settling time has a slight influence on the torque, but below $100\mu s$ there is a minor decrease. The reason for this is believed to be that the chopping starts before the current reaches the current limit, because the coil current does not settle until after approximately $100\mu s$. On the other hand, the current draw increases significantly, if the settling time is too long. By comparing these graphs, a settling time between $100 - 130\mu s$ is a decent estimate for the optimal settling time variable.

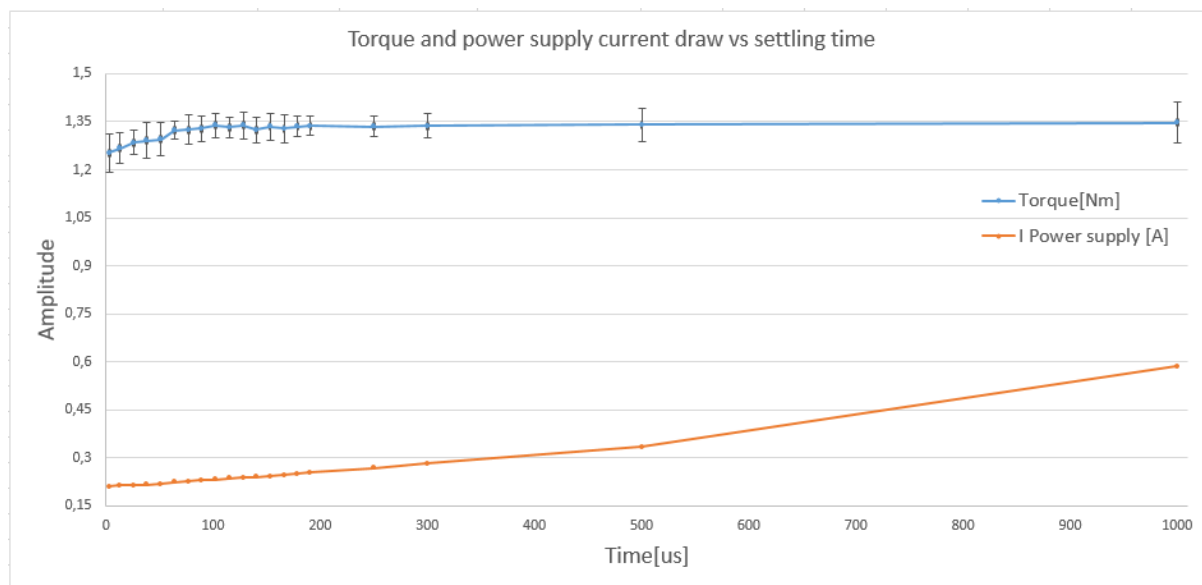


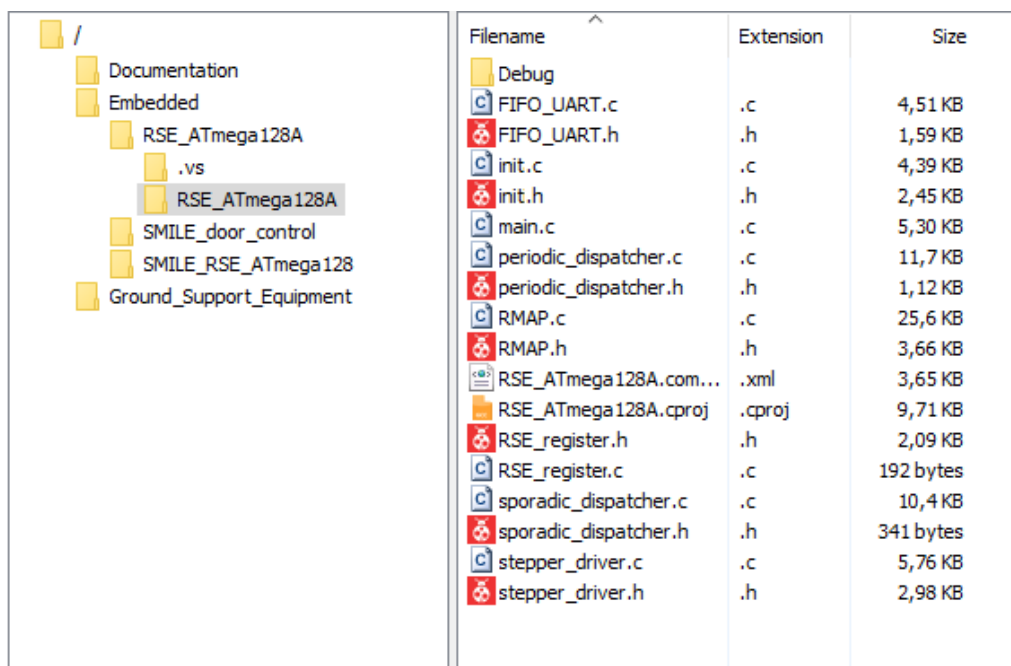
Figure 52. Settling time parameter effect

B.6 Test Conclusion

In these experiments, the effects of adjusting the motor driver's parameter are examined. As a result, the operation range of V_{ctrl} is found to be 25 – 100% duty cycle, which corresponds to a current limit of $0.16 A - 0.8 A$. The rated motor torque at $0.6 A$ matches the calculated value based on the data sheet. The importance of current chopping is confirmed, by using a chop duty cycle of 45% the average current draw is reduced by 70% at the cost of only 5% less torque. The settling time parameter has a negligible effect on the torque characteristics after $100\mu s$, but shows significant influence on the current draw.

Appendix C Software Organising

Figure 53 shows an overview of the git repository where the current project's files are displayed (right). The repository is divided into three top-level folders: documentation, embedded and ground support equipment. As the name implies, the “documentation” folder contains documentation established during the RSE development, such as SRS, test reports, and other vital materials. The “embedded” folder contains the embedded software, where RSE_ATmega128A is the latest version (31.05.2018); Table 29 offers a brief description of its contents. Finally, the DPU simulator is located in the “ground support equipment” folder.



Filename	Extension	Size
Debug		
FIFO_UART.c	.c	4,51 KB
FIFO_UART.h	.h	1,59 KB
init.c	.c	4,39 KB
init.h	.h	2,45 KB
main.c	.c	5,30 KB
periodic_dispatcher.c	.c	11,7 KB
periodic_dispatcher.h	.h	1,12 KB
RMAP.c	.c	25,6 KB
RMAP.h	.h	3,66 KB
RSE_ATmega128A.com...	.xml	3,65 KB
RSE_ATmega128A.cproj	.cproj	9,71 KB
RSE_register.h	.h	2,09 KB
RSE_register.c	.c	192 bytes
sporadic_dispatcher.c	.c	10,4 KB
sporadic_dispatcher.h	.h	341 bytes
stepper_driver.c	.c	5,76 KB
stepper_driver.h	.h	2,98 KB

Figure 53. Git repository overview

Table 29. RSE embedded software files

Filename	File type	Description
main.c	Source file	Contains the main function which enables the initialisation routines and the scheduler. In addition, all ISRs are located in this file.
init.c	Source file	Contains all initialisation functions
init.h	Header file	Contains macro for system settings and function prototypes
sporadic_dispatcher.c	Source file	Contains functions that interact with the stepper motor or the HDRM. (sporadic tasks)
sporadic_dispatcher.h	Header file	Contains prototypes and variable declarations for functions/ variables that needs a scope outside the source file.
periodic_dispatcher.c	Source file	Contains functions that enables periodic measurements
periodic_dispatcher.h	Header file	Contains, function prototypes and macros to configure the periodic measurements.
RSE_register.c	Source file	Contains the struct definition for all RSE registers.
RSE_register.h	Header file	Contains the struct declaration for all RSE registers. In addition, all macros for identifying status events are located in this file
RMAP.c	Source file	Contains all functions for the communication task
RMAP.h	Header file	Contains communication task macros and function prototypes
FIFO_UART.c	Source file	Contains a software implemented FIFO for transmitting data.
FIFO_UART.h	Header file	FIFO function prototype.

Appendix D RSE Pin Map

Table 30 presents the pin map between the microcontroller and the stepper driver. All pins from the microcontroller are labelled and routed onto the STK600 development board. Because, the stepper drivers' ports are unlabelled, a picture of the driver and an illustration which identifies that ports are shown in Figure 54. Furthermore, Table 31 presents the serial interface between the microcontroller (RSE) and the DPU.

A 4-pin connector is connected between the driver and the motor. This contact can be coupled in two possible combinations (i.e., by horizontally flipping it), which only affect the stepper motor rotating direction. Because the motor direction is not yet finalized, a pin map for this contact is not provided.

Table 30. Pin map for ATmegaS128 and stepper driver.

AtmegaS128	Stepper driver
PA0	PA coil1
PA1	PB coil1
PE4	NA coil1
PE3	NB coil1
PA2	PA coil2
PA3	PB coil2
PB7	NB coil2
PE5	NB coil2
PB4	Vctrl pin
GND	GND
GND	GND

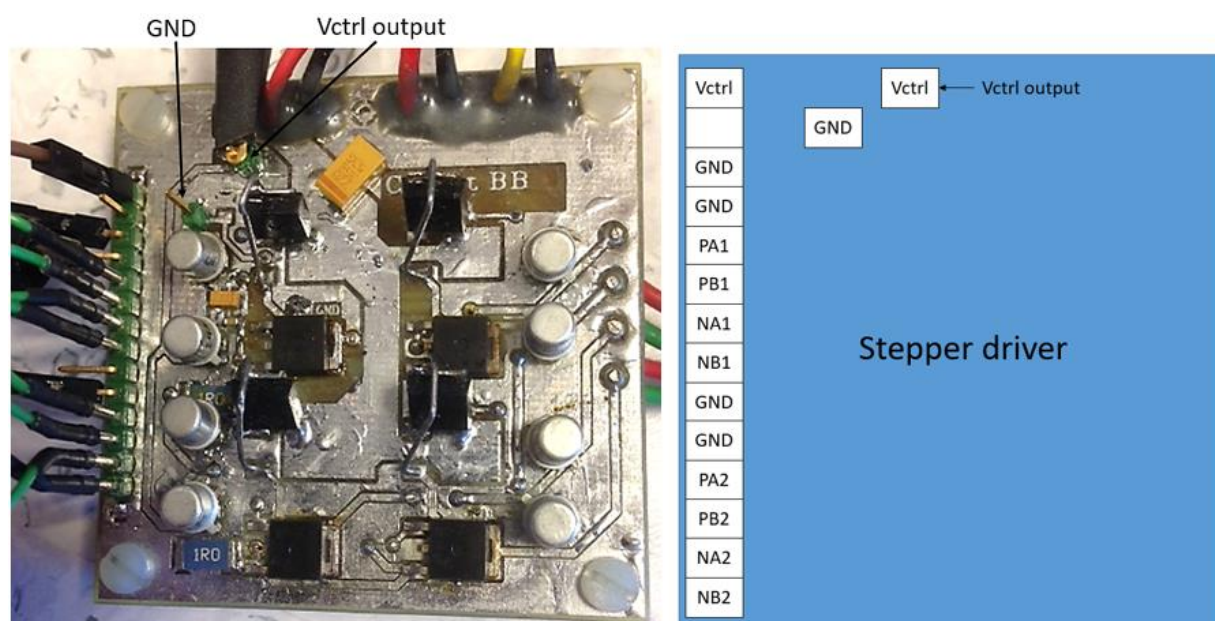


Figure 54. Stepper driver (left) and an illustration identifying the drivers' ports (right)

Table 31. Pin map for ATmegS128 and DPU

ATmegaS128	DPU
PE1	RX
PE0	TX
GND	GND

7 References

- [1] G. Branduardi-Raymont and C. Wang, "Joint Scientific Space Mission Chinese Academy of Science (CAS) - European Space Agency (ESA) SMILE, mission proposal [Internal report]," n.d.
- [2] J. A. e. a. Carter, "A high charge state coronal mass ejection seen through solar wind charge exchange emission as detected by XMM–Newton, article in a periodical," *Monthly Notices of the Royal Astronomical Society*, pp. 867-878, 02 February 2010.
- [3] Birkeland Centre For Space Science [Internal report], "UB Design and Development Plan for Radiation Shutter V001_1dF," University of Bergen, Bergen, 2018.
- [4] N. Fox and J. L. Burch, "Science Objectives and Rationale for the Radiation Belt, book section," in *The Van Allen Probes Mission*, vol. 179, Laurel, Springer, Boston, MA, 2013, pp. 4-12.
- [5] Birkeland Centre For Space Science [Internal report], "Design Report for RSE V001_1dE," University of Bergen, Bergen, 2018.
- [6] Birkeland Centre For Space Science [Internal report], "Design Report for RSM V002_1dE," University of Bergen, Bergen, 2018.
- [7] Phytron, "VSS Stepper Motor for Applications up to Ultra-high-vacuum, documentation," Phytron, Gröbenzell, 2016.
- [8] Z. Cao, Y. Shao, M. Rao and W. Yu, "Effects of the gear eccentricities on the dynamic performance of a planetary gear set, journal article," *Nonlinear Dynamics*, vol. 91, no. 1, pp. 1-15, 07 November 2017.
- [9] Microchip, "STK600 AVR® Flash MCU Starter Kit User's Guide," 2018. [Online]. Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/STK600-AVR-Flash-MCU-Starter-Kit-Users-Guide-DS40001904B.pdf>. [Accessed 24 Mai 2018].
- [10] Microchip, "AVR Dragon, datasheet," Atmel, April 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/atmel-42723-avr-dragon_userguide.pdf. [Accessed 24 Mai 2018].
- [11] Microchip, "Atmel Studio, user guide," September 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42167-Atmel-Studio_User%20Guide.pdf. [Accessed 5 Mai 2018].
- [12] A. Rivetti, "Radiation Damage," in *CMOS front-end electronics for radiation sensors, book section*, New York, Taylor & Francis Group, 2015, pp. 677-681.

- [13] Microchip, "ATmegaS128, datasheet," Microchip, USA, 2017.
- [14] P. Horváth, "SMILE SXI PSU Regular Bytestream DAQ Protocol Definition, documentation [Internal report]," Budapest, Könyves Kálmán, 2018.
- [15] P. Horváth, "SMILE SXI PSU, RMAP over RBDP Protocol Definition, documentation [Internal report]," Péter Horváth, Budapest, 2018.
- [16] X. Hu, "An Introduction to Stepper Motors, document from website," n.d. [Online]. Available: https://wp.optics.arizona.edu/optomech/wp-content/uploads/sites/53/2016/10/Tutorial_Xinda-Hu.pdf. [Accessed 2 January 2018].
- [17] R. Halstead, "Considerations for Step Motors in Space Applications, web site," Empiremagnetics, n.d. [Online]. Available: http://www.empiremagnetics.com/articles/space_applications.htm. [Accessed 2 January 2018].
- [18] S. Murugesan, "An Overview of Electric Motors for Space Applications, document from website," November 1981. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4159575>. [Accessed 20 December 2017].
- [19] PAControl, "Stepper Motor Basics," n.d. [Online]. Available: <http://www.pacontrol.com/download/Stepper-Motor-Basics.pdf>. [Accessed 3 January 2018].
- [20] C. Reston and D. W. Jones, "Stepping Motors Fundamentals, document from website," 26 January 2004. [Online]. Available: <http://www.divms.uiowa.edu/~jones/step/an907a.pdf>. [Accessed 3 January 2018].
- [21] Birkeland Centre For Space Science [Internal report], "Stepper driver schematic," University of Bergen, Bergen, 2018.
- [22] T. Hopkins, "Stepper motor driving, application note," November 2012. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/application_note/57/c8/7c/c1/0d/91/46/89/CD00003774.pdf/files/CD00003774.pdf/jcr:content/translations/en.CD00003774.pdf. [Accessed 3 January 2018].
- [23] R. Oshana, Software engineering for embedded systems, book, Amsterdam: Elsevier/Newnes, 2013, pp. 9-25.
- [24] D. Camera, "Newbie's Guide to AVR Interrupts, website," AVRFREAKS, 16 February 2010. [Online]. Available: <https://www.avrfreaks.net/forum/tut-newbies-guide-avr-interrupts?name=PNphpBB2&file=viewtopic&t=89843>. [Accessed 02 April 2018].
- [25] R. Williams, Real-Time Systems Development, book, 1 ed., Burlington: Elsevier Science & Technology, 2005, pp. 29-201.

-
- [26] Y. Bassil, "A Simulation Model for the Waterfall Software Development Life Cycle, Journal article," *International Journal of Engineering & Technology*, vol. 2, no. 5, p. 7, 2012.
- [27] N. Jones, "Introduction to MISRA C, web site," Embedded, 01 July 2002. [Online]. Available: <https://www.embedded.com/electronics-blogs/beginner-s-corner/4023981/Introduction-to-MISRA-C>. [Accessed 25 Mai 2018].
- [28] MISRA, "Achieving Compliance with MISRA Coding Guidelines," April 2016. [Online]. Available: https://www.misra.org.uk/LinkClick.aspx?fileticket=w_Syhpkf7xA%3D&tabid=57. [Accessed 25 Mai 2018].
- [29] A. Rongala, "Benefits of Python over Other Programming Languages, website," *invenis*, December 2015. [Online]. Available: <https://www.invenis.net/blog/it/benefits-of-python-over-other-programming-languages/>. [Accessed 26 Mai 2018].
- [30] The MuShield Company, "About mu-metal," 2013. [Online]. Available: http://www.mumetal.com/about_mumetal-pdf.pdf. [Accessed 29 Mai 2018].
- [31] Birkeland Centre For Space Science [Internal report], "Test report for stepper motor magnetic field measurements, V002," University of Bergen, 2018.
- [32] M. Lefebvre, "Dynamics of Hybrid Stepper Motors, website," *Machinedesign*, 17 February 2005. [Online]. Available: <http://www.machinedesign.com/archive/dynamics-hybrid-stepper-motors>. [Accessed 2 January 2018].