DEGREE PROJECT, IN SYSTEM-ON-CHIP DESIGN , SECOND LEVEL
STOCKHOLM, SWEDEN 2014

Sounding Rocket Experiment Electronics - RTL Design and Validation

RUSLAN KEVORKOV

KTH ROYAL INSTITUTE OF TECHNOLOGY

ELECTRICAL ENGINEERING, SPACE AND PLASMA PHYSICS DEPARTMENT

TRITA XR-EE-SPP 2014:002

TRITA XR-EE-SPP 2014:003

ROYAL INSTITUTE
OF TECHNOLOGY

Master of Science Thesis

Sounding Rocket Experiment Electronics – RTL Design and Validation

Ruslan Kevorkov

Department of Space and Plasma Physics

Royal Institute of Technology

2014

Abstract

The Infrared Spectroscopy to Analyse the middle Atmosphere Composition (ISAAC) is an experimental module designed by KTH students. It consists of a Rocket Mounted Unit (RMU) and two Free-Falling Units (FFU) carried inside. The main objective of the experiment is to demonstrate ability of one FFU to track the other and to carry out measurements in cooperation.

This Master's thesis covers the development and implementation of the ejection system as well as data acquisition for the ISAAC experiment to have well-timed ejection of the FFUs and data for a post-flight analysis. Ejection control and communication is implemented in a Field-Programmable Gate Array (FPGA) using VHDL hardware description language. Newly developed firmware verification and the post-flight analysis results are also presented in the report.

The ISAAC experiment was launched on May 29 from Esrange, Kiruna onboard the REXUS15 rocket.

Sammanfattning

ISAAC (Infrared Spectroscopy to Analyse the Middle Atmosphere Composition) är en raketmonterad experimentmodul designad av studenter på KTH. Modulen består av en raketmonterad modul benämnd RMU (Rocket Mounted Module), i vilken två mindre fritt fallande enheter benämnda FFU (Free Falling Units) sitter monterade. Huvudmålet med experimentet är att demonstrera förmågan för den ena FFU:n att spåra den andra FFU:n samt förmågan att genomföra koordinerade mätningar.

Detta examensarbete behandlar utvecklandet och implementationen av utskjutningssystemet samt datainsamlingen för ISAAC-experimentet. Dessa delar görs för att kunna genomföra utskjutningen vid en lämplig tidpunkt samt få data till efterbehandling. Utskjutningskontroll samt kommunikation är implementerade i en FPGA (Field Programmable Gate Array) i det hårdvarubeskrivande språket VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language). Verifikation av nyutvecklad inbyggd programvara samt analysresultat av data från uppskjutningen presenteras också.

Uppskjutningen av ISAAC-experimentet skedde den 29:e maj 2014 från rymdbasen Esrange i Kiruna ombord på raketen REXUS15.

Acknowledgement

Current thesis work has been performed at the Space and Plasma Physics department at Royal Institute of Technology (Kungliga Tekniska Högskolan, Stockholm) under supervision of Professor Nickolay Ivchenko. The research is a part of the international students program REXUS/BEXUS supported by SNSB and DLR.

First of all, I would like to express sincere gratitude to every person working at Space and Plasma Physics Department for creating friendly and productive atmosphere.

I am extremely grateful to my supervisor Professor Nickolay Ivchenko for his constant guidance, valuable advices and constructive criticism. Special thanks to the ISAAC team, especially Vlad Grigore, Georg Balmer, Audrey Berquand, Jorge Pacheco, and Master's degree student Marcus Lindh. Thanks for all your efforts, help and unforgettable time during trips to Germany and Esrange.

Without SNSB and DLR support, I would not have an opportunity to be involved in such an exciting project and these people must be appreciated as well.

University for provided sponsorship for all my trips.

Last but not least, I want to thank my family and friends, who believed in me and morally supported throughout whole my life.

Contents

List of figures

List of tables

List of abbreviations

ADC	Analogue-to-Digital Converter
CU	Common Unit
DLR	German Aerospace Center
DMM	Digital MultiMeter
Esrange	European Space and Sounding Rocket Range
FFU	Free-Falling Unit
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface
IR	Infrared
ISAAC	Infrared Spectroscopy to Analyse the middle Atmosphere Composition
LSB	Least significant bit
PCB	Printed Circuit Board
REXUS	Rocket Experiments for University Students
RMU	Rocket Mounted Unit
RTL	Register Transfer Level
RXSM	REXUS Service Module
RxSU	Receiver Specific Unit
SED	Student Experiment Document
SNSB	Swedish National Space Board
SOC	System-on-Chip
SU	Specific Unit
TxSU	Transmitter Specific Unit

Chapter 1 Introduction

1.1 Background

Every year up to eight different teams of students from European universities get an opportunity to take part in Rocket Experiments for University Students (REXUS) program initiated by bilateral cooperation of Swedish National Space Board (SNSB) and German Aerospace Center (DLR). Prospective engineers work on complex mechanical and electrical systems in order to apply earned knowledge, gain skills and conduct successful scientific space experiments [1].

ISAAC (Infrared Spectroscopy to Analyse the middle Atmosphere Composition) is an experimental module designed by KTH students. The main intention for the team was to design, construct and demonstrate a system with two autonomous Free-Falling Units (FFUs), carried inside the Rocket Mounted Unit (RMU), where one FFU can track the other after ejection from the sounding rocket. Each of them consists of a Common Unit (CU), which is inherited from the MUSCAT [2] and the RAIN [3] experiments, and a Specific Unit (SU). The concept of tracking lies in capability of Receiver Specific Unit (RxSU) to fix its position relatively to Transmitter Specific Unit (TxSU) and keep the last within a narrow field of view. Establishing of in a way connection between units makes it reasonable to conduct a scientific experiment. As the secondary objective, definition of profile of carbon dioxide (CO_2) concentration in the middle atmosphere using infrared (IR) spectroscopy method is declared [4].

The ISAAC experiment can be split into six different subsystems from the electronic point of view. The electronics on-board the RMU powers the whole experiment as well as provides communication with FFUs through ground station and schedules ejection. CU is responsible for localization, landing and flight trajectory logging for further reconstruction. Smooth and safe landing is ensured by a parachute deployment system.

After the parachute deploys at the altitude of around 6 km, localization system activates the transmission of corresponding FFU GPS coordinates. Constant experiment positioning is essential for recovery. General purpose of the TxSU is light emission for reaching both experiment goals. It has 20 high power LED sources and eight IR lamps around the periphery. The RxSU, in turn, is divided into a power system and two independent parts, namely the tracking system and the IR spectroscopy system. The power system has six DC/DC converters that provide power for the whole RxSU module. The tracking system gathers and processes data from angular rate sensor, sun sensors and CMOS camera to command two stepper motors. The IR spectroscopy system controls thermoelectric coolers and saves digitalized infrared sensors data to memory for post-flight analysis [4].

The ISAAC was declared as a unique experiment with its controlled ejection and objectives that expect a collaboration of two independent FFUs. However, due to inability to ensure safe flight as well as full operation of the FFUs, a decision to scope down the objectives of the experiment was made in April 2014. The ejection system test was declared as the new goal of the experiment. The rocket was launched on May 29 from Esrange, Kiruna and all objectives of the current thesis work were successfully reached. According to the post-flight analysis based on real-time communication packets, the FFUs were ejected in the correct time window.

1.2 Problem description

The major portion of complex electronics system has been developed in the individual project courses, the group project for System-on-Chip Design (SOC) students and several degree projects. However, several important parts still need to be developed and implemented. The author developed and verified electronics for the RMU including FPGA firmware for ejection algorithm and communication. The ejection algorithm is based on real-time sun sensor and gyroscope data, allowing to eject the FFUs perpendicularly to the Sun position. The communication line shall provide a ground station operator with valuable information during the experiment execution time for pre-flight preparation, monitoring as well as post-flight analysis.

1.3 Method

The method followed in the degree project is applied [5]. The research is based on solving practical problems.

In order to get the whole picture of the experiment, Student Experiment Document (SED) and previously conducted researches were studied during first two weeks. Next week after the pre-study period were assigned for software examination and preparation for off-site testing. On week 6 and week 11, the author was involved in bench test and payload integration test in Germany. The last 2 weeks were reserved for the launch campaign in Esrange. All the remaining time the author was focused on practical part and report writing.

1.4 Delimitation

Since there are only 20 weeks allocated for degree project, the author was focused only on the RMU system. FFUs are not a part of discussion in this Master's Thesis.

1.5 Goal and public welfare

The goal of the degree project is to develop and verify the RMU and ensure a successful launch campaign that implies well-timed ejection of the FFUs and data collection for post-processing.

Other KTH teams in later projects will reuse some of the unique parts of the ISAAC experiment or use them as starting points.

1.6 Outline

Logic structure of the report lies in consistent description of the conducted research. It starts with broad overview of the system under research and finishes with a discussion of obtained results and suggestions for future work.

Chapter 2 describes the RMU system and its requirements.

Chapter 3 is dedicated to development part of the RMU system.

Verification phase is described in details in Chapter 4.

Chapter 5 is devoted to post-flight analysis.

In Chapter 6, the conducted work is discussed, bringing suggestions for future work.

Chapter 7 concludes the report.

Information sources cited in the thesis are shown in Chapter 8.

Chapter 2 RMU system

The RMU is a mechanical module fixed on the rocket. From the electrical point of view, it is an intermediate unit between the rocket and the FFUs, which are carried inside. The electronics on board solves several very important tasks, namely communication, battery charging, data acquisition and ejection's "brain". Although the unit is mostly inherited from previous experiments, radical change in FFUs ejection principle appears in the ISAAC. A block diagram of RMU electrical system is show in Figure 2-1.

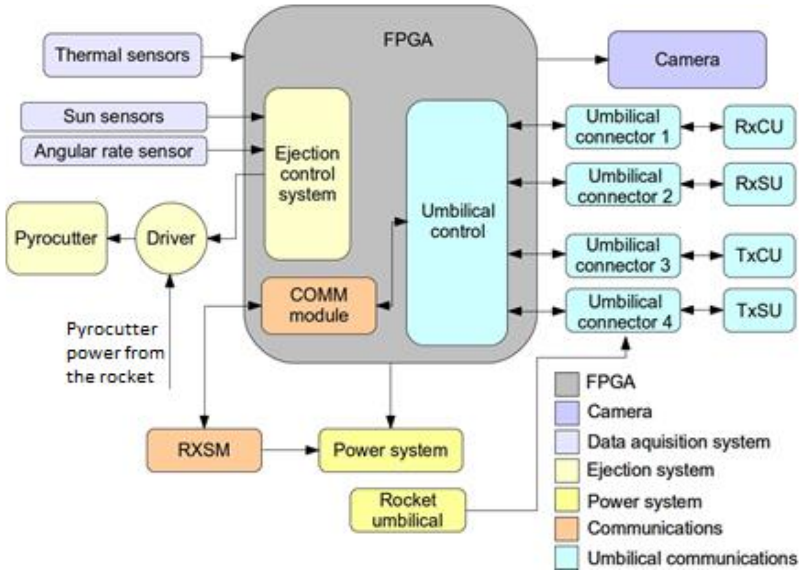


Figure 2-1 RMU electrical system [4, p. 67]

The RMU electronics can be divided into 6 different blocks: data acquisition system, ejection system, power system, camera, communication and umbilical communication. The ejection system is responsible for cutting cables using a pyrocutter. The power for the cutter is provided by the REXUS Service Module (RXSM). A driver and an ejection control system ensure a possibility to control ejection time. The data acquisition system is based on a thermal sensor, sun sensors and an angular rate sensor. Readings of these sensors are processed by the FPGA both for housekeeping and ejection control. The umbilical communication allows to control the FFUs while they are inside the RMU. The camera is in charge of ejection video recording. The communication system allows to monitor and control the experiment from a ground station using downlink and uplink. The power system supplies the whole RMU board as well as charges batteries in the FFUs.

Electrical schematics and a PCB layout was developed by a team of SOC students that the author was part of.

2.1 FPGA

The RMU contains an FPGA that allows the above-mentioned functionality to be implemented. Actel Proasic3 A3P250 is chosen due to its low power consumption, high performance and enough high capacity. It has 250 000 gates, and 3.3V and 1.5V as supply voltage [6]. FPGA is programmed through integrated JTAG interface in Microsemi Libero IDE software using VHDL hardware description language.

The RMU FPGA is in charge of data processing, camera control, ejection scheduling, communication with the FFUs as well as communication with a ground station in real-time mode.

2.2 Communication

Communication in the RMU allows sending commands from a ground station to the FFUs through RXSM. All incoming and outgoing messages are processed in the FPGA and then redirected to FFUs if it is needed.

Communication protocol used in the experiment is RS-422. While the rocket is on the ground, RXSM provides the ground station operator with 38.4 Kbit/s both uplink and downlink, however after liftoff only downlink is available. The example of downlink data structure is shown in Figure 2-2, where SYNC1 and SYNC2 are synchronization bits, MSGID is a message ID, MCNT is a current message number. CSM and CRC are checksum and cyclic redundancy check bits, respectively.

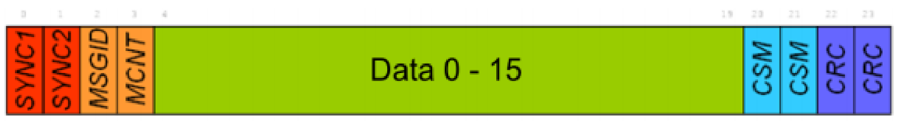


Figure 2-2 Downlink protocol example

Each FFU has two independent umbilical connectors: one is for SU and the other is for CU. The pinout of the umbilical connector is shown in Table 2-1.

Table 2-1 Umbilical connector pinout

Pin №	Pin name	Function
1	ROCKET_PIN	FFU state control (sleep/mission mode)
2	TX	Transmitting channel to RMU
3	RX	Receiving channel from RMU
4	CHARGE	Battery charging
5	GND	Common ground

The RMU continuously provides a ground station operator with real-time data. Communication is essential for pre-flight preparations and post-flight analysis. All flash memories have to be erased and rewound; ejection algorithm execution starts only after manual arming. For the post-flight analysis, communication is the only way to receive stored data from flash memory.

2.3 Data acquisition

Data acquisition system in the ISAAC RMU consists of a temperature sensor, three sun sensors and an angular rate sensor. The last two provide ejection algorithm implemented in the FPGA with necessary input data.

The angular rate sensor L3G4200D is a low-power three-axis gyroscope. It is ultra-stable device with digital output and built-in I^{2}C/SPI communication interfaces. [7]

The sun sensors are SLSD-71N6 planar photodiodes mounted on the skin of the rocket [8]. The distance between three sensors corresponds to 30° angle.

Output of analogue thermal and sun sensors is digitalized with 12-bit MAX11617 analogue-to-digital converter (ADC) with 2.048 V internal reference voltage [9]. The RMU has two identical ADCs: housekeeping and ejection. The housekeeping one digitalizes voltage levels and current consumed by the FFUs as well as temperature inside the RMU module. The second ADC digitalizes outputs of sun sensors. Both ADCs are shown on Figure 2-3.

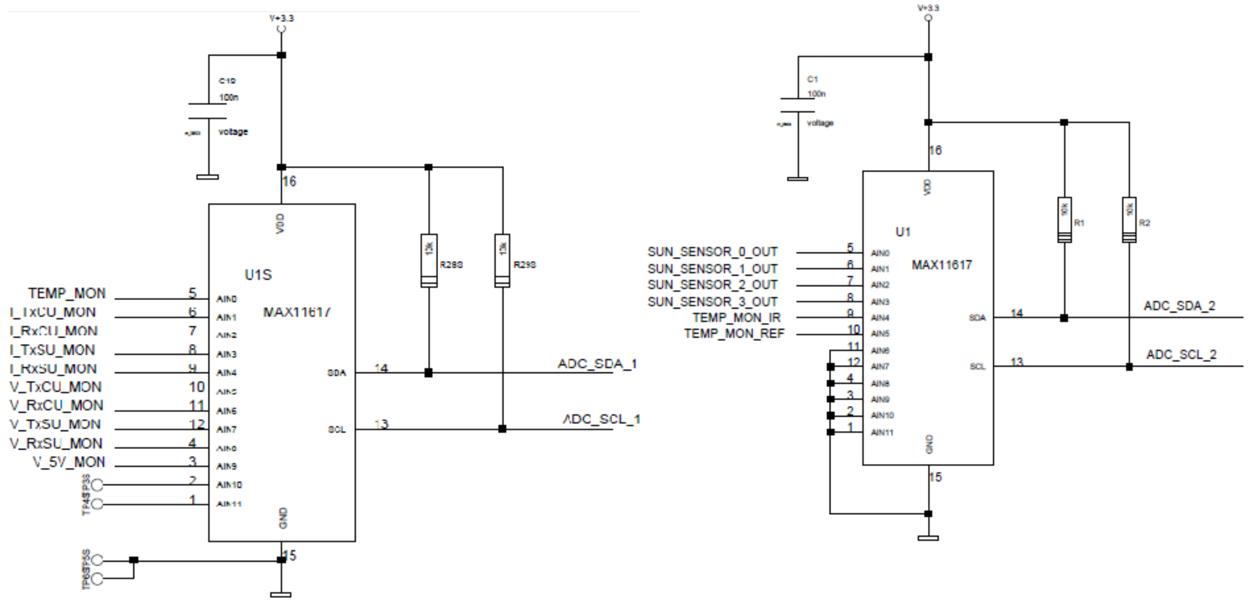


Figure 2-3 Housekeeping (left) and Ejection (right) ADCs electrical schematics

2.4 Camera

The whole flight is recorded with a HD GoPro Hero3 camera. The video will be used later in post-processing to verify the FFUs ejection time and direction relatively to the Sun. Recording function is manually activated by a ground station operator at time t-150s and deactivated automatically at time t+510s, where t is the time of the rocket liftoff. The video will be stored on 32GB microSD card. The RMU camera is mounted on the skin of the rocket inside the module and looks out through a hole. [4]

The camera itself is inherited from the MUSCAT experiment. The battery has been removed and electrical power is provided by the RMU. Moreover, it is fully controlled by the FPGA.

2.5 Ejection system

In this project, presence of voltage in power supply line for pyrocutter is not the only necessary and sufficient condition for ejection of modules. The ejection system can be divided into two major parts: an ejection algorithm and a power driver. The algorithm is being constantly executed producing output after manual arming of the system. The idea of controlled ejection lies in continuous position tracking relatively to the Sun and calculation of a proper instant of time to eject the FFUs. In order to avoid sun influence on the FFU tracking ability, they shall be ejected perpendicularly to the Sun position with allowable deviation of ±20°. Top and side view of the desired ejection is shown in Figure 2-4.

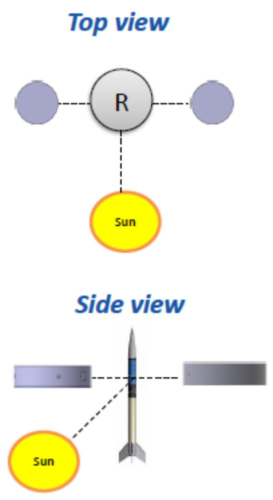


Figure 2-4 Desired ejection direction [4, p.7]

The ejection system is controlled by the RMU and has independent power line from the rocket, which is activated by a flight operator at time t + 90 s, where t is the time of liftoff. The power driver represents an electrical switch that consists of two MOSFET transistors. Electrical schematic of the power driver is shown in Figure 2-5.

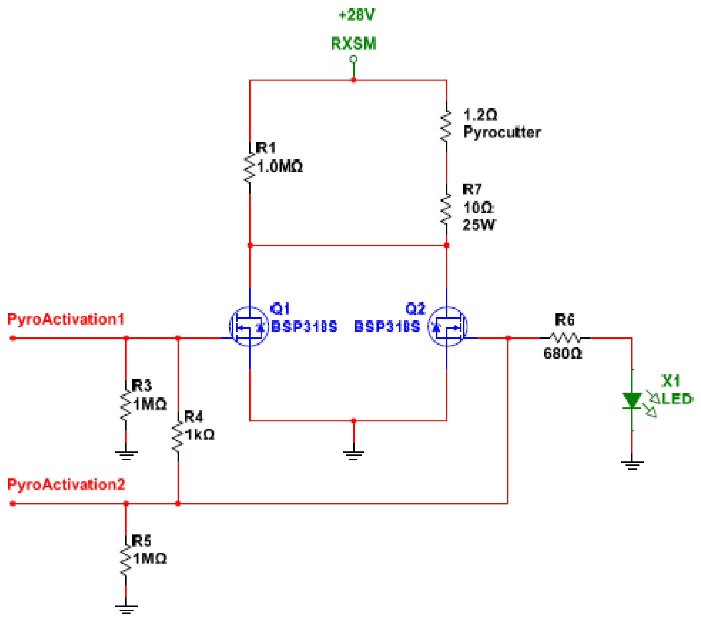


Figure 2-5 Ejection system power driver

Gates of N-channel SIPMOS BSP318S are connected to the FPGA output pins. The chosen transistors provide enough current (over 1.2 A) to a pyrocutter with V_{GS} voltage of 3V at high temperatures [10]. The pyrocutter has 1.2 Ω resistance [11] and it is connected in series with high power 10Ω resistor in order to limit current.

Chapter 3 RMU development

The complexity of the multitasking PCB demands different approaches. The design flow of the electronic system on board the RMU includes both hardware and firmware development.

The RMU hardware consists of four PCBs, where three of them are located on the rocket skin with photodiodes mounted on their surfaces. The main RMU board has 291 components, excluding wires and connectors. Careful soldering is very important for tiny components' operation under space conditions. All components, except angular rate sensor, were soldered manually in KTH SPP students lab. The gyroscope package requires BGA soldering technique and therefore it was mounted by the PCB manufacturer.

3.1 Sun sensors

In order to calculate relative position of the Sun, sun sensors are used. They provide the FPGA with real-time illumination intensity. Figure 3-1 shows top and bottom view of a soldered PCB.

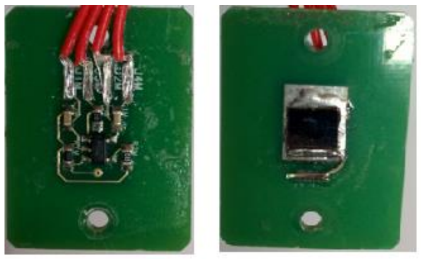


Figure 3-1 Sun sensor PCB

Current, generated by SLSD-71N6 planar photodiode, changes its value proportionally to light density, what in turn changes voltage drop across a resistor. Before sending analogue voltage to ADC, it is being amplified. Figure 3-2 shows the schematic for light to voltage conversion.

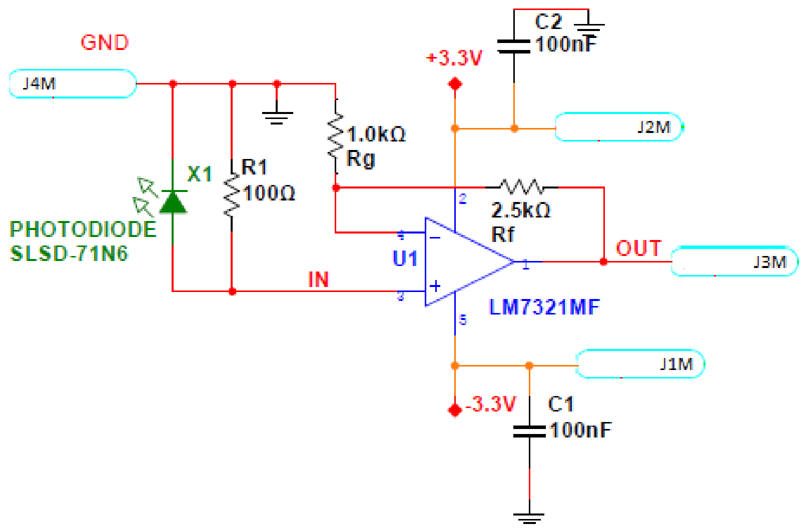


Figure 3-2 Electrical schematic for sun sensor PCB

For signal amplification, LM7321 dual-rail operational amplifier is used [13]. +3.3V and -3.3V supply voltages are provided by the RMU.

Optimal gain should match sun sensor output voltage to the ADC range. The internal reference voltage of the ADC is 2.048 V, which limits the gain. Sun sensor output voltage higher than the ADC reference will saturate it. Applying equation below [12], the optimal gain value of 3.5 is obtained, which is further verified empirically.

$A_V = 1 + \frac{R_f}{R_g}$, where

R_f – feedback resistor [Ω]

R_g – gain set resistor [Ω]

A_v – operational amplifier gain

Sun sensor gain testing was done in sunny and serene day within the precincts of the KTH Campus. All three independent PCBs were powered with batteries and output voltage was measured with digital multimeter (DMM). The maximum indication was 1.1 V, what gives us approximately 1V safety margin.

3.2 Ejection algorithm

Ejection control is one of unique features of the ISAAC. Perpendicular to the Sun position ejection requires continuous data processing in real-time mode. Several approaches of implementation were investigated and analyzed.

Approach 1: In this method, the only source of information are sun sensors. Searching for maximum intensity values, the Sun direction can be easily found. Moreover, observation of more than one rotation allows to determine rotation frequency of the rocket. This method uses fewer sources of data; however, the complexity of firmware code is higher due to necessity of implementation of parallel division. The FPGA space is limited with 250 000 gates and this is the reason why parallel division of two variables cannot be implemented in the RMU. Besides that, requirements fulfilment cannot be guaranteed because only approximate values can be found. The allowable maximum value of deviation is ±20°.

Approach 2: In this method, both light intensity and angular speed are processed from independent sources. The implementation is much easier, occupies less than 50% of FPGA space and results in calculations that are more precise. Division is used to process angular velocity, which do not require high processing speed and therefore can be implemented using sequential division. This method is chosen to be implemented in the final version of the firmware.

The ejection algorithm consists of seven concurrent processes: sun sensors data receiving, gyroscope data receiving, counter, ejection timeout, sun sensors data processing, gyroscope data processing and ejection activation. The last three processes are implement as Moore finite state machines. The code for ejection algorithm is attached to Appendix A. Further discussion analyzes all processes independently. Figure 3-3 shows the structure of the ejection algorithm and an interaction of internal processes.

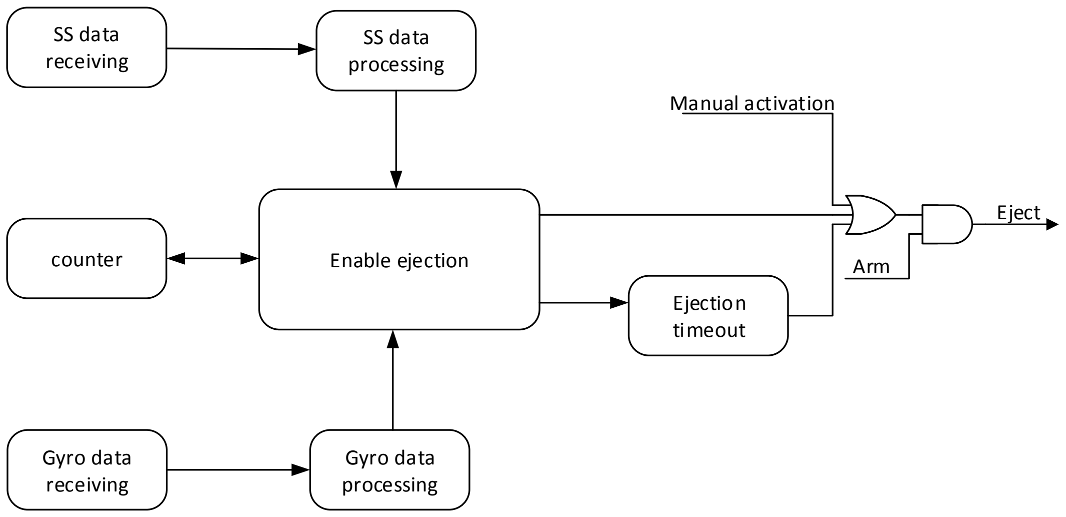


Figure 3-3 Ejection algorithm block diagram

3.2.1 Data receiving processes

Sun sensors data receiving and gyroscope data receiving processes have identical purposes. Both ejection ADC and gyroscope have their own controllers that output readings bytewise. The processes are responsible for data collection and packing into 64-bit signals for further processing. This is done by shifting in bytes of information when new data is ready.

3.2.2 Timing

In the ejection algorithm, two different systems of time are in use simultaneously, namely absolute and relative time. The absolute one starts counting when the FPGA is reset. It is represented with 21 bit vector, where the least significant bit (LSB) is changed every 0.5 ms. The other bits change their values with the frequencies shown in Table 3-1.

Table 3-1 Global time vector

Bit position	Frequency
GLOBAL_TIME[0]	2 kHz
GLOBAL_TIME[1]	1 kHz
GLOBAL_TIME[2]	0.5 kHz
...	
GLOBAL_TIME[11]	1 Hz
...	
GLOBAL_TIME[20]	2 mHz

Counter is a process that determines relative time between positions with the highest light intensity. In other words, it counts the time for one 360° rotation. It is triggered every half a millisecond and on a special signal from sun sensors data handling process, which occurs when a new maximum intensity is defined. After every new maximum value found the counter is reset.

3.2.3 Sun sensors data handling process

Direction of the ejection is tightly coupled with the Sun position. Sun sensors provide the FPGA with light intensity in real-time mode, what makes it difficult to determine a peak value on fly. This problem can be solved with introducing a margin, after which the attitude can be stated as a sun-directed. However, exact light intensity depends on many factors and is unknown, what, in turn injects additional sources of errors. For that reason, the following concept is implemented. Incoming new data is compared with a value stored in a variable, which is initialized with zero on the FPGA reset. If the new data is smaller than the data stored in the variable, the algorithm waits for next sun sensors data. Otherwise, the variable value is updated with the largest one. If the value of the variable has not been updated for last t ms, where t = T / 2 + 20 ms, then it corresponds to the maximum intensity value and in that moment the middle sun sensor was pointing onto the Sun. Adding the values from all three sources simplifies the implementation. In this case, a peak readout occurs when the source is straightly opposite the middle photodiode.

All three sun sensors readings are shown in Figure 3-4. Figure 3-5 represents the ADC data while the RMU was rotating on constant 3.05 Hz frequency on rotary table. The curves overlap without any glitch in between, what allows processing only a combined value instead of single readings. The resulting curve of the combined value is shown in Figure 3-6.

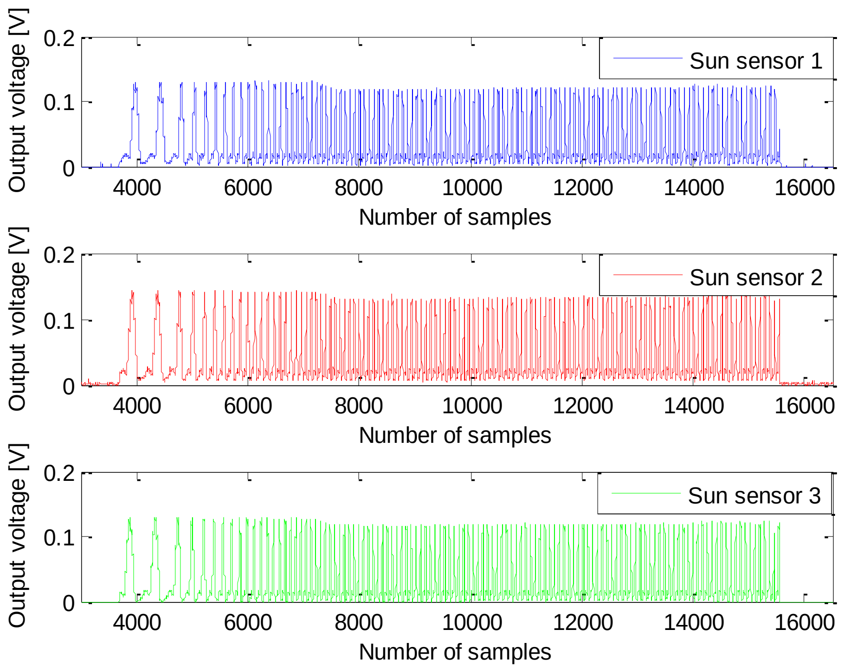


Figure 3-4 Sun sensors readings separately

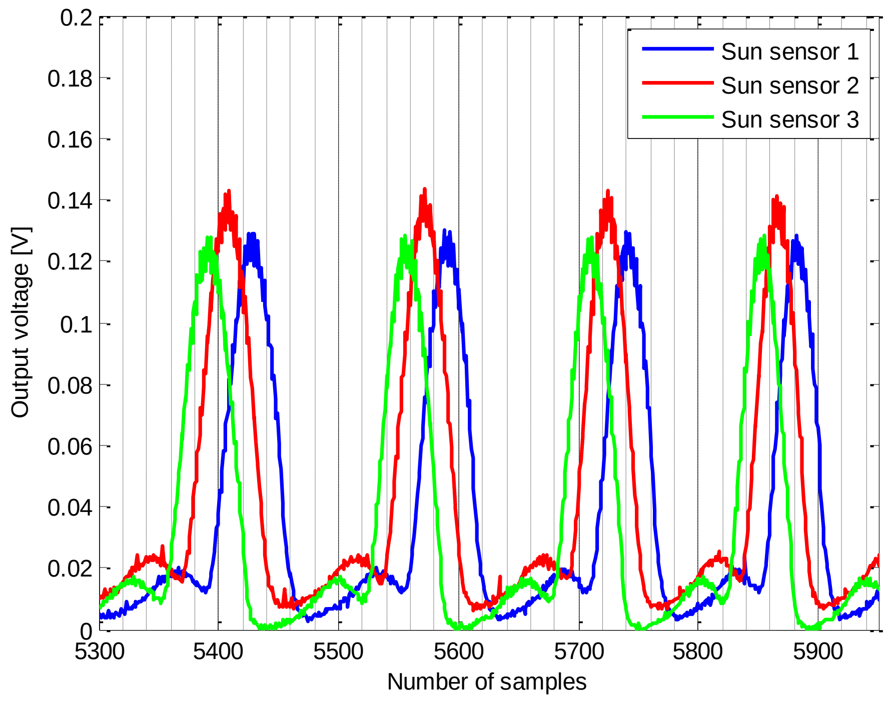


Figure 3-5 Sun sensors readings

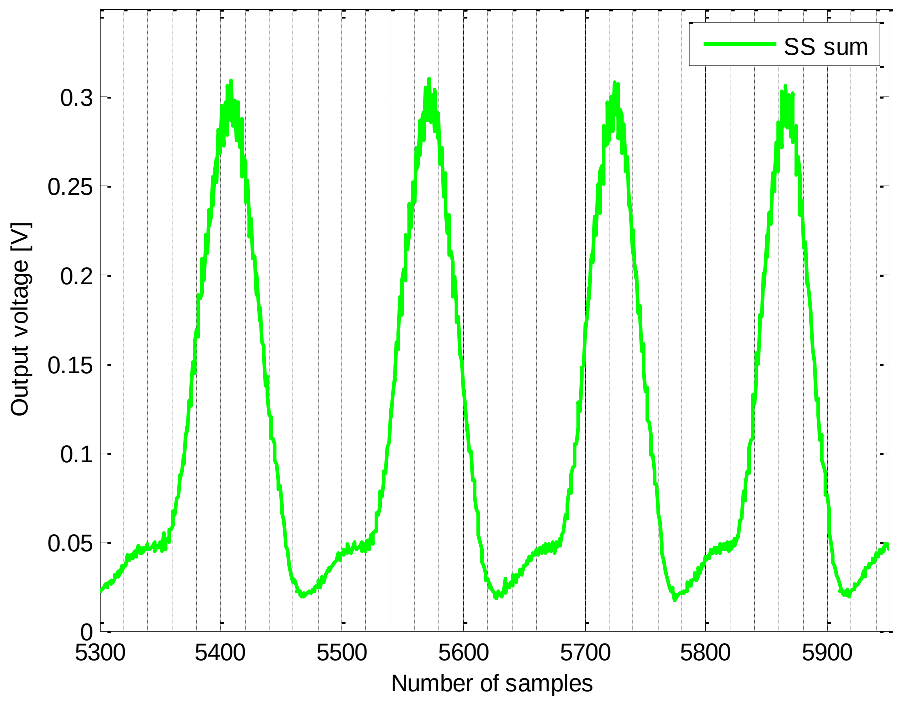


Figure 3-6 Sum of readings

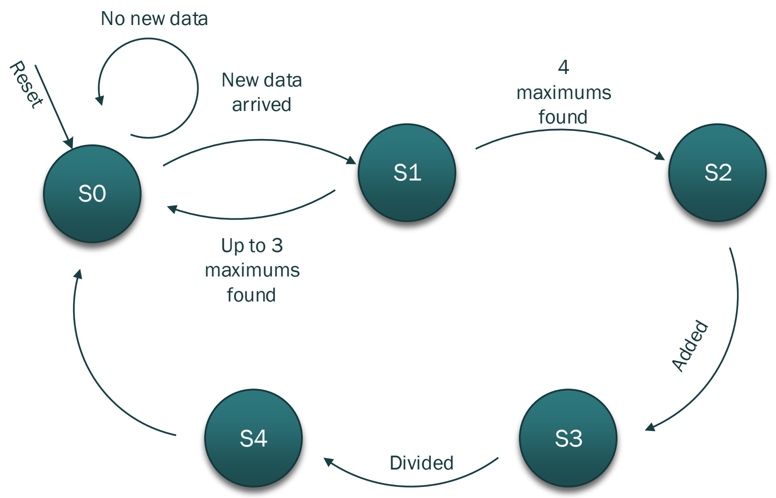


Figure 3-7 Sun sensors data handling finite state machine

Every rocket rotation relative time of the determined maximum intensity is stored in a 1x4 array. These values are used for mean time calculation and more accurate time of the next peak intensity prediction.

A finite state machine for the above-discussed process, namely sun sensors data processing, is shown in Figure 3-7 and includes the following states:

S0: The state is triggered on every rising edge of the clock signal unless new data with a valid header is arrived. Upon arrival, all three single sun sensor readings are summed up and the process moves to the next state.

S1: If the new data has higher value than the one stored before, maximum value and its absolute time of occurrence is overwritten; otherwise, if no higher value has appeared during the time needed to pass 180° plus 20 ms since last stored maximum, the relative time of its appearance is stored in the array. The process starts from the beginning (S0) unless all four maximum values are found. After the fourth consecutive maximum intensity is defined, the process moves to the next state (S2).

S2: In this state first stage of mean time of 360° rotation is calculated. All four values from the time array are summed up.

S3: Result from S2 is divided by four. Division is implemented using bit shifting. Two bits are shifted to the right.

S4: The final state ensures mean time to be calculated and gives a flag to ejection activation process.

3.2.4 Gyroscope data handling process

All three sun sensors are located on the same side of the mechanical module as the lower FFU and according to the requirements, the FFUs shall be ejected perpendicularly to the Sun position. In order to fulfil the condition, time of 90° rotation is calculated using angular rate sensor readings. Figure 3-8 represents a finite state machine for gyroscope data processing.

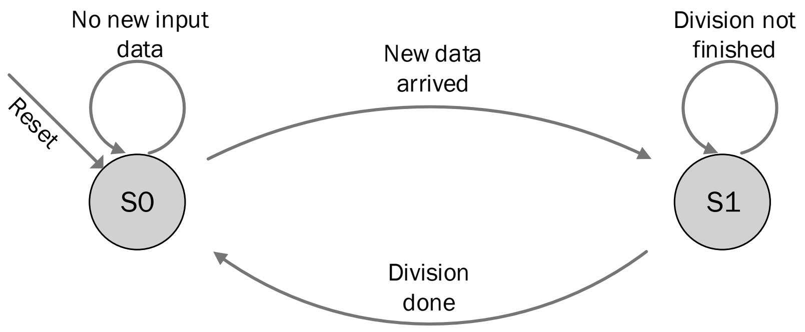


Figure 3-8 Gyroscope data processing finite state machine

S0: When new valid data arrives, it is converted from two's complement to binary, preparations for division are done and the process goes to the next state. Otherwise, waits for new data.

S1: Division procedure is being done in this state. It is triggered 16 times. Time needed for 90° rotation is stored in a variable. Process continues with S0.

Output of the angular rate sensor is given in degrees per second (dps). Taking into account a sensitivity factor of 70 mdps/digit, desired value is a ratio of 1285714 and the gyroscope actual reading.

Division is implemented in sequential way by using a simple comparator and takes 16 clock cycles. The dividend is stored in a (n+k)-bit temporary buffer, where n is number of bits in a dividend and k is number of bits in a divisor. Every clock cycle buffer[(n+k-2) : (k-1)] is compared with the divisor. If the divisor is greater than the compared part of the buffer, the buffer is shifted one bit to the left. Otherwise, the buffer(n+k-1) equals to '0' and the buffer[(n+k-2) : k] equals to a difference of the buffer[(n+k-3) : (k-1)] and the divisor[(k-2) : 0]; '1' is shifted in to the right of the buffer[(k-1) : 0]. Current procedure is repeated k times. At clock cycle k the solution equals to buffer[(k-1) : 0] and the remainder equals to buffer[(n+k-2) : k].

The division example is shown in Table 3-2, where the dividend is 14 and the divisor is 5. The part of the buffer vector that is compared with the divisor is highlighted with red color. The divisor is a 4 bit vector in this example (k = 4), thus the whole procedure takes 4 clock cycles. At clock cycle number 4, the first 4 bits of the buffer correspond to the solution and the next three correspond to the remainder. The solution and the remainder for the given example is highlighted with blue and brown colors, respectively. Cycle 0 is used for preparation, namely shifting the dividend into the buffer vector and resetting counter.

Table 3-2 Division example

CLK cycle	Buffer	Divisor
0	0000 1110	0101
1	0001 1100	0101
2	0011 1000	0101
3	0010 0001	0101
4	0100 0010	0101

3.2.5 Ejection activation process

Ejection signal activation process starts execution only when sun sensors data handling process is in the final state (S4), which is called a flag state. After receiving a flag signal, ejection time is calculated taking into account mean time of one 360° rotation, mechanical and electrical delays as well as time of 90° rotation. Firing signal activates as soon as the calculated condition is met and the signal is kept high for 12 ms. A finite state machine for the ejection activation process is shown in Figure 3-9.

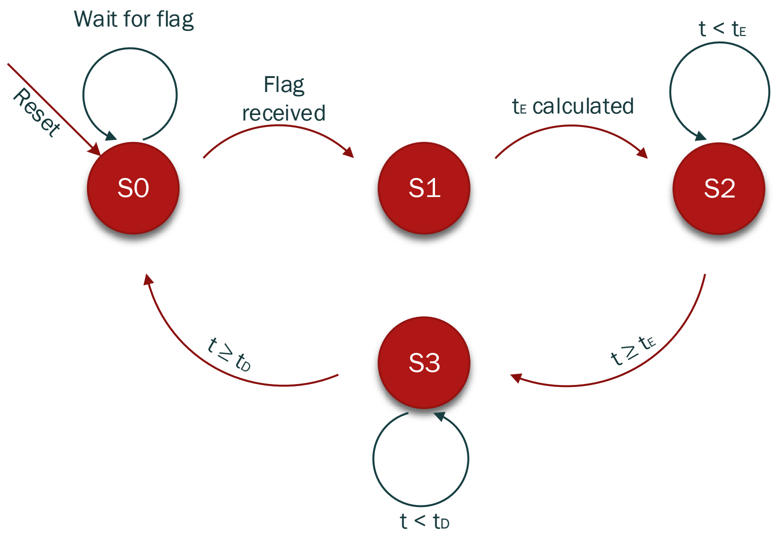


Figure 3-9 Ejection activation process finite state machine

The finite state machine has the following states:

S0: The entire process stores absolute time of the last maximum intensity and goes to the next state if sun sensors data handling process is in the final state (S3).

S1: Ejection time is calculated and state is changed to S2.

S2: If the absolute time is smaller than the calculated ejection time, then the ejection signal is low. Otherwise, it goes high, deactivation condition is calculated and state is changed to S3.

S3: The ejection signal is kept high until deactivation condition is met. When the absolute time reaches deactivation time, the ejection signal is forced to zero and state is changed to S0.

3.2.6 Ejection timeout process

Ejection is a very critical part of the experiment and it shall happen in any case even if some of the supporting parts as gyroscope, ADC or sun sensors malfunction. In order to ensure ejection of the FFUs, timeout procedure is implemented in the algorithm. The timeout signal activates every 4 s and stays high for 1 s. However, it is a backup signal and therefore, the activation condition is recalculated every time when the signal from the ejection activation process goes high, postponing timeout for another 4 s.

Undesirable ejection during testing procedures can lead to damages in the RMU module as well as in the FFUs. To avoid any accident, manual arming of the ejection is implemented as a safety measure. This is done by injecting an internal signal, which can be controlled only by an operator through a communication line. The resulting output signal of the whole algorithm is produced according to the following logic equation:

$S = (S_{EJ} \vee S_{TO} \vee S_{M}) \wedge S_{ARM}$, where

S_{EJ} – signal from enable ejection process
S_{TO} – timeout ejection
S_{M} – manual activation signal
S_{ARM} – manual arm signal
S – resulting output of the ejection block
$\vee$ - logic OR operator
$\wedge$ - logic AND operator

The manual activation signal is implemented for early stage verification purpose and measurement of electrical parameters of the circuit.

3.3 Communication

Some of the RMU functionalities require direct involvement of a ground station operator. During the final countdown preparations, camera mounted on the skin of the rocket shall be properly switched on and ejection system must be armed. Moreover, downlink provided by the RXSM makes it possible to monitor all the important parameters of the experiment both before and after liftoff.

The communication between ground station and the experiment is divided into two different sections: the RMU and the FFUs communication, respectively. Independently of a type, all messages are processed by the ISAAC RMU Controller, which is partially inherited from the MUSCAT experiment.

3.3.1 RMU communication

The RMU continuously sends housekeeping data packets to the ground station, which contain the most important data and allow monitoring the whole experiment's performance. Each 26 bytes real-time message has the structure shown in Table 3-3, where the HEADER is "#0" and it represents that the message contains the RMU related data.

Table 3-3 Real-time message structure

Real-time message					
HEADER	TIME	Binary zeroes	Status	HK data 1	HK data 2
16 bits	28 bits	24 bits	12 bits	64 bits	64 bits

Post-flight analysis requires careful data conversion and plotting all the readings versus time. This becomes possible due to TIME vector contained in the real-time message. The vector consists of three parts: 2-bit header, 18-bit vector for seconds since last FPGA reset and 8 bits for M_TIME [25:18] time vector. Changing frequencies of the last one are shown in Table 3-5 and the structure of the TIME vector is shown in Table 3-4.

Table 3-4 Time vector structure

Time vector		
Header	Seconds since reset	Time within seconds
2 bits	18 bits	8 bits

Table 3-5 "Time within a second" vector

Bit number	Changing frequency
M_TIME [18]	128 Hz
M_TIME [19]	64 Hz
…	
M_TIME [23]	4 Hz
M_TIME [25]	1 Hz

Another important part of the real-time message is a status vector, which gives an overview of events happening in the FPGA. It consists of 12 bits, where every single bit is related to a specific event. The list of the events with corresponding bit number is shown in Table 3-6.

Table 3-6 Status vector structure

Bit	Event
STATUS_VECTOR[0]	Rotator
STATUS_VECTOR[1]	Maximum
STATUS_VECTOR[2]	Fire
STATUS_VECTOR[3]	Enable ejection
STATUS_VECTOR[4]	Camera button 1
STATUS_VECTOR[5]	Camera button 2
STATUS_VECTOR[6]	Camera power ON
STATUS_VECTOR[7]	Recording
STATUS_VECTOR[8]	RxCU sleep
STATUS_VECTOR[9]	RxSU sleep
STATUS_VECTOR[10]	TxCU sleep
STATUS_VECTOR[11]	TxSU sleep

The RMU sends real-time messages only in Rotator mode. The FPGA sends its data and asks for real-time data from every FFU by turn. The experiment is in rotator mode by default, however it can be deactivated by sending " ‘ " character. STATUS_VECTOR[1] changes its state when a new maximum intensity of the light is found. According to the ejection algorithm, it toggles in $t = t_{½} + 20$ [ms] after the maximum intensity is captured by photodiodes, where $t_{½}$ corresponds to the time needed to pass 180°. Fire event indicates the signal driven to the ejection system's power driver. In case of manual activation, character "P" shall be sent to the controller. Enable ejection shows the ejection arming status. By sending "E" character, the experiment can be armed or disarmed. Next 4 bits are related to camera control. The camera has two buttons and they are "pressed" automatically by an algorithm in this case. Moreover, all of these camera single events can be toggled by sending "*", "+", "," and "-", respectively. The ability of full control is very important taking into account that the camera is modified especially for the experiment. It has no battery and it is impossible to press buttons on the camera itself. In case of improper shutdown, the camera control algorithm will be terminated in a wrong state, what may lead to malfunctions. The last four bits indicate sleep/awake statuses of the FFUs and "#", "$", "!" and " " " are the characters for switching state given in order as provided by the status vector.

The third important part of the real-time message is housekeeping data. The ground station operator receives all crucial readings provided by the ADCs and the angular rate sensor before and during the experiment execution in order to monitor correctness and readiness of the system.

Housekeeping data packets consist of information from several sources, which is sampled at different time slots. The housekeeping ADC digitalizes all FFUs' voltages and currents, what becomes uninteresting for the operator due to a down scope in the ISAAC experiment objectives. In order to receive only valuable information and simplify monitoring process, *data_sender* block was developed. Full VHDL code for the block is attached to Appendix B.

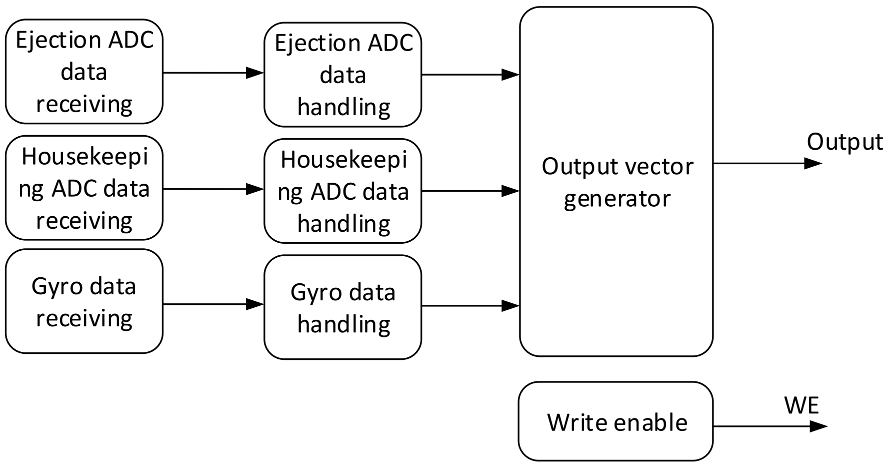

Ejection ADC data receiving
Housekeeping ADC data receiving
Gyro data receiving
Ejection ADC data handling
Housekeeping ADC data handling
Gyro data handling
Output vector generator
Output
Write enable
WE

Figure 3-10 Data_sender: interaction of internal processes

Figure 3-10 illustrates the interaction of internal processes in the data_sender block. Full packages coming from the ADCs and the gyroscope are processed individually and necessary parts are stored in intermediate signals. The output vector generating process selectively combines intermediate signals into one 48-bit signal, which is being output adding a header in the beginning. The header for the packets has two different values: "01BE" and "02BE" for the first and the second housekeeping data packet, respectively.

Housekeeping data packet 1 contains sun sensors readings and its structure is shown in Table 3-7. The structure of the housekeeping data packet 2 is shown in Table 3-8. Data provided by both the ejection ADC and the gyroscope are combined in this packet.

Table 3-7 Housekeeping data packet 1

HK data 1				
Header	Binary zeroes	SS1	SS2	SS3
16 bits	12 bits	12 bits	12 bits	12 bits

Table 3-8 Housekeeping data packet 2

HK data 2				
Header	Temperature sensor	1.5V	Gyro temp	Gyro Z axis
16 bits	12 bits	12 bits	8 bits	16 bits

Table 3-9 illustrates an example real-time message received from the RMU.

Table 3-9 Real-time message example

#0	400362C	000000	F43	01BE00012516014D	02BEAEABB8120001
Header	Time		Status vector	Housekeeping data 1	Housekeeping data 2

3.3.2 General FFU communication

The ISAAC experiment has two different-purpose FFUs with identical CUs and different SUs. The TxSU is fully autonomous, however both CUs and the RxSU require communication lines for various operations with memories, status monitoring and flight preparations. FFU commands sent to the RMU are being processed by the ISAAC RMU Controller as it was in the previous case, however switching blocks are needed to redirect messages to the proper FFU. 3-to-1 multiplexer is implemented for a transmitting line and 1-to-3 demultiplexer for a receiving line. Channel selection is done by sending special characters to the controller, e.g. “1” for the TxCU, “2” for the RxCU, “3” for the RxSU and “0” for the RMU. After choosing a proper addressee, an internal communication line is redirected to its FPGA pins. The VHDL code for communication switches is attached to Appendix C.

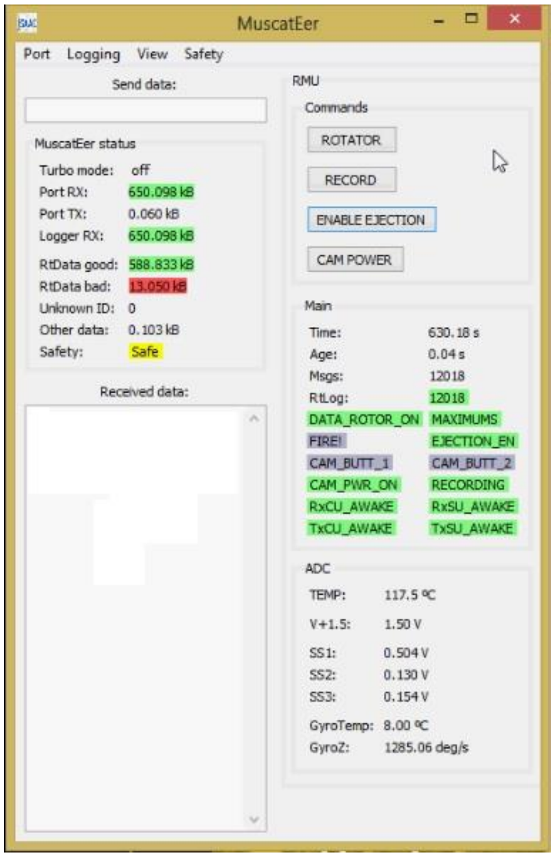
3.3.3 RxSU communication

Although the RxSU has a complex structure with four different FPGAs, communication with it is done through a single line. A special switch implemented in the main FPGA, which is responsible for the FFU sun sensors data processing, ensures accessibility of every single FPGA. The main FPGA permanently listens to the communication line and redirects messages if one of the following ASCII characters appear: “%” for the sun sensors FPGA, “&” for the camera FPGA, “(” for the SmartFusion2 and “)” for the IR FPGA.

The implementation is separated into two blocks: a communication controller and a switch itself. The code for these blocks is brought in Appendix D. The switch is made in a multiplexer style, where the communication controller drives a selecting signal. The purpose of the switch is to internally connect proper FPGA pins with each other and pull-up unused ones.

3.3.4 GUI

Real-time communication with a ground station is done through serial port and it is nothing but a stream of packets, represented in hex format. Due to difficulties in following the readings using terminal, graphical user interface (GUI) Muscateer is used. The GUI is fully inherited from the MUSCAT experiment with some minor changes.



MuscatEer status

Turbo mode:	off
Port RX:	650.098 kB
Port TX:	0.060 kB
Logger RX:	650.098 kB
RtData good:	588.833 kB
RtData bad:	13.050 kB
Unknown ID:	0
Other data:	0.103 kB
Safety:	Safe

Main

Time:	630.18 s
Age:	0.04 s
Msgs:	12018
RtLog:	12018

ADC

TEMP:	117.5 ºC
V+1.5:	1.50 V
SS1:	0.504 V
SS2:	0.130 V
SS3:	0.154 V
GyroTemp:	8.00 ºC
GyroZ:	1285.06 deg/s

Figure 3-11 Ground station GUI

It is written in C++ programming language and adapted to the ISAAC experiment, because of distinctions in real-time message structure. Besides that, the ISAAC messages include some data, which are not compatible with the type declared in the initial version of the software. The angular rate sensor's data are represented in two's complement format and, thus, can have both positive and negative values. The adapted version has buttons for pre-flight preparations to minimize any accidental mistake occurrence, visualization for new statuses and proper conversion for raw data.

Chapter 4 RMU verification

Any development is always followed by a verification phase. In this section, verification process of the RMU system is discussed. Depending on the part under test, verification is done using different methods. Some of them require simulations, other just a communication line or even a high definition camera.

4.1 Communication

The RMU PCB does not have any memory and this makes the communication line extremely important for the experiment. The main source of information for post-flight analysis is the messages received through downlink.

The verification procedure examines proper functionality of not only UART communication blocks implemented in the FPGA, but also the ADCs, the angular rate sensor as well as the ISAAC RMU Controller block. Only having all the above-mentioned blocks correctly functioning, the expected and the received data can match.

In order to obtain better resolution of diagrams, communication baud rate is increased to 115.2 Kbit/s for testing purpose. The ejection arm and the eject signals are illustrated in Figure 4-1. The diagram clearly shows proper functionality of both downlink and uplink. The eject signal appears only when the ejection algorithm is manually activated by the operator. The arm signal changes its value when the RMU receives “E” command from the ground station. Horizontal axis of the graph represents time after the FPGA reset.

Example log file is shown in Figure 4-2, where each line consists of one real-time message.

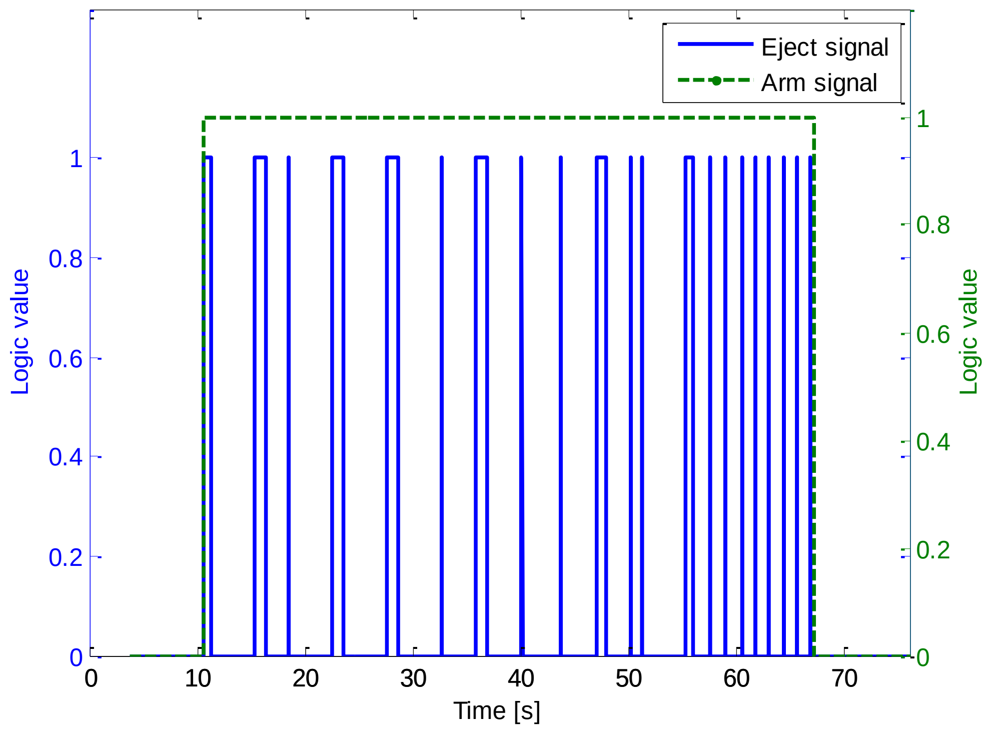
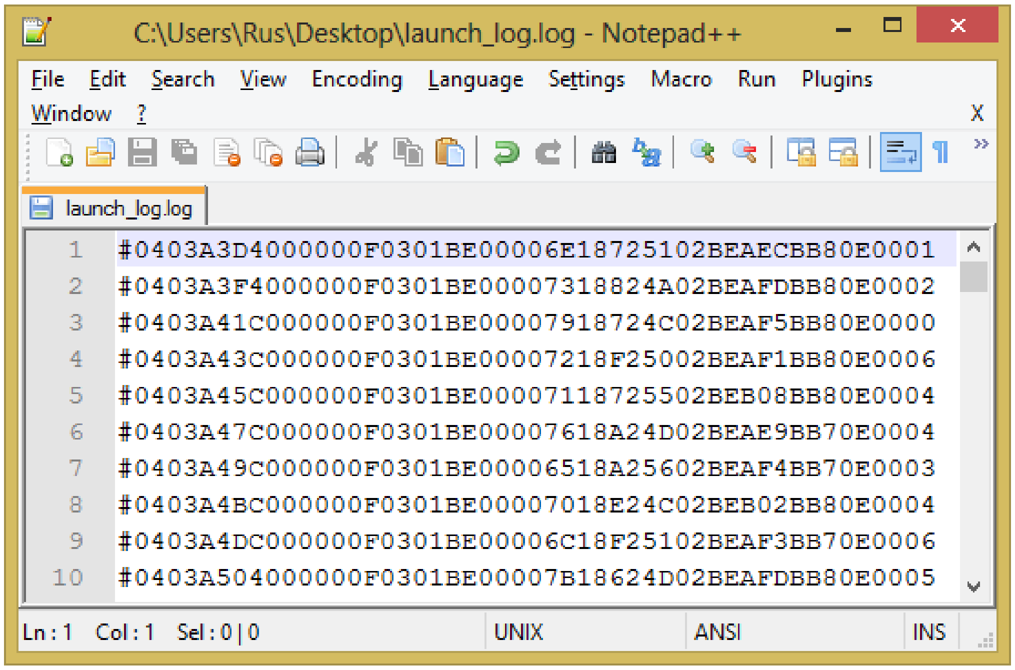


Figure 4-1 Uplink and downlink test


C:\Users\Rus\Desktop\launch_log.log - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins
Window ?
launch_log.log
Ln : 1 Col : 1 Sel : 0 | 0
UNIX
ANSI
INS

```
#0403A3D4000000F0301BE00006E18725102BEAECBB80E0001
#0403A3F4000000F0301BE00007318824A02BEAFDBB80E0002
#0403A41C000000F0301BE00007918724C02BEAF5BB80E0000
#0403A43C000000F0301BE00007218F25002BEAF1BB80E0006
#0403A45C000000F0301BE00007118725502BEB08BB80E0004
#0403A47C000000F0301BE00007618A24D02BEAE9BB70E0004
#0403A49C000000F0301BE00006518A25602BEAF4BB70E0003
#0403A4BC000000F0301BE00007018E24C02BEB02BB80E0004
#0403A4DC000000F0301BE00006C18F25102BEAF3BB70E0006
#0403A504000000F0301BE00007B18624D02BEAFDBB80E0005
```

Figure 4-2 Log file

4.2 Camera

The GoPro Hero 3 camera is an independent device and its testing method lies in remote activation, record starting and automatic video saving after 660 s. Although it can be fully controlled through uplink, it is very difficult to perform remotely. In order to avoid any unexpected behavior, the camera was tested and critical cases, which can lead to malfunction, were discovered.

Figure 4-3 illustrates camera signals and their behavior according to the recording algorithm. Recording process lasts exactly 660 s, starting from t = 110 s and finishing at t = 770 s. However, the camera is not switched off, but recording is stopped, video is saved and a new one started. At t = 860 s the camera was turned off by the operator.

Incorrect shutdown inevitably leads to malfunction next time the camera will be used. Simple and effective remedy to have safe recording requires additional powering on the camera before any recording process. The principle is shown on the graph. The camera is switched on at t = 0 s and switched off at t = 80 s.

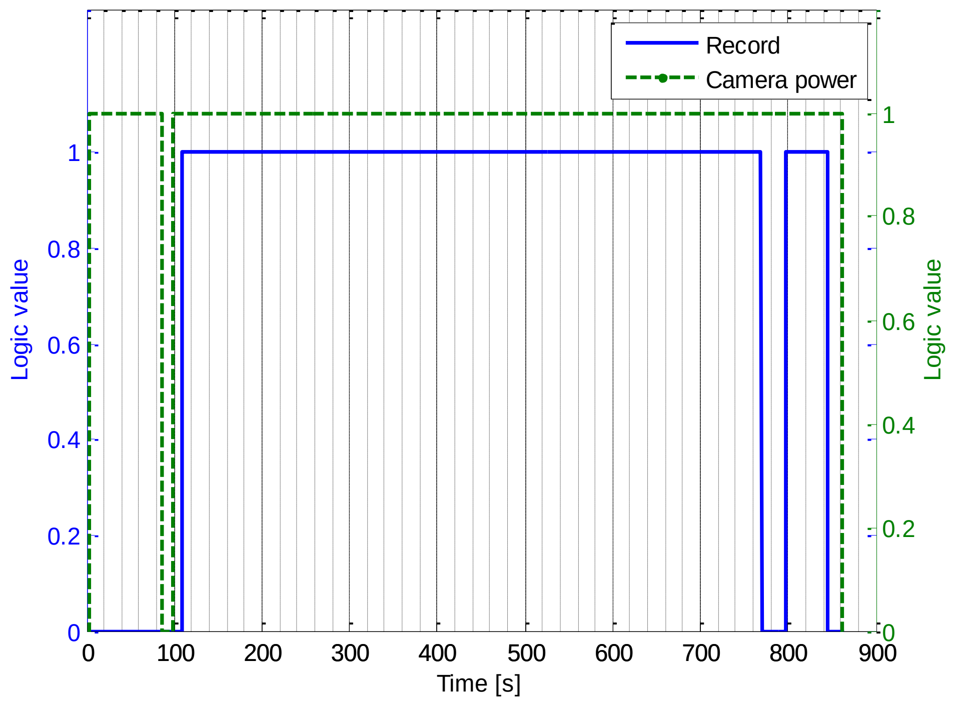


Figure 4-3 Camera signals

4.3 Ejection algorithm

Complex structure of the ejection algorithm requires several different approaches to verify completely the whole block. Verification process of all key blocks is discussed in this section. The angular rate sensor data is processed using mathematical division, which is verified separately.

Flight conditions for the RMU are simulated using a rotary table with adjustable rotational speed. To communicate with the rotating PCB is difficult having cables connected to the ground station. Solution was found in Wireless Bluetooth RF Transceiver Module serial RS232 TTL HC-05 [14].

4.3.1 Division

The implemented division block using VHDL hardware description language was verified using ModelSim simulation. Obtained waveforms are shown in Figure 4-4.

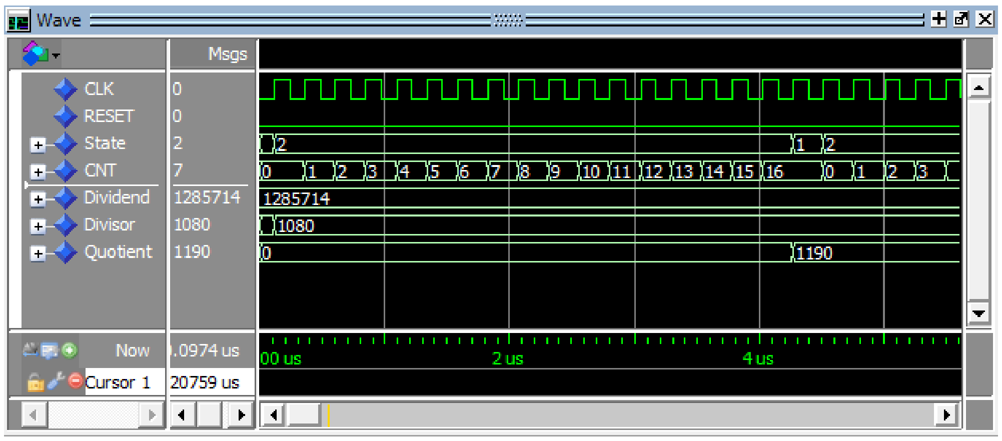


Figure 4-4 Division block simulation

All numbers are chosen as close as possible to the realistic ones. The dividend has the same value as in the ejection algorithm. Although the divisor will not be constant during the flight, the approximate rotation frequency is stated to be 3 Hz, which, in turn, is 1080 dps. Due to the fact that all terms of division belong to integer set, the fraction is omitted, what injects an additional error. In the examined example, the fraction value is 0.47(592), which equals to 0.514 ms or 0.56° of rotation.

4.3.2 Angular rate sensor readings

The gyroscope functionality is verified on the rotary table. A peculiarity of the installation is inability of drastic acceleration. Figure 4-5 shows the angular rate sensor's output during the test.

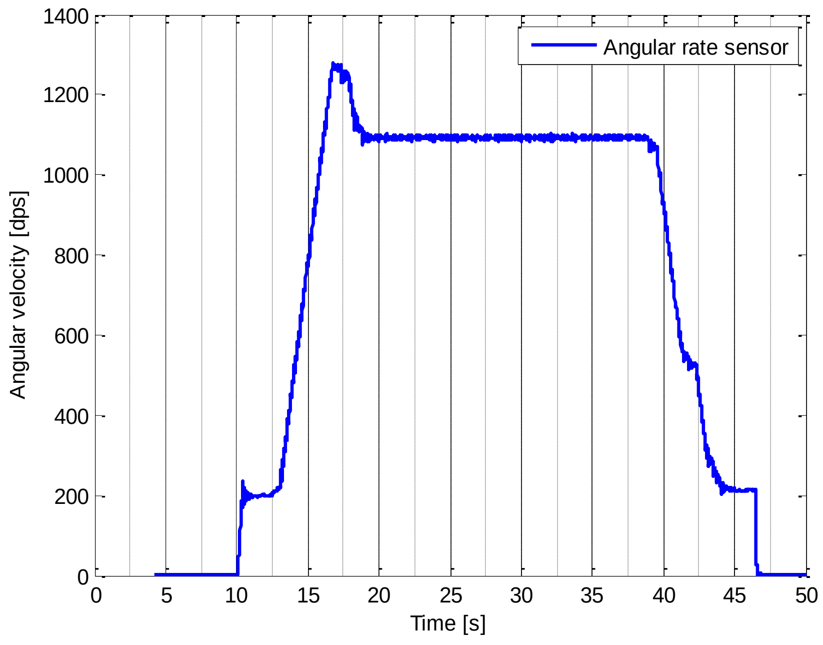


Figure 4-5 Gyroscope data

The rotary table was smoothly sped up to 1300 dps for a short time and then rotation speed was held at constant 3 Hz level for about 20 s. The output precisely follows the test bench's performance.

4.3.3 Sun sensors readings

The sun sensors output voltage is represented in Figure 4-6. The given graph is based on communication data received while the RMU was rotating at constant 3 Hz frequency. All three waveforms overlap, which allows to process a combination of them instead of single readings, as it was mentioned before.

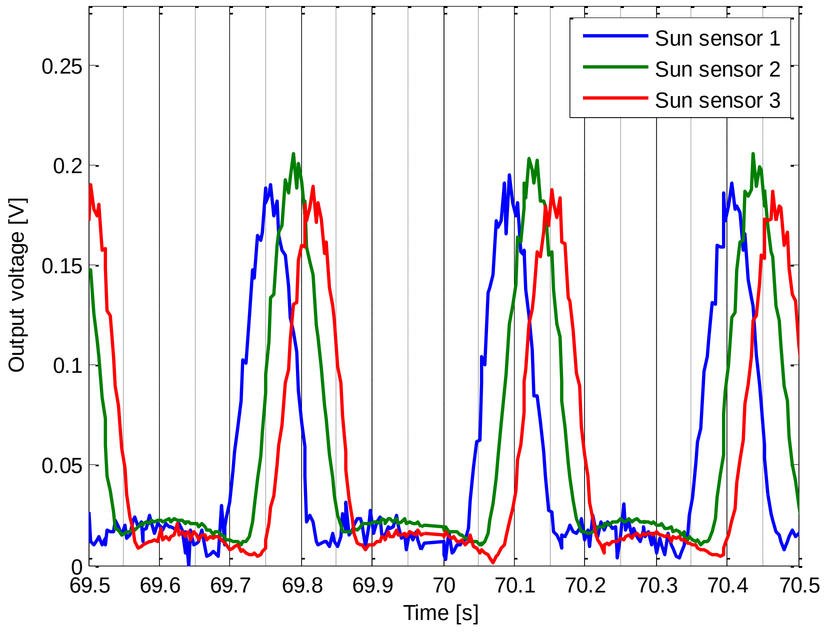


Figure 4-6 Sun sensor data

Analyzing the graph, approximate period can be calculated.

$70.1 - 69.75 = 0.35s$, where 70.1 s and 69.75 s are the time values for two consecutive peaks of the blue waveform.

The difference is 0.35 s, which is close to 333 ms for 3 Hz frequency.

The results fully correspond to the expectations.

4.3.4 Sun sensors data handling

The purpose of the sun sensors data handling process is to follow the ejection ADC data and search for maximum light intensity. The combined sun sensors readings and the maximum signal are illustrated in Figure 4-7. Every half of a rotation period plus 20 ms the maximum signal changes its value to the opposite. Time, when the signal is constant high or constant low, equals to relative time of one 360° rotation.

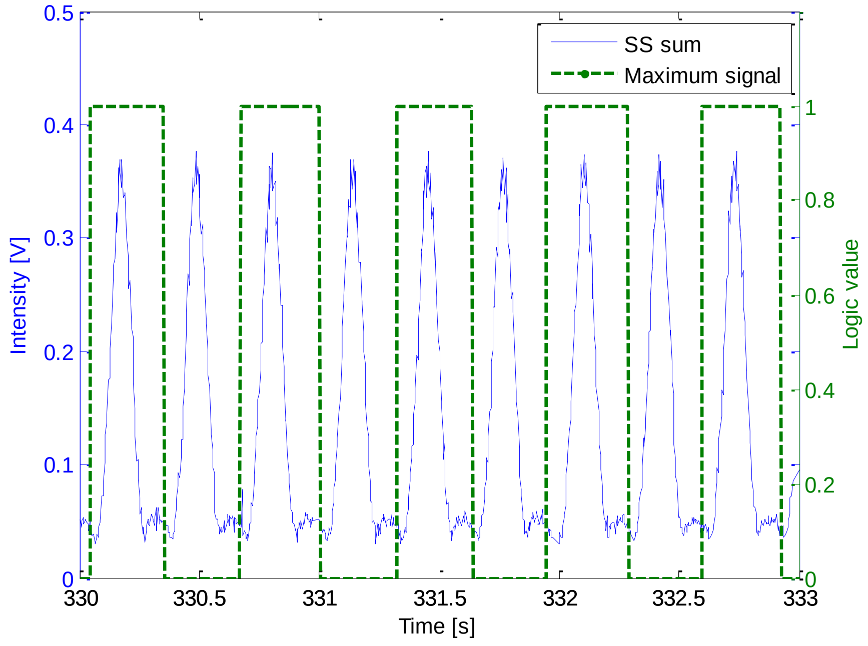


Figure 4-7 Maximum light intensity identification

Analyzing the graph, it is clearly seen that the maximum signal edges are slightly to the right from lines of symmetry between two consecutive peaks. Besides that, every peak intensity point is properly identified by the algorithm, what makes this process fully meeting the requirements.

4.3.5 Arming procedure

The ejection algorithm executes continuously, however the output is blocked by the arm signal. This preventing mechanism is shown in Figure 4-8. The diagram illustrates behavior of the output under different arm signal states. Disabling the arm signal is an important safety measure, which allows to prevent unanticipated ejection.

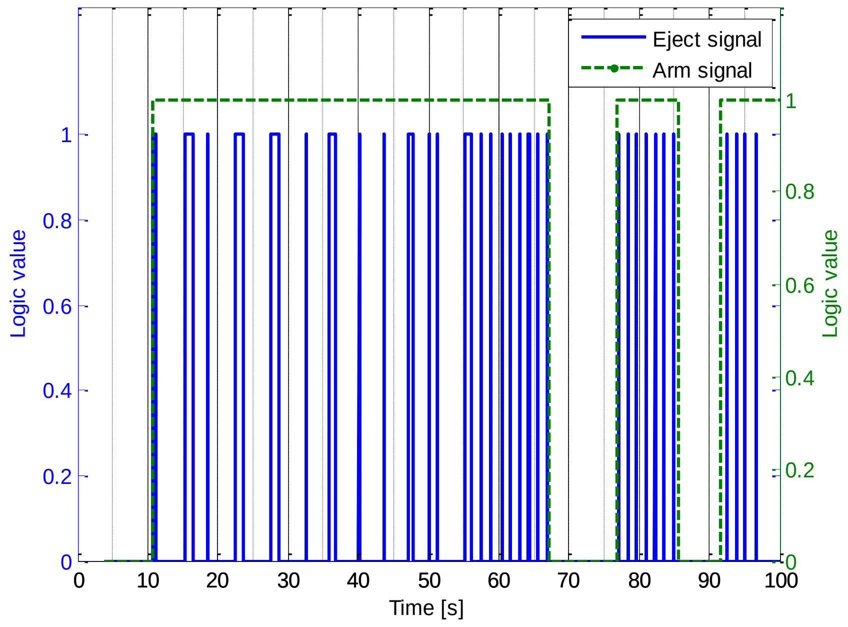


Figure 4-8 Ejection arm signal

The signal is controlled by the ground station operator and it shall be activated before liftoff due to absence of uplink during flight.

4.3.6 Ejection timeout

The timeout process is implemented as an alternate ejection method in case of malfunctions in the gyroscope, the photodiodes or the ADC. The operation principle of the process is shown in Figure 4-9.

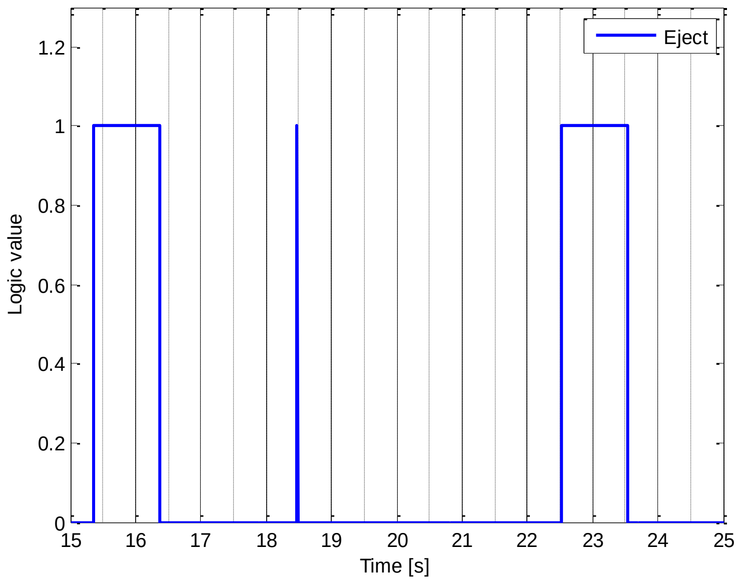


Figure 4-9 Timeout signal

The timeout process forces the output to logic 1 for 1 s. In accordance with the graph, it is high from 15.4 s to 16.4 s and 22.5 s to 23.5 s. The activation condition is 4 s without any activity in the output signal. Absence of the signal at t = 20.4 s proves its secondary importance, namely, every eject signal appearance postpones the timeout condition for another 4 s.

4.3.7 Ejection activation process

Another important part to verify is correctness of the eject signal activation time. The process calculates the time of every fifth consecutive peak intensity appearance and ejects the FFUs, taking into account delays. Figure 4-10 proves the before-mentioned algorithm.

At approximately t = 330.2 s the first maximum intensity in captured. The second peak appears at t = 330.5 s. The third one is at 330.8 s and the last is at 331.2 s. After all 4 values are found, the ejection signal goes high at 331.4 s, just before the next peak intensity because of mechanical delay.

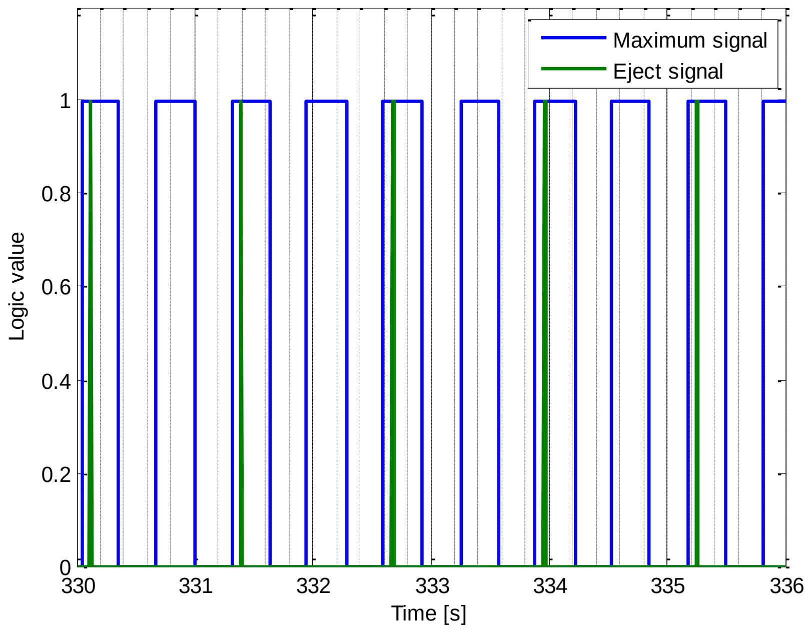


Figure 4-10 Ejection activation algorithm

4.4 Environmental tests

Under conditions of absence of convectional cooling and temperature differential during countdown, launch and flight, experiment electronics must be tested to ensure nominal operation in all the worst cases. For that reason, vacuum and thermal test are conducted.

4.4.1 Vacuum test

Vacuum test was conducted using a special vacuum chamber provided by the university. The range of pressure inside the chamber varies from normal room pressure in the lab down to 80 microns Hg (0,1067 mbar). The flight mode of the RMU PCB was activated during the test.

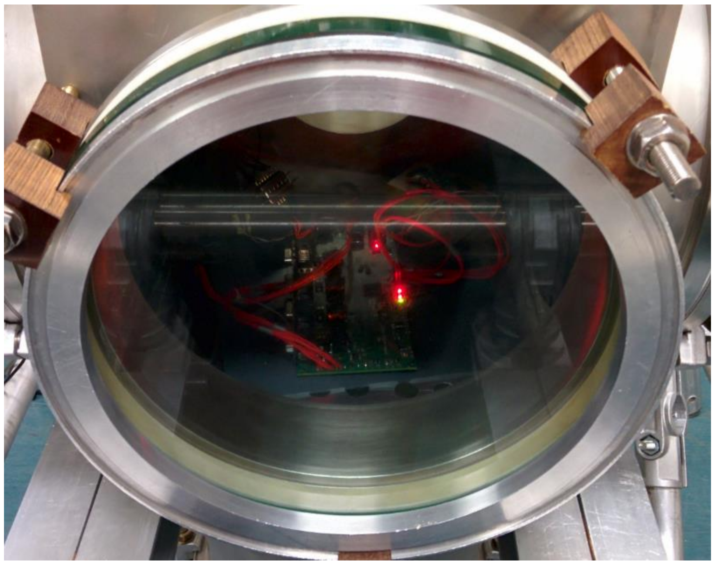


Figure 4-11 Vacuum chamber

In order to reach such low pressure inside the vacuum chamber and keep ability to power on the PCB, special adapter was made. The cable, shown in Figure 4-12, has two D-SUB 15 connectors with a sealing adapter in between, which prevents airflow from outside the chamber.

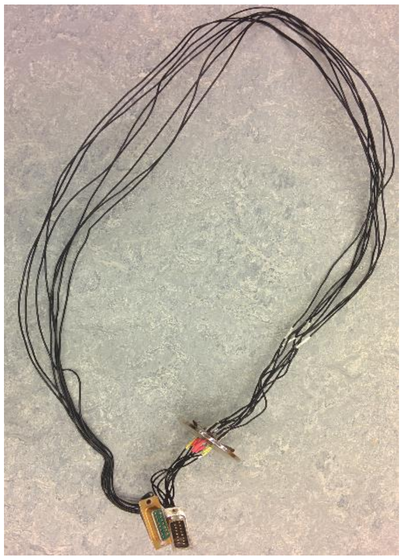


Figure 4-12 RMU power cable for vacuum test

The RMU housekeeping data was logged for 700 ms when vacuum conditions were established. Decoded data is represented below in Figure 4-13.

The diagram illustrates stability of the RxSU and the TxSU supply voltage. Several different circuits take part in output voltage generation. The RMU is powered with 28V, which is further converted into 4.2V, 3.85V for the FFUs and 5V. The last, in turn, is converted into 3.3V for feeding digital electronics. A stable final stage ensures stability in all previous stages. Moreover, continuity of the time axis confirms uninterrupted operation of the FPGA.

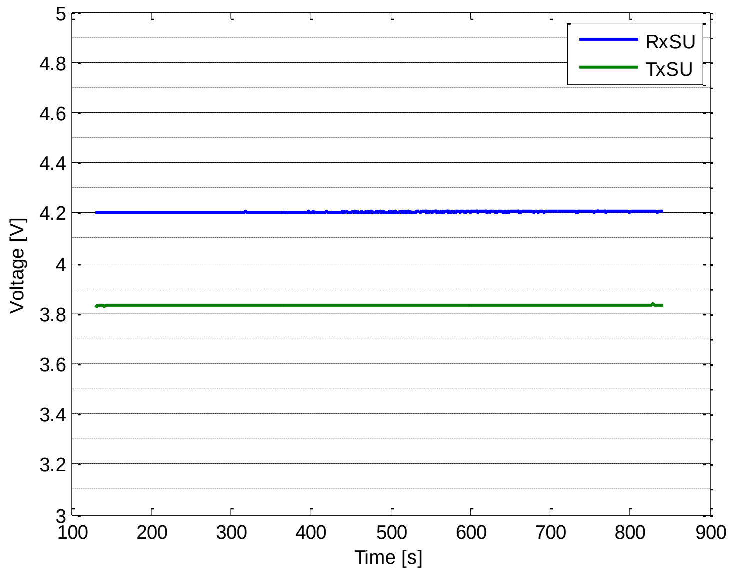


Figure 4-13 Output voltage under vacuum condition

4.4.2 Thermal test

The RMU electronics behavior under temperature variation was tested in a special thermal chamber. The entire test was conducted in two phases: a hot test and a cold test. Peak temperatures for these tests were +60°C and -30°C, respectively. The RMU data was being recorded during the temperature variation process as well 15 minutes after temperature was stabilized.

Voltage behavior across the temperature variation is nominal with maximum deviation of 1.5% and 2.3% for the hot and the cold test, respectively. The calculations are based on the data represented in Figure 4-14 and Figure 4-15.

Heating:

$3.894 - 3.836 = 0.058\,V$

$0.058 \div 3.836 \cdot 100\% = 1.5\%$

Cooling:

$3.832 - 3.744 = 0.088\,V$

$0.088 \div 3.744 \cdot 100\% = 2.3\%$

Both the FFU's supply voltage and the temperature inside the chamber follow the same trend. Maximum deviation of the voltage trace is 0.058V (Figure 4-14) under heat effect and 0.088V (Figure 4-15) under cold effect.

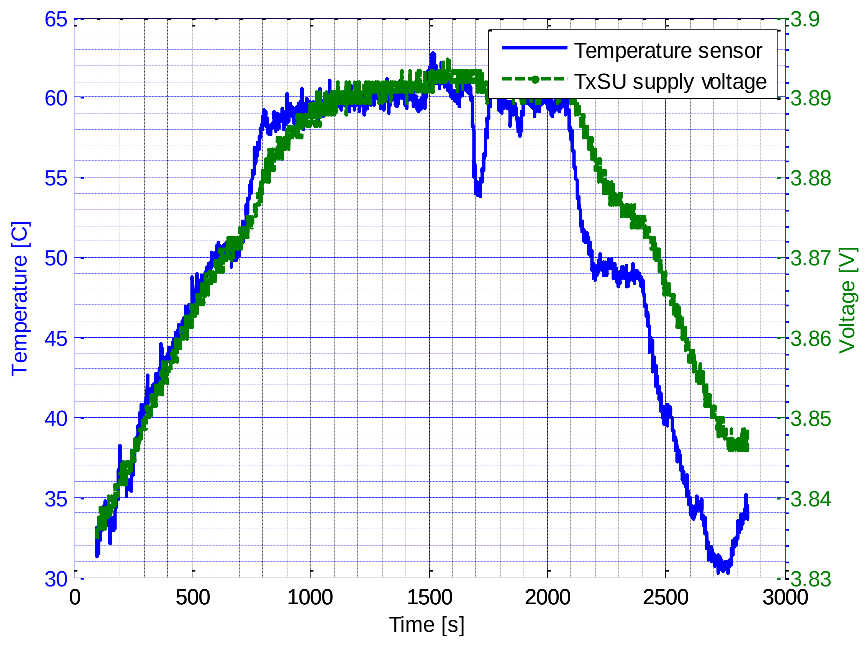


Figure 4-14 Heat influence on voltage

The reason for changes in output voltage is resistance and temperature relationship, which is described with the following equation:

$R = R_{ref} \cdot [1 + \alpha(T - T_{ref})]$, where

R – Conductor resistance at temperature T,

R_{ref} – Conductor resistance at reference temperature T_{ref}

α – Temperature coefficient of resistance for the conductor material

T – Conductor temperature in degrees Celsius

T_{ref} – Reference temperature that α is specified at for the given conductor material

The above-mentioned equation proves the behavior of the voltage traces. Any change in temperature cause a change in aggregate resistance of the electrical circuit.

Unexpected short-time temperature fall in Figure 4-14 at time t = 1700s can be caused by improper actions of a test operator.

The diagram related to the influence of cold clearly recognizes a problem. At temperature levels of -18°C and -30°C, the FPGA resets. The malfunction appeared when the 18-bit time vector value reached 1023, however this problem does not occur during the hot test. Recommended operation conditions for ProAsic3 Family FPGAs lie between -40°C and +85°C [6] and the test conditions are close to the boundary ones. Further investigation of this phenomenon needs to be done.

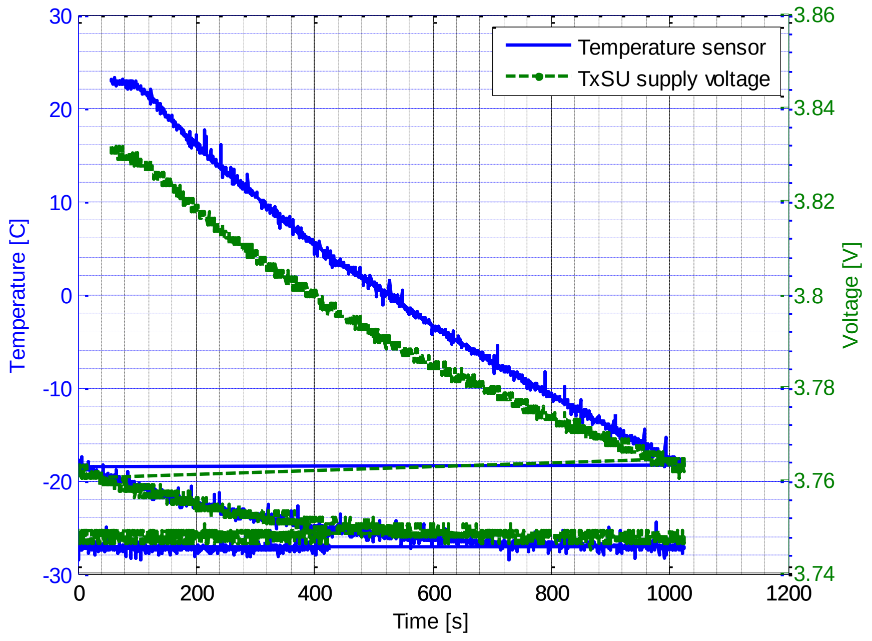


Figure 4-15 Cold influence on voltage

Chapter 5 Post-flight analysis

The launch campaign was held in Esrange, Kiruna from 19th of May until 1st of June 2014 with the launch itself on 29th of May. All real-time data before, during and after liftoff were properly logged, what allows to analyze the RMU functionality during the flight.

The experiments on-board the rocket were activated 600 s before liftoff and switched off 600 s later. Unfortunately, the FPGA reset at time t = 1024 s. However, all the most crucial data for the ISAAC experiment is collected and the malfunction had no serious impact. In general, all parts developed during the degree project time successfully accomplished the task and the further analysis proves that.

Peak spinning rate reached during the whole flight is 1408.12 dps at time t = 621.516 s. Angular velocity is plotted in Figure 5-1. Real-time communication packets of the ISAAC experiment do not allow to define exact ejection moment. The problem is solved using data provided by the FOVS experiment. Figure 5-2 illustrates accelerometer's reading in time interval from 90 s to 100 s after liftoff. According to the diagram, the ejection of the FFUs was approximately at time t = 691 s after the experiment activation. At the moment of ejection, rotation speed of the rocket was 1092.4 dps, which is very close to the expected one.

Behavior of the photodiodes is shown in Figure 5-3. Analyzing the traces, the graph can be divided into 3 stages with t_1 as a starting point and t_2 as a finishing point of the time interval.

t_1 = 590 s, t_2 = 600 s: The rocket is in stationary state on the ground and the photodiodes do not point in the Sun direction.

t_1 = 600 s, t_2 = 700 s: The rocket is in flight. Active phase of the experiment.

t_1 = 700 s, t_2 = 800 s: Yo-yo de-spin with further stable and very slow rotation speed.

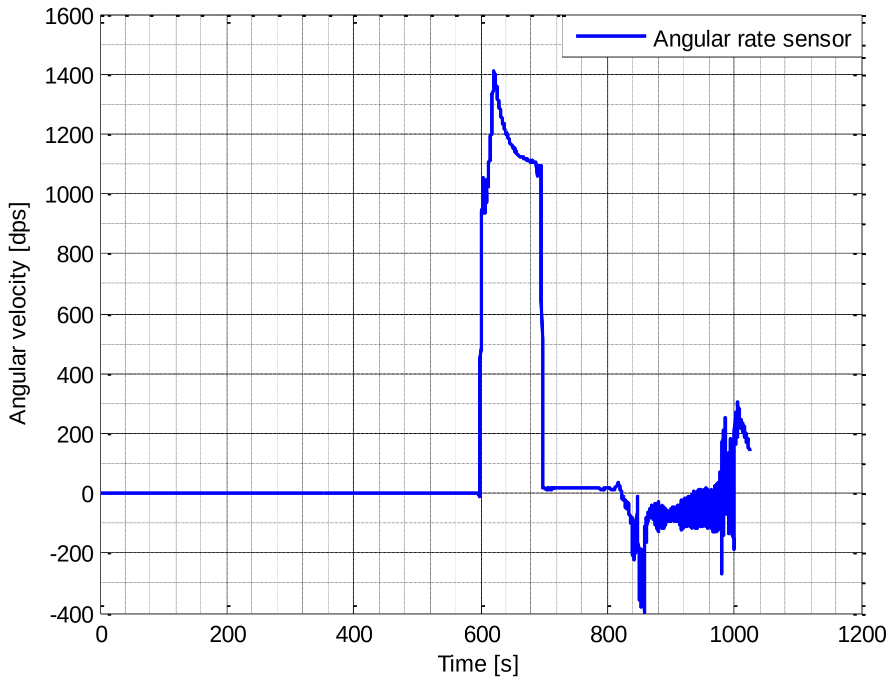


Figure 5-1 Angular velocity of the rocket

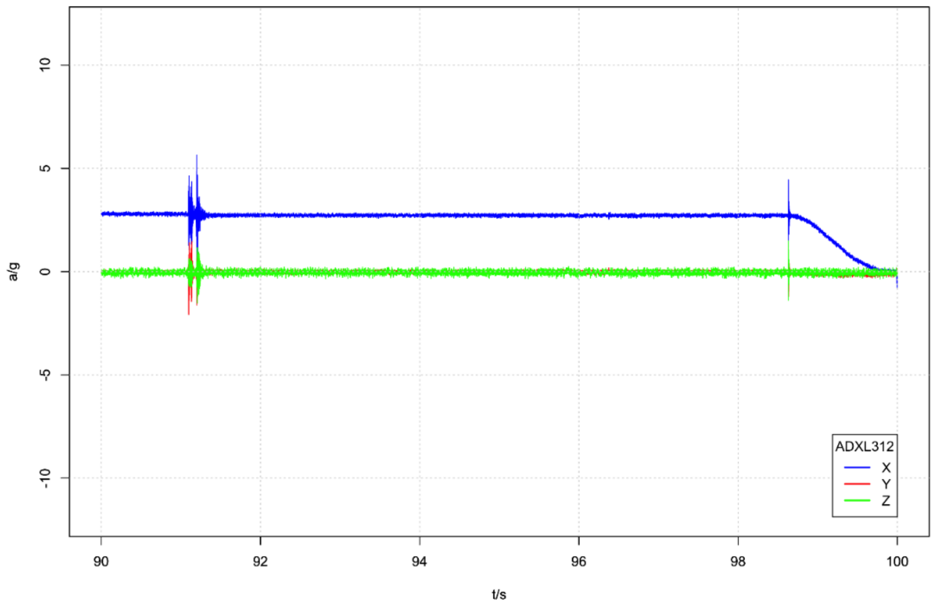


Figure 5-2 Accelerometer data

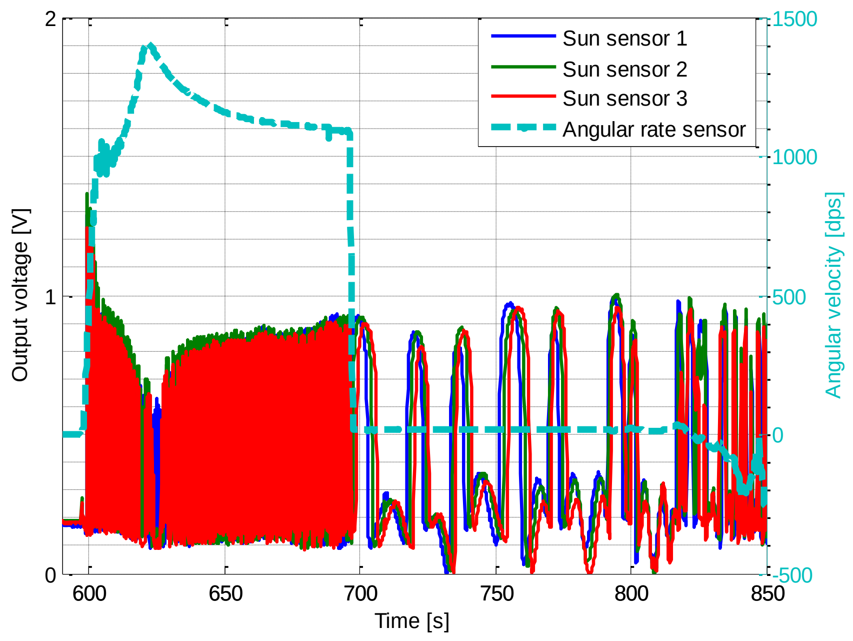


Figure 5-3 Sun sensors and gyro behavior during the flight

Maximum output voltage of the sun sensors is 1.4 V which proves the correctness of the chosen amplification gain.

Video recording related signals are shown in Figure 5-4. Due to malfunction in the FPGA and its unexpected reset, the flight video stops just after the Yo-yo de-spin.

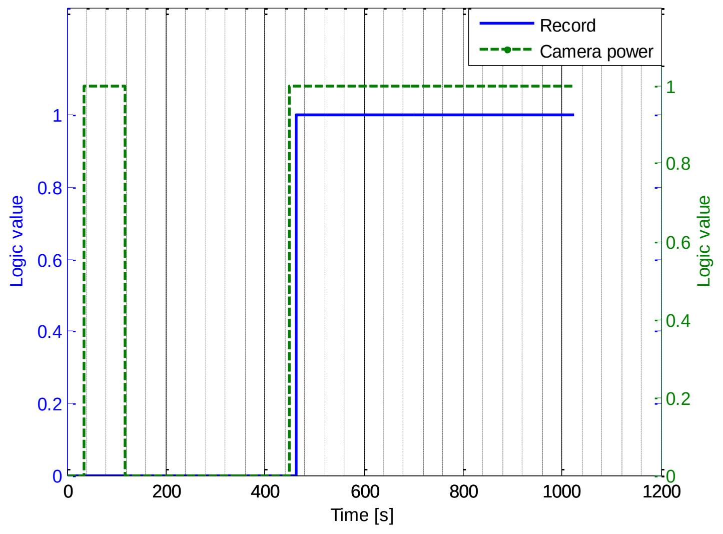


Figure 5-4 Camera signals

Some housekeeping data is illustrated in Figure 5-5. The temperature sensor is mounted on the aluminum rocket skin, which is heating very fast due to friction. FPGA supply voltage is constant during the whole period. Neither temperature change nor vacuum had any impact on the electrical parameter.

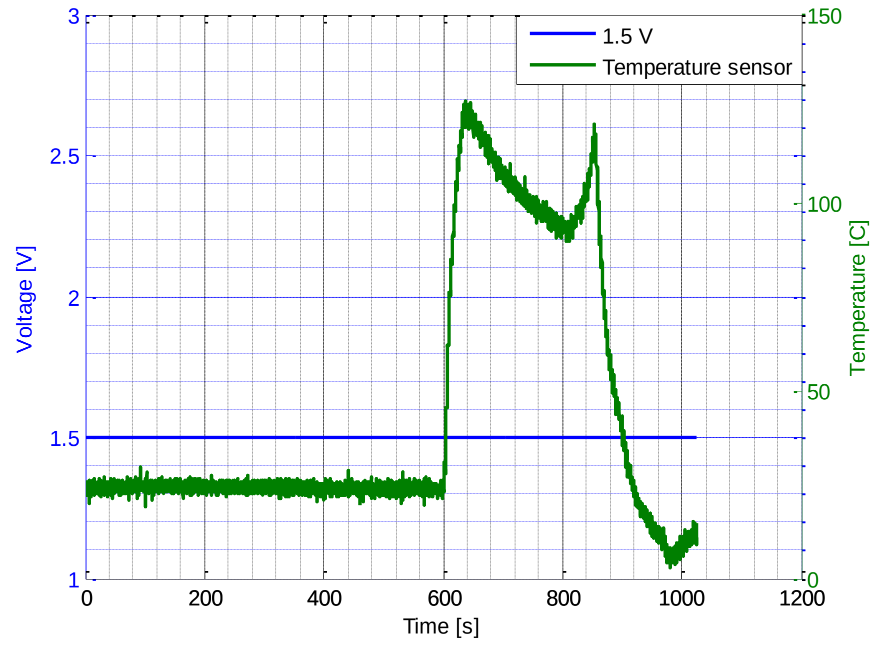


Figure 5-5 Housekeeping data

Duty cycle of the eject signal is very low, because it goes high only for 12 ms. Figure 5-6 illustrates a time window of the ejection. Readings of accelerometer show that ejection was in the interval from 690s to 692s, however, the signal is absent. The reason for that is low communication speed and messages with high eject signal are lost. Figure 5-7 is a good evidence for that. The diagram shows the closest eject signal triggered before.

If there were no any short ejection signal after, the timeout process would definitely force the eject signal high for 1s.

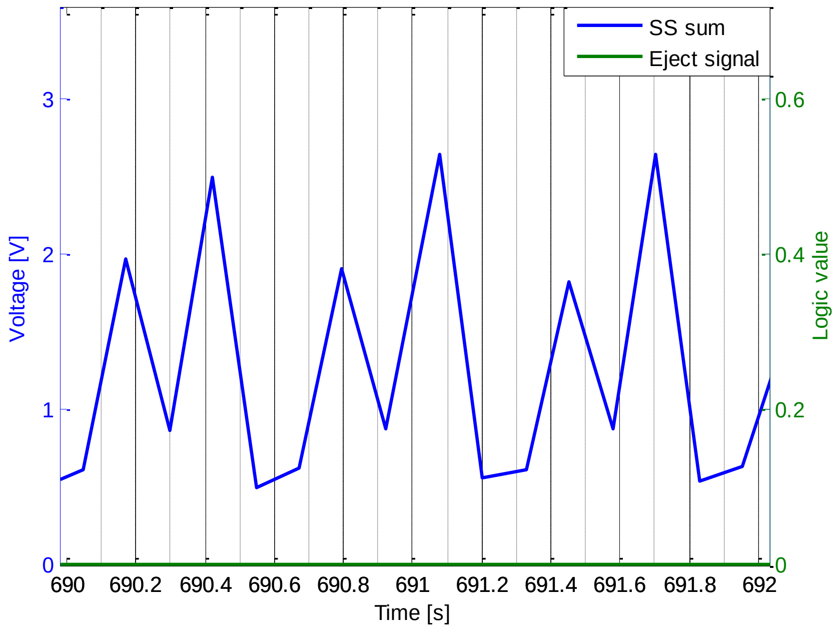


Figure 5-6 Sun sensors readings during the ejection moment

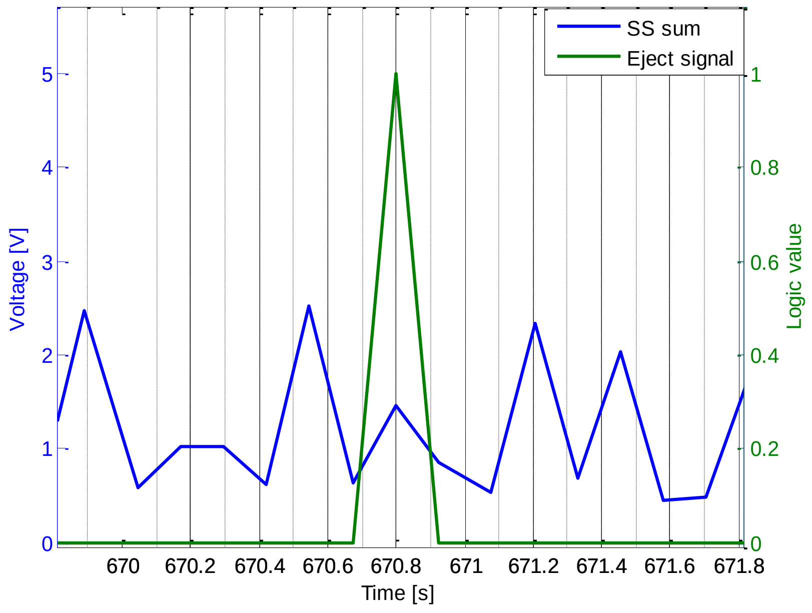


Figure 5-7 Example of ejection signal

Presence of the camera allows making a video analysis of the ejection. Figure 5-8 and Figure 5-9 illustrate video frames 27224 and 27243, respectively. The camera was set to 240 frames per second recording, what makes it possible to examine a picture every 4.17 ms. A thin black line at the bottom right hand corner of Figure 5-8 is a hatch. It is the first appearance of any part involved in the ejection procedure and, thus, the frame is stated as the beginning of the ejection process. On the other hand, Figure 5-9 has the Sun in the middle and this frame shall be 90° far away from the starting point of the ejection. Taking into account the angular rate sensor reading, the angular velocity during the ejection was 1092.4 dps or rotational frequency was 3.034 Hz. Having all this information the actual and the proper moment of ejection is calculated below.

$1 \div 240 = 0.0041(6)\ s \approx 4.17\ ms \quad time\ between\ 2\ consecutive\ frames$

$27243 - 27224 = 19\ frames \quad number\ of\ frames$

$4.17 \cdot 19 = 79.23\ ms \quad time\ between\ frames$

$1092.4 \div 360 = 3.03(4)\ Hz \quad rotation\ frequency$

$1 \div 3.03(4) \approx 0.329\ s = 329\ ms \quad time\ to\ pass\ 360°$

$329 \div 360 \cdot 90 = 82.25\ ms \quad time\ to\ pass\ 90°$

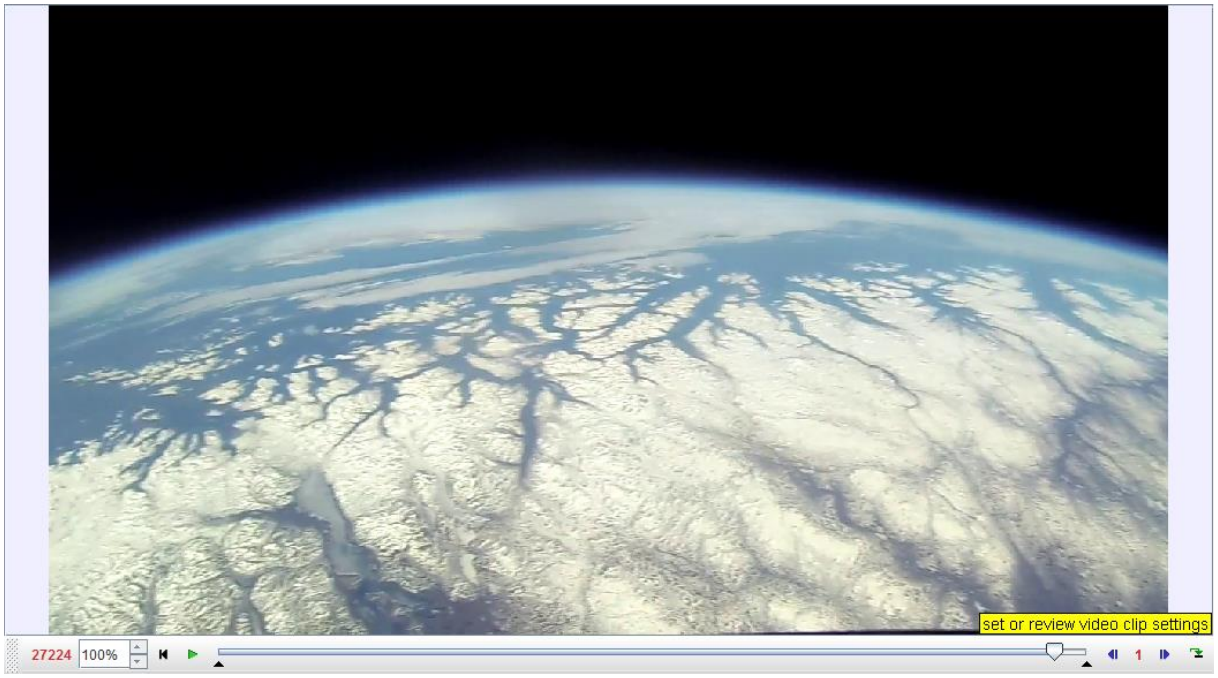


Figure 5-8 Ejection video frame 27224

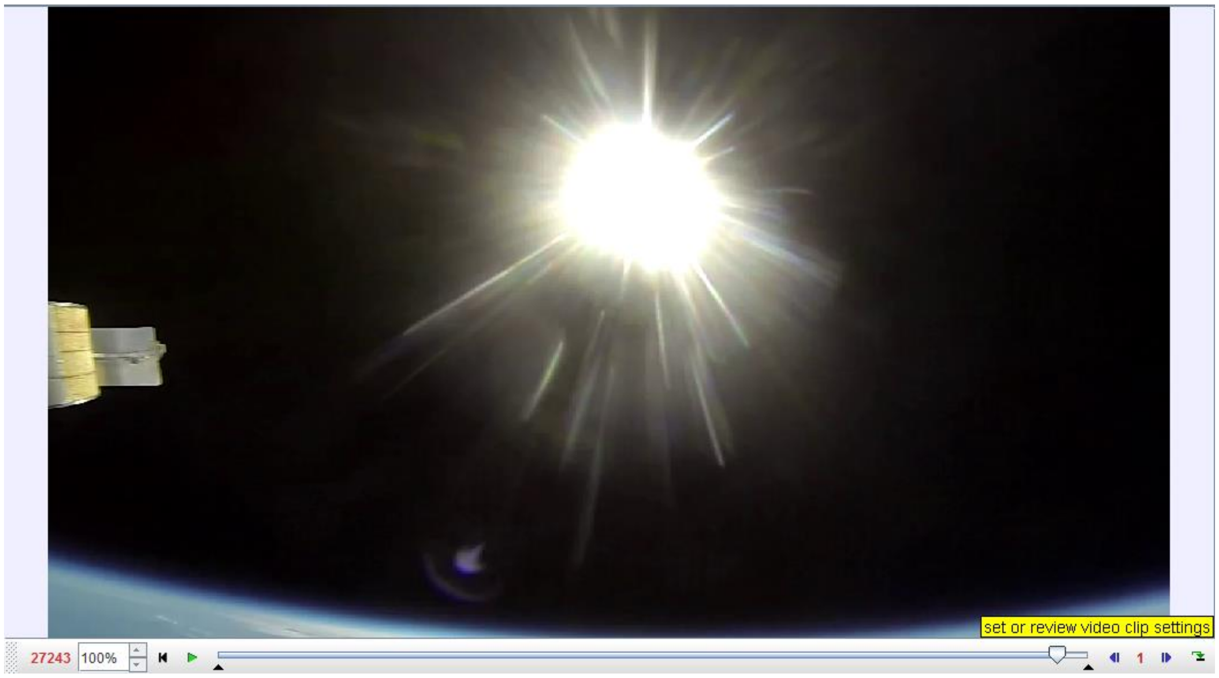


Figure 5-9 Ejection video frame 27243

In order to pass 90° with a constant speed of 1092.4 dps, 82.25 ms are needed. The time calculated using video analysis is 79.23 ms without consideration of uncertainty, which is caused by the minimal time unit of 4.17 ms, imprecise selection of a starting frame and angular rate sensor's accuracy. Possible solutions to improve ejection timing are discussed in the discussion section.

$90 - 79.23 \cdot 90 \div 82.25 = 3.3\,°$ $\qquad$ *deviation angle*

$82.25 - 79.23 = 3.02\,ms$ $\qquad$ *deviation time*

According to the calculations, the ejection took place 3.3° or 3.02 ms earlier than it should have happened. The obtained deviation is smaller than the time between two consecutive frames. The result fully corresponds to the requirement.

Chapter 6 Discussion and future work

The degree project is focused on the RMU electronics development and validation, what is clearly documented in previous chapters. The designed and implemented firmware for the RMU FPGA resulted in successful ejection control with informative real-time communication packets and high-definition video. Several new features for the ISAAC experiment were developed as well as adapted and improved the inherited ones.

A foundation of the ejection algorithm is the ability of the module to correctly identify the Sun position. Three stand-alone sun sensors were developed with the appropriate amplification gain, which matches the output with the reference voltage of the ADC. The proper gain of 3.5 was found empirically and the correctness is proved in the post-flight analysis.

The complex ejection algorithm was developed and implemented in VHDL hardware description language with interaction of seven concurrent processes. The verification phase of every single process as well as the post-flight analysis demonstrate its full compliance with the requirements. However, there is a way to improve the accuracy of the ejection, which is left for a future work. Mechanical delay of the FFUs ejection is found empirically during dynamic ejection tests and it is declared as a constant variable. In order to improve ejection timing, more test should be conducted with the FFUs of the same mass as the ones for a flight. Different delay values can be injected into the algorithm switching a value depending on the actual angular velocity.

Another important part of the developed firmware is the communication. The implemented block provides a ground station operator with all crucial and interesting information during flight; however, it has a serious drawback. The communication speed of downlink is slow and a lot of information is lost. Moreover, the block itself slows down twice a real-time message sampling rate, because it consists of two different vectors and each of them is sampled on the rising edge of the write enable signal. Despite an artificial decrease of the communication speed, it is fast enough for real-time monitoring in the GUI. For future work some improving suggestions can be given, namely to use flash memory as a storage for the real-time data and to change structure of the output_vector process in the data_sender block. Changing value of the mixed_data signal using a time vector as a triggering event will allow to get rid of necessity to wait until rising edge of the write enable signal and therefore to increase the communication speed. Supplementing flash memory will make a post-flight analysis independent on downlink speed.

The video recording algorithm for a GoPro Hero 3 camera was improved adding a possibility to switch on and off the camera without recording. The present option is important to avoid malfunctions after an improper camera shutdown.

The entire conducted work showed itself as successful and meeting all the requirements.

For future work, necessity of accelerometer and another camera should be investigated. Having accelerometer on the PCB, the ejection moment can be analyzed more precisely. The camera in the ISAAC experiment is mounted above the bottom FFU and starting ejection moment is clearly identifiable. However, due to mechanical construction, the bottom FFU is ejected later than the top one. Another camera below the top FFU will allow to make better analysis of the ejection.

Moreover, unsuitable behavior of the FPGA at time t = 1024 s needs further investigation.

Chapter 7 Conclusion

The RMU electronics for the ISAAC experiment was developed and implemented in this Master's Thesis. The RMU PCB and sun sensors PCBs were soldered, troubleshot and verified. The FPGA firmware, which embraces newly implemented ejection algorithm, real-time communication processing blocks and many others inherited from the previous experiments, was implemented in VHDL hardware description language with following validation.

The ISAAC experiment was launched on-board the REXUS15 sounding rocket on May 29 from Esrange, Kiruna. All real-time data messages were properly logged and used in the post-flight analysis for validation of the ejection algorithm. The data received during the active phase of the experiment is complete and reasonable. Unfortunately, a malfunction occurred in the FPGA in 1024 s after powering on the experiment. However, it affected only the video, having stopped it after 236 s of recording.

The results obtained in the verification phase and the post-flight analysis prove a success in reaching the objectives of this thesis work.

Chapter 8 Bibliography

[1] REXUS/BEXUS: http://www.rexusbexus.net/
(Last visited on 17th June 2014)

[2] MUSCAT (RX13) SED 5.1, September 24, 2013

[3] RAIN (RX11) SED v3.0, September 7, 2011

[4] ISAAC (RX15) SED v4.2, March 17, 2014

[5] A. Håkansson, Portal of Research Methods and Methodologies for Research Projects and Degree Projects. WORLDCOMP'13 - The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing, 22-25 July, 2013 Las Vegas, Nevada; USA.

[6] Microsemi ProASIC3 Flash Family FPGAs datasheet.

[7] L3G4200D Gyroscope datasheet.

[8] SLSD-71N6 Solderable planar photodiode datasheet.

[9] MAX 11617 12-bit ADC datasheet.

[10] N-channel SIPMOS BSP318S datasheet.

[11] TRW Airbag Systems GmbH Cutter 77003198 Typ A datasheet

[12] B. Razavi, Design of Analog CMOS Integrated Circuits, New York: McGraw-Hill, 2001, 684 pages, ISBN-10: 0072380322.

[13] LM7321 Operational amplifier datasheet.

[14] Wireless Bluetooth RF Transceiver Module serial RS232 TTL HC-05 datasheet.

Appendix A: Ejection algorithm

```
-- Ejection.vhd

library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use ieee.std_logic_arith.all;


entity EjectionControl is
PORT (
    CLK                      : in  std_logic;
    RESET                    : in  std_logic;
    Ejection_ENABLE          : in  std_logic;
    M_TIME_0                 : in  std_logic;
    M_TIME_14                : in  std_logic;
    -- From Ejection ADC
    New_Sun_Data             : in  std_logic;
    SunSensors               : in  std_logic_vector(7 downto 0);
    -- From Gyro
    New_Gyro_Data            : in  std_logic;
    GyroData                 : in  std_logic_vector(7 downto 0);
    -- Ejection
    manual_activation        : in std_logic;
    ejection_test            : out std_logic;
    ejection_1               : out std_logic
    );
end EjectionControl;

ARCHITECTURE behaviour OF EjectionControl IS

signal GLOBAL_TIME               : std_logic_vector (20 downto 0);
signal timeout_5s                : std_logic := '0';
signal SS_packed                 : std_logic_vector (63 downto 0);
signal Gyro_packed               : std_logic_vector (63 downto 0);
signal ss_max                    : std_logic_vector (12 downto 0);
signal ss_max_time               : std_logic_vector (20 downto 0);
signal reset_cond                : std_logic_vector (20 downto 0);
signal ss_LASTmax_time           : std_logic_vector (20 downto 0);
signal SS_sum                    : std_logic_vector (12 downto 0);
signal time_90_deg               : std_logic_vector (15 downto 0);
signal timeout_time_5            : std_logic_vector (9 downto 0);
signal pyro_activation_int       : std_logic;
signal state                     : std_logic_vector (3 downto 0);
signal buffer_div                 : std_logic_vector(31 DOWNTO 0);
signal cnt_div                    : std_logic_vector(4 DOWNTO 0);
signal gy_data                    : std_logic_vector (15 downto 0);
signal time_array_avg             : std_logic_vector (11 downto 0);
signal time_array_avg2            : std_logic_vector (9 downto 0);
signal count                      : integer range 0 to 3;
signal flag                       : std_logic;
type matrix is array (3 downto 0) of std_logic_vector (9 downto 0);

signal flag_old : std_logic;
signal time_array                 : matrix;
signal timeout_cond               : std_logic_Vector ( 9 downto 0);

signal state4                     : std_logic_vector (2 downto 0);
constant delay                    : std_logic_vector (11 downto 0) := x"120"; -- 144.125 ms {144ms FFUout
delay + 4.125ms activation delay}
constant zero_vector              : std_logic_vector (23 downto 0) := (others =>'0');
constant zero_vector2             : std_logic_vector (34 downto 0) := (others =>'0');
constant deg_per_ms               : std_logic_vector (23 downto 0) := x"139E52";   -- 90 * 1000 * 1000 / 70
signal condition1                 : std_logic_vector (20 downto 0);
signal condition2                 : std_logic_vector (20 downto 0);
signal state2                     : std_logic_vector (3 downto 0);
signal counter                    : std_logic_vector (9 downto 0);
signal ejection_test_int          : std_logic;


begin

ejection_1 <= (timeout_5s OR pyro_activation_int OR manual_activation) AND Ejection_ENABLE;

ejection_test <= ejection_test_int;



timer:process (M_TIME_14,RESET)
begin
   if RESET ='1' then
      GLOBAL_TIME <= (others => '0');
   elsif rising_edge (M_TIME_14) then
      GLOBAL_TIME <= GLOBAL_TIME + 1;
   end if;
end process;


SS_data_packer:process(M_TIME_0, RESET)
   begin
      if RESET='1' then
         SS_packed <= (others => '0');
      elsif rising_edge(M_TIME_0) AND New_Sun_Data='1' then
         if New_Sun_Data='1' then
            SS_packed <= SS_packed(55 downto 0) & SunSensors;
         end if;
      end if;
   end process;
```

```
SS_combine : process (RESET, CLK)

  begin
    if RESET = '1' then
      SS_sum             <= (others => '0');
      ss_max_time        <= (others => '0');
      ss_max             <= (others => '0');
      time_array_avg     <= (others => '0');
      time_array_avg2    <= (others => '0');
      time_array         <= (others =>(others =>'0'));
      reset_cond         <= (others => '0');
      flag               <= '0';
      count              <= 0;
      state2             <= x"0";
      ejection_test_int  <= '0';

    elsif rising_edge (CLK) then
      case state2 is
        when x"0" =>
              if New_Sun_Data = '0' AND SS_packed (63 downto 48) = x"01AD" then
                SS_sum <= ('0' & SS_packed(11 downto 0)) + ('0' & SS_packed(23 downto 12)) + ('0' &
SS_packed(35 downto 24));
                state2 <= x"1";
              else
                state2 <= x"0";
              end if;

        when x"1" =>
              time_array(count) <= counter;
              if  ss_max  < SS_sum then
                ss_max     <= SS_sum;
                ss_max_time  <= GLOBAL_TIME;
                reset_cond <= GLOBAL_TIME + (time_90_deg & "00" + x"28");

                state2 <= x"0";
              else
                state2 <= x"0";
                if GLOBAL_TIME(20 downto 0) >  reset_cond then
                    flag <= not(flag);
                    ejection_test_int <= not (ejection_test_int);
                    ss_max   <= (others => '0');
                    case count is
                              when 3  =>
                                      state2 <= x"2";
                                      count <= 0;
                              when others => count <= count + 1; state2 <= x"0";
                    end case;
                end if;
              end if;
```

```
            when x"2" =>
                        time_array_avg  <=  ("00"  &  time_array(0))  +  ("00"  &  time_array(1))  +  ("00"  &
time_array(2)) + ("00" & time_array(3));
                        state2 <= x"3";
            when x"3" =>
                        time_array_avg2 <= time_array_avg (11 downto 2);
                        state2 <= x"4";
            when x"4" =>
                        state2 <= x"0";
            when others =>
                        state2 <= x"0";
            end case;
        end if;
    end process;


max_counter : process (M_TIME_14, RESET,flag)
begin
    if RESET = '1' then
        counter <= (others => '0');
        flag_old <= '0';
    elsif rising_edge (M_TIME_14) then
        if (flag_old = flag) then
            counter <= counter + 1;
        else
            counter <= (others => '0');
        end if;
        flag_old <= flag;
    end if;
end process;


enable_ejection : process (CLK,RESET)
    begin
      if RESET = '1' then
        state4 <= "000";
        pyro_activation_int <= '0';
        condition1 <= (others => '0');
        condition2 <= (others => '0');
        ss_LASTmax_time <=(others => '0');
  elsif rising_edge(CLK) then
        case state4 is
            when "000" =>
               if (state2 = x"4") then
                    ss_LASTmax_time <= ss_max_time;
                    state4 <= "001";
               else
                    state4 <= "000";
               end if;
            when "001" =>
                    state4 <= "010";
                    condition1 <= ss_LASTmax_time +  (time_array_avg2 & '0') +(time_90_deg & '0') - delay;
            when "010" =>

                  if GLOBAL_TIME < condition1 then
                           pyro_activation_int <= '0';
                           state4<= "010";
                  else
                           condition2 <= condition1 + x"18"; -- +12 ms
                           state4 <= "011";
                   end if;
            when "011" =>
                 if GLOBAL_TIME < condition2 then
                           pyro_activation_int <= '1';
                           state4 <= "011";
                 else
                           pyro_activation_int <= '0';
                           state4 <= "000";
                 end if;
            when others => state4 <= "000";
        end case;
    end if;
end process;


Gyro_packer:process(M_TIME_0, RESET)
    begin
        if RESET='1' then
            Gyro_packed <= (others => '0');
          elsif rising_edge(M_TIME_0) and New_Gyro_Data='1' then
            if New_Gyro_Data='1' then
                Gyro_packed <= Gyro_packed(55 downto 0) & GyroData;
            end if;
        end if;
    end process;


Gyro_data_packer:process(CLK, RESET)
    begin
        if RESET='1' then
           time_90_deg <= (others => '0');
            gy_data <= (others => '0');
            state <= x"0";
            cnt_div <= (others => '0');
            buffer_div <= (others => '0');
        elsif rising_edge(CLK) then
            case state is
              when x"0" =>
                           if (New_Gyro_Data='0' AND Gyro_packed (63 downto 56) = x"0C") then
                           --------- 2's complement to binary ----------
                              if Gyro_packed (15) = '1' then
                                 gy_data <= not(Gyro_packed (15 downto 0)) + x"1";
                              else
                                   gy_data <= Gyro_packed (15 downto 0);
                              end if;
```

```
                    ------------
                      state <= x"1";
                      cnt_div <= "00000";
                      buffer_div <= x"00" & deg_per_ms;
                    else
                      state <= x"0";
                    end if;

          when x"1" =>                    -- division state
                    if buffer_div(30 DOWNTO 15) >= gy_data then
                      buffer_div(31 downto 16) <= '0' & (buffer_div(29 downto 15) - gy_data(14 downto 0));
                      buffer_div(15 downto 0) <= buffer_div(14 downto 0) & '1';
                    else
                      buffer_div <= buffer_div((30) downto 0) & '0';
                    end if;

                    if cnt_div /= "10000" then -- 16 cycles
                      cnt_div <= cnt_div + 1;
                      state <= x"1";
                    else        -- Division is done
                      time_90_deg <= buffer_div(15 downto 0);
                      state <= x"0";
                    end if;

                    when others =>
                      state <= x"0";
                    end case;
      end if;
   end process;



timeout_ejection: process (CLK,RESET,pyro_activation_int)
begin
      if RESET = '1' then
          timeout_5s <= '0';
          timeout_time_5 <= (others => '0');
          timeout_cond <= "00" & x"04";
      elsif rising_edge (CLK) then
        timeout_time_5 <= GLOBAL_TIME(20 downto 11);
        if pyro_activation_int = '1' then
           timeout_cond <= GLOBAL_TIME(20 downto 11) + x"4";
        end if;
        if (timeout_time_5 < timeout_cond) then
              timeout_5s <= '0';
        elsif (timeout_time_5 = (timeout_cond+x"1")) then
              timeout_5s <= '0';
              timeout_cond <= GLOBAL_TIME(20 downto 11) + x"4";
        else
              timeout_5s <= '1';
        end if;
      end if;
end process;

end behaviour;
```

Appendix B: Data_sender

```
-- data_sender.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity data_sender is
PORT (
        M_TIME_0            : IN std_logic;
        M_TIME_6_0          : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
        CLK                 : IN std_logic;
        RESET               : IN std_logic;
        H_ADC_RE            : IN std_logic;             -- Housekeeping ADC
        H_ADC_Data          : IN std_logic_vector (7 downto 0);  -- Housekeeping ADC
        Gyro_RE             : IN std_logic;
        Gyro_Data           : IN std_logic_vector (7 downto 0);
        E_ADC_RE            : IN std_logic;             -- Ejection ADC
        E_ADC_Data          : IN std_logic_vector (7 downto 0); -- Ejection ADC
        Exp_Data            : OUT std_logic_vector (7 downto 0);
        Exp_Data_WE         : OUT std_logic
        );
end data_sender;


architecture behaviour of data_sender is

signal ADC_E_packed             : std_logic_vector (63 downto 0);
signal ADC_H_packed             : std_logic_vector (63 downto 0);
signal Gyro_packed              : std_logic_vector (63 downto 0);
signal mixed_data               : std_logic_vector (47 downto 0);
signal ADC_H_temp               : std_logic_vector (23 downto 0);
signal Gyro_temp                : std_logic_vector (23 downto 0);
signal ADC_E_temp               : std_logic_vector (35 downto 0);
signal cnt                              : std_logic ;
signal cnt_old                          : std_logic;
signal re                               : std_logic;
signal Exp_Data_WE_int                  : std_logic;


begin

        Exp_Data_WE <= Exp_Data_WE_int;

ADC_E_Packer:process(M_TIME_0, RESET)
begin
        if RESET='1' then
                ADC_E_packed <= (others => '0');
        elsif rising_edge(M_TIME_0) AND E_ADC_RE='1' then
       if E_ADC_RE='1' then
                ADC_E_packed <= ADC_E_packed(55 downto 0) & E_ADC_Data;
       end if;
  end if;
end process;


ADC_H_Packer:process(M_TIME_0, RESET)
begin
  if RESET='1' then
                ADC_H_packed <= (others => '0');
        elsif rising_edge(M_TIME_0) AND H_ADC_RE='1' then
       if H_ADC_RE='1' then
                ADC_H_packed <= ADC_H_packed(55 downto 0) & H_ADC_Data;
       end if;
        end if;
end process;


Gyro_Packer:process(M_TIME_0, RESET)
begin
        if RESET='1' then
    Gyro_packed <= (others => '0');
  elsif rising_edge(M_TIME_0) AND Gyro_RE='1' then
```

```
            if Gyro_RE='1' then
                        Gyro_packed <= Gyro_packed(55 downto 0) & Gyro_Data;
            end if;
    end if;
end process;


H_ADC_handling: process(M_TIME_0, RESET)
begin
            if RESET = '1' then
                        ADC_H_temp <= (others => '0');
            elsif rising_edge (M_TIME_0) AND H_ADC_RE = '0' then
                        if ADC_H_packed (63 downto 48) = x"01AD" then
                                    ADC_H_temp (23 downto 12) <= ADC_H_packed (47 downto 36);
                        elsif ADC_H_packed (63 downto 48) = x"03AD" then
                                    ADC_H_temp (11 downto 0) <= ADC_H_packed (11 downto 0);
                        end if;
            end if;
end process;


gyro_handling:process (CLK, RESET)
begin
            if RESET = '1' then
                        Gyro_temp <= (others => '0');
            elsif rising_edge (CLK) AND Gyro_RE = '0' then
                        if Gyro_packed(63 downto 56) = x"0C" then
                                    Gyro_temp(23 downto 16) <= Gyro_packed (55 downto 48);
                                    Gyro_temp(15 downto 0) <= Gyro_packed (15 downto 0);
                        end if;
            end if;
end process;


E_ADC_handling:process (CLK, RESET)
begin
            if RESET = '1' then
                        ADC_E_temp <= (others => '0');
```

```
		elsif rising_edge (CLK) AND E_ADC_RE = '0' then
			if ADC_E_packed(63 downto 48) = x"01AD" then
				ADC_E_temp <= ADC_E_packed (35 downto 0);
			end if;
		end if;
	end process;


	output_vector: process (CLK, RESET)
	begin
		if RESET = '1' then
			mixed_data <= (others => '0');
			cnt <= '0';
			re <= '0';
			cnt_old <= '0';
		elsif rising_edge(CLK) then
			re <= Exp_Data_WE_int;
			if ((re = '1' and Exp_Data_WE_int = '0')  ) then
				if cnt = '0' then
					mixed_data <= ADC_H_temp & Gyro_temp;
					cnt <= '1';

				else
					mixed_data <= x"000" & ADC_E_temp;
					cnt <= '0';

				end if;
			end if;
		end if;
	end process;


	WE:process (M_TIME_6_0(4), RESET)
	begin
		if RESET = '1' then
			Exp_Data_WE_int <= '0';
		elsif rising_edge(M_TIME_6_0(4)) then
			if M_TIME_6_0(6 downto 5) = "10" then
```

```
                        Exp_Data_WE_int <= '1';
                else
                        Exp_Data_WE_int <= '0';
                end if;
        end if;
end process;


Exp_Data <=  x"0"&"000"& cnt            when M_TIME_6_0(3 downto 1)="000" else
        x"BE"                           when M_TIME_6_0(3 downto 1)="001" else
        mixed_data(47 downto 40)        when M_TIME_6_0(3 downto 1)="010" else
        mixed_data(39 downto 32)        when M_TIME_6_0(3 downto 1)="011" else
        mixed_data(31 downto 24)        when M_TIME_6_0(3 downto 1)="100" else
        mixed_data(23 downto 16)        when M_TIME_6_0(3 downto 1)="101" else
        mixed_data(15 downto 8)         when M_TIME_6_0(3 downto 1)="110" else
        mixed_data(7 downto 0);


end behaviour;
```

Appendix C: FFU Comm switches

```
-- ffu_mux_rx.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity ffu_mux_rx is
PORT (
        MUX                     : in std_logic_vector (1 downto 0);
        serial_out      : in std_logic;
        RXSU_RX         : out std_logic;
        RXCU_RX         : out std_logic;
        TXCU_RX         : out std_logic
  );
end ffu_mux_rx;


ARCHITECTURE mux OF ffu_mux_rx IS
begin
process (serial_out,MUX)
begin
case MUX is
  when "01" =>    TXCU_RX <= serial_out;
  when "10" =>    RXCU_RX <= serial_out;
  when "11" =>    RXSU_RX <= serial_out;
  when others => null;
end case;
end process;
end mux;
```

```
-- ffu_mux_tx.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity ffu_mux_tx is
PORT (
            MUX          : in std_logic_vector (1 downto 0);
            RX_out        : out std_logic;
            RXSU_RX_in    : in std_logic;
            RXCU_RX_in    : in std_logic;
            TXCU_RX_in    : in std_logic
   );
end ffu_mux_tx;


ARCHITECTURE mux OF ffu_mux_tx IS
begin
RX_out <= TXCU_RX_in when MUX = "01" else
        RXCU_RX_in when MUX = "10" else
        RXSU_RX_in when MUX = "11" else
        '1';
end mux;
```

Appendix D: RxSU Comm switch

```
-- switch2.vhd


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;


entity switch2 is

    Port(
        TX_MAIN         :in std_logic;
        UART_CAM_TX     :in std_logic;
        UART_SF2_TX     :in std_logic;
        UART_IR_TX      :in std_logic;
        UART_SENS_RX    :in std_logic;
        switch          :in std_logic_vector(1 downto 0);
        RX_MAIN         :out std_logic;
        UART_CAM_RX     :out std_logic;
        UART_SENS_TX    :out std_logic;
        UART_IR_RX      :out std_logic;
        UART_SF2_RX     :out std_logic
        );
end switch2;



architecture behavioral of switch2 is
begin

RX_MAIN <= UART_SENS_RX;

mux : process (TX_MAIN,UART_CAM_TX,UART_SF2_TX,UART_SENS_RX,switch)
begin
    case switch is
        when "00" =>
                    UART_SENS_TX <= TX_MAIN;
                    UART_CAM_RX <= '1';
                    UART_SF2_RX <= '1';
                    UART_IR_RX <= '1';
        when "01" =>
                    UART_CAM_RX <= UART_SENS_RX;
                    UART_SENS_TX <= UART_CAM_TX;
                    UART_SF2_RX <= '1';
                    UART_IR_RX <= '1';
        when "10" =>
                    UART_SF2_RX <= UART_SENS_RX;
                    UART_SENS_TX <= UART_SF2_TX;
                    UART_CAM_RX <= '1';
```

```
                        UART_IR_RX <= '1';
        when "11" =>
                        UART_IR_RX <= UART_SENS_RX;
                        UART_SENS_TX <= UART_IR_TX;
                        UART_CAM_RX <= '1';
                        UART_SF2_RX <= '1';
        when others => null;
    end case;
end process;

end behavioral;
```

```
-- CommuCon.vhd

library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

entity CommuCon is
port (
        CLK             : in  std_logic;
                RESET       : in  std_logic;
                RX_BYTE     : in  std_logic_vector(7 downto 0);
                RX_BYTE_OK   : in  std_logic;
        switch      : out std_logic_vector (1 downto 0)
    );
end CommuCon;

architecture behavioral of CommuCon is
        signal rxByte   : std_logic_vector(7 downto 0);
        signal rxBit0   : std_logic;
        signal rxBit1   : std_logic;
  signal rx_state  : std_logic;
  signal switch_int : std_logic_vector (1 downto 0);
begin
  switch <= switch_int;
        process (CLK, RESET)
        begin
        if RESET = '1' then
                rx_state <= '0';
                rxByte  <= (others => '0');
                rxBit0 <= '0';
                rxBit1 <= '0';
                switch_int <= "10";
        elsif rising_edge(CLK) then
                rxBit0 <= RX_BYTE_OK;
                rxBit1 <= rxBit0;
                        if rx_state = '0' then
                                if rxBit0 = '1' and rxBit1 = '0' then
                                        rxByte <= RX_BYTE;
                                        rx_state <= '1';
                                else
                                        rx_state <= '0';
                                end if;
                        else
                                rx_state <= '0';
                                case rxByte is
                                        when x"25" => switch_int <= "00"; -- % for sensor FPGA
                                        when x"26" => switch_int <= "01"; -- & for camera FPGA
                                        when x"28" => switch_int <= "10"; -- ( for SmartFusion2
                                        when x"29" => switch_int <= "00"; -- ) for IR FPGA
                                        when others => null;
                                end case;
                        end if;
        end if;
end process;
end behavioral;
```