Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Tuva Okkenhaug Moxnes

A common software framework for a CubeSat with multiple payloads

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
Co-supervisor: Sivert Bakken, Roger Birkeland

June 2021

NTNU
Norwegian University of
Science and Technology

Tuva Okkenhaug Moxnes

A common software framework for a CubeSat with multiple payloads

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
Co-supervisor: Sivert Bakken, Roger Birkeland
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Summary

Observation of the ocean is motivated by the need of an increased understanding of climate change effects. A multidisciplinary team is working together at the SmallSatLab at the Norwegian University of Science and Technology (NTNU) with a common goal of designing and developing a hyperspectral payload for a CubeSat called HYPSO. This payload will be integrated in a satellite bus developed by NanoAvionics, and the satellite is planned to be ready for launch in the fourth quarter of 2021. A second mission is planned to follow after this, launching HYPSO-2. This satellite will feature a second payload, the Software Defined Radio (SDR).

The goal of this master's thesis is to demonstrate how to further develop a software platform to have a flexible architecture which can support multiple payloads. The integration of a second payload, the SDR, into the code repository already working for the hyperspectral payload, is used as an example. The design of the software architecture has been crucial for the resulting development, and agile development methods have been investigated and used. Modularization, refactoring and generalization have been the key principles used to obtain the desired result. This has resulted in a new software architecture, which with slight modifications, can support any payload.

Sammendrag

Overvåking av verdenshavene er nødvendig for å øke vår kunnskap om effekten av klimaendringene. Et tverrfaglig team jobber sammen på SmallSatLab ved Norges teknisk-naturvitenskapelige universitet (NTNU). Teamet har som mål å designe og utvikle en hyperspektral nyttelast for en CubeSat kalt HYPSO. Nyttelasten vil bli integrert i en satelitt-bus utviklet av NanoAvionics. Satelitten er planlagt å være klar for oppskyting i fjerde kvartal 2021. Det er planlagt ett etterfølgende trinn med oppskytning av HYPSO-2. Denne nye satelitten vil ha med seg en nyttelast i tillegg, "Software Defined Radio (SDR)".

Målet med denne masteroppgaven er å demonstrere hvordan en kan videreutvikle en programvareplattform til å få en fleksible arkitektur som kan støtte flere ulike nyttelaster. Integrasjon av SDRen inn i kodebiblioteket allerede utviklet for den hyperspektrale nyttelasten er brukt som eksempel. Design av programvarearkitekturen har vært avgjørende for utviklingen, og fleksible (agile) utviklingmetoder har blitt undersøkt og brukt. Modularisering, omstrukturering og generalisering har vært nøkkelprinsipper benyttet for å oppnå det ønskede resultat. Dette har resultert i en ny programvarearkitektur som med små endringer vil kunne støtte enhver nyttelast.

Preface

Private Repositories and Internal Documents

Access has to be granted to the private GitHub repositories of the NTNU SmallSat Lab organization, or internal documents of the NTNU SmallSat Lab organization which are referenced in this master's thesis. This access can be requested from the author's co-supervisor Sivert Bakken at sivert.bakken@ntnu.no.

Personal Information

Consent has been given by everyone featured in the issues or pull requests in the appendix of this master's thesis to have their opinion and identity shown.

Previous work

Prior to working on this master's thesis, the author wrote a specialization project report [1] featuring the design and implementation of a telemetry service logging system of the software variables on the same project. This work was finished during the first sprint of this master's thesis. In addition to this, the author had a summer internship developing software at the SmallSat lab for the HYPSO project. Some of the chapters from the specialization project report are relevant for this master's thesis. The sections listed below can be found with a varying degree of similarity in the specialization project report:

- Chapter 1: section 1.2 is based on the same section in the project report.
- Chapter 2: all sections except 2.2.3, 2.3.3 and 2.4.4 are based on sections in the project report.
- Chapter 4: all sections except 4.3.7 are based on the project report.

Acknowledgements

I want to thank all members of the HYPSO project, and especially the ones on the software team for all the support I have gotten throughout the semester. Whenever I needed help with something hardware related, Roger Birkeland was always there to give me hints and tips on how to solve the problems. Dennis D. Langer has been an important resource who always had an answer when it came to questions related to the code-base. He would also help if I was stuck on a debugging problem, or had ideas to discuss regarding how to solve a problem.

Table of Contents

List of Figures

List of Tables

Acronyms

ACK Acknowledgement. 11

ADCS Attitude Determination Control System. 7

AOSN Autonomous Ocean Sampling Network. 1

ARM Acorn RISC Machine. 9, 10, 13

ASV Autonomous Surface Vehicle. 1

AUV Autonomous Underwater Vehicle. 1

BoB Breakout Board. iv, 8, 9, 20

BSc Bachelor of Science. 1

CAN Controller Area Network. iv, 7, 8, 10–12, 17, 18

CLAW Colored Littoral Zone and Algae Watcher. 6, 7, 12, 17

CLI Command Line Interface. 12, 17, 18, 49

CPU Central Processing Unit. 9

CRC Cyclic Redundancy Check. 11

CSP Cubesat Space Protocol. iv–vi, 10–13, 15, 17, 18, 23, 27, 35–37, 39–43, 53

eMMC Embedded Multimedia Card. 9

EOF End-Of-Frame. 11

EPS Electrical Power System. 6–8, 17, 18, 35, 39, 41

ESD Electrostatic Discharge. v, 20

FC Flight Computer. 7, 17

FPGA Field-programmable Gate Array. 9, 10

FT File Transfer. 15

GB Gigabyte. 9, 10

GPS Global Positioning System. 7, 8

GS Ground Station. 7, 10

HSI Hyperspectral Imaging. viii, 1, 3, 4, 6–8, 11, 15, 17, 34, 38

HW Hardware. iv, viii, 7, 8, 13, 17, 21, 23, 39

HYPSO Hyperspectral Small Satellite for Ocean Observation. i, iii–v, viii, 1, 3–8, 10–12, 16, 17, 21–29, 32, 33, 50, 51, 53, 54

IP Internet Protocol. 20

LEO Low Earth Orbit. 2

M6P Multi-Purpose Nano-Satellite Bus. 6–8, 11, 17

MIO Multiplexed IO. 9

MSc Master of Science. 1

NA NanoAvionics. i, 6, 7, 10, 17

NTNU Norwegian University of Science and Technology. i, iii, 1, 8, 12, 17, 20, 22

OPU On-board Processing Unit. iv, v, 3, 7, 8, 12, 13, 15–18, 26, 32, 35–39, 41–46, 48, 52, 53

OS Operating System. v, 8, 16, 17

PC Payload Controller. 7, 8, 11, 12, 17, 18

PhD Doctor of Philosophy. 1

PL Payload. viii, ix, 1–4, 6–9, 11, 12, 15, 17, 26, 33–38, 42, 43, 45–49, 52–54

PPS Pulse-Per-Second. 8

PR Pull Request. vii, 28, 30–32, 34, 35, 38–49, 51–53, 111, 121, 127, 130, 132, 134, 137, 142

PS Processing System. 9, 10

QSPI Quad-SPI. 9

RAM Random Access Memory. 9, 10

RF Radio Frequency. 9

RGB Red Green Blue. 1, 8, 15

SDLC Software Development Life Cycle. 27

SDR Software Defined Radio. i, ii, iv–vi, viii, ix, 1–4, 6–9, 12, 15, 17–20, 22–24, 26, 32–48, 50–54

SoC System-on-chip. 9, 10

SOF Start-Of-Frame. 11

SoM System-on-module. 9

SSH Secure Shell. 20

SW Software. v, 3–5, 8, 12, 13, 17, 21–24, 26, 27, 33, 35, 36, 42, 50, 51, 53, 54

TM Telemetry. ix, 15, 18, 34, 35, 38–40, 44–48, 52, 53

UART Universal Asynchronous Receiver-Transmitter. 39

UAV Unmanned Aerial Vehicle. 1

UHF Ultra-high Frequency. 1, 2, 7, 8

USB Universal Serial Bus. 8, 9, 11, 17

VCS Version Control System. 29

VPN Virtual Private Network. 20

Chapter 1

Introduction

1.1 The HYPSO Mission

Hyperspectral Small Satellite for Ocean Observation (HYPSO) is a satellite project planned and developed from the SmallSat Lab at the Norwegian University of Science and Technology (NTNU) with a first mission (HYPSO-1) planned to launch in the fourth quarter of 2021 followed by a second mission (HYPSO-2) later. The team consists of multiple departments and disciplines cooperating to finalize the satellite projects. The first mission, HYPSO-1, is a CubeSat containing a Hyperspectral Imaging (HSI) Payload (PL) which aim is to gather, monitor and analyze ocean-color data. This is to be done with intelligent on-board processing using the data from HSI cameras. The desired result from this mission is to have observation of oceanographic phenomena which is close to real time. HYPSO-2 will be an improvement of HYPSO-1. In addition to implement even better solutions to the HSI payload, this mission will include a Software Defined Radio (SDR) as a secondary payload to acquire sensor data where there are harsh environments which induce both high cost and risk at operation. The mission of this PL is to gather data regarding the Ultra-high Frequency (UHF) communication channel and on-orbit interference over selected areas such as the Arctic.

Documentation of work done on the project is extremely important as the team changes every semester. The foundation of the team consist of PhD candidates and postdocs which stay on the team for longer periods of three years or longer. Nevertheless does a great part of the team members include MSc and BSc students which only join the project for one or two semesters. A team that changes this frequently can be an advantage in the regards that new students might see problems differently, but the disadvantage is still the time it requires to understand the project and different concepts. Hence, documentation is key for a successful project.

Ocean observation is motivated by the need of an increased understanding of climate change effects. A HSI camera is used because it can detect multiple wavelengths compared to regular RGB cameras, and one result of this is that it can discover algal blooms. Light is diffracted into separate wavelengths inside the HSI camera, and it can detect wavelengths from 400-800nm. Hence, it is able to detect light given off by algae blooms in the near-infrared spectra. Sea temperatures are expected to rise, and for this reason, algae blooms will likely also increase both in severity and frequency [2]. A reason to care about these algae blooms, is that they are a threat to the fish farming industry in Norway as they are harmful to the fish in the farms, but also to the ecosystems surrounding them [2]. Detecting algae blooms early with satellites can help the owners of the farms save their fish. The satellite will then downlink data to be a part of an Autonomous Ocean Sampling Network (AOSN) including Autonomous Surface Vehicle (ASV)s, Autonomous Underwater Vehicle (AUV)s and UAVs illustrated in Figure 1.1 which can investigate the situation further.

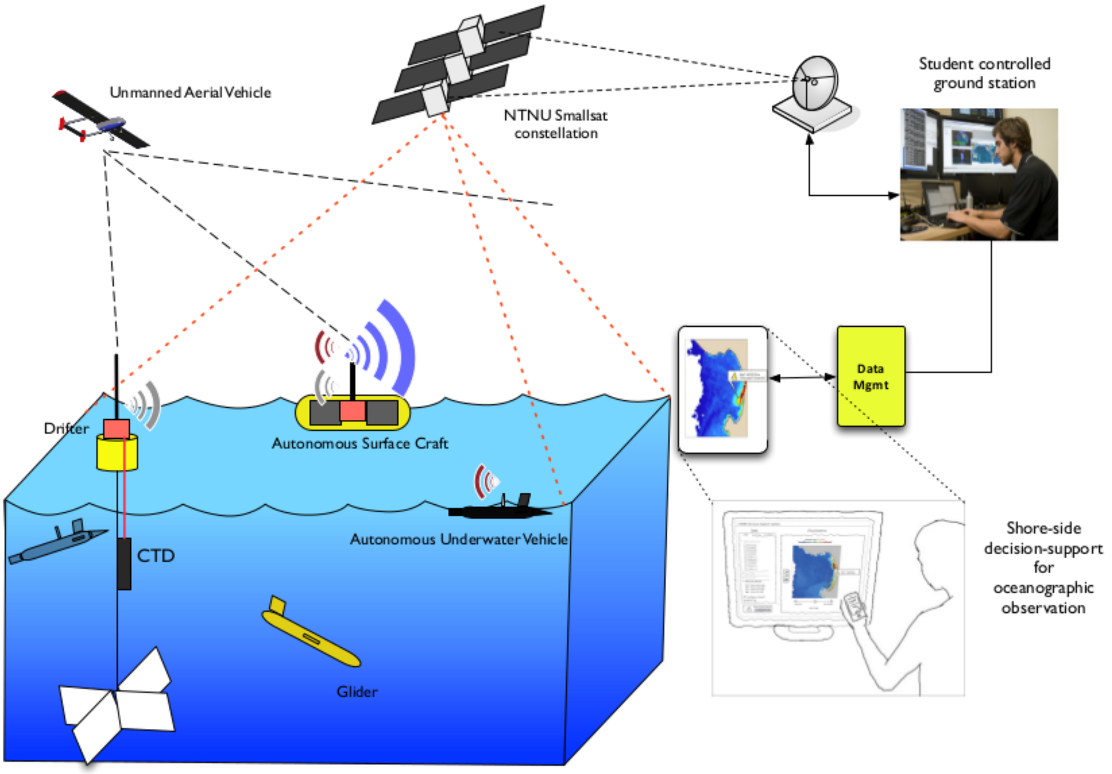

Unmanned Aerial Vehicle
NTNU Smallsat constellation
Student controlled ground station
Drifter
Autonomous Surface Craft
CTD
Autonomous Underwater Vehicle
Glider
Data Mgmt
Shore-side decision-support for oceanographic observation

Figure 1.1: Autonomous network - internal powerpoint

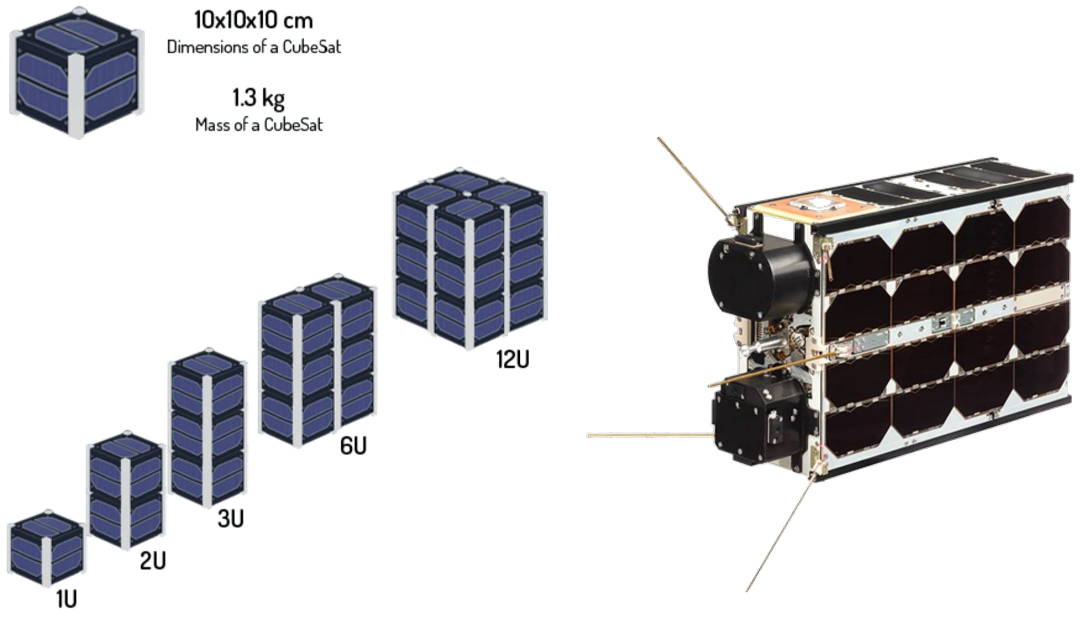
Using a SDR to gather information on communication over the Arctic is driven by the wish to design a communication system between Low Earth Orbit (LEO) satellites and Arctic sensors [3]. A reliable communication system is needed to give Arctic researchers faster access to scientific data where the alternative is expeditions. The SDR is suitable as a secondary PL as its impact on the primary mission can be limited because the measurements can be carried out in a flexible way. Robust and energy efficient high gain antennas are not available for sensor nodes in the Arctic, which is why the lower frequency band UHF is desired.

1.2 CubeSats

This section is based on the same section in the specialization project report [1]. Development and launching of a traditional satellite is very expensive, both with regards to money and time. A small satellite is significantly smaller than a conventional satellite, and a CubeSat is a special version of a SmallSat. Since 1999 CubeSats have been developed to provide affordable access to space for the university science community [4].

One difference between a small satellite and a CubeSat, is the fact that a CubeSat is obligated to have a specific shape, size and weight. While a small satellite is any satellite weighing less than 300kg, a CubeSat has to be a composition of CubeSat units [4]. A Cubesat unit, 1U, is a cube with sides of 10cm weighing up to 1.33 kg. HYPSO is a 6U CubeSat.

Specific standards is contributing to reduced costs when producing CubeSats as it is feasible to mass-produce the components. Hence, companies can buy off-the-shelf parts. There are also reduced costs in transporting and launching satellites of standardized sizes.



(a) CubeSat units. Courtesy of Alén Space. **(b)** 6U CubeSat. Courtesy of NanoAvionics.

Figure 1.2: CubeSat dimensions.

.

1.3 Integrating a Second Payload into Existing Software

The project needs a software platform that can support multiple payloads. Originally, HYPSO-1 was planned to include two payloads, the HSI and the SDR. However, when the final design of HYPSO-1 was decided, the SDR PL was not included, and postponed to HYPSO-2. One of the reasons behind this choice was capacity. A lot of the decisions regarding the Software (SW) and its architecture were therefore made solely with respect to the On-board Processing Unit (OPU). In addition to this, not many detailed functional requirements were made. This has resulted in a development method where the focus is to make the code work, but not on specific requirements. The SW architecture was not explicit at the project beginning, resulting in a gradual definition throughout the life cycle of the system. Definitions of interfaces and architecture are required to facilitate for modular SW. In order to make the SW function with multiple PLs, the new components have to be integrated, and some of the SW architecture has to be redefined. Hence, a lot of changes has to be made in the SW to make it work with multiple PLs.

The SW architecture should be able to support any PL in addition to the OPU, not just the SDR. This includes future PLs that are not yet planned for. An important step on the way, is ensuring that code used by more payloads is generic rather than specific so that code only has to be changed in one place to change the functionality of the whole system. A lot of the code written for the OPU can also work for the SDR with a few modifications. Consequently, refactoring of the code resulting in the PLs sharing some modules is beneficial as the shared code then only has to be updated in one place when changing functionality. Ensuring that all code is made up of modules with specified inputs and outputs, will also give developers a better overview in the future. Making everything generic will be more time consuming, but this will also make it easier for others to make changes in the future. Both adding more payloads, and changing the functionality already existing will be easier when the SW is made more modular.

1.4 Objective and Structure of the Master's Thesis

The overall goal of this thesis is to demonstrate how to further develop a SW platform to have a flexible architecture which can support multiple PLs using the integration of the SDR PL into the code repository already working for the HSI PL as an example. In addition to this, the development done as a part of the thesis will contribute to the HYPSO mission with useful SW. A key aspect of the work done, is planning and design of how to change the code repository and develop it further. As background research before the planning of the SW to be implemented began, different SW architecture fundamentals were investigated in order to write code which is as useful and reusable for others as possible. These fundamentals were used when planning for the implementation of code, and the main focus was to generate generalized and modular code.

The SW implementation consists of refactoring and improving old code in addition to developing new code. To be able to do this as effectively and with as generic solutions as possible, the author came up with suggestions which were discussed with the other team members before implementing the solutions. After implementation, testing of the functionality was performed both by the author and other team members. The main design and implementation job has had a focus on integrating the SDR into the already existing SW used to communicate with the HSI PL. Furthermore, the telemetry logging developed in the specialization project report [1] will be completed with a few additional features. A discussion will reflect upon why it is important for a project to have generalized and modular code, and which considerations were made during development of the code related to this thesis.

The structure of the thesis is as illustrated in Figure 1.3. This differs from a general master structure due to the fact that Chapter 2 and Chapter 3 together make up the background research of the thesis. Chapter 2 is a thorough explanation of the (planned) structure and connections of HYPSO-2 relevant for this thesis, and Chapter 3 includes the research done by the author on principles which should be satisfied in order to build a flexible SW architecture.

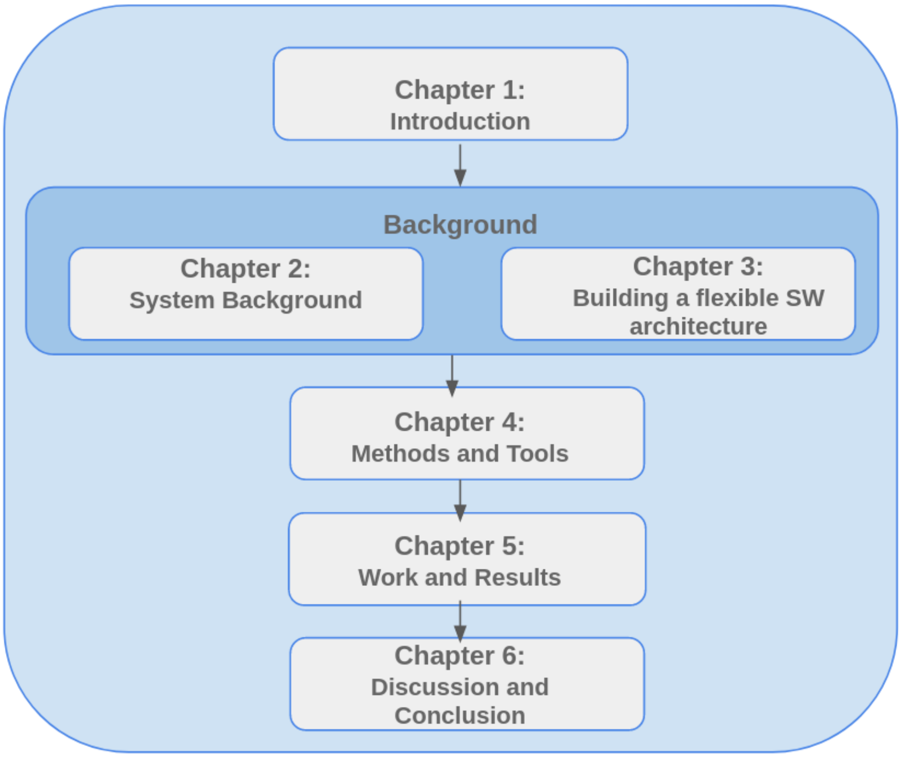


Figure 1.3: Structure of Master's Thesis.

The contents of the different chapters are as listed below:

- Chapter 1: brief introduction with a main focus on the HYPSO mission, the satellite and the goal of the thesis.
- Chapter 2: background theory of the system to understand the work done. Explanation of the satellite, its structure and connections.
- Chapter 3: on principles and advantages of developing a flexible and modular SW architecture.
- Chapter 4: describes the different methods and tools that have been used while developing code for the project.
- Chapter 5: gives a summary of the work done and the results obtained.
- Chapter 6: a discussion and summary of the obtained results, and what could have been done differently. This chapter also features a section on future work.

Chapter 2

System Background

In the specialization project report [1], the author focused on explaining the satellite structure and connections of HYPSO-1. This chapter will rather explain the plans for HYPSO-2, as the master's thesis regards the integration of the second PL planned for this mission. The present chapter will introduce theory about the satellite, its structure and connections relevant for the work performed by the author during the scope of this thesis. It is based on the corresponding chapter of the project thesis [1].

2.1 The HYPSO Satellite

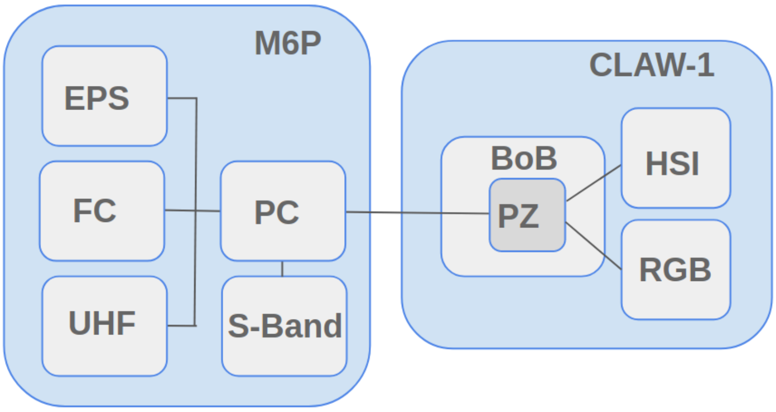


Figure 2.1: Satellite connections of HYPSO-1

As mentioned in Section 1.1, HYPSO-1 consists of a satellite bus developed by NanoAvionics (NA) and a HSI payload while the most important difference for HYPSO-2 is the secondary PL, the SDR. NA specializes on CubeSats, and delivers a 6U nanosatellite bus, M6P with dimensions $10cm \times 20cm \times 30cm$, to the HYPSO project. As stated on NAs website [5], the M6P satellite bus will let the customers focus on the goals of the mission and implement high-level mission tasks only. The different parts and interfaces of HYPSO-1 are shown in Figure 2.1, while the differences in HYPSO-2 are illustrated in Figure 2.2. The power cables connected to the Colored Littoral Zone and Algae Watcher (CLAW) and the SDR from the EPS are not present in the figures, but are illustrated in Figure 2.9 in Section 2.5.

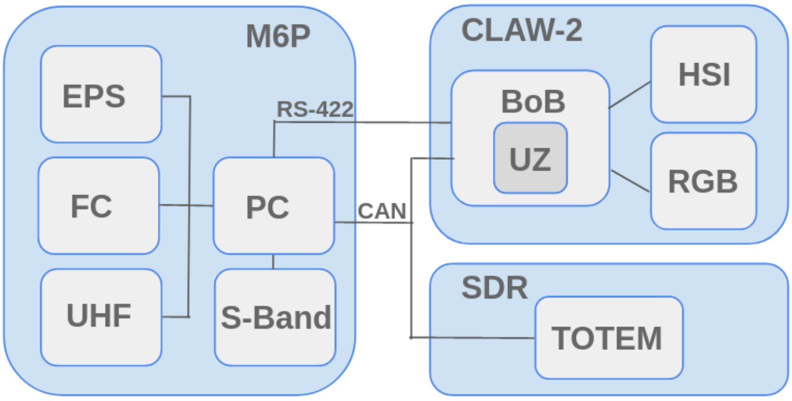


Figure 2.2: Satellite connections of HYPSO-2

The different modules of the SDR and the HSI payload, CLAW-2, will be explained in Section 2.2. Listed below are the different modules in the M6P, where some of its subsystems are excluded from the figure:

- **FC:** the Flight Computer performs activities related to the ADCS and collects data from the sensors and GPS. Scripts can also be used on the FC in order to make it execute different commands at given times. One example of such a command is booting the OPU or make it take an image.
  - **GPS:** the Global Positioning System is the navigation system of the satellite.
  - **ADCS:** the Attitude Determination Control System is responsible of pointing and slew maneuvering. It has a redudndant pair of avaliable actuators which are based on different technologies.
- **EPS:** the Electrical Power System collects its energy from the solar panels and stores it in batteries. Whenever other subsystems need power, the EPS provides and regulates it. Moreover, it is equipped with fail-safe mechanisms making it avoid electical damage, both to other subsystems and itself.
- **PC:** the Payload Controller controls every interface between the PLs and the satellite platform. This also includes the CAN bus explained in Section 2.3.2 and power connections from the EPS.
- **UHF:** the Ultra-high Frequency radio communicates with the Ground Station (GS).
- **S-band:** this radio is also used to communicate with the GS. The radio has larger bandwidth and throughput than the UHF, but had the disadvantage that it requires the satellite to point directly at theGS it is communicating with.
- **Solar Panels:** Attached to the satellite frame, used to collect energy from the sun to the satellite.

More information about the satellite bus can be found on NAs website [5].

2.2 Planned Payload Hardware for HYPSO-2

HYPSO-2 is planned to have two payloads; in addition to the main HSI payload which is called CLAW-2, this satellite will also feature a SDR payload. The different HW instruments of the two PLs, OPU[6] and the SDR [7] are listed below:

- **On-board Processing Unit (OPU)**:

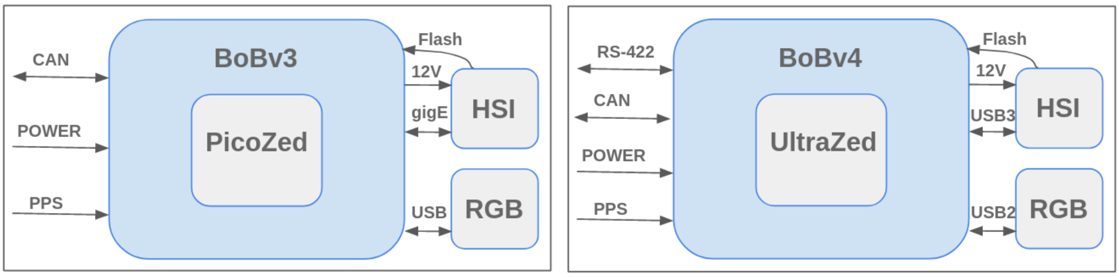
- **UltraBOB:** on-board processing board and interface to satellite PL. Combination of a Breakout Board (BoB) and a UltraZed running the SW of the on-board processing.
- **HSI (Hyperspectral Imaging):** for high-spectral images.
- **RGB Camera:** used to georeference and register the images obtained with the HSI camera.
- **Payload structure:** support for ensuring the mechanical integration on the M6P.

- **Software Defined Radio (SDR)**:
  - **Totem:** high-performance nanosatellite SDR platform. Features an UHF front end, embedded linux and has support to deploy multiple SDR applications

HYPSO-1 has a quite similar PL HW instruments as the ones listed for HYPSO-2 above. The main difference is that the SDR is not included in HYPSO-1. In addition to this, HYPSO-1 has a PicoZed instead of an UltraZed. The payload HW relevant for this thesis are the Breakout Board and the OPU, which features the UltraZed, in addition to the TOTEM. These will be explained next.

2.2.1 Breakout Board (BoB)

The different versions of the Breakout Board are developed at NTNU, and their purpose is to be interfaces, both mechanical and electrical, between the other modules of the satellite [8]. HYPSO-1 use BoBv3, while HYPSO-2 will use BoBv4 which is still being developed at the SmallSat Laboratory. As seen in Figure 2.3, the BoBs are used to route signals and power, and the interfaces are slightly different for HYPSO-1 and HYPSO-2. For HYPSO-1, the BoBv3 is connecting the PicoZed to the cameras and the M6P satellite bus. The HSI camera is connected through a Gigabit-Ethernet cable and using a HIROSE cable for power and flash signal. The PicoBoB is connected to the PC with a CAN interface which will be explained in Section 2.3.2. The BoBv4 in HYPSO-2 will connect the UltraZed to the cameras and the M6P satellite bus. Here, the planned interface for the HSI camera is a USB3 cable in addition to the HIROSE cable for flash signal and power. The reason for upgrading the Gigabit-Ethernet cable from HYPSO-1 to a USB3 cable is to enable for a higher frame rate. As mentioned in Section 2.3.3, the UltraBoB is interfacing with the PC with a RS-422 cable for faster data transfer in addition to the CAN bus. Both the PicoBoB and the UltraBoB connect to the RGB camera with a USB2 cable, to the EPS with powerlines and to the GPS with a PPS(Pulse-Per-Second) signal which goes through the PC.



(a) HYPSO-1: PicoBoB

(b) HYPSO-2: UltraBoB

Figure 2.3: HSI Payload Hardware with interfaces

2.2.2 On-board Processing Unit

The On-board Processing Unit (OPU) is the control part of the HSI Payload which performs processing on the HSI data. It is the processing platform which together with the HSI camera and the RGB camera forms the PL. The OS (Operating System) running on the OPU is Embedded Linux, which will be explained in Section 2.4.5. The OPU in HYPSO-2 will feature an UltraZed

SoM (System-on-module) developed by Avnet which will be mounted on the BoB as seen in Figure 2.3, resulting in the UltraBoB. From the designer's guide [9], the specifications of the UltraZed are as follows:

- PS (Processing System): Xilinx XCZU7EV-1FBVB900 industrial grade SoC (System-on-chip) featuring
  - ARM (Acorn RISC Machine) processor.
  - Two Central Processing Unit (CPU) cores.
- Programmable Logic: FPGA (Field-programmable Gate Array).
- Memory:
  - 4GB RAM (Random Access Memory)
  - 8GB eMMC (Embedded Multimedia Card)
  - 64 MB QSPI (Quad-SPI)
- Interfaces:
  - Gigabit Ethernet
  - USB 2.0 controller
  - MIO (Multiplexed IO) pins
  - Three 100-pin Micro Headers

The 152-pin Micro Headers are holding the electrical interfaces of the UltraZed. These lets the UltraZed interface with the BoB to construct the UltraBoB. Hence, the UltraZed performs processing on the PL when mounted on the BoB which lets the UltraZed interface with other modules on the satellite.

2.2.3 Software Defined Radio

The processing on the second PL is done by the TOTEM SDR which also runs a Linux kernel on a dual ARM Cortex-A9 processor. TOTEM was chosen due to cost and schedule constraints, and consists of an analogue RF (Radio Frequency) front-end and a SDR motherboard with an RF transceiver and a SoC based on Xilinx boards(Zync 7020). An illustration of the SDR and its interfaces can be seen in Figure 2.4. The programmable logic of the TOTEM is also a FPGA.

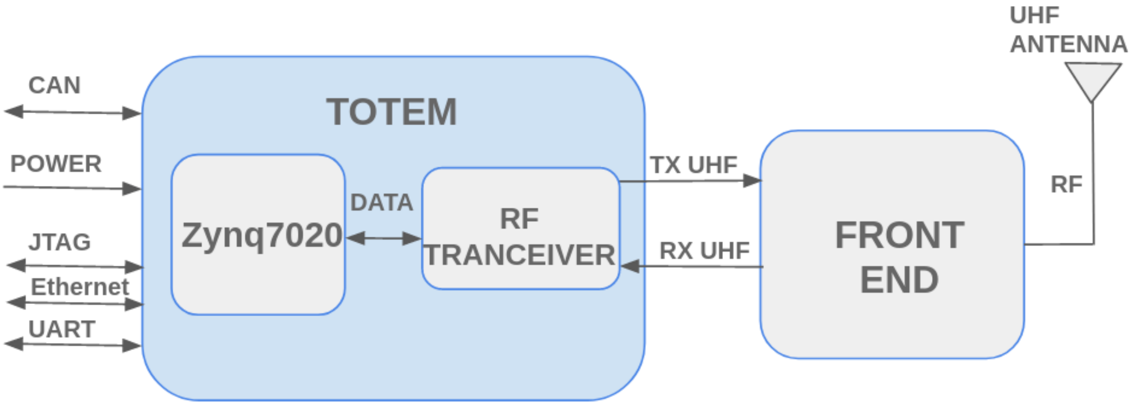


Figure 2.4: SDR with interfaces

From the datasheet [7], the specifications of the TOTEM are as follows:

- PS (Processing System):a SoC based on Xilinx boards(Zync 7020) (System-on-chip) featuring a Dual ARM Cortex-A9 processor.
- Programmable Logic: FPGA (Field-programmable Gate Array).
- Memory:
  - 1GB RAM
  - 1GB NAND Flash
- Interfaces:
  - CAN Bus
  - UART
  - JTAG
  - Ethernet

2.3 Network Communication

The following section is a modified version of what was presented as Section 2.2 in the project thesis [1] written by the author. All subsections apart from Section 2.3.3 are inspired by this.

The understanding of the different communication protocols that are used in this project can be obtained by looking at the five-layer network model seen in Table 2.1. This network model is explained in [10]. Viewing the complex network communication in layers can increase the understanding of it, as opposed to considering it as a single system. System layers provide structure to network designers when they are designing protocols. Every layer in the model provides a service to the layer above by performing certain actions itself and using the services from the layer directly below it. Each layer's implementation is independent of the other layers as the implementation in encapsulated, they only use each other's services.

#	Layer
5	Application
4	Transport
3	Network
2	Link
1	Physical

Table 2.1: Five-layer Internet protocol stack

The implemented protocols in the HYPSO CubeSat related to this thesis are the link layer protocol CAN (Controller Area Network), and the transport layer protocol CSP (Cubesat Space Protocol). The transport layer provides transportation of application-layer messages between application endpoint, and is providing increased reliability for data delivery. The link layer delivers finitely long messages between two nodes on the route from one host to another. As seen in Table 2.1, the transport layer is above the link layer. Thus, CAN provides services to CSP.

2.3.1 CSP

The Cubesat Space Protocol is a transport layer protocol developed specifically for CubeSats. CSP has a router-core (Network Layer) which again features interfaces to the link and physical layers in the satellite [11]. Thus, the CSP network has multiple ways of connecting its nodes. The payload and its subsystems in the satellite, employs CSP as its external communication protocol for communication with the satellite bus. The M6P satellite bus from NanoAvionics communicates using CSP both between its internal sub modules and externally to the GS [12]. Each sub module in the satellite has a unique CSP address, and the operator at the GS will have one as well [13].

2.3.2 CAN

The CAN-bus is a physical connection between different modules of the satellite. The bus has two wires for transportation, ideally twisted to reduce noise. The extended CAN is a protocol in the link layer, and it is specified by the ISO-11898:2003 standard [14]. This standard states that CAN has a maximum signal rate of 1 Mbps when the bus has 30 nodes or less and a length of up to 40m. The standard also lists different features of the protocol:

- Has a 29-bit identifier field. (11 for standard CAN)
- Can send a maximum of 8 Bytes per frame.
- Does CRC (Cyclic Redundancy Check): contains checksum for verification of the received messages integrity.
- Has ACK (Acknowledgement): used to acknowledge the integrity of the data.
- Marks SOF (Start-Of-Frame) and EOF (End-Of-Frame).

There are two CAN-buses in the satellite which are routed through the Payload Controller. CAN1 connects the modules of the M6P bus. CAN2 connects the two PLs to the PC, which communicates the signals to the rest of the M6P. The way the subsystems are connected can be seen in Figure 2.5. The different CAN interfaces will route CSP packets encapsulated in CAN back and forth, making the different subsystems communicate.

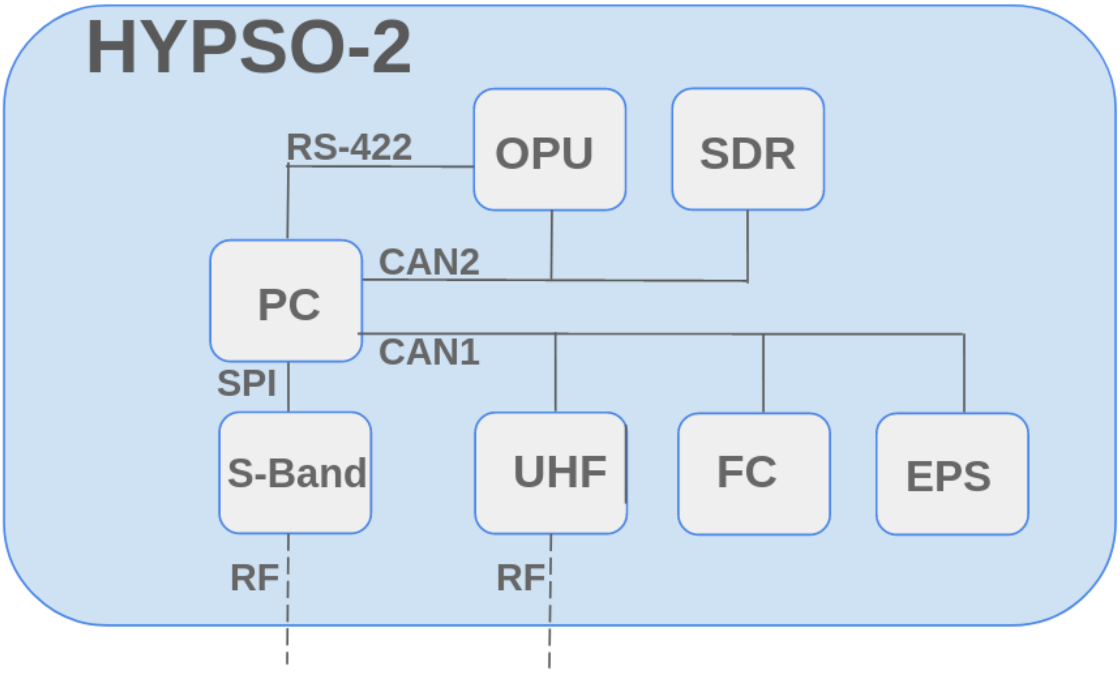


Figure 2.5: CAN network in HYPSO-2

Another link layer protocol in the HYPSO-1 satellite is the Gigabit Ethernet, which is a substandard of Ethernet. This is used as an interface between the PicoBoB explained in Section 2.2 and the HSI camera. Currently, this protocol is considered changed for a USB3 for the HYPSO-2 because it might enable for higher frame rates.

2.3.3 RS-422

The following section is based on an internal document regarding the A difference for the HYPSO-2 compared to the HYPSO-1 is the RS-422 which at the moment is planned to be another link layer connection in addition to the CAN interface between the CLAW-2 and the PC. This can be seen in Figure 2.5 from Section 2.3.2. The RS-422 is a definition of the electrical signal interface between nodes of a serial data bus which transforms a ground referenced serial input signal to a differential serial output signal with a different voltage at the transceiver side, and vice-versa on the receiver side [15]. The reason for adding this to HYPSO-2 is that CAN has limited throughput, and the actual overhead of CAN was larger than expected and not sufficient when transferring large amounts of data from the PL. RS-422 can be configured to a higher data transfer rate compared to CAN. The anticipated difference between RS-422 and CAN is a net rate on 1-3Mbps for RS-422 compared to 0.4Mbps for CAN. For HYPSO-2, CAN will be kept in parallel as a backup.

2.4 Payload Software

The following section is a continuation of Section 2.4 merged with section 4.4 in the specialization project report [1]. The code developed for this master's thesis, is a part of the code under version control on GitHub for the NTNU-SmallSatLab organization. GitHub and its methods will be described in Section 4.3. There are three repositories which are relevant to understand in order to understand the development done in this thesis. The main one is `hypso-sw` [16], which is the repository the author has developed code for. There has not been written any code by the author for the remaining two, `opu-system` [17] and `sdr-system` [18]. These repositories contain the necessary tools for building the firmware for the respective system, and they are used by `hypso-sw` to build the executables. Within the scope of this thesis, the author has solemnly used the repositories to build executables in `hypso-sw`.

2.4.1 Toolchains

In the context of SW development, the *Toolchain* refers to the process of refining the source code files into a program or other files that are needed for the system. As listed in [19], the process of creating a program of code can consist of several different steps such as a static analyzer and a compiler. The static analyser will determine if the program behaviour is correct with the help of program invariants, while the compiler will be used to translate the developed code to machine code. A toolchain can consist of other steps as well. The toolchain used for the `hypso-sw` repository consists of makefiles. These makefiles will link files developed by the SW team with other libraries to compile the source code.

Docker ensures that the building of the executables from the `hypos-sw` repository is consistent within the team. The reason for using it is that a specific toolchain is needed for compiling the source code. The way docker handles this is by running a virtual environment - a docker container - on any computer[20]. Specific software is installed inside the container so that when a developer uses it, the software accessible will be the same regardless of the computer it is run on. Thus, the docker container can be used to contain the specific toolchain needed to compile the source code for `hypso-sw`. The container will guarantee that the same environment is used every time the exectables are built, not depending on the operating system or the host computer used.

2.4.2 hypso-sw

The following is based on the authors experience using the repository, in addition to the repository itself [16]. All source code contained in this repository is written in C. The source code generates executables, where the three that are relevant for this thesis are the software to run on the OPU, the software to run on the SDR, and a Command Line Interface (CLI) to communicate with the two others using CSP. Thus, the repository compiles one client that pairs up with both PL services:

- `hypso-cli`: compiled for regular computers(x86). The executable will parse user input and create CSP packets to be distributed over the CAN network in the satellite. This is the main tool for communicating with the satellite.

- `opu-services`: mainly compiled for ARM which is the processor of the OPU. It can also be compiled for x86 architecture for SW testing that does not require HW. This version was mostly used early in the timeline of the project and is not relevant for this thesis. The executable will interpret CSP packets, and performs the actions requested.
- `sdr-services`: compiled for ARM which is the processor of the `totem`. Just like for `opu-services`, a version can be compiled for x86 architecture. The `sdr-services` executable will interpret CSP packets similar to `opu-services`, and perform the actions requested.

To make the process of compiling the SW easier, the toolchain is encapsulated inside a Docker container as explained in Section 2.4.1. Hence, the code is compiled inside the container by the command `make` to generate `hypso-cli`, `opu-services` and `sdr-services` for x86. To generate the ARM version of `opu-services`, the developer writes the command `make ARCH=arm` inside the Docker container. The `sdr-services` is generated using the command `make ARCH=sdr`. The order of the generation of the executables can be seen in Figure 2.6 from starting the Docker container inside `hypso-sw/scripts`. The script will scan the file `CMakeLists.txt` for dependencies for the executable which is to be built, and the connections between the build process and the `CMakeLists.txt` is illustrated in the figure with the numbered boxes instead of drawing out each arrow as this version is clearer. This file has to be changed every time a new source file is added to the `hypso-sw` repository. The new file has to be added to the list of executables for the applications the files are to be used in. The toolchain works in reverse for building `sdr-services` compared to `opu-services`. To be able to build `sdr-services`, a pre-compiled version of `sdr-system` has to be present in the same repository as `hypso-sw`. The reason for this is that the compiler has to be built before `sdr-services` can be built in Docker. For the OPU it is the other way around. The compiler for `opu-services` is already installed in the Docker container, and it is ready to be built. To be able to build `opu-system`, a pre-built version of `opu-services` has to be present in a `hypso-sw/build/arm` folder located at the same computer as the `opu-system` repository in order to include the SW in the system.

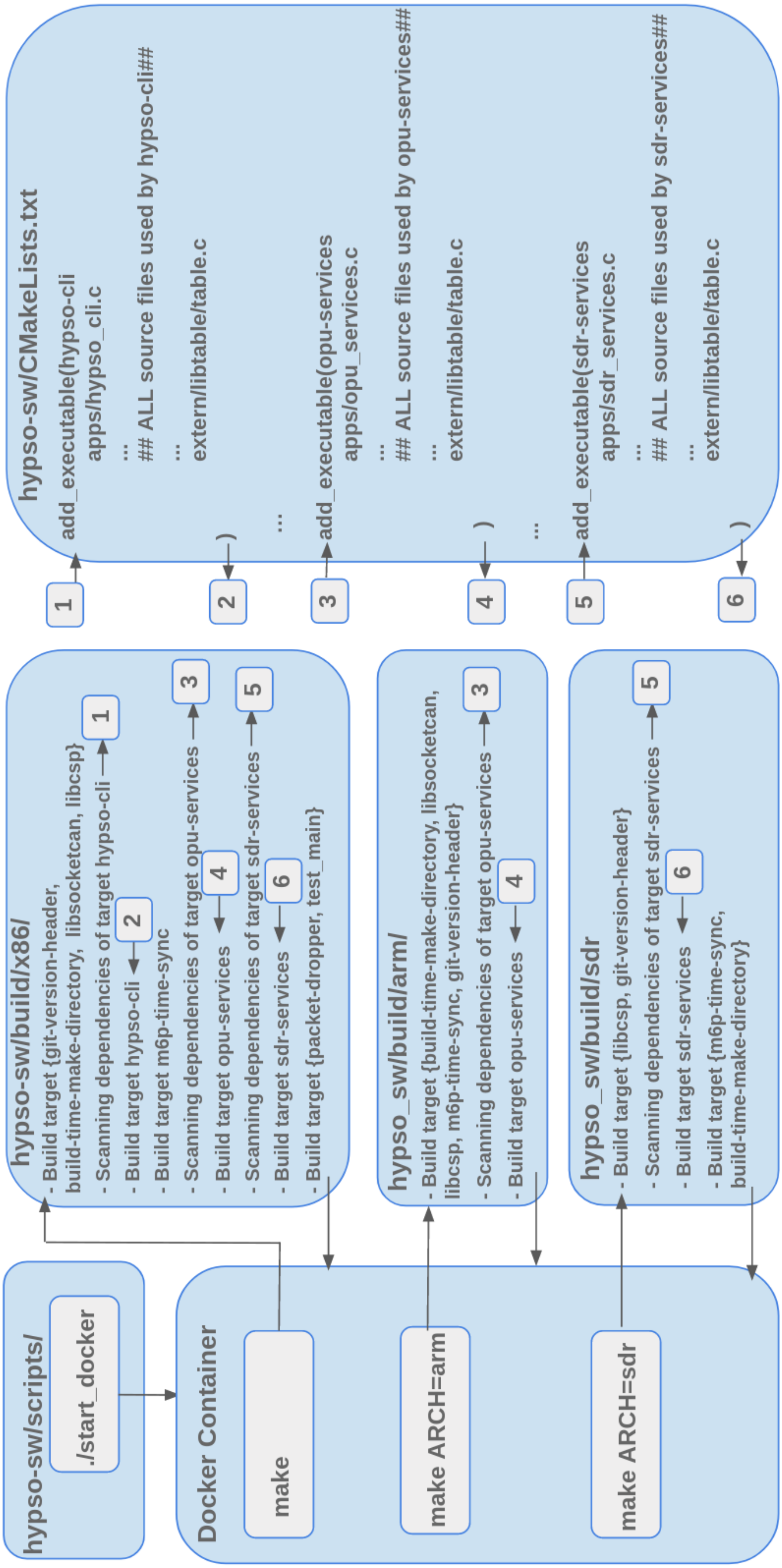

hypso-sw/scripts/
./start_docker
Docker Container
make
make ARCH=arm
make ARCH=sdr
hypso-sw/build/x86/
- Build target {git-version-header, build-time-make-directory, libsocketcan, libcsp}
- Scanning dependencies of target hypso-cli
1
- Build target hypso-cli
2
- Build target m6p-time-sync
- Scanning dependencies of target opu-services
3
- Build target opu-services
4
- Scanning dependencies of target sdr-services
5
- Build target sdr-services
6
- Build target {packet-dropper, test_main}
hypso_sw/build/arm/
- Build target {build-time-make-directory, libsocketcan, libcsp, m6p-time-sync, git-version-header}
- Scanning dependencies of target opu-services
3
- Build target opu-services
4
hypso_sw/build/sdr
- Build target {libcsp, git-version-header}
- Scanning dependencies of target sdr-services
5
- Build target sdr-services
6
- Build target {m6p-time-sync, build-time-make-directory}
hypso-sw/CMakeLists.txt
add_executable(hypso-cli
apps/hypso_cli.c
...
## ALL source files used by hypso-cli##
...
extern/libtable/table.c
)
...
add_executable(opu-services
apps/opu_services.c
...
## ALL source files used by opu-services##
...
extern/libtable/table.c
)
...
add_executable(sdr-services
apps/sdr_services.c
...
## ALL source files used by sdr-services##
...
extern/libtable/table.c
)
1
2
3
4
5
6

Figure 2.6: Build executables from `hypso-sw`(top left), using the make-commands from the Docker Container.

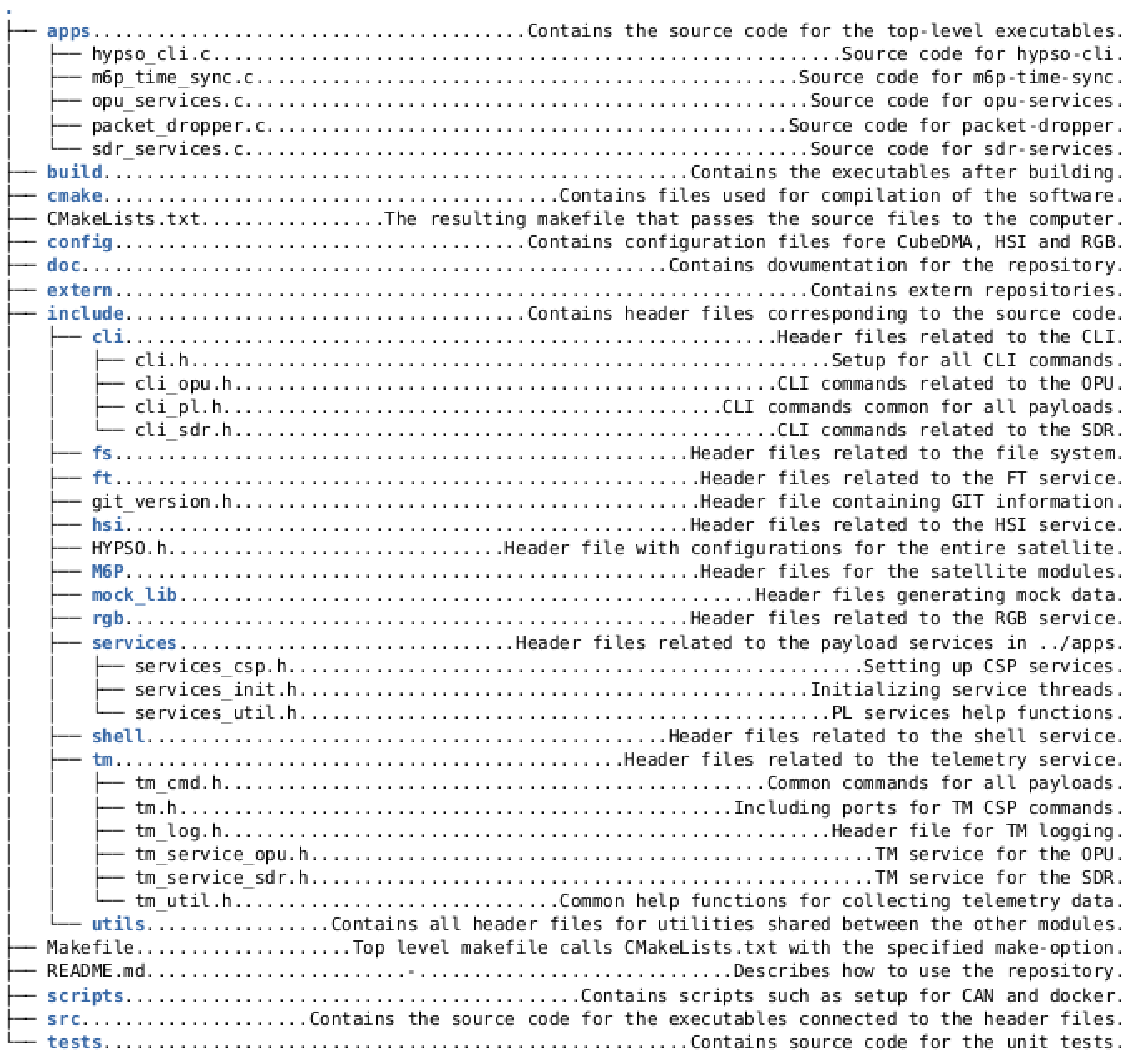
When `sdr-services` starts running on the SDR, or `opu-services` starts running on the OPU, several threads which function as services will be initialized. The services accept CSP connections sent to their ports by or another client interface. Two services are related to the cameras, and are only running on the OPU:

- RGB service: this task handles the interfacing with the RGB camera, and the commands related.
- HSI service: this task handles the interfacing with the HSI camera, and typically captures and processes HSI cubes.

The SDR does not have any private services yet, but there will probably be developed a service in the future to run application code related to the radio. An example of use for this service would be to gather data connected to the on-orbit interference over the Arctic. The rest of the services run on both PLs, and their tasks are as follows:

- FT (File Transfer) service: this task manages file operations.
- Shell service: this task accepts commands and executes them in a shell.
- TM service: this task handles requests related to TM.
- CSP service: this task task accepts all CSP connections that are not bound by any of the other services.

The structure of the repository the structure at the present time is illustrated in Figure 2.7. It includes the main directories in addition to the files that were significant in the development part of this thesis.



```
.
├── apps.......................................Contains the source code for the top-level executables.
│   ├── hypso_cli.c.....................................................Source code for hypso-cli.
│   ├── m6p_time_sync.c.............................................Source code for m6p-time-sync.
│   ├── opu_services.c...............................................Source code for opu-services.
│   ├── packet_dropper.c...........................................Source code for packet-dropper.
│   └── sdr_services.c...............................................Source code for sdr-services.
├── build....................................................Contains the executables after building.
├── cmake........................................Contains files used for compilation of the software.
├── CMakeLists.txt................The resulting makefile that passes the source files to the computer.
├── config....................................Contains configuration files fore CubeDMA, HSI and RGB.
├── doc.................................................Contains dovumentation for the repository.
├── extern.........................................................Contains extern repositories.
├── include...................................Contains header files corresponding to the source code.
│   ├── cli.......................................................Header files related to the CLI.
│   │   ├── cli.h.....................................................Setup for all CLI commands.
│   │   ├── cli_opu.h...............................................CLI commands related to the OPU.
│   │   ├── cli_pl.h.........................................CLI commands common for all payloads.
│   │   └── cli_sdr.h...............................................CLI commands related to the SDR.
│   ├── fs...................................................Header files related to the file system.
│   ├── ft.....................................................Header files related to the FT service.
│   ├── git_version.h.......................................Header file containing GIT information.
│   ├── hsi..................................................Header files related to the HSI service.
│   ├── HYPSO.h............................Header file with configurations for the entire satellite.
│   ├── M6P.....................................................Header files for the satellite modules.
│   ├── mock_lib..................................................Header files generating mock data.
│   ├── rgb..................................................Header files related to the RGB service.
│   ├── services..............................Header files related to the payload services in ../apps.
│   │   ├── services_csp.h.................................................Setting up CSP services.
│   │   ├── services_init.h............................................Initializing service threads.
│   │   └── services_util.h...............................................PL services help functions.
│   ├── shell...............................................Header files related to the shell service.
│   ├── tm..............................................Header files related to the telemetry service.
│   │   ├── tm_cmd.h...............................................Common commands for all payloads.
│   │   ├── tm.h...............................................Including ports for TM CSP commands.
│   │   ├── tm_log.h........................................................Header file for TM logging.
│   │   ├── tm_service_opu.h.......................................................TM service for the OPU.
│   │   ├── tm_service_sdr.h.......................................................TM service for the SDR.
│   │   └── tm_util.h............................Common help functions for collecting telemetry data.
│   └── utils................Contains all header files for utilities shared between the other modules.
├── Makefile...................Top level makefile calls CMakeLists.txt with the specified make-option.
├── README.md......................-...........................Describes how to use the repository.
├── scripts.......................................Contains scripts such as setup for CAN and docker.
├── src....................Contains the source code for the executables connected to the header files.
└── tests...................................................Contains source code for the unit tests.
```

Figure 2.7: Structure of the hypso-sw repository

2.4.3 opu-system

The explanations in this subsection are based on the README-file of the repository [17] and the authors experience with it. The authors use of the repository `opu-system` in the thesis consisted of generating bootfiles for the OPU. The bootfiles are used for booting the OS of the OPU, meaning the files locate, load and initialize the OS when the OPU is powered on. The main bootfile of the OPU is the image `image.ub`. As for now, the test-setup for the OPU in the lab is the one of HYPSO-1. Hence, the bootfiles are built for the PicoZed, not the UltraZed. They can be built either in Prototyping Mode, which only creates a primary image, and where `opu-services` has to be started manually after boot. This version can be good for testing purposes as the developer might want to test different versions of `opu-services`. The other mode, Deployment Mode, is the one that is to actually run on the satellite after deployment. Here, `opu-services` will start automatically at boot. In addition to this a backup image, the Golden Image, will be generated. The current test setup is run in Deployment Mode, and the development done with regards to this thesis did not require this image to be changed. Hence, the repository was only used by `hypso-sw` to build `opu-services`.

2.4.4 sdr-system

As for the OPU, the explanations in this subsection are based on the README-file of the repository itself [18] and the authors experiences. The content of the repository is the tools necessary for building the TOTEM firmware which is provided from Alén Space. Just like `hypso-sw`, `sdr-system` uses a Docker container for the build process. During the scope of the master's thesis, the firmware was never a problem, and therefore never had to be updated by the author. Thus, this repository was mainly used in order to understand the system in addition to building the `sdr-services` from . For the building of `sdr-services`, a version of `sdr-system` with a prebuilt firmware has to be present in the same folder as the `hypso-sw` repository which the `sdr-services` is built from.

2.4.5 Operating System

As mentioned in section 2.2.2 and section 2.2.3, both the OS running on the OPU and the one running on the SDR will be Embedded Linux. The version running on the OPU is Linux based on the yocto project [21]. The embedded Linux system which is run on the TOTEM is built using Buildroot [18]. Linux is a free and open-source OS. Hence it can be customized for a particular use-case. Embedded Linux is a result of such a customization to generate a light-weight version of Linux. The use-case in mind in the yocto project is embedded devices as for instance HYPSO's OPU and SDR.

2.5 Lab setup and testing

Testing is done in a similar way as for the specialization project report [1], thus the following section is inspired by Section 2.6 regarding the testing of the code developed for this. To understand the testing done of the different SW functionality developed, there are several parts of the HW setups in the lab which are relevant. The focus in the earlier parts of this chapter has been on HYPSO-2, as the development of code done within the scope of this master's thesis has been for this version of the satellite. Even though that is the case, the testing has been done on a CLAW-1 setup for the OPU part as this is what is present in the lab for the time being. Thus, testing is done on a PicoZed, not an UltraZed. A RS-422 cable between the PC and the OPU is not present, and neither is the USB3 cable for the HSI camera. The SDR setup in the lab represents what is meant to be in HYPSO-2. The testing of the OPU is done on a machine called LidSat while the machine for testing the SDR is called Totem, and they are both located in the SmallSat Laboratory at NTNU. Furthermore, one part of the test setup is located at NanoAvionics (NA)' facilities in Vilnius, Lithuania. This part is called FlatSat, and it is connected to LidSat through a virtual CAN-bridge. An illustration of the whole setup can be seen in Figure 2.8.

Most of the satellite modules from the M6P bus explained in Section 2.1 are present in the test setup. This includes the EPS and the PC which are parts of LidSat, while the FC and the rest of the satellite bus are parts of the FlatSat[22]. The EPS is used when testing to check the power status of the different PL modules, or to turn them on and off. The PC connects the payload to the rest of the satellite bus. The developer can connect to the EPS through `hypso-cli` with CSP address **4** using the command `shell remote 4 <timeout(s)>`, and similarly using CSP address **6** for the PC. As seen in Figure 2.8, the equipment in the SmallSat Laboratory at NTNU includes the HSI payload and the SDR payload in addition to the mentioned M6P modules. From `hypso-cli`, the user can connect to the OPU through CSP address **12**, and the SDR through CSP address **13**. Further is `hypso-cli` itself started on the LidSat from CSP address **14**.

There are two CLIs in use in the lab for testing purposes. When possible, it is preferable to use `hypso-cli`, which was explained in Section 2.4.2. This has been sufficient for all testing done related to this thesis. This is used for communication with the payloads, both the OPU and the SDR. It can send commands to the PLs, and remotely log into them. NanoMCS is the other CLI in use in the lab, and it is developed by NA. NanoMCS can be used both for operating the FlatSat and the subsystems in the lab, and it is typically used if there is a lack of functionality in

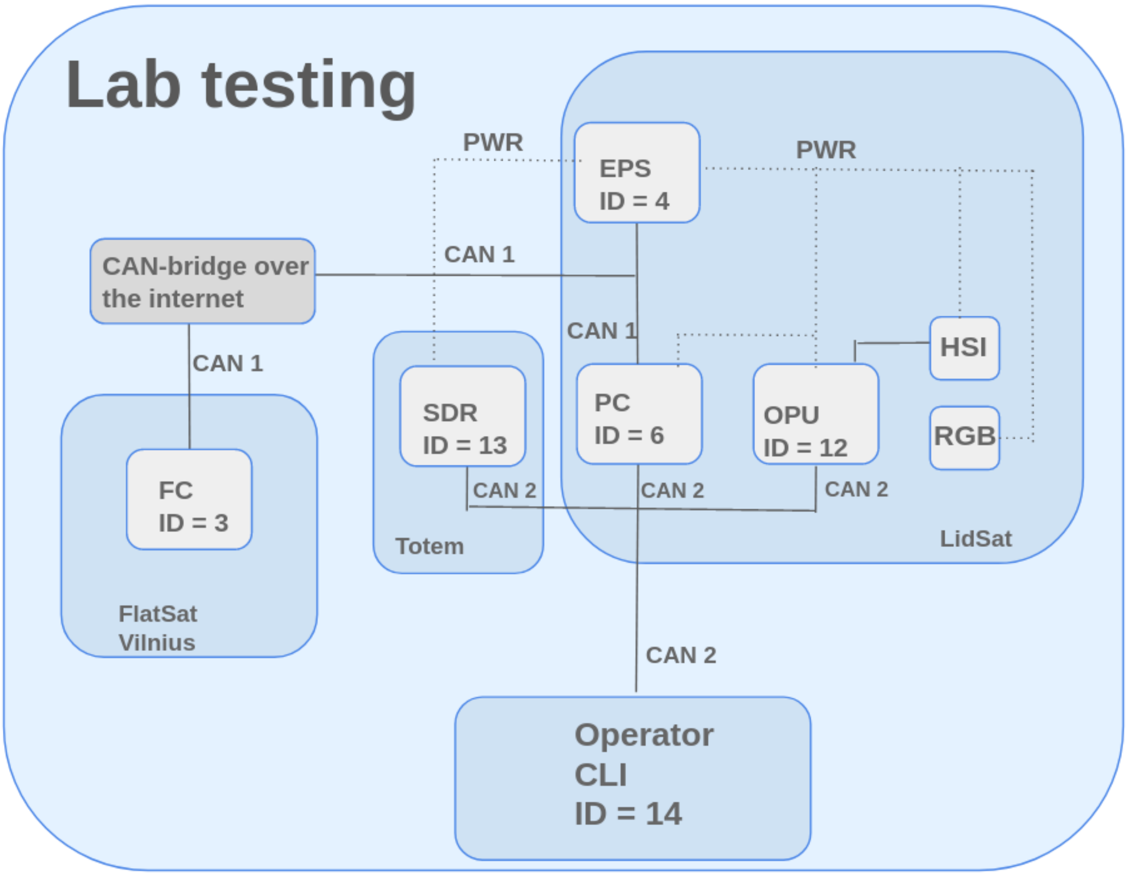


Figure 2.8: LidSat and FlatSat

`hypso-cli`. NanoMCS was used to parse the log files generated by the Telemetry-Service in this thesis. While testing, the CLI is run on the operator computer in Figure 2.8. Further, the user input is parsed into CSP commands. These commands will then be distributed over the CAN-network in the satellite. When the OPU is running with its services, these will accept the CSP commands sent from the CLI, and with the added functionality of this thesis, the same applies to the SDR.

To enable for parallel testing of the OPU, there are two PicoBobs (Section 2.2) in the test setup. One PicoBob means one OPU, hence two people can perform testing at the same time with two PicoBobs. With a lot of team members working remotely (Section 2.5.2) due to COVID-19 this has been even more important so that multiple people can access OPUs at the same time. Upon starting `hypso-cli`, the user tells it which PicoBob to send messages to. During the scope of the master thesis, all testing done by the author has been through **ssh** which is explained in Section 2.5.2. Running of `hypso-cli` is done by signing in to the LidSat, this is also where `opu-system` is started from which is automatically running `opu-services` as explained in Section 2.4.3. To run `sdr-services`, the developer has to sign in to the TOTEM, and start the preferred version through CSP address **13**.

In conclusion, the author will use a running version of `hypso-cli` on the operator computer to communicate with a running version of `opu-services` on the OPU, and a running version of `sdr-services` on the TOTEM, in order to test new functionality developed in `hypso-sw`. Images of the setup in the lab are shown in Figure 2.9 and Figure 2.10. From the left in Figure 2.9 are: the CAN-adapters are the two boxes left of the black lid, the PC is the grey box, the EPS is in the bottom right corner and the PicoBoB is located in the top right corner. The cameras are not featured in the image. The SDR is connected to the EPS with a long power cable, and to the PC with one of the CAN adapters. The SDR itself is featured in Figure 2.10, and the CAN bus can be seen in the top right corner..

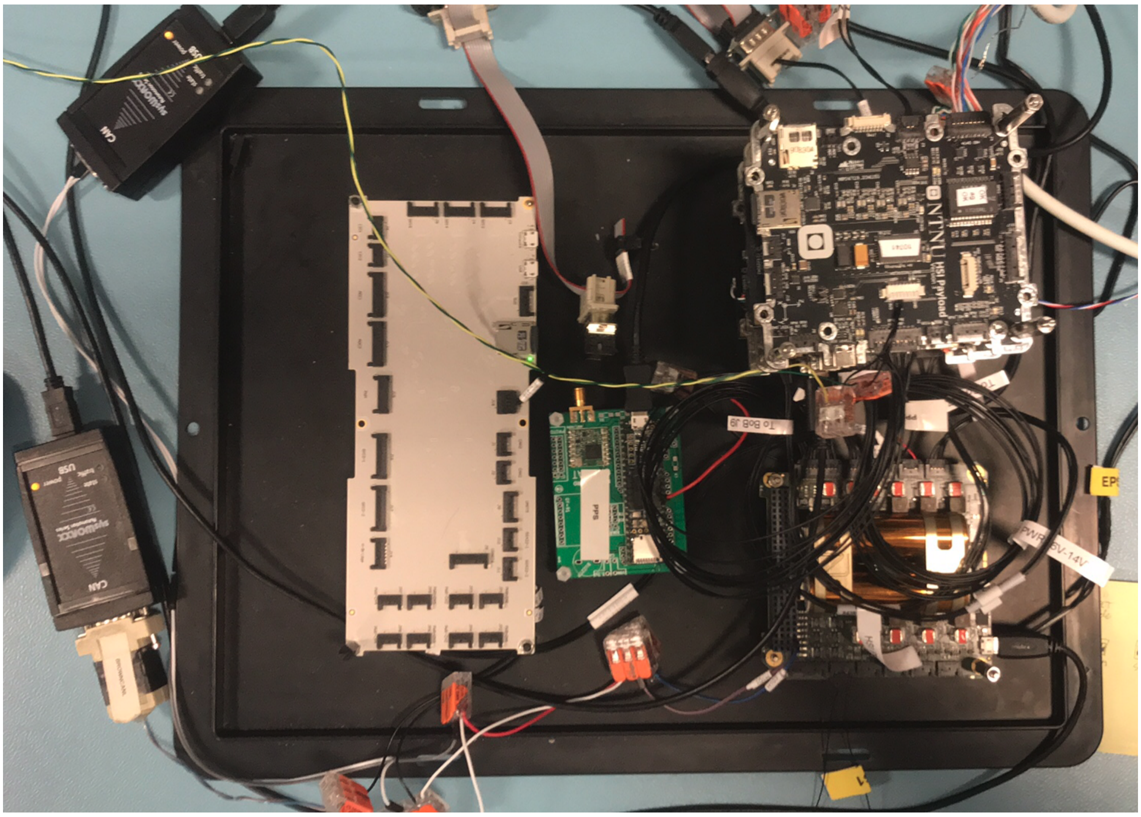


Figure 2.9: Lab setup

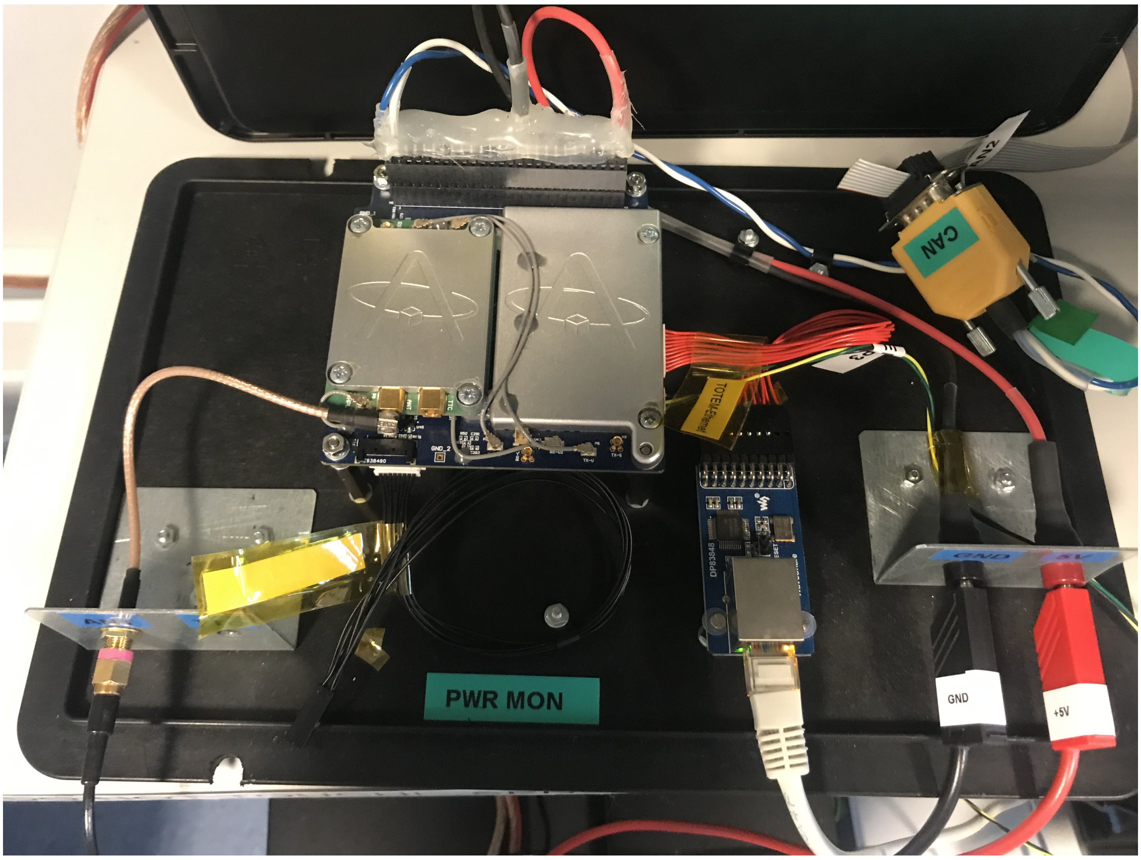


Figure 2.10: Lab setup, SDR

2.5.1 Electrostatic Discharge (ESD)

Both the SDR, the BoB and the PicoZed/UltraZed must be protected against Electrostatic Discharge (ESD). Thus, a part of the SmallSatLab is ESD secure, and people handling equipment sensitive to ESD must do so in this area. All instruments pictured above are located inside the ESD secure zone of the lab. In addition to this, the operator must also use ESD protected gear including a coat, a bracelet and shoes.

2.5.2 Remote Work

COVID-19 has made home-office more common. To be able to continue testing in the lab while working from home, VPN and SSH have been crucial. A lot of the lab equipment must be ESD protected as mentioned in Section 2.2, thus it is preferable to reduce the amount of physical contact with the lab setup and SSH is used in the lab to access the computers inside the ESD secure area. Thus SSH is also used in the lab when testing software

VPN is short for Virtual Private Network, and allows the user to create a secure connection to another network. This has been used to connect to the network at NTNU to further be able to connect to the computers at the SmallSat lab using SSH.

SSH is short for Secure Shell, and allows the user to access the terminal shell on the target computer. When using SSH, you must know the IP address of the target computer. To prevent the computers in the lab from acquiring new IP addresses on reboot, their IP addresses are static. Hence, the IP address of any computer in the lab is always known.

Chapter 3

Building a Flexible Software Architecture

There are multiple ways of developing a system. A traditional system life cycle is a linear sequence of stages like the waterfall model [23]. In comparison, the agile system life cycle can exist in multiple stages at the same time, and studies have shown that agile systems provide sufficient and necessary conditions for a system to have adaptability and flexibility [24]. This can be a benefit when the stakeholders of a project wish to change their demands, or the requirements of the project are unknown in the beginning. When developing scientific code in Software projects where the requirements are changing frequently or are unknown at the beginning of the project, an agile approach is preferred compared to a traditional approach [25]. The HYPSO team has chosen the Scrum methodology as the project manager had familiarity with using it from earlier projects in SW and HW for developing agile products. Further, a few of the students had used it before as well. This type of system development is desired as the requirements and feature demands of the project are changing a lot. Sletholt et al. [26] performed a literature review based on 35 different agile practices leading to their findings of the agile practices such as Scrum being well fit for activities related to testing. This holds for the HYPSO project as it is a technology demonstrator which demands a lot of testing.

In the context of systems engineering, Agile can be defined as "Adaptability and sustainment of adaptability" [24]. A system benefits from performing good in unpredictable and uncertain environments, and the agile systems-capability is supposed to address risky, unpredictable, uncertain and variable system environments, and adapt to them. Hence, the use of agile methodologies in software often focuses on the ability to react to changes requested from customers as well as a changing environment [27]. Using methods of agile development in software increased in frequency in the end of the 1990s, leading up to the publishing of the *"Manifesto for Agile Software Development"* [28] in 2001, featuring four concepts and twelve backup principles. The manifesto focuses on the importance of being able to respond to a change rather than following an exact plan, cooperate with the customers over negotiate on a contract, having a software which works instead of comprehensive documentation, and interaction with individuals over tools and processes [28].

Agile systems are designed to accommodate for structural change [24]. Hence, they can be changed in ways such as restructuring with regards to internal relationships, up- or down-scaling an augmentation to fit to a changed environment. The methods used in this master's thesis for agile development and design of an agile architecture will be explained in Chapter 4 and connected to the theory on agile systems and architecture presented in this chapter.

3.1 Agile System Life Cycle

Whereas a traditional system life cycle from concept to disposal, consists of a linear sequence of stages, where the stages are non-repeating, the agile life cycle works differently [23]. One example of a traditional life cycle is the waterfall model which is illustrated in Figure 3.1.

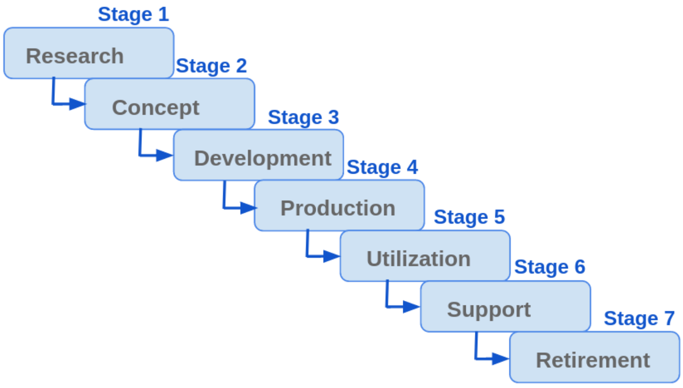


Figure 3.1: Classic waterfall model

An agile system differs from this model because it exists in a dual state where it is both in the state of development, and in the state of utilization, at the same time [23]. From the definition, the agile systems engineering process is capable of responding to an environment as it changes. In Dove and LaBarge, a sequential system engineering maturity transition is described [23], where each of the stages is a combination of all the previous stages, running concurrently as pictured in Figure 3.2. In order to achieve this, it is important that the architecture of the system facilitates change which is justified throughout the development in addition to subsequent support stages and utilization. The production stage will be the first stage in the agile development life cycle which puts a working product into the user environment. Within the next two stages; utilization and support, subsequent increments and development iterations take place [24]. The SW management process, Scrum, is connected to this type of agile development. This is the planning-tool used of the SW team of the HYPSO-project and will be explained in Section 4.2.

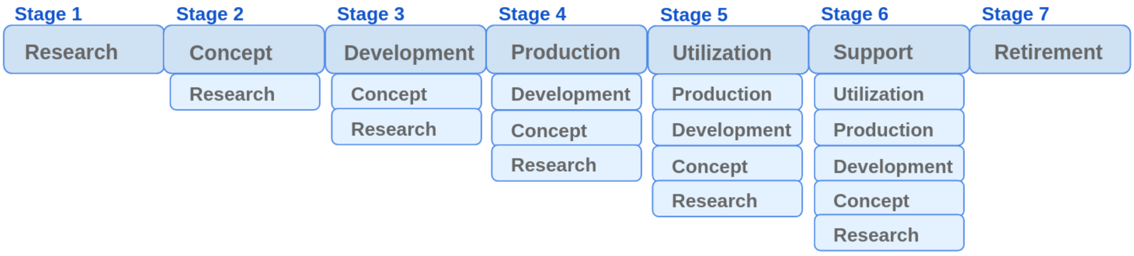


Figure 3.2: Agile system engineering lifecycle framework, figure inspired by [23]

As for the HYPSO life cycle, the System and its modules are subject to continuous development. The HYPSO project is a technology demonstrator, and the purpose of developing the satellite is to contribute to new knowledge rather than mass-production to earn money. A typical scientific software development project continues as new projects researching different objects, perhaps with different developers wanting to change the functionality [26]. This is the case when expanding the HYPSO mission to include the SDR, and the technology will likely be used in different satellite projects at NTNU in the future. In addition to this, the team changes every semester because it mainly consists of master students. New ideas come with new developers contributing to the team. As a result of the above, being able to change the requirements and add new functionality is crucial for the project. When requirements change, or a new kind of performance is requested, this might change the whole architecture. A traditional life cycle with a pre-defined end result through a negotiated contract would not give the team the same possibilities of changing the system or software in the desired ways, and the agile life cycle is preferred as planning of scientific SW development projects can be challenging.

3.2 Agile Software development

The agile workflow described in Section 3.1 facilitates for making changes in the code base of the SW as the requirements of the project changes. In order to make those changes efficient, an agile SW architecture is the desired goal of the code base. In Dove and LaBarge, a foundation of fundamentals for agile systems creation is provided [24]. The research done includes empirical studies and discoveries regarding man-made systems including enterprise processes, manufacturing processes, HW systems and SW systems. They have concluded that an agile architectural pattern consists of three important elements: *drag-and-drop* of different encapsulated *modules*, *plug-and-play* interconnection enabled by a *passive infrastructure*, and an *active infrastructure* [24]. It is claimed that if the architecture includes these elements, it improves the systems adaptability and flexibility.

A *module* can be defined as a self-contained encapsulated unit having interfaces which are defined precisely and complies with the *plug-and-play* passive infrastructure. This simplifies the *drag-and-drop* in the system and connection to other modules [24]. Having a system containing encapsulated modules is essential to have their functionality and methods independent of other modules. To achieve this in SW, careful planning and clear definitions of modules are key factors. A clear definition will be beneficial when deciding what functionality to put inside of a module, and ensuring that every part of the code is in the right place. When defining a module, the SW developer needs to decide what is supposed to go inside of it, what functionality the module should offer, and what the module should provide for the rest of the SW system. In the context of this master thesis, modularity is important between the different modules in the SW architecture. A change or an improvement could be made rapidly to the implementation or internal design of a subsystem when the change is not impacting its external function, fit or form. When developing a system, it is desired with an effective replacement of modules and the ability to make internal changes without side effects. This is also a central point when planning for further SW development using issues in GitHub, which will be explained in Section 4.3.1, and during the scrum meetings where development is planned, which will be explained in Section 4.2.

Modularity is also a factor when the different teams are working together on the satellite, each contributing with their speciality, and the HW and SW teams are connecting the whole system and its different functionalities together. Once a requirement is defined by another team, the SW team can deliver incremental capabilities to meet the requirements as soon as possible. The modules in `hypso-sw` are clearly defined, and an important part of changing the code base to integrate the SDR has been to put the new code into fitting modules as well as defining new modules wherever code has been changed. This method is called refactoring and is heavily included in the agile development, due to the common step of changing code without changing semantics [29]. This is often done to prepare the code for development of new components or improve its quality and can be connected to. One example is that it is better to divide the functionality into more files than having a lot of different functionality in one place. When the module has a specified functionality, input and output, it will not affect other modules in unpredictable ways. This results in a good overview of the code-base, and easier error detection.

The *drag-and-drop* connection between the modules is provided by the *passive infrastructure.* It is ensuring that the different encapsulated modules are isolated, resulting in less side effects and easier implementation of new functionality [24]. These connections are related to Section 2.3 regarding network communication. The connection of the different modules within `hypso-sw`, which is the part of the HYPSO project the author has worked with, is done through CSP packets. The implementation of these connections were already finalized by another student to enable for the different modules in the SW to be plug compatible, before the author started writing this master thesis. Clear interfaces are beneficial for replacement and integration of different modules within the system. Clear interfaces is also in mind when the author is defining inputs and outputs of different functions in order to make them generalized.

The *active infrastructure* is an important element in an agile system as new requirements will result in new system configurations. For enabling of new configurations it is important to ensure that the existing modules are upgraded, insufficient modules are removed, and new modules are added [24]. An important part of the integration of the SDR into the existing SW has been to keep track of and change the modules when it seemed beneficial to the author. Upgrading of modules can include reusing modules or parts of modules instead of writing the same code over again, which is connected to the modularization and refactoring of code. In this master, typical reuse consists of refactoring code to be able to reuse functionality which is similar for the different payloads. Another part of the active infrastructure, is always having the sufficient modules ready for deployment. This is connected to the agile building block of continuous integration which is crucial for the reason of not breaking the functionality which already works with the new changes or implementations. The focus here is on uploading the source code which is being developed regularly into a shared repository within the team. This is done to facilitate for automated building of the code in addition to automated quality control tools and tests made for detecting issues early [29]. The SW team in the HYPSO project are working in such a way that the master branch of `hypso-sw` is always ready for deployment. No new functionality is added to this part of the SW before team members have tested and approved the code to work. This method of working is connected to the agile life cycle and will be explained in Section 4.3.

These fundamentals are appropriate with regards to the best practices for scientific SW development according to Wilson et al. [30] which are stating that they include: writing programs which are understandable for the people which are going to read them, re-use code when it is possible, changes should be made incremental, make modular code over copy pasting, use version control, refactoring code rather than explaining the functionality, and collaboration with code reviews. This type of agile design makes it easier to change the system. Different types of structural changes could be adding functionality, restructuring of internal subsystems, scaling or reshaping in order to make the system compatible with a different environment. To enable for these types of changes, an architecture which accommodates structural change is required. This architecture will consist of many different connected components interacting with each other in specific ways. Using these types of principles in SW development will also make a code base which is an easy subject to change. This is a benefit when developing code for a satellite where the demands can change suddenly. Dividing the code into modules with clear interfaces will make it easy to find the part of the code which has to be changed to make the system function as desired. Furthermore, it will simplify the job of changing different parts of the code without it affecting the rest of the system when the inputs and outputs of functions and modules are clear. With regards to these fundamentals, the main focus of this master's thesis will be the *module* element and maintaining an *active* infrastructure. A lot of the modules had to be changed in order to integrate the SDR into the code-base, in addition to adding new modules, and deleting the ones not needed anymore after changes were made.

3.3 Agility in the HYPSO project

Agility is used in multiple ways in the HYPSO project, and the two most relevant ways with regards to this thesis are the agile life cycle interpreted through scrum and the implementation of modular and reusable code in order to make the SW agile. The cycle of development used in the project can be seen in Figure 3.3. As can be seen there are several steps which together make up the workflow which is supported by the Scrum Method (Section 4.2) and the GitHub workflow (Section 4.3). This cycle is repeated every two weeks with planning with the whole team, deciding what is supposed to be the focus in the upcoming weeks. In the same meeting, an analysis is done on the past two weeks. When the tasks have been set for the week, every team member performs its tasks with design before implementation. Then coding and debugging will be performed, before it is tested in the lab. When the team member believes that the development is sufficient, other team members will test and review the code. If every test is working, and the code is reasonably written, the newly developed code will be approved to be integrated to the master branch of the `hypso-sw` repository. In this way, the master branch is never updated before the features are ready, and hence always working unless bugs are not discovered when testing new code. Iterative development and continuous integration are results of this workflow.

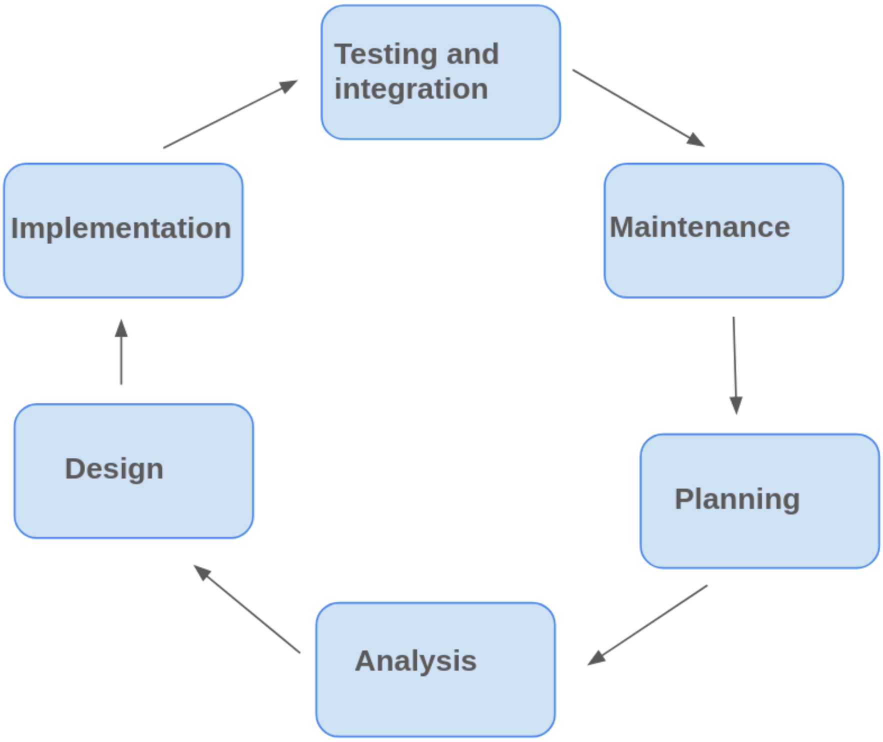


Figure 3.3: Development cycle in the HYPSO project

Chapter 4

Methods and Tools

This chapter will focus on describing the different methods and tools that have been used when designing and developing code for the SmallSat lab.

4.1 Software Development

This section is based on section 4.1 in the author's specialization project report [1]. Improved Software Development is important for a project. As McConnel states in *Code Complete* [31], *construction*(*coding* and *debugging*) is the central activity in software development, hence it is the only activity that is guaranteed to be done in every project. However, for efficient construction, it is beneficial to *plan* the project by first defining *requirements* and *architecture* well. The planning in this project has been done using GitHub, and the process will be described in Section 4.3. After construction, the system *testing* is done to verify correct functionality of the system. There are different parts of testing as well; *unit testing* - on each module, *integration testing* - module with the rest of the program, and *system testing* of the whole system or program.

Prior to writing code for this master's thesis, there was done a lot of planning to ensure that the desired functionality would be obtained. And maybe more importantly, that a desired architecture would be obtained. When making the designs for the architecture, the agile principles from Chapter 3 were taken into account, with a main focus on making generalized and modular code. A lot of the code was refactored to generalize the functionality which could be re-used with the SDR. Thus, the generalized code could be used for both PLs instead making a copy of the code developed for the OPU and customize it to fit the SDR. After developing the code, there was performed testing on the requirements from the design to verify that the code worked as intended. In-between the two, code was developed and debugged. In this section, some important aspects of code quality will be listed, as well as a description of the development style used in the HYPSO project; Agile Software Development.

4.1.1 Code Quality

There are two types of software quality characteristics mentioned in the book *Code Complete* [31]. External characteristics are the ones which the user care about, typically how easy it is for them to use the software. From a programmers point of view, it is rather important how easy it is to modify the code. As for the HYPSO project it is especially important that the code written is understandable and use-able by others, as there are a lot of different people working together to develop the SW. In addition to this, new programmers are on-boarding the project frequently, and it is very helpful for them if the code is of great quality. Hence, the focus here (as in the book) will be on the internal quality characteristics, as the main focus has been writing code of quality which will also be easy to modify for others. The following characteristics are listed in the book, and are some of the things which has been considered by the author while writing code for the HYPSO project:

- **Maintainability:** The ease with which the code can be changed or edited to add functionality. Here, the focus has typically been to ensure that the code written is encapsulated and abstracted such as the different network layers in Section 2.3. A developer can create a function in `hypso-cli` to send a message to one of the services in `opu-services` through a CSP-connection, without really understanding the implementation of the code related to CSP.
- **Reusability:** The extent to which you can use parts of the code in other parts of the program. An great part of the work done related to this master's thesis has had this focus. Several functions have been re-written in order to use the same generalized functions for any payload. Designing an agile SW architecture as expla
- **Readability:** The ease at which the source code can be read and understood. A way of ensuring readability, is creating names for functions and variables which state exactly what they do or what they represent. In cases where representative names is not enough for the code to be readable, comments can be a necessary tool for documentation. For example, it can be useful for others to know where the value of a constant comes from. The formatting of the code is important as well. In this project, all code is formatted with clang, which is explained in Section 4.1.1.
- **Testability:** Make it possible to test if the functions implemented actually work with a standardized unit test. All developed functionality has been thoroughly tested throughout the semester, and the return-values of the function will indicate whether or not it has passed a test. Hence, having return values which describe the outcome of a function is desirable

Clang formatting

The clang-format is used by the software team in the HYPSO project to format the C code to tidy up the code so that everything is written with similar formatting. It has a few different use cases [32]: reformatting code to the kernel style, finding mistakes in the style, as well as helping the user following the rules of the coding style. The clang-format is applied after adding code to one of the repositories, before the code is submitted.

4.1.2 Agile development

The software group in the HYPSO project uses an agile approach for the Software Development Life Cycle (SDLC). The agile approach differs from the traditional methodologies where the success relies on knowing all the requirements before the development begins [33]. For the agile approach, the development is rather incremental and iterative. The traditional methods define requirements before designing and planning takes place, then the system is constructed before testing is performed. Instead of having one large process model, the agile approach will divide the project into different smaller parts, and will then focus on one of these increments at a time. This approach is used in the HYPSO project due to a few of its advantages listed in the article [33]: it is suitable for small projects, the testing is done on every iteration, the rework cost is low, the development direction is readily changeable and the rework cost is low. The fact that the master students on the team change every year is also a reason fur using this approach. The framework used in the HYPSO project to implement Agile development is called *Scrum*, and is described in Section 4.2.

4.2 Scrum

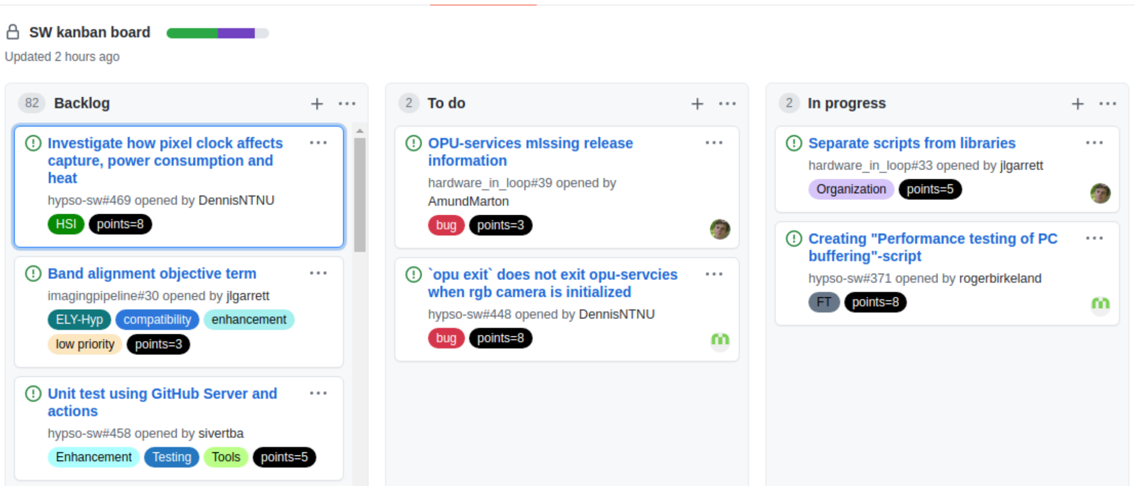
The following section is based on Section 4.2 of the author's specialization project report [1]. Scrum is well suited for the HYPSO project as the team is working on different parts of a complex product and this is a framework that implements the agile methods explained in Section 4.1.2 [34]. This is a tool that helps dividing up the functionality needed for the software, prioritizing the tasks, and delegate the tasks between the team members. It can be a helpful tool or development process to achieve a higher level of transparency within the team.

The product is delivered incrementally through a series of short development phases called *sprints* [34]. In the software group in HYPSO, a sprint lasts for two weeks and is followed by a review combined with planning of the next sprint. Frequent smaller meetings are held during the sprint so that everyone involved are updated on each others progress, and can discuss problems if they emerge. The main meeting for the software group is held for an hour every Tuesday. The whole HYPSO team also has a meeting called *stand up* every weekday, where every team member is invited to update the others on what they are currently working on.

In each sprint *planning*, tasks are delegated to the different members of the team. These tasks are predefined in issues in GitHub, these issues will be explained in Section 4.3. Here, the tasks are explained thoroughly so that they should be possible to complete without further information. Each issue will be assigned a proper amount of points to estimate its workload. When the sprint is planned, the issues are divided into the following columns in a board called Kanban, and the explanations are a result of the authors experience working with it:

- **Backlog:** All currently identified issues.
- **To Do:** Issues delegated for the current sprint.
- **In progress:** Issues that have been started, but are not quite finished yet.
- **Review in progress:** After completing a PR (Section 4.3.4), the issue is moved here.
- **Done:** The issue is moved to this column when the pull request it is related to is approved.
- **Blocked:** Issues that can not yet be completed, because it relies on another issue to be completed first.

Figure 4.1 and Figure 4.2 are together an example of what this Kanban board looks like.



SW kanban board

Updated 2 hours ago

82 Backlog

Investigate how pixel clock affects capture, power consumption and heat

hypso-sw#469 opened by DennisNTNU

HSI points=8

Band alignment objective term

imagingpipeline#30 opened by jlgarrett

ELY-Hyp compatibility enhancement low priority points=3

Unit test using GitHub Server and actions

hypso-sw#458 opened by sivertba

Enhancement Testing Tools points=5

2 To do

OPU-services mIssing release information

hardware_in_loop#39 opened by AmundMarton

bug points=3

\`opu exit\` does not exit opu-servcies when rgb camera is initialized

hypso-sw#448 opened by DennisNTNU

bug points=8

2 In progress

Separate scripts from libraries

hardware_in_loop#33 opened by jlgarrett

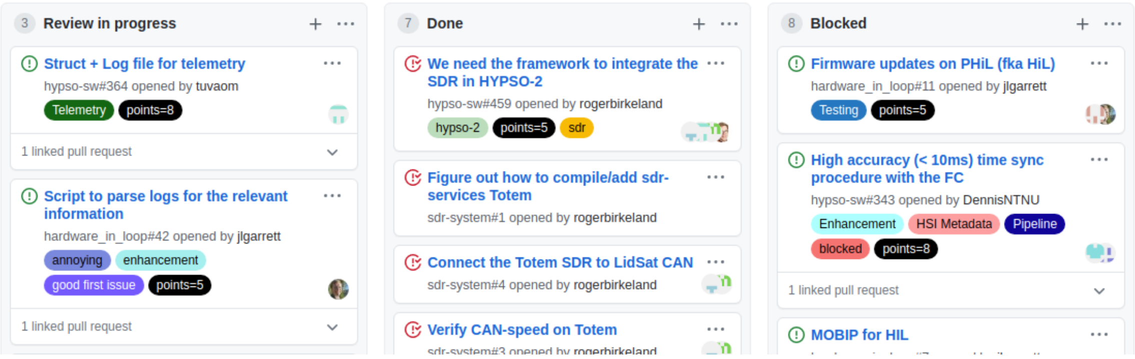
Organization points=5

Creating "Performance testing of PC buffering"-script

hypso-sw#371 opened by rogerbirkeland

FT points=8

Figure 4.1: Kanban: backlog, to do in progress



3 Review in progress + ···

Struct + Log file for telemetry ···
hypso-sw#364 opened by tuvaom
Telemetry points=8

1 linked pull request

Script to parse logs for the relevant information ···
hardware_in_loop#42 opened by jlgarrett
annoying enhancement
good first issue points=5

1 linked pull request

7 Done + ···

We need the framework to integrate the SDR in HYPSO-2 ···
hypso-sw#459 opened by rogerbirkeland
hypso-2 points=5 sdr

Figure out how to compile/add sdr-services Totem ···
sdr-system#1 opened by rogerbirkeland

Connect the Totem SDR to LidSat CAN ···
sdr-system#4 opened by rogerbirkeland

Verify CAN-speed on Totem ···
sdr-system#3 opened by rogerbirkeland

8 Blocked + ···

Firmware updates on PHiL (fka HiL) ···
hardware_in_loop#11 opened by jlgarrett
Testing points=5

High accuracy (< 10ms) time sync procedure with the FC ···
hypso-sw#343 opened by DennisNTNU
Enhancement HSI Metadata Pipeline
blocked points=8

1 linked pull request

MOBIP for HIL ···

Figure 4.2: Kanban: review in progress, done, blocked

In the *review* part of the sprint, the progress of the last two weeks will be discussed. Both the issues that are done, and the ones that weren't completed and why. A track of the scores will be kept so that one can see the difference between estimated work, and work that was actually completed in each sprint.

4.3 Git

Except from section Section 4.3.7, this section is a continuation of section 4.3 of the specialization project report [1] written by the author. To keep track of the history of different versions of the software while developing different functionality, the free and open source Version Control System (VCS) Git is used. As explained in the book *Pro Git*[35], Git performs version control of a file system called *repository* through snapshots called *commits*. This can be done both *locally* on a computer or *remote* on a server.

The remote service GitHub is used in the HYPSO project for online hosting of Git repositories. This is only one out of several similar services, and it is not associated directly with Git. One of the additional services GitHub provides is improved collaboration by enabling for *teams* and *organizations* to work together.

The GitHub workflow is branch-based[36], where the master branch is not affected by another until requested changes are reviewed by a team member. The following sub-sections are based on this website[36], and images used was also found there.

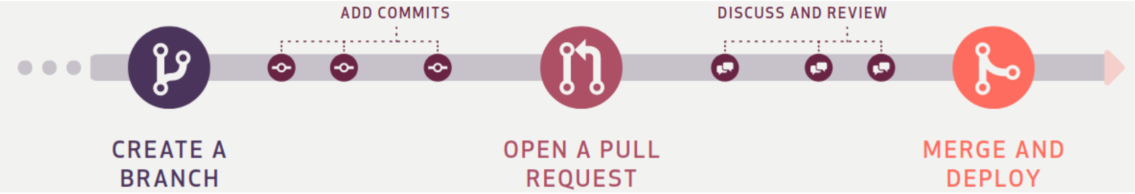


Figure 4.3: The GitHub Workflow[36]

4.3.1 Issues

The GitHub workflow solves bugs or features requested for the code base. A way of keeping track of the problems team members encounters is through issues. These are written descriptions of bugs, features or other code-related matter. All team members can generate new issues as they arise, and comment on the issues of others. In the HYPSO software team, Scrum is used to keep track of issues. This framework is described in Section 4.2.

4.3.2 Branch

An issue often results in a new branch made to solve the bug or feature request. The new branch will be a copy of the master branch. This is done to experiment with new ideas without affecting the master branch until the new branch is sufficiently developed and tested so that it is ready to be a part of the core functionality of the project. The construction of a new branch is illustrated in Figure 4.4. The name of the branch should be descriptive and give a hint of which issue it is solving.

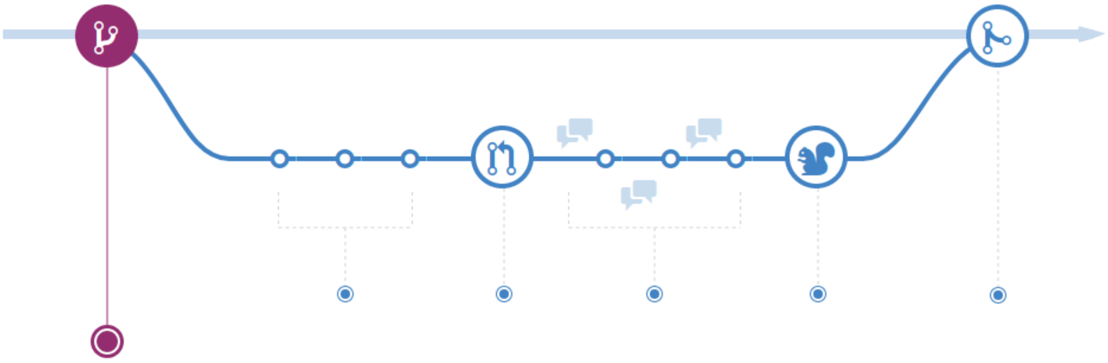


Figure 4.4: Creating a branch[36]

4.3.3 Commit

After the branch is created, changes made is tracked by commits(Figure 4.5 made to the branch, hence tracking the progress. This history will also let team members look at the history of each others branches. A message associated with each commit is made to describe why a specific change was made. Each commit also has a hash value which will let you erase your changes if for example a bug is detected. Commits are *pushed* to move local changes to the remote server. A developer cannot commit changes directly to the master branch of `hypos-sw` or `opu-services`, this is a way of trying to keep the main repositories free of errors and bugs.

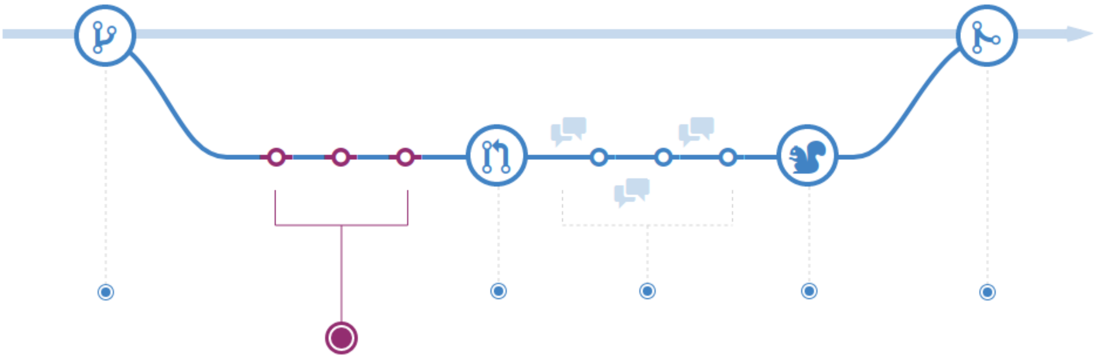


Figure 4.5: Committing[36]

4.3.4 Pull Request

A Pull Request (PR) is opened to initiate a discussion with the rest of the team about the changes made to a branch. This is typically done when the developer consider the feature to be working as intended, and wants the fellow team members to review the changes. Great practice is to describe the changes made, and how to test if they work as intended to ease the workload of the team members that are going to review the changes. If the changes are related to an issue, this should also be linked up to the PR. Particular team members can be requested to review the code. In Figure 4.6, a PR is highlighted.

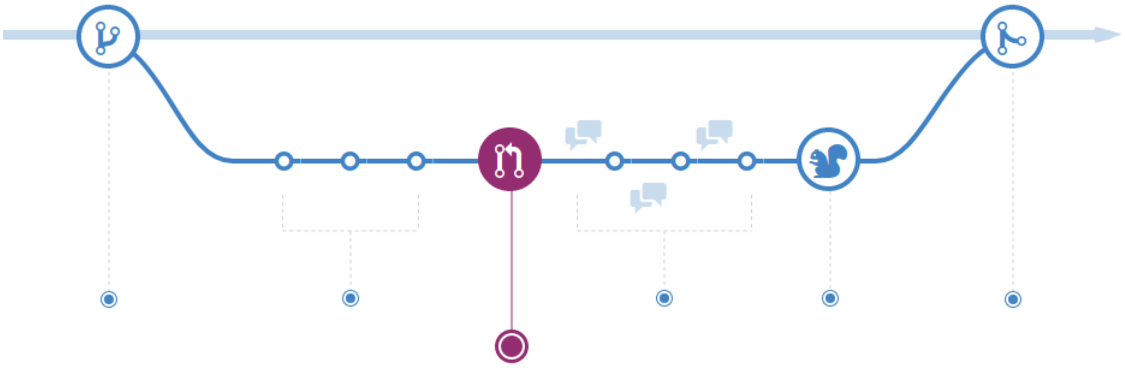


Figure 4.6: Pull Request[36]

4.3.5 Review

Some PRs are approved right away, others may take time either to test or due to team members requesting changes to the functionality or the coding-style. The reviewers task is to test the functionality of the new features of the branch as well as ensuring that the rest of the system still works as intended when the new branch is integrated. Once the PR is created, commits can still be made and pushed reflecting the feedback from the review of other team members. The new commits will be showed in the PR in GitHub together with the comments from the reviewers.

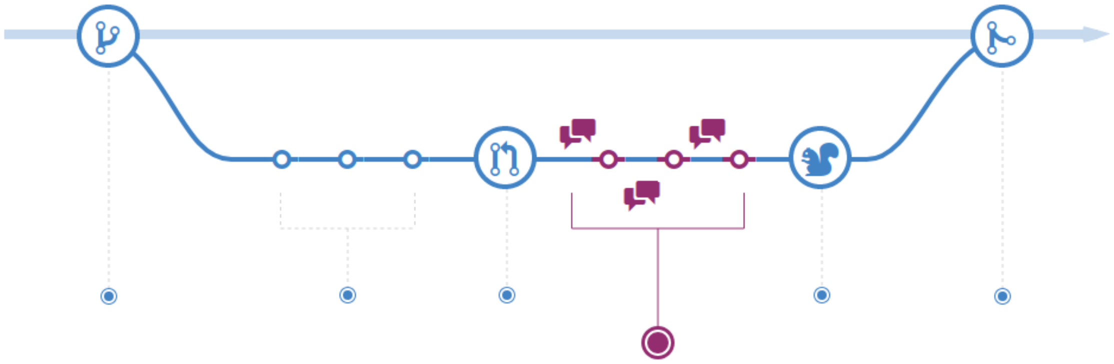


Figure 4.7: Review[36]

4.3.6 Merge

When the changes in a branch is accepted by at least one other team member, it is ready to be merged into the master branch by the developer making the PR. This is the final step of the GitHub workflow, and is pictured in Figure 4.8

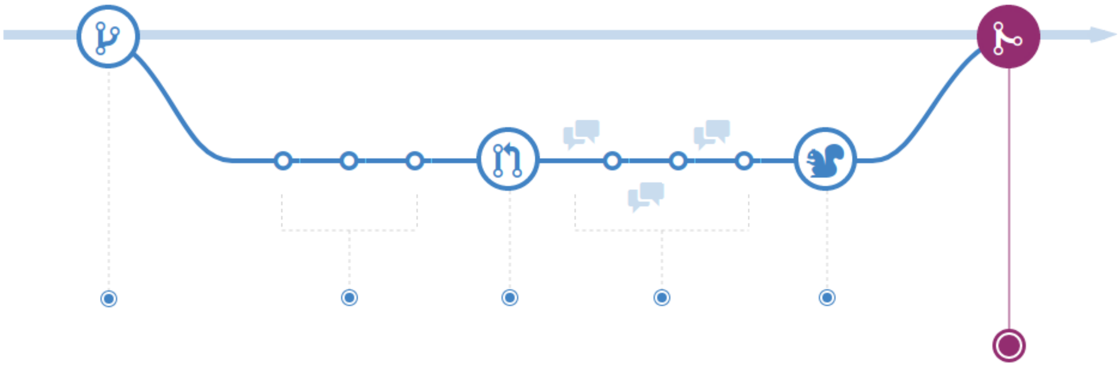


Figure 4.8: Merge[36]

4.3.7 Development on another branch than master

A special method used while developing code for this thesis, has been to branch out from a different branch than the master branch of `hypso-sw`. As the code for HYPSO-1 is only meant for the OPU, the team decided to rather make another branch for development of code for the SDR. This branch was called `sdr-services`, and an important step of updating this branch is to always merge the newest commits added to the master branch into it. When a new feature or update is requested for the integration of the SDR into the code-base, a new branch is made out of the `sdr-services` branch. When the development of the new issue is completed, a merge is requested into the `sdr-services` branch with a PR, and it is reviewed by another team member just like a normal PR. If the PR is approved, it can be merged into the `sdr-services` branch, and the development branch can be safely deleted.

Chapter 5

Work and Results

Most of the contribution in this thesis has been related to designing how to integrate the SDR into the existing SW of the HYPSO project in a way that is efficient and relating to the principles listed in Section 4.1.1 and the agile fundamentals described in Section 3.2. This way, the author made a flexible framework supporting a satellite with multiple payloads. The code-base is developed in a way so it can be further re-used for future projects and satellites too. The results of this work will serve as an example of how to implement a flexible SW architecture which works with multiple PLs. As explained in Section 4.2, sprints are used to implement Agile development in the project. During the first eight sprints this semester consisted of, different functionality has been developed and merged into the `sdr-services` branch of `hypso-sw`, resulting in the current version of the SW related to the SDR, as the code-base is under continuous development and integration. There has also been developed some improved functionality on previous work (Section 5.1). In Table 5.1, the start- and end-date of each sprint is listed together with the main focus of the author during the corresponding sprint. The other participants in the SW group of the HYPSO project had different main focuses throughout each sprint. This chapter will feature the author's contributions, both the development process of the different functionality as well as the planning of design and different options. As described in chapter 4, the main methods used for developing the code for the project are:

1. Planning: planning of the development has been a combination of the author designing solutions for further development and discussions with the team in issues in Github and during the bi-weekly sprint-meetings.
2. Construction (coding and debugging): making modular and generalized code of quality through agile development.
    - Making sure the code developed is: maintainable, reusable, readable and testable.
    - Agile development through sprints.
3. Testing using the setup explained in Section 2.5.

Sprint#	Start	End	Author's contribution
1	January 14^{th}	January 28^{th}	Finalize Telemetry Service
2	January 28^{th}	February 11^{th}	How to mirror opu-commands
3	February 11^{th}	February 25^{th}	Implement sdr-commands
4	February 25^{th}	March 11^{th}	Telemetry service for the SDR
5	March 11^{th}	March 25^{th}	User access on SDR
6	March 25^{th}	April 13^{th}	Refactor code
7	April 13^{th}	April 29^{nd}	Telemetry service for the SDR
8	April 29^{th}	May 20^{th}	Generalize functionality & logging ADC values
9	May 20^{th}	June 7^{th}	Finalize Master's Thesis

Table 5.1: Sprints

Each sprint will be featured in its own section with subsections describing the different issues and PRs related to the authors contribution within the corresponding period. These issues and PRs will be listed in tables. Some of the titles were made shorter to fit the tables, but the essence is the same. The actual issues and PRs can be found in the appendixes linked in the tables with a varying degree of discussion between the team members of the project. With a few exceptions, all issues and PRs referred to in this thesis are located in the `hypso-sw` repository explained in Section 2.4.2. The relevant repository for the remaining ones will be referred to as the issue or PR is discussed.

5.1 Sprint 1 - Finalize Telemetry Service

The authors main contribution to the project during this sprint, was to finalize the telemetry service developed while working with the specialization project [1]. There were a few feature requests from the other team members which were still to be completed. No points were given to these changes, as no new issues were created. Most of the features were requested as comments in the final PR made as a part of the specialization project, and it can be seen in Appendix S.

As mentioned in Section 6.4, called Future Work, of the authors project thesis [1], there were a few remaining problems to solve in order to make the Telemetry Service work as desired. The TM logging had an impact on the performance of the rest of the system which was larger than acceptable during HSI capture. Hence, it was solved by turning off the logging before every capture, and turning it back on when the capture is done. The possibility to turn the logging on and off, and change the logging-interval through `hypso-cli` was also implemented. It was requested that the logging interval is saved to a file when changed so that the TM service can read this file upon boot to set the correct interval. Having the integration of the SDR in mind, the functionality was made general with specified inputs and outputs in order to make it work with any PL. After completing these changes in the `telemetry-service` branch of `hypso-sw`, the PR was tested by some of the other team members and approved. This resulted in the PR being merged into the master-branch of `hypso-sw`, and it is finally a part of the main software of the project. After months of development, the `telemetry-service` branch could be deleted.

5.2 Sprint 2 - How to mirror opu-commands

Other members of the team worked with the SDR over Christmas, and found a lot of necessities for `hypso-sw` in order to make it work as desired with the SDR. The issues the author worked with during this sprint (Table 5.2) were a result of this.

Issue	Title	Points	Linked PR	Appendix
#476	How to mirror opu-commands	5	-	Appendix E
#475	opu check does not work on SDR	5	-	Appendix D
#473	Verify CSP commands on sdr-services	3	-	Appendix C

Table 5.2: Issues worked with in sprint 2

5.2.1 How to mirror OPU commands

Attempting to solve these issues, the author started going through the `sdr-services` branch made in `hypso-sw` to get an overview of what was different from the original branch. The findings of this work, were a lot of similar code between the two PLs which could be refactored and generalized so that the similar functionality could be shared. The authors main focus during this sprint, was to plan the integration of the SDR into the already existing SW as requested in issue #476. Hence, no new PRs were created by the author during this sprint. The work rather resulted in more issues for the author to solve, these will be featured in Section 5.2.4. The main goal when making these issues was to describe a refactoring of code which would result in an agile architecture with modules and clear interfaces.

5.2.2 Making opu check work for the SDR

While going through the code compiled to create the `sdr-services`, the author found the source of the problem in issue #475. The issue reports that the command to compare a local and a remote checksum[1] does not work on the SDR even though the TOTEM has the `md5sum` tool, which is the tool used to find the checksum, implemented. This problem was a result of the TM service not being included in `sdr-services`, and `opu check` being a TM service command.

5.2.3 CSP commands for the SDR

Issue #473 should in theory be an easy issue to test and solve, as it simply is to verify that a few commands work. This turned out to be a bit more complicated due to two reasons. Firstly, the SDR was not yet connected to the EPS, meaning that testing the command `csp shutdown` and `csp reboot` would require someone to be present at the lab to flip the switch to turn the SDR back on. As the author, and most of the other team members, at this time worked from home due to COVID-19, this issue was hard to test. The other problem was the fact that whether these commands work, is dependent on which user the `sdr-services` is run from. They only work using the `root` user, and not the `totem` user which was the user the `sdr-services` was initially run from. Due to this, issue #473 was postponed to a later sprint.

5.2.4 Issues created

Due to the agile approach of the project, issues are made when they arise. Hence, a great deal of new issues were made as a result of this sprint as the design decisions had started. The new issues were created to describe the tasks to be fulfilled in order to make it possible to use the same commands for both PLs and hence making a flexible SW architecture for easier integration of future PLs. The issues created by the author during this sprint are listed in Table 5.3, and were given points by the team during the following sprint-meeting. One issue made by another team member, #492 is also listed in the table, as this was made after discussions with the author and is relevant for this thesis. This issue is concerning the problem with user permissions for the `sdr-services`. The other issues were made to refactor the already existing code which works for

[1] Checksum: sequence of numbers representing the file to check its integrity. If the checksum of two files match, the files are equal.

the OPU into more files to further develop the code to work with multiple payloads. By doing this, functionality which is common for all PLs will be separated from functionality specified for a specific PL and common functions can be reused instead of having two similar functions with only small tweaks to make them function with different PLs. Having shared functions will save future developers in the project for unnecessary maintenance of multiple functions.

Issue	Title	Points	Linked issue	Appendix
#492	Decide user for sdr-services	5	#476	Appendix H
#493	cli_opu =>cli_opu + cli_sdr + cli_pl	13	#476	Appendix I
#494	refactor tm_service.c	8	#476	Appendix I
#495	Find a way to list all files on the sdr	2	#476	Appendix J
#496	Make opu update include sdr-services	3	#476	Appendix K

Table 5.3: Issues created in sprint 2

5.3 Sprint 3 - Implementing sdr-commands

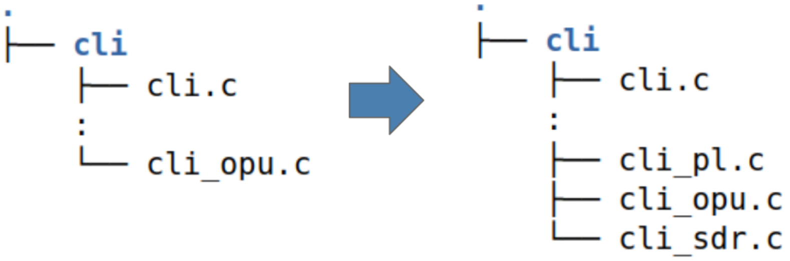
The main focus of this sprint was to make the OPU commands sent from `hypso-cli` available for any PL, which results in generalized code where functionality can be reused instead of implemented over again. In order to do this, wrapper functions which takes in the CSP address of the PL in addition to the other arguments of the different commands were made. The issues linked to this development can be found in Table 5.4. As mentioned in Section 4.3.7, integration of the SDR into `hypso-sw` is developed in its own branch, `sdr-services`, which is branched out from the master branch of `hypso-sw`. This is done to avoid interference with the development of the main SW until all functionality related to the SDR is working as desired. Hence, a new branch, `sdr-opumirror` was branched out from `sdr-serivces` to attempt to solve the issues in this sprint. The new branch will be requested merged into `sdr-services` instead of `master` as explained in Section 4.3.7.

Issue	Title	Points	Linked PR	Appendix
#474	make opu upload support the SDR	5	#506	Appendix D
#475	opu check does not work on SDR	5	#506	Appendix D
#493	cli_opu => cli_opu + cli_sdr + cli_pl	13	#506	Appendix I
#494	Split telemetry service	8	#506	Appendix I
#495	Find a way to list all files on the sdr	2	#506	Appendix J
#496	Make opu update include sdr-services	3	#506	Appendix K

Table 5.4: Issues worked with in sprint 3

5.3.1 Dividing `cli_opu.c`

Some of the previous OPU-commands sent from `hypso`-cli are useful both for the OPU and the SDR, but they are defined specifically to work with the OPU. To solve this, the solution described in issue #493 was chosen, resulting in one general PL-file with common commands, and one specific file for each PL, as seen in Figure 5.1. This provided reusability of the code as the functionality was generalized for both PLs, and modularity as the new files and functions had clear interfaces. Making common functions for the functionality to be used on both PLs also results in a code base which is easier to maintain because a change in functionality only has to be made in one place of the code.

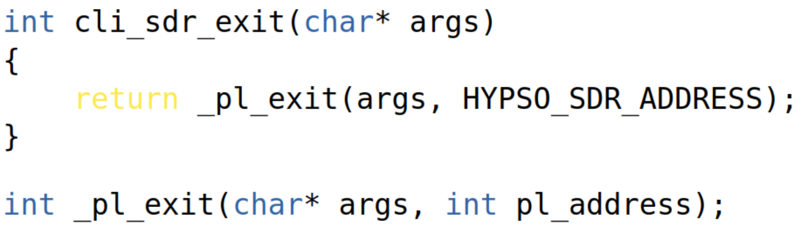


```
.
├── cli
    ├── cli.c
    :
    └── cli_opu.c
```

```
.
├── cli
    ├── cli.c
    :
    ├── cli_pl.c
    ├── cli_opu.c
    └── cli_sdr.c
```

Figure 5.1: Refactoring of cli-files

After some testing, it was discovered that some of the commands worked by starting the SDR with the CSP address of the OPU, while others would need some more modification. Hence, the first step was to change the previous `cli_opu*`-functions to `_pl_*`-functions which takes in the CSP address of the PL as an argument in addition to the others. These functions were placed in the new file `cli_pl.c`, while the OPU-specific functions remained in `cli_opu.c`. Wrapper functions for the OPU were also implemented, connected to the generalized PL-functions, and placed in this file. Similar wrapper functions were also made for the SDR and placed in the new file `cli_sdr.c`, this is also where SDR-specific commands will be placed in the future. An example of one of the wrapper functions can be seen in Figure 5.2 paired with the definition of the `_pl_*`-function from `cli_pl.c`.



```
int cli_sdr_exit(char* args)
{
    return _pl_exit(args, HYPSO_SDR_ADDRESS);
}

int _pl_exit(char* args, int pl_address);
```

Figure 5.2: Examplpe of wrapper function for PL command

Issue #474 was solved by making the SDR wrapper-functions and connect them to the generalized functions. Issue #495 was simply solved by passing different commands for the OPU and SDR from the `_pl_list`-function as the system command used for listing files on the OPU did not work on the SDR. Issue #496 was solved by adding checks for `sdr-services` to the `_pl_update`-function. Most of the previous OPU-commands are now PL-commands with wrappers for both the OPU and the SDR to make the functionality work for both PLs. In addition to this, wrappers were made in `cli_pl.c` for any PL with given CSP address. Adding a third payload would now be less demanding than before, as the wrapper-functions work the same for every payload, and the developer could just add the new PL-address. These changes made it possible for the user to send `sdr` commands, `opu` commands and `pl <address>` commands. Hence, `sdr exit` will perform the same job as `pl exit 13`, as 13 is the CSP address of the SDR. The commands which could be used for communication with the SDR after this are the following:

- `sdr exit`: Request the sdr-services process to exit.
- `sdr list`: List files in SDR's current directory.
- `sdr status`: Get status of SDR (simple telemetry).
- `sdr download`: Download a file from the SDR.
- `sdr upload`: Upload a file to the SDR.
- `sdr update`: Update `sdr-services` on the SDR.
- `sdr check`: Compare local and remote checksums.

- `sdr git`: Get branch and commit of sdr-services.
- `sdr lastcmd`: Request the last command received by one of the sdr services.
- `sdr telemetry`: Get current telemetry status from sdr-services.

There are still some functions that miss wrapper-functions for the SDR, and new issues featured in Section 5.3.4 are requesting the required changes for them to be implemented.

5.3.2 Split the telemetry service

To be able to solve #475, the TM-service had to be added as one of the service-threads of the SDR. This is done by adding the files of the TM service to the dependency list for the sdr-serices in CMakeLists.txt and adding the TM-service thread to sdr_services.c. The TM-service also had to be divided as requested in issue #494 as the values to be logged on the different PLs will differ from each other, and some of the TM commands are OPU-specific. The added files to this service were `src/tm/tm_opu.c`, and `src/tm/tm_sdr.c`. Nothing was added to `src/tm/tm_sdr.c` as the desired telemetry was still to be discussed. The HSI-related commands were moved to `src/tm/tm_opu.c`. Hence, the TM service was refactored as illustrated in Figure 5.3, and this solved #494.

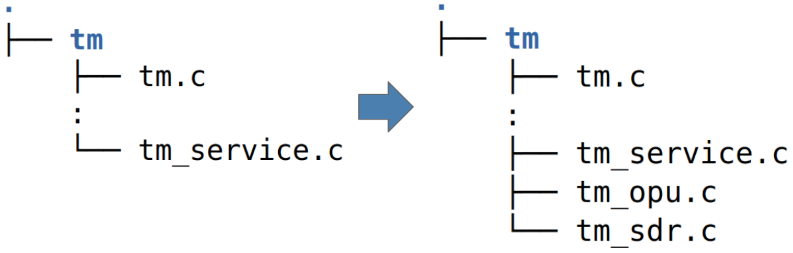


Figure 5.3: Refactoring of tm-files

5.3.3 Pull Requests

The PR in Table 5.5 is including all changes made in this sprint to make generalized functions able to send commands to any PL from hypso-cli. The PR requested the changes made in sdr-opumirror to be merged into the sdr-services branch of hypso-sw. The resulting code is easy to maintain, as the generalized functions can be used of any PL. Hence, making changes in the functionality would only be made in one place, and it would not affect the functionality of the functions in other modules. The PR was not reviewed within this sprint, therefore the merging had to wait.

Pull Request	∼ *Title*	Linked issues	Appendix
#506	Implementing sdr-commands	#474, #475, #493, #494, #495, #496	Appendix T

Table 5.5: Pull request of sprint 3

5.3.4 Issues created

When testing the new functionality on the SDR, a problem occurred with the TM logging. The running `sdr-services` shut down when the log file was to be created. Hence, the TM logging functionality had to be turned off. Issue #501 was created to resolve this problem, with the author suspecting that it is the format of the log file which is not supported on the `TOTEM`.

The remaining issues were made suggesting how to mirror the opu-commands which did not work as intended on the SDR, and hence required larger changes to work. This closed issue #476 as every part of the issue on how to mirror the OPU-commands was designed and explained in other issues, or already implemented. Issue #502, #503 and #504 are all dependent on the solution to issue #492 which is related to user permissions on the `TOTEM`. This resulted in the issues being blocked in the kanban-board. They are also linked to issue #476 of mirroring the OPU commands.

Issue	Title	Points	Linked issue	Appendix
#501	Telemetry service for the sdr	13	-	Appendix L
#502	sdr settime	5	#476, #492	Appendix M
#503	sdr shutdown	5	#476, #492	Appendix M
#504	sdr restart	3	#476, #492	Appendix M
#505	sdr log	3	#476	Appendix M

Table 5.6: Issues created in sprint 3

5.4 Sprint 4 - Telemetry service for the SDR

In this sprint, another team member connected the SDR to the EPS, meaning that the SDR now could be turned on and off remotely, making it easier to test both new functionality, and CSP commands that requires the SDR to reboot or shutdown.

There were made some requested changes to PR #506 by the other team members. These changes included one spelling mistake in the TM-service of the SDR and a request of returning other TM-variables from the command `sdr telemetry`. The author started working on these changes in the `sdr-opumirror`-branch, but unfortunately the team experienced some problems with the `TOTEM` as it was not possible to connect to it over ssh. This made it impossible to test the changes made, and the approval of the PR had to wait until the next sprint. In the end of the sprint, another team member managed to get the `TOTEM` working as before by connecting it to a UART[2] cable. The other issues in focus during this sprint are listed in Table 5.7.

Issue	Title	Points	Linked issue	Appendix
#501	Telemetry service for the SDR	13	#494	Appendix L
#505	sdr log	3	#476	Appendix M

Table 5.7: Issues worked with in sprint 4

5.4.1 SDR log

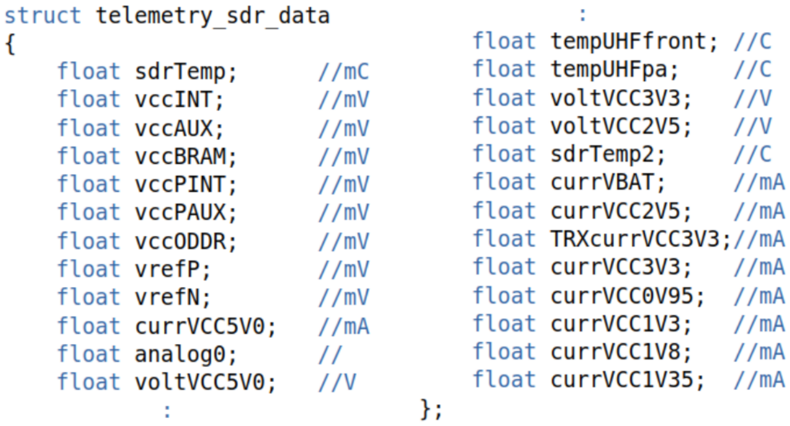
The author dealt with issue #505 before the `TOTEM` shut down. The `TOTEM` could not find the log-file `sdr log` requested because the log-file of the SDR is saved on a different location than the log-file of the OPU, therefore the previous command could not access the log-file as it addressed the wrong location. Logging on the SDR is a message log in a specific location which has now been added to the `_pl_log`-function. As a result, the message log from the SDR is displayed in `hypso-cli` when the command `sdr log` is sent. There was made a commit to include this functionality in PR #506.

5.4.2 Telemetry service for the SDR

As mentioned in Section 5.3.4, a problem occurred with the TM-logging when strarting the TM service from the SDR. The author started the work of integrating the TM-service in the SDR during this sprint. A new branch, `sdr-telemetry` was branched out from `sdr-services` for this

[2] Universal Asynchronous Receiver-Transmitter (UART): HW device used for asynchronous serial communication. The transmission speeds and data format can be configured [37]

development. A struct[3] was defined with a draft of variables desired for logging, this draft is pictured in Figure 5.4. As it was requested in PR #506 that the values from the `xadc_values`[4] command on the `TOTEM` would be displayed in the TM logging, these were the ones in the first draft on the struct. Development of functionality to fetch these variables was also started, but due to the issues with the `TOTEM`, testing could not be done, and the continuation of solving this issue had to wait until Section 5.7. The variables were saved as **floats** because they are all of different units, making it difficult not to use decimal numbers. In addition to this, some of the values can be negative. Hence, the number type **uint** which is used for the telemetry variables of the `opu`, would not be sufficient as it saves values without sign.



```
struct telemetry_sdr_data
{
    float sdrTemp;      //mC
    float vccINT;       //mV
    float vccAUX;       //mV
    float vccBRAM;      //mV
    float vccPINT;      //mV
    float vccPAUX;      //mV
    float vccODDR;      //mV
    float vrefP;        //mV
    float vrefN;        //mV
    float currVCC5V0;   //mA
    float analog0;      //
    float voltVCC5V0;   //V
            :
```

```
            :
    float tempUHFfront; //C
    float tempUHFpa;    //C
    float voltVCC3V3;   //V
    float voltVCC2V5;   //V
    float sdrTemp2;     //C
    float currVBAT;     //mA
    float currVCC2V5;   //mA
    float TRXcurrVCC3V3;//mA
    float currVCC3V3;   //mA
    float currVCC0V95;  //mA
    float currVCC1V3;   //mA
    float currVCC1V8;   //mA
    float currVCC1V35;  //mA
};
```

Figure 5.4: Initial TM struct for SDR with units of variables.

5.4.3 Issues created

There were made no new issues by the author during this sprint as no new problems occurred regarding the work on integrating the SDR.

5.5 Sprint 5 - User permissions for the SDR

Early in this sprint, the changes requested for PR #506 were completed and tested. This included changing the output of the command `sdr telemetry` as well as one minor spelling mistake. The PR was then tested again, resulting in an approval of the changes, and a merge into the `sdr-services` branch in `hypso-sw`. The author worked with a lot of issues(Table 5.8) during the rest of this sprint, as many of them were connected to issue #492, which was the main focus.

Issue	Title	Points	Linked issue	Appendix
#492	Decide which user sdr-services shall run under	5	-	Appendix H
#473	Verify CSP commands on sdr-services	3	#492	Appendix C
#502	sdr settime	5	#492	Appendix M
#503	sdr shutdown	5	#492	Appendix M
#504	sdr restart	3	#492	Appendix M
#471	Change handle_restart_request() for sdr-services	3	#504	Appendix B
#472	Consider csp shutdown for sdr-services	3	#492	Appendix B
#478	Update Totem time after reboot	3	#492	Appendix G

Table 5.8: Issues related to sprint #5

[3] Struct: short for structure. A structure in C is a user defined data type where variables of different data types can be put together[38]. This abbreviation will be used during the rest of the thesis

[4] currents and voltages of the `TOTEM`

5.5.1 User permissions

A great deal of the issues connected to the SDR were still dependent on which user `sdr-services` is run from. Thereby, there was held a meeting with some of the other team members discussing pros and cons of the different possibilities, as well as possible solutions. With the `sdr-services` run from `root`, it has all permissions, while it is more limited from the `totem` user. The optimal solution would be to create a new user with the permissions needed, but still not full access to every command available in the system. The team agreed upon that making this user would be a problem for future work, and that for development we would use the `root` user for testing. Hence, user permissions would no longer be an obstacle for further development and integration of the SDR.

After this decision, it was possible to access `time`, `can`, `shutdown`, `restart` and `tm-logging`. Hence, issue #502 was resolved instantly, as the problem simply was that the `totem` user could not access `time`.

5.5.2 SDR commands

When the user permissions were decided in Section 5.5.2 and the SDR was connected to the EPS in sprint #4 (Section 5.4), the commands `sdr shutdown` and `sdr restart` could be implemented. This was done by making two wrapper functions in `cli_sdr.c` connected to the generalized functions already implemented in `cli_pl.c`. In addition to this, a few lines had to be changed in `apps/sdr_services.c` in order to power off the SDR through the EPS when the commands were called from `hypso-cli`. A call to restart a given version of `sdr-services` also had to be added in order to make `sdr-restart` work resolving #471 which asked to change or delete the function for restarting `sdr-services` in order to make it work. As this functionality was connected to the mirroring of the OPU commands, the changes were made in the `sdr-opumirror` branch from Section 5.3. These changes resolved issues #503 and #504.

5.5.3 Verification of CSP commands

The CSP commands `csp reboot`, `csp shutdown` and `csp uptime` could also be confirmed when `sdr-services` was run from root and the SDR was connected to the EPS. Completing the functionality in `apps/sdr_services.c` for `sdr shutdown` and `sdr restart` as mentioned in Section 5.5.2 was necessarry to make `csp shutdown` and `csp reboot` work. The testing was done by sending the commands from `hypso-cli` followed by the CSP address of the SDR, 13, solving issue #473. Issue #472 was also resolved as it addressed the issue that `csp shutdown` could not be implemented for the SDR as at the time of the generation of the issue, the SDR was not yet connected to the EPS.

5.5.4 Update Totem time after reboot

As a part of the meeting with the other team members, a new startup script was added to the `totem`. The script will start `sdr-services` on boot and synchronize the `totem` time with the EPS. This resolves issue #478 requesting that the `totem` time is updated, as it was previously restarted after reboot.

5.5.5 Pull request

There was made a PR, referenced in Table 5.9 to summarize the decisions and changes made in this sprint. It requested the changes made in the branch `sdr-opumirror` to be merged into the branch `sdr-services`. The changes were tested and approved of one of the other team members within the period of this sprint, and merged by the author.

Pull Request	~ *Title*	Linked issues	Appendix
#520	Run sdr-services from root	#492, #502, #503, #504, #478	Appendix U

Table 5.9: Pull request of sprint 3

5.5.6 Issues created

Issue #521 was created by one of the other team members as a result of the new start-up script implemented, it was blocking the `reboot` and `shutdown` commands. This was resolved by the same team member in sprint #7, simply by requesting the service to start in the background with a `<&>`. The other new issue listed in Table 5.10 is addresses the use of port names in `hypso-sw` as the common ports for OPU and SDR should be called `PL_*_*PORT`, not `OPU_*_PORT` in order to not confuse the developers as the different PLs all use the same port numbers for their common services.

Issue	Title	Points	Linked issue	Appendix
#518	Change common ports to PL-ports	3	-	Appendix N
#521	SDR start-up script blocks commands	8	-	Appendix O

Table 5.10: Issues created in sprint 5

5.6 Sprint 6 - Refactoring code

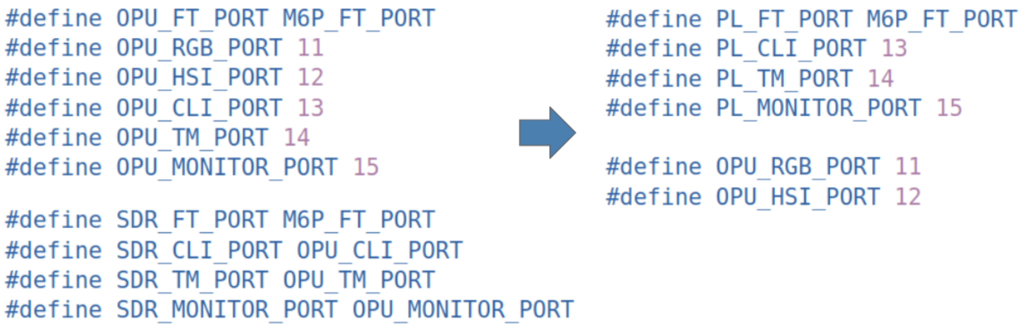
This sprint mainly consisted of refactoring `opu_services.c` and `sdr_services.c` in order to remove duplicate code as well as making the new functions general for any payload. The other issue in Table 5.11 consisted of renaming the ports in `hypso-sw`.

Issue	Title	Points	Linked PR	Appendix
#518	Change common ports for and to PL-ports	3	-	Appendix N
#477	Refactor common code between *_services.c	8	-	Appendix F

Table 5.11: Issues liked to sprint 6

5.6.1 Changing ports

In the beginning of this sprint, issue #518 was solved - resulting in PR #523 listed in Table 5.12. The ports defined in `HYPSO.h` tells the SW to which CSP port to send a command. As the ports of the service threads that are common for the SDR and the OPU also have the same CSP address, the author found it useful to change the common port names to `PL_*_PORT` instead of having two definitions of the same CSP address. This was also done to make the code more understandable, as the common generalized functions were using the `OPU_*_PORT` definitions that could possibly result in confusion when using the functions with the SDR. The work consisted of changing the definitions in `HYPSO.h` as illustrated in Figure 5.5 in addition to changing all port names in the code from `OPU_*_PORT` to `PL_*_PORT`. The work was done in the branch `sdr-opumirror` and the PR was tested by another team member at the last day of the sprint and merged by the author into the branch `sdr-services` of `hypso-sw`.



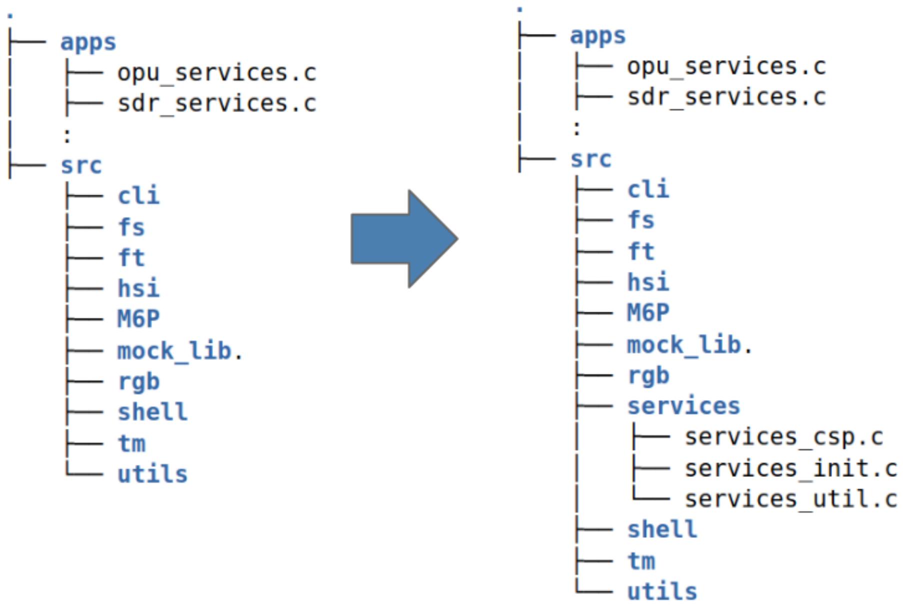
```
#define OPU_FT_PORT M6P_FT_PORT
#define OPU_RGB_PORT 11
#define OPU_HSI_PORT 12
#define OPU_CLI_PORT 13
#define OPU_TM_PORT 14
#define OPU_MONITOR_PORT 15

#define SDR_FT_PORT M6P_FT_PORT
#define SDR_CLI_PORT OPU_CLI_PORT
#define SDR_TM_PORT OPU_TM_PORT
#define SDR_MONITOR_PORT OPU_MONITOR_PORT
```

```
#define PL_FT_PORT M6P_FT_PORT
#define PL_CLI_PORT 13
#define PL_TM_PORT 14
#define PL_MONITOR_PORT 15

#define OPU_RGB_PORT 11
#define OPU_HSI_PORT 12
```

Figure 5.5: New port names.

5.6.2 Refactor

As mentioned in Section 1.3, the code-base for the satellite was initially customized for the OPU. When the SDR was added, most of the source code for `opu-services(/apps/opu_services.c)` was copied into the source code of `sdr-services(/apps/sdr_services.c)` by another team member. As this would result in double work while changing these files in the future, issue #477 was made by one of the other team members as they did this copying asking for the code to be refactored. This issue was solved making PR #529. Firstly, the author went through the two files to find the common code in order to design a new module for common app code, making sure it was general and easily used with any type of PL. Modularization, re-use and interfaces were in focus. The contribution consisted of making a new folder containing new files with the common code for the services. This code is general for any payload, and can be used if a new payload is to be added to the satellite in the future as well. The new folder, `/src/services`, is consisting of the three files `services_csp.c`, `services_init.c` and `services_util.c`. An illustration of the architecture can be seen in the figure below(Figure 5.6).



```
.
├── apps
│   ├── opu_services.c
│   ├── sdr_services.c
│   :
├── src
    ├── cli
    ├── fs
    ├── ft
    ├── hsi
    ├── M6P
    ├── mock_lib.
    ├── rgb
    ├── shell
    ├── tm
    └── utils
```

```
.
├── apps
│   ├── opu_services.c
│   ├── sdr_services.c
│   :
├── src
    ├── cli
    ├── fs
    ├── ft
    ├── hsi
    ├── M6P
    ├── mock_lib.
    ├── rgb
    ├── services
    │   ├── services_csp.c
    │   ├── services_init.c
    │   └── services_util.c
    ├── shell
    ├── tm
    └── utils
```

Figure 5.6: Added files.

Setup of the CSP services is done in `services_csp.c`, initializing of service threads is done in `services_init.c` and general help functions are present in `services_util.c`. The new files were all added to `CMakeLists.txt` for both payloads. The common code was removed from the original files and replaced by calls to the new functions.

The author got a few problems working with this issue as the development was done in the same branch as of issue #523, sdr-opumirror. As the other issue was not yet tested and merged, commits from this issue was pushed into the previous PR. This was undesirable as there was a different issue being solved, and the author wished to make a new PR after completing it as it was a rather large issue. The trouble arose when the author branched out from sdr-opumirror to make a new branch with the commits due to the refactoring while deleting the commits from the old branch. The new branch was not pushed into GitHub until the commits were already deleted from the old branch. This resulted in the new branch not seeing the commits made, and they could not be pushed into GitHub. After trying a lot of different git hacks, the final solution was to wait until PR #532 was merged, use the sdr-opumirror branch again and collect the old commits before pushing them again and making a new PR, #529, with the changes from the refactoring.

Pull Request	∼ *Title*	Linked issues	Appendix
#523	Changing port names to PL_*_PORT	#518	Appendix V
#529	Sdr refactor	#477	Appendix W

Table 5.12: Pull request of sprint 6

5.7 Sprint 7 - Telemetry service for the SDR

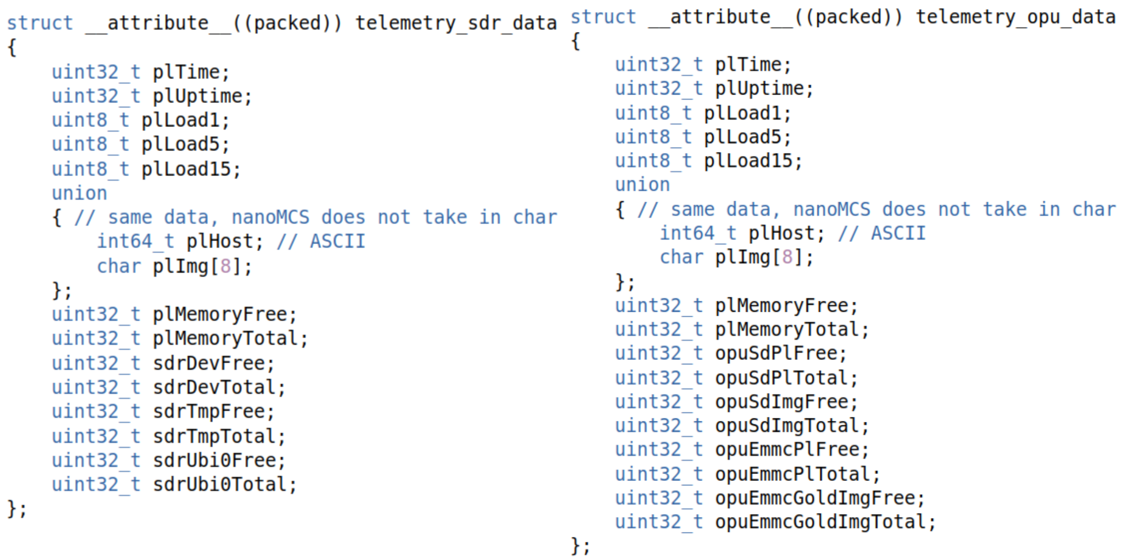
Within the first week of this sprint, PR #529 from sprint #6 (Section 5.6.2) was tested and approved of another team member, and later the branch, sdr-opumirror was again merged into the sdr-services branch of hypso-sw. Following this, the issue in focus for this sprint was #501 concerning the TM-service of the SDR, featured in Table 5.13. Working on this issue, the development from Section 5.3.2 was continued in the sdr-telemetry branch of hypso-sw. Before starting the further development of this branch, the new commits from sdr-service were merged into sdr-telemetry.

Issue	Title	Points	Linked PR	Appendix
#501	Telemetry service for the sdr	13	#506	Appendix L

Table 5.13: Issues liked to sprint 7

As mentioned in Section 5.3.4, the sdr-services shut down due to the creation of the TM log file. This part off issue #501 was resolved by starting sdr-services from the root user of the totem as explained in Section 5.5. The author suspected that it was the file-type of the log-file that was the problem, but it turned out to be the user permissions. After solving this part of the issue, there were still some remaining changes to complete to make the TM service work for multiple payloads.

The TM struct from Figure 5.4 was redefined during this sprint because there was a wish of also collecting similar variables as from the OPU. In addition to this, it was not yet decided on how to log the xadc_values. Hence an issue was made on this to be resolved in a later sprint, see Table 5.15 for reference. The new struct can be seen together with the TM struct for the OPU in the figure below(Figure 5.7).



```c
struct __attribute__((packed)) telemetry_sdr_data
{
    uint32_t plTime;
    uint32_t plUptime;
    uint8_t plLoad1;
    uint8_t plLoad5;
    uint8_t plLoad15;
    union
    { // same data, nanoMCS does not take in char
        int64_t plHost; // ASCII
        char plImg[8];
    };
    uint32_t plMemoryFree;
    uint32_t plMemoryTotal;
    uint32_t sdrDevFree;
    uint32_t sdrDevTotal;
    uint32_t sdrTmpFree;
    uint32_t sdrTmpTotal;
    uint32_t sdrUbi0Free;
    uint32_t sdrUbi0Total;
};
```

(a) SDR

```c
struct __attribute__((packed)) telemetry_opu_data
{
    uint32_t plTime;
    uint32_t plUptime;
    uint8_t plLoad1;
    uint8_t plLoad5;
    uint8_t plLoad15;
    union
    { // same data, nanoMCS does not take in char
        int64_t plHost; // ASCII
        char plImg[8];
    };
    uint32_t plMemoryFree;
    uint32_t plMemoryTotal;
    uint32_t opuSdPlFree;
    uint32_t opuSdPlTotal;
    uint32_t opuSdImgFree;
    uint32_t opuSdImgTotal;
    uint32_t opuEmmcPlFree;
    uint32_t opuEmmcPlTotal;
    uint32_t opuEmmcGoldImgFree;
    uint32_t opuEmmcGoldImgTotal;
};
```

(b) OPU

Figure 5.7: TM structs

The main remaining issue was the fact that structs with different variables - Figure 5.7 - were to be recorded for the different PLs. As inserting elements and extracting elements from structs in the programming language `C` has to be done manually, some of the telemetry functions had to be made specific for each payload as the developer also has to specify the type of struct to use as an argument to the function. As a result of this, the old file `/src/tm/tm_service.c` was divided into three files. The command functions that were common for all PLs were moved to the new file `/src/tm/tm_cmd.c`, and there was made one TM thread for each PL. Hence, the new files `/src/tm/tm_service_opu.c` and `/src/tm/tm_service_sdr.c` has a lot of quite similar but not identical code, and some specific functions concerning each payload. These functions are typically the ones that were put in `tm_opu.c` in Section 5.3.2. In addition to this, a few functions that turned out to be PL-specific had to be moved to these files from `src/tm/tm_util.c`. Examples of such functions are the ones used for logging as they take different structs as input. The new files are illustrated in Figure 5.8, and it can be seen that `tm_opu.c` and `tm_sdr.c` were removed as their functionality were moved into the new service files.

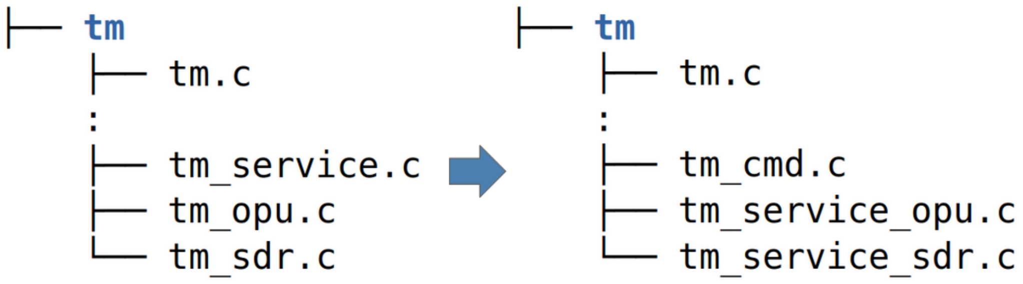


Figure 5.8: Refactoring of tm files

Some of the functions in `src/tm/tm_util.c` has been made specifically for the OPU. As a final part of completing and refactoring the TM-service, these functions were generalized with specified inputs and outputs instead of setting TM struct variables directly to modularize the code. This restructured the functions in a way to make them more compatible with an agile architecture as they would only deliver the values requested instead of changing values themselves.

The final work done by the author before making the PR, was to add two commands to `tm_service_sdr.c`. The first one `sdr xadc` was made to give the user access to the values from this `totem` command throgh `hypso-cli`. The other one, was `sdr tmlog`, which can change the log ineterval and turn on/off logging on the SDR. This command which was made accessible through the generalized function in tm_cmd.c, as the funcionality is similar for the OPU and the SDR. Every change mentioned in this sprint was done in the sdr-telemetry branch, and the PR featured in Table 5.14 was made. It was approved within the last day of this sprint by another team member, and after correcting one spelling mistake in a printout, the branch was merged into the sdr-services branch of hypso-sw by the author.

Pull Request	∼ *Title*	Linked issues	Appendix
#538	Sdr telemetry	#501	Appendix X

Table 5.14: Pull request of sprint 7

5.7.1 Issues created

One issue still remained for the TM logging of the SDR to be sufficient; to find a way to log the xadc_values of the TOTEM. This is discussed with quite a few options in the issue linked in Table 5.15, and continued in Section 5.8. The second issue was made regarding a few functions in cli_pl.c that were not yet fully genrralized for any PL.

Issue	Title	Points	Linked issue	Appendix
#542	Logging of xadc_values	8	-	Appendix Q
#543	Generalizing some _pl_-functions	5	-	Appendix R

Table 5.15: Issues created in sprint 7

5.8 Sprint 8 - Generalizing functionality and logging of XADC values

This sprint consisted of the issues in Table 5.16. The sprint lasted for three weeks instead of two as May has a few bank holidays in Norway and the initial date landed on one of these. The main focus for the author within this sprint was to design the final PL commands to be generalized in order to make the code reusable for any PL and making it easier to maintain the functionality when changes are requested.

Issue	Title	Points	Linked PR	Appendix
#541	Add payload telemetry struct to repository	1	-	Appendix P
#542	Logging of xadc_values	8	#546	Appendix Q
#543	Generalizing some _pl_-functions	5	#547	Appendix R

Table 5.16: Issues worked with in sprint 8

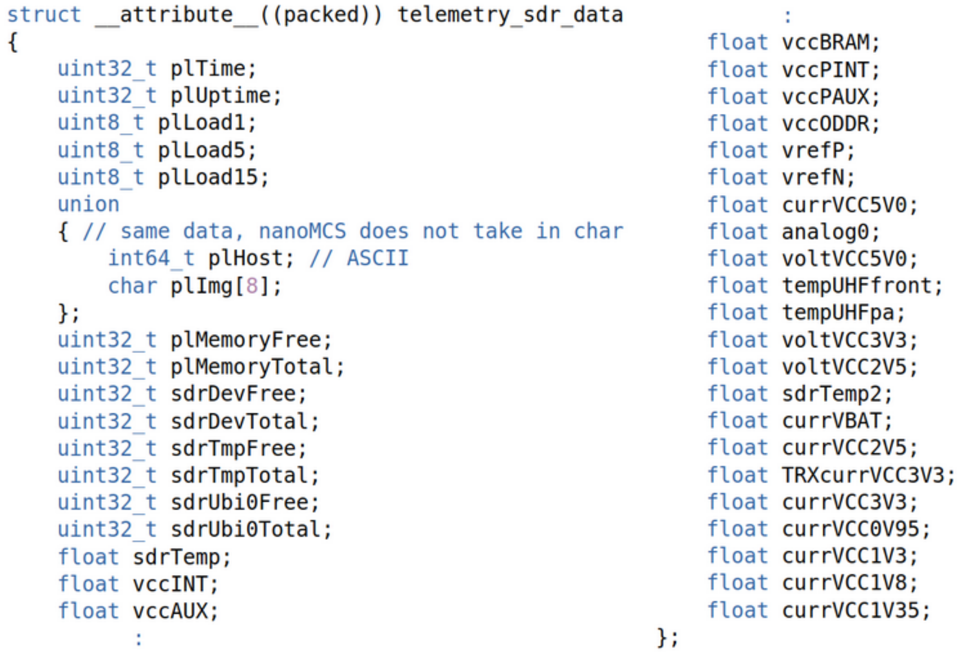
5.8.1 Add payload telemetry struct to repository

The first issue, #541, was simply to upload two text files to Github. These files were containing the TM structs for the OPU and the SDR similar to the ones illustrated earlier in Figure 5.7. These files are used to parse the TM logging files in nano mcs. As seen in Figure 5.7, the file displays the order, type and size of the elements in the struct. Storing these structs in known locations makes it easier for the other team members both to use and test the code. The structs were saved in a different repository called hypso-telemetry-c-structs [39]. Before uploading the files, the author created a new branch add-tmstructs and uploaded the structs to folders representing their

PL. A PR was made to merge the branch into the master branch of the repository, and it was approved and merged by one of the other team members within the period of the current sprint.

5.8.2 Logging of xadc-values

The author branched out a new branch from the `sdr-services` branch of `hypso-sw`, called `sdr-xadc`, in order to solve issue #542. As temperatures, voltages and currents from the `totem` is not already being logged as a part of the firmware, this has to be included in the telemetry logging. These are the values made available in `hypso-cli` with the command `sdr xadc` in Section 5.7. To be able to log them in the telemetry file, the values had to be collected and put into the already existing TM struct. Resulting in an updated TM struct on the format seen in Figure 5.9. Hence, the file representing the sdr-struct from Section 5.8.1 had to be updated as well.



```
struct __attribute__((packed)) telemetry_sdr_data
{
    uint32_t plTime;
    uint32_t plUptime;
    uint8_t plLoad1;
    uint8_t plLoad5;
    uint8_t plLoad15;
    union
    { // same data, nanoMCS does not take in char
        int64_t plHost; // ASCII
        char plImg[8];
    };
    uint32_t plMemoryFree;
    uint32_t plMemoryTotal;
    uint32_t sdrDevFree;
    uint32_t sdrDevTotal;
    uint32_t sdrTmpFree;
    uint32_t sdrTmpTotal;
    uint32_t sdrUbi0Free;
    uint32_t sdrUbi0Total;
    float sdrTemp;
    float vccINT;
    float vccAUX;
          :
```

```
          :
    float vccBRAM;
    float vccPINT;
    float vccPAUX;
    float vccODDR;
    float vrefP;
    float vrefN;
    float currVCC5V0;
    float analog0;
    float voltVCC5V0;
    float tempUHFfront;
    float tempUHFpa;
    float voltVCC3V3;
    float voltVCC2V5;
    float sdrTemp2;
    float currVBAT;
    float currVCC2V5;
    float TRXcurrVCC3V3;
    float currVCC3V3;
    float currVCC0V95;
    float currVCC1V3;
    float currVCC1V8;
    float currVCC1V35;
};
```

Figure 5.9: Initial TM struct for SDR.

As can be seen in the issue #542, there was suggested a few options by the author, but none of them were optimal to solve this problem. The issue requested the possibility of logging the output form the `xadc_values` command on the `TOTEM` which generates a printed output though a script. Saving this in a struct included knowing the exact content and order of the command to be able to parse it. A change in this order would mean a change in the command, which is undesirable. The other suggested solution, was to log the whole output of the command at each logging instance. The downside of this solution was that the parser used to read the log files would not understand this format as it converts files based on structs and their sizes. Hence, the solution would have resulted in the need of a new tool to parse the files. In addition to this, it would result in a larger log file as unnecessary info would be logged as well, and memory is a restriction in the PLs.

As none of the suggested solutions were optimal, the scrum method and issues became beneficial. Other team members also have access to the issues, and one of the team members with more knowledge about the systems and its files suggested an alternative solution. The raw values can be found in a specific folder on the `totem` in the same files used to generate the script for the `xadc_values` command. Hence, these variables can be read, and after some calculations with offset and scaling they could be converted to intelligible values. This solution made it easier to store the variables in the telemetry struct, and the formula to convert the raw values can be seen below. All values could be collected from files on the TOTEM, and the desired result is the mag_value.

$$adc\_value = (raw\_value + offset\_value) * scale\_value; \tag{5.1}$$

$$mag\_value = (adc\_value * mag\_scale) + mag\_offset; \tag{5.2}$$

An issue arose while the author was testing this functionality, as the system could not log the instance to the file as it was too large. When creating the log file, an input size is also set dependent on the size of the struct. Therefore, the old log-file has to be deleted when changing the size of the struct.

The contributions to the code base were to add the xadc values to the TM struct of the SDR in addition to adding two new functions to tm_service_sdr.c in order to read the files containing the values and calculate the final values. These changes resulted in PR #546 referenced in Table 5.17. Here, the branch sdr-xadc was requested merged into sdr-services

5.8.3 Generalizing _pl_functions

Some of the PL functions that were rewritten in PR #506 (Section 5.3.1), were working for both the OPU and the SDR without being fully generalized. This meant that for using the functions with another PL later, it would be required to add another if-sentence. This is not optimal, and the goal of issue #543 was to make these functions work with any PL with specific input and output. In this way, the functions can be re-used with another PL as requested in Section 4.1.1. They will also be easier to maintain. Working with this issue, a new branch, sdr-generalize was branched out from the hypso-sw branch sdr-services.

The first function to be changed was _pl_list which had different options for the list command of the OPU and the SDR. Instead of setting this inside the function, the function was redefined (Figure 5.10) with list_command as an argument so that the wrapper functions could send in what command to send in addition to the PL. A similar change was made for _pl_log, where the location of the log was sent as an argument instead of set inside of the function.

```
int cli_sdr_list(char* args)
{
    const char* list_command = "ls -Rl;"

    return _pl_list(args, HYPSO_SDR_ADDRESS, list_command);
}

int _pl_list(char* args, int pl_address, const char* list_command);
```

Figure 5.10: Wrapper function and definition of generalized log function.

The formatting of the feedback of the TM command also had to be moved in order to make all functions in cli_pl.c general for any PL. Since all PLs have different TM structs, the output of this command will be different for each PL. Therefore, the formatting of the output was moved to the PL specific files, and only the TM request was made in the general file. The had to be removed because the variables depend on the type of PL, resulting in the commands sdr telemetry and opu telemetry being the available commands to fetch instant telemetry.

The final function that had to be changed in order to make all _pl_-functions general was _pl_update. This functions was refactored to four functions, the PL specific parts of the function were moved to cli_sdr.c and cli_opu.c, and the refactored functions were called from there. The new functions are as listed:

- _pl_update_check: general checks (as path checks and arg checks).

- `_pl_check_img`: checking if the file for update is an image.ub-file.
- `_pl_check_exe_arm`: checking if file is an arm executable(`opu-services` and `sdr-services`);
- `_pl_tar`: compresses file to be updated.

The result of the work done om the branch `sdr-generalize` is that all `_pl_`-functions are now generalized. Meaning that all functions located in `src/cli/cli_pl.c` have a specified input and output, and could be used by any PL. Functions that are not general are located in the CLI file which fit their PL. All changes were committed to the `sdr-generalized` branch, and PR #547 which is listed below was made to merge it into the `sdr-services` branch of `hypso-sw`. The team did not have the time to test it within the scope of this sprint, so it was not yet merged.

Pull Request	∼ *Title*	Linked issues	Appendix
#546	Sdr xadc	#542	Appendix Y
#547	Sdr generalize	#543	Appendix Z

Table 5.17: Pull request of sprint 8

Chapter 6

Discussion and Conclusion

Throughout the semester, the software team has developed code incrementally with an agile approach using the methods from Chapter 4. The work has consisted of planning, construction and testing.

6.1 Planning

The planning tied to this master's thesis includes thorough design of the desired architecture combined with discussions with other team members to conclude with the best solution to each problem. A light version of the results of the design was posted as issues in GitHub, often suggesting one preferred solution of implementation joined by one or multiple backup solutions. The focus when designing the architecture will be discussed in Section 6.2, while this section will address the approach towards the planning, including the Scrum method and the GitHub workflow.

6.1.1 GitHub Workflow

After working on the HYPSO project for exactly a year, starting as a summer intern in June of 2020, the author appreciates the GitHub workflow even more than in the beginning. Making issues in GitHub at the time they are encountered following the agile approach, helps tidying up the work. If an issue is explained well, it simplifies the understanding of a problem for the other members on the team. This is especially helpful when solving an issue made by another team member. Using issues on GitHub has also been efficient with regards to documentation of work, as anyone on the team can look up any issue made in a common GitHub repository whenever they want, for example to look up a design choice or a solution to a bug. A possible result could be less time spent on fixing of bugs. This can also be helpful for the new students on-boarding the project every semester in order to get into the project. Creating an issue also allow other team members to comment with other possible solutions to the problem encountered, or even add to the issue with problems the author of the issue did not predict. The feedback from other team members in GitHub issues has helped the author making design choices when integrating the SDR into the `hypso-sw` code base, it has also made the design process more efficient.

Having a common repository and workflow for all team members involved in the project, makes continuous integration of development possible. This is a huge benefit as opposed to integrating different contributions after the implementations are done when collaborating on a SW project. As everyone on the team can look into the code developed by others in the common repositories, both reuse and consistency is improved across the whole code base. An important part of the GitHub workflow, is having a main branch which is always working. When a bug or feature request is fixed or developed in its own, dedicated branch, the new features will not be merged into the main branch until the development branch is reviewed and tested. Hence, if anything goes wrong while developing, nothing happens to the main branch, and the developer can start over again.

A difference from the work done in the specialization project, is the development branch `sdr-services` which branch out beyond the main branch. Development was done here to separate the code which integrates the SDR from the code meant for HYPSO-1. A lot of the code base has been refactored during the 8 sprints this semester consisted of, and everything was tested and merged into the `sdr-services` branch instead of the master branch. Thus, a large PR has to be made in order to merge everything into the master branch when the team decides that the time has come for that to happen. This includes even more testing to make sure there are no bugs and everything works as desired. An issue of waiting to merge the two branches, is that there is a risk that the two branches then diverges when new features are implemented to the master branch. In order to avoid this, it is cruicial to merge the master branch into the `sdr-services` branch when changes are made to the master branch. The main reason for not merging the two branches yes, is that the functionality regarding the SDR is not needed on HYPSO-1, and there is always some risk that there is an unnoticed bug after refactoring such a large amount of code.

6.1.2 Scrum Method

Within the HYPSO SW team, the Scrum framework is used to facilitate for agile development. Breaking down tasks and setting goals for two weeks at a time has been beneficial as there has been requested a lot of new features in addition to the change of some requirements. Discussing the prioritization and workload of the issues made since the last sprint is helpful before deciding what each team member shall focus on for the next weeks. As the team has helped prioritize the issues the author has solved, the resulting order of issues solved lead to a progress which benefited the whole team. The discussions with the team has improved the authors judgement of how time-consuming different tasks are, and it has helped the author in prioritizing tasks. The Kanban board was a useful tool when planning with the team, as it is easier to follow when all issues are displayed visually in different columns. It is also an excellent way of cooperating with the rest of the team to structure work. During the time between sprint meetings, the Kanban board could be used to track the progress, get an overview of the current status of the whole SW team, or check what is planned if someone has forgotten.

6.2 Construction

The code development of this master's thesis has been constructed around the architectural designs made by the author. In order to develop a flexible architecture which can support multiple payloads, modularity, reusability and generalization has been in focus. To be able to satisfy the fundamentals of an agile architecture, the problem has been broken down to smaller tasks to be able to focus on one task at a time which have been solved and continuously integrated into the `sdr-services` branch pg `hypso-sw`.

6.2.1 Code quality

Working in group projects amplifies the importance of code quality. While working on the project thesis in the previous semester, the author spent quite a bit of time trying to understand code written by other people before adding functionality to the project. During this work, the importance of having maintainable, reusable, readable and testable code was made clearer to the author. Thereby, the author spent a lot of time on writing code of quality during the development of the code connected to this master's thesis to make the job easier for other people continuing the work on this SW later. The main focuses were to make functions general, modularize code, and intuitive naming of files and functions.

Modularization has been in mind, both when deciding inputs and outputs of different functions, and when deciding on the content of different files. When rewriting the util-functions in Section 5.7, they were given specified inputs and outputs instead of setting values to variables outside of the function. Hence, the functions became self-contained and encapsulated, and even reusable for the SDR. Clear definitions have thus proven beneficial as the functions now have functionality and methods independent of other modules.

Generalization of code has been a tool for making the code modular, as the generalization of code previously dedicated to the OPU has made new modules with clear interfaces in order to make them work with any type of PL. This allows for easy plug-and-play with different PLs, which is one of the main goals of modularization. The result of the new modularized architecture, is an adapted and flexible architecture which easily can integrate a new PL.

6.2.2 Functionality

The resulting functionality developed during the scope of this master's thesis worked as intended during the planning and designing. Some of the design choices made to obtain an agile sw framework will be discussed below.

Finalizing previous work

The final commits to fix the remaining issues in PR #426 (Appendix S) were important to merge the `telemetry-service` branch into the master branch of `hypso-sw`. As the TM-service was also a significant part of the development done to integrate the SDR into the code-base, the approval of this PR and merge into the master branch were crucial for the final result. The functions added to the PR with these commits, were general so that they could also be used for the SDR.

Mirroring of OPU commands

The decision on how to solve this issue was well designed. The author made the design thoroughly as a lot of the development branched out from this and a poor design would have resulted in a lot of extra work in order to be fixed. The main focus when designing the architecture before making issues on how to implement the mirroring of the commands were to make the new architecture as general and modular as possible. In order to obtain this, the functions originally designed for the OPU was generalized to work for any PL. The actual implementation of this functionality was similar for each of the functions. Hence, after making the design and testing it with one function, the rest of the functionality was easy to implement.

After generalizing the functions and making wrappers for the different PLs, adding a third payload would be less demanding than before. This will be an advantage in the future, as the development made now will be brought forward to new small satellites with different missions. With general code, the development will be less demanding and time consuming in the future. Since the wrapper-functions work similarly for any PL, the developer could simply add the new PL address and use it as an argument in the wrapper-functions of the new PL.

Integrating the SDR in the TM service

The initial idea was to use the same TM-service thread for both PLs, and having different PL specific functions in their own files. The design plan for this solution would have been to keep the service common with generalized functionality, and moving the PL-specific functions and commands to their own files. This would result in good re-use of code in addition to generalization and modularity.

A problem arose making this a tricky issue to solve as because the TM logging is a part of the constant TM-service loop, and the variables to be logged differ for the different PLs. As the logging is to happen automatically as a part of the service, neither the author, nor any of the team members, saw any possibility to put it anywhere else. This resulted in different TM-serivce threads for the different PLs, but the code was still made as refactored as possible with all common functions put in files that were common for both services. Such an obstacle was not foreseen, but after some re-thinking of the issue, the new design also managed to re-use parts of the code resulting in a more flexible architecture.

Refactoring main source files

The file `sdr_services.c` was generated as a modified copy of `opu_services.c`. If the source code of these files were to be updated, the same changes would have been made in two places. This is undesirable over having common functions for the shared functionality. The change resulted in new, generalized functions ready for reuse by any payload. The common functionality connected to CSP commands, initialization of services, and other utilities were all refactored as their own modules. The new modules have clear interfaces, and independent functionality of the other modules in order to contribute to the agle SW architecture.

Generalizing functionality

Some functionality present the current version of the SW is not well suited for generalization, as it is too specific. Examples are fetching of telemetry and updating specific files on a PL. As the TM variables to be fetched are different for each PL, the struct definitions also differ from each other. The operational part of the team also wishes to have the opportunity of faster update of some files. Such functionality is hard to generalize as a lot of parameters are different for the different PLs. Hence, the functions were broken down to smaller parts of common functionality, resulting in larger wrapper functions calling many generalized functions. This part of the architecture is, to a certain extent generalized, but unfortunately more work is needed to use this functionality with a new PL.

6.3 Testing

Testing is an important part of any SW project. In the HYPSO SW team, it has been decided that a branch cannot be merged into the master branch before at least one other team member than the one making the PR has approved it. Hence, the same rule was used for the `sdr-services` branch, even though it is not blocked automatically by GitHub like merging of the master branch. Most of the testing related to the development of this thesis was done by the author before committing and opening a PR to confirm the functionality developed. Having team members testing the functionality is useful as well. Another team member might test for different scenarios, and hence pick up bugs that were not found by the one making the PR. In addition to manually testing the code, automated tests can be run by a computer called Jenkins in order to test for common scenarios on the OPU. These automated tests can not be run on the SDR as there is no SDR setup connected to Jenkins. As the architecture of the OPU commands was changed to work in the same way as the SDR commands, these tests would reflect the implementation done. Even so, it would have been beneficial to run the automated tests on the SDR as well.

Having a workflow which includes other team members testing the code developed helped finding bugs, and ensured better quality of the code developed. Testing can also be a way of getting to know the new features made by other team members and learning how to use them. As the different issues were divided and made as small as possible, no PRs made as a result of this thesis took too long to get tested and approved. Some PRs were approved quickly as the changes to the code base were minimal, whereas for the PRs with the largest amount of refactoring, more extensive testing was required. Even more extensive testing would be required before merging the `sdr-services` branch into the main branch of `hypso-sw` as the consequences of introducing bugs can be large. It would be beneficial to have Jenkins run its automated tests on the SDR as well before merging. As a lot of code has been changed, the probability of there being some scenario which has not been tested for is present.

6.4 Future Work

As a result of the development done by the author within the scope of this master's thesis, the SDR is fully integrated into the SW. Every issue were solved regarding refactoring and generalizing of code in order to develop a flexible SW platform to support multiple PLs. Nevertheless, there is remaining work to be done before the part of the code related to the SDR is ready to be launched with HYPSO-2. A lot of the future work includes developing functionality specifically for the SDR.

The only issue regarding the SDR remaining from the scope of this master's thesis, is the problem with user permissions. This is an issue which has to be solved in `sdr-system`, and the goal is generate a user which has user permissions which is somewhere in-between the `root` and the `totem` user.

Before starting the development of SDR-specific features, the team has to make a decision of when and how to merge the `sdr-services` branch into the master branch of `hypso-sw`. Things to be considered are diverging branches, and sufficient testing of the branch before merging. Hence, the sdr could be considered connected to Jenkins in order to run automated tests.

The SDR does not have any private services yet, but there will probably evolve a need in the future for a service which shall run application code related to the radio. An example of use for this service would be to gather data connected to the on-orbit interference over the Arctic.

Bibliography

[1] Tuva Okkenhaug Moxnes. *Telemetry Service Logging System for CubeSat.* Non-published. 2020.

[2] Mariusz Grøtte. *Mission Operations Plan.* HYPSO-MOP-001. Internal document. Non-published.

[3] Gara Quintana-Díaz et al. "An SDR mission measuring UHF signal propagation and interference between small satellites in LEO and Arctic sensors". In: (2021 - non-published).

[4] NASA CubeSat Launch Initiative. *CubeSat101, Basic Concepts and Processes for First-Time CubeSat Developers.* `hhttps://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf`. Accessed: 2020-12-22. 2011.

[5] *NanoAvionics website.* `https://nanoavionics.com/nanosatellite-buses/6u-nanosatellite-bus-m6p/`. Accessed: 2020-12-15.

[6] HYPSO project team. *System Design Report.* HYPSO-DR-001. Internal document. Non-published.

[7] Avnet. "TOTEM Motherboard - Softwae-Defined Radio for nanosatellites - Datasheet". In: (2019).

[8] Amund Gjersvik. *Breakout Board V3 ICD.* HYPSO-ICD-003. Internal document. Non-published.

[9] Avnet. "UltraZed-EV™ Carrier Card Designer's Guide". Version 1.1. In: (2018).

[10] Keith W. Ross James F. Kurose. *Computer Networking - A Top-Down Approach.* 6th ed. 2013.

[11] GomSpace. *CubeSat Space Protocol.* `https://bytebucket.org/bbruner0/albertasat-on-board-computer/wiki/1.%20Resources/1.1.%20DataSheets/CSP/GS-CSP-1.1.pdf?rev=316ebd49bed49fdbb1d74efdeab74430e7cc726a`. Accessed: 2020-12-14. 2011.

[12] J. Garret S. Bakken and R. Birkeland. *HYPSO SW Design Report.* HYPSO-DR-005. Internal document. Non-published.

[13] M. Hjertenæs and Magnus Danielsen. *Platform-Payload Interface Control Document.* HYPSO-ICD-001. Internal document. Non-published.

[14] Texas Instruments. *Introduction to the Controller Area Network (CAN).* `https://www.ti.com/lit/an/sloa101b/sloa101b.pdf`. Accessed: 2020-12-14. 2016.

[15] Amund Gjersvik. *Journal on RS-422 transceiver selection process.* Internal document. Non-published. 2021.

[16] HYPSO-SW team. *hypso-sw.* `https://github.com/NTNU-SmallSat-Lab/hypso-sw`. GitHub repository.

[17] HYPSO-SW team. *opu-system.* `https://github.com/NTNU-SmallSat-Lab/opu-system`. GitHub repository.

[18] HYPSO-SW team. *sdr-system.* `https://github.com/NTNU-SmallSat-Lab/sdr-system`. GitHub repository.

[19] Andrew W. Appel. "Verified Software Toolchain". In: *European Symposium on Programming* (2011), pp. 1–17.

[20] *Docker website.* https://www.docker.com/. Accessed: 2020-12-21.

[21] A. Vaduva. *Learning Embedded Linux Using the Yocto Project.* 6th ed. Packt Publishing, 2015.

[22] Roger Birkeland and Dennis Langer. *Manual for Flatsat and Lidsat.* HYPSO-UM-004. Internal document. Non-published.

[23] Rick Dove and Ralph LaBarge. "Fundamentals of Agile Systems Engineering - Part 2". In: (2014).

[24] Rick Dove and Ralph LaBarge. "Fundamentals of Agile Systems Engineering - Part 1". In: (2014).

[25] Kaisa Könnöla et al. "Can embedded space system development benefit from agile practices?" In: *J Embedded Systems* (2017). ISSN: 1687-3963. DOI: https://doi.org/10.1186/s13639-016-0040-z. URL: https://jes-eurasipjournals.springeropen.com/articles/10.1186/s13639-016-0040-z#citeas.

[26] Magnus Thorstein Sletholt et al. "A Literature Review of Agile Practices and Their Effects in Scientific Software Development". In: *Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering.* SECSE '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 1–9. ISBN: 9781450305983. DOI: 10.1145/1985782.1985784. URL: https://doi.org/10.1145/1985782.1985784.

[27] Stefanie Paluch et al. "Stage-gate and agile development in the digital age: Promises, perils, and boundary conditions". In: *Journal of Business Research* 110 (2020), pp. 495–501. ISSN: 0148-2963. DOI: https://doi.org/10.1016/j.jbusres.2019.01.063. URL: https://www.sciencedirect.com/science/article/pii/S0148296319300827.

[28] A. van Bennekum et al. K. Beck M. Beedle. *Manifesto for Agile Software Development.* URL: https://agilemanifesto.org/. (accessed: 05.06.2020).

[29] European Cooperation for Space Standardization. *Space engineering - Agile software development handbook.* ECSS-E-HB-40-01A. ESA Requirements and Standards Division. 2020.

[30] Greg Wilson et al. "Best Practices for Scientific Computing". In: *PLOS Biology* 12.1 (Jan. 2014), pp. 1–7. DOI: 10.1371/journal.pbio.1001745. URL: https://doi.org/10.1371/journal.pbio.1001745.

[31] Steve McConnel. *Code Complete.* 2nd ed. Microsoft, 2004.

[32] *Clang-format.* https://www.kernel.org/doc/html/latest/process/clang-format.html. Accessed: 2020-12-21.

[33] Tham Wai Yip Loo Wooi Khong Leau Yu Beng and Tan Soo Fun. "Software development life cycle agile vs traditional approaches". In: (2012).

[34] N. S. Janoff and Linda Rising. "The scrum software development process for small teams". In: (2000).

[35] Scott Chacon and Ben Straub. *Pro Git.* 7th ed. Apress, 2020.

[36] *Understanding the GitHub flow.* https://guides.github.com/introduction/flow/. Accessed: 2020-12-10.

[37] Wikipedia. *Universal asynchronous receiver-transmitter.* URL: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter. (accessed: 04.06.2021).

[38] *C - Structures.* https://www.tutorialspoint.com/cprogramming/c_structures.htm. Accessed: 2020-12-22.

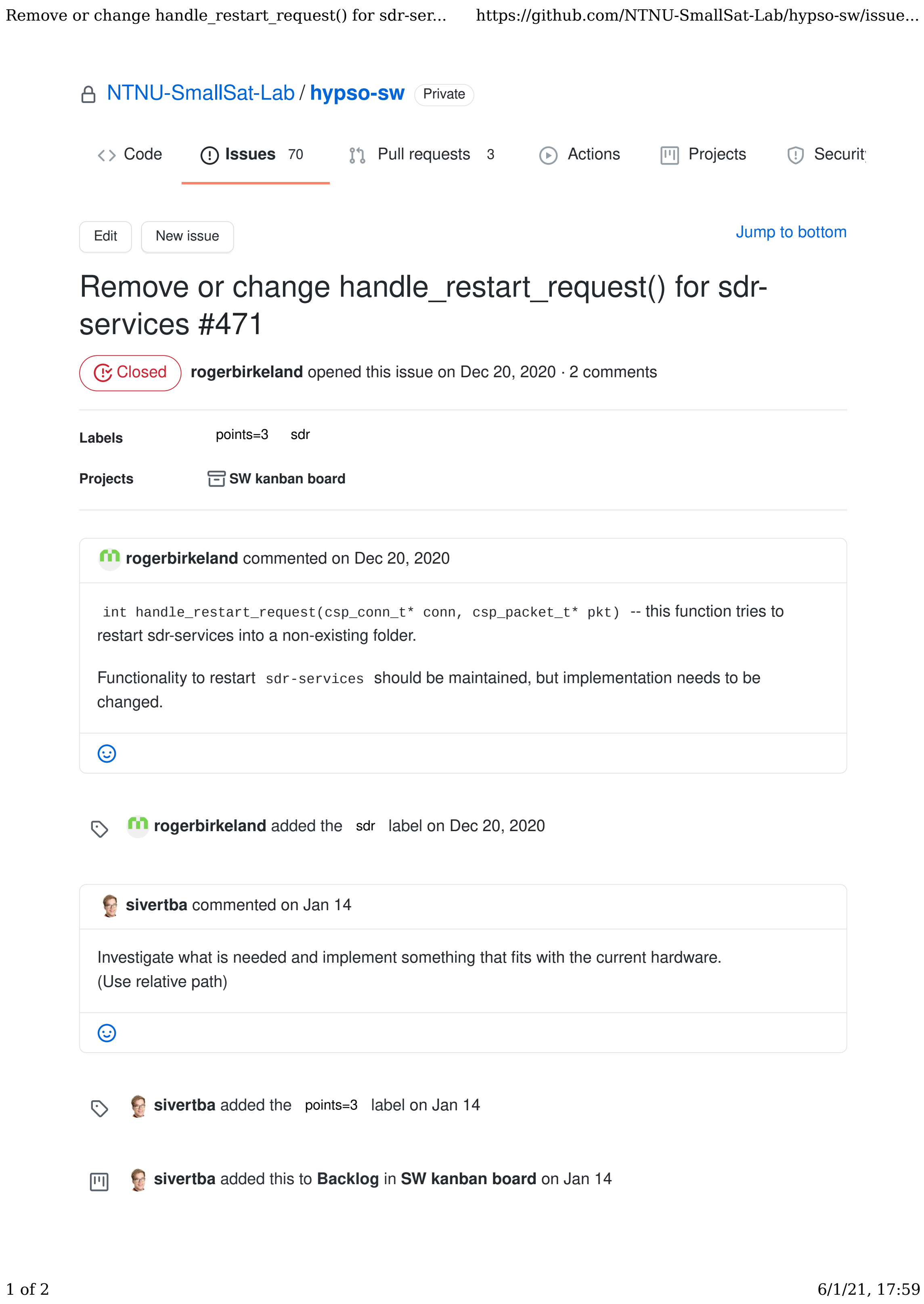
[39] HYPSO-SW team. *hypso-telemetry-c-structs.* https://github.com/NTNU-SmallSat-Lab/hypso-telemetry-c-structs. GitHub repository.

Appendix

A tm_log.h

```
#ifndef TM_LOG_H
#define TM_LOG_H

#define TELEMETRY_SAVE_FOLDER "telemetry/"
#define TELEMETRY_SAVE_FILE "telemetry.log"
#define TELEMETRY_SAVE_INTERVAL "telemetry_interval.txt"
#define TELEMETRY_LOG_INTERVAL 60 // seconds
#define TM_FILE_ID 1
#define TM_ENTRY_SZ_OPU sizeof(struct telemetry_opu_data)
#define TM_ENTRY_SZ_SDR sizeof(struct telemetry_sdr_data)
#define TM_ENTRY_COUNT 2100 // 7(days)*(10*30)(min on-time)

#include <time.h>
#include <stdio.h> //for FILE*

bool tm_logging_on;
uint16_t tm_logging_interval;

struct __attribute__((packed)) telemetry_opu_data
{
    uint32_t plTime;
    uint32_t plUptime;
    uint8_t plLoad1;
    uint8_t plLoad5;
    uint8_t plLoad15;
    union
    { // same data, nanoMCS does not take in char
        int64_t plHost; // ASCII
        char plImg[8];
    };
    uint32_t plMemoryFree;
    uint32_t plMemoryTotal;
    uint32_t opuSdPlFree;
    uint32_t opuSdPlTotal;
    uint32_t opuSdImgFree;
    uint32_t opuSdImgTotal;
    uint32_t opuEmmcPlFree;
    uint32_t opuEmmcPlTotal;
    uint32_t opuEmmcGoldImgFree;
    uint32_t opuEmmcGoldImgTotal;
};

struct __attribute__((packed)) telemetry_sdr_data
{
    uint32_t plTime;
    uint32_t plUptime;
    uint8_t plLoad1;
    uint8_t plLoad5;
    uint8_t plLoad15;
    union
    { // same data, nanoMCS does not take in char
        int64_t plHost; // ASCII
        char plImg[8];
    };
    uint32_t plMemoryFree;
    uint32_t plMemoryTotal;
    uint32_t sdrDevFree;
    uint32_t sdrDevTotal;
    uint32_t sdrTmpFree;
    uint32_t sdrTmpTotal;
    uint32_t sdrUbi0Free;
    uint32_t sdrUbi0Total;

    float sdrTemp;         //mC
    float vccINT;          //mV
    float vccAUX;          //mV
    float vccBRAM;         //mV
    float vccPINT;         //mV
    float vccPAUX;         //mV
    float vccODDR;         //mV
    float vrefP;           //mV
    float vrefN;           //mV
    float currVCC5V0;   //mA
    float analog0;         //
    float voltVCC5V0;   //V
    float tempUHFfront; //C
    float tempUHFpa;    //C
    float voltVCC3V3;   //V
    float voltVCC2V5;   //V
    float sdrTemp2;        //C
    float currVBAT;     //mA
    float currVCC2V5;   //mA
    float TRXcurrVCC3V3;//mA
    float currVCC3V3;   //mA
    float currVCC0V95;  //mA
    float currVCC1V3;   //mA
    float currVCC1V8;   //mA
    float currVCC1V35;  //mA
};

struct telemetry_log_file
{
    FILE* p_telemetry_log_file;
    char tm_filepath[100];
    time_t timestamp;
    //  bool tm_log_status;   //global variable instead to be able to change it
    //  from hsi capture without passing the log struct
};

#endif /* TM_LOG_H */
~
~
```

B Issue #471 & #472 hypso-sw



Remove or change handle_restart_request() for sdr-ser... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

Remove or change handle_restart_request() for sdr-services #471

Closed **rogerbirkeland** opened this issue on Dec 20, 2020 · 2 comments

Labels points=3 sdr

Projects **SW kanban board**

rogerbirkeland commented on Dec 20, 2020

`int handle_restart_request(csp_conn_t* conn, csp_packet_t* pkt)` -- this function tries to restart sdr-services into a non-existing folder.

Functionality to restart `sdr-services` should be maintained, but implementation needs to be changed.

rogerbirkeland added the sdr label on Dec 20, 2020

sivertba commented on Jan 14

Investigate what is needed and implement something that fits with the current hardware.
(Use relative path)

sivertba added the points=3 label on Jan 14

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

1 of 2 6/1/21, 17:59

rogerbirkeland commented on Mar 24 Author

Closed due to #504

☺

rogerbirkeland closed this on Mar 24

SW kanban board automation moved this from **Backlog** to **Done** on Mar 24

Assignees

No one—assign yourself

Labels

points=3 sdr

Projects

SW kanban board

Done ▾

Milestone

No milestone

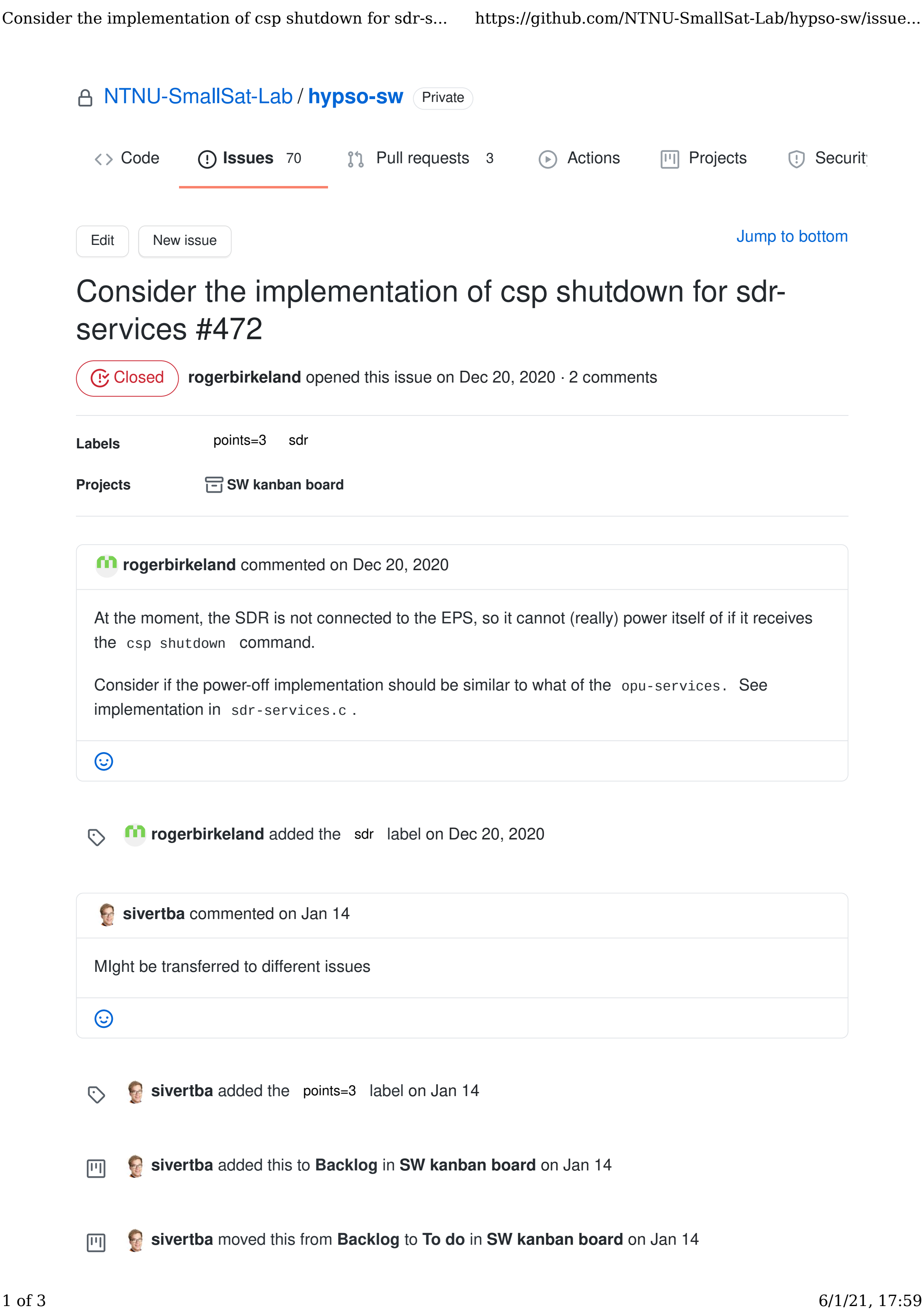
Linked pull requests

Successfully merging a pull request may close this issue.

None yet

2 participants

Pin issue ⓘ



Consider the implementation of csp shutdown for sdr-s... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

Consider the implementation of csp shutdown for sdr-services #472

Closed **rogerbirkeland** opened this issue on Dec 20, 2020 · 2 comments

Labels	points=3 sdr
Projects	SW kanban board

rogerbirkeland commented on Dec 20, 2020

At the moment, the SDR is not connected to the EPS, so it cannot (really) power itself of if it receives the `csp shutdown` command.

Consider if the power-off implementation should be similar to what of the `opu-services`. See implementation in `sdr-services.c`.

rogerbirkeland added the sdr label on Dec 20, 2020

sivertba commented on Jan 14

MIght be transferred to different issues

sivertba added the points=3 label on Jan 14

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

sivertba moved this from **Backlog** to **To do** in **SW kanban board** on Jan 14

1 of 3 6/1/21, 17:59

sivertba moved this from **To do** to **Backlog** in **SW kanban board** on Jan 14

rogerbirkeland commented on Mar 24 Author

Closed due to #503

☺

rogerbirkeland closed this on Mar 24

SW kanban board automation moved this from **Backlog** to **Done** on Mar 24

Assignees

No one—assign yourself

Labels

points=3 sdr

Projects

SW kanban board

Done ▾

Milestone

No milestone

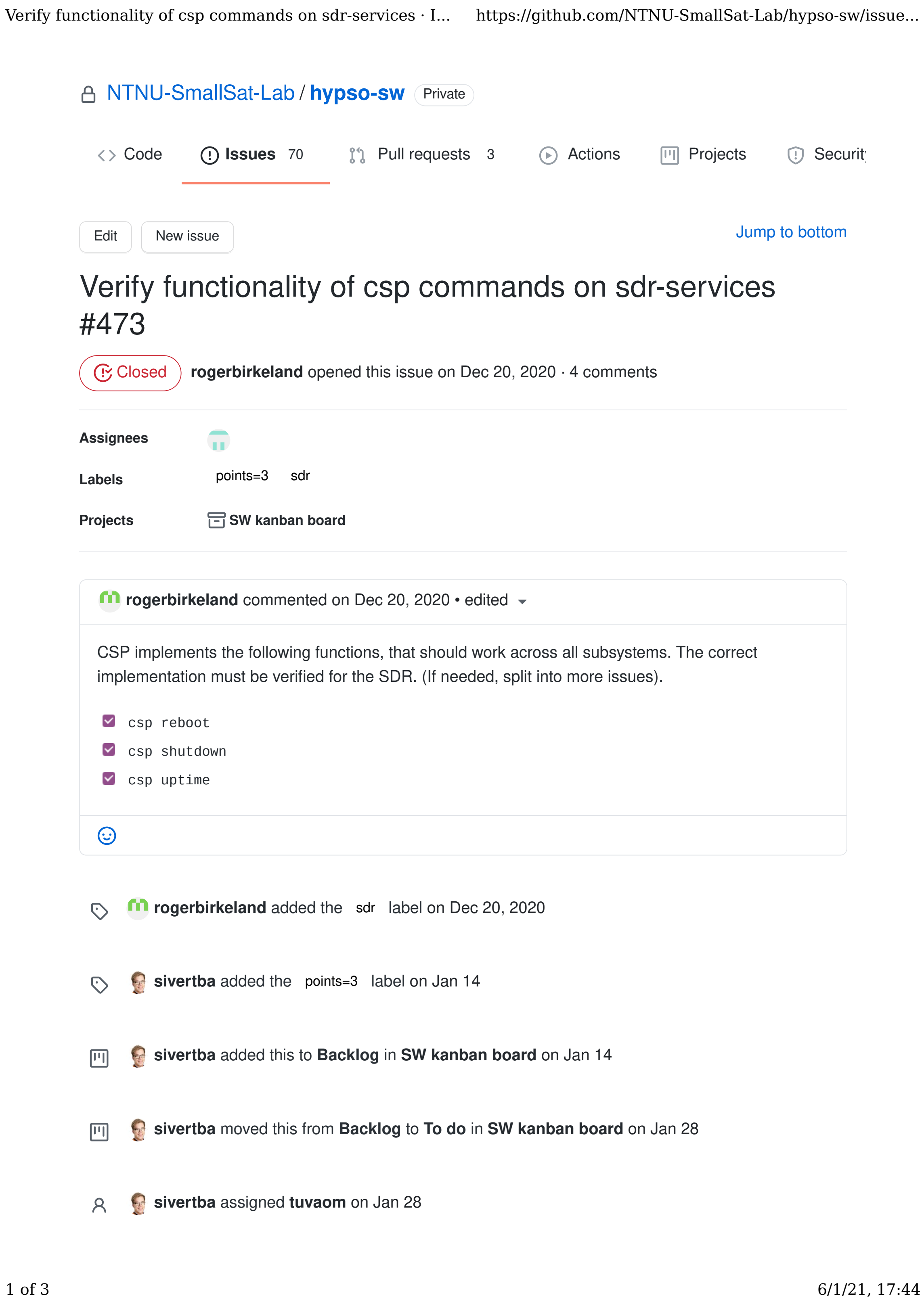
Linked pull requests

Successfully merging a pull request may close this issue.

None yet

2 participants

C Issue #473 hypso-sw



Verify functionality of csp commands on sdr-services · I... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

Verify functionality of csp commands on sdr-services #473

Closed **rogerbirkeland** opened this issue on Dec 20, 2020 · 4 comments

Assignees

Labels points=3 sdr

Projects **SW kanban board**

rogerbirkeland commented on Dec 20, 2020 • edited

CSP implements the following functions, that should work across all subsystems. The correct implementation must be verified for the SDR. (If needed, split into more issues).

- ☑ `csp reboot`
- ☑ `csp shutdown`
- ☑ `csp uptime`

rogerbirkeland added the sdr label on Dec 20, 2020

sivertba added the points=3 label on Jan 14

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

sivertba moved this from **Backlog** to **To do** in **SW kanban board** on Jan 28

sivertba assigned **tuvaom** on Jan 28

1 of 3 6/1/21, 17:44

rogerbirkeland commented on Mar 8 Author

Seem to work when `sdr-services` is run as root.

tuvaom commented on Mar 23

Still get error messages on `csp reboot` and `csp shutdown` , even when run from root.

```
apps/sdr_services.c:80:csp_services_task: CSP Services conn: 0x40a10
Failed to reboot: No error information
apps/sdr_services.c:80:csp_services_task: CSP Services conn: 0x40a48
Failed to shutdown: No error information
```

rogerbirkeland commented on Mar 24 Author

Works when the `sdr-services` is started automatically, but not when it is started through the startup script.

rogerbirkeland commented on Mar 24 Author

Closed due to #520

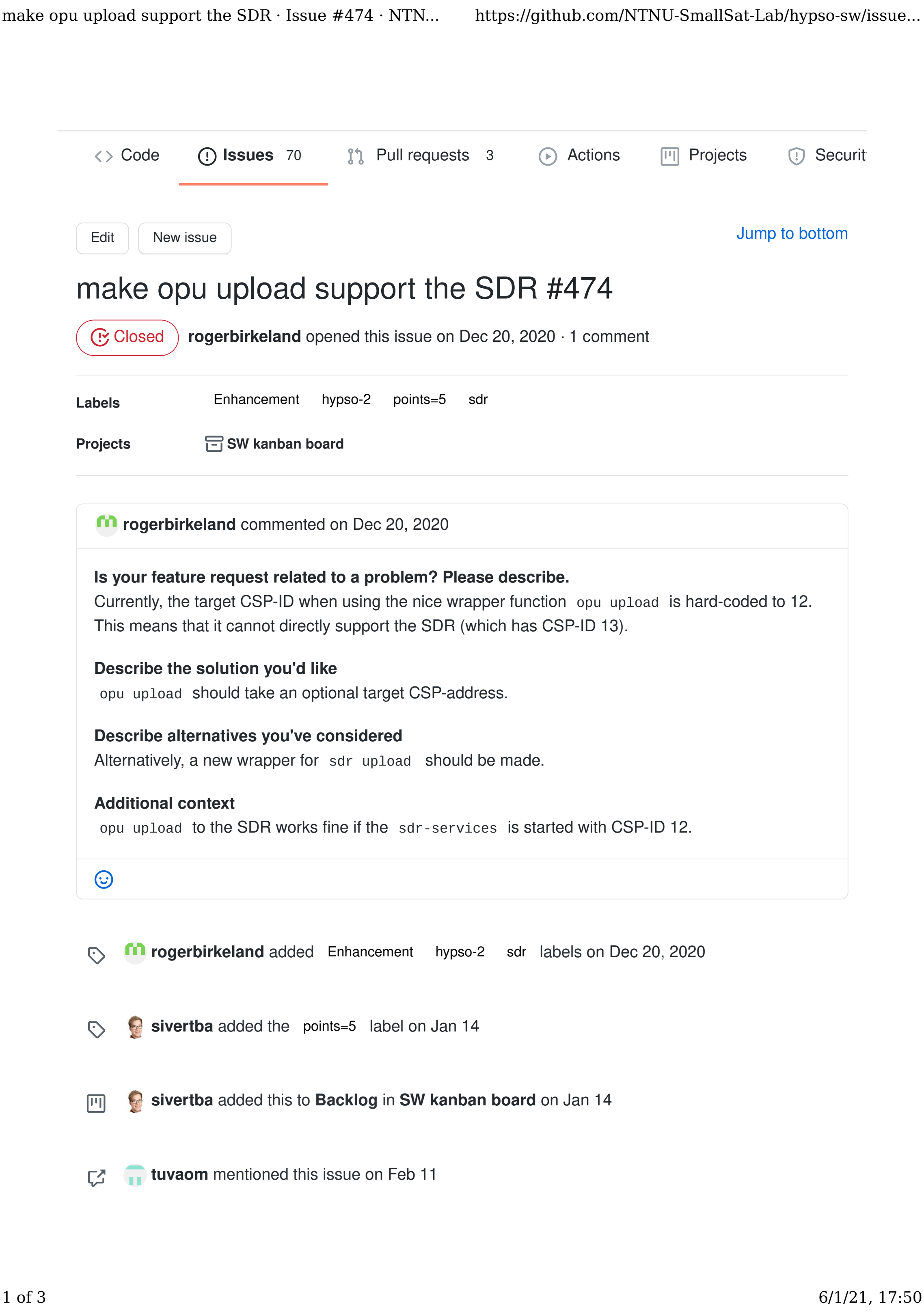
rogerbirkeland closed this on Mar 24

SW kanban board automation moved this from **To do** to **Done** on Mar 24

Assignees

tuvaom

D Issue #474 & #475 hypso-sw



make opu upload support the SDR · Issue #474 · NTN... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

make opu upload support the SDR #474

Closed **rogerbirkeland** opened this issue on Dec 20, 2020 · 1 comment

Labels Enhancement hypso-2 points=5 sdr

Projects **SW kanban board**

rogerbirkeland commented on Dec 20, 2020

Is your feature request related to a problem? Please describe.
Currently, the target CSP-ID when using the nice wrapper function `opu upload` is hard-coded to 12. This means that it cannot directly support the SDR (which has CSP-ID 13).

Describe the solution you'd like
`opu upload` should take an optional target CSP-address.

Describe alternatives you've considered
Alternatively, a new wrapper for `sdr upload` should be made.

Additional context
`opu upload` to the SDR works fine if the `sdr-services` is started with CSP-ID 12.

rogerbirkeland added Enhancement hypso-2 sdr labels on Dec 20, 2020

sivertba added the points=5 label on Jan 14

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

tuvaom mentioned this issue on Feb 11

1 of 3 6/1/21, 17:50

[redacted] mentioned this issue on Mar 17

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

tuvaom commented on Mar 17

closed by #506

tuvaom closed this on Mar 17

SW kanban board automation moved this from **Backlog** to **Done** on Mar 17

Assignees

No one—assign yourself

Labels

Enhancement hypso-2 points=5 sdr

Projects

SW kanban board

Done ▾

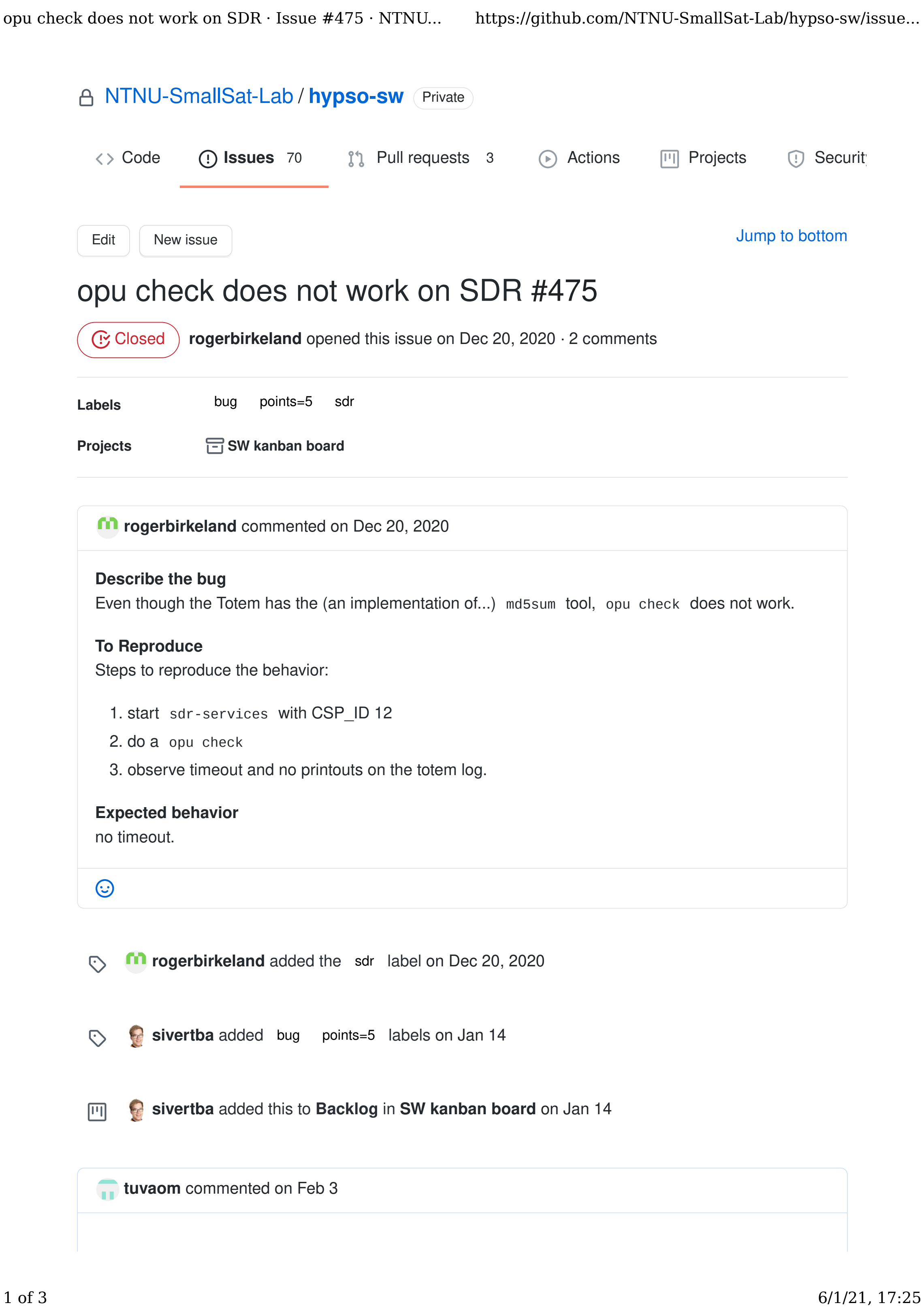
Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet



opu check does not work on SDR · Issue #475 · NTNU... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

opu check does not work on SDR #475

Closed **rogerbirkeland** opened this issue on Dec 20, 2020 · 2 comments

Labels bug points=5 sdr

Projects **SW kanban board**

rogerbirkeland commented on Dec 20, 2020

Describe the bug
Even though the Totem has the (an implementation of...) `md5sum` tool, `opu check` does not work.

To Reproduce
Steps to reproduce the behavior:

1. start `sdr-services` with CSP_ID 12
2. do a `opu check`
3. observe timeout and no printouts on the totem log.

Expected behavior
no timeout.

rogerbirkeland added the sdr label on Dec 20, 2020

sivertba added bug points=5 labels on Jan 14

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

tuvaom commented on Feb 3

1 of 3 6/1/21, 17:25

timeout because opu check is a part of the telemetry-service thread which does not run on the SDR. (replace it?)

tuvaom mentioned this issue on Feb 11

tm_service.c -> tm_service.c + tm_opu.c + tm_sdr.c #494

Closed

tuvaom mentioned this issue on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

tuvaom commented on Mar 18

closed by #506

tuvaom closed this on Mar 18

SW kanban board automation moved this from **Backlog** to **Done** on Mar 18

Assignees

No one—assign yourself

Labels

bug points=5 sdr

Projects

SW kanban board

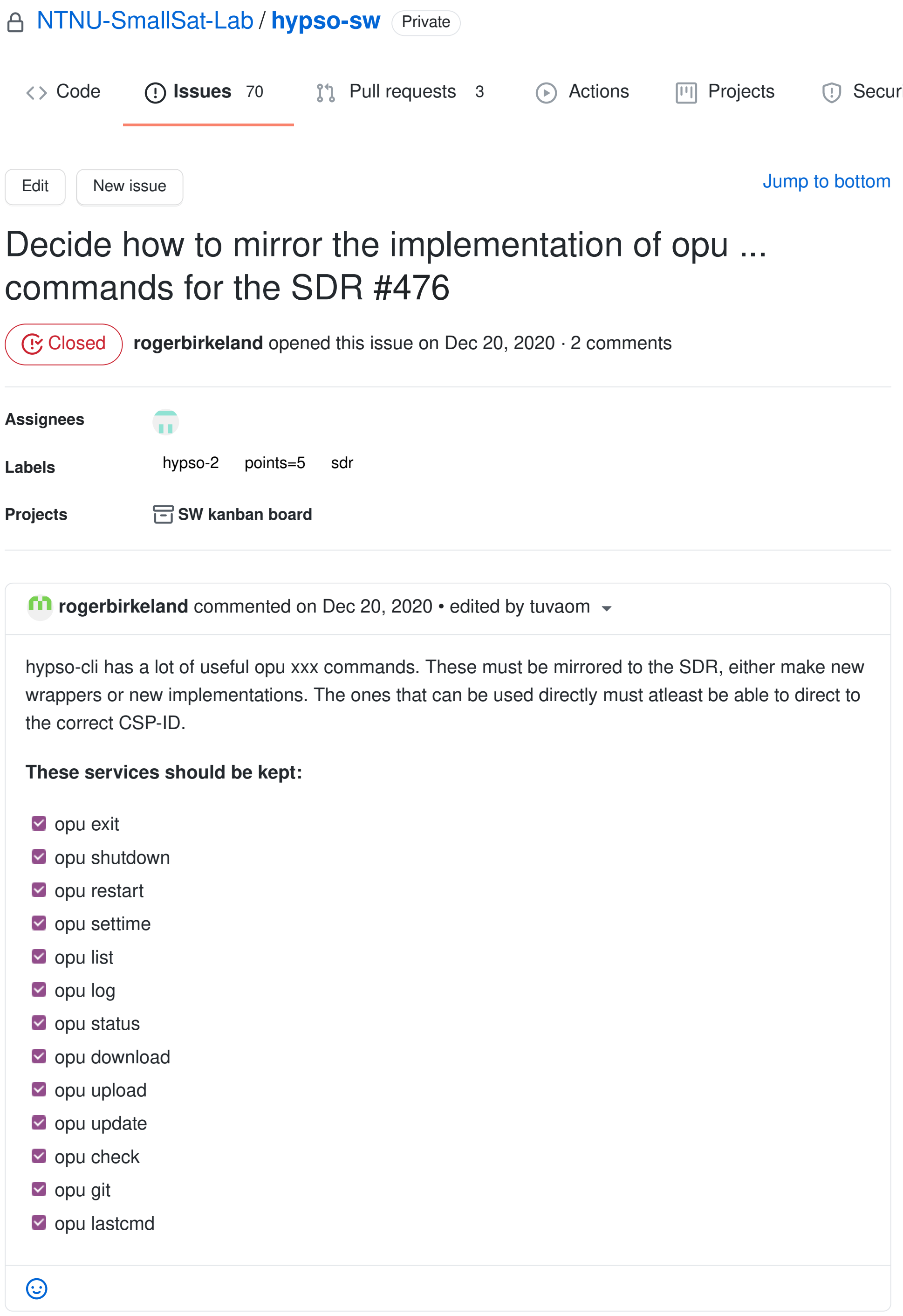
Done ▾

Milestone

No milestone

E Issue #476 hypso-sw

Decide how to mirror the implementation of opu ... c... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...



NTNU-SmallSat-Lab / **hypso-sw** Private

<> Code **Issues** 70 Pull requests 3 Actions Projects Securit

Edit New issue Jump to bottom

Decide how to mirror the implementation of opu ... commands for the SDR #476

Closed **rogerbirkeland** opened this issue on Dec 20, 2020 · 2 comments

Assignees

Labels hypso-2 points=5 sdr

Projects **SW kanban board**

rogerbirkeland commented on Dec 20, 2020 • edited by tuvaom

hypso-cli has a lot of useful opu xxx commands. These must be mirrored to the SDR, either make new wrappers or new implementations. The ones that can be used directly must atleast be able to direct to the correct CSP-ID.

These services should be kept:

- ☑ opu exit
- ☑ opu shutdown
- ☑ opu restart
- ☑ opu settime
- ☑ opu list
- ☑ opu log
- ☑ opu status
- ☑ opu download
- ☑ opu upload
- ☑ opu update
- ☑ opu check
- ☑ opu git
- ☑ opu lastcmd

1 of 3 6/1/21, 17:44

rogerbirkeland added hypso-2 sdr labels on Dec 20, 2020

sivertba commented on Jan 14

related to #477

sivertba added the points=5 label on Jan 14

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

sivertba moved this from **Backlog** to **To do** in **SW kanban board** on Jan 28

sivertba assigned **tuvaom** on Jan 28

sivertba moved this from **To do** to **In progress** in **SW kanban board** on Feb 2

tuvaom mentioned this issue on Feb 11

cli_opu -> cli_opu + cli_sdr + cli_pl #493

Closed

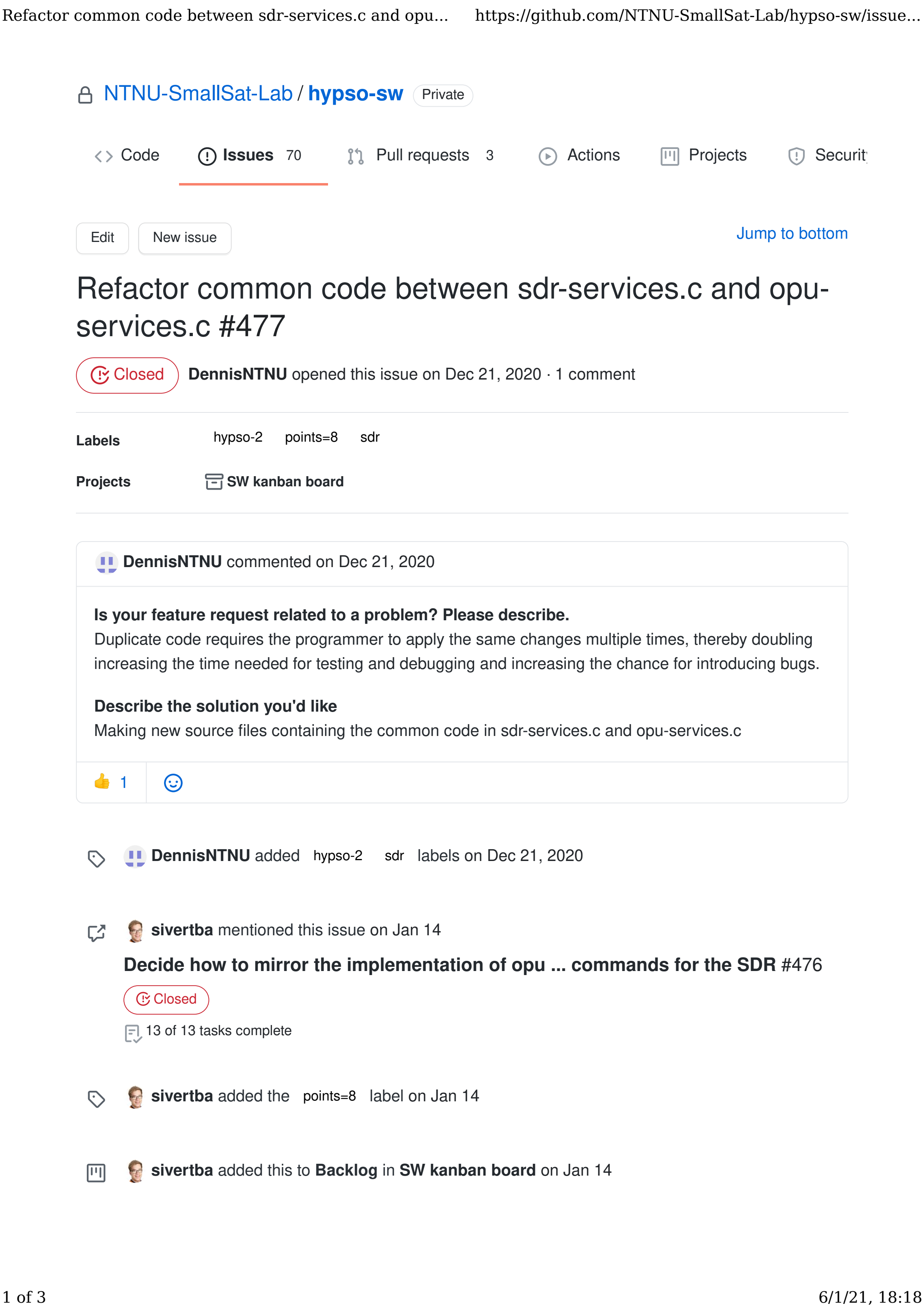
sivertba moved this from **In progress** to **Review in progress** in **SW kanban board** on Feb 25

tuvaom commented on Feb 25

Decided through issues:

- #493 (exit, status, download, upload, update, lastcmd)
- #494(check, git)
- #495 (list)
- #503(shutdown)
- #504(restart)
- #502(settime)
- #505(log)

F Issue #477 hypso-sw



Refactor common code between sdr-services.c and opu... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

Refactor common code between sdr-services.c and opu-services.c #477

Closed **DennisNTNU** opened this issue on Dec 21, 2020 · 1 comment

Labels hypso-2 points=8 sdr

Projects **SW kanban board**

DennisNTNU commented on Dec 21, 2020

Is your feature request related to a problem? Please describe.
Duplicate code requires the programmer to apply the same changes multiple times, thereby doubling increasing the time needed for testing and debugging and increasing the chance for introducing bugs.

Describe the solution you'd like
Making new source files containing the common code in sdr-services.c and opu-services.c

👍 1

DennisNTNU added hypso-2 sdr labels on Dec 21, 2020

sivertba mentioned this issue on Jan 14

Decide how to mirror the implementation of opu ... commands for the SDR #476
Closed
13 of 13 tasks complete

sivertba added the points=8 label on Jan 14

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

1 of 3 6/1/21, 18:18

sivertba moved this from **Backlog** to **To do** in **SW kanban board** on Jan 14

sivertba moved this from **To do** to **Backlog** in **SW kanban board** on Jan 14

sivertba moved this from **Backlog** to **To do** in **SW kanban board** on Jan 28

sivertba moved this from **To do** to **Backlog** in **SW kanban board** on Jan 28

sivertba moved this from **Backlog** to **In progress** in **SW kanban board** on Mar 23

tuvaom mentioned this issue on Apr 12

Refactoring opu_services.c and sdr_services.c #527

Merged

tuvaom commented on Apr 13

closed by #527

tuvaom closed this on Apr 13

SW kanban board automation moved this from **In progress** to **Done** on Apr 13

tuvaom mentioned this issue on Apr 14

Sdr refactor (opu_services.c and sdr_services.c) #529

Merged

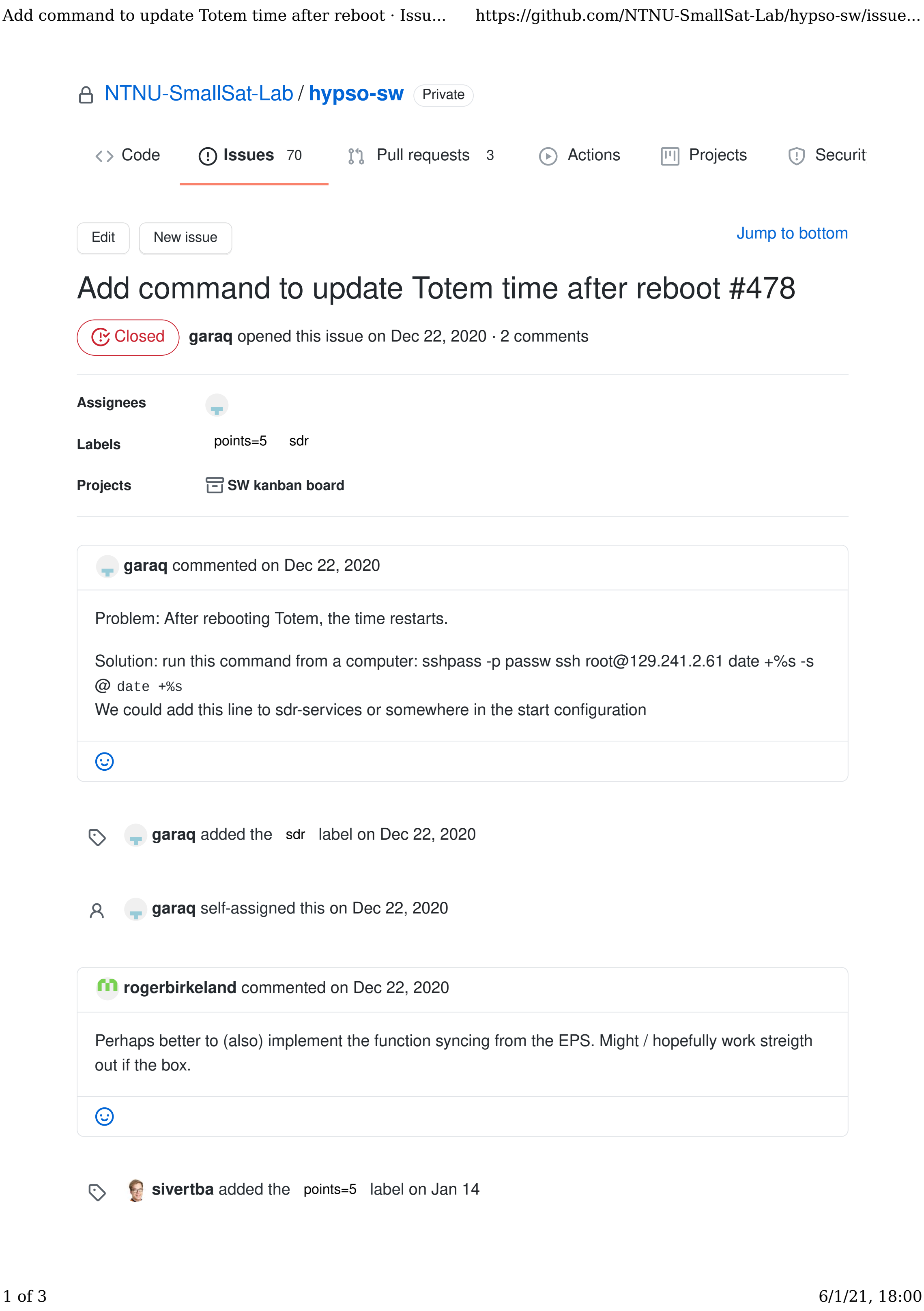
Assignees

No one—assign yourself

Labels

hypso-2 points=8 sdr

G Issue #478 hypso-sw



Add command to update Totem time after reboot · Issu... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

Add command to update Totem time after reboot #478

Closed **garaq** opened this issue on Dec 22, 2020 · 2 comments

Assignees

Labels points=5 sdr

Projects SW kanban board

garaq commented on Dec 22, 2020

Problem: After rebooting Totem, the time restarts.

Solution: run this command from a computer: sshpass -p passw ssh root@129.241.2.61 date +%s -s
@ `date +%s`
We could add this line to sdr-services or somewhere in the start configuration

garaq added the sdr label on Dec 22, 2020

garaq self-assigned this on Dec 22, 2020

rogerbirkeland commented on Dec 22, 2020

Perhaps better to (also) implement the function syncing from the EPS. Might / hopefully work streigth out if the box.

sivertba added the points=5 label on Jan 14

1 of 3 6/1/21, 18:00

sivertba added this to **Backlog** in **SW kanban board** on Jan 14

sivertba moved this from **Backlog** to **In progress** in **SW kanban board** on Mar 23

tuvaom mentioned this issue on Mar 23

Run sdr-services from root to access functionality #520

Merged

rogerbirkeland mentioned this issue on Mar 23

Making Totem startupscript NTNU-SmallSat-Lab/sdr-system#8

Merged

garaq commented on Mar 24 Author

It works.

rogerbirkeland closed this in NTNU-SmallSat-Lab/sdr-system#8 on Mar 24

SW kanban board automation moved this from **In progress** to **Done** on Mar 24

Assignees

garaq

Labels

points=5 sdr

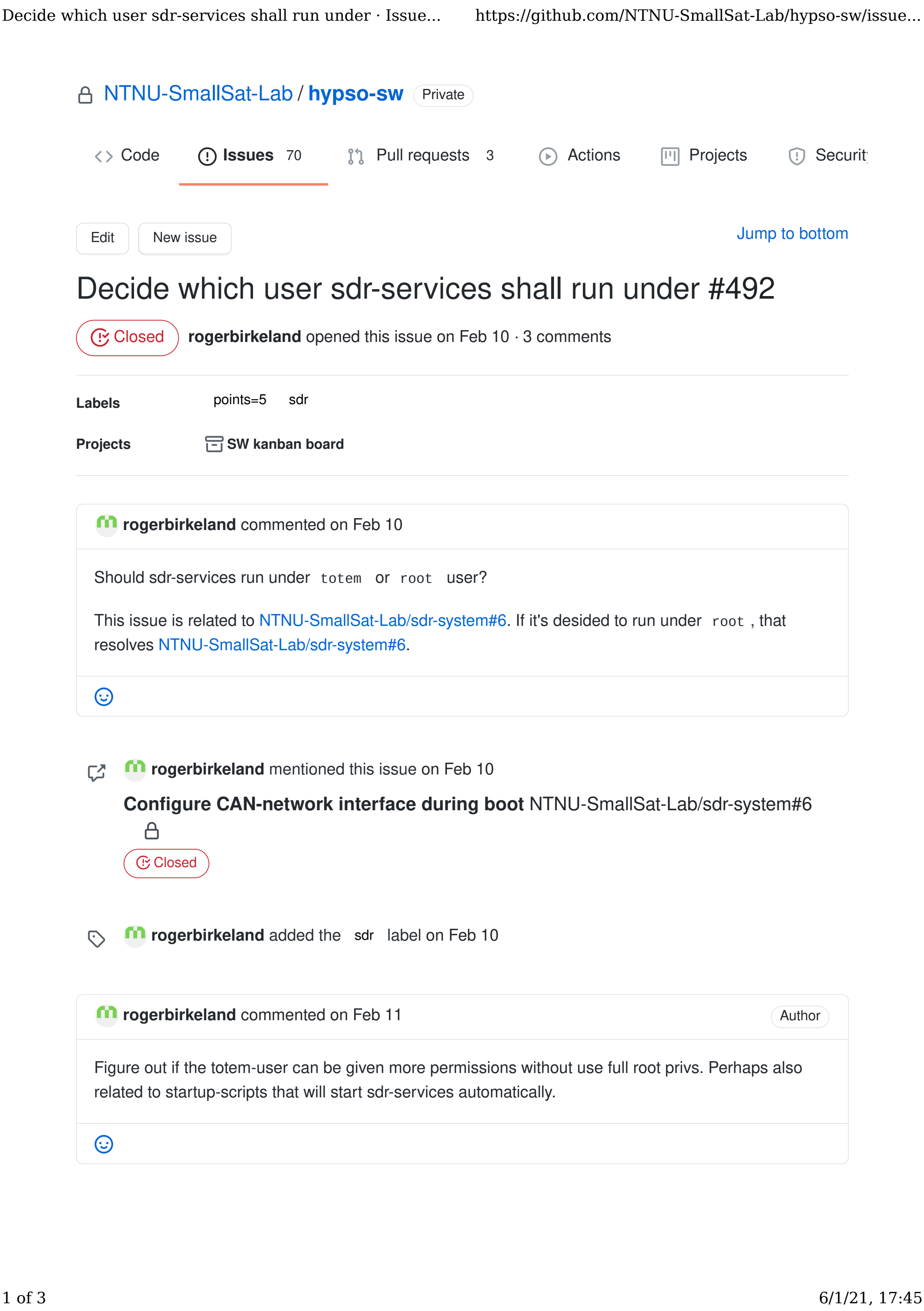
Projects

SW kanban board

Done ▾

Milestone

H Issue #492 hypso-sw



Decide which user sdr-services shall run under · Issue... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

Decide which user sdr-services shall run under #492

Closed **rogerbirkeland** opened this issue on Feb 10 · 3 comments

Labels points=5 sdr

Projects **SW kanban board**

rogerbirkeland commented on Feb 10

Should sdr-services run under `totem` or `root` user?

This issue is related to NTNU-SmallSat-Lab/sdr-system#6. If it's desided to run under `root`, that resolves NTNU-SmallSat-Lab/sdr-system#6.

rogerbirkeland mentioned this issue on Feb 10

Configure CAN-network interface during boot NTNU-SmallSat-Lab/sdr-system#6

Closed

rogerbirkeland added the sdr label on Feb 10

rogerbirkeland commented on Feb 11 Author

Figure out if the totem-user can be given more permissions without use full root privs. Perhaps also related to startup-scripts that will start sdr-services automatically.

1 of 3 6/1/21, 17:45

jlgarrett added this to **Backlog** in **SW kanban board** on Feb 11

jlgarrett added the points=5 label on Feb 11

rogerbirkeland mentioned this issue on Feb 16

Make start-up-scripts for Totem NTNU-SmallSat-Lab/sdr-system#7

Closed

rogerbirkeland commented on Mar 2 Author

Input from Alen Space:

> Regarding running as non-root, it is as you said more secure and helps you identify mistakes as you have more control over what can or cannot be done. For allowing non-root users to access i.e. CAN devices, the user must have proper permissions to the devices in /dev, usually setting device files as owned by a group and making the non-root user part of this group.
> My recommendation for proper non-user device usage is to change the /dev management to mdev and use the /etc/mdev.conf file to set specific groups and permissions to each device (i.e. /dev/can0). Then, change the user_table.txt file to add the user to these groups. Please see the buildroot manual for this, as it offers extensive documentation, and if you need any help you can ask me again.
> https://buildroot.org/downloads/manual/manual.html#_dev_management

tuvaom changed the title ~~**Deside which user sdr-services shall run under**~~ **Decide which user sdr-services shall run under** on Mar 17

sivertba moved this from **Backlog** to **Review in progress** in **SW kanban board** on Mar 23

tuvaom mentioned this issue on Mar 23

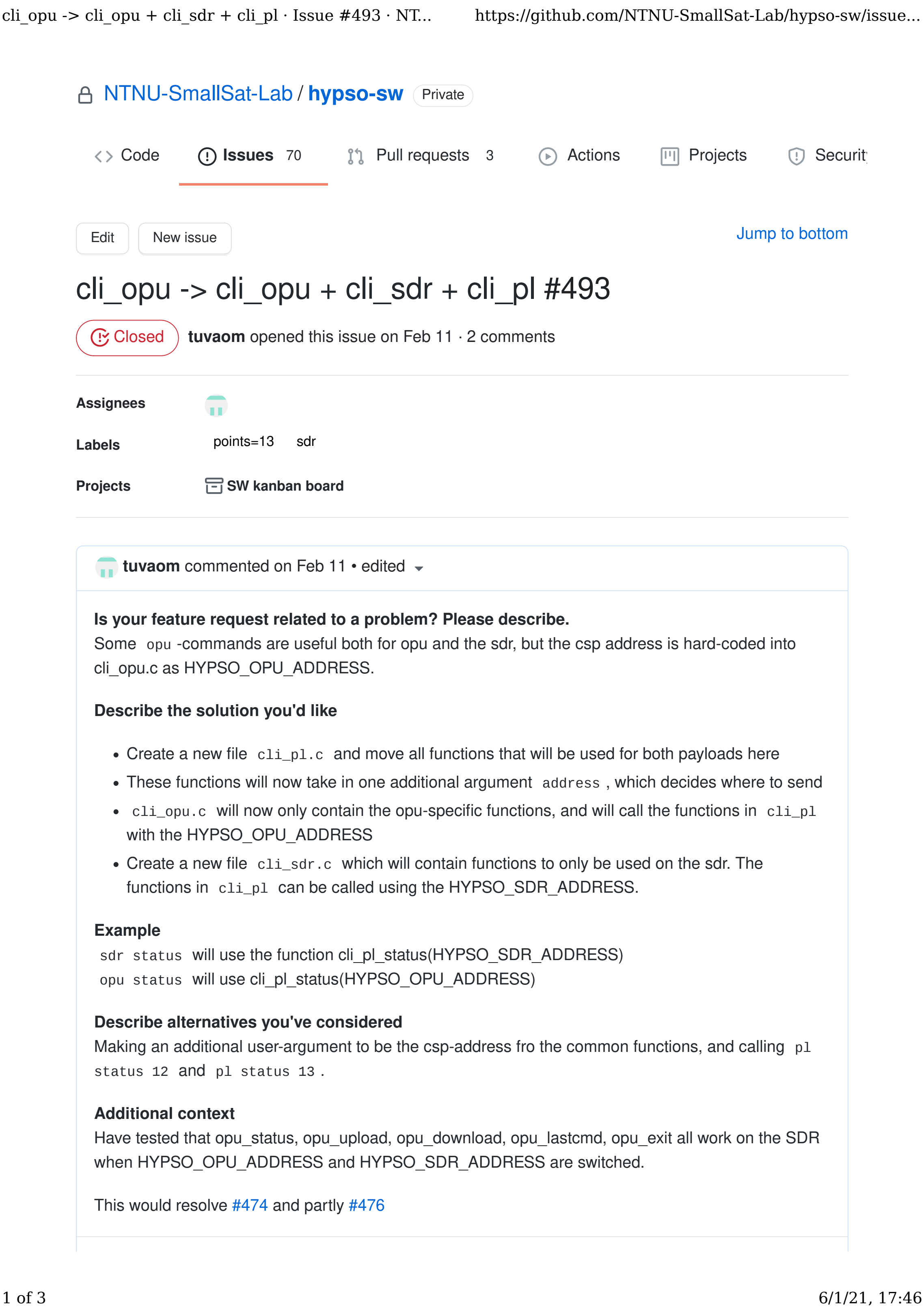
Run sdr-services from root to access functionality #520

Merged

tuvaom commented on Mar 24

closed by #520

I Issue #493 & #494 hypso-sw



cli_opu -> cli_opu + cli_sdr + cli_pl · Issue #493 · NT... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

cli_opu -> cli_opu + cli_sdr + cli_pl #493

Closed **tuvaom** opened this issue on Feb 11 · 2 comments

Assignees

Labels points=13 sdr

Projects **SW kanban board**

tuvaom commented on Feb 11 • edited

Is your feature request related to a problem? Please describe.
Some `opu` -commands are useful both for opu and the sdr, but the csp address is hard-coded into cli_opu.c as HYPSO_OPU_ADDRESS.

Describe the solution you'd like

- Create a new file `cli_pl.c` and move all functions that will be used for both payloads here
- These functions will now take in one additional argument `address` , which decides where to send
- `cli_opu.c` will now only contain the opu-specific functions, and will call the functions in `cli_pl` with the HYPSO_OPU_ADDRESS
- Create a new file `cli_sdr.c` which will contain functions to only be used on the sdr. The functions in `cli_pl` can be called using the HYPSO_SDR_ADDRESS.

Example
`sdr status` will use the function cli_pl_status(HYPSO_SDR_ADDRESS)
`opu status` will use cli_pl_status(HYPSO_OPU_ADDRESS)

Describe alternatives you've considered
Making an additional user-argument to be the csp-address fro the common functions, and calling `pl status 12` and `pl status 13` .

Additional context
Have tested that opu_status, opu_upload, opu_download, opu_lastcmd, opu_exit all work on the SDR when HYPSO_OPU_ADDRESS and HYPSO_SDR_ADDRESS are switched.

This would resolve #474 and partly #476

1 of 3 6/1/21, 17:46

tuvaom added the sdr label on Feb 11

tuvaom changed the title ~~**cli_opu -> cli_opu + cli_sdr + cli_payload**~~ **cli_opu -> cli_opu + cli_sdr + cli_pl** on Feb 11

rogerbirkeland commented on Feb 11

Consider both ways: `sdr/opu statu` s and `pl status x`

rogerbirkeland added the points=13 label on Feb 11

jlgarrett added this to **Backlog** in **SW kanban board** on Feb 11

jlgarrett moved this from **Backlog** to **To do** in **SW kanban board** on Feb 11

tuvaom self-assigned this on Feb 11

sivertba moved this from **To do** to **In progress** in **SW kanban board** on Feb 23

sivertba moved this from **In progress** to **Review in progress** in **SW kanban board** on Feb 25

This was referenced on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

Decide how to mirror the implementation of opu ... commands for the SDR #476

Closed

tuvaom commented on Mar 18 Author

closed by #506

tuvaom closed this on Mar 18

SW kanban board automation moved this from **Review in progress** to **Done** on Mar 18

Assignees

tuvaom

Labels

points=13 sdr

Projects

SW kanban board

Done ▾

Milestone

No milestone

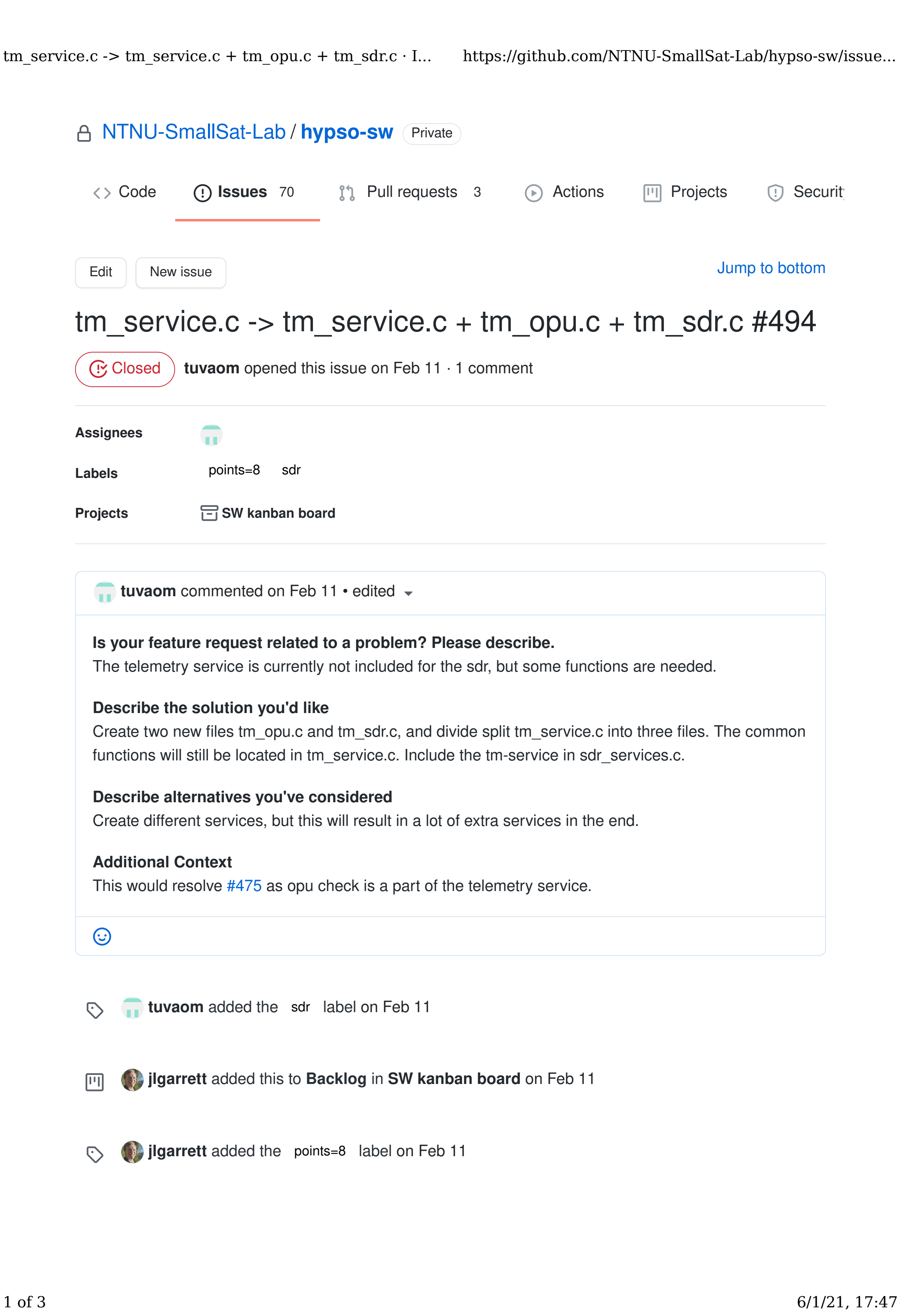
Linked pull requests

Successfully merging a pull request may close this issue.

None yet

2 participants

Pin issue ⓘ



tm_service.c -> tm_service.c + tm_opu.c + tm_sdr.c · I...   https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

tm_service.c -> tm_service.c + tm_opu.c + tm_sdr.c #494

Closed **tuvaom** opened this issue on Feb 11 · 1 comment

Assignees

Labels points=8 sdr

Projects SW kanban board

tuvaom commented on Feb 11 • edited

Is your feature request related to a problem? Please describe.
The telemetry service is currently not included for the sdr, but some functions are needed.

Describe the solution you'd like
Create two new files tm_opu.c and tm_sdr.c, and divide split tm_service.c into three files. The common functions will still be located in tm_service.c. Include the tm-service in sdr_services.c.

Describe alternatives you've considered
Create different services, but this will result in a lot of extra services in the end.

Additional Context
This would resolve #475 as opu check is a part of the telemetry service.

tuvaom added the sdr label on Feb 11

jlgarrett added this to **Backlog** in **SW kanban board** on Feb 11

jlgarrett added the points=8 label on Feb 11

1 of 3   6/1/21, 17:47

jlgarrett moved this from **Backlog** to **To do** in **SW kanban board** on Feb 11

tuvaom self-assigned this on Feb 11

sivertba moved this from **To do** to **In progress** in **SW kanban board** on Feb 23

sivertba moved this from **In progress** to **Review in progress** in **SW kanban board** on Feb 25

This was referenced on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

Decide how to mirror the implementation of opu ... commands for the SDR #476

Closed

tuvaom commented on Mar 18 Author

closed by #506

tuvaom closed this on Mar 18

SW kanban board automation moved this from **Review in progress** to **Done** on Mar 18

Assignees

tuvaom

Labels

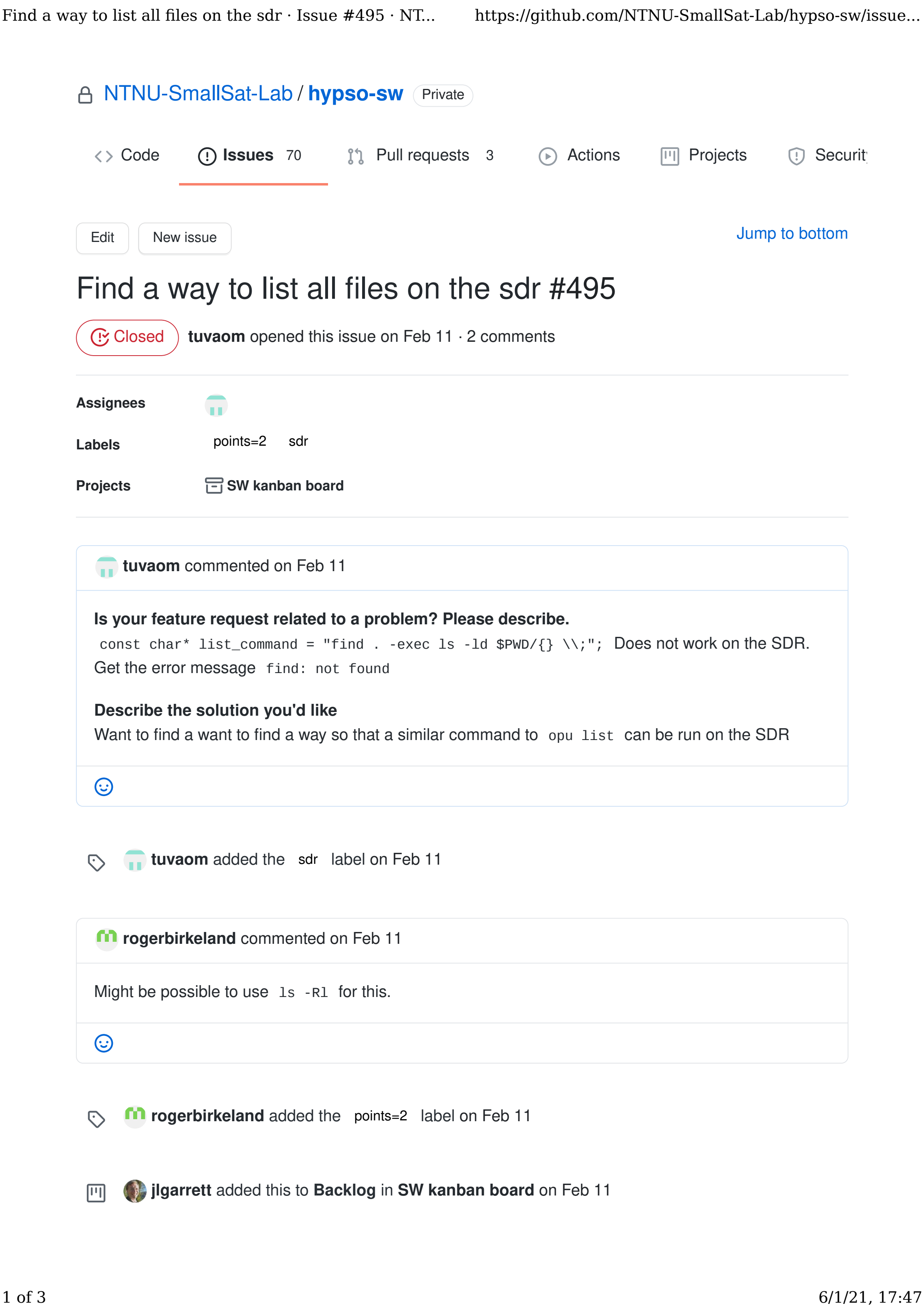
points=8 sdr

Projects

SW kanban board

Done ▾

J Issue #495 hypso-sw



Find a way to list all files on the sdr · Issue #495 · NT... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

Find a way to list all files on the sdr #495

Closed **tuvaom** opened this issue on Feb 11 · 2 comments

Assignees

Labels points=2 sdr

Projects **SW kanban board**

tuvaom commented on Feb 11

Is your feature request related to a problem? Please describe.
`const char* list_comand = "find . -exec ls -ld $PWD/{} \\;";` Does not work on the SDR. Get the error message `find: not found`

Describe the solution you'd like
Want to find a want to find a way so that a similar command to `opu list` can be run on the SDR

tuvaom added the sdr label on Feb 11

rogerbirkeland commented on Feb 11

Might be possible to use `ls -Rl` for this.

rogerbirkeland added the points=2 label on Feb 11

jlgarrett added this to **Backlog** in **SW kanban board** on Feb 11

1 of 3 6/1/21, 17:47

tuvaom self-assigned this on Feb 25

sivertba moved this from **Backlog** to **Review in progress** in **SW kanban board** on Feb 25

This was referenced on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

Decide how to mirror the implementation of opu ... commands for the SDR #476

Closed

tuvaom commented on Mar 18 Author

closed by #506

tuvaom closed this on Mar 18

SW kanban board automation moved this from **Review in progress** to **Done** on Mar 18

Assignees

tuvaom

Labels

points=2 sdr

Projects

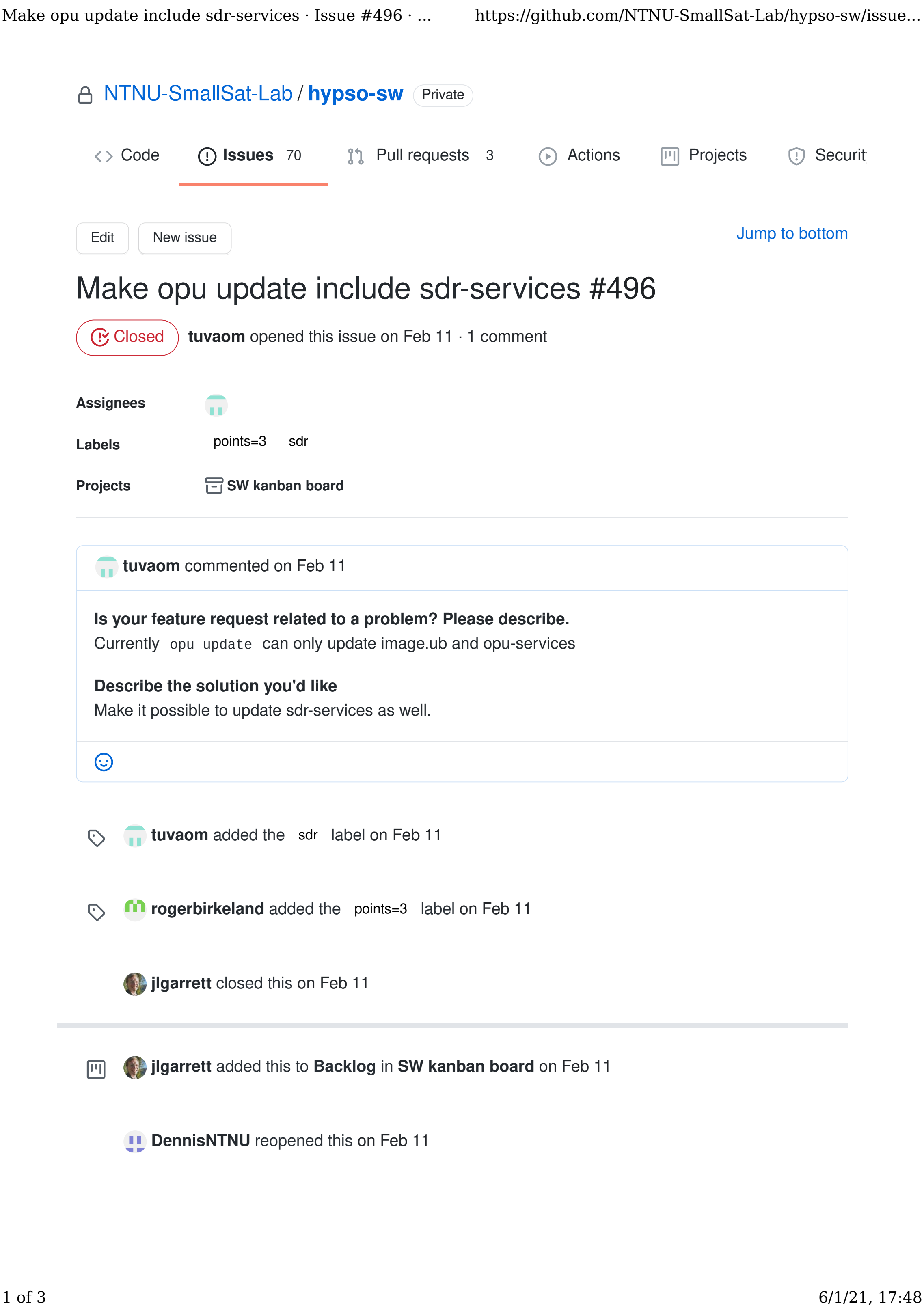
SW kanban board

Done ▾

Milestone

No milestone

K Issue #496 hypso-sw



Make opu update include sdr-services · Issue #496 · ... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

Make opu update include sdr-services #496

Closed **tuvaom** opened this issue on Feb 11 · 1 comment

Assignees

Labels points=3 sdr

Projects **SW kanban board**

tuvaom commented on Feb 11

Is your feature request related to a problem? Please describe.
Currently `opu update` can only update image.ub and opu-services

Describe the solution you'd like
Make it possible to update sdr-services as well.

tuvaom added the sdr label on Feb 11

rogerbirkeland added the points=3 label on Feb 11

jlgarrett closed this on Feb 11

jlgarrett added this to **Backlog** in **SW kanban board** on Feb 11

DennisNTNU reopened this on Feb 11

1 of 3 6/1/21, 17:48

jlgarrett moved this from **Backlog** to **To do** in **SW kanban board** on Feb 11

tuvaom self-assigned this on Feb 11

sivertba moved this from **To do** to **Review in progress** in **SW kanban board** on Feb 25

tuvaom mentioned this issue on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

tuvaom commented on Mar 18 Author

closed by #506

tuvaom closed this on Mar 18

SW kanban board automation moved this from **Review in progress** to **Done** on Mar 18

Assignees

tuvaom

Labels

points=3 sdr

Projects

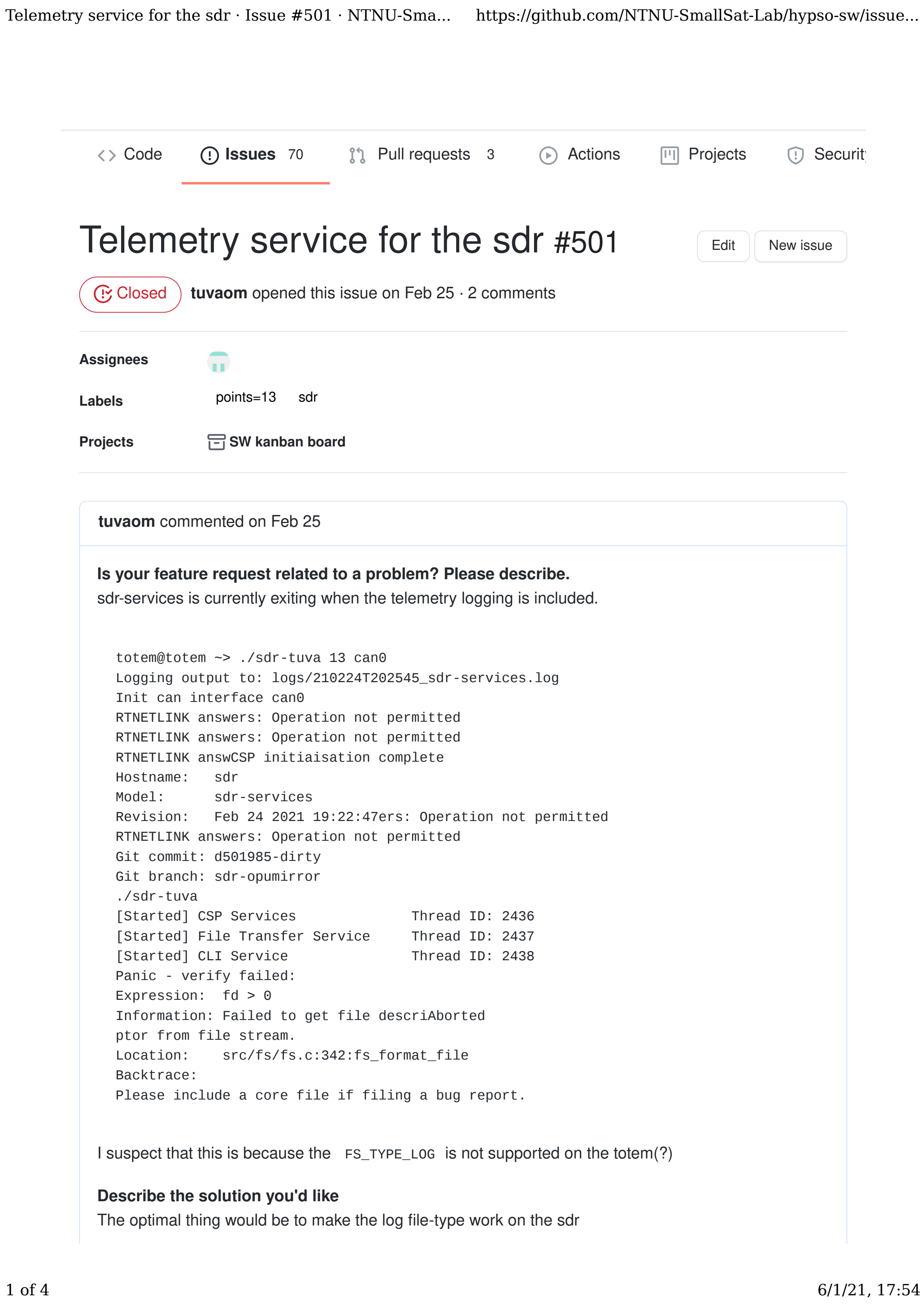
SW kanban board

Done ▾

Milestone

No milestone

L Issue #501 hypso-sw



Telemetry service for the sdr #501

Closed **tuvaom** opened this issue on Feb 25 · 2 comments

Assignees

Labels points=13 sdr

Projects **SW kanban board**

tuvaom commented on Feb 25

Is your feature request related to a problem? Please describe.
sdr-services is currently exiting when the telemetry logging is included.

```
totem@totem ~> ./sdr-tuva 13 can0
Logging output to: logs/210224T202545_sdr-services.log
Init can interface can0
RTNETLINK answers: Operation not permitted
RTNETLINK answers: Operation not permitted
RTNETLINK answCSP initiaisation complete
Hostname:   sdr
Model:      sdr-services
Revision:   Feb 24 2021 19:22:47ers: Operation not permitted
RTNETLINK answers: Operation not permitted
Git commit: d501985-dirty
Git branch: sdr-opumirror
./sdr-tuva
[Started] CSP Services              Thread ID: 2436
[Started] File Transfer Service     Thread ID: 2437
[Started] CLI Service               Thread ID: 2438
Panic - verify failed:
Expression:  fd > 0
Information: Failed to get file descriAborted
ptor from file stream.
Location:    src/fs/fs.c:342:fs_format_file
Backtrace:
Please include a core file if filing a bug report.
```

I suspect that this is because the `FS_TYPE_LOG` is not supported on the totem(?)

Describe the solution you'd like
The optimal thing would be to make the log file-type work on the sdr

Additional context
We also have to figure out what we wish to include in the telemetry for the SDR.

tuvaom added the sdr label on Feb 25

sivertba commented on Feb 25

Talk with Alen Space / Totem ppl

sivertba added the points=13 label on Feb 25

sivertba added this to **Backlog** in **SW kanban board** on Feb 25

sivertba moved this from **Backlog** to **To do** in **SW kanban board** on Feb 25

tuvaom self-assigned this on Feb 25

tuvaom mentioned this issue on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

sivertba moved this from **To do** to **In progress** in **SW kanban board** on Mar 11

tuvaom mentioned this issue on Apr 23

Sdr telemetry #538

Merged

jlgarrett moved this from **In progress** to **Review in progress** in **SW kanban board** on Apr 27

Sdr telemetry #538 Merged

tuvaom commented on Apr 29 • edited ▾ Author

closed by #538

☺

tuvaom closed this on Apr 29

SW kanban board automation moved this from **Review in progress** to **Done** on Apr 29

Assignees

tuvaom

Labels

points=13 sdr

Projects

SW kanban board

Done ▾

Milestone

No milestone

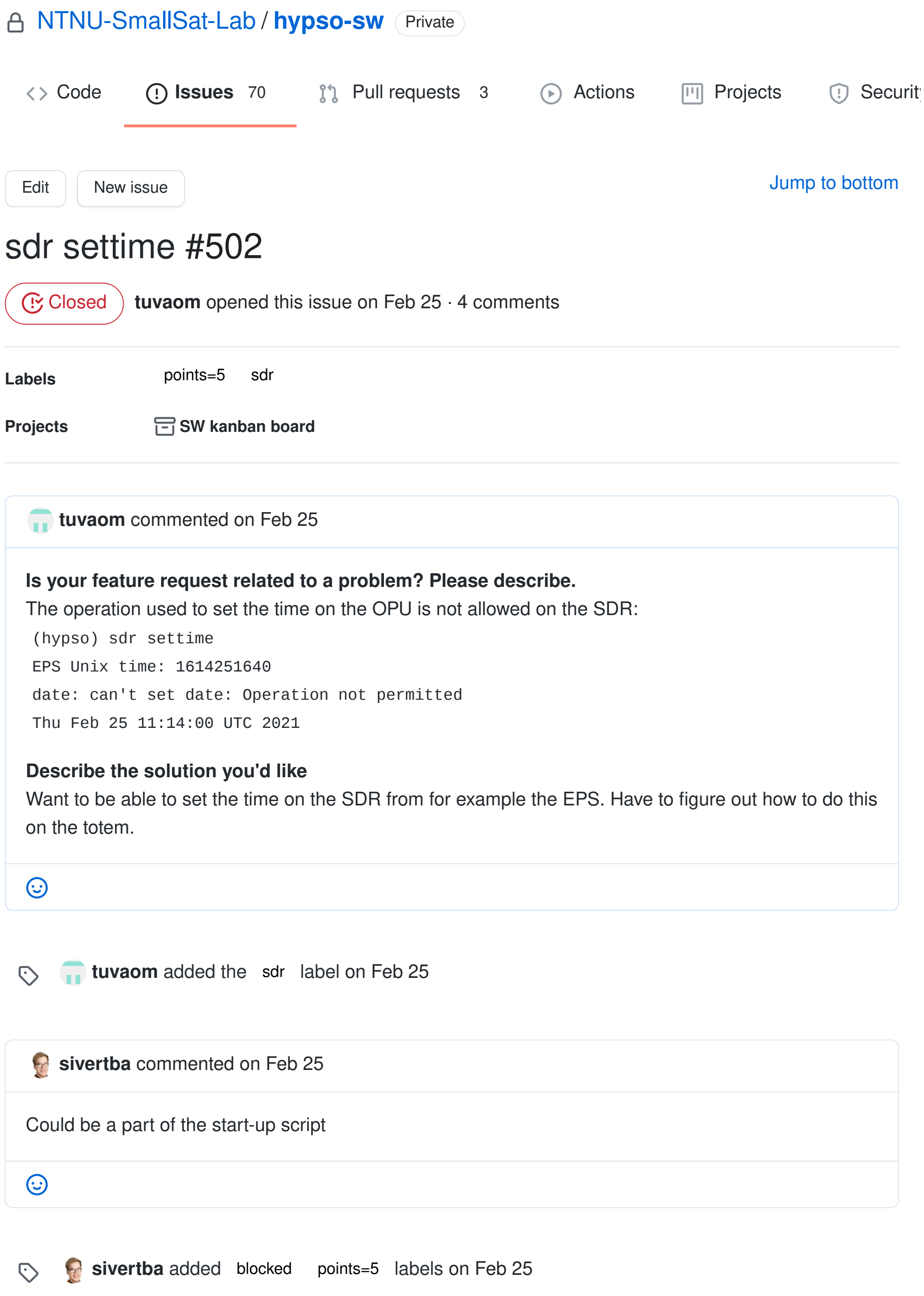
Linked pull requests

Successfully merging a pull request may close this issue.

Sdr telemetry

2 participants

M Issue #502, #503, #504 & #505 hypso-sw



sdr settime #502

Closed **tuvaom** opened this issue on Feb 25 · 4 comments

Labels points=5 sdr

Projects SW kanban board

tuvaom commented on Feb 25

Is your feature request related to a problem? Please describe.
The operation used to set the time on the OPU is not allowed on the SDR:

```
(hypso) sdr settime
EPS Unix time: 1614251640
date: can't set date: Operation not permitted
Thu Feb 25 11:14:00 UTC 2021
```

Describe the solution you'd like
Want to be able to set the time on the SDR from for example the EPS. Have to figure out how to do this on the totem.

tuvaom added the sdr label on Feb 25

sivertba commented on Feb 25

Could be a part of the start-up script

sivertba added blocked points=5 labels on Feb 25

sivertba added this to **In progress** in **SW kanban board** on Feb 25

sivertba moved this from **In progress** to **Done** in **SW kanban board** on Feb 25

sivertba moved this from **Done** to **Blocked** in **SW kanban board** on Feb 25

rogerbirkeland commented on Feb 25

Try running as root, to prove functionality?

This was referenced on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

Decide how to mirror the implementation of opu ... commands for the SDR #476

Closed

rogerbirkeland commented on Mar 1

Works when started as root.

sivertba moved this from **Blocked** to **Backlog** in **SW kanban board** on Mar 11

sivertba removed the blocked label on Mar 11

sivertba moved this from **Backlog** to **In progress** in **SW kanban board** on Mar 23

tuvaom mentioned this issue on Mar 23

Run sdr-services from root to access functionality #520

Merged

tuvaom commented on Mar 24 Author

closed by #520

tuvaom closed this on Mar 24

Assignees

SW kanban board automation moved this from **In progress** to **Done** on Mar 24

Labels

points=5 sdr

Projects

SW kanban board

Done ▾

Milestone

No milestone

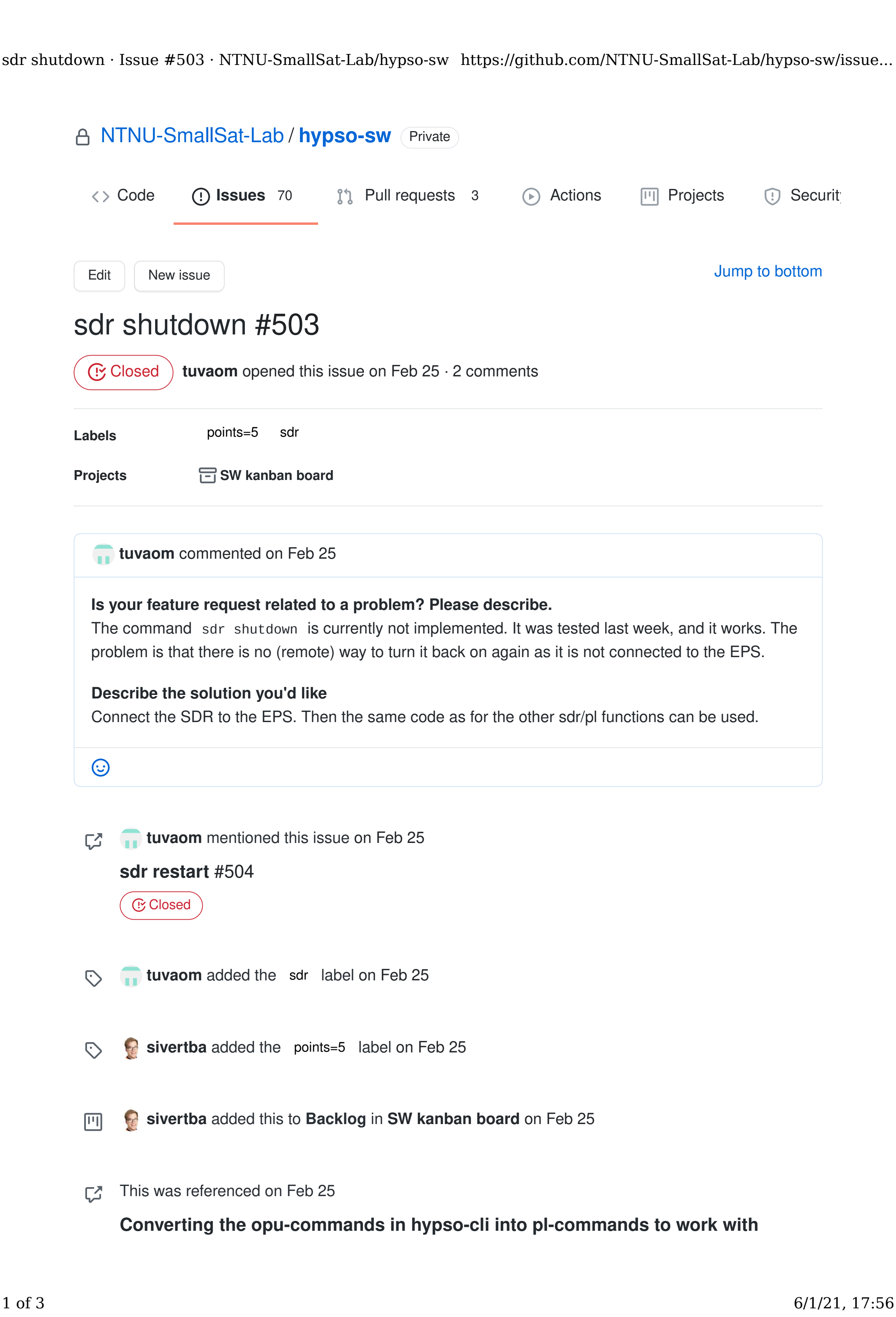
Linked pull requests

Successfully merging a pull request may close this issue.

None yet

3 participants

Pin issue ⓘ



sdr shutdown #503

Closed **tuvaom** opened this issue on Feb 25 · 2 comments

Labels points=5 sdr

Projects SW kanban board

tuvaom commented on Feb 25

Is your feature request related to a problem? Please describe.
The command `sdr shutdown` is currently not implemented. It was tested last week, and it works. The problem is that there is no (remote) way to turn it back on again as it is not connected to the EPS.

Describe the solution you'd like
Connect the SDR to the EPS. Then the same code as for the other sdr/pl functions can be used.

tuvaom mentioned this issue on Feb 25

sdr restart #504
Closed

tuvaom added the sdr label on Feb 25

sivertba added the points=5 label on Feb 25

sivertba added this to **Backlog** in **SW kanban board** on Feb 25

This was referenced on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with

multiple payloads #506
Merged

Decide how to mirror the implementation of opu ... commands for the SDR #476
Closed

sivertba moved this from **Backlog** to **In progress** in **SW kanban board** on Mar 23

tuvaom mentioned this issue on Mar 23

Run sdr-services from root to access functionality #520
Merged

garaq commented on Mar 24

It is implemented and tested. It works if sdr-services is started manually but not if on the start-up script

tuvaom commented on Mar 24 Author

closed by #520

tuvaom closed this on Mar 24

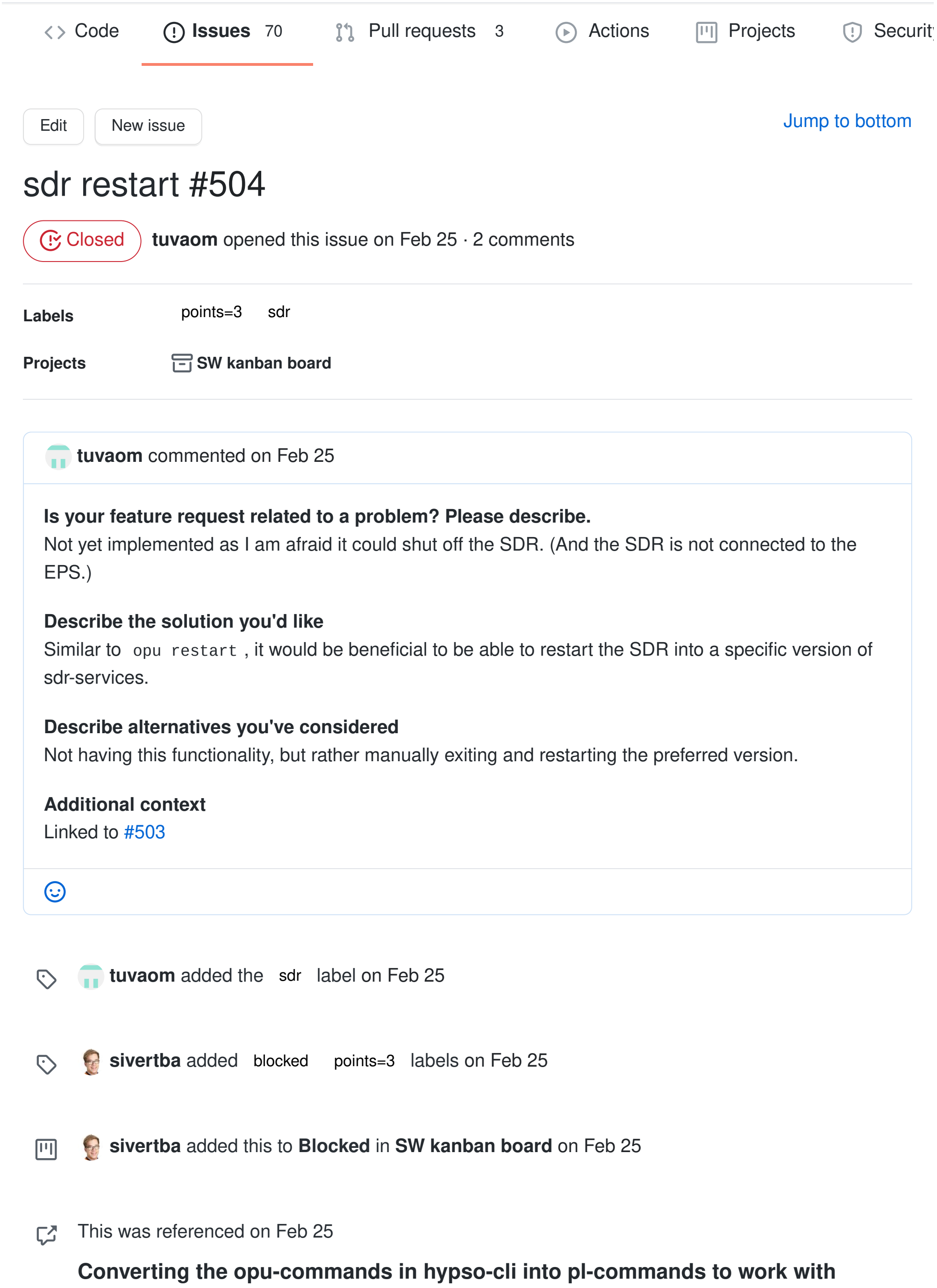
SW kanban board automation moved this from **In progress** to **Done** on Mar 24

rogerbirkeland mentioned this issue on Mar 24

Consider the implementation of csp shutdown for sdr-services #472
Closed

Assignees

No one—assign yourself



sdr restart #504

Closed **tuvaom** opened this issue on Feb 25 · 2 comments

Labels points=3 sdr

Projects SW kanban board

tuvaom commented on Feb 25

Is your feature request related to a problem? Please describe.
Not yet implemented as I am afraid it could shut off the SDR. (And the SDR is not connected to the EPS.)

Describe the solution you'd like
Similar to `opu restart`, it would be beneficial to be able to restart the SDR into a specific version of sdr-services.

Describe alternatives you've considered
Not having this functionality, but rather manually exiting and restarting the preferred version.

Additional context
Linked to #503

tuvaom added the sdr label on Feb 25

sivertba added blocked points=3 labels on Feb 25

sivertba added this to **Blocked** in **SW kanban board** on Feb 25

This was referenced on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with

Decide how to mirror the implementation of opu ... commands for the SDR #476

Closed

sivertba moved this from **Blocked** to **Backlog** in **SW kanban board** on Mar 11

sivertba removed the blocked label on Mar 11

sivertba moved this from **Backlog** to **In progress** in **SW kanban board** on Mar 23

tuvaom mentioned this issue on Mar 23

Run sdr-services from root to access functionality #520

Merged

garaq commented on Mar 24

Solved and tested. Output is:

```
apps/sdr_services.c:419:s_service_monitor_task: New service monitor conn: 0x40a0
Request to restart sdr-services received.
Starting /home/totem/hypso/sdr-tuva ...

Logging output to: logs/210324T140729_sdr-services.log
xilinx_can e0008000.can can0: bitrate error 0.0%
Init can interface can0
CSP initiaisation complete
Hostname:   sdr
Model:      sdr-services
Revision:   Mar 23 2021 11:01:02
Git commit: b80a940-dirty
Git branch: sdr-opumirror
/home/totem/hypso/sdr-tuva
[Started] CSP Services           Thread ID: 1578
[Started] File Transfer Service  Thread ID: 1579
[Started] CLI Service            Thread ID: 1580
[Started] Telemetry Service      Thread ID: 1581
```

closed by #520

tuvaom closed this on Mar 24

SW kanban board automation moved this from **In progress** to **Done** on Mar 24

rogerbirkeland mentioned this issue on Mar 24

Remove or change handle_restart_request() for sdr-services #471

Closed

Assignees

No one—assign yourself

Labels

points=3 sdr

Projects

SW kanban board

Done ▾

Milestone

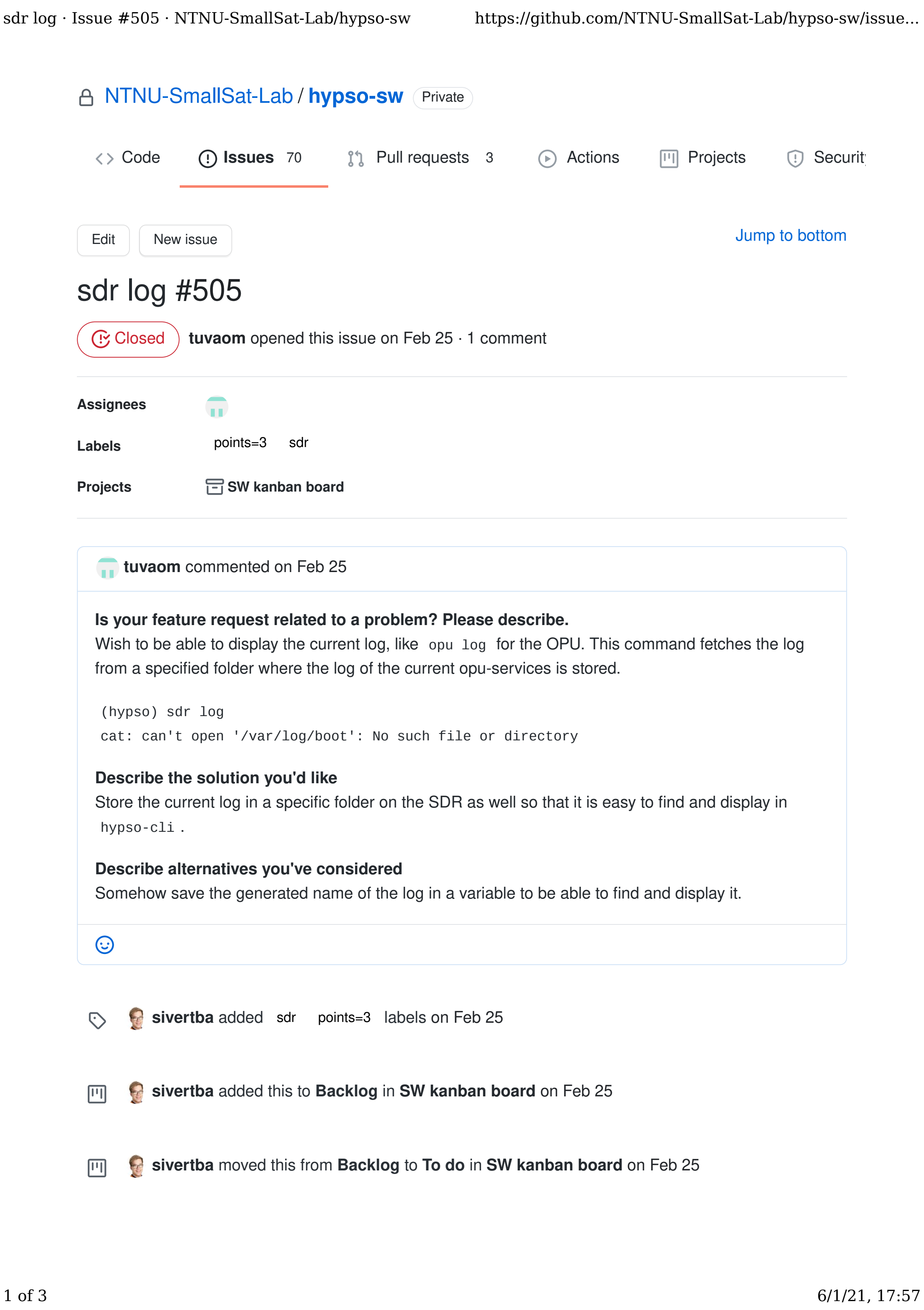
No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

3 participants



sdr log · Issue #505 · NTNU-SmallSat-Lab/hypso-sw https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

sdr log #505

Closed **tuvaom** opened this issue on Feb 25 · 1 comment

Assignees

Labels points=3 sdr

Projects **SW kanban board**

tuvaom commented on Feb 25

Is your feature request related to a problem? Please describe.
Wish to be able to display the current log, like `opu log` for the OPU. This command fetches the log from a specified folder where the log of the current opu-services is stored.

```
(hypso) sdr log
cat: can't open '/var/log/boot': No such file or directory
```

Describe the solution you'd like
Store the current log in a specific folder on the SDR as well so that it is easy to find and display in `hypso-cli`.

Describe alternatives you've considered
Somehow save the generated name of the log in a variable to be able to find and display it.

sivertba added sdr points=3 labels on Feb 25

sivertba added this to **Backlog** in **SW kanban board** on Feb 25

sivertba moved this from **Backlog** to **To do** in **SW kanban board** on Feb 25

1 of 3 6/1/21, 17:57

tuvaom self-assigned this on Feb 25

This was referenced on Feb 25

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged

Decide how to mirror the implementation of opu ... commands for the SDR #476

Closed

sivertba moved this from **To do** to **In progress** in **SW kanban board** on Mar 11

tuvaom commented on Mar 18 Author

closed by #506

☺

tuvaom closed this on Mar 18

SW kanban board automation moved this from **In progress** to **Done** on Mar 18

Assignees

tuvaom

Labels

points=3 sdr

Projects

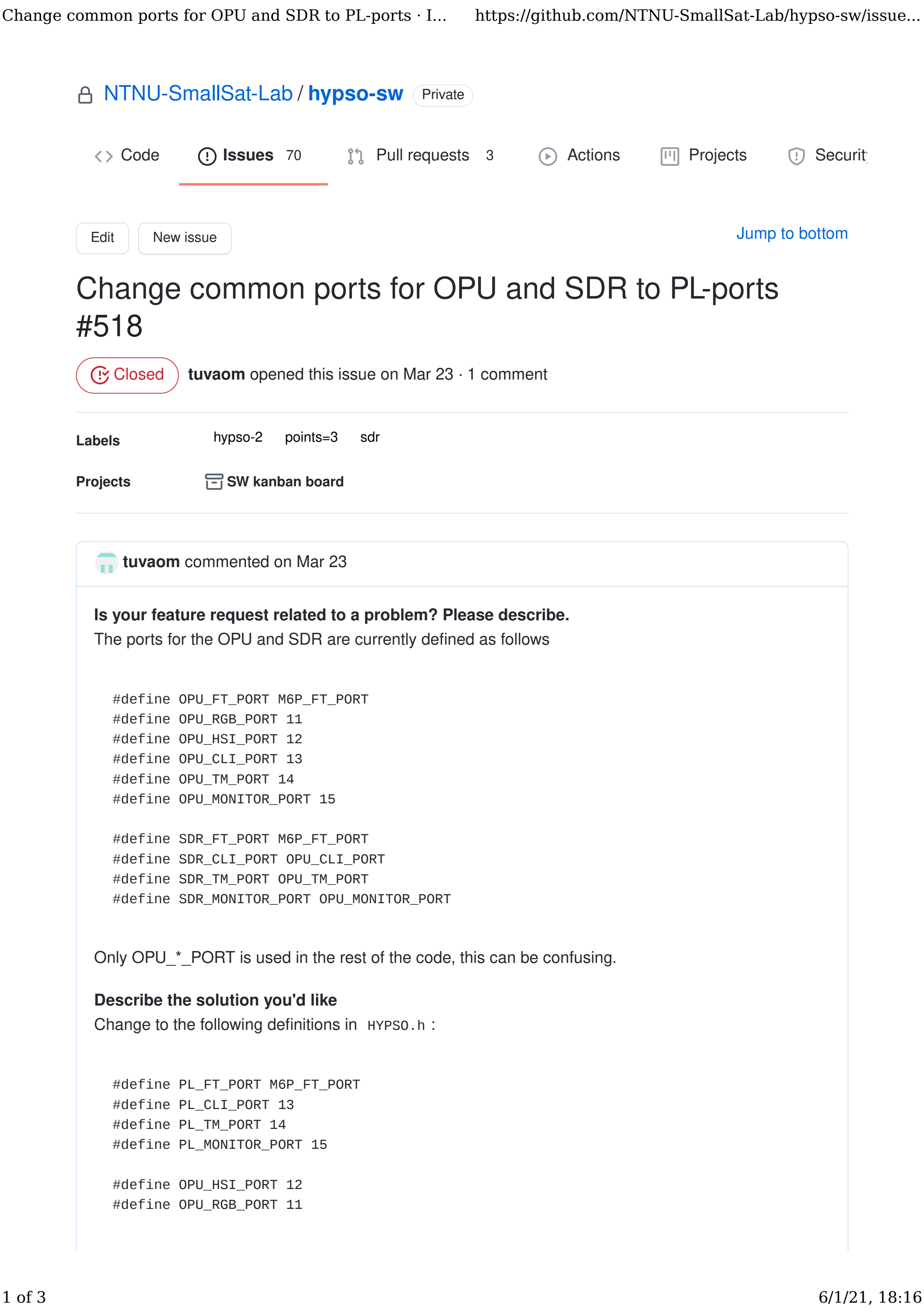
SW kanban board

Done ▾

Milestone

No milestone

N Issue #518 hypso-sw



Change common ports for OPU and SDR to PL-ports · I...   https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...

NTNU-SmallSat-Lab / **hypso-sw** Private

Change common ports for OPU and SDR to PL-ports #518

Closed **tuvaom** opened this issue on Mar 23 · 1 comment

Labels hypso-2 points=3 sdr

Projects **SW kanban board**

tuvaom commented on Mar 23

Is your feature request related to a problem? Please describe.
The ports for the OPU and SDR are currently defined as follows

```
#define OPU_FT_PORT M6P_FT_PORT
#define OPU_RGB_PORT 11
#define OPU_HSI_PORT 12
#define OPU_CLI_PORT 13
#define OPU_TM_PORT 14
#define OPU_MONITOR_PORT 15

#define SDR_FT_PORT M6P_FT_PORT
#define SDR_CLI_PORT OPU_CLI_PORT
#define SDR_TM_PORT OPU_TM_PORT
#define SDR_MONITOR_PORT OPU_MONITOR_PORT
```

Only OPU_*_PORT is used in the rest of the code, this can be confusing.

Describe the solution you'd like
Change to the following definitions in `HYPSO.h` :

```
#define PL_FT_PORT M6P_FT_PORT
#define PL_CLI_PORT 13
#define PL_TM_PORT 14
#define PL_MONITOR_PORT 15

#define OPU_HSI_PORT 12
#define OPU_RGB_PORT 11
```

1 of 3   6/1/21, 18:16

And replace the OPU_*_PORTs in the rest of hypso-sw.

Describe alternatives you've considered

```
#define PL_FT_PORT M6P_FT_PORT
#define PL_CLI_PORT 13
#define PL_TM_PORT 14
#define PL_MONITOR_PORT 15#define OPU_HSI_PORT 12

#define OPU_RGB_PORT 11
#define OPU_FT_PORT M6P_FT_PORT
#define OPU_CLI_PORT PL_CLI_PORT
#define OPU_TM_PORT PL_TM_PORT
#define OPU_MONITOR_PORT PL_MONITOR_PORT

#define SDR_FT_PORT M6P_FT_PORT
#define SDR_CLI_PORT PL_CLI_PORT
#define SDR_TM_PORT PL_TM_PORT
#define SDR_MONITOR_PORT PL_MONITOR_PORT
```

But this would still be more confusing than helpful, and we probably won't change the port numbers of OPU or SDR independently.

tuvaom added hypso-2 sdr labels on Mar 23

rogerbirkeland added the points=3 label on Mar 25

rogerbirkeland added this to **Backlog** in **SW kanban board** on Mar 25

rogerbirkeland moved this from **Backlog** to **To do** in **SW kanban board** on Mar 25

tuvaom mentioned this issue on Apr 2

changing port names to PL_*_PORT #523

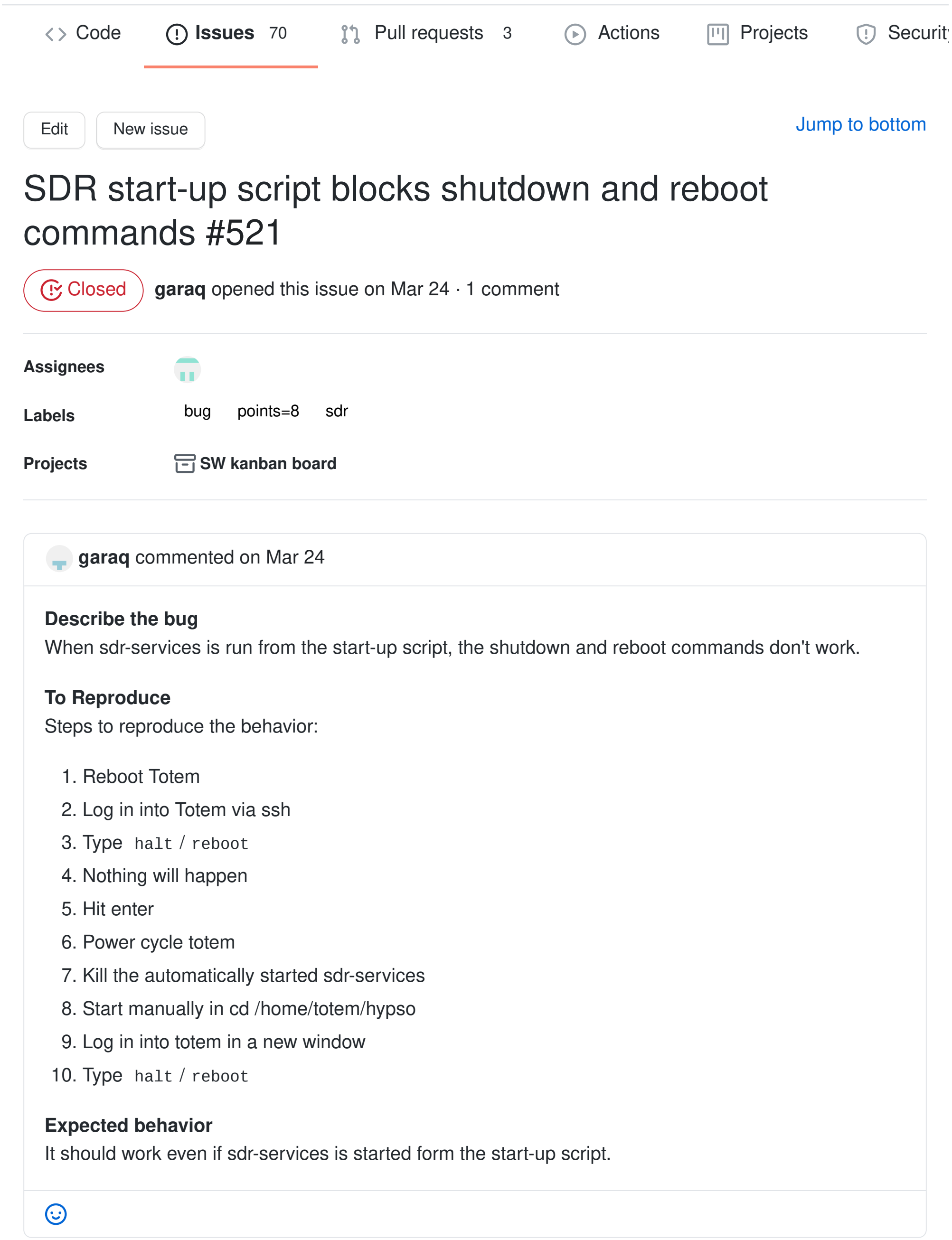
Merged

tuvaom moved this from **To do** to **Review in progress** in **SW kanban board** on Apr 2

tuvaom commented on Apr 13 Author

O Issue #521 hypso-sw

SDR start-up script blocks shutdown and reboot comm... https://github.com/NTNU-SmallSat-Lab/hypso-sw/issue...



SDR start-up script blocks shutdown and reboot commands #521

Closed **garaq** opened this issue on Mar 24 · 1 comment

Assignees

Labels bug points=8 sdr

Projects SW kanban board

garaq commented on Mar 24

Describe the bug
When sdr-services is run from the start-up script, the shutdown and reboot commands don't work.

To Reproduce
Steps to reproduce the behavior:

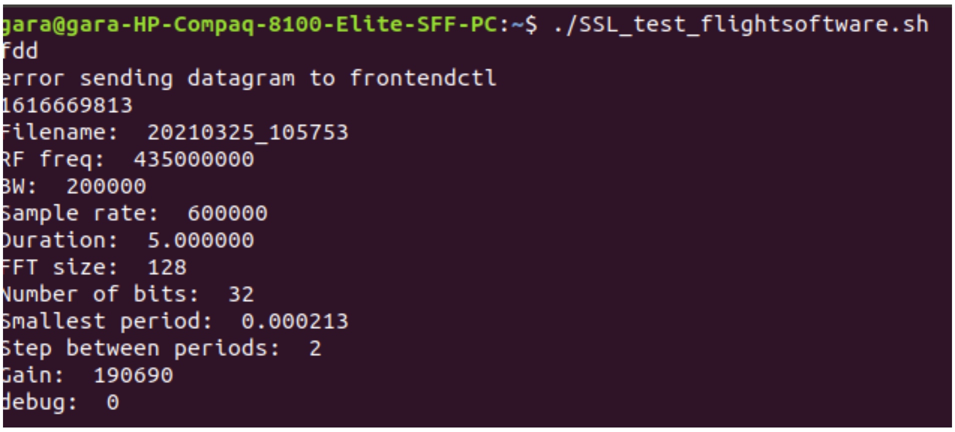
1. Reboot Totem
2. Log in into Totem via ssh
3. Type `halt` / `reboot`
4. Nothing will happen
5. Hit enter
6. Power cycle totem
7. Kill the automatically started sdr-services
8. Start manually in cd /home/totem/hypso
9. Log in into totem in a new window
10. Type `halt` / `reboot`

Expected behavior
It should work even if sdr-services is started form the start-up script.

1 of 3 6/1/21, 18:17

garaq commented on Mar 25 • edited ▾ Author

Also I got a error in a dataframe I sent to Totem while running my tests



```
gara@gara-HP-Compaq-8100-Elite-SFF-PC:~$ ./SSL_test_flightsoftware.sh
fdd
error sending datagram to frontendctl
1616669813
Filename:  20210325_105753
RF freq:  435000000
BW:  200000
Sample rate:  600000
Duration:  5.000000
FFT size:  128
Number of bits:  32
Smallest period:  0.000213
Step between periods:  2
Gain:  190690
debug:  0
```

The receiver does not work either.

rogerbirkeland added the points=8 label on Mar 25

rogerbirkeland added this to **Backlog** in **SW kanban board** on Mar 25

rogerbirkeland moved this from **Backlog** to **To do** in **SW kanban board** on Mar 25

sivertba assigned **tuvaom** on Apr 13

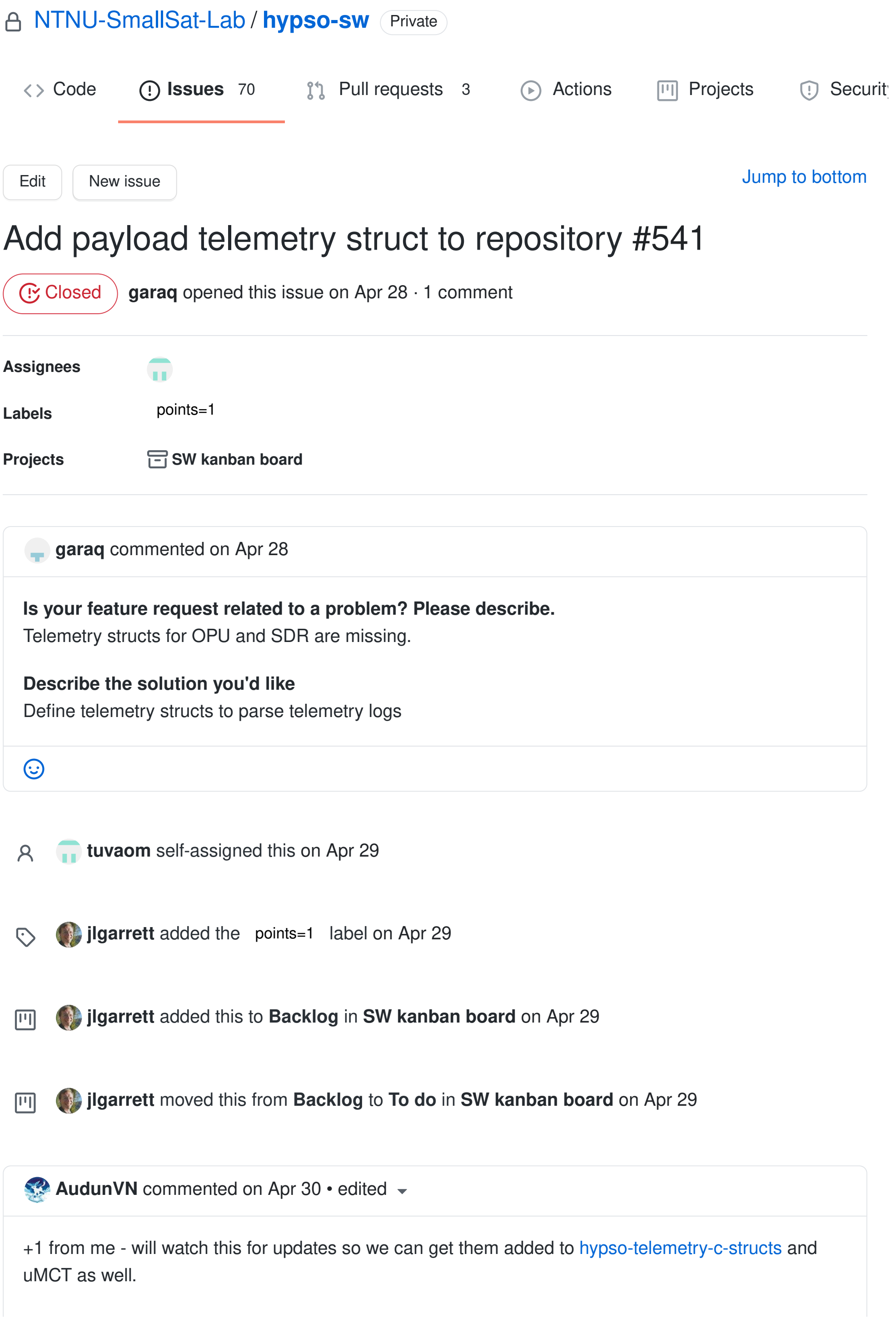
garaq mentioned this issue on Apr 16

Added <&> to start the service in the background NTNU-SmallSat-Lab/sdr-system#9

Merged

garaq closed this in NTNU-SmallSat-Lab/sdr-system#9 on Apr 16

P Issue #541 hypso-sw



NTNU-SmallSat-Lab / **hypso-sw** Private

Add payload telemetry struct to repository #541

Closed **garaq** opened this issue on Apr 28 · 1 comment

Assignees	
Labels	points=1
Projects	**SW kanban board**

garaq commented on Apr 28

Is your feature request related to a problem? Please describe.
Telemetry structs for OPU and SDR are missing.

Describe the solution you'd like
Define telemetry structs to parse telemetry logs

tuvaom self-assigned this on Apr 29

jlgarrett added the points=1 label on Apr 29

jlgarrett added this to **Backlog** in **SW kanban board** on Apr 29

jlgarrett moved this from **Backlog** to **To do** in **SW kanban board** on Apr 29

AudunVN commented on Apr 30 • edited

+1 from me - will watch this for updates so we can get them added to hypso-telemetry-c-structs and uMCT as well.

EDIT: Sorry for accidentally closing the issue for a few seconds, misclicked `Close with comment` instead of `Comment` .

AudunVN closed this on Apr 30

SW kanban board automation moved this from **To do** to **Done** on Apr 30

AudunVN reopened this on Apr 30

AudunVN moved this from **Done** to **To do** in **SW kanban board** on Apr 30

jlgarrett moved this from **To do** to **In progress** in **SW kanban board** on May 4

tuvaom mentioned this issue 25 days ago

Add tmstructs NTNU-SmallSat-Lab/hypso-telemetry-c-structs#2

Merged

rogerbirkeland moved this from **In progress** to **Review in progress** in **SW kanban board** 19 days ago

AudunVN closed this in NTNU-SmallSat-Lab/hypso-telemetry-c-structs#2 19 days ago

SW kanban board automation moved this from **Review in progress** to **Done** 19 days ago

Assignees

tuvaom

Labels

points=1

Projects

SW kanban board

Done ▾

Milestone

No milestone

Linked pull requests

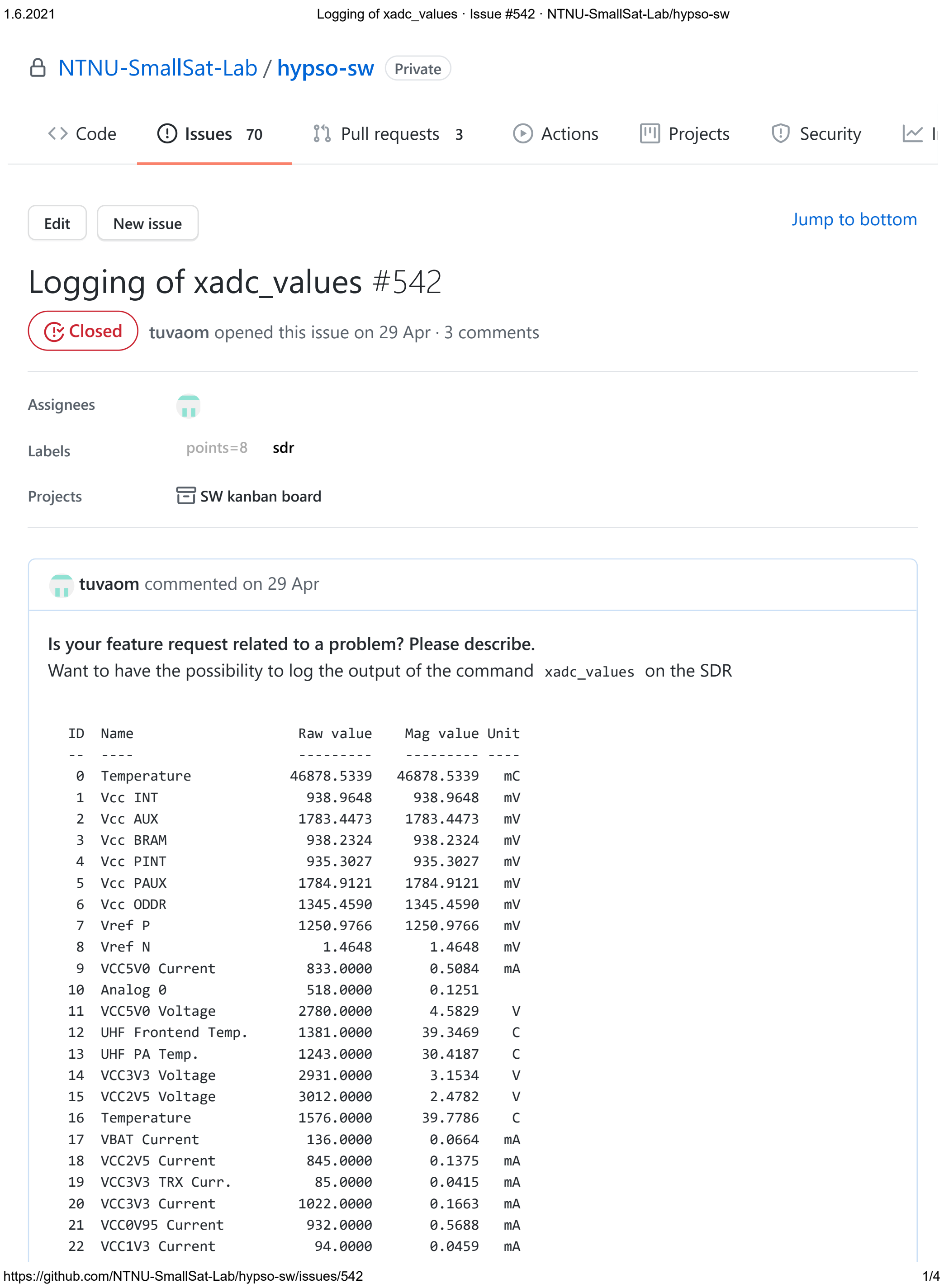
Successfully merging a pull request may close this issue.

Add tmstructs

4 participants

Pin issue ⓘ

Q Issue #542 hypso-sw



1.6.2021 Logging of xadc_values · Issue #542 · NTNU-SmallSat-Lab/hypso-sw

NTNU-SmallSat-Lab / **hypso-sw** Private

Logging of xadc_values #542

Closed **tuvaom** opened this issue on 29 Apr · 3 comments

Assignees

Labels points=8 sdr

Projects SW kanban board

tuvaom commented on 29 Apr

Is your feature request related to a problem? Please describe.
Want to have the possibility to log the output of the command `xadc_values` on the SDR

```
ID  Name                     Raw value   Mag value Unit
--  ----                     ---------   --------- ----
 0  Temperature             46878.5339  46878.5339   mC
 1  Vcc INT                   938.9648    938.9648   mV
 2  Vcc AUX                  1783.4473   1783.4473   mV
 3  Vcc BRAM                  938.2324    938.2324   mV
 4  Vcc PINT                  935.3027    935.3027   mV
 5  Vcc PAUX                 1784.9121   1784.9121   mV
 6  Vcc ODDR                 1345.4590   1345.4590   mV
 7  Vref P                   1250.9766   1250.9766   mV
 8  Vref N                      1.4648      1.4648   mV
 9  VCC5V0 Current            833.0000      0.5084   mA
10  Analog 0                  518.0000      0.1251
11  VCC5V0 Voltage           2780.0000      4.5829    V
12  UHF Frontend Temp.       1381.0000     39.3469    C
13  UHF PA Temp.             1243.0000     30.4187    C
14  VCC3V3 Voltage           2931.0000      3.1534    V
15  VCC2V5 Voltage           3012.0000      2.4782    V
16  Temperature              1576.0000     39.7786    C
17  VBAT Current              136.0000      0.0664   mA
18  VCC2V5 Current            845.0000      0.1375   mA
19  VCC3V3 TRX Curr.           85.0000      0.0415   mA
20  VCC3V3 Current           1022.0000      0.1663   mA
21  VCC0V95 Current           932.0000      0.5688   mA
22  VCC1V3 Current             94.0000      0.0459   mA
```

https://github.com/NTNU-SmallSat-Lab/hypso-sw/issues/542 1/4

```
  23  VCC1V8 Current           304.0000       0.1855    mA
  24  VCC1V35 Current          770.0000       0.3133    mA
```

Describe the solution you'd like
Similar logging as telemetry. As I see it now, there are two alternatives - but other suggestions would be nice.

1. **Save the values in a struct like the telemetry**
   a) Save them in the tm_data_sdr struct, and log in the same file -- think this might result in a larger struct than desired
   b) Make new struct tm_xadc_sdr and log in new file, still in telemetry folder -

Issues with this option: would have to know the content (and its order) of this command already to make the struct, and then to save the values in the struct it would be something like "After the third big gap in this line, this value is written". If the output of the command changes, the code would have to change as well.
Pros: probably easier to log as soon as the values are fetched as they would be saved on the same format as the other tm-values. We also have the parser from NA to read the file after download.

2. **Log the whole output of this command to the file at each logging instance**
   Still save the new file in the telemetry folder. - I'm imagining one line at a time from this command inserted to the file.

Issues with this option: we don't have any tools for reading this file (as far as I know?) - the parser in nanoMCS uses structs. Would also have to log this in a different way than the structs - but I think any input can be logged with the fs_log_write_entry-function.
Pros: don't have to know the content and order of the command.

tuvaom self-assigned this on 29 Apr

tuvaom added the sdr label on 29 Apr

jlgarrett added the points=8 label on 29 Apr

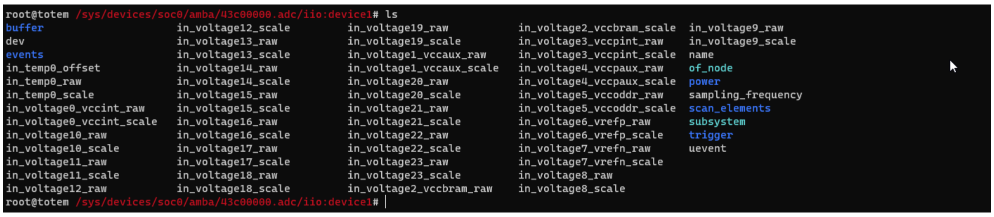
jlgarrett added this to **Backlog** in **SW kanban board** on 29 Apr

jlgarrett moved this from **Backlog** to **To do** in **SW kanban board** on 29 Apr

rogerbirkeland commented 20 days ago • edited

The correct(?) solution would be to read each value directly and build a struct for it. I suggest logging only the converted values in the struct.

All values can be read from the device directly: `/sys/devices/soc0/amba/43c00000.adc/iio:device1` , and all values are exposed as files. There is a script (`/usr/bin/xadc_values`) that also has all scaling factors so it can be used as inspiration.



```
root@totem /sys/devices/soc0/amba/43c00000.adc/iio:device1# ls
buffer                     in_voltage12_scale         in_voltage19_raw           in_voltage2_vccbram_scale  in_voltage9_raw
dev                        in_voltage13_raw           in_voltage19_scale         in_voltage3_vccpint_raw    in_voltage9_scale
events                     in_voltage13_scale         in_voltage1_vccaux_raw     in_voltage3_vccpint_scale  name
in_temp0_offset            in_voltage14_raw           in_voltage1_vccaux_scale   in_voltage4_vccpaux_raw    of_node
in_temp0_raw               in_voltage14_scale         in_voltage20_raw           in_voltage4_vccpaux_scale  power
in_temp0_scale             in_voltage15_raw           in_voltage20_scale         in_voltage5_vccoddr_raw    sampling_frequency
in_voltage0_vccint_raw     in_voltage15_scale         in_voltage21_raw           in_voltage5_vccoddr_scale  scan_elements
in_voltage0_vccint_scale   in_voltage16_raw           in_voltage21_scale         in_voltage6_vrefp_raw      subsystem
in_voltage10_raw           in_voltage16_scale         in_voltage22_raw           in_voltage6_vrefp_scale    trigger
in_voltage10_scale         in_voltage17_raw           in_voltage22_scale         in_voltage7_vrefn_raw      uevent
in_voltage11_raw           in_voltage17_scale         in_voltage23_raw           in_voltage7_vrefn_scale
in_voltage11_scale         in_voltage18_raw           in_voltage23_scale         in_voltage8_raw
in_voltage12_raw           in_voltage18_scale         in_voltage2_vccbram_raw    in_voltage8_scale
root@totem /sys/devices/soc0/amba/43c00000.adc/iio:device1# |
```

Alternatively: Make a copy of the said script, outputing only an index and the converted value, and then read that script from C?

☺

tuvaom mentioned this issue 19 days ago

Sdr xadc #546

Merged

rogerbirkeland moved this from **To do** to **Review in progress** in **SW kanban board** 13 days ago

rogerbirkeland commented 13 days ago

Probably resolved by #546

☺

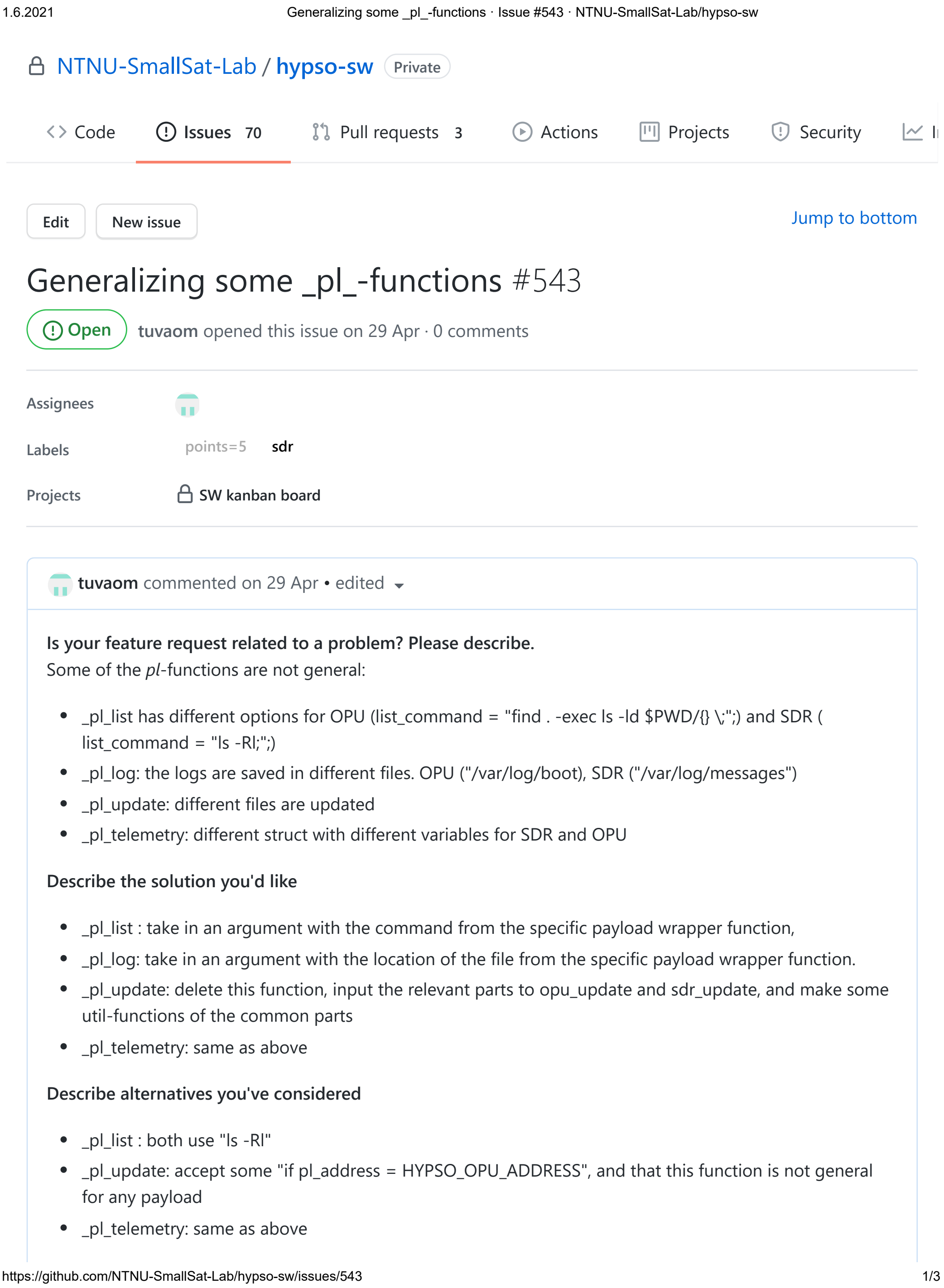
rogerbirkeland linked a pull request that will close this issue 13 days ago

Sdr xadc #546 Merged

tuvaom commented 12 days ago Author

closed by #546

R Issue #543 hypso-sw



1.6.2021 Generalizing some _pl_-functions · Issue #543 · NTNU-SmallSat-Lab/hypso-sw

NTNU-SmallSat-Lab / **hypso-sw** Private

Generalizing some _pl_-functions #543

Open **tuvaom** opened this issue on 29 Apr · 0 comments

Assignees	
Labels	points=5 sdr
Projects	SW kanban board

tuvaom commented on 29 Apr • edited

Is your feature request related to a problem? Please describe.
Some of the *pl*-functions are not general:

- _pl_list has different options for OPU (list_command = "find . -exec ls -ld $PWD/{} \;";) and SDR (list_command = "ls -Rl;";)
- _pl_log: the logs are saved in different files. OPU ("/var/log/boot), SDR ("/var/log/messages")
- _pl_update: different files are updated
- _pl_telemetry: different struct with different variables for SDR and OPU

Describe the solution you'd like

- _pl_list : take in an argument with the command from the specific payload wrapper function,
- _pl_log: take in an argument with the location of the file from the specific payload wrapper function.
- _pl_update: delete this function, input the relevant parts to opu_update and sdr_update, and make some util-functions of the common parts
- _pl_telemetry: same as above

Describe alternatives you've considered

- _pl_list : both use "ls -Rl"
- _pl_update: accept some "if pl_address = HYPSO_OPU_ADDRESS", and that this function is not general for any payload
- _pl_telemetry: same as above

https://github.com/NTNU-SmallSat-Lab/hypso-sw/issues/543 1/3

Would be nice with some comments on preferred options - also if there are any other suggestions.

tuvaom self-assigned this on 29 Apr

tuvaom added the **sdr** label on 29 Apr

jlgarrett added the points=5 label on 29 Apr

jlgarrett added this to **Backlog** in **SW kanban board** on 29 Apr

jlgarrett moved this from **Backlog** to **To do** in **SW kanban board** on 29 Apr

jlgarrett moved this from **To do** to **In progress** in **SW kanban board** 28 days ago

tuvaom mentioned this issue 12 days ago

Sdr generalize #547

Open

jlgarrett moved this from **In progress** to **Review in progress** in **SW kanban board** 7 days ago

Assignees

tuvaom

Labels

points=5 sdr

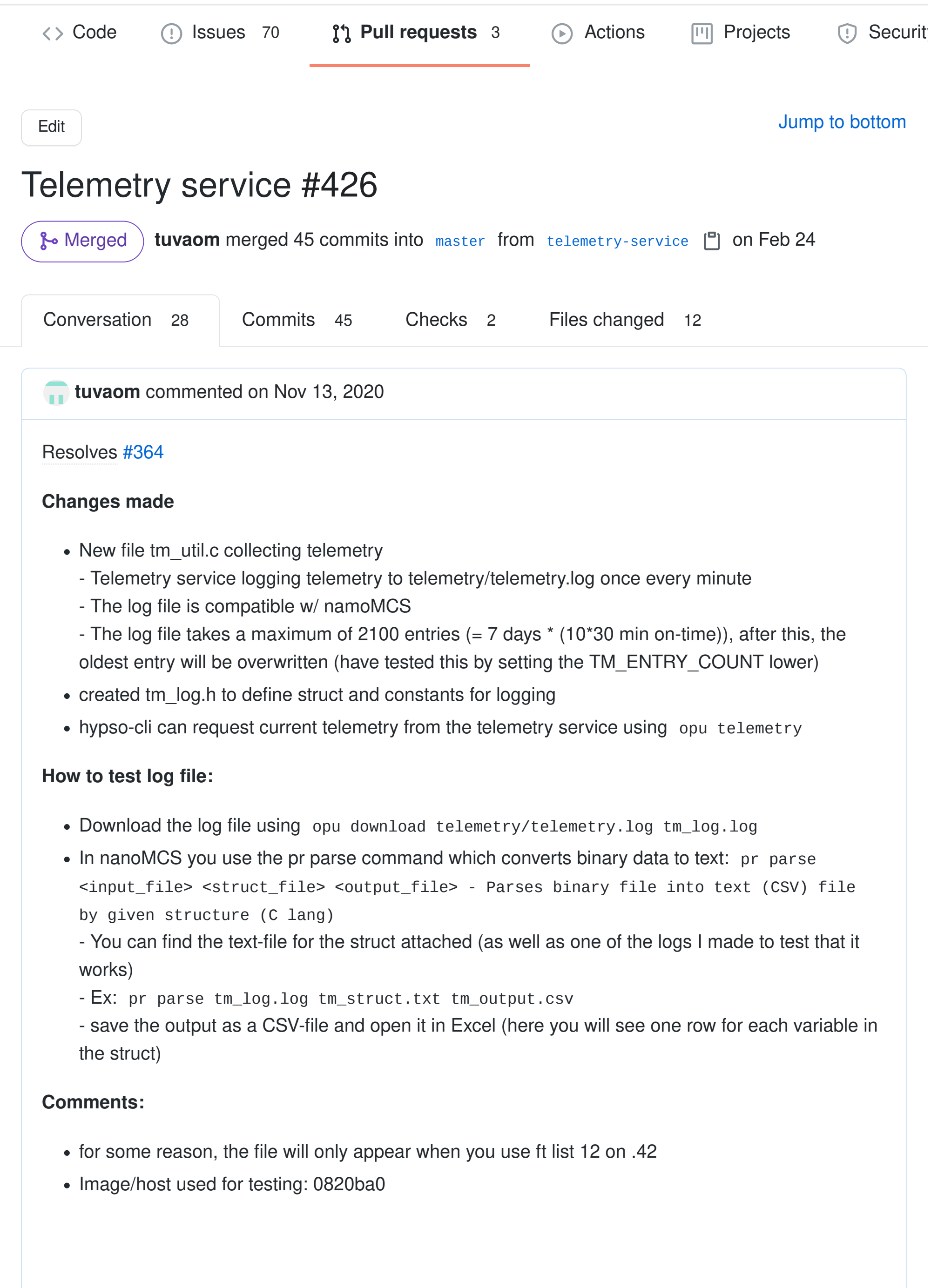
Projects

SW kanban board

Review in progress ▾

S PR #426 hypso-sw

Telemetry service by tuvaom · Pull Request #426 · NT... https://github.com/NTNU-SmallSat-Lab/hypso-sw/pull/426



Telemetry service #426

Merged **tuvaom** merged 45 commits into `master` from `telemetry-service` on Feb 24

tuvaom commented on Nov 13, 2020

Resolves #364

Changes made

- New file tm_util.c collecting telemetry
  - Telemetry service logging telemetry to telemetry/telemetry.log once every minute
  - The log file is compatible w/ namoMCS
  - The log file takes a maximum of 2100 entries (= 7 days * (10*30 min on-time)), after this, the oldest entry will be overwritten (have tested this by setting the TM_ENTRY_COUNT lower)
- created tm_log.h to define struct and constants for logging
- hypso-cli can request current telemetry from the telemetry service using `opu telemetry`

How to test log file:

- Download the log file using `opu download telemetry/telemetry.log tm_log.log`
- In nanoMCS you use the pr parse command which converts binary data to text: `pr parse <input_file> <struct_file> <output_file> - Parses binary file into text (CSV) file by given structure (C lang)`
  - You can find the text-file for the struct attached (as well as one of the logs I made to test that it works)
  - Ex: `pr parse tm_log.log tm_struct.txt tm_output.csv`
  - save the output as a CSV-file and open it in Excel (here you will see one row for each variable in the struct)

Comments:

- for some reason, the file will only appear when you use ft list 12 on .42
- Image/host used for testing: 0820ba0

1 of 11 6/1/21, 17:42

tuvaom added 29 commits on Oct 5, 2020

make opu telemetry command	cca7b3e
basic tm_util w/functions to get telemetry	88f0f29
add get_image	1d434c3
get disk statistics	0b4bcf6
define telemetry struct	b3facc0
create directory to save tm logs	5254ba6
init logging, start log func	7930563
log correct memory, error handling for util funcs	76e8dc7
only create folder + init logfile if nonexixting when thread starts	df339b1
timestamp human readable + not, log appending until maxsize, log inte… …	cbb480a
Merge branch 'master' into telemetry-service	1a80b08
start fix get current telemetry from hypso-cli	3f47519
change sizes in struct	af5b19c
fix struct sizes and logging loop	3df9df8
opu telemetry returns stuct to hypso-cli, logging included in CSP loop	43f27fa
add constants for circular logging	257686e
Fix opu Get telemetry	fb243b3
circular log w/ FS_TYPE_LOG	160e245
uncomment csp_buffer_free	8f81b07
opu telemetry from hypso-cli now returns correct struct	121ee72
cleanup of telemetry structs	569068a
merge conflict fix	3a01cf4
TM struct compatible w/ nanoMCS	47e15d7
Remove unnecessary functions from tm_util	50733ad

`logging works w/ nanoMCS` 8835f1c

`Removing commments and functions not in use` a40263c

`get all of img` 617e8ac

tuvaom requested a review from **DennisNTNU** on Nov 13, 2020

14 hidden items
Load more...

magne-hov approved these changes on Nov 17, 2020

View changes

magne-hov left a comment

Thanks for making the changes!

☺

magne-hov commented on Nov 18, 2020 • edited ▾

I just tried running `opu-services` and I get a crash:

```
[Started] CLAW-1 Service            Thread ID: 134974

Panic - verify failed:
Expression:  file
Location:    src/fs/fs.c:338:fs_format_file
Backtrace:
[Started] Telemetry Service         Thread ID: 134975
  /home/magne/repos/hypso-sw/build/x86/opu-services(print_backtrace+0x35)
[0x55f180eb12b5]
  /home/magne/repos/hypso-sw/build/x86/opu-services(panic+0x1b0) [0x55f180eb14f0]
  /home/magne/repos/hypso-sw/build/x86/opu-services(fs_format_file+0x1c5)
[0x55f180eb2aa5]
  /home/magne/repos/hypso-sw/build/x86/opu-services(tm_util_init_log+0x2c)
[0x55f180ec550c]
  /home/magne/repos/hypso-sw/build/x86/opu-services(tm_service_task+0x2c4)
[0x55f180ec4b34]
  /usr/lib/libpthread.so.0(+0x93e9) [0x7fd74a59a3e9]
```

```
aborted (core dumped)
~/repos/hypso-sw/build/x86 $
```

Opening the core dump in GDB (can be done with `coredumpctl debug` if your system runs systemd) shows that the crash happened here:

hypso-sw/src/tm/tm_util.c
Lines 372 to 375 in `d83e938`

```
372     tm_log_file->p_telemetry_log_file = fs_open_rw(tm_log_file->tm_filepath);
373     fs_format_file(FS_TYPE_LOG, TM_FILE_ID, TM_ENTRY_SZ, TM_ENTRY_COUNT,
374                    tm_log_file->p_telemetry_log_file);
375
```

I assume that this happens becaues I hadn't made the `telemetry` directory yet, so it fails to open the file and we try to format `NULL` .

```
(gdb) p tm_log_file->tm_filepath
$1 = "telemetry/telemetry.log", '\000' <repeats 76 times>
(gdb)
```

Does this happen for you as well?

rogerbirkeland commented on Nov 18, 2020

> I assume that this happens becaues I hadn't made the `telemetry` directory yet, so it fails to open the file and we try to format `NULL` .
>
> ```
> (gdb) p tm_log_file->tm_filepath
> $1 = "telemetry/telemetry.log", '\000' <repeats 76 times>
> (gdb)
> ```
>
> Does this happen for you as well?

I couldn't reproduce. Deleted the telemetry-folder from another opu-services version, then started the tm-version and it worked fine.

magne-hov commented on Nov 18, 2020

> I couldn't reproduce.

`mkdir returns 0 on success, not fail` `507038f`

tuvaom commented on Nov 19, 2020 Author

> > I couldn't reproduce.
>
> As long as it's not happening for you then it's not a problem.

I think the problem might have been that when I was checking if mkdir failed, I was really checking if it didn't fail. Therefore the init log-file was made in the current directory (even though the telemetry directory was created), while at the first logging, when checking if the directory exists - it does, and a new log-file is then created in the telemetry directory.

(see changes in my latest commit)

sivertba requested changes on Nov 20, 2020

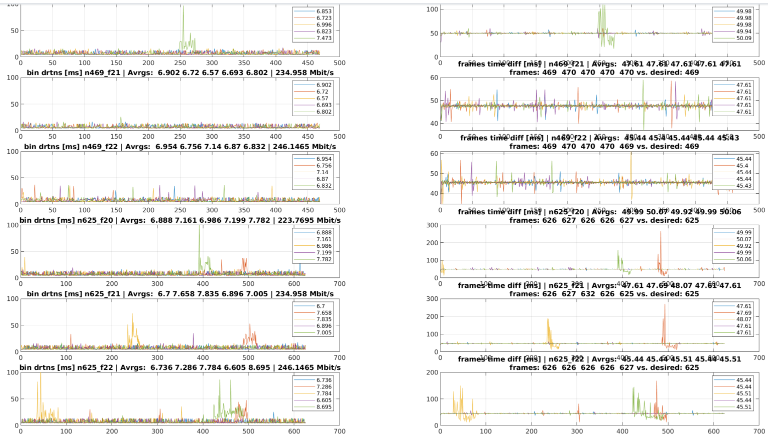
View changes

sivertba left a comment

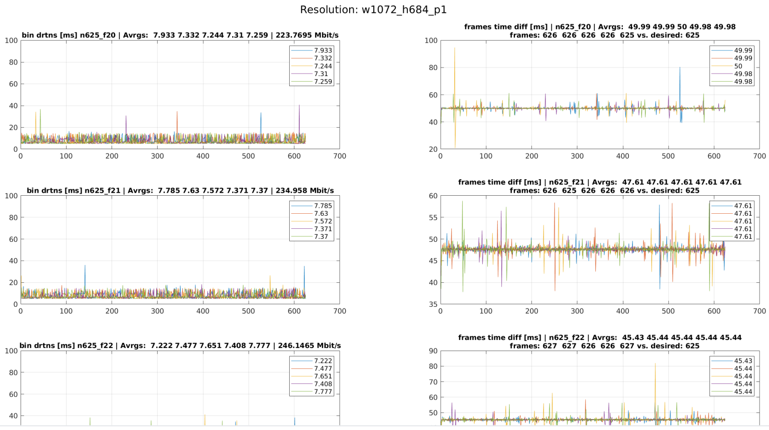
Wait till patch release is merged ...

sivertba commented on Nov 25, 2020 • edited by DennisNTNU ▾

From performance testing on target hardware it seems like hsi capture performance is degraded, see figure. Needs to be resolved, or some other mitigation actions needs to be taken before merging. **@DennisNTNU** will do some further testing.



Example from before:



rogerbirkeland commented on Nov 25, 2020

Interesting. Think I need an explanation to go with these graphs. But anyway, easy(?) workaround might be to disable tm during capture? I dont see a problem with the potential lack of data in that short period.

👍 1

sivertba commented on Nov 26, 2020

tuvaom added 3 commits on Jan 20

`hypso-cli can stop telemetry logging, hsi capture will stop telemetry…` ··· e7c5631

`Merge branch 'master' into telemetry-service` ✓ 2117015

`Update cli_opu.c` ··· ✓ d35982f

tuvaom commented on Jan 21 Author

> Should consider having the hsi capture say start and stop to the telemetry service via CSP packets.
> Should also be possible to start and stop telemetry service via hypso cli

The telemetry logging will now be turned off before capture, and started again when capture is finished. The logging can also be turned on/off in hypso-cli with the command `opu tmlog [on | off]` . The current logging status will be displayed in hypso-cli when de command is run without an option (`opu tmlog`).

`hsi capture turning off/on tm logging from the OPU, not from hypso-cli` ✓ e8da2cb

rogerbirkeland commented on Feb 2

Compile error:

```
[ 26%] Building C object CMakeFiles/opu-services.dir/src/tm/tm_service.c.o
/home/hypso/src/tm/tm_service.c: In function 'tm_service_task':
/home/hypso/src/tm/tm_service.c:49:16: error: 'struct telemetry_log_file' has no
member named 'tm_log_status'
     tm_log_file.tm_log_status = true;
                ^
CMakeFiles/opu-services.dir/build.make:998: recipe for target 'CMakeFiles/opu-
services.dir/src/tm/tm_service.c.o' failed
make[3]: *** [CMakeFiles/opu-services.dir/src/tm/tm_service.c.o] Error 1
make[3]: Leaving directory '/home/hypso/build/x86'
CMakeFiles/Makefile2:105: recipe for target 'CMakeFiles/opu-services.dir/all' failed
make[2]: *** [CMakeFiles/opu-services.dir/all] Error 2
make[2]: Leaving directory '/home/hypso/build/x86'
Makefile:94: recipe for target 'all' failed
make[1]: *** [all] Error 2
make[1]: Leaving directory '/home/hypso/build/x86'
Makefile:8: recipe for target 'all' failed
```

☺

`remove old logging variable` ✓ 828f786

rogerbirkeland commented on Feb 2

Did a quick test and the module works. Would like to see this change:

- Be able to set a config value for the telemetry logging period. Perhaps between 5 s to 5 min.
- Config value should survive reboot....

☺

tuvaom added 3 commits on Feb 3

`Logic added to specify tm log period in hypso-cli` ✓ df6676c

`Saved to file when log interval is changed, tm service reads from fil…` ✓ 15c20cc

…

`Log interval works, addd clang` ✓ f048787

rogerbirkeland commented on Feb 17

Almost there; but something is weird with the uptime-column:

Short cut-out from the parsed TM-file:

```
opuTime          opuUptime  diff_oU   diff_oT
1613546726      165
1613546746      203     38      20
1613546751      213     10      5
1613546756      223     10      5
1613546761      233     10      5
1613546766      243     10      5
1613546771      252     9       5
1613546776      262     10      5
1613546791      291     29      15
1613546806      320     29      15
1613546821      350     30      15
1613546828      365     15      7
1613546833      374     9       5
1613546861      430     56      28
```

```
1613546914      512     9       5
1613546919      522     10      5
1613546924      532     10      5
1613546929      542     10      5
1613546934      552     10      5
1613546939      560     8       5
1613546944      570     10      5
1613546949      579     9       5
1613546954      589     10      5
1613546959      599     10      5
1613546964      609     10      5
1613546969      619     10      5
1613546974      627     8       5
1613546979      637     10      5
1613546984      647     10      5
1613546989      657     10      5
1613546994      667     10      5
1613546999      677     10      5
1613547004      685     8       5
1613547009      695     10      5
1613547014      705     10      5
1613547019      715     10      5
1613547024      725     10      5
1613547029      735     10      5
1613547034      745     10      5
1613547094      861     116     60
1613547154      979     118     60
1613547214      1097    118     60
```

☺

uptime value corrected ✓ fd58576

rogerbirkeland approved these changes on Feb 17

View changes

Tested, seems to do the trick now.

Could change log interval
Could turn of logging
Logging is off during capture
Uptime seem now to be correct.

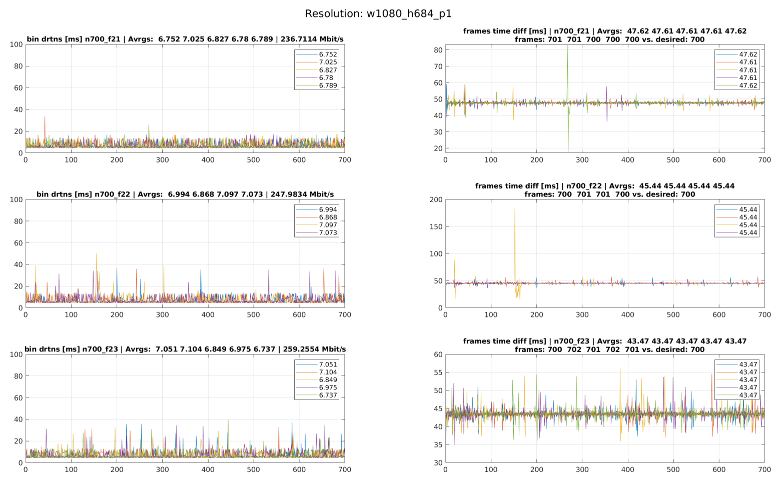
sivertba approved these changes on Feb 17

View changes

DennisNTNU approved these changes on Feb 24

View changes

DennisNTNU left a comment



Plots look good, lets get this merged.

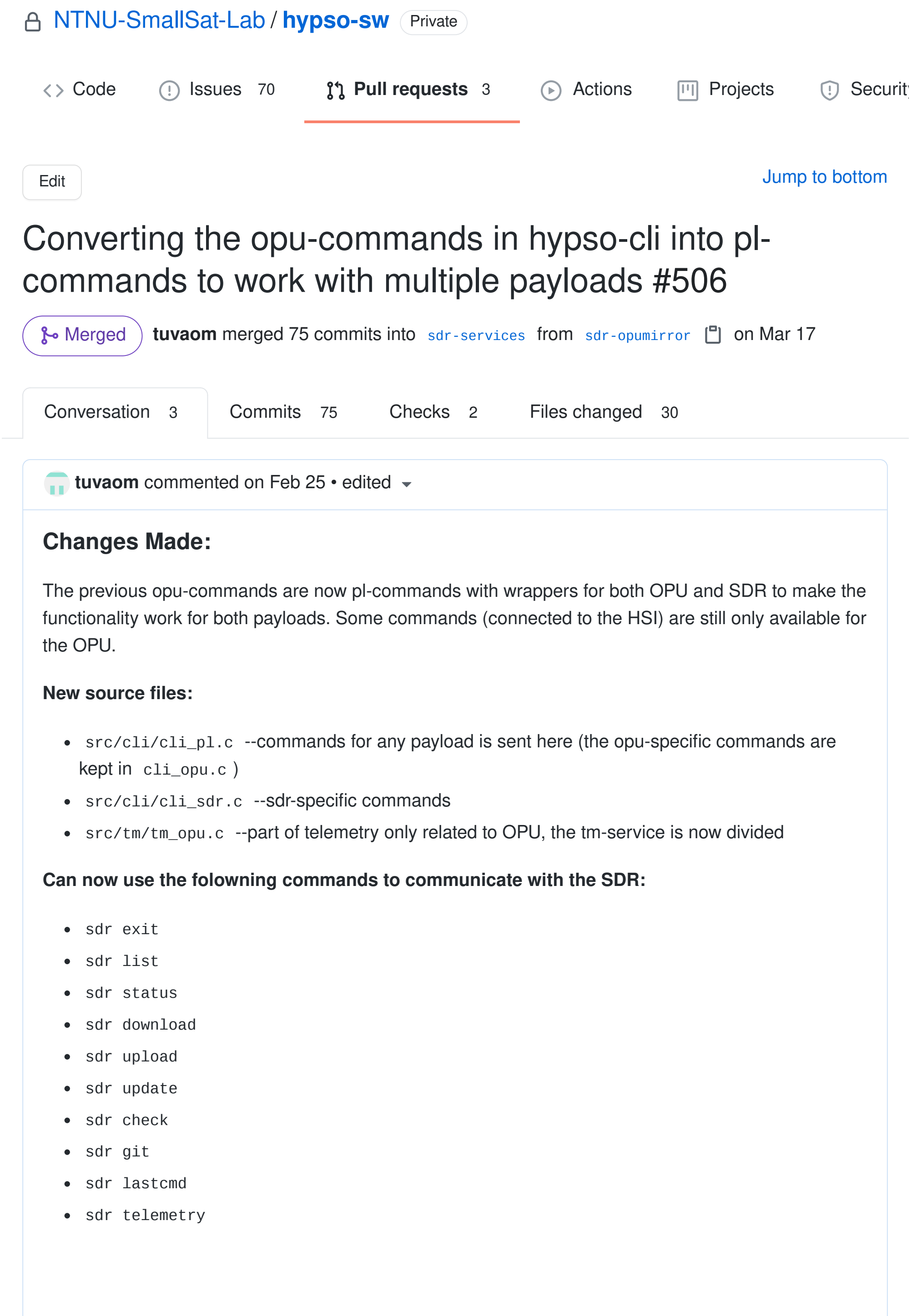
👍 1

Reviewers

Merge branch 'master' into telemetry-service ✓ 82387fe ✓

rogerbirkeland ✓

T PR #506 hypso-sw



NTNU-SmallSat-Lab / **hypso-sw** Private

Converting the opu-commands in hypso-cli into pl-commands to work with multiple payloads #506

Merged **tuvaom** merged 75 commits into `sdr-services` from `sdr-opumirror` on Mar 17

tuvaom commented on Feb 25 • edited

Changes Made:

The previous opu-commands are now pl-commands with wrappers for both OPU and SDR to make the functionality work for both payloads. Some commands (connected to the HSI) are still only available for the OPU.

New source files:

- `src/cli/cli_pl.c` --commands for any payload is sent here (the opu-specific commands are kept in `cli_opu.c`)
- `src/cli/cli_sdr.c` --sdr-specific commands
- `src/tm/tm_opu.c` --part of telemetry only related to OPU, the tm-service is now divided

Can now use the folowning commands to communicate with the SDR:

- `sdr exit`
- `sdr list`
- `sdr status`
- `sdr download`
- `sdr upload`
- `sdr update`
- `sdr check`
- `sdr git`
- `sdr lastcmd`
- `sdr telemetry`

The general `pl <number>` can also be used:

```
sdr exit  = pl exit 13
opu update opu-services sd = pl update 12 opu-services sd
```

Still need:

- `sdr log` (#505)
- `sdr settime` (#502)
- `sdr shutdown` (#503)
- `sdr restart` (#504)
- choose specific telemetry for logging and `sdr telemetry` (#501)
- a file `src/tm/tm_sdr.c` if any sdr-specific telemetry functions arise

How to test

- start the (new) sdr-services from totem@129.241.2.61 with command `./sdr-services 13 can0`
- use the commands from hypso-cli, works just like the OPU
- (should probably also make sure that the commands still work for the OPU)

Related issues

- Resolves #474
- Resolves #475
- Resolves #493
- Resolves #494
- Resolves #495
- Resolves #496

tuvaom added 30 commits on Oct 5, 2020

make opu telemetry command	cca7b3e
basic tm_util w/functions to get telemetry	88f0f29
add get_image	1d434c3
get disk statistics	0b4bcf6
define telemetry struct	b3facc0
create directory to save tm logs	5254ba6
init logging, start log func	7930563
log correct memory, error handling for util funcs	76e8dc7
only create folder + init logfile if nonexixting when thread starts	df339b1
timestamp human readable + not, log appending until maxsize, log inte…	cbb480a

···

Merge branch 'master' into telemetry-service		1a80b08
start fix get current telemetry from hypso-cli		3f47519
change sizes in struct		af5b19c
fix struct sizes and logging loop		3df9df8
opu telemetry returns stuct to hypso-cli, logging included in CSP loop		43f27fa
add constants for circular logging		257686e
Fix opu Get telemetry		fb243b3
circular log w/ FS_TYPE_LOG		160e245
uncomment csp_buffer_free		8f81b07
opu telemetry from hypso-cli now returns correct struct		121ee72
cleanup of telemetry structs		569068a
merge conflict fix		3a01cf4
TM struct compatible w/ nanoMCS		47e15d7
Remove unnecessary functions from tm_util		50733ad
struct directly to fs_log, worksgit add tm_util.c		12091d8
rolling logging		0a223f1
logging works w/ nanoMCS		8835f1c
Removing commments and functions not in use		a40263c
get all of img		617e8ac
applying clang format		2654828

29 hidden items
Load more...

tuvaom and others added 13 commits on Feb 8

Saved to file when log interval is changed, tm service reads from fil… ···	✓	15c20cc
Log interval works, addd clang	✓	f048787
Creating cli_pl for common payload functions from cli_opu		ad4fe49

`opu and pl commands work for opu` 87de6dd

`uptime value corrected` ✓ fd58576

`can send commands to sdr from hypso-cli` 7490611

`Added hostname to opu status and opu git commands` … b251c9e

`dividing telemetry service for sdr` 9db8bcc

`added sdr-services to pl_update` d501985

`Merge branch 'master' into telemetry-service` ✓ 82387fe

`Merge pull request #426 from NTNU-SmallSat-Lab/telemetry-service` … 044903a

`Merge with tm-changes in master. More checks on user input. Sdr update.` 3c74781

`Address fix in pl commands, now pl ..., opu ... and sdr ... should al…` … ✓ 4e04d50

garaq self-requested a review on Mar 5

garaq requested changes on Mar 5

View changes

garaq left a comment

Correct:

1. The output of SDR git should say "Payload system" or "SDR system" when the info is from the SDR.

```
(hypso) sdr git
<--
Git commit: 3c74781-dirty
Git branch: sdr-opumirror
./sdr-services
Opu-system: totem
```

2. SDR telemetry should display the xadc_values in addition, for now.

☺

garaq commented on Mar 5

csp shutdown and csp reboot works when sdr-services is run as root.
sdr settime works with root and couldn't test without root

☺

tuvaom added 3 commits on Mar 5

`fix sdr log`	✓ 55f3097
`pl/opu/sdr git says payload-system instead of opu-system`	0349427
`Display xadc_values for 'sdr telemetry'`	✓ 5677c4f

tuvaom commented on Mar 17 Author

With commit `55f3097` , now also resolves #505

☺

garaq approved these changes on Mar 17

View changes

tuvaom merged commit **`762adce`** into `sdr-services` on Mar 17 View details Revert
2 checks passed

This was referenced on Mar 17

make opu upload support the SDR #474

Closed

opu check does not work on SDR #475

Closed

cli_opu -> cli_opu + cli_sdr + cli_pl #493

Closed

tm_service.c -> tm_service.c + tm_opu.c + tm_sdr.c #494

Closed

Find a way to list all files on the sdr #495

Closed

Make opu update include sdr-services #496

Closed

sdr log #505

Closed

Reviewers

garaq ✓

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

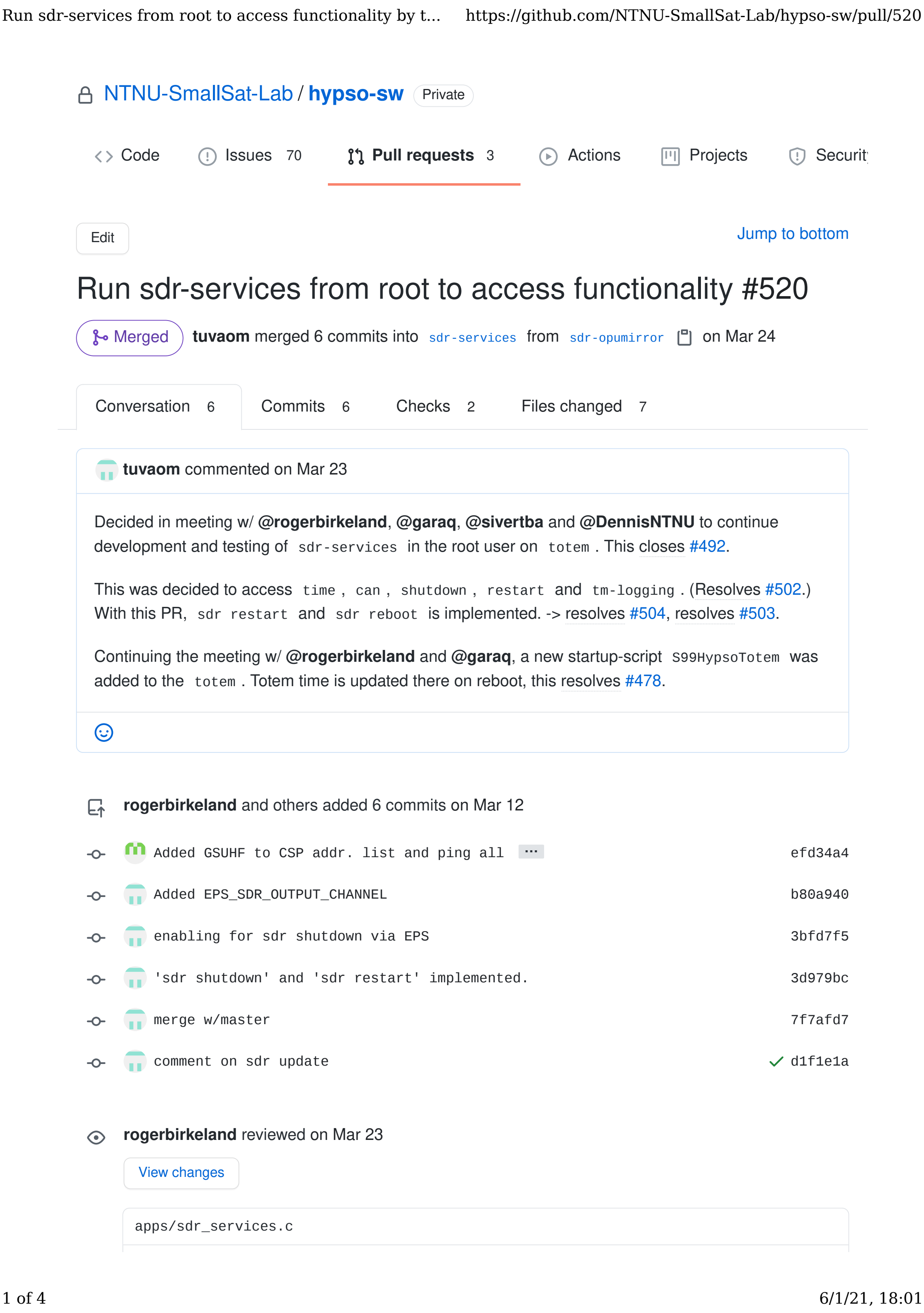
Linked issues

Successfully merging this pull request may close these issues.

None yet

5 participants

U PR #520 hypso-sw



Run sdr-services from root to access functionality by t... https://github.com/NTNU-SmallSat-Lab/hypso-sw/pull/520

Run sdr-services from root to access functionality #520

Merged **tuvaom** merged 6 commits into `sdr-services` from `sdr-opumirror` on Mar 24

tuvaom commented on Mar 23

Decided in meeting w/ **@rogerbirkeland**, **@garaq**, **@sivertba** and **@DennisNTNU** to continue development and testing of `sdr-services` in the root user on `totem` . This closes #492.

This was decided to access `time` , `can` , `shutdown` , `restart` and `tm-logging` . (Resolves #502.) With this PR, `sdr restart` and `sdr reboot` is implemented. -> resolves #504, resolves #503.

Continuing the meeting w/ **@rogerbirkeland** and **@garaq**, a new startup-script `S99HypsoTotem` was added to the `totem` . Totem time is updated there on reboot, this resolves #478.

rogerbirkeland and others added 6 commits on Mar 12

```
Added GSUHF to CSP addr. list and ping all  ...      efd34a4
Added EPS_SDR_OUTPUT_CHANNEL                          b80a940
enabling for sdr shutdown via EPS                     3bfd7f5
'sdr shutdown' and 'sdr restart' implemented.         3d979bc
merge w/master                                        7f7afd7
comment on sdr update                               ✓ d1f1e1a
```

rogerbirkeland reviewed on Mar 23

```
apps/sdr_services.c
```

```
                            if (EPS_Single_Output_Channel_Control(
-                                   EPS_OPU_OUTPUT_CHANNEL, 0, 30) != 0)
+                                   EPS_SDR_OUTPUT_CHANNEL, 0, 30) != 0)
```

rogerbirkeland on Mar 23

Not sure if we want to have this active now. Will be a bit of an annoyance for Gara and Stian, I think. Can leave the infrastructure for it, but disable it?

tuvaom on Mar 23 Author

This is for the `sdr shutdown` command, not the automatic reboot.

Reply…

Resolve conversation

rogerbirkeland reviewed on Mar 23

View changes

src/cli/cli_pl.c

```
-               args);
+       printf("%sWarning!%s %s%s%s\n", ANSI_COLOUR_YELLOW, ANSI_COLOU
+               "You are about to request a restart from the file '",
+               args, "' in your PL working directory.");
        }
```

rogerbirkeland on Mar 23

Will this work for the totem? I guess it will just restart itself?

tuvaom on Mar 23 Author

It works, tried it with two different versions. One including the tm-logging.

tuvaom on Mar 23 Author

uses the handle_restart_request function in sdr_services.c, which executes:

```
int ret_local =
    execl(exec_file_path, exec_file_path, "13", "can0", (char*)NULL);
```

Reply…

Resolve conversation

garaq self-requested a review on Mar 24

garaq approved these changes on Mar 24

View changes

garaq left a comment

sdr shutdown, csp shutdown and csp reboot don't work when sdr-services are run with the start-up script. We'll make a separate issue for that.
The PR is approved and can be merged.

tuvaom merged commit `214017b` into `sdr-services` on Mar 24

2 checks passed

View details Revert

This was referenced on Mar 24

Decide which user sdr-services shall run under #492

Closed

sdr settime #502

Closed

sdr restart #504

Closed

sdr shutdown #503

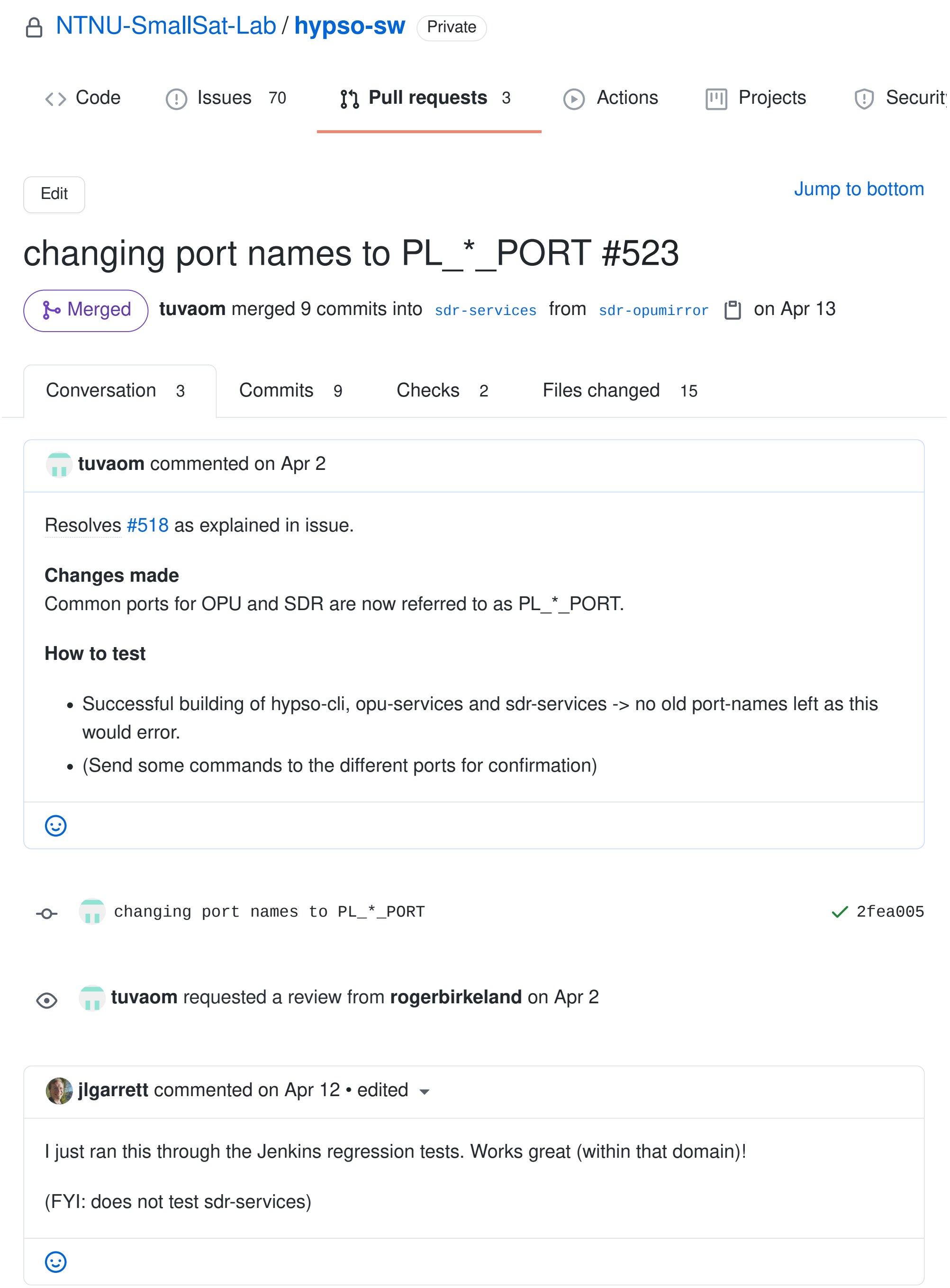
Closed

rogerbirkeland mentioned this pull request on Mar 24

Verify functionality of csp commands on sdr-services #473

Closed

V PR #523 hypso-sw



NTNU-SmallSat-Lab / **hypso-sw** Private

changing port names to PL_*_PORT #523

Merged **tuvaom** merged 9 commits into `sdr-services` from `sdr-opumirror` on Apr 13

tuvaom commented on Apr 2

Resolves #518 as explained in issue.

Changes made
Common ports for OPU and SDR are now referred to as PL_*_PORT.

How to test

- Successful building of hypso-cli, opu-services and sdr-services -> no old port-names left as this would error.
- (Send some commands to the different ports for confirmation)

```
changing port names to PL_*_PORT                                    ✓ 2fea005
```

tuvaom requested a review from **rogerbirkeland** on Apr 2

jlgarrett commented on Apr 12 • edited

I just ran this through the Jenkins regression tests. Works great (within that domain)!

(FYI: does not test sdr-services)

tuvaom added 8 commits on Apr 12

`Change help text of pl restart`	f614f1a
`refactoring opu_services.c and sdr_services.c`	f646762
`adding HYPSO_CSP_BUFFER_COUNT to HYPSO.h`	b77e985
`including new files in CMakeLists.txt`	✓ 77fd326
`Revert "including new files in CMakeLists.txt"` …	3475ec6
`Revert "adding HYPSO_CSP_BUFFER_COUNT to HYPSO.h"` …	91ab599
`Revert "refactoring opu_services.c and sdr_services.c"` …	a25d777
`Revert "Change help text of pl restart"` …	✓ d9489fd

rogerbirkeland approved these changes on Apr 13

View changes

rogerbirkeland left a comment

Tested:
With regression tests on Jenkins --> Pass
Manually: Built and tested briefly on OPU.

rogerbirkeland commented on Apr 13

OK to merge! Tested briefly on OPU, SDR and with Jenkins.

tuvaom merged commit **`79c808c`** into `sdr-services` on Apr 13
2 checks passed

View details Revert

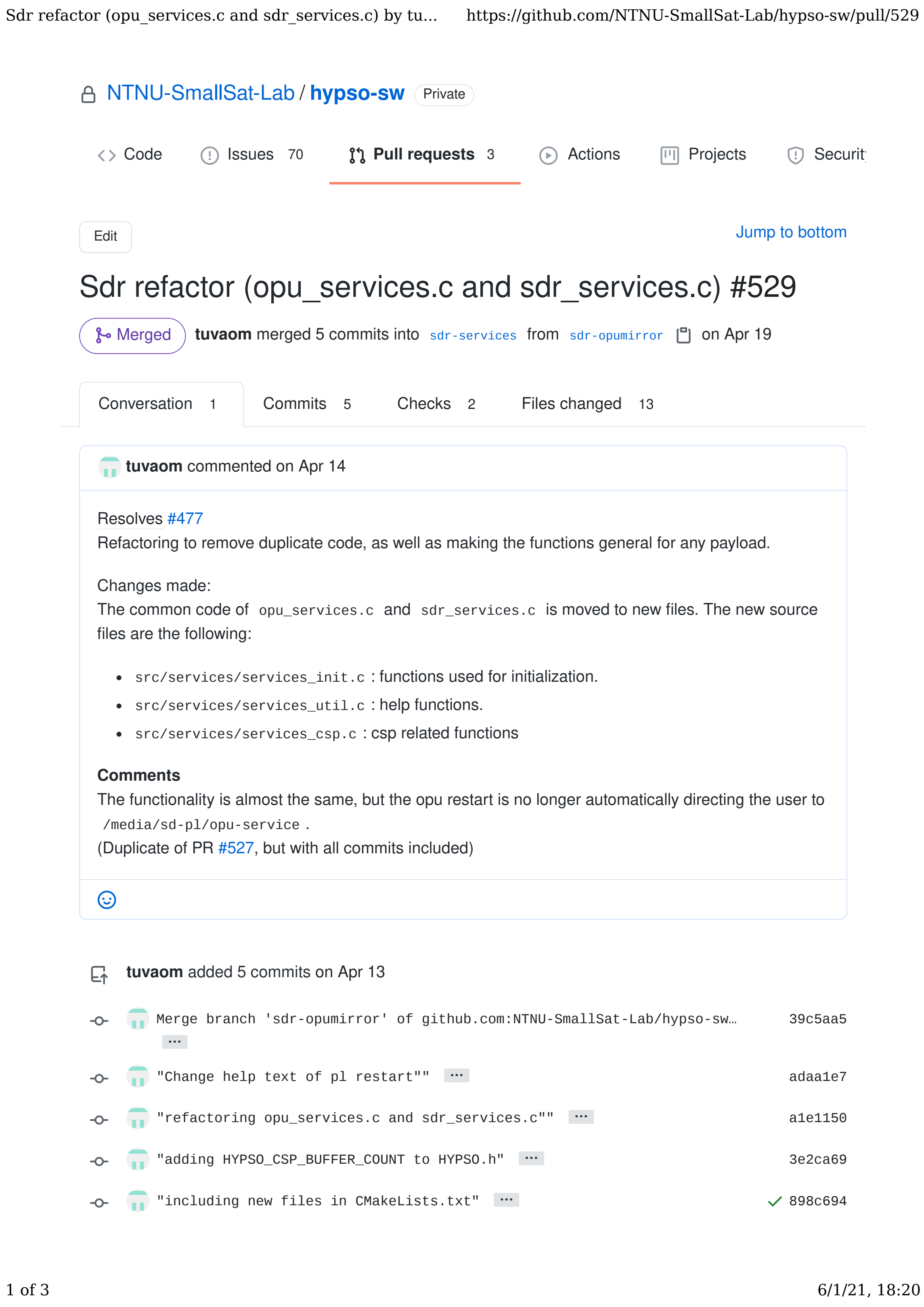
tuvaom mentioned this pull request on Apr 13

Change common ports for OPU and SDR to PL-ports #518

Closed

Reviewers

W PR #529 hypso-sw



Sdr refactor (opu_services.c and sdr_services.c) by tu... https://github.com/NTNU-SmallSat-Lab/hypso-sw/pull/529

NTNU-SmallSat-Lab / **hypso-sw** Private

Sdr refactor (opu_services.c and sdr_services.c) #529

Merged **tuvaom** merged 5 commits into `sdr-services` from `sdr-opumirror` on Apr 19

tuvaom commented on Apr 14

Resolves #477
Refactoring to remove duplicate code, as well as making the functions general for any payload.

Changes made:
The common code of `opu_services.c` and `sdr_services.c` is moved to new files. The new source files are the following:

- `src/services/services_init.c` : functions used for initialization.
- `src/services/services_util.c` : help functions.
- `src/services/services_csp.c` : csp related functions

Comments
The functionality is almost the same, but the opu restart is no longer automatically directing the user to `/media/sd-pl/opu-service` .
(Duplicate of PR #527, but with all commits included)

tuvaom added 5 commits on Apr 13

```
Merge branch 'sdr-opumirror' of github.com:NTNU-SmallSat-Lab/hypso-sw…   39c5aa5
"Change help text of pl restart""   adaa1e7
"refactoring opu_services.c and sdr_services.c""   a1e1150
"adding HYPSO_CSP_BUFFER_COUNT to HYPSO.h"   3e2ca69
"including new files in CMakeLists.txt"   ✓ 898c694
```

1 of 3 6/1/21, 18:20

rogerbirkeland self-requested a review on Apr 14

rogerbirkeland approved these changes on Apr 14

View changes

rogerbirkeland left a comment

Approving, since this is the same as yesterday.

tuvaom merged commit **`9a7dc51`** into `sdr-services` on Apr 19

2 checks passed

View details Revert

jlgarrett added this to **Done** in **SW kanban board** on Apr 29

Reviewers

rogerbirkeland ✓

Assignees

No one—assign yourself

Labels

None yet

Projects

SW kanban board

Done ▾

Milestone

No milestone

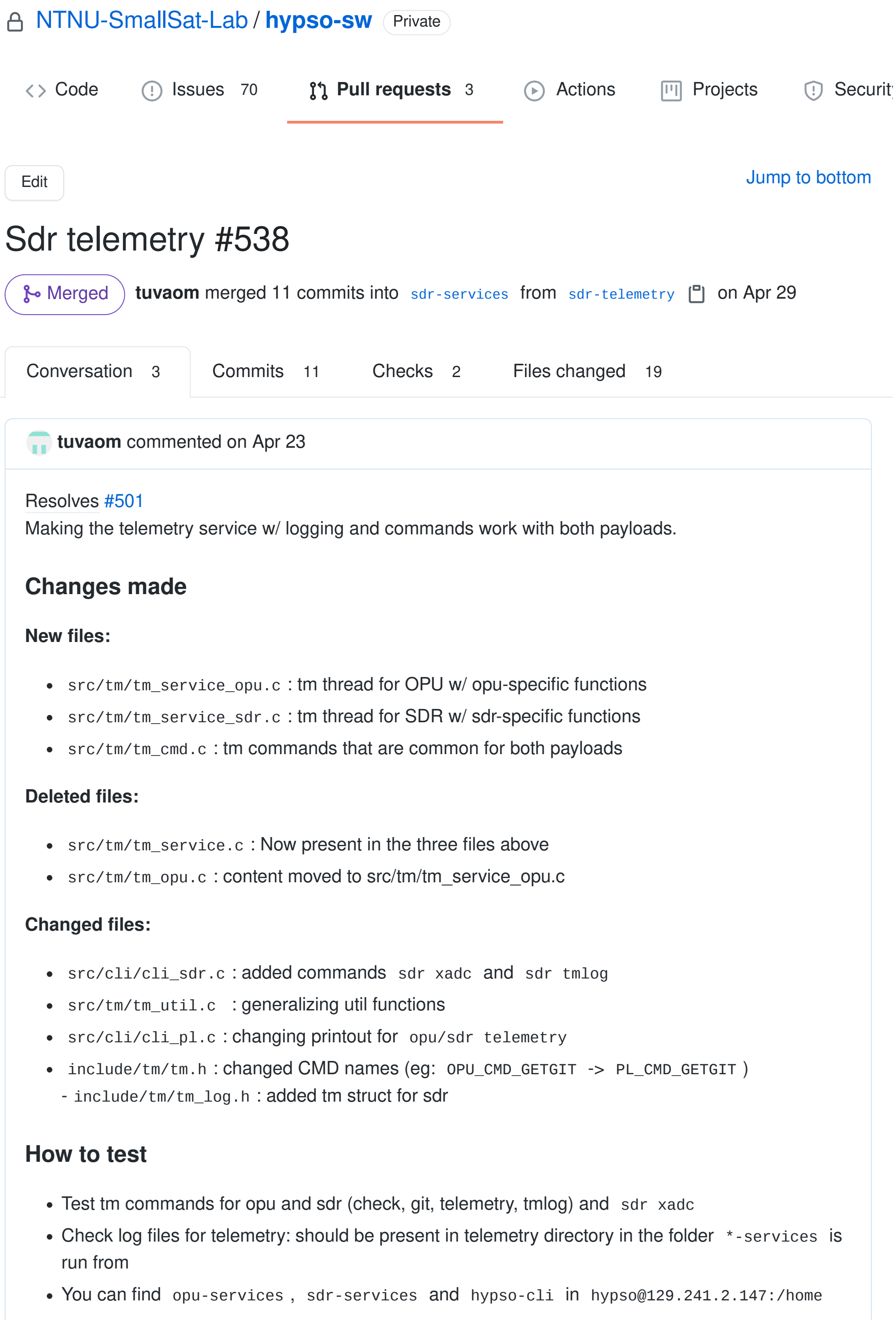
Linked issues

Successfully merging this pull request may close these issues.

None yet

X PR #538 hypso-sw

Sdr telemetry by tuvaom · Pull Request #538 · NTNU-... https://github.com/NTNU-SmallSat-Lab/hypso-sw/pull/538



Sdr telemetry #538

Merged **tuvaom** merged 11 commits into `sdr-services` from `sdr-telemetry` on Apr 29

tuvaom commented on Apr 23

Resolves #501
Making the telemetry service w/ logging and commands work with both payloads.

Changes made

New files:

- `src/tm/tm_service_opu.c` : tm thread for OPU w/ opu-specific functions
- `src/tm/tm_service_sdr.c` : tm thread for SDR w/ sdr-specific functions
- `src/tm/tm_cmd.c` : tm commands that are common for both payloads

Deleted files:

- `src/tm/tm_service.c` : Now present in the three files above
- `src/tm/tm_opu.c` : content moved to src/tm/tm_service_opu.c

Changed files:

- `src/cli/cli_sdr.c` : added commands `sdr xadc` and `sdr tmlog`
- `src/tm/tm_util.c` : generalizing util functions
- `src/cli/cli_pl.c` : changing printout for `opu/sdr telemetry`
- `include/tm/tm.h` : changed CMD names (eg: `OPU_CMD_GETGIT -> PL_CMD_GETGIT`)
  - `include/tm/tm_log.h` : added tm struct for sdr

How to test

- Test tm commands for opu and sdr (check, git, telemetry, tmlog) and `sdr xadc`
- Check log files for telemetry: should be present in telemetry directory in the folder `*-services` is run from
- You can find `opu-services` , `sdr-services` and `hypso-cli` in `hypso@129.241.2.147:/home`

1 of 4 6/1/21, 18:20

`/hypso/tuva/sdr` , and the correct verison of the sdr-services is also uploaded for root in `/home/totem/hypso/sdr-services_tm`

tuvaom added 9 commits on Apr 23

`refactoring common sdr and opu tm-commands`	`ee0b656`
`telemetry service for sdr`	`7e6dce1`
`telemetry service for opu`	`34e642e`
`generalizing util functions`	`dd50e1f`
`Deleting old files after refactoring`	`c543b29`
`Adding functionality 'sdr xadc' and 'sdr tmlog'`	`aa31fe9`
`Correcting telemetry printout for opu and sdr`	`9cacd4f`
`including new telemetry files in appps and CMake`	`557e5fd`
`removed unnecessary print`	✓ `df92377`

tuvaom requested review from **rogerbirkeland** and **garaq** and removed request for **rogerbirkeland** and **garaq** on Apr 23

rogerbirkeland linked an issue that may be closed by this pull request on Apr 28

Telemetry service for the sdr #501 Closed

`Tm Logging time`	✓ `3faa578`

rogerbirkeland reviewed on Apr 29

View changes

`src/tm/tm_service_sdr.c` Outdated

```
+               tm_cmd_sdr_get_telemetry(conn, packet);
+               break;
+           case PL_CMD_TMLOG:
+               snprintf(last_cmd_p->cmd_str, LAST_CMD_STR_LENGTH, "opu tm
```

rogerbirkeland on Apr 29

Should this line say 'opu'?

Reply…

Resolve conversation

rogerbirkeland requested changes on Apr 29

View changes

rogerbirkeland left a comment

The functionality is tested both on OPU and SDR. This looks good.

In the code, there is one reference to a opu-command in one of the SDR-files. Check that before approve/merge.

There should be made a new issue on how to log the xadc-values from the SDR.

`updating lastcmd string 'pl tmlog'` ✓ `cf003ec`

rogerbirkeland approved these changes on Apr 29

View changes

rogerbirkeland left a comment

I think this looks good!

tuvaom merged commit **`f739dcc`** into `sdr-services` on Apr 29
2 checks passed

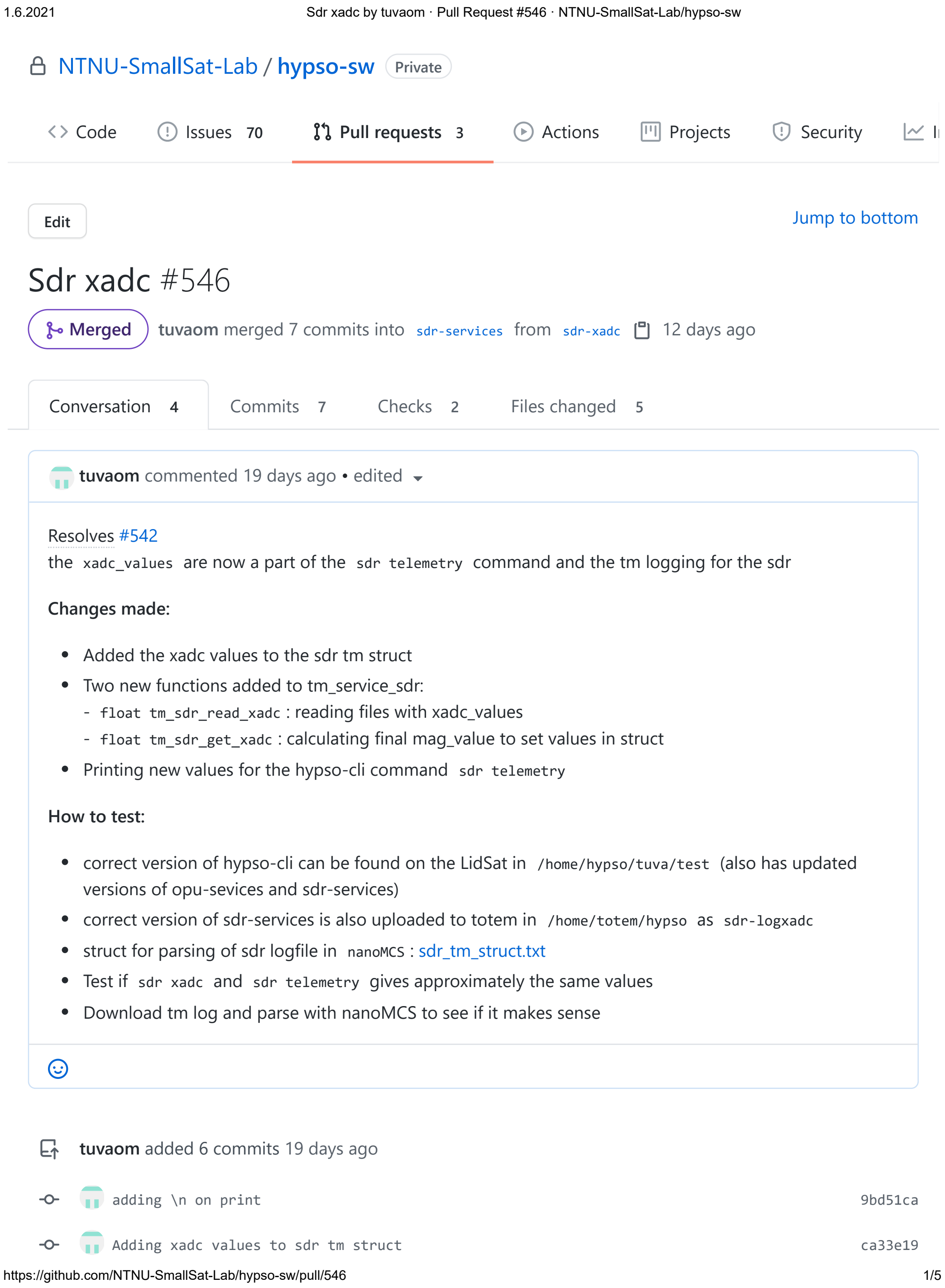
View details Revert

tuvaom mentioned this pull request on Apr 29

Telemetry service for the sdr #501

Closed

Y PR #546 hypso-sw



1.6.2021 Sdr xadc by tuvaom · Pull Request #546 · NTNU-SmallSat-Lab/hypso-sw

Sdr xadc #546

Merged tuvaom merged 7 commits into `sdr-services` from `sdr-xadc` 12 days ago

tuvaom commented 19 days ago • edited

Resolves #542
the `xadc_values` are now a part of the `sdr telemetry` command and the tm logging for the sdr

Changes made:

- Added the xadc values to the sdr tm struct
- Two new functions added to tm_service_sdr:
  - `float tm_sdr_read_xadc` : reading files with xadc_values
  - `float tm_sdr_get_xadc` : calculating final mag_value to set values in struct
- Printing new values for the hypso-cli command `sdr telemetry`

How to test:

- correct version of hypso-cli can be found on the LidSat in `/home/hypso/tuva/test` (also has updated versions of opu-sevices and sdr-services)
- correct version of sdr-services is also uploaded to totem in `/home/totem/hypso` as `sdr-logxadc`
- struct for parsing of sdr logfile in `nanoMCS` : sdr_tm_struct.txt
- Test if `sdr xadc` and `sdr telemetry` gives approximately the same values
- Download tm log and parse with nanoMCS to see if it makes sense

tuvaom added 6 commits 19 days ago

```
adding \n on print                                             9bd51ca
Adding xadc values to sdr tm struct                            ca33e19
```

https://github.com/NTNU-SmallSat-Lab/hypso-sw/pull/546 1/5

```
collecting xadc values for sdr tm struct                     72e6c36
'sdr telemetry' now printing collected xadc values           28cef71
FOrmat telemetry sdr print                                   1cd621d
checking for correct xadc file (scale not offset)          ✓ 4a060a6
```

tuvaom requested a review from **rogerbirkeland** 19 days ago

rogerbirkeland reviewed 13 days ago

View changes

src/tm/tm_service_sdr.c Outdated

```
+     tm_log_data->vrefN = tm_sdr_get_xadc("in_voltage7_vrefn_raw", NULL,
                                            "in_voltage7_vrefn_scale", 0, 1);
+     tm_log_data->currVCC5V0 =
+         tm_sdr_get_xadc("in_voltage8_raw", NULL, NULL, 0, 0.0006103516);
```

rogerbirkeland 13 days ago

It would have been nicer if the scales and offsets were defines or constants instead if hard-coded into the function calls.

tuvaom 13 days ago Author

Thanks for feedback, updated with constant definitions now.

Reply...

Resolve conversation

rogerbirkeland self-requested a review 13 days ago

rogerbirkeland reviewed 13 days ago

View changes

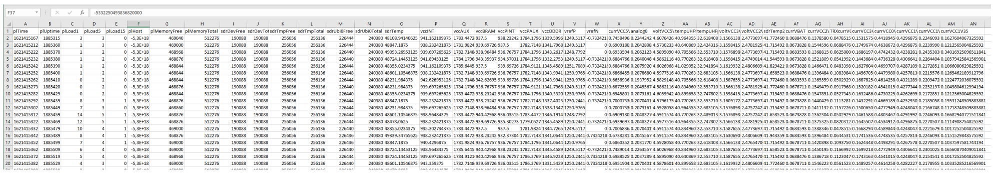
rogerbirkeland left a comment

This looks nice. The xadc-values and telemetry values are similar:

```
(hypso) sdr telemetry
Sending telemetry request
===========================  Telemetry Info  ===========================
                 --------- System values ---------
Uptime(sec since boot): 1888035
Memory(kB) -                free: 468472                total: 512276
Dev(kB) -                 free: 190088                total: 190088
Tmp(kB) -                free: 256056                total: 256136
Ubi0(kB) -               free: 226408                total: 240380
Load*100 -          1 min: 5       5 min: 7       15 min: 1
Image :             totem

                  --------- ADC values -------
Temperature(mC): 48601.105469           UHF PA Temp.(C): 32.618408
Vcc INT(mV): 941.162109                 VCC3V3 Voltage(V): 3.156614
Vcc AUX(mV): 1784.912109                VCC2V5 Voltage(V): 2.477370
Vcc BRAM(mV): 938.232422                Temperature(C): 41.829422
Vcc PINT(mV): 938.232422                VBAT Current(mA): 0.068848
Vcc PAUX(mV): 1783.447266               VCC2V5 Current(mA): 0.153646
Vcc ODDR(mV): 1343.994141               VCC3V3 TRX Curr.(mA): 0.121582
Vref P(mV): 1250.976562                 VCC3V3 Current(mA): 0.155599
Vref N(mV): 0.000000                    VCC0V95 Current(mA): 0.459595
VCC5V0 Current(mA): 0.707398            VCC1V3 Current(mA): 0.423340
Analog 0: 0.212007                      VCC1V8 Current(mA): 0.220947
VCC5V0 Voltage(V): 4.577969             VCC1V35 Current(mA): 0.104574
UHF Frontend Temp.(C):40.640869
=======================================================================
(hypso) sdr xadc
ID  Name                    Raw value    Mag value Unit
--  ----                    ---------    --------- ----
 0  Temperature           48355.0232   48355.0232   mC
 1  Vcc INT                 934.5703     934.5703   mV
 2  Vcc AUX                1784.1797    1784.1797   mV
 3  Vcc BRAM                940.4297     940.4297   mV
 4  Vcc PINT                936.0352     936.0352   mV
 5  Vcc PAUX               1781.2500    1781.2500   mV
 6  Vcc ODDR               1343.9941    1343.9941   mV
 7  Vref P                 1249.5117    1249.5117   mV
 8  Vref N                    0.0000       0.0000   mV
 9  VCC5V0 Current         1134.0000       0.6921   mA
10  Analog 0                883.0000       0.2258
11  VCC5V0 Voltage         2784.0000       4.5895    V
12  UHF Frontend Temp.     1404.0000      40.8350    C
13  UHF PA Temp.           1279.0000      32.7478    C
14  VCC3V3 Voltage         2937.0000       3.1598    V
15  VCC2V5 Voltage         3009.0000       2.4757    V
16  Temperature            1611.0000      41.7725    C
17  VBAT Current            137.0000       0.0669   mA
18  VCC2V5 Current          864.0000       0.1406   mA
19  VCC3V3 TRX Curr.        122.0000       0.0596   mA
20  VCC3V3 Current          918.0000       0.1494   mA
```

```
    21  VCC0V95 Current          920.0000       0.5615    mA
    22  VCC1V3 Current           888.0000       0.4336    mA
    23  VCC1V8 Current           367.0000       0.2240    mA
    24  VCC1V35 Current          296.0000       0.1204    mA

    (hypso)
```

The TM files can be parsed and look like this:



F37 -5332250493836820000

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN
1	plTime	plUptime	plLoad1	plLoad5	plLoad15	plHost	plMemoryFree	plMemoryTotal	sdrDevFree	sdrDevTot	sdrTmpFree	sdrTmpTotal	sdrUbi0Free	sdrUbi0Total	sdrTemp	vccINT	vccAUX	vccBRAM	vccPINT	vccPAUX	vccODDR	vrefP	vrefN	currVCC5	analog0	voltVCC5	tempUHF	tempUHF	voltVCC3	voltVCC2	sdrTemp2	currVBAT	currVCC2	TRXcurrV	currVCC3	currVCC0	currVCC1	currVCC1	currVCC1V35	
2	1621415167	1885315	3	3	0	-5,3E+18	469040	512276	190088	190088	256056	256136	226444	240380	48108.94140625	941.162109375	1783.4472	937.5	938.23242	1784.1796	1339.5996	1249.5117	-0.732421	0.7458496	0.2244242	4.6076426	40.964355	32.618408	3.1566138	2.4773697	41.772460	0.0688476	0.1378580	0.0478515	0.1531575	0.4418945	0.4296875	0.2246093	0.1627604067325592	
3	1621415212	1885360	1	3	0	-5,3E+18	469000	512276	190088	190088	256056	256136	226444	240380	48847.1875	938.232421875	1781.9824	939.69726	937.5	1782.7148	1341.7968	1249.5117	0	0.6909180	0.2042808	4.5730233	40.640869	32.424316	3.1598415	2.4781925	41.715492	0.0673828	0.1544596	0.0688476	0.1749674	0.4638672	0.4296875	0.2239990	0.1212565004825592	
4	1621415222	1885370	1	2	0	-5,3E+18	468968	512276	190088	190088	256056	256136	226440	240380	49093.26953125	939.697265625	1782.7148	938.96484	936.76757	1784.1796	1343.2617	1248.7792	0	0.6933594	0.2062123	4.5895090	40.705566	32.553710	3.1576898	2.4773697	41.715492	0.0683593	0.1368815	0.0625000	0.1686197	0.4742432	0.4238281	0.2435303	0.34016925096511841	
5	1621415232	1885380	1	2	0	-5,3E+18	468876	512276	190088	190088	256056	256136	226440	240380	48724.14453125	941.89453125	1784.1796	943.35937	934.57031	1784.1796	1332.2753	1249.5117	-0.732421	0.6884766	0.2040048	4.5862116	40.770263	32.618408	3.1598415	2.4749014	41.544593	0.0673828	0.1521809	0.0541992	0.1443684	0.4736328	0.4306641	0.2264404	0.10579425841569901	
6	1621415242	1885390	1	2	0	-5,3E+18	468844	512276	190088	190088	256056	256136	226444	240380	48355.0234375	941.162109375	1785.6445	937.5	939.69726	1784.9121	1345.4589	1251.7089	-0.732421	0.6884766	0.2075920	4.6026968	41.029052	32.941894	3.1619932	2.4806609	41.829421	0.0673828	0.1466471	0.0483398	0.1627604	0.4699707	0.4287109	0.2172851	0.1066080629825592	
7	1621415252	1885400	1	2	0	-5,3E+18	468844	512276	190088	190088	256056	256136	226448	240380	48601.10546875	938.232421875	1782.7148	939.69726	936.76757	1782.7148	1343.9941	1250.9765	-0.732421	0.6866455	0.2078680	4.5977516	40.770263	32.683105	3.1566138	2.4773697	41.658523	0.0688476	0.1396484	0.1083984	0.1456705	0.4479980	0.4257813	0.2215576	0.1265462189912796	
8	1621415262	1885410	0	2	0	-5,3E+18	468844	512276	190088	190088	256056	256136	226444	240380	48231.984375	942.626953125	1782.7148	942.62695	939.69726	1784.1796	1343.9941	1250.9765	-0.732421	0.6658936	0.1937952	4.5829148	40.705566	32.618408	3.1587655	2.4773697	41.772460	0.0683593	0.1365559	0.0502929	0.1687825	0.4614258	0.4321289	0.2209472	0.1224772036075592	
9	1621415273	1885420	0	2	0	-5,3E+18	468876	512276	190088	190088	256056	256136	226440	240380	48231.984375	939.697265625	1784.1796	936.76757	936.76757	1784.9121	1341.7968	1249.5117	-0.732421	0.6872559	0.2045567	4.5862116	40.834960	32.553710	3.1566138	2.4781925	41.772460	0.0678711	0.1549479	0.0917968	0.1520182	0.4541015	0.4277344	0.2252197	0.1049804612994194	
10	1621415282	1885429	0	2	0	-5,3E+18	468844	512276	190088	190088	256056	256136	226440	240380	48355.0234375	939.697265625	1783.4472	938.23242	936.76757	1784.1796	1340.3320	1250.9765	-0.732421	0.6945801	0.2073161	4.6059942	40.899658	32.747802	3.1598415	2.4773697	41.715492	0.0688476	0.1547851	0.0527343	0.1632486	0.4730225	0.4262695	0.2172851	0.1212565004825592	
11	1621415292	1885439	8	3	1	-5,3E+18	468876	512276	190088	190088	256056	256136	226444	240380	48847.1875	938.232421875	1783.4472	938.23242	936.76757	1782.7148	1337.4023	1250.2441	-0.732421	0.7000733	0.2070401	4.5796175	40.770263	32.553710	3.1609172	2.4773697	41.715492	0.0673828	0.1440429	0.1113281	0.1432291	0.4669189	0.4252930	0.2185058	0.19531248509883881	
12	1621415302	1885449	7	3	0	-5,3E+18	468860	512276	190088	190088	256056	256136	226444	240380	48231.984375	939.697265625	1782.7148	938.96484	936.76757	1782.7148	1338.1347	1250.9765	0	0.7000733	0.2073161	4.5928058	40.964355	32.683105	3.1576898	2.4757242	41.715492	0.0678711	0.1411132	0.1157226	0.1500650	0.4772949	0.4248047	0.2166748	0.11718748509883881	
13	1621415312	1885459	14	5	1	-5,3E+18	468876	512276	190088	190088	256056	256136	226444	240380	48601.10546875	938.96484375	1783.4472	940.42968	936.03515	1783.4472	1346.1914	1248.7792	0	0.6909180	0.2048327	4.5911574	40.770263	32.489013	3.1576898	2.4757242	41.658523	0.0673828	0.1362304	0.0502929	0.1461588	0.4803467	0.4291992	0.2246093	0.16682940721511841	
14	1621415322	1885469	12	5	1	-5,3E+18	468876	512276	190088	190088	256056	256136	226444	240380	48478.0625	938.232421875	1783.4472	939.69726	935.30273	1779.0527	1345.4589	1250.2441	-0.732421	0.6939697	0.2048327	4.5977516	40.964355	32.747802	3.1566138	2.4781925	41.658523	0.0678711	0.1375325	0.0820312	0.1645507	0.4534912	0.4296875	0.2270507	0.1114908754825592	
15	1621415332	1885479	10	4	1	-5,3E+18	469000	512276	190088	190088	256056	256136	226440	240380	48355.0234375	935.302734375	1783.4472	937.5	937.5	1781.9824	1344.7265	1249.5117	0	0.7006836	0.2070401	4.5911574	40.834960	32.553710	3.1587655	2.4773697	41.715492	0.0683593	0.1388346	0.0478515	0.1668294	0.4589844	0.4248047	0.2221679	0.1017252504825592	
16	1621415342	1885489	8	4	1	-5,3E+18	468876	512276	190088	190088	256056	256136	226436	240380	49339.34765625	938.232421875	1783.4472	938.23242	932.37304	1782.7148	1341.0644	1250.2441	0.7324218	0.6738281	0.2045567	4.5911574	40.834960	32.683105	3.1630690	2.4806609	41.943359	0.0683593	0.1396484	0.0644531	0.1741536	0.4748535	0.4257813	0.2246093	0.1151529848575592	
17	1621415352	1885499	7	4	1	-5,3E+18	469000	512276	190088	190088	256056	256136	226436	240380	48847.1875	940.4296875	1781.9824	936.76757	936.76757	1784.1796	1341.0644	1250.9765	0	0.6860352	0.2031770	4.5928058	40.770263	32.618408	3.1566138	2.4765470	41.715492	0.0678711	0.1420898	0.1093750	0.1624348	0.4498291	0.4267578	0.2270507	0.1037597581744194	
18	1621415362	1885509	6	4	0	-5,3E+18	469000	512276	190088	190088	256056	256136	226440	240380	48724.14453125	938.96484375	1785.6445	940.42968	938.23242	1782.7148	1345.4589	1249.5117	-0.732421	0.7489014	0.2263557	4.6026968	40.834960	32.683105	3.1587655	2.4773697	41.658523	0.0678711	0.1450195	0.1166992	0.1699218	0.4772949	0.4306641	0.2301025	0.16560870409011841	
19	1621415372	1885519	5	4	0	-5,3E+18	468968	512276	190088	190088	256056	256136	226444	240380	48724.14453125	939.697265625	1784.9121	940.42968	936.76757	1786.3769	1346.9238	1250.2441	0.7324218	0.6988525	0.2037289	4.5895090	40.640869	32.553710	3.1587655	2.4765470	41.715492	0.0688476	0.1386718	0.1123047	0.1743163	0.4541015	0.4248047	0.2154541	0.1017252504825592	
20	1621415382	1885529	4	3	0	-5,3E+18	469000	512276	190088	190088	256056	256136	226440	240380	48601.10546875	943.359375	1782.7148	939.69726	936.03515	1786.3769	1331.5429	1250.2441	0.7324218	0.6951904	0.2070401	4.5878601	40.899658	32.683105	3.1619932	2.4806609	41.772460	0.0678711	0.1546223	0.0561523	0.1489257	0.4614258	0.4282227	0.2178955	0.10335285216569901	

☺

rogerbirkeland mentioned this pull request 13 days ago

Logging of xadc_values #542

Closed

rogerbirkeland linked an issue that may be closed by this pull request 13 days ago

Logging of xadc_values #542 Closed

`xadc scales and offsets defines as constants` ✓ cdce64e

rogerbirkeland approved these changes 12 days ago

View changes

rogerbirkeland left a comment

Looks good!

☺

tuvaom merged commit **d8f0049** into `sdr-services` 12 days ago View details Revert

2 checks passed

tuvaom deleted the `sdr-xadc` branch 12 days ago

Restore branch

jlgarrett added this to **Done** in **SW kanban board** 7 days ago

Reviewers

rogerbirkeland ✓

Assignees

No one—assign yourself

Labels

None yet

Projects

SW kanban board

Done ▾

Milestone

No milestone

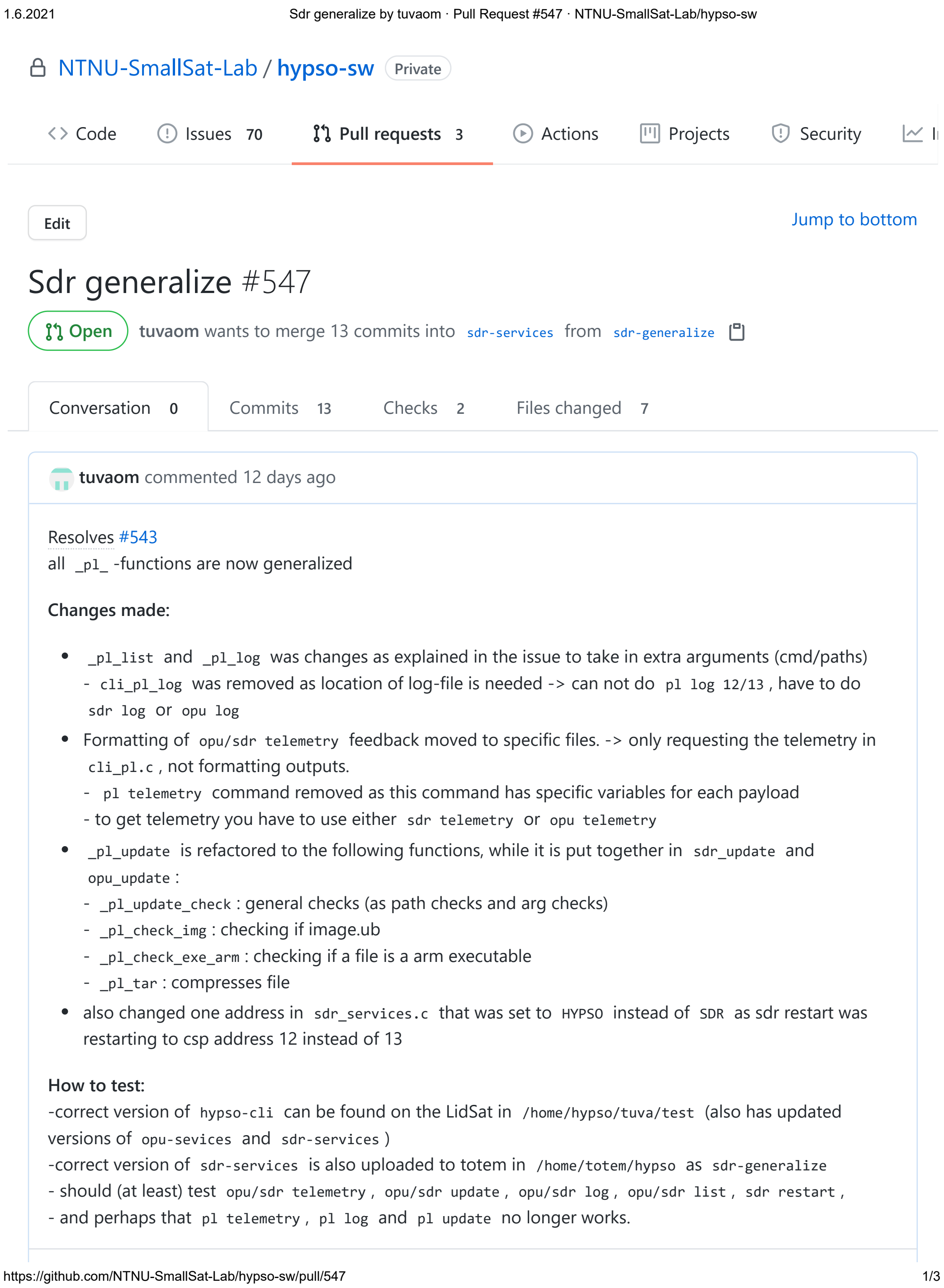
Linked issues

Successfully merging this pull request may close these issues.

Logging of xadc_values

2 participants

Z PR #547 hypso-sw



1.6.2021 Sdr generalize by tuvaom · Pull Request #547 · NTNU-SmallSat-Lab/hypso-sw

NTNU-SmallSat-Lab / **hypso-sw** Private

Sdr generalize #547

Open **tuvaom** wants to merge 13 commits into `sdr-services` from `sdr-generalize`

tuvaom commented 12 days ago

Resolves #543
all `_pl_` -functions are now generalized

Changes made:

- `_pl_list` and `_pl_log` was changes as explained in the issue to take in extra arguments (cmd/paths)
  - `cli_pl_log` was removed as location of log-file is needed -> can not do `pl log 12/13`, have to do `sdr log` or `opu log`
- Formatting of `opu/sdr telemetry` feedback moved to specific files. -> only requesting the telemetry in `cli_pl.c`, not formatting outputs.
  - `pl telemetry` command removed as this command has specific variables for each payload
  - to get telemetry you have to use either `sdr telemetry` or `opu telemetry`
- `_pl_update` is refactored to the following functions, while it is put together in `sdr_update` and `opu_update`:
  - `_pl_update_check`: general checks (as path checks and arg checks)
  - `_pl_check_img`: checking if image.ub
  - `_pl_check_exe_arm`: checking if a file is a arm executable
  - `_pl_tar`: compresses file
- also changed one address in `sdr_services.c` that was set to `HYPSO` instead of `SDR` as sdr restart was restarting to csp address 12 instead of 13

How to test:
-correct version of `hypso-cli` can be found on the LidSat in `/home/hypso/tuva/test` (also has updated versions of `opu-sevices` and `sdr-services`)
-correct version of `sdr-services` is also uploaded to totem in `/home/totem/hypso` as `sdr-generalize`
- should (at least) test `opu/sdr telemetry`, `opu/sdr update`, `opu/sdr log`, `opu/sdr list`, `sdr restart`,
- and perhaps that `pl telemetry`, `pl log` and `pl update` no longer works.

https://github.com/NTNU-SmallSat-Lab/hypso-sw/pull/547 1/3

tuvaom added 13 commits on 29 Apr

generalize _pl_log and _pl_list	e439243
refactoring opu telemetry and sdr telemetry	d828cfb
refalctor pl tm and remove not generalized pl funcs	61cef09
refactoring and generalizing pl update	0bc92af
add changes from sdr-xadc branch	78954a1
Merge branch 'sdr-xadc' into sdr-generalize	ac00eb7
Add cat to pl log cmd	55f5a80
removing cleanup cmd -> moved to other files	06b55d2
removing test print	f6ea8b9
adding z to zip properly	e4e307d
Merge branch 'sdr-services' into sdr-generalize	f41cfda
not tar-ing sdr-services	99cd5ac
sdr monitor task to correct csp address	✓ b498272

tuvaom requested a review from **rogerbirkeland** 12 days ago

jlgarrett added this to **Review in progress** in **SW kanban board** 7 days ago

Reviewers

rogerbirkeland ●

Assignees

No one—assign yourself

Labels

None yet

Tuva Okkenhaug Moxnes

A common software framework for a CubeSat with multiple payloads

NTNU

Norwegian University of
Science and Technology