Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Software tool for a configuration of a radiation detector for space applications

Natálie Vítová

Supervisor: Ing. Pavel Brož
Field of study: Cybernetics and Robotics
Subfield: Cybernetics and Robotics
August 2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Vítová Natálie**	Personal ID number:	**501209**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Software tool for a configuration of a radiation detector for space applications

Master’s thesis title in Czech:

Softwarový nástroj pro konfiguraci radiačního detektoru pro kosmické aplikace

Guidelines:

Design and implement a software tool which allows configuring SXRM radiation detector (SpacePix Radiation Monitor) in 2SD instrument for Cubesats.
Briefly describe the instrument and radiation detector which you will work with. Describe the configuration of the detector and configuration subsystem of the instrument.
Prepare a SW design document which will describe the tool (static structure and dynamic behavior).
Test the developed tool with engineering model of the 2SD instrument.
The SW tool shall comply with the following requirements:

- To be based on GUI (Graphical User Interface)
- To allow to set full configuration (global + local) of each radiation sensor
- To allow to read and set configuration of the instrument
- To start & stop data acquisition of the detector
- To display acquired data

The SW tool shall be implemented in Python and the graphical interface shall be based on Tkinter. Reasonable documentation such as design and interface descriptions shall be provided.

Bibliography / sources:

[1] Moore A. D., Python GUI Programming with Tkinter, Packt Publishing, 2018, ISBN 978-1788835886
[2] Zaccone G., Python Parallel Programming Cookbook, Packt Publishing, 2015, ISBN 978-1785289583
[3] Cox T., Fernandes S. L., Yamanoor Say, Yamanoor Srihari, Vaish D., Getting Started with Python for the Internet of Things, Packt Publishing, 2019, ISBN 978-1838555795

Name and workplace of master’s thesis supervisor:

Ing. Pavel Brož esc Aerospace s.r.o., Čs. armády 14, 160 00 Praha 6

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **22.11.2021** Deadline for master's thesis submission: **15.08.2022**

Assignment valid until:
by the end of summer semester 2022/2023

Ing. Pavel Brož	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to thank my supervisor Ing. Pavel Brož, for his guidance and assistance with my thesis. I am especially grateful for his enthusiasm when it comes to explaining anything about work in the space industry.

I would like to thank my parents for their continuous support not just in my studies but in my whole life, my grandparents and my brother for believing in me, and my boyfriend for his patience.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 15 August 2022

Abstract

The thesis deals with design of configuration software, ConfPix, for radiation detecting sensor - SpacePix2. It describes design and implementation, as well as the testing procedure. The software is implemented in Python, with Graphical User Interface created using the Tkinter module for Python. The application is designed as versatile and can be used in future missions requiring the configuration of pixel radiation detectors from SpacePix family.

The SpacePix2 sensor is a new technology developed at the Faculty of Nuclear Sciences and Physical Engineering at Czech Technical University in Prague and will be tested in orbit as a part of the Space Dosimetry System Demonstrator instrument developed by the esc Aerospace s.r.o. company.

Cosmic radiation sources and radiation detectors are described in the theoretical part of the thesis. The Space Dosimetry System Demonstrator instrument, SpacePix Radiation Monitor detector and the SpacePix2 sensor are described as well.

Keywords: radiation detection, sensor configuration, Space Dosimetry System Demonstrator, SpacePix Radiation Monitor, SpacePix-2-Lin-S

Supervisor: Ing. Pavel Brož

Abstrakt

Tato práce se zabývá návrhem konfiguračního softwaru ConfPix pro konfiguraci senzoru detekujícího záření (SpacePix2). Práce popisuje návrh a implementaci, stejně jako postup testování. Software je napsán v programovacím jazyce Python, s grafickým uživatelským prostředím (GUI), vytvořeným pomocí modulu Tkinter pro Python. Software je navržen tak, aby byl univerzální a mohl být použit v budoucích misích, vyžadujících konfiguraci radiačních senzorů řady SpacePix.

Senzor SpacePix2 je nová technologie, vyvinutá na Fakultě jaderné a fyzikálně inženýrské na ČVUT v Praze a bude testována na oběžné dráze jako součást nástroje Space Dosimetry System Demonstrator vyvinutého společností esc Aerospace s.r.o.

Zdroje vesmírného záření a typy detektorů záření jsou popsány v teoretické části práce. Detailně jsou popsány i zařízení Space Dosimetry System Demonstrator, detektor SpacePix Radiation Monitor a senzor SpacePix2.

Klíčová slova: detekce radiace, konfigurace senzoru, Space Dosimetry System Demonstrator, SpacePix Radiation Monitor, SpacePix-2-Lin-S

Překlad názvu: Softwarový nástroj pro konfiguraci radiačního detektoru pro kosmické aplikace

Contents

Figures

Tables

Introduction

Esc Aerospace s.r.o. is a small company focusing mainly on mission-critical systems such as qualified flight software, On-Board Computer (OBC) and radiation monitor and sensor systems. One of their products is a Space Dosimetry System Demonstrator (2SD) instrument, which will be flown on the CubeSatCarrier 2 satellite, one of two 6-Unit CubeSats. The satellites are developed as a platform for the validation and demonstration of new technologies in the EU Horizon 2020 Programme frame. The satellite platform development and integration are provided by the Innovative Solutions In Space (ISISpace) company. The same company will also ensure later operations in orbit. [1] The CubeSats will be launched to the Low Earth Orbit (LEO) where the 2SD instrument shall operate, thereby verifying the new technology it contains - the pixel radiation sensor SpacePix-2-Lin-S (SpacePix2). The sensor was designed at the Faculty of Nuclear Sciences and Physical Engineering (FNSPE) at the Czech Technical University (CTU) in Prague. The five SpacePix2 sensors embedded in the 2SD instrument must be configured prior to measurement in order to operate correctly.

This work aimed to design and implement software for managing a SpacePix2 configuration loaded in 2SD instrument with the capability to configure all five sensors independently of each other. The proposed software shall work with the engineering and qualification model on the ground and later with the flight model in orbit. Except for direct connection with engineering or qualification model on the ground, the software shall allow generating commands to upload the configuration via a communication link to the device in orbit. Making the configuration automatic saves time and makes it easier for users to navigate themselves in setting individual parameters.

The thesis is divided into two parts which are further divided into chapters. Chapters one and two belong to the first - theoretical part, while chapters three and four belong to the second - design and implementation part. The first chapter briefly introduces state of the art, while the second chapter focuses on the 2SD instrument, SpacePix Radiation Monitor detector and SpacePix2 sensors. In chapter three, the design of the software is described. That includes Graphical User Interface (GUI) design, static architecture, class diagrams and description of software modules. Chapter four covers software validation and results and is followed by the conclusion.

Part I

Theoretical part

Chapter 1
State of the art

In this chapter, the theoretical background is presented.

1.1 Radiation detectors

Radiation measuring instruments for space applications are usually designed specifically for a particular mission and depend on the species and energy levels which need to be observed. The instruments are limited in several areas, such as weight, dimensions and power consumption. In order to measure a radiation field, they use either direct or indirect approaches. The direct methods detect high-energy photons and particles ranging from 100 to 10^{15} eV. This range can be extended using indirect measurements up to above 10^{20} eV. In the following subsections, radiation, its sources and types will be described, followed by techniques used to measure radiation in space. [2]

1.1.1 Radiation

Radiation in space occurs in two forms - corpuscular and electromagnetic (photons). Corpuscular radiation is represented by charged particles - protons, electrons, heavy ions, and neutrons. Rays in the X-ray and gamma spectrum represent electromagnetic radiation. The radiation sources in space are radiation belts (so-called Van Allen belts), Solar Particle Events (SPE) (or coronal mass ejections) such as solar wind or solar flare and lastly, Galactic Cosmic Rays (GCR). GCR do not come from the solar system, but their primal sources are within the Milky Way galaxy. [3], [4]

The Van Allen belts are the closest radiation source to Earth and are formed by the inner and outer belts. In the inner belt, called the proton belt, there are trapped high energetic protons (around 100 MeV) and trapped electrons (around 30 MeV). In the outer one, there are mainly trapped electrons (around 7 MeV); thus, it is called an electron belt. The GCR are mainly formed by interplanetary high energetic protons and from around 10% by heavy ionized nuclei - alpha particles (helium nuclei) and heavy nuclei ions with high charge and energy. The solar wind consists of a large part of protons, and the rest is electrons. [3], [4] Radiation from space has nearly no effects on humans on Earth but can have even deadly effects on them in space. Cancer or degenerative diseases can be caused by excessive exposure to GCR or SPE. For more extended stays in orbit, on the surface of the moon or interplanetary travels, it is essential to understand how radiation affects the human body. This is one of the main reasons why it is necessary to send as many radiation detectors as possible to space. Humans are not the only ones that are affected by space radiation. Materials like solar cells or spacecraft (S/C)'s shielding degrade, and electronics are highly influenced too. [3], [4]

1.1.2 Measuring X-ray and γ-ray photons

The direct approach is used when measuring X-ray and γ-ray photons with energies from $\sim 0.1\,\text{keV}$ to $\sim 300\,\text{GeV}$. Many instruments were designed for this purpose using different techniques. These include collimation (restricting X-ray to a given area), grazing-incidence optics, coded aperture mask and pair-production tracking. [2]

The collimation methods use gaseous detectors or phoswich detectors and do not provide imaging of X-ray sources. Grazing-incidence optics technique is based on the reflectivity of mirror surfaces. The X-rays can be reflected as long as the critical angle is higher than the glancing angle and the incident angle is shallow. Both the mirror material and the X-ray energy influence the critical angle. The technique covers energies up to $\sim 10\,\text{keV}$ and has high angular resolution and a Field Of View (FOV) of about 1°. To accomplish imaging above $\sim 10\,\text{keV}$, a coded aperture masks method can be used. The technique uses a position-sensitive photosensor placed under a coded aperture mask, which places a unique pattern for different source directions on the photosensor. This technique can cover large collection areas and achieve a good angular resolution. After applying deconvolution to the detected photons on the photosensor, the image of the radiation source can be obtained. [2]

To observe high-energy γ-rays, a pair conversion is used. This phenomenon dawns above an energy threshold, which differs for each environment but generally is around a few MeV. The γ-rays can be observed by following the momentum vectors of the e^+e^- pairs and measuring their energy together. [2]

1.1.3 Measuring GCR

As indicated above, the GCR are formed by high-energetic (even energies exceeding $10^{20}\,\text{eV}$) particles that almost reach light speed. Different approaches have to be used to measure these than X-rays and γ-rays. These include Time-Of-Flight (TOF) measurements, $dE/dx - E$ technique, magnetic rigidity spectrometers, ionisation calorimeters or indirect methods. [2]

Ions with energies lower than several MeV per nucleon can be measured using a TOF mass spectrometer. This technique uses a series of thin metal foils, MicroChannel Plates (MCPs) and electrostatic mirrors. The mass of a particle can be calculated using the total kinetic energy of the particle and its measured velocity. This method cannot measure the charge of a particle. [2]

To measure energies below several hundreds of MeV per nucleon, a $dE/dx-E$ technique is used. It is one of the most common methods used to measure the radiation of GCR. The dE/dx indicates energy loss, while the E indicates total kinetic energy. Each of these requires a separate detector. Silicon diodes are usually used. The first detector shall be as thin as possible, while the second must be thick enough to stop particles with energies in the required range. This technique can detect all the particles' charge, mass and energy. Several methods to accomplish maximum resolution include position-sensitive solid-state-detector hodoscope or drift chambers and scintillating-optical-fibre hodoscopes. [2]

Another technique to measure GCR is using magnetic rigidity spectrometers. They aim to measure curved trajectories of charged particles in a powerful magnetic field. This method can estimate charge, charge sign, magnetic rigidity and velocity. From this information, momentum, mass and kinetic energy can be derived. The magnetic rigidity spectrometers can measure energies up to approximately 1 TeV. To reach higher orders of energy, ionisation calorimeters are utilised. This technique is commonly used to measure the energies of electrons and hadrons at accelerators, but unique requirements for space instruments do not allow using these accelerator calorimeters. The space calorimeters need to be thin, which supports higher resolution for electrons and photons but limits resolution for hadrons. [2]

Lastly, an indirect approach must be used to measure the extremely high energies of some cosmic particles. That is because the detectors are too large for direct techniques. Indirect measurements include, for example, using a radio signal, detecting X-ray synchrotron photons or the fluorescence technique, which uses excited atmospheric nitrogen. [2]

1.2 Orbits and satellites

Both natural and artificial objects move around other objects in space along a curved path called an orbit. The curvature of the path is caused by gravity and momentum. Objects are attracted to each other due to gravity, and they can begin to orbit each other when they have enough momentum. Objects orbiting other objects are called satellites and are divided into natural ones (such as moons and planets) and artificial ones, which are manufactured. Artificial satellites orbiting Earth are used mainly for telecommunication, astronomy observation, weather forecast and navigation. Satellites orbiting other celestial bodies serve mainly as scientific experiments, gathering essential data. [5]

An orbit, its orientation around the central body and the position of a satellite in orbit are given by satellite orbital elements or Keplerian elements (see Figure 1 and Table 1).

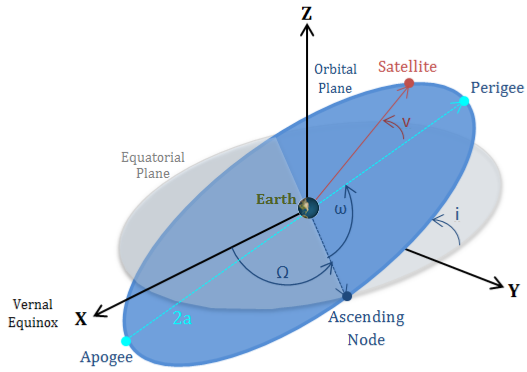


Figure 1: Keplerian elements. [6]

symbol	name	meaning
a	semi-major axis	the semi-major axis of the ellipse which defines the orbit
e	eccentricity	shape of the orbit; a circular orbit has eccentricity of zero
i	inclination angle	angle between the orbital plane and the central body's equator
Ω	RAAN	rotation of the orbital plane and reference axis
ω	argument of perigee	angle between the ascending nodes and the perigee point, measured along the orbit in the direction of the satellite's motion
ν	true anomaly	location of the satellite on the orbit

Table 1: Keplerian elements. (RAAN .. Right Ascension of the Ascending Node) [6]–[8]

1.2.1 Types of orbits

Each mission has different objectives and thus needs a specific orbit. There are several commonly used orbits around Earth with specified parameters. Apart from the orbits mentioned below, there are several others like polar orbit, Sun-Synchronous Orbit (SSO), Geostationary Transfer Orbit (GTO) or Lagrange points.

- Geostationary Orbit/ Geosynchronous Equatorial Orbit (GEO)

  A geosynchronous orbit is a circular orbit with an altitude of 35 786 kilometres, allowing satellites to synchronise with Earth's rotation. Moving with a speed of around 3 kilometres per second, a satellite will finish one circle around Earth in 23 hours, 56 minutes and 4 seconds ([5]). A geostationary orbit is a geosynchronous orbit with zero eccentricity and low enough (ideally right on the equator) inclination that the satellite seems to stay above a certain Earth point. Using the feature of staying above one precise point on Earth all the time, this orbit is mainly used by telecommunication and weather monitoring satellites. Antennas on Earth can easily target telecommunication satellites without changing direction. Weather monitoring satellites can monitor weather trends in specific areas. [5], [7]

- Low Earth Orbit

  LEO is the closest possible Earth orbit with an altitude lower than 1000 kilometres, with the lower boundary being around 160 kilometres. Travelling at a speed of around 7.8 kilometres per second, it takes a satellite to orbit Earth in approximately 90 minutes. Due to no requirement on the inclination of LEO, many possible ways around the Earth can be used. This is one of the features that make LEO a frequently used orbit. LEO is used mainly for imaging, International Space Station (ISS) or even telecommunication. When used for telecommunication, satellites usually cooperate, creating a constellation constantly covering required areas. [5]

- Medium Earth Orbit (MEO)

  MEO altitude is limited by GEO from above and by LEO from below. Similarly to LEO, MEO is not restricted to a specific inclination. The orbit is used by satellites for many different purposes, one of them generally navigation. [5]

1.3 Satellite communication

When communicating with satellites in orbit, data is sent in packets. Packets sent to the S/C are called telecommands, and packets received from the S/C telemetries. Each packet has a header consisting of service type and subservice type numbers. A service is a group of telecommands and telemetries with a common targeting field, e.g. memory, housekeeping, specific sensor settings or measurements. Some service numbers are generally reserved for specific groups, and some are free to be defined according to a particular project. Individual packets in a particular service are subservices. In the headers of responses, there is a status byte which indicates if the command execution went as expected or if there occurred an error - the status can describe the type of the error. [9]

The Cyclic Redundancy Check (CRC) error-detecting method checks if any error occurred during data transfer. The CRC alone is a natural number calculated from message data using a given polynomial. It is then appended to the message and sent. At the receiving end, the CRC is acquired similarly, and if it matches the sent CRC, the message is considered error-free. CRC is usually attached to the end of each packet. [10]

To encode and decode packets, their format needs to be defined. A formal notation Abstract Syntax Notation One (ASN.1) was created to this purpose. It is not dependent on language implementation or the physical representation of the data. It provides several predefined basic types and several constructed types. It only serves to define the structures of packets but does not provide any tools to process the data. The ASN.1 is connected with several standardized ways of encoding the values specified in the ASN.1. [11] To give a user more ways to structure the memory layout of data structures in ASN.1, the ASN.1 companion language was created. [12]

To make the development of embedded real-time systems easier, European Space Agency (ESA), in collaboration with several participants from the space industry, developed a set of tools called TASTE. One of the supported technologies is above mentioned ASN.1 standard. In general, the goal of TASTE is to use automation to improve the software development life cycle. Among other things, it allows users to generate low-level error-prone code for micro-controllers. [13] To handle all data modelling for space applications, the ASN1SCC compiler for ASN.1 was explicitly developed for ESA. [14]

1.4 Used tools

The tools used for software development were chosen according to the system requirements (see section 3.1). As a programming language, Python 3.8 was chosen; for GUI design, the Tkinter module for Python was selected.

1.4.1 Python

The choice of Python as a programming language for the proposed software is mainly because of its portability. At esc Aerospace, the computers run on both Linux and Windows, so it is convenient to use a portable language. Because it is an object-oriented programming language, it allows high modularity. That is useful mainly for future uses as the 2SD instrument is already part of two projects with different communication protocols. Python also has an extensive standard library and supports many third-party modules. Moreover, Python does not have to be compiled to machine code ahead of time as some other languages. [15]

1.4.2 Tkinter

As a part of the Python standard library, Tkinter is one of the possible frameworks for GUI development. It is an interface to the Tk GUI library, which originated from the Tool Command Language (TCL). Compared to other frameworks like PyQt5, Tkinter is relatively simple and fast. Its development is slow and stable; thus, applications using Tkinter will have a lower probability of complication occurrence after new Tkinter releases. It is not overly complex, does not require installation of other modules and does not need much memory space. Since it is simple and uses basic widgets and their configuration, it can be used effectively and quickly. Only the knowledge of Python as a programming language is required. [16]

The proposed software shall be as minimalistic as possible, using just a few libraries. Therefore using Tkinter for creating its GUI is very practical. Another possible option could be Qt since it is widely supported, but the high complexity of this framework would not be a benefit in this case. Qt alone is a set of many libraries, requires an installation and has many unnecessary features to create a basic GUI. [17]

Chapter 2

Space Dosimetry System Demonstrator

2.1 Introduction

The 2SD instrument is an ionising particle monitor for space applications developed as a technology demonstrator. The instrument consists of two radiation detectors - SpacePix Radiation Monitor (SXRM) and Soft X-ray Monitor (SXM). The mission's primary focus is put on the SXRM detector containing five SpacePix2 sensors. Details will be discussed in section 2.2. The SXM detector containing an X-CHIP-03-SXR sensor is embedded to validate a different development branch of monolithic pixel detectors. This detector specialises in flux and spectrum measurements of soft X-ray photons emitted during transient events in the magnetosphere, such as X-ray flares during magnetic reconnection. The SXM detector is not part of this thesis, so it is not described, and the software will not consider it. The placement of individual detectors within the instrument is visualised in Figure 2. The qualification model is shown in Figure 3. [18]

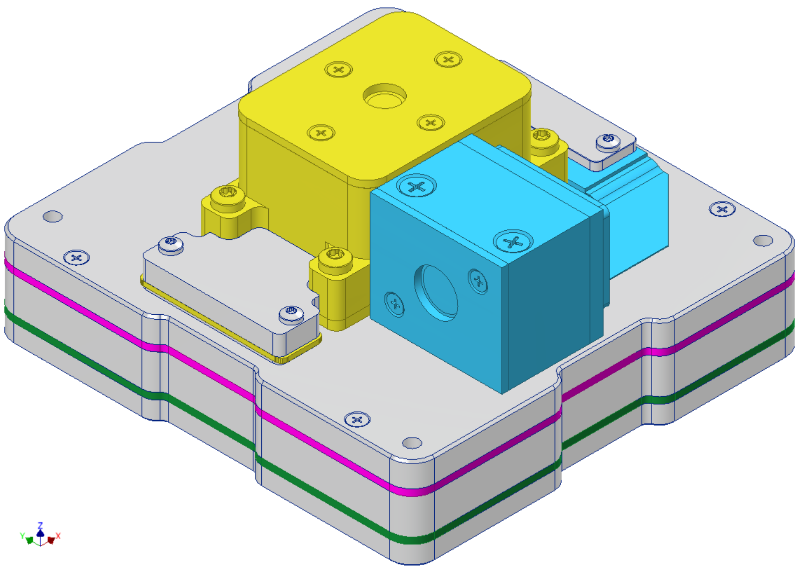
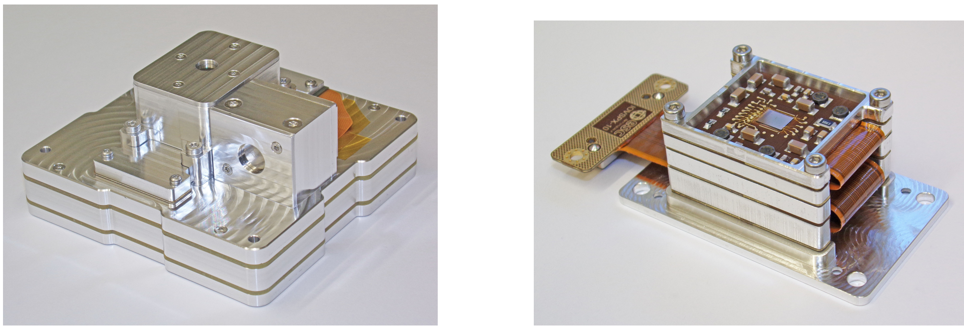


Figure 2: A 3D model of the 2SD instrument. Yellow - SXRM detector, blue - SXM detector, magenta - sensor board, green - motherboard. [18]



(a) : The 2SD instrument. **(b) :** The SXRM detector.

Figure 3: The qualification model.

The 2SD device has already been launched in the frame of the VZLUSAT-2 project. It was embedded in the project's satellite and is currently orbiting the Earth on LEO. Its purpose is to test the previous versions of both sensors and thus detectors in orbit. The new version of the 2SD device contains improved sensors. It is planned to be launched in the autumn of 2022 within the In Orbit Validation (IOV) project. [18]

2.2 SXRM detector

The SXRM radiation detector was constructed for a wide range of applications. As the device can monitor the radiation component of space weather, it can be considered for different types of satellites as (part of) a space environment awareness monitor. The most important part of this detector is an SpacePix2 sensor which is closely described in section 2.3. [18]

The principle of the SXRM lies in using the pattern recognition technique. The detector comprises five detection layers containing the SpacePix2 sensors interleaved with energy-absorbers. The layers are arranged into a telescopic configuration. This design allows the device to cover an extensive particle energy range. Thus it is sensitive to individual particles ranging from electrons and protons trapped in the van Allen belts to heavy ions from the GCR. The monitor can operate in a variable range of solar particle events with fluxes of up to $10^6\,\mathrm{protons cm^{-2} s^{-1}}$. Designed to last up to 15 years, the radiation monitor is considered for long-term orbital (LEO, MEO, GEO) and interplanetary missions. [18]

2.3 SpacePix-2-Lin-S ASIC

The SXRM detector is based on radiation-resistant, monolithic pixel detector called SpacePix2. The sensing elements are represented by 4096 pixels arranged in a grid of 64×64 pixels. Each pixel has the size of $60 \times 60\,\mu$m, which forms a sensitive area of $3840 \times 3840\,\mu$m on an approximately $4 \times 5.5\,$mm large detection chip. It was developed using the Silicon on Insulator (SoI)

Complementary Metal-Oxide-Semiconductor (CMOS) technology for space dosimetry and charged particle detection, with a 180 nm technology. [18], [19]

The purpose of the chip is to measure and visualise radiation and its interaction. That can be achieved by measuring the energy deposited in every single pixel in the pixel grid. The range of five orders of magnitude allows the sensor to distinguish particles from low-energy electrons to high-energy heavy ions. [19]

The operation cycle of SpacePix2 is divided into several phases, starting with Power-On, following with Readout Mode Selection, Configuration, Measurement and ending with Data Readout. In the two first phases, as their names suggest, the chip is powered on, and readout mode is selected. There are two possible readout modes, SPI mode and LVDS mode, the first designed to be used with a microcontroller while the second for high-speed operation controlled by Field Programmable Gate Array (FPGA). The most crucial phase of this work is the Configuration phase which is discussed in more detail in subsection 2.3.1. The phase is explained from the software point of view rather than the hardware one. Lastly, the Measurement and Data Readout phases are summed in subsection 2.3.2. [19]

2.3.1 Sensor configuration

Configuration of the SpacePix2 chip is essential, considering it affects the measurements and thus the acquired data. The sensor requires two types of configuration - pixel matrix configuration (later referred to as a local configuration) and global chip configuration (hereafter global configuration). It is important to remember that there are five sensors in the SXRM detector, so these configurations shall be set for all of them, whether the same or different. In order to accommodate more configurations for multiple sensors the Configuration groups have been implemented. A configuration group groups multiple sensor configurations (global and local) for all sensors of the detector.

The parameters for both types of configuration are listed in Table 2 and Table 3. [19]

parameter name	default value
TDAC[3:0]	1000
INJECT_EN	0
HIT_GLOBAL_EN	0

Table 2: List of local configuration parameters. The parameters can differ for each pixel. The default values are in binary. [19]

parameter name	default value	parameter name	default value
VBP_CSA	1000 0000	F_SEL_0	0
VBN_CSA	1000 0000	F_SEL_1	1
VFB_CSA	1000 0000	F_SEL_2	0
VBN_PDH	1000 0000	BACKSIDE_DEBUG_EN	0
VBP_HYST	1000 0000	VREF_EN	1
VBP_COMP	1000 0000	BACKSIDE_LOW_LEAK_EN	0
VBN_TDAC	1000 0000	BECKSIDE_INJECT_EN	0
VBP_LCC	1000 0000	ANALOG_OUT_0_EN	0
VTHR	10 0000 0000	ANALOG_OUT_1_EN	0
VBN_ADC	1000 0000	ANALOG_OUT_2_EN	0
LVDS_CM	1000 0000	ANALOG_OUT_3_EN	0
LVDS_STRENGTH	1000 0000	BACKSIDE_EN	0
SF	1000 0000	TEMP_SENS_EN	0
TAIL	1000 0000	ADC_PIN_EN	0
TEST	1000 0000		

Table 3: List of global configuration parameters. The default values are in binary. [19]

2.3.2 Measurement and Data Readout

The measurements phase starts with the the activation of the shutter signal. After the measurement phase is finished, it is followed by the data readout phase, in which the acquired events are digitized and read out. The shutter period parameter gives the length of exposition. This parameter should be at least 100 μs because it takes approximately 10 μs to recover from reset at the beginning of the shutter period. The maximum useful shutter period has to be detected experimentally, but a reasonable estimate is $100 - 200$ ms. During the exposition, each pixel activates its Peak Detector Hold (PDH) circuit. The circuit enables them to record the peak voltage of a Charge Sensitive Amplifier (CSA), which amplifies the signal coming from the sensitive diode. When the exposition ends, peak voltages from the PDH circuit are transferred to Analog to Digital Converters (ADCs), where they are digitized row by row. The resolution of each ADCs is 10 bit and ranges from 0 to 1023 mV, with each code value approximately corresponding to the voltage in millivolts. After digitization, data are loaded to the Row Shift Register (RSR) and transmitted out. [19]

Part II

Software design and implementation

Chapter 3

Design and architecture

To fulfil the aim of this work, the ConfPix software (later referred to only as ConfPix) was designed. This chapter presents both system and software requirements on ConfPix. It then describes the design of ConfPix from graphical and software points of view.

3.1 Requirements

The system requirements on ConfPix are listed below.
Software shall:

- be implemented in Python and be compatible with Python 3.8 and newer
- be based on GUI using Tkinter for Python
- allow to set full configuration (global and local) of each radiation sensor
- allow to read and set configuration of the instrument
- allow to start and stop data acquisition of the detector
- display acquired data

The software requirements specification derived from the system requirements, including verification methods can be found in Appendix C.

3.2 GUI design

The GUI of the application (see Figure 4) is based on the software requirements. It was created using the Tkinter module for Python. The main window is divided into multiple areas which are connection, configuration, configuration parameters, data acquisition and logging.

The connection part currently supports only connection via Universal Asynchronous Receiver-Transmitter (UART), but the design can be easily modified for additional communication interfaces such as Ethernet. The configuration and configuration parameters parts are part of a SpacePix tab. The choice of tabs is a preparation for a possible upgrade of the application for configuration of another type of detector such as the SXM (the secondary radiation detector of the 2SD instrument). The configuration parameters part changes according to the selected configuration type (global/local).

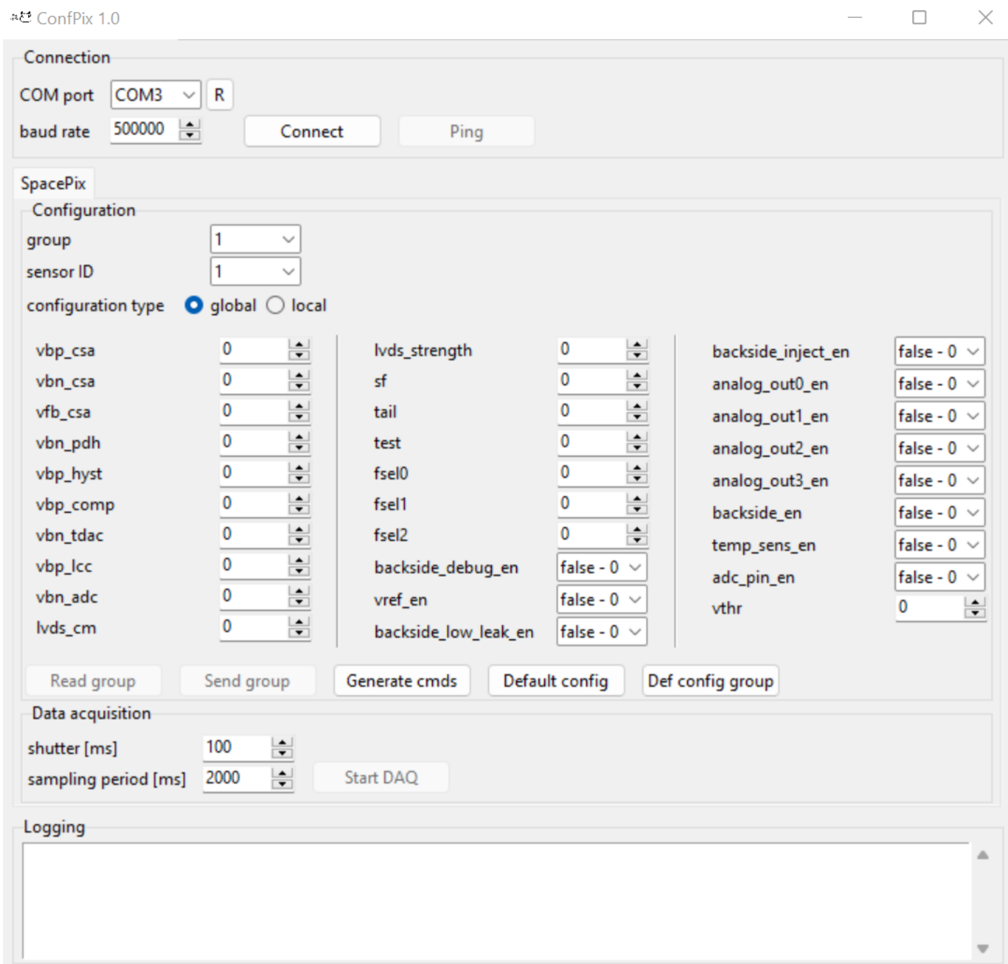


Figure 4: GUI design of ConfPix after startup.

The global parameters' layout is shown in Figure 4. It allows the user to set all of the global configuration parameters. The parameters with the boolean data type have the `Combobox` widget to choose from `true-1` and `false-0`. The value is displayed in human-readable form (true/false) but also in the form of actual numerical value in order to avoid doubts in case of the use of inverse logic (for future parameter updates). For the other parameters, the `Spinbox` widget is used with set limits.

The local configuration layout is shown in Figure 5. The user can select one pixel, a range of pixels or all pixels from the pixel grid. The local configuration parameters will be changed only for the selected pixels. The configuration section also contains buttons to read and send configurations for the selected group and only the selected type of configuration. The `Generate commands` button generates commands for the selected group and both configuration types regardless of the selected type. The `Default config` button sets the configuration parameters for the selected sensor to default values, and the `Def config group` button sets the configuration parameters for all sensors in the selected configuration group to default values.

In the data acquisition part, both parameters needed for data acquisition - the shutter duration and the sampling period - can be set. If the instrument is connected, the Start DAQ button starts data acquisition using the currently selected configuration group, opens the data acquisition window and changes its own name to Stop DAQ. The Stop DAQ button closes the data acquisition window, stops data acquisition, and changes the button's name to Start DAQ. In the logging part, the logged messages are shown to the user. The state of sending/reading is shown when sending and reading configurations.

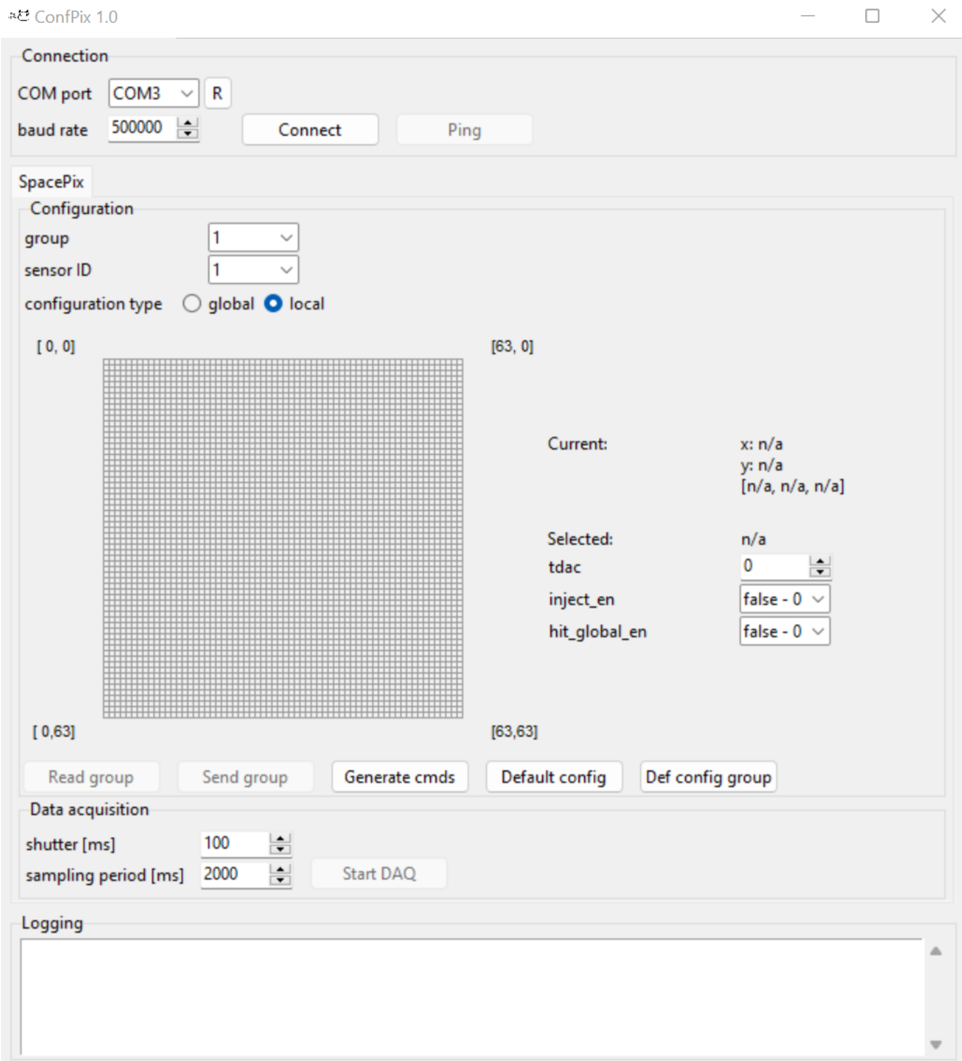


Figure 5: GUI design when local configuration is chosen.

The design of the data acquisition window is shown in Figure 6. There are currently five sensors in the detector, so there are five sensors in the window.

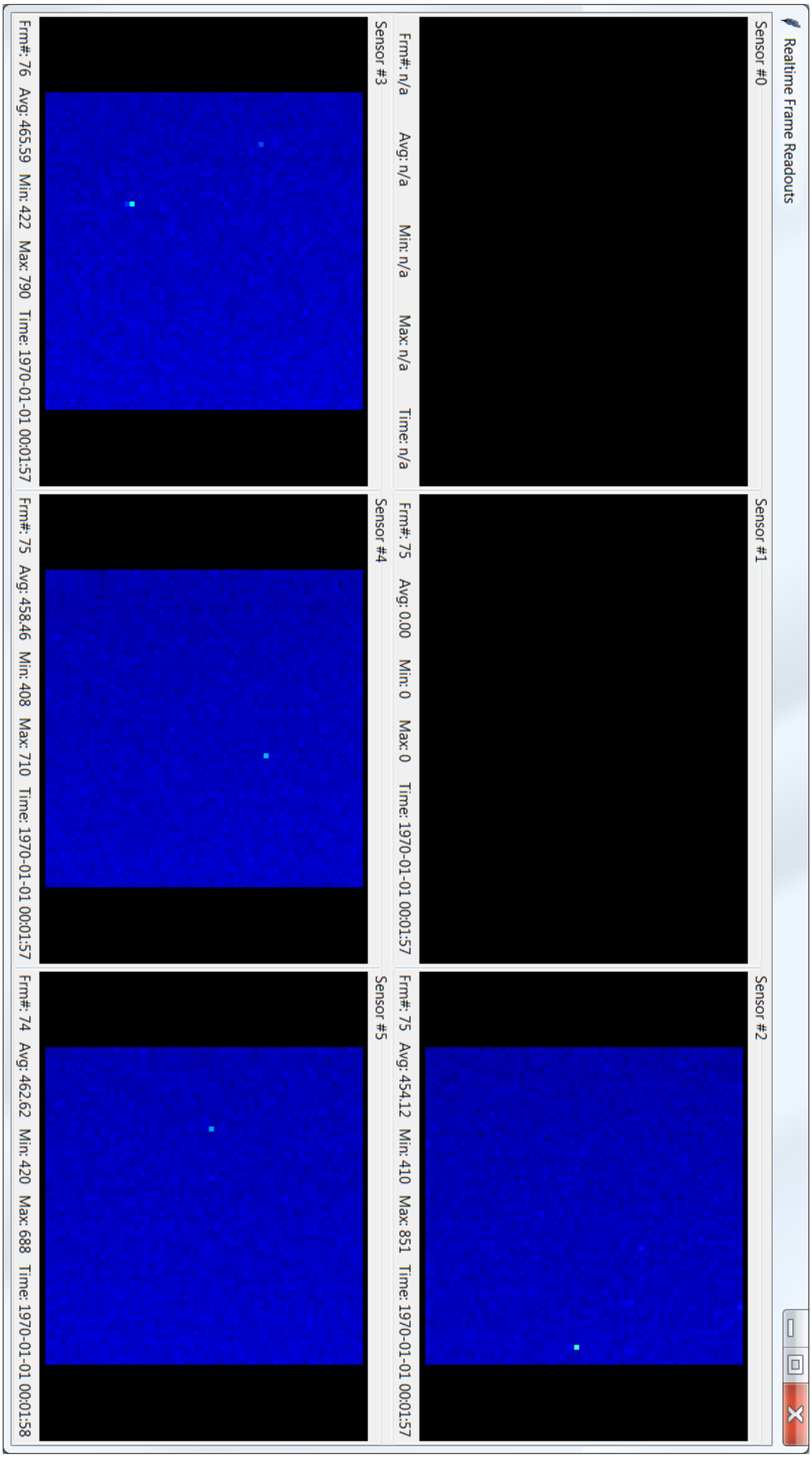


Figure 6: Data acquisition window after starting data acquisition (taken on qualification model of the 2SD). (Sensor #0 is there just to fill space, it does not represent an actual sensor. Sensor #1 was mechanically damaged, otherwise would also show acquired data.

This can be easily changed if more sensors were added.

3.3 Static architecture

The software's architecture is based on a variant of the Model-View-Controller (MVC) pattern. The functionality of one of MVC's variations is shown in Figure 7. The model serves as the data part in the pattern and does not deal with GUI widgets, data presentation, or data processing. The view shows data and control widgets to the user, practically comprising the GUI. Generally, the view does not have to have access to the model, and if it does, it is usually read-only. Lastly, the controller's job is to take care of the user's requests and data flow between the model and the view.

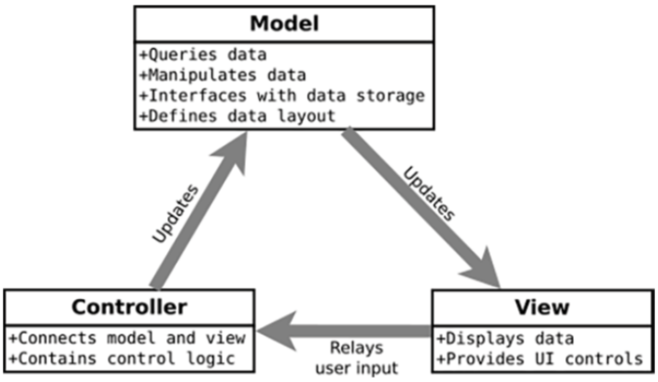


Figure 7: Individual parts of the MVC pattern, including their relations and their main functions. [20]

The ConfPix's structure differs from the one in Figure 7 in two main aspects. First, the view does not have access to data via model but only via the controller. Second, the data between view and controller goes both ways, as well as the data between model and controller. The MVC pattern only shows the main blocks of the design, while individual modules and their relations are shown in Figure 8.

Classes, their attributes and methods are presented in the UML diagram in Figure 9. The controller class is specified in Figure 10, the local configuration and its parts in Figure 11, and finally, the data acquisition class and its parts in Figure 12. The functionality of individual modules is further described in section 3.4.

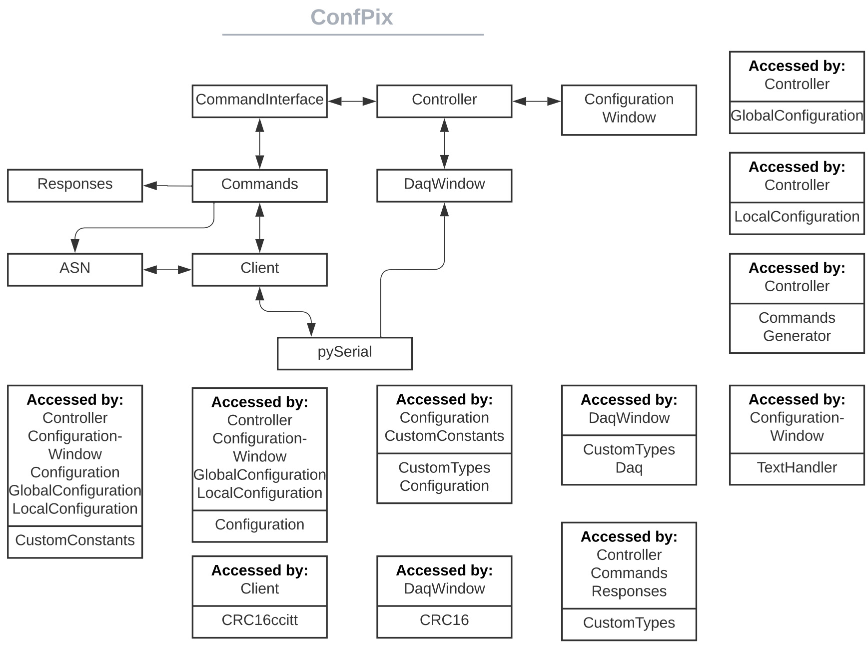


Figure 8: Static architecture of ConfPix.

3.4 Modules' description

This section describes individual modules of the ConfPix software. Most of the modules are grouped into three groups - IOV, UART and View. Main modules stand alone.

Some of the modules were automatically generated from an Interface Control Document represented in the form of a structured text file in YAML format. A unique pair of a service number and a subservice number describes each packet - service numbers group packets with the same coverage. Additionally, one whole module was entirely generated from ASN and ACN files. These files contain almost the same data as the YAML file but in different format. The corresponding part describes further details.

- ConfPix

  The confpix module is the main module and serves to run the program.

- Controller

  An instance of the Controller class manages all events triggered in the GUI. It takes care of buttons' behaviour by getting the data from the user and relaying them into an instance of Command Interface. It then reacts to the response from Command Interface and, if required, passes data back to the GUI. It controls the validity of input data, such as its data type or being within the required interval.

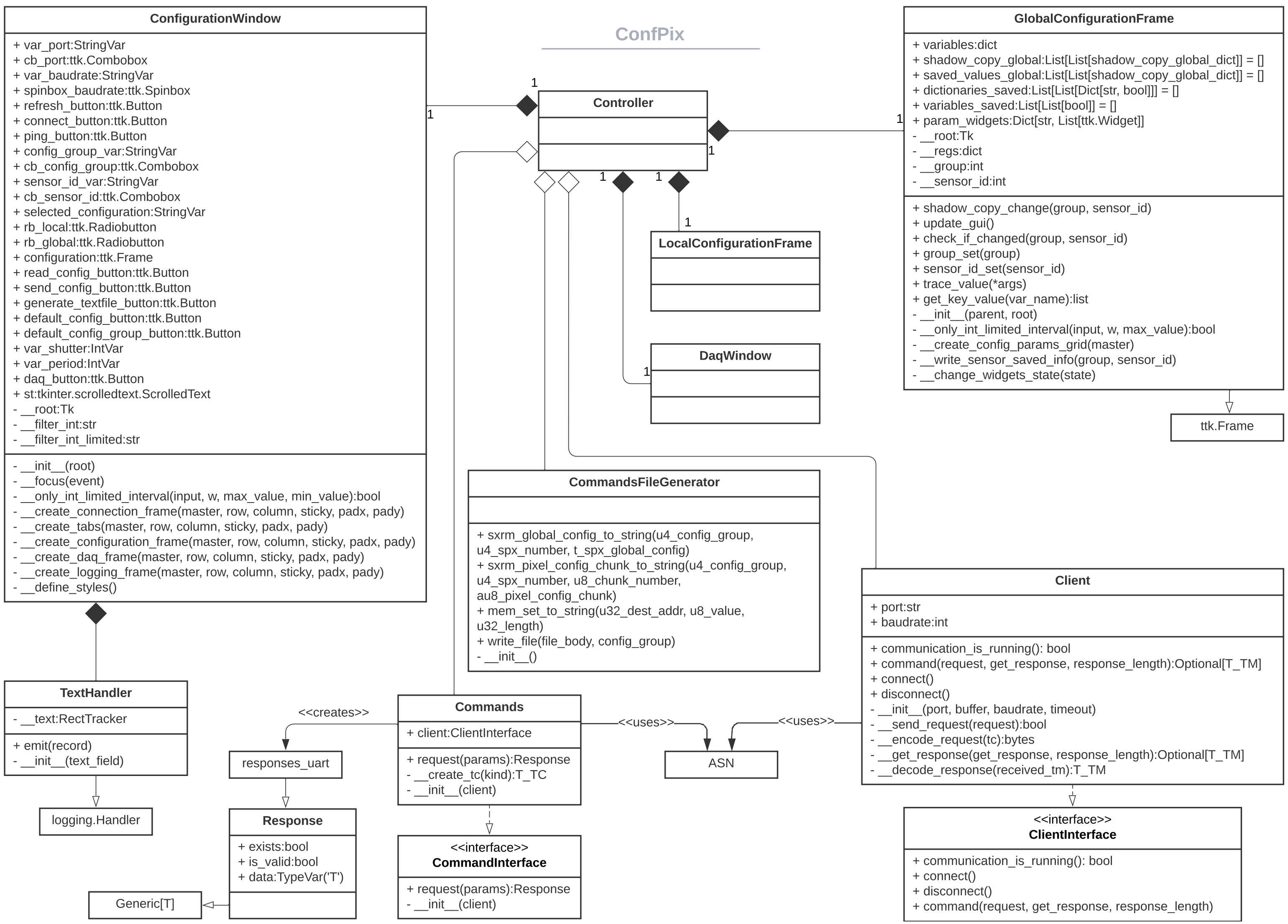


Figure 9: Class diagram of ConfPix.

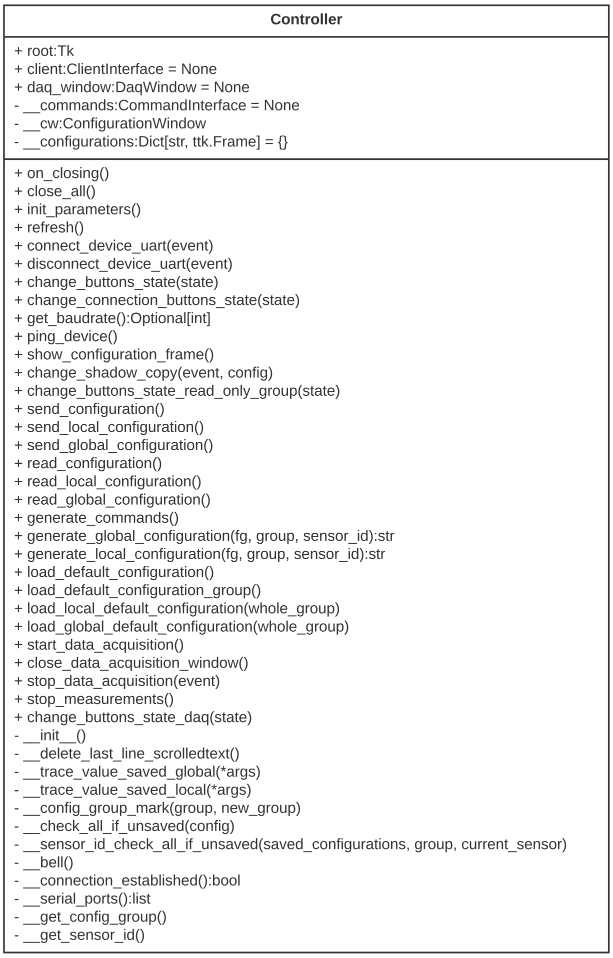


Figure 10: Class diagram of Controller.

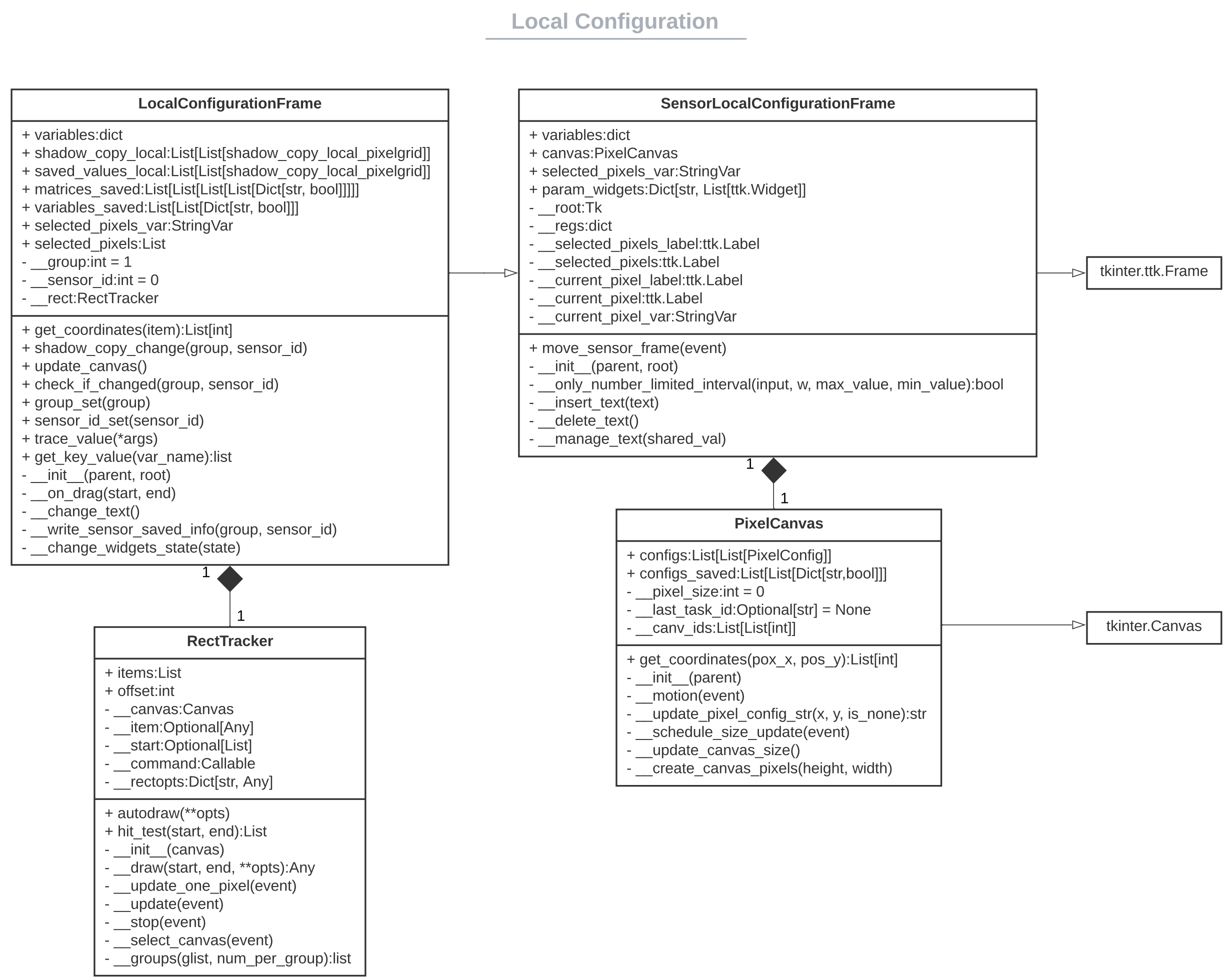


Figure 11: Class diagram of Local Configuration.

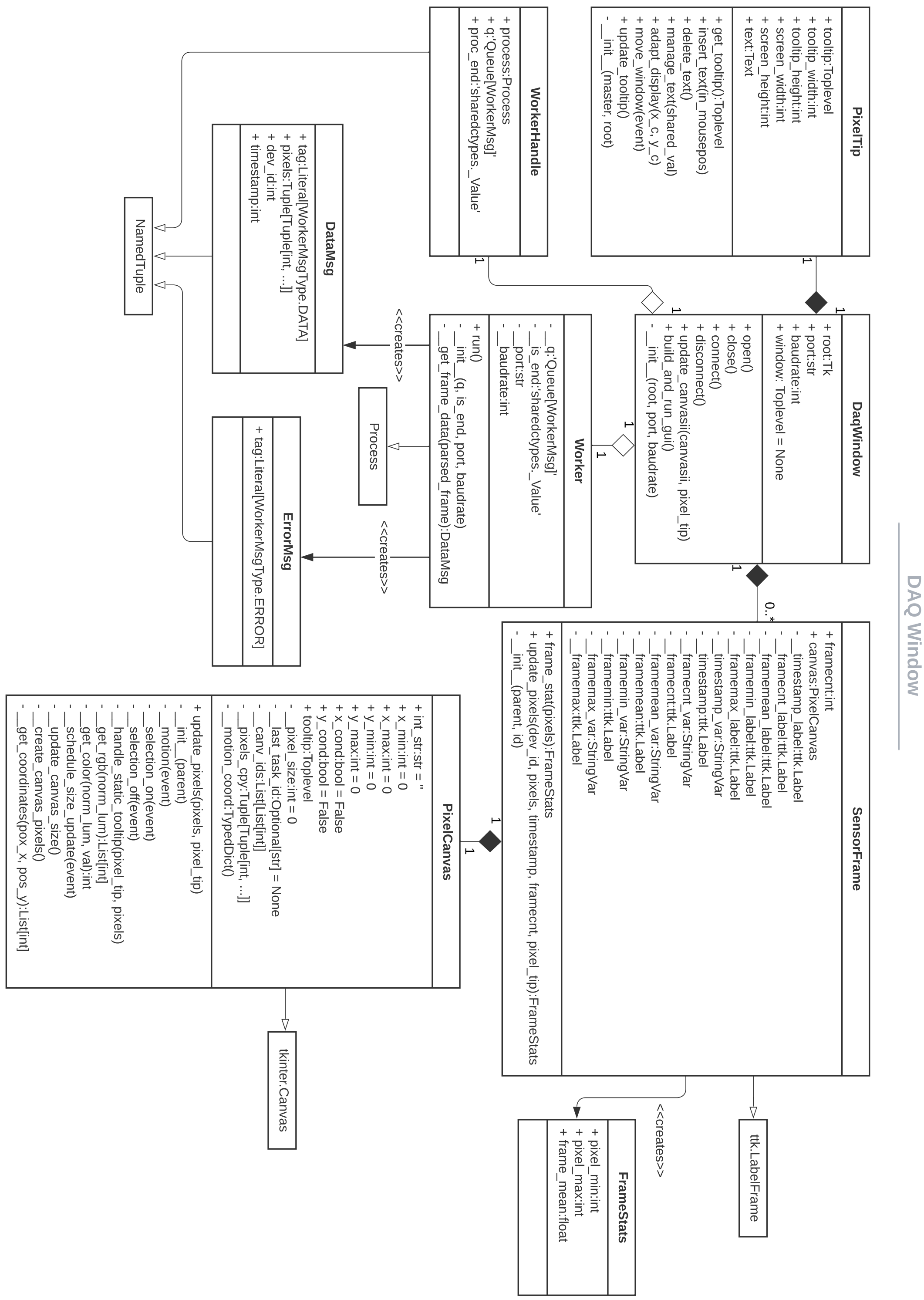


Figure 12: Class diagram of DAQ Window.

Opening and closing GUI window also fall within its function's scope. Lastly, the controller logs events in the application, displays them in the GUI and saves in a LOG file.

- Client Interface

  The ClientInterface provides a protocol-independent interface that allows communication link establishment and performing data transfers. The methods which the interface provides are listed below.

  - `connect()`
  - `disconnect()`
  - `communication_is_running()`
  - `command(self, request, get_response: bool = True, response_length: Optional[int] = None)`

  The `connect()` and `disconnect()` methods start and stop communication. They do not take any arguments since the type of communication is unknown, and the arguments might differ for each type. The `communication_is_running()` method is used to check if communication was established. It returns true if the communication is open and false if it is closed. Method `command()` handles sending the command requests to the instrument and optionally getting a response if requested. The method accepts three arguments - one compulsory and two optional. A command request object is compulsory, and its data type depends on the type of communication. Argument `get_response` is optional and provides information about whether a response is expected or not. For most of the commands, a response is required. That is why the flag is set to true by default. Lastly, argument `response_length`, which is also optional, serves for possible faster data exchange. The length is considered unknown if it is set to None, which is by default. The implementation of this interface for UART is described below in subsection 3.4.2 - Client UART.

- Commands Generator

  After creating a required configuration on the spare instrument, the configuration needs to be sent to the device in orbit. For that, a file containing commands following the control software's command-line interface format is needed. To generate such a file, the CommandsFileGenerator class is used.

  The CubeSatCarrier2 communicates with the ground via ground station terminal software which is emulated for development by Satellite Interface Simulator (SIS) commanding interface. The generator produces a file with commands to set global and local configurations for the chosen configuration group in the SIS commanding interface format. For this purpose, the class provides several methods listed below.

- `sxrm_global_config_to_string(self, u4_config_group: int, u4_spx_number: int, t_spx_global_config: List[str])`
- `sxrm_pixel_config_chunk_to_string(self, u4_config_group: int, u4_spx_number: int, u8_chunk_number: int, au8_pixel_config_chunk: List[int])`
- `mem_set(self, u32_dest_addr: int, u8_value: int, u32_length: int)`
- `write_file(self, file_body, config_group)`

The first three methods generate strings from inserted parameters in the required format. The `mem_set()` method can be used instead of the `sxrm_pixel_config_chunk_to_string()` method if all pixels for one sensor and configuration group have the same configuration value. The benefit of this switch is a faster data exchange. The `write_file()` method creates a unique file name and produces a new file with this name. The file includes commands passed on to this method as an argument.

3.4.1 IOV modules

- Command Interface

  CommandInterface interface provides methods to command the 2SD instrument. It was automatically generated from the YAML file already mentioned above. The interface does not depend on the type of communication used; the classes implementing this interface do. Each method corresponds to one subservice, taking all parameters of this subservice as the method's arguments. The methods return an instance of the `Response` class. The class has three attributes - `exists`, `is_valid` and `data`. If response was created, the `exists` parameter is set to `True`. If data received are valid, the `is_valid` parameter is set to `True`. Even if data are not valid, the data received are stored in the `data` parameter. The interface requires one attribute, which is the client, a ClientInterface instance.

- Commands UART

  The Commands UART module includes the class Commands which inherits from the CommandInterface interface (see section 3.4 - Command Interface). This class is designed for communication via UART; thus, its attribute client is an instance of the Client class, described later. The class was automatically generated from the YAML file.

  Nearly every method returns an instance of the Response class. As an argument, each instance of the Response class is given a data type of the response. Module Responses UART (see below) contains all possible responses, each being a single class. Some subservices and thus methods in the Commands class do not have a response. That information is passed on as a parameter to the `command()` method of the client attribute. The method then does not expect any response.

If the response cannot be read or does not correspond to the expected format, the response is not created, and the attribute `exists` is set to `False`. If a response exists, but the status returned is not 0 (meaning some error occurred), or the CRC returned is equal to 0 (meaning some error might have occurred), the `is_valid` attribute is `False`.

An example of a command method is shown in Listing 1.

- Responses UART

  As already indicated above, module Responses UART consists of many classes. Each serves as a response to only one request. Every time a request is sent, a new instance of the particular response class is created. Each class only requires one argument, a decoded response with a `T_TM` data type. When a new instance is created, all class attributes are extracted from this argument. The `T_TM` data type is described below in the ASN paragraph (subsection 3.4.2) and can be used only for communication via UART. A new Responses module would have to be created for a different type of communication. Since this is an automatically generated module from the YAML file, it would only require changing the rules for creating a single class, and a new module could be generated.

  An example of a response class is shown in Listing 2.

- Custom Types

  Custom Types is a module which supports both Commands UART and Responses UART modules. It consists of enum classes and classes simulating structures. These classes were generated from the YAML file. Their purpose is to make the file more readable, avoid duplicates, and restrict parameters to only used values. The classes are designed in a way that helps to create more object-oriented responses.

```python
def power_data_request(self, u8_pwr_rail_id: int) ->
↪  Response[responses.power_data_response]:
    """(3, 138) Gets full power data from selected power rail
    """
    tc = self.__create_tc(DV.tc_3_138_PowerDataRequest_PRESENT)
    tc.sentData.tc_3_138_PowerDataRequest.u8PwrRailId.Set(u8_p⌋
    ↪  wr_rail_id)

    UART_response: T_TM = self.client.command(tc)
    response = responses.power_data_response(UART_response)
    exists = response.response_created
    is_valid = None
    if exists:
        is_valid = (response.crc != 0 and response.status == 0)

    return Response(exists, is_valid, response)
```

Listing 1: Example of a command method.

```python
class power_data_response:
    """(3, 139) Gets full power data from selected power rail
    """
    def __init__(self, tm:T_TM):
        try:
            self.crc = tm.crc.Get()
            self.status = tm.status.Get()
            self.u8_pwr_rail_id = tm.replyData.tm_3_139_PowerD⌋
            ↪  ataResponse.u8PwrRailId.Get()
            self.b_power_rail_status = bool(tm.replyData.tm_3_⌋
            ↪  139_PowerDataResponse.bPowerRailStatus.Get())
            self.b_readout_validity = bool(tm.replyData.tm_3_1⌋
            ↪  39_PowerDataResponse.bReadoutValidity.Get())
            self.u6_reserved = tm.replyData.tm_3_139_PowerData⌋
            ↪  Response.u6Reserved.Get()
            self.u16_current_ua = tm.replyData.tm_3_139_PowerD⌋
            ↪  ataResponse.u16CurrentUa.Get()
            self.u16_voltage_mv = tm.replyData.tm_3_139_PowerD⌋
            ↪  ataResponse.u16VoltageMv.Get()
            self.response_created = True

        except:
            self.response_created = False
            logger.error(f'Creating response failed.')
```

Listing 2: Example of a response class.

3.4.2 UART modules

- Client UART

  The client UART module contains one class - Client, which implements the ClientInterface interface (see section - Client Interface). It provides a connection with the device via a serial link. In addition to those methods inherited from the interface, the class has several other methods to handle the requests and responses. One pair of methods serves to encode and send a request to the device. The other pair serves to receive and decode a response from the device.

  The request parameter in method `command()` has a `T_TC` data type, and the method returns a response with a `T_TM` or `None` data type. To process the request it calls `__send_request()` and `__get_response()` methods and returns response built by the latter. The `__get_response()` tries to read the answer from the device and decode it. If an error occurs in any process step, it returns None to signalize that fact.

- ASN

  The ASN module includes the following files: DV_Types.py, DV.py, iov_asn.py, Stubs.py, iov_getset.so and winasn_64b.dll. All these files were generated from iov.asn and iov.acn files using the ASN1SCC compiler. In the ASN and ACN files, all packets are described with all their parameters. The ACN file specifies individual memory sizes, especially for enums, which take less than one byte. It also connects service and subservice numbers with corresponding telecommands and telemetries.

  PY files and the SO file can be generated using a script (not part of this thesis) which first generates C files and, from those, the PY files and SO file. The DLL library can be generated using a different script (not part of this thesis). The script also generates HTML and CSS files, which serve as documentation.

- CRC 16 ccitt

  This module serves for CRC calculation when sending packets to the device. The CRC is two bytes of information appended at the end of a packet, calculated from the data sent. It is used only by the Client class (see subsection 3.4.2 - Client UART).

- CRC 16

  This module serves for CRC calculation when reading frames from the device after data acquisition is started. It is calculated to verify that no data were lost or changed during the transmission. It is used only by the Worker class (see subsection 3.4.3 - DAQ Window).

  The difference between CRC 16 and CRC 16 ccitt modules is in the parameters used. All parameters used for CRC calculation - Polynomial, Initial Value and Final XOR Value - differ for both modules. While the CRC 16 ccitt module uses a 0x1021 polynomial, a 0x1D0F initial value and a 0x0000 final XOR value, the CRC 16 module uses a 0xD175 polynomial, a 0xFFFF initial value and a 0xFFFF final XOR value.

3.4.3 View modules

- Configuration Window

  The Configuration Window module contains a `ConfigurationWindow` class. The class inherits from the `ttk.Frame` class and takes one parameter - the root. The root has the `Tk` data type and is used as the master when initialising the frame. `ConfigurationWindow` is the primary GUI handler which means it builds the main window and all the widgets it contains. The placement of individual widgets is described in more detail in section 3.2. This module only handles the visual side; it does not manage any data. The only action which could be considered data handling is limiting inputs of several widgets to integers only. With the command from the controller, it can also set the maximum value and will not allow the user to set higher values into the specified widgets. This feature shall limit the values in case of user's error.

- Custom Types Configuration

  Both local and global configurations have several parameters to set (see Table 2 and Table 3). Custom Types Configuration module contains one class - `Parameter` - to make handling configuration parameters easier. The class inherits from the `NamedTuple` class from typing module.

  The `Parameter` class was designed to make configuration parameters easier to edit and prepare for prospective future changes in design and thus in parameters. It has the following attributes: name, type, `default_value`, `max_value`, row and column. The name of a parameter has to be unique because later, it serves as a key in many dictionaries. The parameter type is a string and currently supports 'int', 'bool' and other types, meaning any other string will be handled uniformly. The default value is a string because that is the value which will be shown to the user. For example, instead of 1, which represents true, showing a 'true-1' string would be easier for the user to read. On the other hand, the max value is an integer because that is the maximum value which can be sent to the device. Attributes row and column can be either integers or None. If set to None, this attribute is ignored; if set to an integer, the software will try to organize the parameters grid in GUI accordingly. This feature serves users who could find out that some parameters need to be changed more often than others and would want to have them next to each other or, for example, in a particular column. For example if no parameter has preferences set (see Listing 3) the GUI layout would look like the one in Figure 13a. If the user would set the parameters as in Listing 4 (the rest of the parameters would stay the same), the GUI layout would look like the one in Figure 13b). Users can change the number of columns for global configuration too. It can be changed in the Global Configuration module (see below).

```
Parameter('backside_en', 'bool', 0b0, 0b1, None, None),
Parameter('temp_sens_en', 'bool', 0b0, 0b1, None, None),
Parameter('adc_pin_en', 'bool', 0b0, 0b1, None, None),
Parameter('vthr', 'int', 0x200, 0b1111111111, None, None)
```

Listing 3: Example of parameters with no preference of row and column.

```
Parameter('backside_en', 'bool', 0b0, 0b1, None, None),
Parameter('temp_sens_en', 'bool', 0b0, 0b1, None, None),
Parameter('adc_pin_en', 'bool', 0b0, 0b1, 1, 0),
Parameter('vthr', 'int', 0x200, 0b1111111111, 0, 0)
```

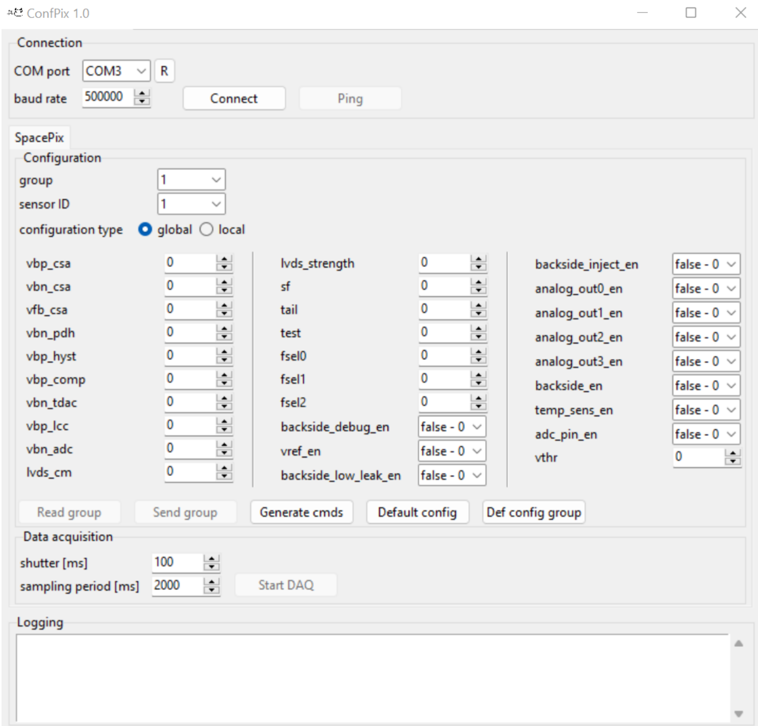
Listing 4: Example of parameters with chosen preferences of row and column. Parameter `adc_pin_en` will be in row 1 in column 0 and parameter `vthr` will be in row 0 in column 0.

- Configuration

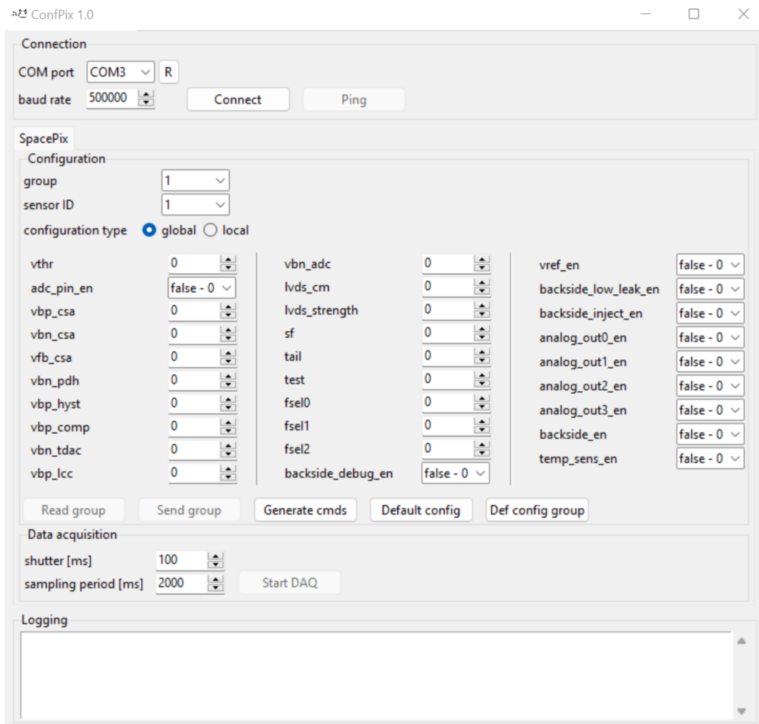
  The Configuration module provides several methods to make handling configuration parameters partly automatic. These functions are described below. It also provides a method which limits widgets' inputs on integers and a method which does the same but also sets the upper limit to the given value. The methods are used in the Controller, Configuration Window, Local Configuration and Global Configuration modules.

  - `create_variables(parameters: List[Parameter])`
  - `get_default_config(parameters: List[Parameter], zeros: bool = False)`
  - `get_bool_parameters_dic(parameters: List[Parameter], value: bool = True)`
  - `get_bool_str(val: int)`
  - `get_int_from_str(val: str)`
  - `grid_parameter(master: Misc, parameter: Parameter, row: int, variables: dict, reg, add_space_label: bool = False)`

  The methods `create_variables`, `get_default_config` and `get_bool_parameters_dic` all take a parameter 'parameters' which is a list of Parameter instances and return a dictionary with the names of parameters as keys. For the first mentioned, the values are corresponding Tkinter variables - `IntVar` or `StringVar` depending on the parameters' types. These are connected to parameter widgets and allow tracing variable changes caused by the user. For the second mentioned, the values are the



(a) : Global configuration GUI layout without any row and column preference.



(b) : Global configuration GUI layout with parameter `adc_pin_en` in row 1 in column 0 and parameter `vthr` in row 0 in column 0.

Figure 13: Comparison of global configuration GUI layout before and after setting row and column preferences for `adc_pin_en` and `vthr` parameters.

parameters' default values or zeros (or string equivalent) if parameter `zeros` is set to True. For the last function, the values are set to True or False depending on the parameter `value`. The parameter is set to True by default.

Methods `get_bool_str` and `get_int_from_str` convert integers (0, 1) into string representations of booleans (currently 'false - 0' and 'true - 1' are used) and string representations of booleans back to integers. The last method is the `grid_paramter` method. It creates a label and a corresponding widget for the given parameter. It connects the widget with a matching variable from the variables dictionary, configures widgets' styles and sets the master of both widgets to the master, which was passed on as a parameter. It finally grids both widgets into the given row and columns 0 and 1, the label being in column 0.

- Local Configuration

  The local Configuration module consists of the following classes:

  - `PixelCanvas(Canvas)`
  - `RectTracker`
  - `SensorLocalConfigurationFrame(ttk.Frame)`
  - `LocalConfigurationFrame(SensorLocalConfigurationFrame)`

  The `PixelCanvas` class inherits from Tkinter's Canvas class and is responsible for creating the pixel grid for local configuration. It keeps current and saved configurations for all pixels in the grid. It does not have information about the sensor and the configuration group. The class defines a method bound to the cursor's movement and updates the coordinates and configuration of the pixel to which the cursor currently points.

  The `RectTracker` class defines methods allowing users to select a rectangle, set of rectangles or all rectangles from a canvas. When combined with the `PixelCanvas`, the `RectTracker` highlights currently selected pixels by changing their colour. In this way, the user may select pixels which shall be configured.

  The `SensorLocalConfigurationFrame` class has a `PixelCavas` object as an attribute. It also has other attributes - widgets to show configuration parameters, information about selected pixels, and the pixel to which the cursor is pointing.

  The `LocalConfigurationFrame` inherits from the `SensorFrame` class and adds dynamical behaviour to it. First, it binds the selection of pixels using a `RectTracker`'s instance to the `PixelCanvas` instance. Secondly, it binds updating the cursor position in the `PixelCanvas` to the cursor's motion. It keeps the shadow copy of the pixels' configuration and saved values for all pixels, sensors and configuration groups. The controller works with an instance of the `LocalConfigurationFrame` class and the layout when the user chooses the local configuration type.

- Global Configuration

  The Global Configuration module, unlike the Local Configuration one, consists only of one class, `GlobalConfigurationFrame`, which inherits from the `ttk.Frame class`. Since global and local configurations inherit from the `ttk.Frame` class, it is easy to switch them in the main window. The `GlobalConfigurationFrame` class, similarly to the `LocalConfigurationFrame` one, keeps the shadow copy of configuration values and saved values for all sensors and configuration groups. The module also defines several constants to arrange the global configuration's parameters into columns.

- Custom Types DAQ

  Custom Types DAQ module contains four classes, all inheriting from the `NamedTuple` class from typing module, and one enum class to support handling serial communication and measured data during data acquisition. These classes are:

  - `WorkerMsgType(Enum)`
  - `DataMsg(NamedTuple)`
  - `ErrorMsg(NamedTuple)`
  - `WorkerHandle(NamedTuple)`
  - `FrameStats(NamedTuple)`

  The `WorkerMsgType` is an enum with two values - DATA and ERROR. It serves to distinguish between valid frames and invalid frames. The `ErrorMsg` has only one attribute - a tag set to `WorkerMsgType.ERROR`. `DataMsg`, on the other hand, except for a tag attribute set to `WorkerMsgType.DATA` has also attributes `pixels`, `dev_id` and `timestamp`. `FrameStats` class has three attributes - `pixel_min`, `pixel_max` and `frame_mean` - and serves to keep statistics about data frames. The `WorkerHandle` class helps to access serial communication better. It has three attributes - process, q (`Queue[WorkerMsg]` data type), and `proc_end`.

- DAQ Window

  The DAQ Window module consists of the following classes:

  - `PixelTip()`
  - `PixelCanvas(Canvas)`
  - `SensorFrame(ttk.LabelFrame)`
  - `Worker(Process)`
  - `DaqWindow()`

`PixelTip` class represents a small window appended to the cursor when the cursor is above a `PixelCanvas`. The window contains essential information about a particular pixel, like the configuration value and pixel coordinates. The window is hidden when the cursor is not above a `PixelCanvas` instance.

The `PixelCanvas` class is similar to the one in the local configuration module. It does not keep shadow copies or saved values. It calculates the colour of each pixel according to its value.

The `SensorFrame` class represents data received from one sensor. Except for the pixel canvas, it also has several variables to keep frame statistics.

The `Worker` class inherits from the `Process` class and represents serial communication. It takes `q`, `is_end`, `port` and `baudrate` as parameters and starts new serial communication with the instrument. An instance of serial communication cannot be shared between classes, so the old connection needs to be stopped, and this new one started. If any error occurs when the serial communication is started, an instance of `ErrorMsg` class is put to the Queue. If the communication is started, the Worker will read and process the data received and put an instance of `DataMsg` into the Queue.

The `DaqWindow` class builds a data acquisition window representing all the instrument's sensors in the selected configuration group. It opens a serial communication via a `Worker` instance, processes the data from the Queue and shows them to the user. When the window is closed or when the STOP DAQ button is pressed, the `DaqWindow` stops the communication and closes the window. New serial communication is started from the controller.

- Text Handler

  The `TextHandler` class inherits from the `logging.Handler` class. It overrides the `__init__()` method and sets its attribute `__text_field` to the passed parameter. The only other method - `emit` - append a logged message to the `__text_field` widget.

- Custom Constants In the Custom Constants module, the given information is set in the form of constants. The information includes parameters' limits, ranges, widgets' sizes, and parameters for global and local configurations. The parameters are set as an array of instances of the Parameter class.

Chapter 4

Tests

To test the ConfPix, several Test Cases were designed. Since the software has a GUI, manual testing was proposed. Individual test cases are listed in Appendix F. Firstly the engineering model was used for testing. The qualification model was used for the final verifications. For each test, the prerequisites had to be met, and the required tools had to be prepared. After that, the test case's scenario was followed one step at a time. The scenarios need to be strictly followed so the test is repeatable and its results from several runs are comparable.

The mapping of software requirements to test cases and of test cases to software requirements is illustrated in tables in Appendix D and Appendix E, respectively. Results of individual test cases are shown in Table 4.

Test case ID	result
TC-CP-001	PASSED
TC-CP-002	PASSED
TC-CP-003	PASSED
TC-CP-004	PASSED
TC-CP-005	PASSED
TC-CP-006	PASSED
TC-CP-007	PASSED
TC-CP-008	PASSED
TC-CP-009	PASSED
TC-CP-010	PASSED
TC-CP-011	PASSED

Table 4: Test results.

The Test Cases only include software requirements with the test verification method. The requirements with the review verification method were verified by reading relevant parts of the documentation. The requirements with the inspection verifying method were verified visually in the code.

Since all tests passed and all the system requirements were covered during the testing, the software can be considered functional.

Conclusion

The aim of the thesis - developing software for the configuration of the SpacePix sensor - as well as the esc Aerospace company and its 2SD instrument are introduced at the beginning of this thesis. The software was designed according to the system and software requirements. Several requirements were added during the development, mainly to make the GUI more intuitive. While working on the project, continual cooperation with the 2SD instrument's developers was maintained.

The software allows the user to set and read local and global configurations. Its design makes it more straightforward for users to orient in the parameters. It is also easier to change individual parameters. The user can start data acquisition to check the impact of the current configuration on the measurements. It aims to save time and allow configuring of the sensors also for non-expert users. The software is versatile, so it can be used for future projects with minimum changes. After several minor changes, the software can be used for the 2SD instrument on the VZLUSAT-2 satellite, which is currently in orbit.

To create ConfPix, several supporting codes and documents had to be written. This includes the ASN and ACN files from which the ASN module is generated. Other supporting codes are the scripts which generate the command interface, commands UART, responses UART and custom types modules. These scripts parse the YAML file, which defines all telecommands and telemetries which can be exchanged with the instrument. This thesis also serves as documentation and a software design document.

Although the software meets all the requirements, several possible new features were discovered during the development. One of them is adding a second tab to configure the SXM sensor. Since the main window is more compact than expected, having both global and local configurations shown simultaneously could make navigation in the window faster. Reading local configuration is currently slower than the sending. This is because the instrument has no function implemented to read a series of pixels simultaneously, but it has such a function for sending configuration. This can be solved using its `mem_dump` function, which returns a certain number of bytes from a given address. After the software is tested in practice, several other extensions can be found.

Appendices

Appendix A

Bibliography

[1] A. Gantea, *Isispace selected by esa to provide iod/iov service using cubesats, in the frame of eu horizon 2020 programme*, May 2020. [Online]. Available: https://www.isispace.nl/news/isispace-selected-by-esa-to-provide-iod-iov-service-using-cubesats-in-the-frame-of-european-union-horizon-2020-programme/ (visited on Jul. 15, 2022).

[2] C. Grupen and I. Buvat, *Handbook of particle detection and imaging*. Springer Science & Business Media, 2012, ISBN: 978-3-642-13270-4.

[3] J. Ejemalm, *Radiation environment 1, 2, 3*, lecture, 2021.

[4] J. Perez, *Why space radiation matters*, Oct. 2019. [Online]. Available: https://www.nasa.gov/analogs/nsrl/why-space-radiation-matters (visited on Jul. 18, 2022).

[5] *Types of orbits*, Mar. 2020. [Online]. Available: https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits (visited on Jul. 15, 2022).

[6] *Orbital and technical parameters*. [Online]. Available: https://www.gsc-europa.eu/system-service-status/orbital-and-technical-parameters (visited on Jul. 15, 2022).

[7] *Basics of space flight - solar system exploration: Nasa science*. [Online]. Available: https://solarsystem.nasa.gov/basics/chapter5-1/ (visited on Jul. 15, 2022).

[8] *Glossary - k*. [Online]. Available: https://www.grc.nasa.gov/www/k-12/TRC/laefs/laefs_k.html#keplerian_elements (visited on Jul. 15, 2022).

[9] P. Brož, *On satellites' communication*, 2022.

[10] J. S. Sobolewski, "Cyclic redundancy check", in *Encyclopedia of Computer Science*, 2003, pp. 476–479.

[11] ITU, *Introduction to asn.1*, 2022. [Online]. Available: https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx (visited on Jul. 16, 2022).

[12] *Technical topic: Asn.1 - an introduction to acn*, Mar. 2022. [Online]. Available: `https://taste.tuxfamily.org/wiki/index.php?title=Technical_topic%5C%3A_ASN.1_-_An_introduction_to_ACN` (visited on Jul. 16, 2022).

[13] *Taste*, Jun. 2022. [Online]. Available: `https://taste.tuxfamily.org/wiki/index.php?title=Main_Page` (visited on Jul. 16, 2022).

[14] E. S. Agency, *Asn1scc - asn.1 space certifiable compiler*, May 2018. [Online]. Available: `https://essr.esa.int/project/asn1scc-asn-1-space-certifiable-compiler` (visited on Jul. 16, 2022).

[15] P. S. Foundation, *General python faq*, 2022. [Online]. Available: `https://docs.python.org/3/faq/general.html#id4` (visited on Jul. 16, 2022).

[16] D. Amos, "Python gui programming with tkinter", *Tersedia: https://realpython.com/python-gui-tkinter*, 2020.

[17] *Pyqt5*, Jun. 2022. [Online]. Available: `https://pypi.org/project/PyQt5/` (visited on Jul. 16, 2022).

[18] P. Brož. 2021.

[19] M. Havránek. 2021, pp. 1–31.

[20] A. D. Moore, *Python GUI Programming with Tkinter: Develop responsive and powerful GUI applications with Tkinter*. Packt Publishing Ltd, 2018.

Appendix B

Acronyms

2SD Space Dosimetry System Demonstrator

ADC Analog to Digital Converter

ASIC Application-Specific Integrated Circuit

ASN.1 Abstract Syntax Notation One

CMOS Complementary Metal-Oxide-Semiconductor

CRC Cyclic Redundancy Check

CSA Charge Sensitive Amplifier

CTU Czech Technical University

DAQ data acquisition

ESA European Space Agency

FOV Field Of View

FNSPE Faculty of Nuclear Sciences and Physical Engineering

FPGA Field Programmable Gate Array

GEO Geostationary Orbit/ Geosynchronous Equatorial Orbit

GCR Galactic Cosmic Rays

GTO Geostationary Transfer Orbit

GUI Graphical User Interface

IOV In Orbit Validation

ISISpace Innovative Solutions In Space

ISS International Space Station

LEO Low Earth Orbit

MCP MicroChannel Plate

MEO Medium Earth Orbit

MVC Model-View-Controller

OBC On-Board Computer

PDH Peak Detector Hold

RAAN Right Ascension of the Ascending Node

RSR Row Shift Register

S/C spacecraft

SIS Satellite Interface Simulator

SoI Silicon on Insulator

SPE Solar Particle Events

SpacePix2 SpacePix-2-Lin-S

SSO Sun-Synchronous Orbit

SXM Soft X-ray Monitor

SXRM SpacePix Radiation Monitor

TCL Tool Command Language

TOF Time-Of-Flight

UART Universal Asynchronous Receiver-Transmitter

Appendix C

Software requirements specifications

Table 5: Software requirements specifications.
(VM = Verification Method; T = test, R = review, I = inspection, A = analysis)

ID	Requirement description	Category	VM
SRS-001	SW shall allow user to configure SpacePix Radiation Monitor (SXRM) detector in 2SD instrument.	-	R
SRS-002	SW shall implement configuration for SpacePix-2-Lin-S ASIC.	-	R
SRS-003	SW shall be implemented in Python. Note: Compatibility with Python 3.8 and newer shall be ensured.	-	R
SRS-004	SW shall implement a GUI using Tkinter for Python.	-	R
SRS-005	SW shall communicate with 2SD via UART.	Communication	T
SRS-006	SW shall allow to set port for UART communication. Note: Only when the communication is not open.	Communication	T
SRS-007	SW shall allow to set speed of UART communication. Note: Only when the communication is not open.	Communication	T
SRS-008	SW shall inform user about the UART connection status.	Communication	T
SRS-009	SW shall check if communication with the device was established before any attempt to exchange data with the device.	Communication	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-010	GUI shall contain an UI widget which allows user to enter the name of the port for UART communication. Note: Widget shall be described either by text written on itself or by a corresponding label widget.	Communication	T
SRS-078	SW shall check that the selected port is valid (in the COM# form) and is available.	Communication	T
SRS-079	The UART selection widget shall, when initialized, list all available serial ports.	Communication	T
SRS-080	The port selection witdget shall lists available ports in descending order.	Communication	T
SRS-081	GUI shall contain a refresh button widget which, after being clicked on, emits an event that checks all available ports and shows them to the user (in the corresponding widget). Note: Widget shall be described by text written on itself.	Communication	T
SRS-082	The refresh button shall be enabled after start of application.	Communication	T
SRS-083	The refresh button shall be disabled when a connection with the device is established.	Communication	T
SRS-084	The refresh button shall be enabled when a connection with the device is closed.	Communication	T
SRS-085	The refresh button shall be disabled when DAQ is started.	Communication	T
SRS-086	The refresh button shall be enabled when DAQ is stopped.	Communication	T
SRS-011	GUI shall contain an UI widget which allows user to enter the baud rate for UART communication. Note: Widget shall be described either by text written on itself or by a corresponding label widget.	Communication	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-087	UART communication speed shall be set to 500000 baud by default.	Communication	T
SRS-088	SW shall check that the baud rate is valid and in range (9600-5000000).	Communication	T
SRS-012	GUI shall contain a button widget to connect and disconnect the device to/from UART. Note: Widget shall be described by text written on itself. The button can be in the state 'CONNECT' or 'DISCONNECT'.	Communication	T
SRS-089	If in the 'CONNECT' state, the communication button, after being clicked on, shall emit an event that attempts to start communication with the device. Note: If successful, it shall change the button's state to 'DISCONNECT'.	Communication	T
SRS-090	If in the 'DISCONNECT' state, the communication button, after being clicked on, shall emit an event that attempts to stop communication with the device. Note: If successful, it shall change the button's state to 'CONNECT'.	Communication	T
SRS-091	The result of each connection/disconnection attempt shall be logged.	Communication	T
SRS-092	The connection button shall be disabled when DAQ is started.	Communication	T
SRS-093	The connection button shall be enabled when DAQ is stopped.	Communication	T
SRS-094	The connection button shall be in the state 'CONNECT' by default.	Communication	T
SRS-013	GUI shall contain a button widget which, after being clicked on, emits an event that attempts to ping the device. Note: Widget shall be described by text written on itself.	Communication	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-095	The ping button shall be disabled by default.	Communication	T
SRS-096	The ping button shall be enabled when communication with the device is successfully established.	Communication	T
SRS-097	The ping button shall be disabled when communication with the device is closed.	Communication	T
SRS-098	The result of ping attempt shall be logged.	Communication	T
SRS-099	The ping button shall be disabled when DAQ is started.	Communication	T
SRS-100	The ping button shall be enabled when DAQ is stopped.	Communication	T
SRS-014	SW shall allow to set global and local configuration.	Configuration	T
SRS-015	SW shall allow to read and display any selected configuration.	Configuration	T
SRS-016	SW shall allow to write/modify selected configuration if the configuration is not read only.	Configuration	T
SRS-017	SW shall allow user to choose between global and local (pixel) configuration.	Configuration	T
SRS-023	GUI shall contain an UI widget which allows user to choose between global and local configuration. Note: Only one option can be selected at a time. Widget shall be described either by text written on itself or by a corresponding label widget.	Configuration	T
SRS-024	Global configuration shall be selected by default.	Configuration	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-025	GUI shall change layout if global configuration is selected - GUI shall contain an UI widget/s which allows user to enter each of the global configuration parameters. Note: Each widget shall be described (name of the parameter) either by text written on itself or by a corresponding label widget.	Configuration	T
SRS-026	GUI shall change layout if local configuration is selected - GUI shall contain an UI widget showing a matrix representing the sensor pixel array and an UI widget/s to enter configuration values. Note: Each widget shall be described either by text written on itself or by a corresponding label widget.	Configuration	T
SRS-027	GUI shall contain a button widget which, after being clicked on, emits an event that sends configuration (either global or local) for all sensors in the selected group to the device. Note: Only if the selected group is not read-only.	Configuration	T
SRS-105	The send button shall be disabled by default.	Configuration	T
SRS-106	The send button shall be enabled when communication with the device is successfully established. Note: Only if the selected group is not read-only.	Configuration	T
SRS-107	The send button shall be disabled when communication with the device is closed.	Configuration	T
SRS-108	The result of sending configuration shall be logged for each sensor in the group.	Configuration	T
SRS-109	The send button shall be disabled when DAQ is started.	Configuration	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-110	The send button shall be enabled when DAQ is stopped.	Configuration	T
SRS-028	GUI shall contain a button widget which, after being clicked on, emits an event that gets (reads) the current configuration (either global or local) for all sensors in the selected group from the device.	Configuration	T
SRS-111	The read button shall be disabled by default.	Configuration	T
SRS-112	The read button shall be enabled when communication with the device is successfully established.	Configuration	T
SRS-113	The read button shall be disabled when communication with the device is closed.	Configuration	T
SRS-114	The result of reading configuration shall be logged for each sensor in the group.	Configuration	T
SRS-115	The read button shall be disabled when DAQ is started.	Configuration	T
SRS-116	The read button shall be enabled when DAQ is stopped.	Configuration	T
SRS-029	GUI shall display current configurtion (read or set by user) in corresponding widget/s.	Configuration	T
SRS-030	SW shall check that all parameters are set and in range. Note: Range for each parameter shall be provided in code, accessible by an advanced user who can change it there.	Configuration	T
SRS-076	GUI shall contain a button widget which, after being clicked on, emits an event that sets configuration (either global or local) to the default values for the selected configuration group and sensor. Note: Default values shall be set in code and accessible by an advanced user who can change it there. Configuration group #0 shall not support this feature.	Configuration	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-117	The default configuration button shall be enabled by default.	Configuration	T
SRS-118	The default configuration button shall be disabled when a read-only configuration group is selected.	Configuration	T
SRS-119	The default configuration button shall be enabled when a rewritable configuration group is selected.	Configuration	T
SRS-120	The default configuration button shall be disabled when DAQ is started.	Configuration	T
SRS-121	The default configuration button shall be enabled when DAQ is stopped.	Configuration	T
SRS-077	GUI shall contain a button widget which, after being clicked on, emits an event that sets configuration (either global or local) to the default values for all sensors in the selected configuration group. Note: Default configuration shall be set in code and accessible by an advanced user who can change it there. Configuration group #0 shall not support this feature.	Configuration	T
SRS-122	The group default configuration button shall be enabled by default.	Configuration	T
SRS-123	The group default configuration button shall be disabled when a read-only group is selected.	Configuration	T
SRS-124	The group default configuration button shall be enabled when a rewritable group is selected.	Configuration	T
SRS-125	The group default configuration button shall be disabled when DAQ is started.	Configuration	T
SRS-126	The group default configuration button shall be enabled when DAQ is stopped.	Configuration	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-018	SW shall allow user to set all parameters for global configuration of SpacePix ASIC. Note: Only if global configuration option is chosen. The selected group shall not be read-only.	Global configuration	T
SRS-019	SW shall allow user to read all parameters for global configuration of SpacePix ASIC. Note: Only if global configuration option is chosen.	Global configuration	T
SRS-020	SW shall allow the user to select one or more pixels to be configured from a matrix representing the sensor pixel array. Note: Only if the local configuration option is chosen.	Local configuration	T
SRS-101	SW shall allow user to select all pixels by pressing Ctr+A combination.	Local configuration	T
SRS-102	The pixel selection widget shall permanently display coordinates of corner pixels. Note: The coordinates shall help the operator to select the correct pixels.	Local configuration	T
SRS-103	SW shall show selected range of pixel in text form.	Local configuration	T
SRS-104	SW shall show configuration of pixel above which is currently cursor. Note: If cursor is not above pixel grid, information will not be updated or 'n/a' may be shown.	Local configuration	T
SRS-021	SW shall allow to set configuration for selected pixel or pixels. Note: Only if local configuration option is chosen. Pixel or pixels shall be selected first. Only when the selected group is not read-only.	Local configuration	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-022	SW shall allow to read configuration of all pixels. Note: Only if local configuration option is chosen.	Local configuration	T
SRS-032	SW shall allow to configure up to 5 SpacePix radiation sensors.	Sensors	T
SRS-033	Sensors shall be internally indexed by positive integer from 0 to 4 (inclusive).	Sensors	I
SRS-034	Sensors shall be indexed for user by positive integer from 1 to 5 (inclusive).	Sensors	T
SRS-035	SW shall allow user to select the sensor (1-5) to be configured.	Sensors	T
SRS-036	GUI shall contain an UI widget which allows user to select ID of sensor to be configured. Note: Widget shall be described either by text written on itself or by a corresponding label widget.	Sensors	T
SRS-037	Sensor number 1 shall be selected by default.	Sensors	T
SRS-038	Configurations (global + local) for each of 5 SpacePix sensors shall be grouped into Configuration group.	Configuration group	T
SRS-039	SW shall support up to 6 configuration groups. Note: 6 config. groups * 5 sensors = 30 unique configurations.	Configuration group	T
SRS-040	Configuration group shall be indexed by positive integer from 0 to 5 (inclusive).	Configuration group	T
SRS-041	Configuration group number 0 shall be read only. Note: No configuration can be written to config. group #0 but it can be read.	Configuration group	T
SRS-042	SW shall allow user to select configuration group number (0-5).	Configuration group	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-043	GUI shall contain an UI widget which allows user to select configuration group number. Note: Widget shall be described either by text written on itself or by a corresponding label widget.	Configuration group	T
SRS-044	Configuration group number 1 shall be selected by default.	Configuration group	T
SRS-045	SW shall allow to set shutter duration.	Data acquisition	T
SRS-046	100 ms shutter duration shall be set by default.	Data acquisition	T
SRS-047	SW shall allow to set sampling period.	Data acquisition	T
SRS-048	2 s sampling period shall be set by default.	Data acquisition	T
SRS-049	Measuring mode FRAMES shall be set. Note: User shall not be allowed to change this mode.	Data acquisition	I
SRS-050	Test mode shall be set internally to send the measured frames via UART.	Data acquisition	I
SRS-051	After setting test mode, the device shall be reset.	Data acquisition	I
SRS-052	GUI shall contain an UI widget which allows user to enter shutter duration. Note: Widget shall be described either by text written on itself or by a corresponding label widget.	Data acquisition	T
SRS-053	GUI shall contain an UI widget which allows user to enter sampling period. Note: Widget shall be described either by text written on itself or by a corresponding label widget.	Data acquisition	T
SRS-055	SW shall check that both shutter duration and sampling period are set and in range. Note: Shutter duration range: 1-65536, sampling period range: 1000-65536.	Data acquisition	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-056	SW shall allow to start and stop data acquisition.	Data acquisition	T
SRS-057	SW shall display acquired data.	Data acquisition	T
SRS-058	When starting data acquisition SW shall simultaneously open a new window showing data acquired by SpacePix ASIC.	Data acquisition	T
SRS-059	When stopping data acquisition SW shall simultaneously close the window showing acquired data.	Data acquisition	T
SRS-060	GUI shall contain a button widget to start/stop data acquisition. Note: The button can be in the state 'START' or 'STOP'.	Data acquisition	T
SRS-127	The data acquisition button shall be disabled by default.	Data acquisition	T
SRS-128	The data acquisition button shall be enabled when communication with the device is successfully established.	Data acquisition	T
SRS-129	The data acquisition button shall be disabled when communication with the device is closed.	Data acquisition	T
SRS-062	The action of the data acquisition button shall start data acquisition when the button is in the 'START' state. Then switch the state.	Data acquisition	T
SRS-130	The data acquisition button shall be in the state 'START' by default.	Data acquisition	T
SRS-063	The data acquisition button shall stop data acquisition when the button is in the 'STOP' state. Then switch the state.	Data acquisition	T
SRS-131	After stopping data acquisition SW shall close the window showing acquired data.	Data acquisition	T
SRS-132	After stopping data acquisition test mode shall be changed back to standard mode.	Data acquisition	I
SRS-133	After changing mode to standard, device shall be reset.	Data acquisition	I

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-065	SW shall allow the user to generate a text file containing commands to load the selected configuration into the instrument. Note: Format of the commands shall follow command-line interface format of control software.	File generating	T
SRS-066	GUI shall contain a button widget which, after being clicked on, emits an event that creates a text file containing commands needed to achieve selected configuration. Note: Widget shall be described by text written on itself.	File generating	T
SRS-134	The generate commands button shall be enabled by default.	File generating	T
SRS-135	Configuration commands shall be generated for all pixels in the selected group. Note: Not only the modified/configured pixels.	File generating	T
SRS-136	The generate commands button shall be disabled when DAQ is started.	File generating	T
SRS-137	The generate commands button shall be enabled when DAQ is stopped.	File generating	T
SRS-067	Command text file shall contain both configurations (local and global).	File generating	T
SRS-138	The sequence of commands in the command text file shall be as follows: 1. global configuration for all sensors in ascending order 2. local configuration for all sensors in ascending order	File generating	T
SRS-068	GUI shall contain a UI widget to show logged messages to the user. Note: Widget shall be described either by text written on itself or by a corresponding label widget.	Logging	T
SRS-069	All logged messages shall be saved in a text file.	Logging	T

Continued on next page

Table 5 – continued from previous page

ID	Requirement description	Category	VM
SRS-070	SW shall hold a shadow copy of each configuration pair. Note: For each group and sensor pair.	Shadow copy	T
SRS-071	Value modifications in GUI widgets shall be updated in shadow copy.	Shadow copy	T
SRS-072	A configuration (local/global) read from the device shall be updated in shadow copy.	Shadow copy	T
SRS-073	SW shall highlight configuration widgets with modified values which are not saved. Note: Saved values are values which were sent to the device. Right after the SW is run, values set by default are consider saved values.	Shadow copy	T
SRS-074	SW shall highlight a particular sensor ID in the sensor ID selection widget if there is any unsaved configuration value for the corresponding sensor. Note: Local and global configurations are evaluated separately for this purpose.	Shadow copy	T
SRS-075	SW shall highlight a particular configuration group number in the configuration group selection widget if there is any unsaved configuration in any sensor of the configuration group. Note: Local and global configurations are evaluated separately for this purpose.	Shadow copy	T

Appendix D

Requirements mapped to test cases

Table 6: Requirements to test cases mapping matrix.

Requirement ID	Test cases IDs
SRS-001	-
SRS-002	-
SRS-003	-
SRS-004	-
SRS-005	TC-CP-002
SRS-006	TC-CP-002
SRS-007	TC-CP-002
SRS-008	TC-CP-002
SRS-009	TC-CP-003
SRS-010	TC-CP-001
SRS-011	TC-CP-001
SRS-012	TC-CP-001
SRS-013	TC-CP-001 TC-CP-003
SRS-014	TC-CP-006 TC-CP-007
SRS-015	TC-CP-006 TC-CP-007
SRS-016	TC-CP-005 TC-CP-006 TC-CP-007
SRS-017	TC-CP-004
SRS-018	TC-CP-005 TC-CP-006
SRS-019	TC-CP-006
SRS-020	TC-CP-004
SRS-021	TC-CP-005 TC-CP-007
SRS-022	TC-CP-007

Continued on next page

Table 6 – continued from previous page

Requirement ID	**Test cases IDs**
SRS-023	TC-CP-004
SRS-024	TC-CP-004
SRS-025	TC-CP-004
SRS-026	TC-CP-004
SRS-027	TC-CP-004 TC-CP-006 TC-CP-007
SRS-028	TC-CP-004 TC-CP-006 TC-CP-007
SRS-029	TC-CP-006 TC-CP-007
SRS-030	TC-CP-006 TC-CP-007
SRS-032	TC-CP-005 TC-CP-006 TC-CP-007
SRS-033	-
SRS-034	TC-CP-005
SRS-035	TC-CP-005
SRS-036	TC-CP-005
SRS-037	TC-CP-005
SRS-038	TC-CP-005 TC-CP-006 TC-CP-007
SRS-039	TC-CP-005 TC-CP-006 TC-CP-007
SRS-040	TC-CP-005
SRS-041	TC-CP-005 TC-CP-006 TC-CP-007
SRS-042	TC-CP-005
SRS-043	TC-CP-005
SRS-044	TC-CP-005
SRS-045	TC-CP-008
SRS-046	TC-CP-008
SRS-047	TC-CP-008
SRS-048	TC-CP-008
SRS-049	-
SRS-050	-
SRS-051	-
SRS-052	TC-CP-008

Continued on next page

Table 6 – continued from previous page

Requirement ID	Test cases IDs
SRS-053	TC-CP-008
SRS-055	TC-CP-008
SRS-056	TC-CP-009
SRS-057	TC-CP-009
SRS-058	TC-CP-009
SRS-059	TC-CP-009
SRS-060	TC-CP-008 TC-CP-009
SRS-062	TC-CP-009
SRS-063	TC-CP-009
SRS-065	TC-CP-010
SRS-066	TC-CP-010
SRS-067	TC-CP-010
SRS-068	TC-CP-011
SRS-069	TC-CP-011
SRS-070	TC-CP-006 TC-CP-007
SRS-071	TC-CP-006 TC-CP-007
SRS-072	TC-CP-006 TC-CP-007
SRS-073	TC-CP-006 TC-CP-007
SRS-074	TC-CP-006 TC-CP-007
SRS-075	TC-CP-006 TC-CP-007
SRS-076	TC-CP-004 TC-CP-006 TC-CP-007
SRS-077	TC-CP-004 TC-CP-006 TC-CP-007
SRS-078	TC-CP-002
SRS-079	TC-CP-001
SRS-080	TC-CP-001 TC-CP-002
SRS-081	TC-CP-001 TC-CP-002
SRS-082	TC-CP-001
SRS-083	TC-CP-002
SRS-084	TC-CP-002

Continued on next page

Table 6 – continued from previous page

Requirement ID	Test cases IDs
SRS-085	TC-CP-009
SRS-086	TC-CP-009
SRS-087	TC-CP-001
SRS-088	TC-CP-002
SRS-089	TC-CP-002
SRS-090	TC-CP-002
SRS-091	TC-CP-002
SRS-092	TC-CP-009
SRS-093	TC-CP-009
SRS-094	TC-CP-001
SRS-095	TC-CP-001
SRS-096	TC-CP-003
SRS-097	TC-CP-003
SRS-098	TC-CP-003
SRS-099	TC-CP-009
SRS-100	TC-CP-009
SRS-101	TC-CP-004
SRS-102	TC-CP-004
SRS-103	TC-CP-004
SRS-104	TC-CP-004
SRS-105	TC-CP-004
SRS-106	TC-CP-004
SRS-107	TC-CP-004
SRS-108	TC-CP-006 TC-CP-007
SRS-109	TC-CP-009
SRS-110	TC-CP-009
SRS-111	TC-CP-004
SRS-112	TC-CP-004
SRS-113	TC-CP-004
SRS-114	TC-CP-006 TC-CP-007
SRS-115	TC-CP-009
SRS-116	TC-CP-009
SRS-117	TC-CP-004
SRS-118	TC-CP-005
SRS-119	TC-CP-005
SRS-120	TC-CP-009
SRS-121	TC-CP-009
SRS-122	TC-CP-004
SRS-123	TC-CP-005
SRS-124	TC-CP-005

Continued on next page

Table 6 – continued from previous page

Requirement ID	Test cases IDs
SRS-125	TC-CP-009
SRS-126	TC-CP-009
SRS-127	TC-CP-008
SRS-128	TC-CP-008
SRS-129	TC-CP-008
SRS-130	TC-CP-008
SRS-131	TC-CP-009
SRS-132	-
SRS-133	-
SRS-134	TC-CP-010
SRS-135	TC-CP-010
SRS-136	TC-CP-009
SRS-137	TC-CP-009
SRS-138	TC-CP-010

Appendix E

Test cases mapped to requirements

Table 7: Test cases to requirements mapping matrix.

Test case ID	Requirements IDs
TC-CP-001	SRS-010 SRS-011 SRS-012 SRS-013 SRS-079 SRS-080 SRS-081 SRS-082 SRS-087 SRS-094 SRS-095
TC-CP-002	SRS-005 SRS-006 SRS-007 SRS-008 SRS-078 SRS-080 SRS-081 SRS-083 SRS-084 SRS-088 SRS-089 SRS-090 SRS-091
TC-CP-003	SRS-009 SRS-013 SRS-096 SRS-097 SRS-098

Continued on next page

Table 7 – continued from previous page

Test case ID	Requirements IDs
TC-CP-004	SRS-017 SRS-020 SRS-023 SRS-024 SRS-025 SRS-026 SRS-027 SRS-028 SRS-076 SRS-077 SRS-101 SRS-102 SRS-103 SRS-104 SRS-105 SRS-106 SRS-107 SRS-111 SRS-112 SRS-113 SRS-117 SRS-122
TC-CP-005	SRS-016 SRS-018 SRS-021 SRS-032 SRS-034 SRS-035 SRS-036 SRS-037 SRS-038 SRS-039 SRS-040 SRS-041 SRS-042 SRS-043 SRS-044 SRS-118 SRS-119 SRS-123 SRS-124

Continued on next page

Table 7 – continued from previous page

Test case ID	Requirements IDs
TC-CP-006	SRS-014 SRS-015 SRS-016 SRS-018 SRS-019 SRS-027 SRS-028 SRS-029 SRS-030 SRS-032 SRS-038 SRS-039 SRS-041 SRS-070 SRS-071 SRS-072 SRS-073 SRS-074 SRS-075 SRS-076 SRS-077 SRS-108 SRS-114

Continued on next page

Table 7 – continued from previous page

Test case ID	Requirements IDs
TC-CP-007	SRS-014 SRS-015 SRS-016 SRS-021 SRS-022 SRS-027 SRS-028 SRS-029 SRS-030 SRS-032 SRS-038 SRS-039 SRS-041 SRS-070 SRS-071 SRS-072 SRS-073 SRS-074 SRS-075 SRS-076 SRS-077 SRS-108 SRS-114
TC-CP-008	SRS-045 SRS-046 SRS-047 SRS-048 SRS-052 SRS-053 SRS-055 SRS-060 SRS-127 SRS-128 SRS-129 SRS-130

Continued on next page

Table 7 – continued from previous page

Test case ID	Requirements IDs
TC-CP-009	SRS-056 SRS-057 SRS-058 SRS-059 SRS-060 SRS-062 SRS-063 SRS-085 SRS-086 SRS-092 SRS-093 SRS-099 SRS-100 SRS-109 SRS-110 SRS-115 SRS-116 SRS-120 SRS-121 SRS-125 SRS-126 SRS-131 SRS-136 SRS-137
TC-CP-010	SRS-065 SRS-066 SRS-067 SRS-134 SRS-135 SRS-138
TC-CP-011	SRS-068 SRS-069

Appendix F

Test cases

1 TC-CP-001 - Connection GUI

1.1 Requirements covered

SRS-010
SRS-011
SRS-012
SRS-013
SRS-079
SRS-080
SRS-081
SRS-082
SRS-087
SRS-094
SRS-095

1.2 Purpose

This test verifies that the connection part of GUI and its initial state are according to the requirements.

1.3 Description

This test checks that the GUI contains all widgets required for connection to the instrument. It also checks that initial values are as expected in two possible cases - the instrument is connected, and the instrument is not connected.

1.4 Resources and Tools

- 2SD device
- data cable
- power cable

1.5 Prerequisites

No serial devices shall be connected to the testing computer.

1.6 Scenario

1. Start the application.
2. Verify that there is a widget to enter a port.
3. Verify that the port widget is empty.

4. Verify that there is a widget to enter a baud rate.
5. Verify that the baud rate widget is set to 500000.
6. Verify that there is a connection button.
7. Verify that the connection button is enabled.
8. Verify that the connection button is in a CONNECT state.
9. Verify that there is a button to ping the instrument.
10. Verify that the ping button is disabled.
11. Verify that there is a button to refresh ports in the port widget.
12. Close the application window.
13. Connect the instrument to the testing computer.
14. Start the application.
15. Verify that the port widget contains a string in the COM# form.

2 TC-CP-002 - Connection and disconnection

2.1 Requirements covered

SRS-005
SRS-006
SRS-007
SRS-008
SRS-078
SRS-080
SRS-081
SRS-083
SRS-084
SRS-088
SRS-089
SRS-090
SRS-091

2.2 Purpose

This test shall verify starting and closing communication with the instrument.

2.3 Description

The test checks that the SW reacts as expected when setting an invalid port or baud rate (or both). It verifies that when connected or disconnected, the GUI reacts as expected. It also verifies its behaviour when 0 to 2 devices are connected to the testing computer.

2.4 Resources and Tools

- 2SD device
- data cable
- power cable
- USB to serial converter

2.5 Prerequisites

No serial devices shall be connected to the testing computer.

2.6 Scenario

1. Start the application.

2. Press the connection button.
3. Verify that no device was connected.
4. Verify that the information was logged in a logging widget.

5. Insert a string in a COM# form into the port widget.
6. Press the connection button.
7. Verify that no device was connected.
8. Verify that the information was logged in a logging widget.

9. Connect the instrument to the testing computer.
10. Press the refresh button.
11. Verify that the port widget lists a port in the COM# form.

12. Insert a string into the port widget ('port'). (invalid port)
13. Try to set the baud rate to 500001. (baud rate out of range)
14. Verify that it is not possible.
15. Set the baud rate to 5000. (baud rate out of range)
16. Press the connection button.
17. Verify that no device was connected.
18. Verify that the information was logged in a logging widget.

19. Insert a string into the port widget ('port'). (invalid port)
20. Set the baud rate to 10000. (valid baud rate)
21. Press the connection button.
22. Verify that no device was connected.
23. Verify that the information was logged in a logging widget.

24. Set the port to the one to which the instrument is connected. (valid port)
25. Set the baud rate to 5000. (baud rate out of range)
26. Press the connection button.

27. Verify that no device was connected.
28. Verify that the information was logged in a logging widget.

29. Set the port to the one to which the instrument is connected. (valid port)
30. Set the baud rate to 10000. (valid baud rate)
31. Press the connection button.
32. Verify that the instrument was connected.
33. Verify that the information was logged in a logging widget.
34. Verify that the connection button is in a DISCONNECT state.
35. Verify that the refresh button was disabled.
36. Verify that the port cannot be changed.
37. Verify that the baud rate cannot be changed.

38. Press the connection button.
39. Verify that the instrument was disconnected.
40. Verify that the information was logged in a logging widget.
41. Verify that the connection button is in a CONNECT state.
42. Verify that the refresh button was enabled.
43. Verify that the port can be changed.
44. Verify that the baud rate can be changed.

45. Connect the USB to serial converter.
46. Press the refresh button.
47. Verify that two ports are listed in the port widget.
48. Verify that the ports are ordered in descending order.

3 TC-CP-003 - Ping

3.1 Requirements covered

SRS-009
SRS-013
SRS-096
SRS-097
SRS-098

3.2 Purpose

This test shall verify that the instrument can be pinged and is answering.

3.3 Description

This test checks that the ping button reacts as expected. The test also introduces an unexpected behaviour - disconnecting the instrument while communication is running - and verifies that the SW reacts as expected.

3.4 Resources and Tools

- 2SD device
- data cable
- power cable

3.5 Prerequisites

No serial devices shall be connected to the testing computer.

3.6 Scenario

1. Start the application.
2. Select the port to which the instrument is connected.
3. Press the connection button.
4. Verify that the ping button was enabled.
5. Press the ping button.
6. Verify that the instrument was pinged.
7. Verify that the information was logged in a logging widget.
8. Disconnect the instrument from the testing computer.

9. Press the ping button.
10. Verify that no device answered.
11. Verify that the information was logged in a logging widget.
12. Press the button to disconnect from the instrument.
13. Verify that the ping button was disabled.

4 TC-CP-004 - Configuration GUI

4.1 Requirements covered

SRS-017
SRS-020
SRS-023
SRS-024
SRS-025
SRS-026
SRS-027
SRS-028
SRS-076
SRS-077
SRS-101
SRS-102
SRS-103
SRS-104
SRS-105
SRS-106
SRS-107
SRS-111
SRS-112
SRS-113
SRS-117
SRS-122

4.2 Purpose

This test shall verify that the configuration part of the GUI works as expected.

4.3 Description

This test checks that buttons react to connection and disconnection to/from the instrument. It also checks that switching between local and global configurations works.

4.4 Resources and Tools

- 2SD device
- data cable
- power cable

4.5 Prerequisites

The instrument is connected to the testing computer.

4.6 Scenario

1. Start the application.
2. Verify that there is a send button and it is disabled.
3. Verify that there is a read button and it is disabled.
4. Verify that there is a default configuration button.
5. Verify that the default configuration button is enabled.
6. Verify that there is a group default configuration button.
7. Verify that the group default configuration button is enabled.
8. Verify that there is a widget to select between global and local configurations.
9. Verify that global configuration is selected.
10. Verify that there are widgets to enter individual global configuration parameters.
11. Select local configuration.
12. Verify that GUI changed the layout to a pixel grid and widgets to enter individual local configuration parameters.
13. Verify that the coordinates of the pixel grid's corners are described.
14. Move the cursor over the grid.
15. Verify that there is a message stating the coordinates of the current pixel and its configuration.
16. Click inside the grid.
17. Verify that a pixel was selected.
18. Verify that there is a message stating the coordinates of the selected pixel.
19. Click inside the grid, hold and move.
20. Verify that a range of pixels was selected.
21. Verify that this range is stated in the message about selected pixel/s.
22. Click inside the grid.

23. Press Ctrl+A.
24. Verify that all pixels were selected.
25. Select global configuration.
26. Verify that GUI changed the layout to widgets to enter individual global configuration parameters.

27. Select the port to which the instrument is connected.
28. Press the connection button.
29. Verify that the send button was enabled.
30. Verify that the read button was enabled.
31. Press the button to disconnect from the instrument.
32. Verify that the send button was disabled.
33. Verify that the read button was disabled.

5 TC-CP-005 - Configuration group and sensor ID

5.1 Requirements covered

SRS-016
SRS-018
SRS-021
SRS-032
SRS-034
SRS-035
SRS-036
SRS-037
SRS-038
SRS-039
SRS-040
SRS-041
SRS-042
SRS-043
SRS-044
SRS-118
SRS-119
SRS-123
SRS-124

5.2 Purpose

This test shall verify that when choosing a read-only configuration group, the SW reacts as expected.

5.3 Description

This test checks that there are widgets to select configuration group and sensor id. It then checks that if and only if selecting a read-only group, several features are blocked.

5.4 Resources and Tools

- 2SD device
- data cable
- power cable

5.5 Prerequisites

The instrument is connected to the testing computer.

5.6 Scenario

1. Start the application.
2. Verify that there is a widget which allows the selection of configuration group.
3. Verify that the configuration group is set to 1.
4. Verify that configuration groups 0-5 can be selected.
5. Verify that there is a widget which allows the selection of sensor ID.
6. Verify that the sensor ID is set to 1.
7. Verify that sensor ID 1-5 can be selected.

8. Select the port to which the instrument is connected.
9. Press the connection button.
10. Select configuration group 0.
11. Verify that configuration cannot be changed.
12. Verify that configuration cannot be sent.
13. Verify that the default configuration button is disabled.
14. Verify that the group default configuration button is disabled.
15. Step by step, select a configuration group (starting with 1 and finishing with 5) and every time, verify that:
    - Configuration can be changed.
    - Configuration can be sent.
    - The default configuration button is enabled.
    - The group default configuration button is enabled.

16. Select local configuration.
17. Select configuration group 0.
18. Verify that configuration cannot be changed.
19. Verify that configuration cannot be sent.
20. Verify that the default configuration button is disabled.
21. Verify that the group default configuration button is disabled.

22. Step by step, select a configuration group (starting with 1 and finishing with 5) and every time, verify that:

    - Configuration can be changed.
    - Configuration can be sent.
    - The default configuration button is enabled.
    - The group default configuration button is enabled.

6 TC-CP-006 - Send and read global configuration

6.1 Requirements covered

SRS-014
SRS-015
SRS-016
SRS-018
SRS-019
SRS-027
SRS-028
SRS-029
SRS-030
SRS-032
SRS-038
SRS-039
SRS-041
SRS-070
SRS-071
SRS-072
SRS-073
SRS-074
SRS-075
SRS-076
SRS-077
SRS-108
SRS-114

6.2 Purpose

This test shall verify that sending and reading global configuration works. It shall verify that marking parameters, sensors and groups as changed works.

6.3 Description

This test verifies that global configuration parameters cannot be set to invalid values or values outside of the supported range. It verifies that setting one sensor in a group or the whole group to default configuration works. It checks that sending and reading of the global configuration works. It verifies that the parameter, sensor and group are marked as changed when a parameter is changed.

6.4 Resources and Tools

- 2SD device

- data cable
- power cable

6.5 Prerequisites

The instrument is connected to the testing computer.

6.6 Scenario

1. Start the application.
2. Select the port to which the instrument is connected.
3. Press the connection button.
4. Try to enter a string ('value') to all of the configuration widgets. (invalid value)
5. Verify that it is not possible.
6. Try to enter 1024 to all of the configuration widgets. (value not in range)
7. Verify that it is not possible.
8. Try to enter -1 to all of the configuration widgets. (value not in range)
9. Verify that it is not possible.
10. Delete some of the configuration values, so the widgets are empty.
11. Press the send button.
12. Verify that the configurations were not sent.
13. Verify that the error was logged.

14. For each configuration group in range 1-5:
    - Select configuration group.
    - Verify that no sensor is marked as changed.
    - Select sensor with the same number as the configuration group.
    - Press the default configuration button.
    - Verify that (at least some of) the values in widgets were changed.
    - Verify that the changed values are marked as changed.
    - Verify that the selected sensor is marked as changed.
    - Verify that the selected configuration group is marked as changed.
    - Verify that other sensors in the selected configuration group are not marked as changed.

- Verify that configuration groups with a higher number than the selected group are not marked as changed.

15. Select the configuration group number 5.

16. Select sensor number 1.

17. Set parameters to the following values: vbp_csa to 255, vbn_csa to 254, vfb_csa to 253, vbn_pdh to 252, vbp_hyst to 251, vbp_comp to 250, vbn_tdac to 1, vbp_lcc to 2, vbn_adc to 3, lvds_cm to 4, lvds_strength to 5, sf to 6, tail to 7, test to 8, fsel0 to 1, fsel1 to 0, fsel2 to 1, backside_debug_en to true-1, vref_en to false-0, backside_low_leak_en to true-1, backside_inject_en to true-1, analog_out0_en to true-1, analog_out1_en to true-1, analog_out2_en to true-1, analog_out3_en to true-1, backside_en to true-1, temp_sens_en to true-1, adc_pin_en to true-1, vthr to 1023.

18. Verify that changed values were marked as changed.

19. For each configuration group in range 1-5:

    - Press the send button.
    - Verify that the result of sending the configurations was logged.
    - Verify that no value is marked as changed.
    - Verify that no sensor in the selected configuration group is marked as changed.
    - Verify that the selected group is not marked as changed.

20. For each configuration group in range 1-5:

    - Press the group default configuration button.
    - Verify that all sensors except the one with the same number as the selected configuration group are marked as changed.
    - Select the sensor with the same number as the selected configuration group.
    - Set all configuration values to 0 or false-0 depending on their type.
    - Verify that changed values were marked as changed.
    - Verify that the selected sensor was marked as changed.

21. For each configuration group in range 0-5:

    - Press the read button.
    - Verify that the result of reading the configurations was logged.
    - Verify that the configuration values have changed for each sensor as expected.
    - Verify that no value is marked as changed.

- Verify that no sensor in the selected configuration group is marked as changed.
- Verify that the selected group is not marked as changed.

22. Select the configuration group number 5.

23. Select sensor number 1.

24. Verify that parameters are set to the following values: vbp_csa to 255, vbn_csa to 254, vfb_csa to 253, vbn_pdh to 252, vbp_hyst to 251, vbp_comp to 250, vbn_tdac to 1, vbp_lcc to 2, vbn_adc to 3, lvds_cm to 4, lvds_strength to 5, sf to 6, tail to 7, test to 8, fsel0 to 1, fsel1 to 0, fsel2 to 1, backside_debug_en to true-1, vref_en to false-0, backside_low_leak_en to true-1, backside_inject_en to true-1, analog_out0_en to true-1, analog_out1_en to true-1, analog_out2_en to true-1, analog_out3_en to true-1, backside_en to true-1, temp_sens_en to true-1, adc_pin_en to true-1, vthr to 1023.

7 TC-CP-007 - Send and read local configuration

7.1 Requirements covered

SRS-014
SRS-015
SRS-016
SRS-021
SRS-022
SRS-027
SRS-028
SRS-029
SRS-030
SRS-032
SRS-038
SRS-039
SRS-041
SRS-070
SRS-071
SRS-072
SRS-073
SRS-074
SRS-075
SRS-076
SRS-077
SRS-108
SRS-114

7.2 Purpose

This test shall verify that sending and reading local configuration works. It shall verify that marking parameters, sensors and groups as changed works.

7.3 Description

This test verifies that local configuration parameters cannot be set to invalid values or values outside of the supported range. It verifies that setting one sensor in a group or the whole group to default configuration works. It checks that sending and reading of the local configuration works. It verifies that the parameter, sensor and group are marked as changed when a parameter is changed.

7.4 Resources and Tools

- 2SD device

- data cable
- power cable

7.5 Prerequisites

The instrument is connected to the testing computer.

7.6 Scenario

1. Start the application.
2. Select the port to which the instrument is connected.
3. Press the connection button.
4. Select local configuration.
5. Try to enter a string ('value') to all of the configuration widgets. (invalid value)
6. Verify that it is not possible.
7. Try to enter 16 to all of the configuration widgets. (value not in range)
8. Verify that it is not possible.
9. Try to enter -1 to all of the configuration widgets. (value not in range)
10. Verify that it is not possible.
11. Select a [20,50] pixel.
12. Delete a parameter, so the widget is empty.
13. Press the send button.
14. Verify that the configuration was not sent.
15. Verify that the error was logged.
16. For each configuration group in range 1-5:
    - Select configuration group.
    - Verify that no sensor is marked as changed.
    - Select sensor with the same number as the configuration group.
    - Press the default configuration button.
    - Verify that the configuration values were changed.
    - Verify that the changed values were marked as changed.
    - Verify that the selected sensor was marked as changed.

- Verify that the selected configuration group was marked as changed.
- Verify that other sensors in the selected configuration group are not marked as changed.
- Verify that configuration groups with a higher number than the selected group are not marked as changed.

17. Select configuration group number 5.

18. Select sensor number 1.

19. Set pixels' parameters in the following way: for pixels [0,0]-[0,63] tdac to 1, for pixels [1,0]-[1,63] tdac to 2, for pixels [63,0]-[63,63] inject_en to true-1, for pixels [62,0]-[62,63] hit_global_en to true-1.

20. Verify that changed values were marked as changed.

21. For each configuration group in range 1-5:
    - Press the send button.
    - Verify that the result of sending the configurations was logged.
    - Verify that no value is marked as changed.
    - Verify that no sensor in the selected configuration group is marked as changed.
    - Verify that the selected group is not marked as changed.

22. For each configuration group in range 1-5:
    - Press the group default configuration button.
    - Verify that all sensors except the one with the same number as the selected configuration group are marked as changed.
    - Select the sensor with the same number as the selected configuration group.
    - Select all pixels and set all configuration values to 0 or false-0 depending on their type.
    - Verify that changed values were marked as changed.
    - Verify that the selected sensor was marked as changed.

23. For each configuration group in range 0-5:
    - Press the read button.
    - Verify that the result of reading the configurations was logged.
    - Verify that the configuration values have changed for each sensor as expected.
    - Verify that no value is marked as changed.
    - Verify that no sensor in the selected configuration group is marked as changed.

- Verify that the selected group is not marked as changed.

24. Select configuration group number 5.

25. Select sensor number 1.

26. Verify that pixels' parameters are set in the following way: for pixels [0,0]-[0,63] tdac to 1, for pixels [1,0]-[1,63] tdac to 2, for pixels [63,0]-[63,63] inject_en to true-1, for pixels [62,0]-[62,63] hit_global_en to true-1.

8 TC-CP-008 - Data acquisition GUI

8.1 Requirements covered

SRS-045
SRS-046
SRS-047
SRS-048
SRS-052
SRS-053
SRS-055
SRS-060
SRS-127
SRS-128
SRS-129
SRS-130

8.2 Purpose

This test shall verify that the data acquisition part of GUI and its initial state are according to the requirements.

8.3 Description

This test checks that the GUI contains all widgets required for data acquisition. It also checks that initial values are as expected and that the widgets react to connecting and disconnecting to/from the instrument. It verifies that parameters cannot be set to invalid values or values outside of the supported range.

8.4 Resources and Tools

- 2SD device
- data cable
- power cable

8.5 Prerequisites

The instrument is connected to the testing computer.

8.6 Scenario

1. Start the application.

2. Verify that there is a widget to enter a shutter duration.
3. Verify that the shutter duration widget is set to 100ms.
4. Verify that there is a widget to enter a sampling period.
5. Verify that the sampling period widget is set to 2s.
6. Verify that there is a button to start data acquisition.
7. Verify that the data acquisition button is disabled.
8. Select the port to which the instrument is connected.
9. Press the connection button.
10. Verify that the data acquisition button was enabled.
11. Try to enter a string ('value') to the shutter widget. (invalid value)
12. Try to enter a string ('value') to the sampling period widget. (invalid value)
13. Verify that it is not possible.
14. Try to enter 66000 to the shutter widget. (value not in range)
15. Try to enter 66000 to the sampling period widget. (value not in range)
16. Verify that it is not possible.
17. Try to enter 0 to the shutter widget.(value not in range)
18. Try to enter 1 to the sampling period widget. (value not in range)
19. Verify that it is not possible.
20. Delete shutter value, so the widget is empty.
21. Press the data acquisition button.
22. Verify that the data acquisition was not started
23. Verify that the error was logged.
24. Delete sampling period value, so the widget is empty.
25. Press the data acquisition button.
26. Verify that the data acquisition was not started
27. Verify that the error was logged.
28. Press the button to disconnect from the instrument.
29. Verify that the data acquisition button was disabled.

9 TC-CP-009 - Data acquisition

9.1 Requirements covered

SRS-056
SRS-057
SRS-058
SRS-059
SRS-060
SRS-062
SRS-063
SRS-085
SRS-086
SRS-092
SRS-093
SRS-099
SRS-100
SRS-109
SRS-110
SRS-115
SRS-116
SRS-120
SRS-121
SRS-125
SRS-126
SRS-131
SRS-136
SRS-137

9.2 Purpose

This test shall verify that starting and stopping data acquisition works.

9.3 Description

This test verifies that when starting data acquisition, the buttons in the main window behave as expected. It verifies that a new window opens showing the acquired data. It also checks that when stopping data acquisition, the data acquisition window closes, and the buttons in the main window behave as expected.

9.4 Resources and Tools

- 2SD device
- data cable

- power cable

9.5 Prerequisites

The instrument is connected to the testing computer.

9.6 Scenario

1. Start the application.
2. Select the port to which the instrument is connected.
3. Press the connection button.
4. Press the data acquisition button.
5. Verify that a (data acquisition) window has opened.
6. Verify that the data acquisition window shows acquired data.
7. Verify that the data acquisition button is in a STOP state.
8. Verify that the refresh button was disabled.
9. Verify that the connection button was disabled.
10. Verify that the ping button was disabled.
11. Verify that the send button was disabled.
12. Verify that the read button was disabled.
13. Verify that the default configuration button was disabled.
14. Verify that the group default configuration button was disabled.
15. Verify that the generate commands button was disabled.

16. Press the data acquisition button.
17. Verify that the data acquisition window has closed.
18. Verify that the data acquisition button is in a START state.
19. Verify that the refresh button was enabled.
20. Verify that the connection button was enabled.
21. Verify that the ping button was enabled.
22. Verify that the send button was enabled.
23. Verify that the read button was enabled.

24. Verify that the default configuration button was enabled.
25. Verify that the group default configuration button was enabled.
26. Verify that the generate commands button was enabled.

10 TC-CP-010 - File generating

10.1 Requirements covered

SRS-065
SRS-066
SRS-067
SRS-134
SRS-135
SRS-138

10.2 Purpose

This test shall verify that generating commands for the selected configuration group works.

10.3 Description

This test verifies that commands are generated in the required order for the selected configuration group.

10.4 Resources and Tools

No tools.

10.5 Prerequisites

No prerequisites.

10.6 Scenario

1. Start the application.
2. Verify that there is a generate commands button.
3. Verify that the generate commands button is enabled.
4. Press the generate commands button.
5. Go to the generated_files folder.
6. Verify that there is a new file.
7. Verify that the file contains commands corresponding to the communication with CubeSatCarrier 2.
8. Verify that the file begins with global configuration for all sensors in the configuration group 1.

9. Verify that after the global configuration commands, there are local configuration commands for all sensors and pixels in the configuration group number 1.

10. Select configuration group number 3.
11. Select sensor number 5.
12. Press the generate commands button.
13. Go to the generated_files folder.
14. Verify that there is a new file.
15. Verify that the file contains commands corresponding to the communication with CubeSatCarrier 2.
16. Verify that the file begins with global configuration for all sensors in the configuration group 3.
17. Verify that after the global configuration commands, there are local configuration commands for all sensors and pixels in the configuration group number 3.

11 TC-CP-011 - Logging

11.1 Requirements covered

SRS-068
SRS-069

11.2 Purpose

This test shall verify that logging works.

11.3 Description

This test verifies that there is a logging widget and that a log file is created.

11.4 Resources and Tools

- 2SD device
- data cable
- power cable

11.5 Prerequisites

The instrument is connected to the testing computer.

11.6 Scenario

1. Start the application.
2. Verify that there is a logging widget.
3. Select the port to which the instrument is connected.
4. Press the connection button.
5. Press the connection button.
6. Close the main window.
7. Go to the folder with the software.
8. Verify that there is a confpix.log file.
9. Verify that the file contains logged messages.