Master's thesis

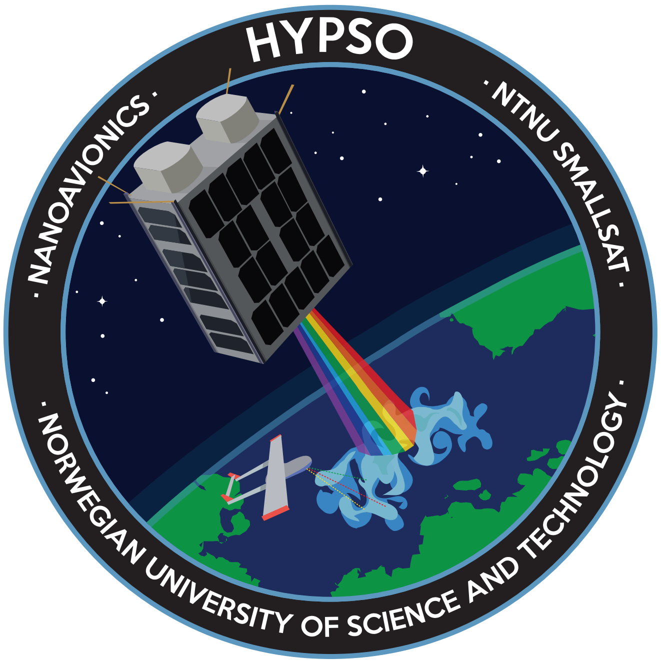
NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Øyvind Paulsen Skaaden

System Integration of the HYPSO-2 SDR

Enabling Fault-Tolerant Payload Operations and an Efficient Development Environment

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Roger Birkeland
June 2023

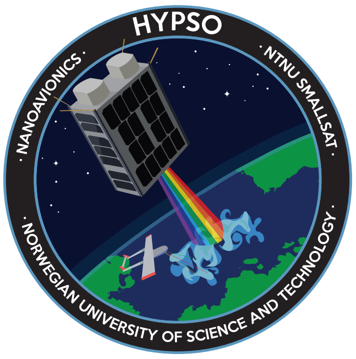


NTNU
Norwegian University of
Science and Technology

Øyvind Paulsen Skaaden

System Integration of the HYPSO-2 SDR

Enabling Fault-Tolerant Payload Operations and an Efficient Development Environment



Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Roger Birkeland
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Abstract

In this thesis, the author discusses how the HYPSO-2 Software-Defined Radio (SDR) Flight Model (FM) is adapted and made ready for integration. The focus is on ease of implementation and usability of the build systems from the developers' perspective and ensuring system resilience during operations in space. The SDR is based on the TOTEM SDR, developed by Alén Space, Spain.

An understandable, reusable, and maintainable development environment is developed by utilizing Buildroot (BR) features, such as packages, external trees, and toolchains. A clear separation of the original configuration by the TOTEM SDR developers, and the adaptations made by the author and the Norwegian University of Science and Technology (NTNU) SmallSat Lab (NSSL) are established to promote reusability and maintainability. The resulting toolchain created by BR is further utilized to develop and implement a framework for developing SDR applications that are automatic, flexible, and easy to use.

The system's resilience against adverse events and conditions is greatly improved by introducing a "chained" startup script incorporating multiple locations for mission-critical Software (SW). In the rare event of failure of all startup locations, the system can disable the writable part of storage, resetting the system back to a known state, increasing resilience, and prolonging the system's expected lifetime.

The contributions implemented by the author will help other students focus on developing SDR applications, promoting rapid development, and maximizing the research potential of the TOTEM SDR.

Sammendrag

I denne masteroppgaven vil forfatteren diskutere og forklare hvordan den programvaredefinerte radioen (SDR) for HYPSO-2 satellitten er tilpasset og gjort klar for integrasjon. Byggesystemene er tilpasset for å gjøre det enkelt å ta i bruk for andre utviklere og studenter. I tillegg er systemets robusthet mot uønskede hendelser og forhold, som for eksempel stråling, forbedret betraktelig i forhold til tidligere versjoner. SDR systemet som brukes er Alén Spaces (Spania) TOTEM SDR plattform.

Ved å benytte funksjoner fra byggesystemet BuildRoot (BR) som pakker, eksterne katalogtrær og verktøykjeder, er det etablert et utviklingsmiljø som er enkelt å forstå og vedlikeholde. For å øke brukervennligheten og potensialet for gjenbruk av kode, er det nå et tydelig skille mellom den originale konfigurasjonen fra Alén Space og tilpasningene gjort av forfatteren og NTNU SmallSat Lab (NSSL). Verktøykjeden BR lager brukes videre for å utvikle og implementere et automatisk og fleksibelt rammeverk for effektiv utvikling av SDR applikasjoner.

Systemets robusthet mot uønskede hendelser og forhold er betraktelig forbedret ved å ta i bruk et oppstartskript som har mulighet for flere "oppstartssteder" for operasjonskritisk programvare (SW). Dersom oppstartsskriptet ikke klarer å starte kritisk programvare fra noen av oppstartsstedene, kan systemet gjøre nødvendige endringer for å koble ut den skrivbare delen av lagringsenheten. Dette vil tvinge systemet inn i en kjent tilstand som igjen vil føre til økt robusthet og øke den forventede levetiden til systemet.

Forfatterens bidrag vil hjelpe utviklere og studenter til å fokusere på utviklingen av SDR applikasjoner. Dette vil bidra til en høyere utnyttelse av forskningspotensialet til HYPSO-2 SDR systemet.

Preface

Contributions and Aims

The contributions from this thesis are mostly qualitative enhancements to the usability and maintainability of the build systems associated with the Hardware (HW) and Software (SW) for the HYPerspectral Smallsat for Ocean observation (HYPSO)-2 Software-Defined Radio (SDR) payload subsystem. These changes are hard to measure directly other than feedback from the developers that utilize the build systems. Enhancements have been implemented incrementally and discussed thoroughly within the Norwegian University of Science and Technology (NTNU) SmallSat Lab (NSSL) to explore what works and what does not.

Private GitHub repositories and Internal Documents

During the development of this thesis, GitHub was used extensively for version control, collaboration, and source code sharing. The NSSL manages and owns an organization on GitHub `https://github.com/NTNU-SmallSat-Lab`, having numerous private repositories.

`https://github.com/NTNU-SmallSat-Lab/sdr-system`
The primary repository contains the original build system for the EM TOTEM.

`https://github.com/NTNU-SmallSat-Lab/sdr-system-h2`
Fork of the primary repository containing the build system for the HYPSO-2 TOTEM.

`https://github.com/NTNU-SmallSat-Lab/sdr-system-hypso-shared`
Shared Buildroot external tree. Shared between the different TOTEM build systems.

`https://github.com/NTNU-SmallSat-Lab/sdr-applications`
SDR application development repository can utilize the different build systems/toolchains.

`https://github.com/NTNU-SmallSat-Lab/hypso-sw`
NSSL repository containing SW for enabling communications on the satellite and payload operations.

The first four repositories were developed extensively during the thesis. To access the above repositories, contact Roger Birkeland by email `roger.birkeland@ntnu.no`.

Previous Work

Before working on this masters thesis, the author wrote a specialization project report [1]. Some chapters and figures from the specialization project report are relevant to this masters thesis. The sections listed below can be found with varying degrees of similarity in the specialization project report:

Chapter 1: Introduction *Sections 1.1 to 1.3* is based on the same sections as in [1].

Chapter 2: Background Sections 2.1 and 2.2 contains material similar to [1] except for *Section 2.2.2: External Trees*. The subsections *Unsorted Block Images* and *Overlay Filesystems* in Section 2.1.2 are from [1]. The section *Section 2.4: The TOTEM SDR System* describing the TOTEM contain parts from [1], most notably *Section 2.4.2: Storage Layout*.

Chapter 3: Methods and Tools *Sections 3.1, 3.3 and 3.4* contain parts that are similar to the same sections in [1]. *Section 3.3.2: Stable Environment with Docker* is from the specialization project [1].

Conventions

File Paths

In this thesis, file paths and directory paths will be referenced. Both Linux File System (FS) paths and repository paths will be used.

Linux Files File paths in the Linux FS will be written `/etc/environment`.

Linux Directories Directory paths in the Linux FS will be written `/etc/init.d/`.

Repository Files File paths in a repository will be written `directory/file`.

Repository Directories Directory paths in a repository will be written `directory/directory/`.

All directories end with a "/", all Linux paths start with the Linux Tux and a "/", and all repositories start with a small folder.

Code Blocks

Multiple code blocks will be used. An inline code block `looks like this!` . This is used for configuration entries and commands like this `rm -rf /*` .

A listing will be written like this:

```
def main()
    print("Hello, HYPSO!")
```

Acknowledgments

I want to thank the entire HYPSO team for being awesome. Simen Berg, Dennis Lager, and Sivert Bakken have always been at the Lab, available for questions a discussion. The SDR team, Giacomo Melloni and Magnhild Eeg, for great discussions on the SDR. Thank you to Roger Birkeland and Milica Orlandic for being my supervisors, constantly answering my silly questions, and engaging in discussions.

And last but not least, a huge thank you to my family and friends for helping and supporting me through this thesis.

Contents

Abbreviations

ADC Analog-to-Digital Converter. 19

ADCS Attitude Determination and Control System. 17, 36

AMOS Autonomous Marine Operations and Systems. 2, 18

AR Acceptance Review. 25

BR Buildroot. iii, 13–17, 19, 22, 28, 32, 33, 40–47, 54–56, 63, 67

CAN Controller Area Network. 18, 36–38, 48, 65, 66

CD Compact Disc. 10

CDR Critical Design Review. 25, 35

COM Communications System. 3, 17, 36

CONOPS Concept of Operations. 2

COTS Commercial Off-The-Shelf. 2–4

CPCL Cal Poly CubeSat Laboratory. 2

CRR Commissioning Result Review. 25

CSP CubeSat Protocol. 18, 21, 36–38

DAC Digital-to-Analog Converter. 19

DTB Device Tree Blob. 10, 14, 22, 41

DVD Digital Video Disc. 10

ECSS European Cooperation for Space Standardization. 24

EEPROM Electrically Erasable Programmable Read-Only Memory. 40, 47, 48

ELR End-of-Life Review. 25

EM Engineering Model. 38, 40, 46, 56, 61, 63, 64

EO Earth Obeservation. 1

EPS Electrical Power System. 3, 18, 36, 37, 48

ESA European Space Agency. 1

ESD Electrostatic Discharge. 64, 65

FC Flight Computer. 17, 36

FM Flight Model. iii, 4, 39, 40, 43, 45, 64–66

FOSS Free and Open-Source Software. xx, 9

FPGA Field-Programmable Gate Array. 3, 10, 19, 21

FRR Flight Readiness Review. 25

FS File System. viii, 4, 8, 10, 12–15, 21, 22, 33, 36, 41, 42, 44, 54–60, 68

FSBL First-Stage Bootloader. 9, 10, 21

FW Firmware. 4, 10, 32, 39, 40, 44, 45, 48, 61, 68

GPOS General-Purpose Operating System. 8

GPU Graphics Processing Unit. 8

GUI Graphical User Interface. 8

HAB Harmful Algal Bloom. 2

HDD Hard Disk Drive. 10, 11

HID Human Interface Device. 8

HSI Hyperspectral Imager. 1, 2, 18, 33, 35, 37, 54

HW Hardware. vii, xix, 3, 8–10, 13–15, 41, 48, 64

HYPSO HYPerspectral Smallsat for Ocean observation. vii, 1, 2, 4, 17–19, 21, 31–33, 35–37, 39–48, 61, 64, 66, 68

ICD Interface Control Documents. 65

IO Input/Output. 48

IoT Internet of Things. 18

IP Internet Protocol. 32, 41, 42, 44, 48, 65

ISS International Space Station. 3

LEB Logical Erase Block. 11, 12

LRR Launch Readiness Review. 25

LTP Latest Takes Precedence. 41, 42

LV Launch Vehicle. 3

MCR Mission Close-out Review. 25

MDR Mission Definition Review. 24

MRD Mission Requirements Document. 1

MTD Memory Technology Device. 11, 21

MVP Minium Viable Product. 40

NA Kongsberg NanoAvionics. 3, 4, 17, 18, 35–37, 65

NASA National Aeronautics and Space Administration. 24

NFR Norges forskningsråd (The Research Council of Norway). 1

NSA Norwegian Space Agency. 1

NSSL NTNU SmallSat Lab. iii, vii, 1, 17–19, 21, 24, 30, 35–37, 40–42, 45, 56, 59, 61, 64, 66, 67

NTNU Norwegian University of Science and Technology. iii, vii, xvii, 1, 17, 18

OPU On-board Processing Unit. 33, 35–37

ORR Operational Readiness Review. 25

OS Operating System. xx, 8–10, 12–15, 17–19, 22, 35, 36, 51, 56, 68

PC Payload Controller. 18, 36, 37

PCB Printed Circuit Board. 18, 19

PDR Preliminary Design Review. 25, 35

PEB Physical Erase Block. 11, 12, 22

PR Pull Request. xix, 29–34, 61, 62

PRR Preliminary Requirements Review. 25

QR Qualification Review. 25

RAM Random-Access Memory. 9, 10, 21, 56

RF Radio Frequency. xix, 3, 4, 18, 19, 21, 37

RGB Red-Green-Blue. 1, 2, 18, 37

RO Read-Only. 10, 13, 33, 42, 54–56, 59, 68

ROM Read-Only Memory. 9, 10

RTOS Real-Time Operating System. 9

RW Read-Write. 13, 56–58

SCM Source-Control Management. 27

SD Secure Digital. xix

SDR Software-Defined Radio. iii, vii, xx, 2–5, 7, 14, 17–19, 21, 31–48, 50–52, 56, 59, 61–64, 66–68

SoC System-on-Chip. xx, 18–20, 47, 65

SRR System Requirements Review. 25

SSBL Second-Stage Bootloader. 10

SSD Solid-State Drive. 10, 11

SSH Secure Shell. 34, 35, 38

SW Software. iii, vii, xx, 3, 4, 7–9, 13–18, 22, 27, 31, 32, 35, 36, 39, 52, 53, 56, 63, 66, 68

UART Universal Asynchronous Receiver/Transmitter. 65, 66

UBI Unsorted Block Image. 11, 12, 21, 22

UHF Ultra High Frequency. 17, 37

VCS Version Control System. 27, 36

WS Workstation. 35, 38

Terms

cross-compilation Compilation to different architectures than the system currently compiling the software. 13

firmware A small program that enables a device to run and control its HW and communicate with other devices. xvi, 4, 10

flatsat Device connecting the different subsystems in a satellite together on the ground. Instead of stacking the subsystems, they are laid out on a flat device. Used to test, verify, and develop satellite buses on a tabletop. 36, 37

Flight Model The flight-ready model intended for use in space. iii, xvi, 4, 39

fork A forked repository shares the same code as the repository being forked, the ”upstream”. The fork can pull changes from the upstream and create Pull Requests (PRs) to the upstream. 46

front-end An intermediate device between the antenna and the RF interface. 3, 4, 19, 20, 37

git System to perform distributed source version control. 17, 22, 27–29, 32, 34, 44, 45, 61, 67

JTAG Protocol for developing, testing, and verifying electronic circuits after manufacturing. Named after the Joint Test Action Group. 65

kernel The core of an operating system. Handles and manages hardware and software resources. 9, 10

lidsat Collection of the different subsystems in the HYPSO satellites for development purposes. Used to test, verify, and develop satellite buses on a tabletop. 35, 36, 40, 61

NAND Logic gate or flash storage device, enabling non-volatile storage. Alternative to other storage mediums like a Secure Digital (SD) card. In this context, NAND refers to the storage medium. 4, 11, 12, 21, 22, 54

submodule A git repository within a repository that is linked to another git repository. Operates as a subdirectory when placed in a parent repository. The submodule is linked to a specific commit or git tag. 17, 22, 32, 44, 45

TOTEM SDR developed by Alén Space, Spain. Based on the Xilinx Zynq-7020 SoC and the AD9364 Transceiver. iii, viii, 4, 11, 18–22, 31–52, 54–57, 59, 61, 63–68

toolchain A toolchain is a set of tools used to compile, link, and build source code into an executable program. iii, 14, 15, 17, 22, 34, 39, 40, 54–56, 61, 63, 64, 68

U-Boot Das U-Boot (The universal bootloader) is a Free and Open-Source Software bootloader used in embedded devices. This is the first Software that runs on a system and loads the rest of the Operating System. 21, 57, 68

virtual memory An abstraction layer for the physical memory space for a system. Each process has its own virtual memory that simplifies and streamlines memory management. This creates the illusion of a large memory space where the actual hardware is mapped. The virtual memory space is often much large than the physical memory space. 8

Chapter 1

Introduction

Haven't you ever wondered what else is out there? There's wonders in this world beyond our wandering.
... I can feel it.

Elanor "Nori" Brandyfoot

We have wondered what's out there since the dawn of man. Beyond the stars, away from Earth. Recently humans have started looking back at the Earth for answers. Earth Obeservation (EO) from space have proved useful in studying how land, air, and sea develop over time. Using a satellite to acquire a quick overview and autonomous vehicles closer to the ground to acquire the details. This requires a concert of vehicles and a simple way of communication to coordinate observations and their results.

1.1 The HYPSO Mission

The HYPerspectral Smallsat for Ocean observation (HYPSO) missions are multidisciplinary research and educational satellite program organized by Norwegian University of Science and Technology (NTNU) SmallSat Lab (NSSL) [2], mainly funded by various entities at NTNU, Norges forskningsråd (The Research Council of Norway) (NFR), and with partial contributions from the industry, European Space Agency (ESA) and the Norwegian Space Agency (NSA). Its primary mission statement from the HYPSO Mission Requirements Document (MRD) [3] is as stated.

> *"To provide and support ocean color mapping through a Hyperspectral Imager (HSI) payload, autonomously processed data, and on-demand autonomous communications in a concert of robotic agents at the Norwegian coast."*

In January 2022, the first satellite, the HYPSO-1, was launched. A 6U CubeSat comprised all the necessary subsystems and a custom NSSL developed payload. The payload is a Hyperspectral Imager (HSI) with a complimentary Red-Green-

Blue (RGB) camera for geo-referencing and the payload computer for controlling the HSI and RGB camera.

By analyzing the HSI data, two hypotheses are that it should be possible to detect Harmful Algal Blooms (HABs) before they become a threat to fish farms and to monitor the health of the oceans.

As part of the greater Autonomous Marine Operations and Systems (AMOS) project, the HYPSO mission aims at observing the oceans and provide a platform for near real-time oceanographic observation of the coastal areas [4], and autonomously control different autonomous vehicles closer to the sea.

To conduct radiometric and communication research, the next iteration of the satellite, the HYPSO-2 satellite, will add an Software-Defined Radio (SDR) payload to the existing payload. Communication with autonomous vessels in the air or at sea is also enabled by the SDR. The planned launch for the HYPSO-2 satellite is in mid-2024, with an expected lifetime of 5 years. A Concept of Operations (CONOPS) for the HYPSO-2 satellite, showcasing the different stages and modes, is shown in Figure 1.1.

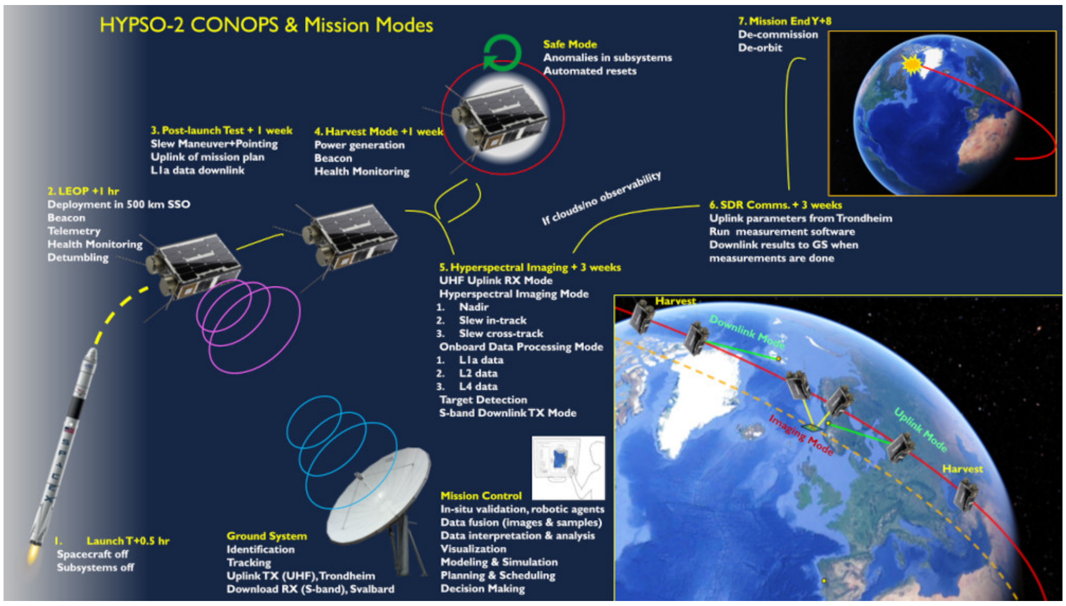
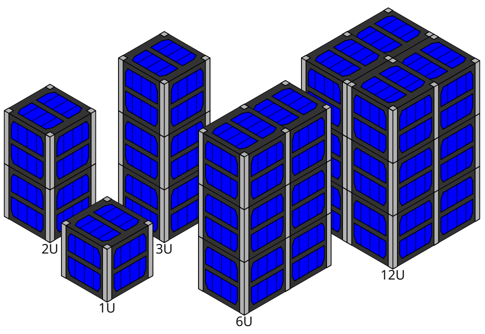


Figure 1.1: Concept of operations of the HYPSO-2 mission. Figure from [5].

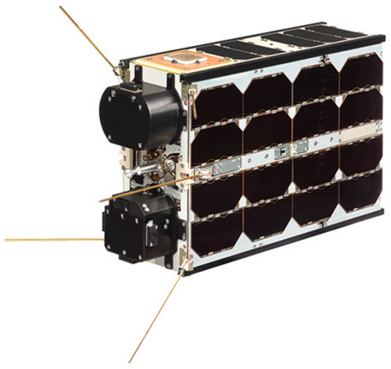
1.2 CubeSats

The original CubeSat concept is a small cube-shaped satellite developed by Cal Poly CubeSat Laboratory (CPCL) [6]. CubeSats are comprised of unit (U) cubes with a side length of 10 cm and a maximum unit weight of 2 kg [7]. The units can be combined into compound sizes ranging from 1U (1x1x1) up to a maximum of 12U (2x2x3) as seen in Figure 1.2a.

The CubeSat developer's extensive use of Commercial Off-The-Shelf (COTS)



(**a**) Different size cube satellites. Range from 1U to 12U.



(**b**) A 6U CubeSat satellite. Picture by Kongsberg NanoAvionics (NA) [8].

Figure 1.2: Different size theoretical cube satellites. A realization of a 6 unit by NA is shown.

components lowers the cost. It optimizes integration, making it highly sought after to conduct educational projects, science experiments, commercial applications, and technology demonstrators. As with the more conventional satellites, the CubeSat includes similar subsystems such as Communications System (COM), Electrical Power System (EPS), computers, and payload. Figure 1.2b shows a complete satellite platform by Kongsberg NanoAvionics (NA). The smaller size and use of COTS in the CubeSat contributes to a significantly shorter development time [9]. To get into space, CubeSats are either deployed by deployers on the International Space Station (ISS), from a Launch Vehicle (LV) as a secondary payload on a conventional satellite, or as part of a ride-share program like the "SpaceX Transporter Program" [10].

1.3 Software-defined Radios (SDR)

Analog radio was first invented in the early 20th century, and since then, the world has gone through a technological revolution. Miniaturization of devices and a significant shift from analog circuitry to digital provide new capabilities and use cases. Utilizing a microprocessor, an embedded system, or even a general-purpose computer to do much of the computing in Software (SW) that normally would be done in Hardware (HW) results in more flexibility.

A Software-Defined Radio (SDR) is a state-of-the-art radio device, replacing most analog circuits like filters, amplifiers, mixers, and other components with SW to extract and modulate information in a signal [11, Introduction, p. xxxiii–xxxvii]. This makes for a highly configurable and flexible radio with rapid reconfiguration capability to adapt different frequencies, modulation schemes, and protocols [12]. Some implementations of an SDR include a Field-Programmable Gate Array (FPGA) to do some of the concurrent parallel processing of the signal. Analog circuitry is still required for the Radio Frequency (RF) front-end to amplify and condition the signal acquired by the antenna. A simplified illustration of an SDR is shown in Figure 1.3.

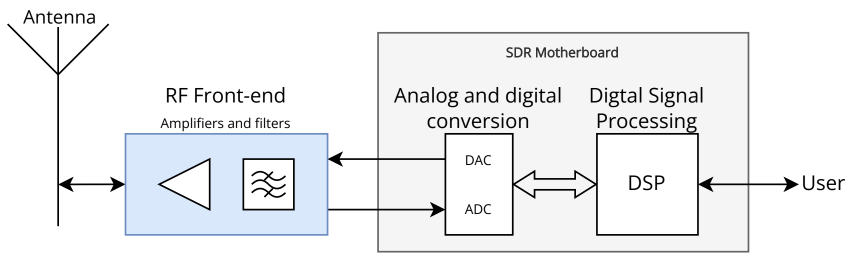


Figure 1.3: A simplified SDR schematic comprising an antenna, RF front-end and the SDR motherboard. Figure from [1].

1.4 Aims and Objectives

Through this thesis, the author will showcase the improvements and further development of the build system and toolchain for the SDR payload included on the HYPSO-2 satellite. The primary focus is on ensuring a robust, maintainable, recreatable system to develop and build Firmware (FW) and SW for the COTS SDR developed by Alén Space, Spain [13] which will be carried by the HYPSO-2 satellite. One essential goal is the ease of use, simplicity, and a gentle learning curve to utilize the build system and tools for SW development, for current and future team members.

A second goal is to ensure robust, resilient, and fault-tolerant operations of the SDR. The thesis will explore possibilities to achieve the above goals while complying with the limitations of the COTS SDR. This includes storing mission-critical SW in multiple redundant locations, fail-safe mechanisms, and dynamic re-configuration of the NAND flash storage and File Systems (FSs).

This work will prove essential for the continued operation of the SDR throughout the lifetime of the HYPSO-2 satellite. Preparations, testing, and development reviews for the Flight Model (FM) TOTEM are conducted before shipping for final integration on the satellite bus developed by Kongsberg NanoAvionics (NA), Lithuania [8].

1.5 Outline

The structure of this thesis is divided into five chapters. This, *Chapter 1: Introduction*, contains a brief overview of the main topics contributed to and the goals and objectives of this thesis. *Chapter 2: Background* briefly explains the topics needed to understand and realize the work done for this thesis. Some of the topics could be moved to *Chapter 3* as they are tools to perform the work, but are more useful to include in *Chapter 2*. *Chapter 3: Methods and Tools* includes the tools and techniques to structure and perform the work. This includes multiple development tools and activities to work on large projects efficiently. Some topics could have been placed in *Chapter 2*, but as they are primarily used as tools, it is most appropriate to keep them in this chapter. The implementation, results, and discussion are combined into one chapter, *Chapter 4: Implementation, Results, and Discussion*, as most of the work done in this thesis is to create a maintainable, repeatable, and robust system and environment to support the SDR throughout its lifetime. Some results and implementation are derived from a discussion on related topics and are therefore combined in one chapter. Conclusion and further work are presented in the last chapter, *Chapter 5: Conclusion*. The bibliography and appendices are placed after *Chapter 5*.

Chapter 2

Background

True education is a kind of never ending story a matter of continual beginnings, of habitual fresh starts, of persistent newness.

J.R.R. Tolkien

In this chapter, the essential concepts and terms needed to understand and realize the work performed in this thesis are described. The selected topics have been carefully studied and are utilized to make the final product.

2.1 Embedded Systems and Operating Systems

Embedded systems are a central part of many payload systems. These are specialized computer systems designed to perform a specific function or set of functions, like an Software-Defined Radio (SDR).

One of the defining features of embedded systems is that they are typically designed to operate without user intervention. They are often pre-programmed with simple yet effective Software (SW) to perform their functions automatically.

Embedded systems vary widely in terms of their complexity and capabilities. This ranges from control systems in airplanes and cars to washing machines and ovens. Still, they all share the characteristic of being tightly integrated with the device or system they are embedded into. They are typically designed to be small, power-efficient, and reliable and may use specialized hardware and software components to achieve these goals [14, Chapter 1.2].

As embedded systems are computers, many use an operating system to manage the system's resources and take them efficiently into use. Embedded Linux is one of the many operating systems to choose from. Many tools and other SW already exist for Linux, which makes Embedded Linux an attractive alternative.

2.1.1 Embedded Linux

Most people have ultimately interacted with an electronic device or system that can be programmed, for instance, a computer or smartphone. These have a lot of different Hardware (HW) systems and often multiple pieces of SW running simultaneously. Managing the HW and SW to work as efficiently as possible is challenging. The Operating System (OS) is the solution to optimizing and managing the system resources effectively.

The concept of a General-Purpose Operating System (GPOS) is dominant whenever flexibility and ease-of-use are essential [15]. These OSs can handle a wide range of HW, include many user applications and SW libraries to aid in both development of other applications and make it easy to operate and use. The penalty for the flexibility is the overhead needed to run the GPOS, both in processing and storage. This overhead might not be ideal for a resource-limited system like an embedded system.

OSs for embedded systems are usually highly tailored. This often includes a specific task or a small number of tasks with a limited power and resource budget [16]. The embedded OSs still have a handful of basic functionality shared with GPOSs [14, p. 2–3]; where some are listed below.

Process Management The OS keeps track of the different executed processes [17]. When different programs are executing, the OS schedules when the different processes can run. An illusion of executing multiple programs simultaneously/concurrently can be obtained by switching between the different programs fast and often.

Memory Management OSs manages the memory in a system. This is a complex task, and most modern OSs simplifies this by implementing virtual memory [18, p. 21-24][19]. The virtual memory lets the process ”believe” that it has more memory than available and homogenizes the system.

File System (FS) The FS enables a system to store data and structures efficiently on a storage device [20]. It keeps track of where and how large the files are. Some file systems will be discussed more in Section 2.1.2.

Device Management A system rarely exists in a ”vacuum”[1]. Most systems have some devices and peripherals. The OS keep track of these devices, such as networking/communication devices, storage, and input/output devices.

The more resource-demanding functions and processes in a GPOS are often omitted in an embedded OS. By removing the Graphics Processing Unit (GPU), Graphical User Interface (GUI), Human Interface Devices (HIDs), and other resource-demanding devices, the system does not need to include the drivers and libraries needed to support these devices.

As the embedded OS usually is tailor-made for a specific task or tasks, the OS

[1]Completely isolated, with only processor and memory. No peripherals, communication capabilities, and no devices.

only need to include necessary drivers and libraries to support the custom HW and SW included in the embedded system. In many cases, real-time capabilities are necessary and built into the kernel, making it a Real-Time Operating System (RTOS). This enables the device to adhere to timing constraints needed in some applications and handle exceptions well. By monitoring the system using built-in and external sensors, the system can react accordingly by utilizing kernel drivers, real-time capabilities, and SW to increase robustness.

Linux is one of many bases for an embedded OS. QNX [21], FreeRTOS [22], and Windows for IoT [23] are among the other widely used embedded OS [24]. While QNX and Windows held around 13.5% market share in 2019, Linux is by far the most popular, with a market share of around 38.5% [24]. There are many reasons for the market dominance, but the key factors are Linux being Free and Open-Source Software (FOSS), lightweight, robust, flexible, and customizable. Due to the open-source nature of Linux, adopting a wide range of HW architectures has dramatically increased the use of Linux in embedded systems [25].

In Figure 2.1, the different layers of the embedded Linux are visualized. A different color separates the layers, which might contain ”sublayers”. The general approach for the boot procedure at system boot is described below.

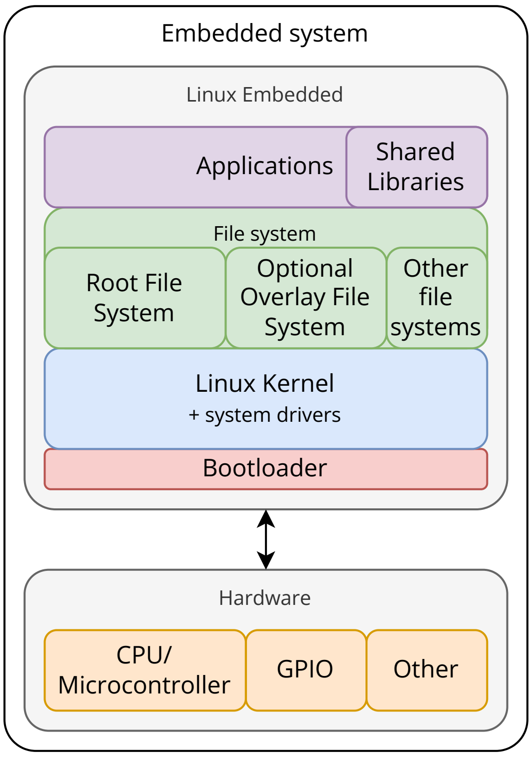


Figure 2.1: Different parts of the Linux Embedded system. Colors are to differentiate between sublayers. Figure from [1].

Bootloader During the start of a system, the First-Stage Bootloader (FSBL) is the first SW that is loaded from Read-Only Memory (ROM) or other non-volatile memory. It will initialize the HW, the Random-Access Memory (RAM), and hand over the execution to the OS or the Second-Stage Bootloader (SSBL) or just "bootloader"[2]. In contrast to the FSBL, which can boot a single OS, the SSBL or bootloader can boot different OSs or different versions of the same OS.

In the case of HW needing a Firmware (FW) or other data from the ROM or other non-volatile memory, for instance, a bitstream to program a Field-Programmable Gate Array (FPGA), the SSBL will perform this task while loading the OS. The FPGA or other devices will then be loaded and programmed during the boot of the OS.

Linux Kernel The Linux kernel handles the HW after the bootloader has handed the execution over to the kernel. The kernel has drivers and support libraries to operate a system's devices and peripherals. Only the necessary drivers and libraries are included to obtain the most efficient use of storage and resources. Non-supported device drivers must be added to the kernel during the build or afterward.

Accompanying the kernel is the Device Tree Blob (DTB). The DTB describes the structure and architecture of the system, including memory locations for ROM, RAM, storage devices, and other devices like an FPGA. For instance, a DTB describes the structure in Figure 2.9a.

To mount and utilize other storage devices with different types of FSs is also the kernels job.

File Systems (FSs) The root FS contains all the necessary files, libraries, and executables to create a functional user space. The user space is where applications run and execute. In embedded Linux, the root FS is usually compressed to a small file with the squashFS FS. This FS is Read-Only (RO) and expands and loads the entire image into RAM. Right before or after the user space has initialized, the kernel might mount multiple other FSs, where OverlayFS is one of them. This is discussed more in Section 2.1.2.

2.1.2 Storage Devices and File Systems

Storage devices are used in computing to store digital data and information. Hard Disk Drives (HDDs), Solid-State Drives (SSDs), flash memory (like USB flash drive), and optical storage mediums like Compact Discs (CDs) and Digital Video Discs (DVDs) are just a few examples of storage devices. The different storage devices have the same goal of storing data, but every device has advantages and disadvantages. Some (like DVDs and HDDs) are suitable for long-term storage but are slow. SSDs and flash memory are faster but have a shorter lifetime and are more error-prone [26].

However, the way that data is structured and kept on a storage device is through file systems. The file system controls how directories and files are stored on

[2]In modern systems the SSBL is commonly referred to as the bootloader.

the storage device, how they are named, and how software programs and the operating system may access and work with them [27, Chapter 5.10]. There are differences amongst file systems regarding the largest file size they can support, how they manage file permissions, and how they handle errors or corruption. Some file systems are even tailor-made for a particular type of storage device like Unsorted Block Image (UBI) file-system (explained in Section 2.1.2 for NAND flash storage [28]).

A suitable file system must be selected for optimal usage, data storage, access, and retrieval. Fast access to commonly used data, reliable backups, and effective use of available storage space can all be provided by a well-designed storage system.

The following sections will describe NAND flash, UBI/Memory Technology Device (MTD), and ”Overlay File System” in more detail as the TOTEM system (more detail in Section 2.4) leverage heavily on these principles.

The two next sections Unsorted Block Images and Overlay Filesystems are from the author’s specialization project [1] written prior to this thesis. The sections include some slight modifications to suit this thesis’ scope.

Unsorted Block Images

This section is from the author’s specialization project [1], with some slight modifications to suit this thesis’ scope.

In contrast to office computers, which use HDDs and SSDs as mass storage, most embedded systems use various forms of simple flash memory. Through an MTD [29] abstraction layer, UBI [30] is a management system to handle raw NAND flash storage [31]. The UBI/MTD system has logical and physical devices. A physical device is a working device that stores data in the UBI/MTD instance as opposed to a logical device that the kernel sees and manages. The number of times the block has been erased is stored in a tiny portion (a header) that separates erase blocks from other types of storage. Because of this, Logical Erase Blocks (LEBs) and Physical Erase Blocks (PEBs) are used to construct the UBI/MTD system.

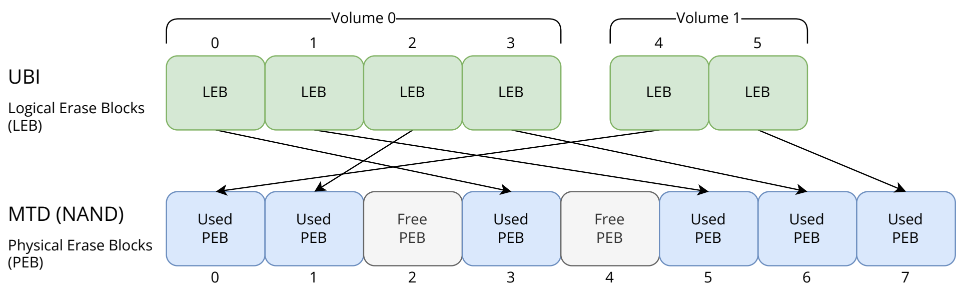


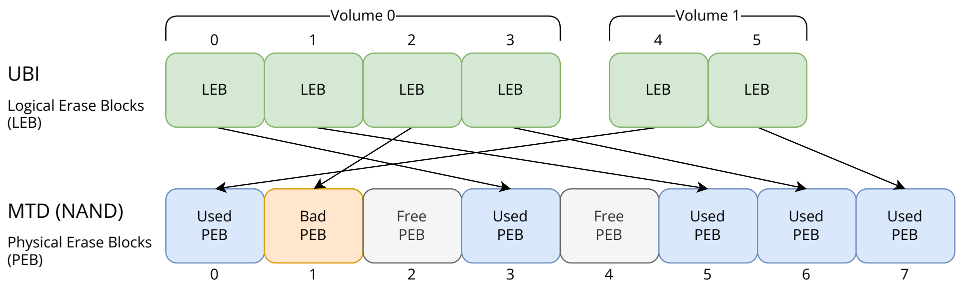
Figure 2.2: Normal operation of a UBI device with two volumes. Note the mapping between the LEB and PEB is unsorted. Figure from [1].

As shown in Figure 2.2, each of the LEBs are dynamically [3] mapped to physical erase blocks, hence the ”Unsorted” part of UBI. A volume comprises a single or multiple LEBs; this is what the OS works with. As NAND flash only has a limited lifetime[4] [32, 33], this system of dynamically assigning LEBs to PEBs can help mitigate data loss and prolong the life of the device. The UBI system accomplishes this through three main processes [30]:

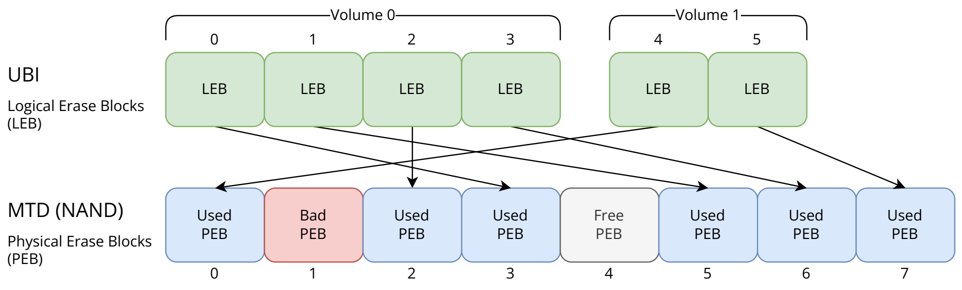
Wear-leveling Move the data in the PEB around to ensure the erased amounts per block remain homogeneous.

Bad block handling The UBI system can detect bad blocks and seamlessly move the data from a bad PEB to a good one. This is demonstrated in Figure 2.3.

Scrubbing The UBI system can detect bit-flips and seamlessly move the data from another PEB to prevent data loss. The old PEB can then be reused.



(a) A single PEB goes bad. PEB1 in this case.



(b) A PEB has gone bad, and the UBI system retargets the LEB to use another PEB. In this case, retarget LEB2 to use PEB2 instead of PEB1.

Figure 2.3: How the UBI operates and recovers from bad blocks. Figures from [1].

Overlay Filesystems

This section is from the author's specialization project [1], with some slight modifications to suit this thesis' scope.

Overlay FS is a FS that merges at minimum two FSs into one final FS [34]. Figure 2.4 shows how the overlay FS takes at least one ”lower” FS (Figure 2.4a) and an

[3]The mapping is not necessarily in order and can change during operation.
[4]A certain number of times it can perform a write-erase cycle to a block.

optional "upper" FS (Figure 2.4b) and merges them into the final FS (Figure 2.4c). The FS term can be somewhat misleading, and the wording "directory tree" might be more correct as the directory trees are the systems being merged in Overlay FS.

Lower The lower FS/directory tree can consist of an arbitrary number of FSs and will result in a single read-only FS, even though the FSs were writable. The order of FS is essential, as the later FSs takes precedence. This creates the basis of the overlay FS.

Upper The upper FS/directory tree can be any FS, both RO and Read-Write (RW). However, only the upper FS/directory tree can be writable. This is implemented this way to prevent conflicts on where to write.

The upper system takes precedence over the lower system when merge conflicts[5] occur. That means that if a file exists in both the lower and the upper systems, the file in the upper system is used. Equal directory trees that exist in both upper and lower are merged.

If a file only exists in the lower FS/directory tree is "removed", the "removal" is stored in the upper FS/directory tree. By not overlaying the upper FS/directory tree, the "removed" file is "restored"[6].

The essential feature of overlay FS is to create a split FS where the lower FS is the core and the upper FS contains the "user-space". This creates a sort of robustness within the FS. A good example is that lower FS can be the RO root FS containing the core operating system. The upper FS contains the user files and the data that can be writable and persistent. The lower FS system will never be written to, and since almost all storage errors happen when writing information, the system will be more robust. The upper FS may become corrupt but will not affect the lower FS system.

2.2 Build System for Embedded Systems

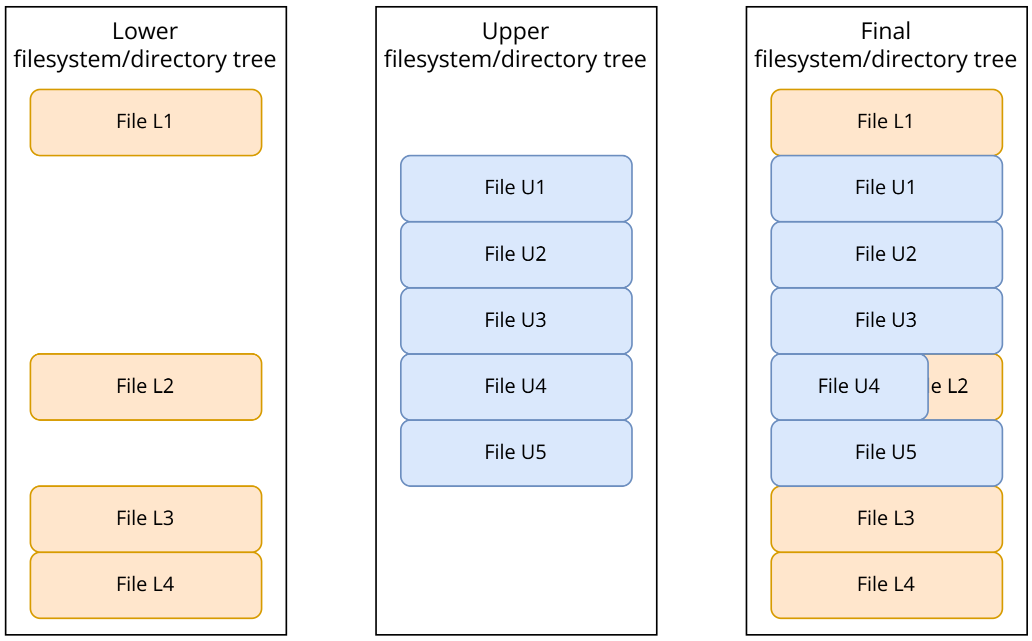
Embedded systems often require a highly tailored OS to suit different HW configurations and many specific packages and SW. There can be several thousand configuration entries[7], especially with Linux, as every part can be customized. This is a good thing; Linux has many great features that are ultimately unnecessary in an embedded system, which then can be disabled with configuration. The process of organizing, compiling (and cross-compilation), and customizing while ensuring maintainability and ease of use is challenging.

A *build system* is a collection of tools and scripts to handle the vast number of configuration entries and customizations. The two most popular build systems for embedded Linux are Buildroot (BR) [35] and Yocto [36]. Both provide a nearly

[5]If the same file/directory exists in both upper and lower FS.

[6]The file was never removed from the RO root FS, there was an instruction that told the OS that the file was removed.

[7]Checked by visual inspection of the generated configuration file with the current build system, with over 4000 configuration lines.



(a) Lower FS/directory tree. There can be multiple lower parts. Becomes read-only.

(b) Upper FS/directory tree. Only one upper part, but can also be omitted. Normally writable.

(c) Final FS/directory tree is the combination of the lower and upper FS/directory trees. Overlapping systems will be merged, and files from the upper will be used.

Figure 2.4: The three parts of an overlay FS. The lower and upper systems are combined/merged to create the final FS. The upper system takes precedence over the lower when there are conflicts. Figures from [1].

ready-to-run solution for most embedded systems. A few adaptations to suit the specific HW, like the memory space and the DTB, are necessary for correct operation. Further customization to the Linux kernel, libraries, and SW follows a strict but flexible configuration system.

BR will be the focus of this section. The name is based on the original task for BR, creating the root FSs, which includes the necessary systems for a complete Linux OS. While it primarily creates the root FSs, it requires a full toolchain for the compilation of the SW for the target[8] embedded system, which later can be used to compile SW for the target. Creating custom packages, changing how the bootloader works, configuring the Linux kernel, how the different FSs are structured, and much more [37, Chapter 6] can easily be configured in BR.

The general approach utilized by BR is shown in Figure 2.5. A target architecture must be specified to ensure that BR creates the correct output. The process can be broken down into a series of automated steps:

[8]The architecture that is the embedded system, like an SDR.

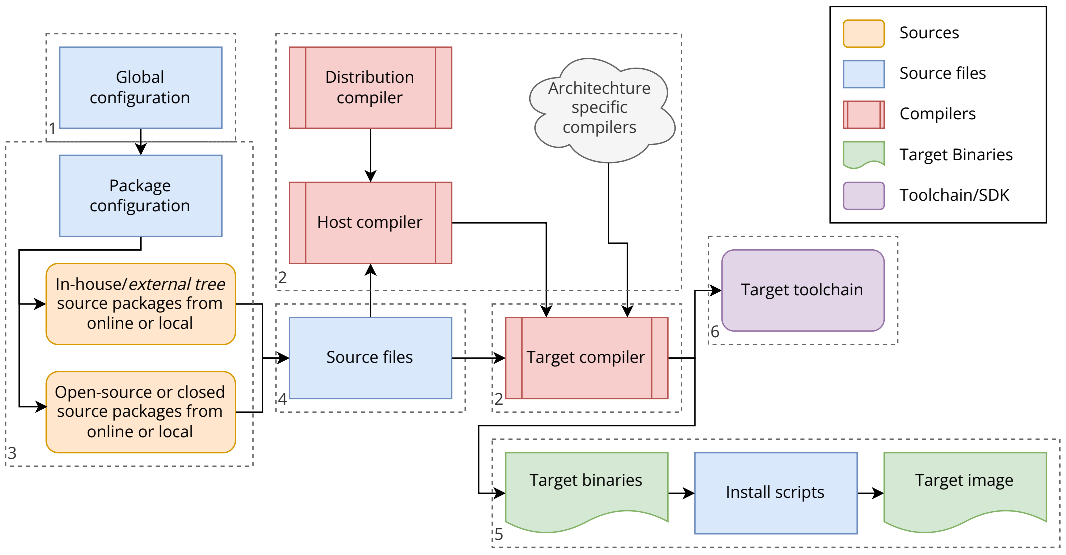


Figure 2.5: General BR build process. The block ”Package configuration” is described further in Section 2.2.1. Figure adapted from [1].

1. Read the main or ”Global” configuration to define the target architecture for the target compiler and what packages are to be compiled and installed.

2. Using the host compiler[9], compile the target compiler based on the configuration defined in ”Step 1”. In many cases, the target compiler can be downloaded from the internet.

3. Based on the packages defined in the main configuration, read the configuration for each package and fetch the source from the internet or any other predefined location.

4. Based on the package configuration, compile the source code with either the host or target compiler. Only SW compiled with the target compiler might run on the target HW.

   This step creates the different FSs based on the main configuration.

5. Using the target compiler and toolchain, combine all the SW and FSs into binaries and complete post-processing with install scripts. The final ”Target Image” is the file ready to be flashed to the target embedded system.

6. The toolchain utilized in the previous steps are made available to be used to compile other SW to be run on target HW.

2.2.1 Packages

At the heart of BR are ”packages”. They can be anything from a full OS like Linux, SW libraries, general SW, and small packages only to include a small file. Each

[9]The compiler that is included in the system compiling SW.

package must contain a configuration on how to fetch the sources and how to install them to the host or target. The configuration can also include dependencies and variables, describe how to build and generate files, define custom downloading processes, and custom install procedures. These are just some of the many capabilities of a package. The general processing for a package is visualized in Figure 2.6, with descriptions listed below.

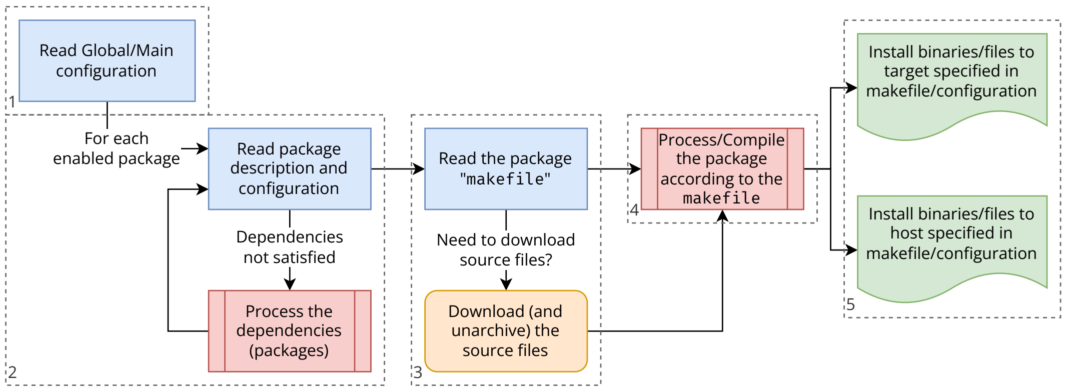

Read Global/Main configuration
1
For each enabled package
Read package description and configuration
Dependencies not satisfied
Process the dependencies (packages)
2
Read the package "makefile"
Need to download source files?
Download (and unarchive) the source files
3
Process/Compile the package according to the makefile
4
Install binaries/files to target specified in makefile/configuration
Install binaries/files to host specified in makefile/configuration
5

Figure 2.6: Packages in BR are treated as subprojects. Figure adapted from [1].

The general steps are also described in the following list:

1. Read the global/main configuration to define the enabled packages.

2. For each enabled package, read their configuration and establish the dependencies.

   If there are dependencies, process the dependency first as packages.

3. Read the package "makefile" to define where the package source files should be fetched from, how to compile or process the package executable, and where or how it should be installed in the system.

   Download the sources if necessary.

4. Compile and build the package based on the configuration established in the last step. A package can be compiled for both the host and the target.

5. Install or include the compiled SW or library based on the package configuration.

2.2.2 External Trees

When creating a custom configuration, BR specifies two ways of storing and maintaining the configuration. The first is to store everything in the root of BR. This has several disadvantages, including the unclear separation between default BR and the customizations and maintainability.

The second recommended way is to create a project-specific *external tree* to store and maintain the configuration [37, Chapter 9.2 & 9.9]. An external tree is a separate directory outside BR to store project-specific configuration, files, and packages. As the configuration of the external tree is applied after the default BR configuration, it is possible to overwrite any configuration entry. Maintainability and distributed development are a lot simpler with the external tree, as the entire BR directory can be included as a git submodule (see Section 3.1.1). This enables the project repository only to contain the project-related configuration.

2.2.3 Cross-compilation and Toolchains

Cross-compilation is the art of using a system with a specific architecture to compile SW for a different processor architecture. This is essential for embedded systems, as a complete toolchain is extensive and resource demanding, therefore usually omitted in an embedded OS.

BR creates an extensive (and custom) toolchain to develop and compile software capable of running on the target device [37, Chapter 6.1]. This toolchain is extensively used by BR to compile packages and other software for the target device.

2.3 The HYPSO Satellite(s)

The HYPerspectral Smallsat for Ocean observation (HYPSO)-2 satellite is a small 6U CubeSat based on the M6P Platform by Kongsberg NanoAvionics (NA) [38]. The satellite bus is developed by Kongsberg NanoAvionics (NA), and the two payloads are developed by Norwegian University of Science and Technology (NTNU) SmallSat Lab (NSSL). A simplified block diagram of the HYPSO-2 satellite bus architecture can be seen in Figure 2.7.

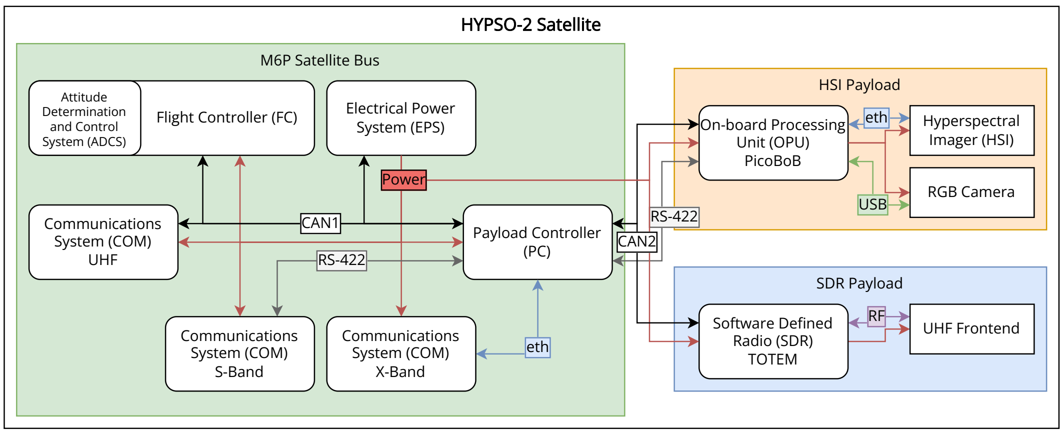

HYPSO-2 Satellite
M6P Satellite Bus
Attitude Determination and Control System (ADCS)
Flight Controller (FC)
Electrical Power System (EPS)
Power
Communications System (COM) UHF
CAN1
RS-422
Payload Controller (PC)
CAN2
Communications System (COM) S-Band
Communications System (COM) X-Band
eth
HSI Payload
On-board Processing Unit (OPU) PicoBoB
eth
Hyperspectral Imager (HSI)
USB
RGB Camera
RS-422
SDR Payload
Software Defined Radio (SDR) TOTEM
RF
UHF Frontend

Figure 2.7: Satellite architecture of the HYPSO-2 satellite. M6P is the satellite bus/platform from Kongsberg NanoAvionics. HYPSO-1 is equivalent, but without the SDR.

The satellite is comprised of four main parts.

- The M6P Satellite Bus from NA, including the Flight Computer (FC), Attitude Determination and Control System (ADCS) (as part of the FC), Communications System (COM) with corresponding radios for Ultra High Frequency

(UHF), S-band and X-band, and the Electrical Power System (EPS). The bus protocol is CubeSat Protocol (CSP) over Controller Area Network (CAN) with two networks, CAN1 and CAN2.

- The M6P Satellite Platform from NA supplies the Structural frame.
- The Hyperspectral Imager (HSI) developed by NSSL.
- The TOTEM SDR developed by Alén Space with SW further developed by NSSL.

These parts comprise the HYPSO-2 satellite. Between the different subsystems provided by NA, the CAN1 network is used for communications. The HSI and the SDR communicate with the Payload Controller (PC) on a separate network, the CAN2 network.

2.3.1 Hyperspectral Imager (HSI)

The Hyperspectral Imager (HSI) is the defining feature for the HYPSO mission. The HSI is an in-house NSSL developed hyperspectral imager [5, Chapter III]. A hyperspectral imager works like a normal Red-Green-Blue (RGB) camera but includes hundreds of spectral bands[10] instead of only three. This enables the imager to record a lot more data about an area. Studying the spectral band for each pixel in the image makes detecting many different types of environments possible. An urban environment has a different signature than a desert, the ocean, or, most interestingly, algae in oceans for the HYPSO mission.

2.3.2 Software-Defined Radio (SDR)

Including an SDR on the HYPSO-2 satellite will enable the satellite to research communications capabilities in the Arctic. As part of the Autonomous Marine Operations and Systems (AMOS) project by NTNU and SINTEF, the satellite will also be a technology demonstrator for communications between sensor nodes and robots in the ocean and land [39]. Even with the ever-growing number of communications satellites from SpaceX and other Internet of Things (IoT) providers, there is still a need to fill the gap for long-range-long-endurance sensor nodes and robots in the Arctic [39]. The SDR will be used by the NSSL to gather knowledge and perform research.

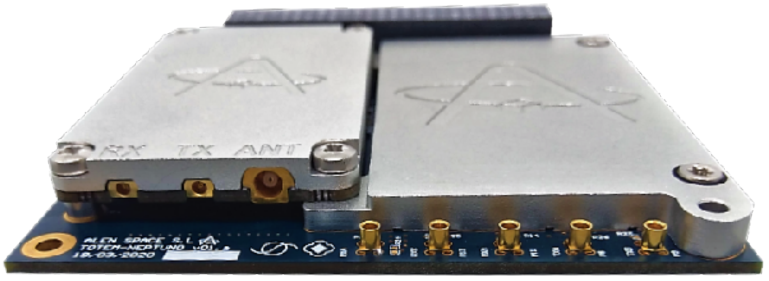
The SDR subsystem is further described in Section 2.4.

2.4 The TOTEM SDR System

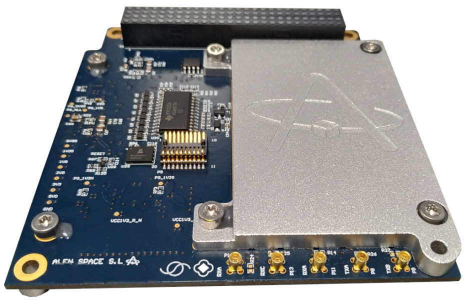
The TOTEM SDR module, Figure 2.8a, developed by Alén Space, Spain [13], is an all-in-one, space-proven SDR module (excluding the antenna). It comprises two Printed Circuit Boards (PCBs). The first, Figure 2.8b, is the motherboard, comprises an System-on-Chip (SoC) for running the Linux-based OS and an Radio Frequency

[10]Bands of the electromagnetic spectrum.

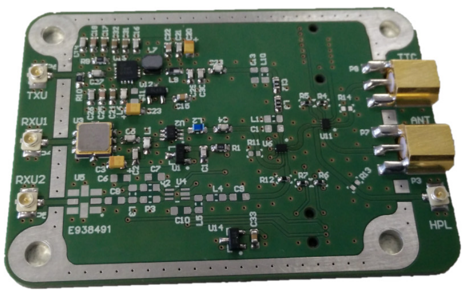
(RF) transceiver. The second PCB, Figure 2.8c, contains the RF front-end. The motherboard, shown in Figure 2.8b, is hereafter referred to as the TOTEM.



(**a**) TOTEM SDR module. Picture by Alén Space [40, p. 1].



(**b**) TOTEM SDR motherboard. Picture by Alén Space [41, p. 1].



(**c**) TOTEM SDR front-end. Picture by Alén Space [42, p. 1].

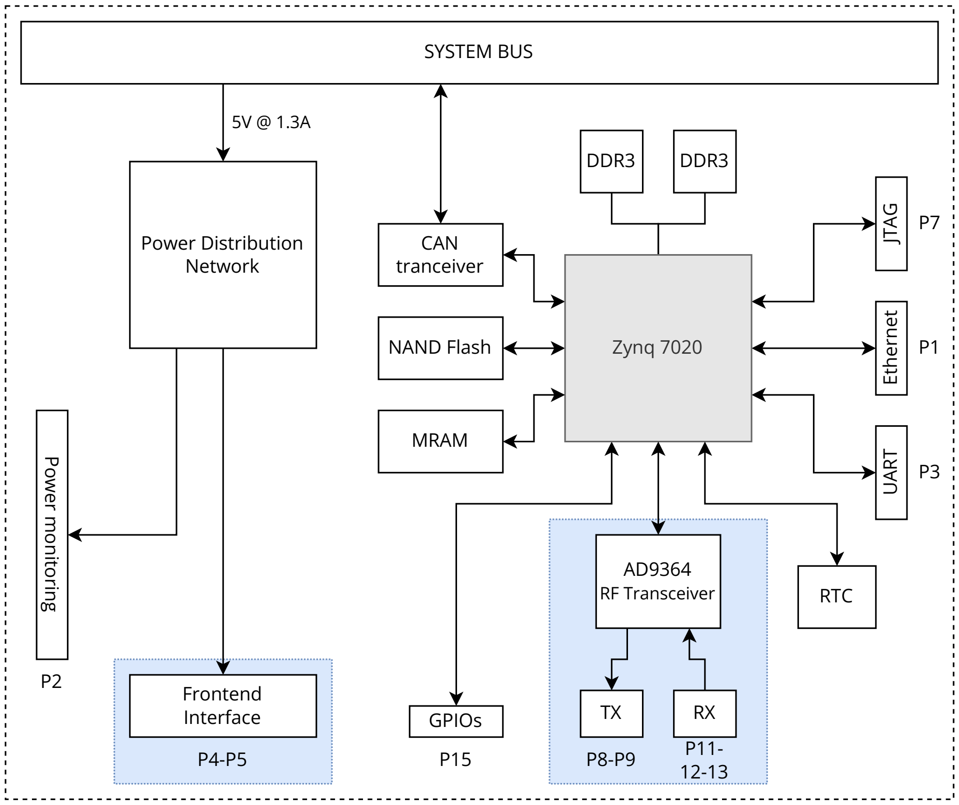
Figure 2.8: TOTEM SDR module.

The TOTEM, Figure 2.9a, is based on a Xilinx Zynq 7020 SoC, which contains a dual-core ARM Cortex-A9 processor and an FPGA [43]. The TOTEM also make use of the AD9364 Transceiver from Analog Devices [44] to generate and receive weak RF signals.

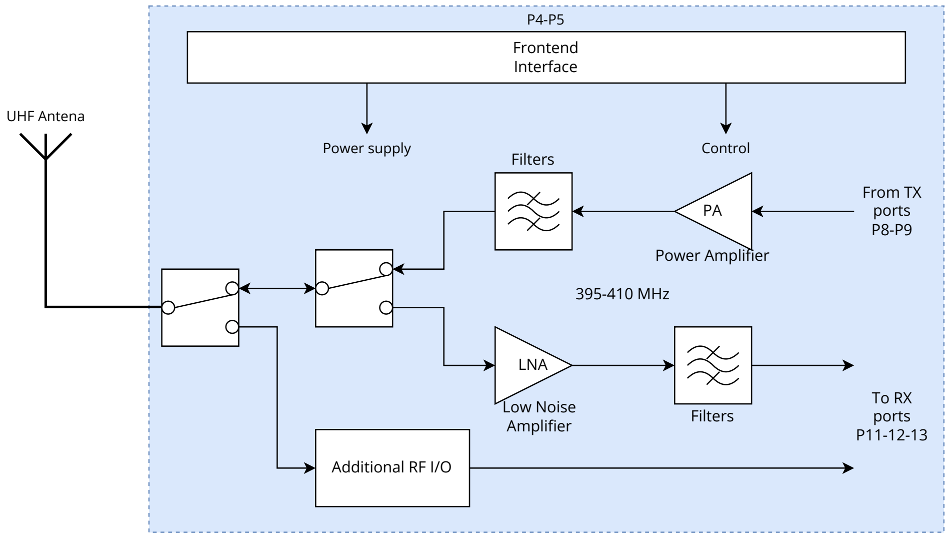
Between the RF antenna and the RF transceiver is the RF front-end. The front-end conditions, filters, and amplifies the signal in both directions, for both transmitting and receiving RF signals. As shown in Figure 2.9b, a number of switches are included in the front-end to control the signal flow between the RF transceiver and the RF antenna. The AD9364 transceiver has a frequency range of 70 MHz to 6.0 GHz, and comprises multiple Digital-to-Analog Converters (DACs), Analog-to-Digital Converter (ADC) for transmission, and data-ports for control with the SoC. To comply with regulations and the permitted frequency range for the HYPSO satellites, the front-end limits the frequency to between 395 MHz and 410 MHz [40].

2.4.1 Embedded Linux

The TOTEM uses a custom embedded OS managed by BR and developed by Alén Space. It is based on the Linux version developed by Analog Devices [45] to increase compatibility with devices like the AD9364 transceiver. The OS is further customized by the NSSL to suit the HYPSO operations. BR configuration for the TOTEM OS is located in a BR external tree in the `sdr-system` repository, as described



(**a**) TOTEM SoC/motherboard architecture. From [1], simplified version of [41, Section 3.1, p. 6].



(**b**) TOTEM front-end architecture. From [1], simplified version of [42, Figure 1, p. 4].

Figure 2.9: TOTEM system architecture. Note the connection points between the SoC (marked with blue in Figure 2.9a) and the front-end. Figures from [1].

in Section 2.4.3. This ensures a highly tailored and optimized Linux distribution for the TOTEM.

The system includes several packages to improve the ease of development and compatibility with other systems. Some of the notable packages included listed below.

- CSP for communication with other systems usually used in a CubeSat [46].
- SDR support library SoapySDR [47].
- GNURadio [48] support libraries for developing SDR applications.
- General packages for interfacing with the FPGA and the RF transceiver.

The enhancements and customization developed by the NSSL and the author in the specialization project [1] includes two packages aimed at the usability of the TOTEM SDR. The two packages contain the two following contributions.

- Environment variables located in `/etc/profile.d/hypso-sdr.sh`. These variables ensure that the developed SDR applications store files at the same location and are a basis for the startup script.
- A startup script to launch the CSP daemon to enable communication between the TOTEM and the other satellite subsystems. The startup script also enables the SDR operations for the HYPSO mission.

2.4.2 Storage Layout

This subsection is from the author's specialization project [1], as the storage layout have not changed. Some slight modifications are performed to suit the scope.

The TOTEM uses a raw NAND flash module as a MTD to manage the 1 GiB flash storage. UBI, as described in Section 2.1.2, together with MTD, handles the storage of data on the TOTEM to protect against data loss and storage failure. The MTD subsystem is split into two parts. One containing the bootloaders (both FSBL and U-Boot) with a size of 32 MiB. The second part is 992 MiB in size and contains the UBI device, containing the rest of the system files.

As seen in Figure 2.10, 5 UBI volumes exist. The `root0` and `data0` volumes will be combined into an OverlayFS. If `data0` volume goes bad or becomes corrupt, the system should still be possible to boot because the OverlayFS is created after Linux has started.

`bitstream0` The bitstream volume contains the bitstreams to program the FPGA when the system boots.

`kernel0` The binaries for the Linux kernel reside in this volume. This is the last volume the bootloader loads into RAM before handing the operation over to Linux. Linux then loads the root FS and the overlay FS.

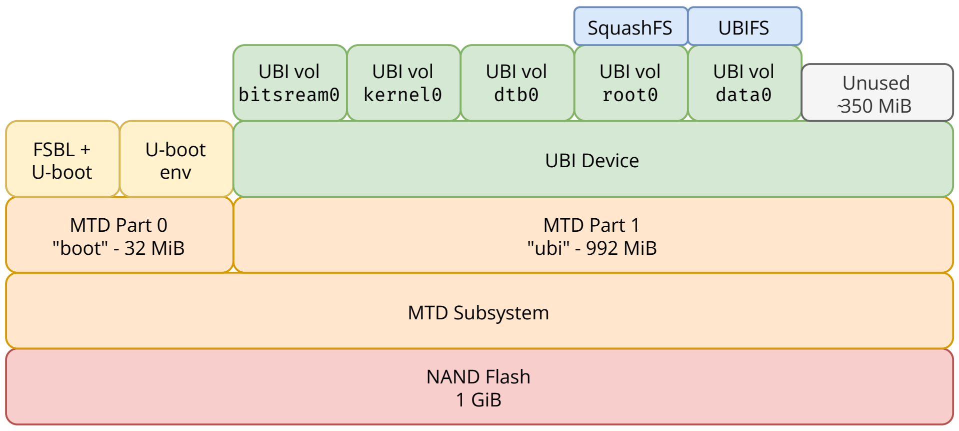


Figure 2.10: Current layout of the TOTEM NAND flash, including the different UBI volumes and bootloader files, based on the TOTEM User Manual [41, Figure 2.3-1]. Figure from [1].

`dtb0` The DTB describes the TOTEM architecture scheme to Linux, what drivers to use, and their options.

`root0` Volume containing the read-only (static) root FS. All critical and static Linux files and configuration for a functioning OS reside in the root FS.

As this is a static volume, the risk of failure due to NAND wear is minimal because most failures happen during the write-erase cycle [33].

After Linux has started, this volume is used as the lower FS in the OverlayFS.

`data0` This volume is the only writable volume in the storage layout.
This is 512 MiB in size and is used as the upper FS in the OverlayFS

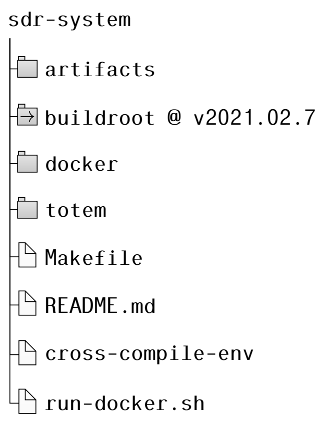
When a file is written, this is where it is stored.

The unused space is essential so the UBI system can utilize its features discussed in Section 2.1.2. UBI uses unused PEBs to ensure proper wear leveling, bad block handling, and scrubbing of PEBs.

2.4.3 The `sdr-system` Development Repository

The `sdr-system` repository contains all the customization, configuration, and development environments to build the customized embedded OS for the TOTEM. As the build system is based on BR, a git submodule with the `v2021.02.7` version of BR is included in `buildroot/`. The `totem/` contains the BR external tree, with the configuration and packages for the TOTEM OS.

`artifacts/` contains a number of smaller archives to remove dependencies from Alén Space. The `docker/` provides the `Dockerfile` to describe the development environment. Several scripts and files are included in the repository root. The most notable is the `cross-compile-env`, which is used to set environment variables to utilize the BR toolchain to compile SW outside BR.



```
sdr-system
├ artifacts
├ buildroot @ v2021.02.7
├ docker
├ totem
├ Makefile
├ README.md
├ cross-compile-env
└ run-docker.sh
```

Figure 2.11: Current structure of the `sdr-syste` repository.

2.5 System Resilience

System resilience has become increasingly important in systems engineering with the expansion of data centers, cloud computing, and the usage of computers for critical operations [49]. It is equally essential for a satellite operating in the harsh and unavailable environment of space. The degree of resilience is not a boolean value but rather a sliding scale of resilience. A system can also be more resilient to certain threats or adverse events and conditions and less to others. [50] defines system resilience as follows.

> ”A system is resilient to the degree to which it rapidly and effectively protects its critical capabilities[11] from disruption caused by adverse events and conditions” [50]

As the quote states, it is how well the system responds to or prevents loss of critical capability. This raises some considerations regarding what type of threats there are, how the system should recover, and what the outcome is if the threats occur and the system cannot be recovered. Some adverse events or conditions will occur and place the system in a degraded state. The system’s resilience is implemented in its protection, comprising four important functions.

Resitance The system can passively prevent or minimize harm from adverse events or conditions.

Detecction The system can actively detect adverse events and conditions, the harm done, and the loss of capabilities.

Reaction The system can actively react and start recovery or stop the occurring or occurred adverse event or condition.

Recovery The system can recover from an adverse event or condition after it has occurred.

[11]A system’s ability to achieve the desired effects [49].

2.6 Satellite Mission Development

As with any large-format project, satellite mission development is subject to standardization. This includes pre-defined mission phases and a limited number of mission phases. There are two main standards used in satellite development, the National Aeronautics and Space Administration (NASA) standard (NASA SP-2016-6105 Rev2) [51] and the European Cooperation for Space Standardization (ECSS) standard (ECSS-E-ST-10C) [52]. They are both similar, sharing many of the same phases and milestone reviews. As the NSSL operates in Europe, the ECSS standard is followed.

The ECSS standard includes 7 mission phases ranging from 0 to F and 13 mission reviews.

Mission Phases and Reviews A timeline of the different phases and their corresponding reviews are visualized in Figure 2.12

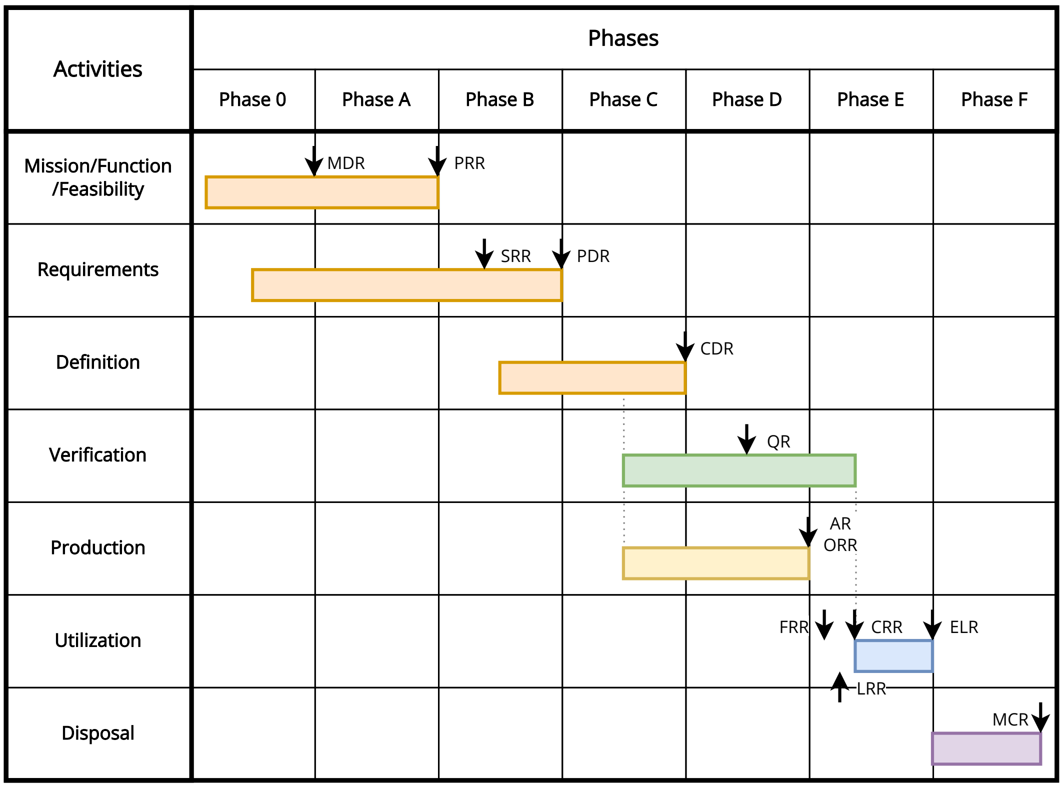


Figure 2.12: Different Phases of the ECSS standard, with the corresponding reviews. Figure adapted and modified from [53].

Phase 0 This phase includes identifying the customer needs and proposing possible system concepts. Implements the Mission Definition Review (MDR) and the MDR actions.

Phase A This phase conducts the feasibility of the proposed concepts and finalizes the customer needs and requirements identified in Phase 0. Supports the

Preliminary Requirements Review (PRR) and the corresponding actions.

Phase B This phase establishes the system's preliminary definition and system solution selected after Phase A. A System Requirements Review (SRR) is conducted to refine the requirements before a Preliminary Design Review (PDR) is held. The approach to development and engineering is defined here.

Phase C The detailed definitions are set, and a Critical Design Review (CDR) is conducted.

Phase D In this phase, the system development is finalized and is verified with a Qualification Review (QR) and Acceptance Review (AR). The system is entering final preparation before utilization.

Phase E This is the phase where the mission is operational. This includes the launch and operation. A Flight Readiness Review (FRR), Operational Readiness Review (ORR), Launch Readiness Review (LRR), Commissioning Result Review (CRR), and End-of-Life Review (ELR) is conducted.

Phase F This phase handles the mission's decommissioning, including the system's disposal. The mission is concluded with a Mission Close-out Review (MCR).

Chapter 3

Methods and Tools

You have my sword. And you have my bow. And my axe.

Aragorn, Legolas, and Gimli

This chapter will explain and showcase the methods and tools used to realize, structure, and aid in the work performed in this thesis. Development tools, development environments, and the laboratory are essential.

3.1 Distributed Version Control using Git

During the development of SW, keeping track of what has been done is essential. Both to see what has been done, or if the SW source needs to be reverted to an earlier point. A Version Control System (VCS) or Source-Control Management (SCM) is a tool that keeps track of development and changes of source code in a systematic and organized way. For smaller projects, this is a ”nice-to-have”[1], but for larger projects, it is crucial. Git is open source and is one of the most widely used VCSs today, with over 90% of developers using git [54]. It features many excellent tools for SW development [55] and is highly tailored towards managing source code.

The following features are essential tools provided by git.

Repositories The core of git is the repository. Each repository contains the history, all the commits, information on branches, and more. But the most important content is the source code itself.

Commits A commit is a ”statement” on a change in a file or files. It comprises the delta[2] of the file or files. By creating a commit, the edited file ”becomes” the current, but as commits are stored as deltas, one can always roll back to an earlier commit or even remove selected commits. This way of storing the changes makes the changes unambiguous and interchangeable.

[1]Not necessary, but very smart.

[2]The difference between the last commit and the new commit.

Branches Branching is a great tool to separate the development into an isolated ”path” or ”tree”. When branching, one can have multiple development trees within one repository. This is useful when multiple developers work on the same repository but with different issues and when compromising the ”main” repository is not an option. Note in Figure 3.1 that the branch is separate from the ”main” working tree and is merged to combine the trees at the end.

Merging Merging is the process of combining two development branches into one. The commits from one branch are combined into the merged branch, and conflicts are resolved.

Submodules Submodules enable the possibility of nesting multiple repositories. This makes it easy to include large projects and implement shared repositories to be used with multiple repositories, like BR. This is explained in more detail in Section 3.1.1.

Distributed When developing in a team, sharing and distributing source code is essential. Git has this at its core. Repositories can be linked with remotes that can be either local or a network (intranet and internet). As each copy of the repository contains the history and branches, each developer in a team can work on the same code from different locations, enabling an efficient way of collaborating on the same project.

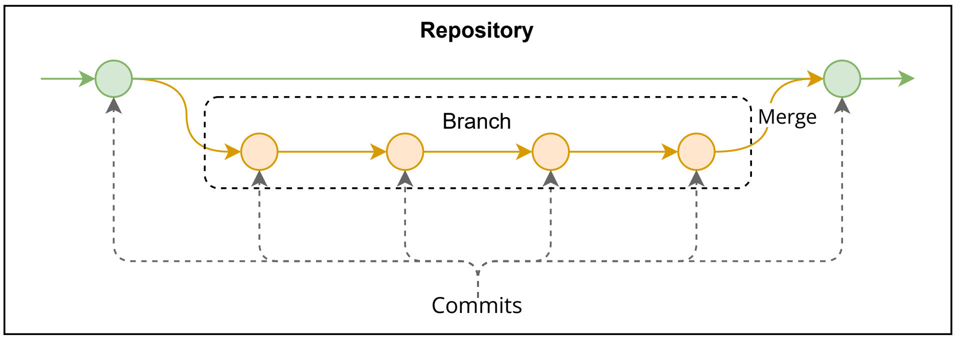


Figure 3.1: Simple view of the git workflow.

3.1.1 Submodules

Submodules are git’s implementation of nested repositories. When adding a submodule, a link to a specific repository version is included [56], effectively locking the ”version” of the submodule, thus increasing the reproducibility. This enables developers to create libraries or a repository with shared files, but as the submodules are repositories, they can be modified. The modifications done in each submodule are stored in the parent repository, which enables customization of the submodule in contrast to prepackaged libraries. The submodules are an effective tool for modularization and maintainability of both small and large projects.

Figure 3.2 shows two different large projects, with ”repository 1” and ”repository 2” respectively. They have different goals but are based on the same libraries and subsystems ”Subsystem 1” to ”Subsystem n”. This project architecture is called a *"Monolith"*, like a monolith. As all the subsystems should be the same across the two projects, keeping them equal is hard in this architecture. Another common problem for monolith projects is that even small projects tend to have a large codebase.

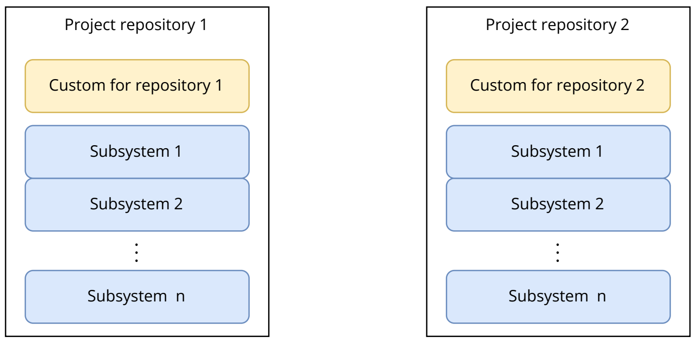


Figure 3.2: Monolith repositories, with overlapping code in the subsystems. Figure from [1].

However, as shown in Figure 3.3, moving the subsystems into submodules allows sharing of the codebase between the two more extensive projects. The project repositories contain a reference to the smaller submodules, preferably with a tag to the version of the submodule. Tagging ensures that if the submodule is updated externally, it does not break the larger project. There are now multiple repositories to maintain, but since the repositories are shared between the larger projects, those are a lot more maintainable.

Moving the subsystems to separate repositories and including them as submodules in the projects greatly improves the reusability of the codebase. As each submodule is locked to a specific commit or tag[3], the possibility of breaking the project if the submodule repository updates are minimal. Instead of having two project repositories, there are now 5, as seen in Figure 3.3. But as each is smaller, the reusability and maintainability are much improved.

3.1.2 GitHub

GitHub [57] is one of the largest git hosting services, with over 100 million developers, over 4 million organizations, and over 330 million repositories [58]. It supplies complimentary features to plain git to make development easier, faster, and better. The most notable additions are Issue tracking, Pull Requests (PRs), Project Planning, and Forks [59]

Issues Issues are created to address an issue, feature, enhancement, or start a discussion on a topic. An issue starts with a "post" that describes the issue; the following is a thread of comments and operations linked to that issue. This can be creating a branch to implement or fix the issue or linking it to other issues and PRs. When working in a team, most of the development should be described in an issue for efficient project planning.

[3]Almost like a version, named point in the repository history.

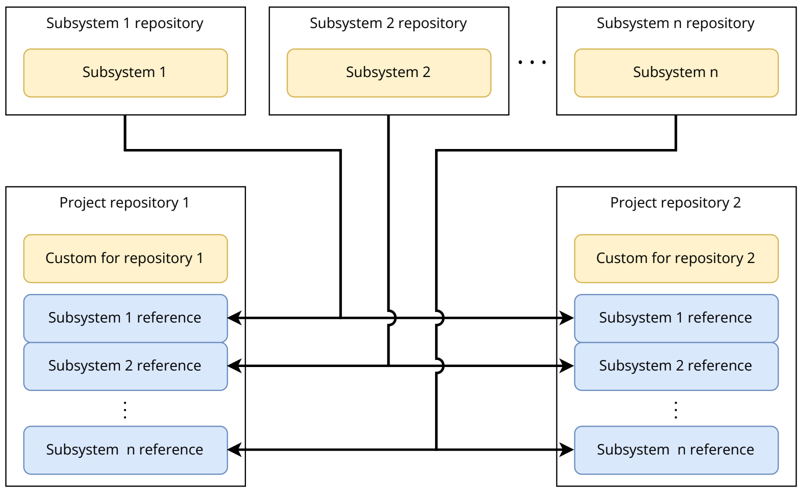


Figure 3.3: Project repositories are sharing a lot of subsystems. Shared code is stored in separate repositories and as submodules in main projects. Figure from [1].

Pull-Requests When a development branch is ready to be merged into the ”main” branch, a PR is created to conduct code review to ensure good quality. This also comprises the same thread as in the issues, with linking and discussions. Any problems with merging will also be revealed with the PR, as GitHub checks if the branch can be merged before doing it. PRs are often linked to specific issues to track the development and automate the project planning.

Project Planning Project planning is at the heart of any development project, small and large. GitHub provides multiple tools for project planning, but the Kanban Board is the choice of many.

Kanban Boards organizes issues and PRs into states (like ”Doing”, ”Done”, ”Backlog”, etc.). These go hand in hand with an agile development methodology, like scrum [60].

Forks *”A fork is a new repository that shares the code and visibility with the original ”upstream” repository”* [61]. Contrary to a separate repository, a fork and the ”upstream” can share code between them by issuing a PR to sync changes. This mimics the functionality of a branch but as two independent repositories.

3.2 Development Cycles in the HYPSO-Project

Structuring and planning development in a large project is essential for high-quality code, meeting deadlines, and sharing the workload. At the NSSL, a modified Scrum [60] methodology is utilized. Each development cycle, called a sprint, is one week and starts with a planning meeting. Issues are generated throughout the sprint, discussed at the meeting, and given an estimate on how much time the implementation should take in days. The meeting ends with planning who’s working on what issue and what should be postponed to the next development sprint. During the week, implementations are completed, and new issues are raised as relevant. At the end of a sprint, the next planning meeting is held. Starting with a small presentation of what's been done since the last time one attended. PRs and finished issues are discussed before the cycle and then repeated.

GitHub is used to manage the issues, PRs, repositories, and project planning. As mentioned in Section 3.1.2, GitHub provides several features directed at project planning. The relevant issues listed in Appendix C, as well as PRs in Appendix D, show how the workflow was applied in the project. These issues and PRs are also discussed in the two following sections.

This is an important part of this master's thesis, where most of the work is presented. Most of the work is qualitative to improve further and enable maintainability, robustness, re-usability, and ease of use. This thesis had an end goal, as discussed in Section 1.4, but how to reach this goal was unknown. The Issues and PRs discussed in Sections 3.2.1 and 3.2.2 describe the author's contributions to the development for HYPSO-2 and the TOTEM SDR. Some of the Issues and PRs below are related to testing and discussing related topics but are not directly part of this thesis. They are shown as part of the contribution made by the author.

3.2.1 Identified Issues

Issues are essential to organize and discuss new features, enhancements, or bugs that arise during development. The list below includes all the significant issues created by the author and issues to which the author contributed significantly in the discussions and/or development and testing. A short description of the author's contribution is included in each issue. Some issues include testing and compiling specific SW on the TOTEM. Most issues are directly linked to a PR, other PRs may relate to multiple issues, and some are merely discussions. The initial presentation of all the issues below is included in Appendix C. A complete view of the individual issues can be found following the link on the issues' names.

`sdr-system#11` **Add `sdr-service` to the built image**
This issue is essential in the improvement of the system's resilience. The contributions are implemented in PR `sdr-system#37` and PR `sdr-system#38` Section 4.2.2 goes into detail on this topic.

`sdr-system#20` **Separate out all HYPSO specific packages and configuration to module**
Separating the HYPSO specific adaptations into a separate external tree improves the understandability and maintainability of the software architecture. This is discussed in Section 4.1.1 and PR `sdr-system#38` includes the contributions related to this issue.

`sdr-system#23` **Change default values**
As the HYPSO specific adaptations and configuration are separated from Alén Space's original configuration, new default values can be set to suit the HYPSO project. These contributions were first introduced in PR `sdr-sys`

tem#25 and later moved with PR sdr-system#38 .

`sdr-system#24` **Create a wiki**
This has been an ongoing activity throughout the thesis work. The author produced significant documentation during the development and implementation. This issue is partly documented in this thesis and partly in the sdr-system repository. The issue will be concluded/closed with the delivery of this thesis. There are no PRs linked to this issue.

`sdr-system#26` **Decide file structure/repo structure to have control over firmware for the FM TOTEM**
Before the delivery of new FW from Alén Space, a discussion on how to structure the two BR build systems for the two TOTEM SDRs was completed. The conclusion was implementing a fork as described and discussed in Section 4.1.3. The contribution is implemented in PR sdr-system-h2#4 .

`sdr-system#28` **Make "chained" startup-script**
This is an essential topic regarding the resilience of the system. A chained startup script enables the system to efficiently handle issues regarding SW critical to the system's operation. This is discussed in detail in Section 4.2.1.

`sdr-system#29` **Separate IP for the FM**
A minor issue regarding the Internet Protocol (IP) address and hostname for the HYPSO-2 TOTEM SDR. Implemented as part of PR sdr-system-h2#4 .

`sdr-system#39` **Move `hypso-shared` to a new repo and include as submodule**
A minor issue where the goal is to move the 🗀 hypso-shared/ BR external tree to a git submodule. The implementation is discussed in Section 4.1.2, and the contribution is included in PR sdr-system#41 .

`sdr-system#40` **Building/inclusion of `sdr-services` fail**
"Bug" related to the inclusion of 🗀 hypso-shared/ BR external tree. Fault analyses by the author revealed that the cause was that the git submodule were not initialized. Updates to documentation were included in PR sdr-system#41 .

`sdr-system-h2#3` **Update to H2 FM Firmware from Alén Space**
Implement the new and updated version of the BR build system developed by Alén Space. The contribution is a major part of PR sdr-system-h2#4 .

`sdr-applications#1` **Add compiling and compile all using the cross compiler from `sdr-system`**
Compiling the SDR applications can be greatly improved by creating a simple-to-use and efficient environment. This is implemented as part of PR sdr-applications#6 and is discussed in Section 4.3.2.

`sdr-applications#5` **Repository structure and example applications**
A good repository structure is essential for efficiently developing SDR applications. By introducing a standardized structure, useability, maintainability, and code quality can be ensured. This structure is introduced in Section 4.3.1

and PR `sdr-applications#6` .

`hypso-sw#754` **Wrong format in m6p and sdr-services?**
With the introduction of the new build system, some integer lengths were changed. This issue describes where the issue is located and some solutions. Contributions made by the author are included in PR `hypso-sw#765` .

`hypso-sw#790` **Testing ethernet throughput on SDR**
There were some issues regarding the Ethernet throughput between the Onboard Processing Unit (OPU) and the HSI. The author contributed to testing Ethernet throughput on the TOTEM SDR. The framework introduced in Issue `sdr-applications#1` and PR `sdr-applications#6` was used to compile the Ethernet test tool. There are no linked PRs to this issue.

3.2.2 Pull-Requests During Development

The PRs listed below describes the author's essential contributions towards the HYPSO project and this thesis. PR `sdr-system#37` , PR `sdr-system#38` , PR `sdr-system#46` , PR `sdr-system-h2#4` , PR `sdr-system-hypso-shared#1` , and PR `sdr-applications#6` are the most significant contributions made by the author. These contain the basis for Sections 4.1 to 4.3 in this thesis.

The presentation of the PRs is included in Appendix D, whilst a complete view of the individual PRs can be viewed when following the link on the PR name below[4].

`sdr-system#25` **Change default values**
This contribution introduces a solution where default values were changed to suit the HYPSO operations.

`sdr-system#37` **Add `sdr-services` to the build image and chained startup script**
Implementing the `sdr-services` executable into the RO FS on the TOTEM SDRs will make them more resistant to adverse events and conditions. This PR was ultimately closed before merging and was included in PR `sdr-system#38` instead.

`sdr-system#38` **Separation of HYPSO specific changes**
Complete separation of the external BR tree `🗀 totem/`, the version-specific HYPSO adaptations in `🗀 hypso-em` and the shared external tree `🗀 hypso-shared`. Inclusion and compilation of `sdr-services` are a part of this PR. This is presented and discussed in Sections 4.1.1, 4.2.1 and 4.2.2.

`sdr-system#41` **Move Shared External Tree To Submodule**
Implementing `🗀 hypso-shared/` as a submodule lays the foundation for creating a maintainable and reusable system for multiple TOTEM SDRs. Essential for an efficient structure discussed in Issue `sdr-system#26` . The contribution is described in Section 4.1.2.

[4]Access to the repositories described in the Preface is required.

`sdr-system#42` **Update `README.md` with ssh based clone**
Small fix to move from `https` based git cloning to Secure Shell (SSH) based.

`sdr-system#44` **Create issue templates**
The ability to easily and quickly create good and descriptive issues is essential for efficient development. This PR implements templates for issues in the `sdr-system` repository.

`sdr-system#45` **Docker hotfix bug**
The base for the Docker environment changed during the performance period, leading to missing dependencies and errors. This PR implemented a fix to lock the base to a version, removing all errors and future problems, making the `sdr-system` Docker more robust.

`sdr-system#46` **Bump hypso shared and submodule update**
Minor PR to update the submodule `hypso-shared/` version to a newer version.

`sdr-system-h2#4` **Update to newest FW version**
This contribution was completed after the completion of Issue `sdr-system#26` and updated the forked repository `sdr-system-h2`. Includes multiple contributions to remove dependencies created by Alén Space, discussed in [1, Chapter 5.1.2]. This is part of the move to multiple repositories, discussed in Section 4.1.3.

`sdr-system-h2#6` **Pull from upstream**
To homologize the systems, update the `sdr-system-h2` with updates from "upstream" `sdr-system`. This includes the updates from PR `sdr-system#41` .

`sdr-system-h2#8` **Upstream docker fix**
To homologize the systems, update the `sdr-system-h2` with updates from "upstream" `sdr-system`. This includes the updates from PR `sdr-system#45` .

`sdr-system-h2#9` **Update hypso-shared to v0.1.0 and fix some Docker dependencies**
To homologize the systems, update the `sdr-system-h2` with updates from "upstream" `sdr-system`. This includes the updates from PR `sdr-system#46` , which is the v0.1.0 version of the shared `hypso-shared/`.

`sdr-system-hypso-shared#1` **Detect stalled services**
Included the detection of stalled execution of `sdr-services` during the start-up to improve further the resilience and recovery of the TOTEM SDRs. This is discussed in Section 4.2.

`sdr-applications#6` **New structure and some example apps**
Implementation of the standardized SDR applications repository and projects. A toolchain agnostic Docker environment is included to make the compilation of SDR applications simple and efficient for multiple TOTEM SDRs. The

contribution is presented and discussed in Section 4.3.

`hypso-sw#765` **Wrong format in m6p and sdr services & bad sdr-services ueye dependency**
Contributed to making `hypso-sw` more compiler agnostic by ensuring it will compile without warnings. This is a collaboration with a PhD student.

3.2.3 Spacecraft Development Reviews

Project reviews, described in Section 2.6, were held during the development of the HYPSO-2 spacecraft to ensure the proper system quality. Before this thesis, the payload Preliminary Design Review (PDR) was completed during the author's specialization project. Some issues arise and are discussed regarding the payloads on the HYPSO-2 satellite, including the HSI, the OPU, and the SDR.

During this thesis, the Critical Design Review (CDR) for the HYPSO-2 satellite was held to ensure the satellite adheres to the specifications set. The payloads were central as the satellite bus by NA has its own CDR. For discussions regarding the SDR, the author was one of the key contributors together with another student and the co-supervisor.

3.3 Development Environment

Development environments are essential for testing and the rapid development of high-quality. This section will describe the important development tools and environments utilized to perform the work in this thesis.

3.3.1 Remote Development With VS Code and SSH

During development, testing SW and OS on the TOTEM SDR is essential. It was, therefore, important to have a proper and flexible development environment within the NSSL. Both for security reasons and the provided flexibility. The lidsat Workstation (WS) is connected to the lidsat and the TOTEM SDR, which makes it flexible and suitable for development on the TOTEM SDR.

A remote Visual Studio Code environment was instantiated, which also had the benefit of being able to connect not only from inside the NSSL. This connection is enabled with the SSH connection between the remote and the connected computer.

3.3.2 Stable Environment with Docker

This subsection is from the author's specialization project [1], with some slight modifications to suit this thesis' scope.

When developing or running software, some critical technical roadblocks make it hard to ensure the desired result. Some of the most important are *dependencies/environment* and *imprecise documentation* [62].

The dependencies can be hard to control as every development system/environment is different. Imprecise documentation can lead to the software not being correctly set up.

Docker [63] is an open-source solution to, among other things, the problems mentioned above. It is built on the concept of *containers*. Each container incorporates a VCS for versioning the Docker container images. A `Dockerfile` describes how the Docker image should be built, including what OS the image should be built from and what packages shall be included. The image is then run as a container within the host OS but with an isolated FS to separate it from the host OS. The host OS can be anything that can run Docker, and the Docker image will be the same for every system because of the `Dockerfile`. This solves the first issue with *dependencies*. The `Dockerfile` also solves the second problem with *imprecise documentation* because it is written in a self-documenting fashion. Maintainability should be more achievable when the environment is equal every time it is used.

Docker comprises the Docker client and the Docker host [63]. The client creates/build the Docker Image, while the Docker host runs the Docker Containers. Only the Docker host is needed if a Docker image is already built. Because of this, Docker can run completed software and ensures the correct environment for developing software. Flexibility and low overhead operations [64] obtained by Docker consequently can ensure reproducibility between different host OSs.

3.3.3 LidSat Communication With `hypso-cli`

The communication and control between the different satellite systems on the HYPSO satellites and the lidsats is done through `hypso-cli`. This is part of the SW suite developed by the NSSL. The communication protocol is CSP [46] over CAN.

The HYPSO OPU and the TOTEM SDR run the counterparts of `hypso-cli`, the `opu-services` and `sdr-services`. These enable the control and operations of the OPU and the TOTEM SDR, both in the lab and in space.

3.4 Lab Setup, Testbed, and Equipment

An assembled satellite is a highly integrated and optimized device, making it challenging to develop, test, and debug. A flatsat comprises the same subsystems as the assembled satellite but spread out on a ”tabletop” to enable rapid development, testing, and debugging. NA is the satellite platform provider for the HYPSO satellites, and also supplies a flatsat of the satellite bus used in HYPSO-1 and -2. It integrates the same subsystems as described in Section 2.3, comprising the FC, the EPS, the PC, the ADCS, and the COM. However, it does not comprise the payloads.

3.4.1 The NSSL FlatSat: The LidSat

The ”LidSat” is NSSL’s flatsat for the HYPSO-1 satellite. This is connected to the FC at NA over the internet with a CAN bridge. The LidSat developed by NSSL, and the development TOTEM SDR was integrated at a later stage by Gara Quintana-Diaz and Roger Birkeland [39, Section 5] for SDR development and testing. Systems, computers, devices, and connections are shown in Figure 3.4. Even though this describes the HYPSO-1 LidSat, this applies to the HYPSO-2 satellite, as these are equivalent. The subsystems included in the LidSat are listed below.

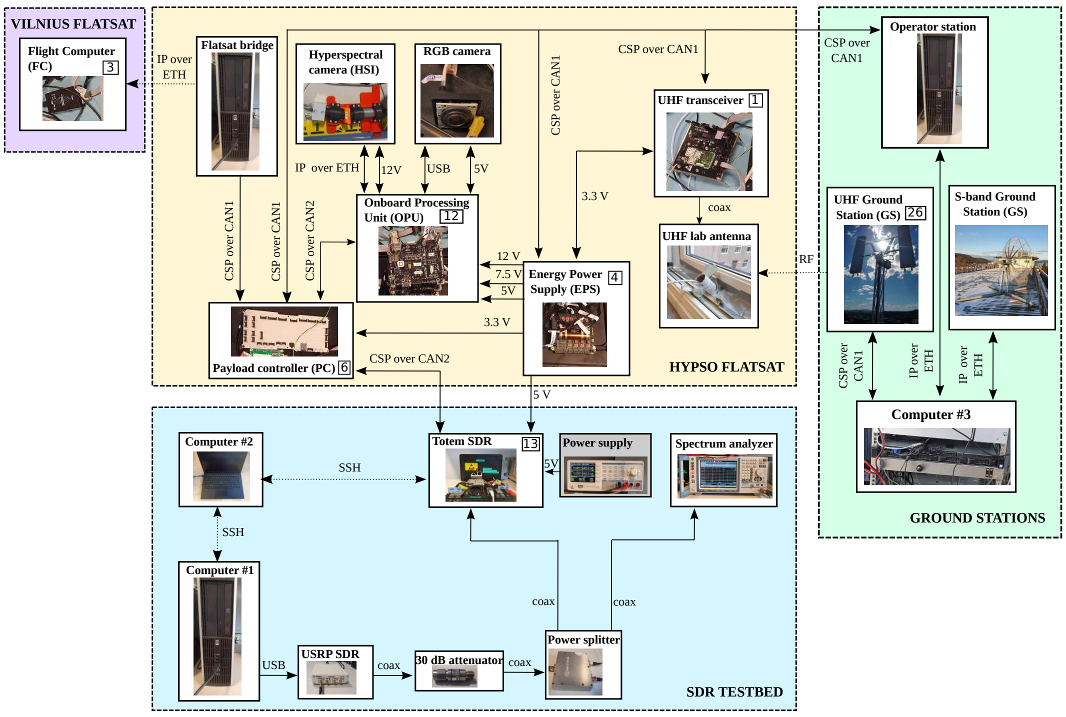


Figure 3.4: Arcitecture of the LidSat testbench and FlatSat system [39, Figure 6] for both the HYPSO-1 and the HYPSO-2 satellites. The numbers in the small squares are the CSP address for the device.

- The HSI camera.
- The RGB camera.
- The OPU to control and store data from the HSI and RGB camera.
- The TOTEM SDR module with a RF front-end and an antenna. A picture of the TOTEM SDR as implemented in the LidSat is shown in Figure 3.5
- The EPS used to provide power. Note that there are two EPSs, one in the LidSat and one in the flatsat at NA. NSSL only utilizes the one in the LidSat.

  This gets its power from an external power supply.
- The PC.
- The UHF transceiver with the corresponding antenna.
- Multiple computers for development and testing.

The satellite uses two CAN busses to communicate between the subsystems with the CSP protocol. The payloads are connected to the CAN1 network, while the rest is on the CAN2 network with multiple operator-, development-, and test computers. Some of the computers are possible to connect using SSH remotely. By using the workflow discussed in Section 3.2 and git described in Section 3.1, rapid system development and testing system ideas on the TOTEM SDR can be achieved.

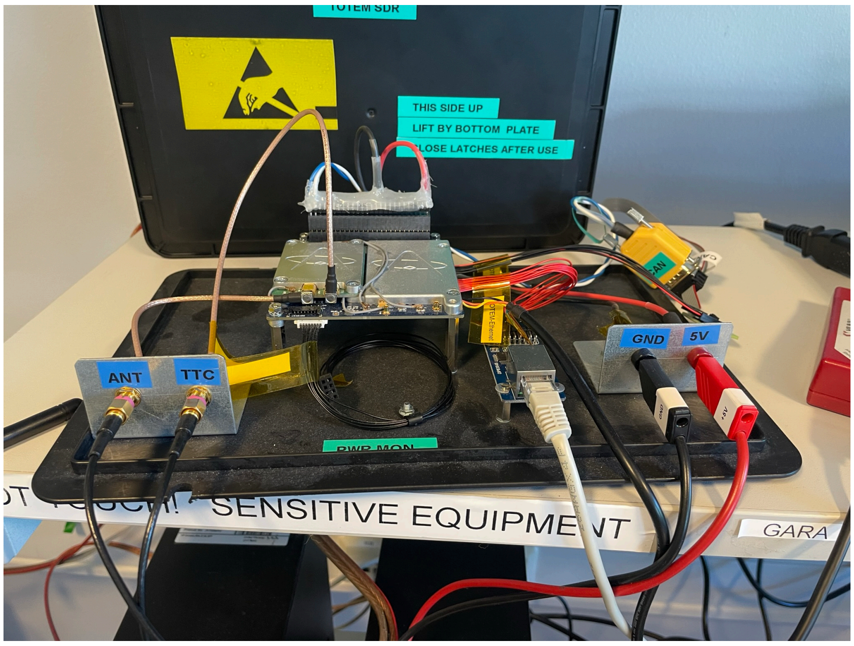


Figure 3.5: The LidSat TOTEM SDR for test and development purposes. This is also referred to as the Engineering Model (EM) version TOTEM SDR.

3.4.2 Testbed for SDR application development

To test and develop SDR applications, an SDR testbed is created. It creates a stable, flexible and predictable environment where different applications can be rapidly tested and developed. The testbed can be remotely accessed while still retaining full functionality. The implementation of the Testbed for the TOTEM SDR is shown in the blue area in Figure 3.4. It comprises the following equipment.

- The TOTEM SDR.
- A Powerful and flexible signal generator can create various signals and modulation schemes.
- Network analyzer capable of analysing frequencies from 395 MHz and up.
- 2-way antenna splitter
- 30 dB attenuator.
- USRP desktop SDR.
- Networked WS running a Docker container with a fully functioning development environment for GNURadio [48].

Chapter 4

Implementation, Results, and Discussion

Even the smallest person can change the course of the future.

Galadriel

This chapter will explain and discuss the implementation, impacts, and results of the work carried out in this thesis. The implementation, results, and discussion are combined as the work performed is a baseline for further development for both the current and future SDRs. The first three sections in this chapter include the procedure and architecture for developing FW for multiple TOTEMs in a sustainable and maintainable manner, system resilience, and an easy compilation procedure of SDR applications. The last section considers the physical development environment of the HYPSO-2 Flight Model (FM) TOTEM SDR. As mentioned in the Preface, these contributions are qualitative and hard to test or measure the impact of directly. Some testing will be discussed in Sections 4.2 and 4.4 to ensure the proper reaction to adverse events and how robust the physical development environment for the HYPSO-2 FM TOTEM SDR is.

4.1 Designing a Flexible and Usable Procedure for Development of Multiple Firmware and Toolcains

Maintaining and developing multiple different FW versions and variants can be complex if not structured properly. By implementing a structure to promote modularization, reusability, and maintainability, the useability of the build systems can greatly increase the simplicity of maintaining complex toolchains for multiple TOTEM SDR systems. As the development of SDR applications require testing and development time, there is a desire only to develop one set of applications that work on all the TOTEM SDRs. This enables the different satellites to perform identical measurements and research with equal SW. This requires the different build systems to be equal for the developers.

Throughout these chapters the notation `sdr-system`, `sdr-system-em`, `sdr-system-h2`, etc. will be used. The suffix denotes the version of the FW referenced. `sdr-system` is the general idea of the build system that is used to build the FW and the toolchain for the different TOTEM SDRs. As the EM version of the TOTEM SDR is the development version, `sdr-system-em` is often referred to as `sdr-system`. The current development version is the `sdr-system-em` (or `sdr-system`), while `sdr-system-h2` denotes the `sdr-system` used for the HYPSO-2. HYPSO-1 does not contain an SDR so it does not have a designator. If a HYPSO-3 is developed containing an SDR, the `sdr-system` will be named `sdr-system-h3`.

4.1.1 Keeping HYPSO Adaptations Outside The Original Build System

Maintainability and reproducibility are essential for achieving a highly homogeneous set of repositories across the different versions. By designing a structure that enables the HYPSO developers to quickly customize the firmware package provided by Alén Space, time spent on developing SDR applications can be maximized. This is achieved by utilizing BR external trees, multiple configuration files, and multiple repositories.

The firmware received from Alén Space is a Minium Viable Product (MVP) that works but does not contain any necessary adaptations to suit the HYPSO satellite operations. During early development (2019-2022) of the HYPSO TOTEM SDR, custom packages were added directly into the structure delivered by Alén Space, and configuration entries were changed. This led to an unorganized structure containing TOTEM configuration and packages and the custom HYPSO code adaptations and packages, as both were located in the same directory with no clear distinction.

Figure 2.11 visualizes the original structure[1] of the build system repository as delivered by Alén Space. The only external BR tree is located in 🗀 `totem/`. By expanding the external tree directory, as shown in Figure 4.1, there is a lack of a clear separation between the packages for the TOTEM and the HYPSO packages, as both types are in the same directory.

In early 2023, the NSSL received the FM version of the TOTEM SDR. This device will be included on the HYPSO-2 satellite. It will complement the EM version included in the lidsat, currently used for development and testing purposes. Development and testing on the EM have been conducted since the start of 2019.

Before receiving the FM TOTEM SDR, only one development environment and build system was necessary to compile and build the FW and toolchain. With two devices, a second build system became necessary as Alén Space changed three libraries and frameworks connected to usage of the Electrically Erasable Programmable Read-Only Memory (EEPROM) and the SDR frontend. The upside was that the build system was still based on BR.

Some configuration entries are changed to support the changed packages and

[1]Before the contributions from the author in this thesis.

```
totem
├── board
│   ├── <FS overlays>
│   ├── <Board configs like DTB>
│   └── <...>
├── config
│   └── <BR configuration files>
├── package
│   ├── <TOTEM packages>
│   ├── <HYPSO packages>
│   └── <...>
├── Config.in
├── external.desc
└── external.mk
```

Figure 4.1: Contents of the `totem/` directory, the most important is the lack of separation between TOTEM and HYPSO packages.

other differences in the TOTEM SDR HW.

For the maintainability and ease of adaptation for multiple versions of the TOTEM SDR, a clear separation of packages from Alén Space and NSSL is highly desirable. Issue `sdr-system#20` and Issue `sdr-system#26` describes this in more detail.

BR allows multiple external trees but limits the configuration files to one as described in [37, Chapter 9]. A script to combine/merge configuration fragment[2] files is supplied by BR at `buildroot/support/kconfig/merge_config.sh`. A helper bash script, `scripts/build_config.sh`, Appendix A.2, that accepts an arbitrary number of configuration files, is developed by the author and included in the `sdr-system` repository. The bash script will combine all configuration files into a single file that BR can read and use. The order of the configurations to be merged is essential. The last configuration entry is used if two or more duplicate configuration entries are defined, often referred to as Latest Takes Precedence (LTP).

As the latest configuration entry is used, the `totem/` external tree can be ”reset” to how it was delivered from Alén Space. NSSL and the author can then include configuration entries to overwrite some configuration entries originally provided by Alén Space, to suit the HYPSO operations. Examples are the IP addresses and passwords for the TOTEMs SDR.

[2]A fragment configuration file only contains a few configuration entries. Not the entire configuration.

Inclusion of multiple BR root FS overlays are done similarly to the configuration. The paths to the different overlays are defined in the `BR2_ROOTFS_OVERLAY` BR entry and are combined and included in the RO FS during the build process. The order of the paths is equally essential in the configuration, as the LTP principle affects the combined files.

An essential contribution by the author was to define the best possible code and repository structure and thus create a second BR external tree named `hypso/` that includes only the adaptations necessary to suit the HYPSO project. This will highlight the separation between the original `totem/` by Alén Space and the contributions made by NSSL and the author in `hypso/`. The adaptations necessary are the following.

- Three packages for correct operations of the HYPSO TOTEMs.
    - `hypso_env`
    - `hypso_init`
    - `sdr-services`, this is discussed in more detail in Section 4.2.

  `hypso_init` and `hypso_env` are described in detail in the author's specialization project [1].

  The start-up script included in `hypso_init` for executing the services needed for communications between the satellite and the TOTEM SDR also handles the fault-tolerance discussed in Section 4.2.

- Network configuration with custom IPs addresses for the TOTEM SDR to suit the network architecture of NSSL.
- Hostname following the naming conventions by NSSL. This is set by the `BR2_TARGET_GENERIC_HOSTNAME` configuration entry.
- Password for the `totem` user, and potentially other users.

  The `BR2_ROOTFS_USERS_TABLES` configuration entry defines a location for the `users_table` file.

- Password for the `root` user.

  Defined in the configuration entry `BR2_TARGET_GENERIC_ROOT_PASSWD`

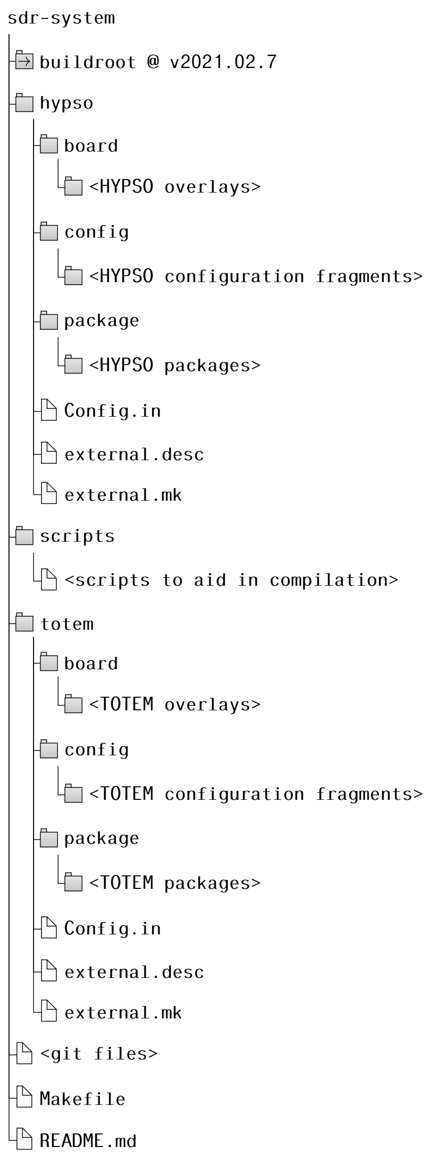
- Configuration to enable the HYPSO packages, custom overlays, and other configurations.
- Overlay FS configuration for BR.

  Configuration entry `BR2_ROOTFS_OVERLAY` contains paths to the different overlays.

- Placeholder files to create folders in the FS on the TOTEM SDR.

The different configuration entries are already defined in the `totem/` external tree, but the author included new values that overwrite the original values when merged with the `scripts/build_config.sh` script.

Sensible and well-structured external trees follow the structure described in the BR Manual [37, Chapter 18]. The resulting structure for the `sdr-system` repository is visualized in Figure 4.2, with the two external trees `hypso/` and `totem/`.



```
sdr-system
├── buildroot @ v2021.02.7
├── hypso
│   ├── board
│   │   └── <HYPSO overlays>
│   ├── config
│   │   └── <HYPSO configuration fragments>
│   ├── package
│   │   └── <HYPSO packages>
│   ├── Config.in
│   ├── external.desc
│   └── external.mk
├── scripts
│   └── <scripts to aid in compilation>
├── totem
│   ├── board
│   │   └── <TOTEM overlays>
│   ├── config
│   │   └── <TOTEM configuration fragments>
│   ├── package
│   │   └── <TOTEM packages>
│   ├── Config.in
│   ├── external.desc
│   └── external.mk
├── <git files>
├── Makefile
└── README.md
```

Figure 4.2: Contents of the `sdr-system` repository after extracting the HYPSO contents to a separate BR external tree.

As all the HYPSO configuration, files, and packages are included through an external BR tree, the `totem/` external tree is ”reset” to the values and configuration as delivered by Alén Space. The described structure will simplify the process of adopting the FM TOTEM SDR to suit HYPSO requirements as only the `totem/` external tree (and maybe the `buildroot/` version) are changed. This allows the

HYPSO adaptations to remain the same between the different versions of TOTEM SDRs. Some minor changes to ensure maintainability are implemented to fetch necessary archives from HYPSO web servers instead of using Alén Space's private git. This is described more in detail in the author's specialization project [1].

4.1.2 Shared Repository for Shared Files

The restructuration of the development repository in Section 4.1.1 aids in adopting the FW delivered by Alén Space to the HYPSO requirements. But there is still a lot to be desired. It is still challenging to differentiate what needs to be changed between each version of the TOTEM SDRs and what is equal or shared between them. One example of a version-specific change is the IP address for the TOTEM SDR. Each one needs a unique IP to work correctly on a IP network. On the other hand, it is desirable to standardize passwords between all the TOTEMs.

BR external trees are again coming to the rescue. By dividing the packages, configurations, and overlays that are shared between the TOTEMs and the version-specific adaptations, the maintainability and understandability of the build system will improve. The single `hypso/` external tree is substituted by the version specific `hypso-em` external tree and the shared `hypso-shared` external tree. The two external trees follow the same structure defined in the BR Manual [37, Chapter 18].

Contents of the `hypso-em/` external tree are listed below.

- Network configuration
- Hostname
- Overlay FS configuration for BR.

Contents of the `hypso-shared/` external tree are listed below. A preliminary structure of the external tree is shown in Figure 4.3

- Placeholder files to create folders in the FS on the TOTEM SDR.
- Password for the `totem` user, and potentially other users.
- Password for the `root` user.
- The three packages for correct operations of the HYPSO TOTEMs.
  - `hypso_env`
  - `hypso_init`
  - `sdr-services`, this is discussed in more detail in Section 4.2.

The restructure enables the `hypso-shared` external tree to be substituted with a git submodule. This has several benefits, but portability and modularization between the different TOTEMs are the most important contributions. Adopting the

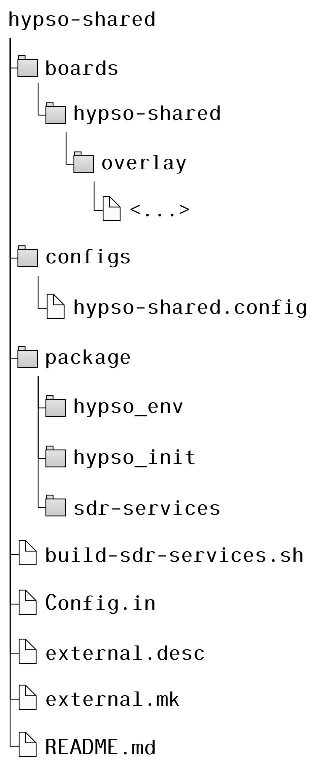


Figure 4.3: Preliminary structure and contents for the `hypso-shared/` BR external tree.

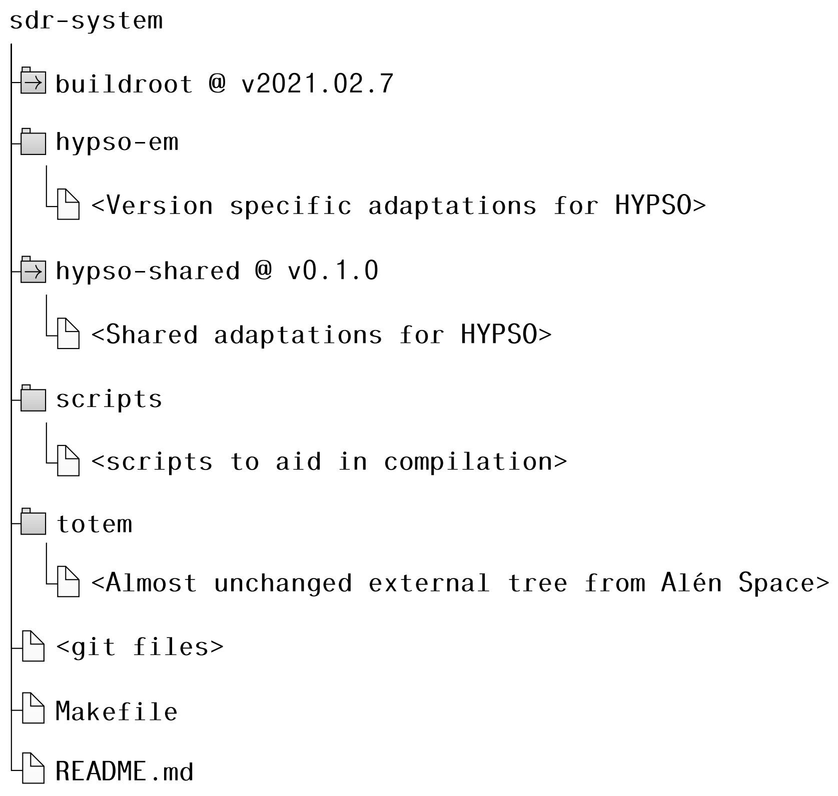
FW to a new TOTEM SDR with a standard external tree for the shared functionality and configuration will ensure a homogeneous system architecture. The inclusion of a git submodule also has the benefit of ”locking” the version of the submodule. This will make each build system repository more robust and keep the build systems working longer using a known working version. The final structure of `sdr-system` is shown in Figure 4.4.

This contribution solves the issue of sharing the standardized configuration across the different TOTEM SDRs. By using submodules, the version of the shared external tree can be locked, increasing the build system's robustness by ensuring a working external tree. Updating the shared external tree is equivalent to updating a normal submodule.

4.1.3 Multiple `sdr-system` Repositories

With a clear separation between what Alén Space has done, what is shared between the TOTEMs, and the version-specific adaptations make it trivial to adopt the new FW for the HYPSO-2 FM TOTEM SDR. Changing the `totem/` external tree (and removing Alén Space's private git dependency) is enough to make sure the FM FW adheres to the NSSL structure. Because of the layering of the external trees, the necessary adaptations for the system are ensured.

This functionality can be implemented in several ways, with pros and cons. The



```
sdr-system
├ buildroot @ v2021.02.7
├ hypso-em
│ └ <Version specific adaptations for HYPSO>
├ hypso-shared @ v0.1.0
│ └ <Shared adaptations for HYPSO>
├ scripts
│ └ <scripts to aid in compilation>
├ totem
│ └ <Almost unchanged external tree from Alén Space>
├ <git files>
├ Makefile
└ README.md
```

Figure 4.4: Structure of `sdr-system` repository after a separation between work done by Alén Space (`totem/`), shared HYPSO adaptations (`hypso-shared/`), and adaptations specific to each version of the TOTEM SDR (`hypso-em/`), in this case, the EM.

first way is to have a different branch for each version of the TOTEM SDR. This approach has advantages such as shared issue tracking and synchronizing changes between the branches. The disadvantage is that even though the different build systems are similar, a complete rebuild of the BR build system is required. A change in core libraries is one of the issues that typically will require a complete rebuild every time a developer changes from one branch to another. The rebuilding process is time-consuming, using around 40 minutes to complete.

Finally, a simpler and better solution is to take advantage of the GitHub functionality, and this method was chosen. Forking the EM repository (the original `sdr-system` repository) enables a clear separation between the different TOTEM SDRs. Much of the same functionality that exists for branches also exist for forks. Pulling changes from the "upstream" repository and creating pull requests to the "upstream" makes it possible to synchronize changes between forks.

The structure between the different versions will be the same, as they are forked from the "upstream" repository. To adopt a new build system for a new TOTEM SDR, only the following steps are required.

1. Create a fork of the main repository (the EM repository).

2. Change the contents in the `totem/` external tree for the new build system supplied by Alén Space. Make sure there are no dependencies from internal servers.

3. If the build system requires a newer version of BR, update BR.

4. Change some variables in the version-specific external tree and rename it. For HYPSO-2, the external tree will be named `hypso-h2` instead of `hypso-em`.

The repository architecture and submodule linking are illustrated in Figure 4.5.

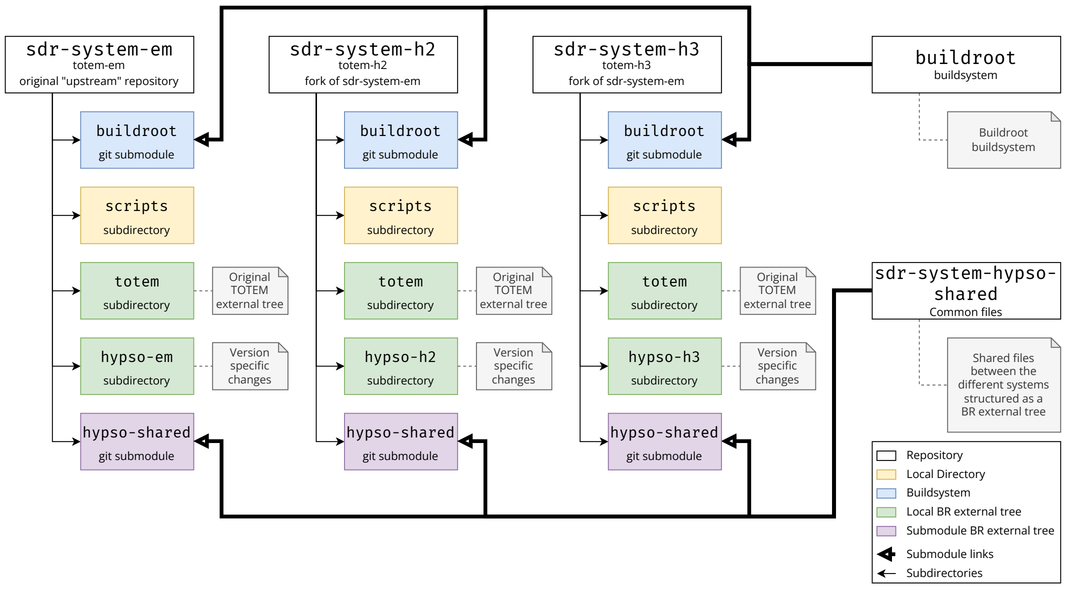


Figure 4.5: The architecture and structure of the different repositories for the different TOTEM SDR versions. The `h*` notation indicates what HYPSO satellite it will be implemented on.

This contribution by the author is essential to differentiate between build systems for different TOTEM SDRs. Using forks enables a clear separation between the systems while enabling shared issue tracking, synchronization, and ease of development. This improves maintainability and understandability.

4.1.4 Imlementation for the current EM and HYPSO-2 FM SDRs

Both versions of the `sdr-system` follow the structure discussed in Sections 4.1.1 to 4.1.3. Some significant library changes are included in `sdr-system-h2`, which includes a single, more capable package to handle the communication between the SoC and the AD9364 transceiver. The library `rx-tools` in `sdr-system-h2` replaces the following packages in `sdr-system`.

- `frontendctld` Radio frontend daemon to control the frontend.

- `i2c-slave-eeprom-ext` I^2C slave backend for Linux that implements a EEPROM memory simulator.

- `i2c-slave-iobuf` I^2C slave backend for Linux that implements a simple EEPROM Input/Output (IO) buffer for the userspace.

Otherwise, the structure of each repository is the same, and is highly maintainable because of the use of forks. All references to `hypso-em` have been changed with `hypso-h2`, including the external tree directory name. The changes are primarily in the version-specific external tree, `hypso-h2`, changing IPs, hostname, and overlay paths.

These contributions are essential to enable the operations on the HYPSO-2 TOTEM SDR. By using the structure described in Sections 4.1.1 to 4.1.3, the systems are equivalent for the developer and SDR applications.

4.2 Ensuring Resilient and Fault-Tolerant Operations for the TOTEM SDR

The design of both the HW, FW, and memory layout done by Alén Space ensure that the system boots correctly. However, it is the program `sdr-services` that enable communications and provide SDR-capabilities to the HYPSO satellite, and this cannot be guaranteed to start. Only one copy of the `sdr-services` executable exists in the system memory, and by corrupting or moving the executable, the system breaks beyond recovery. Communications with the satellite over CAN is not started, and the SDR will only use power until the EPS shuts it down after about 43 minutes. In most circumstances, human error causes the file to be missing or renamed. It is therefore critical that a[3] version of the `sdr-services` starts.

The risk matrix in Table 4.1 is formed by conducting a risk analysis of the outcome and likelihood of the most significant adverse events. As illustrated, no event ends up in the red zone, some are "high" up in the catastrophic column, but the scale is hard to visualize. Some events, like a SDR application fail, or an update of `sdr-services` fails, have little to no effect on the system's capabilities.

4.2.1 Chained Start-Up Script to Start Services

Including multiple instances of the `sdr-services` executable in multiple paths can help increase the system's resilience. This will help in the passive resistance and aid the recovery process. Implementing a script that tries to execute `sdr-services` in a specified order increases the system's passive resistance and aids the recovery process.

The original start-up script, located in `/etc/init.d/S99HypsoTotem`, only executes the `sdr-services` located in `/home/totem/hypso/sdr-services`. With a slight modification, the script can try multiple paths by moving the executing logic to a function that executes based on a path given as an argument.

```
start_sdr_service () {$1/sdr-services -m $HYPSO_SDR_ROOT 13 can0 &}
```

[3]Any version that will start. It can be an old version.

Table 4.1: Risk matrix analysis of the different scenarios.

Likelihood	**Impact**			
	Negligible	**Marginal**	**Critical**	**Catastrophic**
Certain	🟧	🟧	🟥	🟥
Likely	🟨 SDR application crashes	🟧	🟧	🟥
Possible	🟩	🟨 `sdr-services` update fail	🟧	🟥
Unlikely	🟩	🟨 `sdr-services` can not start from primary path	🟨 Need to disable OverlayFS	🟧 `sdr-services` execution check reports correct execution, but is not
Rare	🟩	🟩 Cosmic ray bit-flip	🟨	🟨 `sdr-services` does not start from any path with OverlayFS disabled

Where $1 is the path parameter supplied to the function, and `$HYPSO_SDR_ROOT` is the environment variable discussed in Section 2.4.2 and [1, Chapter 5.2.1]. This will work with both existing and non-existing files.

It would, however, be ideal to log the events with non-existing files. Bash [65] and POSIX Shell [66] provide a way of checking if a file exists. The check is embedded in the `if`-statement and can be used as in Listing 4.1.

```bash
# Test if FILE exists
FILE=$1/sdr-services
if [ -f "$FILE" ]; then
    # As file exists, execute
    echo "[MESSAGE] Starting sdr-services from $FILE".
    $FILE -m $HYPSO_SDR_ROOT 13 can0 &
else
    # As file does not exist, return and try next location
    echo "[ERROR] $FILE does not exist. Trying next location"
    return 1
fi
```

Listing 4.1: Statement to check if a file exists.

Listing 4.1 checks if the file exists. If it does, it tries to execute the file.

It is then also necessary to check that the executable is running. An existing file does not necessarily mean correct execution; this can simply be detected by checking the active processes with Linux commands. The command `ps` prints all the current processes, that is, around 30 processes for the TOTEM. `grep` is a command that can filter any text based on a couple of rules. The command `ps | grep -c '[s]dr-services*'` will run `grep` on the output of `ps` [4]. The `grep` option `-c` makes it only return the number of lines that contains the search string ”sdr-services”. ”Contains” is essential, as the process can run from anywhere in the system and have suffixes.

This can be combined with an `if` statement to check if the process started and is running correctly. If the output of the above `ps | grep -c` command is not 0, then the process is running. As seen in Listing 4.2, it is also possible to test if `sdr-services` started correctly by checking if there are 2 processes with the correct name. This works because `sdr-services` always launches two (2) processes with the same name.

```
# Check if running
if [ "$(ps | grep -c '[s]dr-services*')" -eq 0 ]; then
        echo "[ERROR] sdr-services failed to start, trying backup."
        return 1
elif [ "$(ps | grep -c '[s]dr-services*')" -ne 2 ]; then
        echo "[ERROR] sdr-services failed to start properly, killing
         ↪  and trying backup."
        kill_sdr_services
        return 1
fi
```

Listing 4.2: Statement to check if a process has started and is running correctly.

This contribution solves the problem where the `sdr-services` stalls in an infinite loop, or a malicious program such as in Listing 4.3 replaces the executable.

```
int main(int argc, char **argv) {
    for (;;) {}
    return 0;
}
```

Listing 4.3: A simple program that starts and does nothing for eternity.

Combining Listings 4.1 and 4.2 into a single function to check if a file exists, attempts to run the file, and then checks if it runs, provides the code in Listing 4.4. This function will accomplish this if the `sdr-services` file exists in the provided directory.

As this function requires a path to a directory with a potential `sdr-services`, a simple ”for loop” is necessary to make it work with multiple directories. The loop will go through a list of predefined paths and test each path until a running file is verified or the loop is completed. When correct execution is ensured, the flag `SDR_SERVICES_STARTED` is set.

[4]In more complete Linux versions, there is the command `pgrep` that accomplish this, but it is not included on the TOTEM SDR

```bash
start_sdr_service ()
{
        # Check if existing
        FILE=$1/sdr-services
        if [ -f "$FILE" ]; then
                # As sdr-services could be found, execute
                echo "[MESSAGE] Starting sdr-services from $FILE".
                $FILE -m "$HYPSO_SDR_ROOT" 13 can0 &
        else
                # sdr-services could not be found, return and try the
                ↪ next location
                echo "[ERROR] $FILE does not exist. Trying next
                ↪ location"
                return 1
        fi

        # Allow the process to start and run for a couple of seconds
        sleep 5

        # Check if running
        if [ "$(ps | grep -c '[s]dr-services*')" -eq 0 ]; then
                echo "[ERROR] sdr-services failed to start, trying
                ↪ backup."
                return 1
        elif [ "$(ps | grep -c '[s]dr-services*')" -ne 2 ]; then
                echo "[ERROR] sdr-services failed to start properly,
                ↪ killing and trying backup."
                kill_sdr_services
                return 1
        fi

        echo "[MESSAGE] Started sdr-services from $FILE".

        return 0;
}
```

Listing 4.4: Function to check if the sdr-services exists, run it, and check if it is running.

However, there are scenarios where the SDR_SERVICES_STARTED flag is set even if the execution might be unsuccessful. If the startup of sdr-services is detected as successful, then fails right after the check, the flag will be set with unsuccessful execution. This is, however, an improbable scenario because the execution must start exactly two processes with the correct name and function for at least 5 seconds, as stated in the risk matrix in Table 4.1. Because of the limited capabilities of the TOTEM SDR OS, this issue is not prioritized, but a promising solution is presented in *Section 5.2: Future Work*.

The different starting paths are defined in the variable SDR_SYSTEM_LOC. Three paths are given from the environmental variables described in the Section 2.4.1. A

```
SDR_SERVICES_STARTED=0
## Start sdr-services
# Try the different sdr-services locations,
# if started correctly, break and continue
# Store if it manages to start
for loc in ${SDR_SYSTEM_LOC}
do
if start_sdr_service "$loc"; then
        SDR_SERVICES_STARTED=1
        break 1
fi
done
```

Listing 4.5: Loop for different paths for the `sdr-services` executable. Sets the flag SDR_SERVICES_STARTED if the service is successfully executed.

”hard-coded” path `/bin/` is also included in case of an error with the environment variables. The four paths for the `sdr-services` executable are listed below.

- The HYPSO_SDR_ROOT variable pointed to `/home/totem/hypso/`. This is the path to the main executable.
- The HYPSO_SDR_BACKUP_DIR_USR variable pointed to `/usr/bin/`.
- The HYPSO_SDR_BACKUP_DIR_LOCAL variable pointed to `/usr/local/bin/`.
- ”Hard coded” path `/bin/`.

If the environment variables file is removed or non-existing and can not be set, the author has implemented a check and recovery system for that case. A simple check for the environment variables file is performed using the same file checker discussed above, as shown in Listing 4.6. A default path for HYPSO_SDR_ROOT is set to the same as mentioned above. The two backup paths are not set and will be empty when setting the SDR_SYSTEM_LOC variable. This leads to only the first and last paths being available in the startup loop. The checker can also fail, and the startup script will only have the last path `/bin/` to start from.

Combining Listings 4.4 and 4.5, the final logic for starting `sdr-services` is shown in the flow diagram in Figure 4.6.

A second application is executed on startup, the `m6p-time-sync` executable. It ensures the correct time on the TOTEM by reading the satellite bus to fetch the correct time. The start script is not as complicated as with `sdr-services` as time synchronization is not critical for comm unication between the TOTEM SDR and other satellite systems. The required functionality is ensured by slightly modifying Listing 4.4 to search for the existence of `m6p-time-sync`. The time sync only runs briefly, so checking if it is running is unnecessary, as it is treated as non-critical SW for the operations and will not reduce the capabilities of the TOTEM SDR.

```
# Source the environment vars
# This adds the HYPSO_SDR_ROOT, HYPSO_SDR_APPS and HYPSO_SDR_BACKUP_DIR
 ↪  env vars
# If not found, use a default value
ENV_FILE="/etc/profile.d/hypso_sdr.sh"
if [ -f "$ENV_FILE" ]; then
        . $ENV_FILE
        echo "[MESSAGE] Set environment varialbes from ${ENV_FILE}."
else
        echo "[CRITICAL] Could not initialize the environment
         ↪  variables. Using defaults"
        HYPSO_SDR_ROOT="/home/totem/hypso"
fi
```

Listing 4.6: Check the existence of the environment variables file. Set a default value if not found.

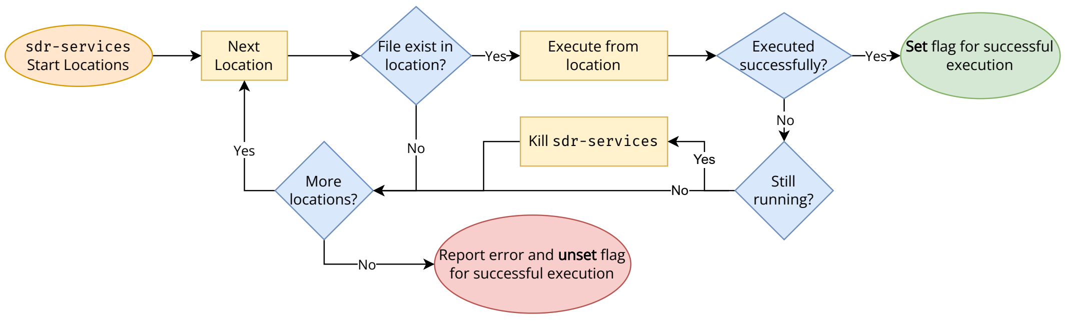


Figure 4.6: Flow diagram of the logic starting `sdr-services`.

A similar "for loop" as shown in Listing 4.5 is used to start `m6p-time-sync`. Combining this with Listing 4.7 results in the following flow diagram as shown in Figure 4.7.

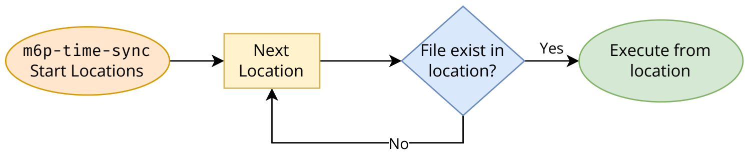


Figure 4.7: Flow diagram for executing `m6p-time-sync`.

These contributions greatly increase the system's resistance to adverse events and conditions. By including multiple paths for the critical SW, one or more paths can fail before the system goes into a reduced state. The first path (`HYPSO_SDR_ROOT`) is treated as a working directory for the executable, and this is where the main executable resides. Updating the `sdr-services` executable will only change the executable in the main path, leaving the three backup paths "alone".

```
start_m6p_time_sync ()
{
        # Check if existing
        FILE=$1/m6p-time-sync
        if [ -f "$FILE" ]; then
                # As m6p-time-sync could be found, execute
                echo "[MESSAGE] Starting m6p-time-sync from $FILE".
                $FILE
        else
                # m6p-time-sync could not be found, return and try the
                ↪  next location
                echo "[ERROR] $FILE does not exist. Trying next
                ↪  location"
                return 1
        fi

        return 0;
}
```

Listing 4.7: Function to check if the `m6p-time-sync` exists, then execute it.

4.2.2 Including `sdr-services` in the Read-Only Filesystem

A significant additional change that will increase the resilience to adverse events is to include the `sdr-services` in the read-only part of memory, i.e., in the `root0` (squashFS, RO partition) discussed in Section 2.4.2. NAND flash storage is less prone to errors and failure when only being read from. Including the `sdr-services` in read-only memory keeps the risk of errors and failure at a minimum.

BR makes this easy; add a package containing the source for `sdr-services`[5]. This would normally build and include the executable in the final RO FS.

Originally the `hypso-sw` repository was mainly created to compile `hypso-cli` and `opu-services` that are needed for the HSI operations. It was later adopted to include support for compiling `sdr-services` as most of the infrastructure and components were already in the `hypso-sw` repository. As a result, the build procedure of `sdr-services` is incompatible with how BR requires the procedure to be formulated, notably the need for a special Docker. The main problem is how `hypso-sw` uses the toolchain for the TOTEM to compile `sdr-services`. It relies on the fact that the toolchain, or `sdr-system`, exists and is located in the same root directory as `hypso-sw`.

To include `sdr-services` as a package is therefore not straightforward without a rewrite of the build procedure in `hypso-sw`. Rewriting is a possibility, but it is outside the scope of this thesis. It requires a different method of initializing compilation, a different maintenance procedure, and special knowledge about BR packages. Maintainability will be harder to ensure. Compiling `sdr-services` as described and documented by `hypso-sw` is easier and only requires learning the

[5]Generated from `hypso-sw`.

compilation process of `hypso-sw`.

```
sdr-services
├─ bin
│  └─ SDR_SERVICE_LOCATION
├─ .gitignore
├─ Config.in
└─ sdr-services.mk
```

Figure 4.8: `sdr-services` dummy package. The `sdr-services.mk` file copies `sdr-services` and `m6p-time-sync` if they are present in `bin/`

An effective solution the author has implemented is to compile `sdr-services` with the documented procedure, using a script to automate the process. BR then use a dummy package to copy the files into the correct path in the RO FS. The paths are the same as the four startup paths discussed in Section 4.2.1. The first step is to build and create the toolchain. Secondly, use that to build `sdr-services` with the documented procedure in `hypso-sw`. The last step is to rerun BR to include the files in the image. BR will work even if the executable `sdr-services` and `m6p-time-sync` files are absent in the first run. This is because it copies the entire `bin/` from the dummy package, which also contains a dummy file named `SDR_SERVICE_LOCATION`. It only contains some text to keep the folder structure after being processed by git. This file is copied into the TOTEM RO FS and deleted immediately after. The BR/Linux `install` command requires at least one file in the source directory to succeed with "installing" the directory. The procedure is visualized in Figure 4.9. The numbers in the gray stapled box correspond to the process steps listed below.

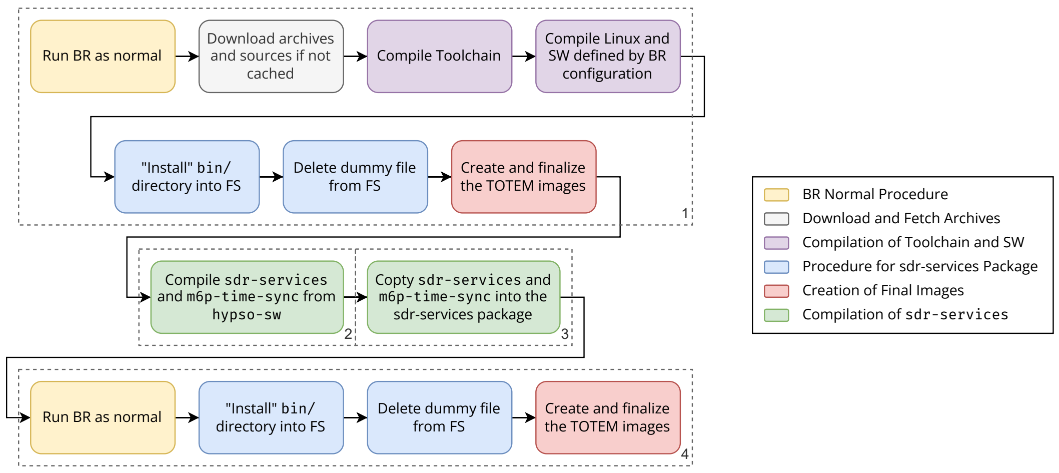

Run BR as normal
Download archives and sources if not cached
Compile Toolchain
Compile Linux and SW defined by BR configuration
"Install" bin/ directory into FS
Delete dummy file from FS
Create and finalize the TOTEM images
1
Compile sdr-services and m6p-time-sync from hypso-sw
2
Copty sdr-services and m6p-time-sync into the sdr-services package
3
Run BR as normal
"Install" bin/ directory into FS
Delete dummy file from FS
Create and finalize the TOTEM images
4
BR Normal Procedure
Download and Fetch Archives
Compilation of Toolchain and SW
Procedure for sdr-services Package
Creation of Final Images
Compilation of sdr-services

Figure 4.9: Flow of operations for first build the BR toolchain, then build `sdr-services`, copy the generated files, and then re-run BR to include the files in the image.

1. Create and compile the BR toolchain for the first time. This includes downloading all the necessary source files and libraries. A complete toolchain is

created.

The first time BR runs, the `sdr-services` and `m6p-time-sync` executables are not present, as they require the toolchain to be compiled. As discussed above, this is not a problem, as a dummy file is "installed" and then immediately removed.

This part of the process is completed by deleting the dummy files from the target RO FS before the FS is compressed into its final image.

2. Build `sdr-services` and `m6p-time-sync` from `hypso-sw` using the toolchain created in the previous step. This uses a custom Docker created to compile SW included with the `hypso-sw` repository.

3. Copy the `sdr-services` and `m6p-time-sync` files into the dummy package `🗀hypso-shared/package/sdr-services/bin/` to prepare for the next step.

4. Re-run BR to include the files copied in the previous step into the finished RO FS image.

The author's contribution of compiling and inclusion of `sdr-services` during the building of the RO FS image for the TOTEM SDR further improves the resistance and active recovery from adverse events. As mentioned in Sections 2.1.1 and 2.4.2 the entire RO FS is loaded into RAM at startup. This makes the system functional and operational without the writable FS.

4.2.3 Disabling the Overlay File System as a Last Resort

It might be applicable and necessary to update the `sdr-services` executable during operations. Either due to new features being implemented or fixes for bugs. Testing of new SW should be done on the ground with the EM version of the TOTEM SDR located at the NSSL. However, some issues may not be revealed during testing on the ground and may only appear after being implemented on the satellite. This can lead to an adverse event, compromising the SDR capabilities. The `S99HypsoTotem` start script should handle the recovery of these events by checking that `sdr-services` executed properly.

However, some features are unavailable because of the simplistic OS on the TOTEMs. This includes service watchdogs like `systemd` and more robust process management than the simple `ps` command.

As mentioned in Sections 4.2.1 and 4.2.2, the `sdr-services` executable is located at multiple paths throughout the RO FS. The start-up script tests all these paths in the order specified and will set a flag, `SDR_SERVICES_STARTED`, to indicate the successful start-up of `sdr-services`. If the execution of `sdr-services` fails to start after testing all paths, the start-up script will try to disable the OverlayFS and reboot if it is not already disabled. This is implemented in Listing 4.8 by the author and shown in the flow diagram in Figure 4.10. This will disable the access to the `data0` partition (the RW partition) but can be mounted for manual recovery and successively re-enable the OverlayFS.

```bash
# If sdr-services did not start, disable
# overlay to boot without potential broken storage in the R/W part of
↪ NAND
# If overlay is already disabled, do not reboot.
if  [ "$SDR_SERVICES_STARTED" = "0" ] && [ "$(check_overlay)" = "1" ];
↪ then
        echo "[CRITICAL] sdr-services could not be found, disabling
         ↪ overlay"
        disable_overlay

        echo "[CRITICAL] rebooting in 5 seconds to disable the overlay"
        sleep 5
        # After next reboot, overlay will be disabled.
        reboot
elif [ "$SDR_SERVICES_STARTED" = "0" ] && [ "$(check_overlay)" = "0" ];
↪ then
        echo "[ERROR] Overlay is disabled and could not start
         ↪ sdr-services from any location!"
elif [ "$SDR_SERVICES_STARTED" = "1" ] && [ "$(check_overlay)" = "0" ];
↪ then
        echo "[MESSAGE] Overlay is disabled!"
fi
```

Listing 4.8: Statement to handle the case where the start-up script could not execute the `sdr-services` from the four paths.

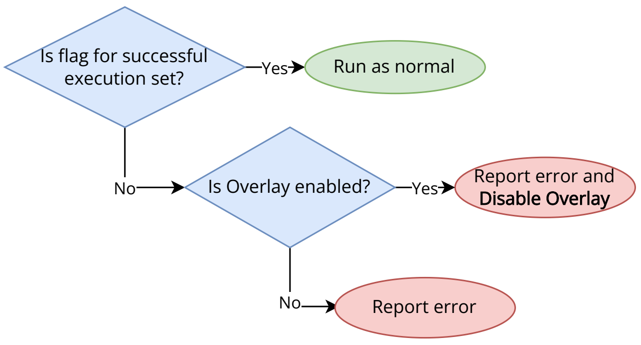

Is flag for successful execution set?
Yes
Run as normal
No
Is Overlay enabled?
Yes
Report error and Disable Overlay
No
Report error

Figure 4.10: Flow diagram of disabling the OverlayFS based on the successful execution of `sdr-services`.

The start-up script includes several helper functions to enable, disable, and check the OverlayFS. These are based on the U-Boot Linux tools `fw_printenv` and `fw_setenv` . They can modify the U-Boot configuration from the Linux user space. Listings 4.9 to 4.11 lists the implementation by the author. The U-Boot variable `bootargs` is read by Linux during start-up, and the "`overlay=`" entry is then forwarded to the `/etc/overlayroot` script to create the OverlayFS. If "`overlay=`" is empty or non-existing (i.e. disabled), the script will create a RamFS for RW capabilities while the TOTEM is powered, but lost when unpowered.

During operations, if the OverlayFS is disabled, a helper function

```
enable_overlay ()
{
        fw_setenv bootargs "$(fw_printenv -n bootargs_safe)"
        ↪ overlay=ubi0:data0
        echo "[CRITICAL] Overlay is enabled, please reboot."
}
```

Listing 4.9: Helper function to enable the OverlayFS at next reboot.

```
disable_overlay ()
{
        fw_setenv bootargs "$(fw_printenv -n bootargs_safe)"
        echo "[CRITICAL] Overlay is disabled, please reboot."
}
```

Listing 4.10: Helper function to disable the OverlayFS at next reboot.

```
check_overlay ()
{
        fw_printenv | grep -c overlay=
}
```

Listing 4.11: Helper function to check if the OverlayFS is enabled. This will return a 1 if it is and a 0 if not.

`enable_data0` will remount the RW partition to 🐧 `/tmp/data0nand/` where recovery can be performed. Then re-enabling the OverlayFS to test if the recovery succeeded. The start-up script will then handle the startup as usual, trying to start `sdr-services` with the OverlayFS enabled and disable it if unsuccessful.

```
enable_data0 ()
{
        if [ "$(check_overlay)" = "1" ]; then
                echo "Overlay is enabled, already mounted at /tmp/rwfs"
                return
        fi
        MOUNTPOINT=/tmp/data0nand
        mkdir -p "${MOUNTPOINT}"
        mount -t ubifs ubi0:data0 "${MOUNTPOINT}"
        echo "[MESSAGE] data0 is mounted at ${MOUNTPOINT}"
}
```

Listing 4.12: Helper function to mount the RW part (`data0`) of the OverlayFS if the overlay is disabled.

Combining the abovementioned procedures, the final logic is visualized in Figure 4.11.

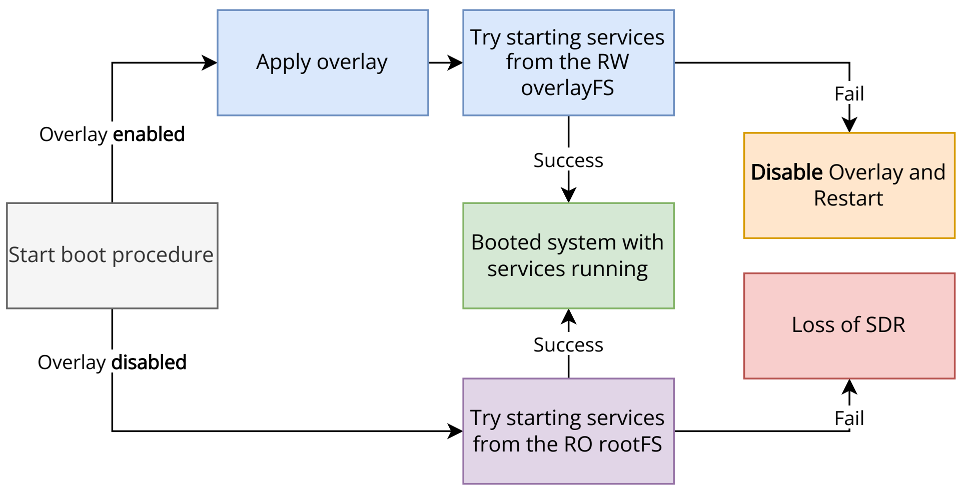

Apply overlay
Try starting services from the RW overlayFS
Fail
Disable Overlay and Restart
Overlay enabled
Success
Start boot procedure
Booted system with services running
Loss of SDR
Success
Overlay disabled
Fail
Try starting services from the RO rootFS

Figure 4.11: Flow diagram of the startup procedure with the disabling of the OverlayFS.

There is, however, a problem with this approach. It is possible to make the start-up script "believe" that the `sdr-services` is started if the `sdr-services` starts and then stops after a short while[6]. This is hard to detect because of the limited watchdog capabilities of the TOTEM Linux. It is easy for the operators to detect this as communication to the SDR is not working, and they can take action to disable the overlay without interaction with Linux. Some solutions to this issue are internally discussed in NSSL, and the most promising solution is presented in Section 5.2.

By disabling the OverlayFS if no execution path functions, the system's resilience's reaction and recovery are further improved. This will also protect against malicious actions to compromise the system, as the RO FS can not be written to, and the entire OverlayFS can be disabled. Contributions by the author enabled the detection and reaction of the script by careful research and development. The helper functions were developed to improve the system's ease of use and robustness.

4.2.4 Testing of the `S99HypsoTotem` Startup Script

Testing critical systems to ensure resilience is essential. The author created scenarios that the `S99HypsoTotem` startup script can act upon, and the logs can be studied to acquire the test results. As the script can handle missing files, startup crashes, and unsuccessful execution, test cases for these scenarios are necessary.

The script lists paths where the `sdr-services` executables are located as discussed in Section 4.2.1. For it to reach the third start path, the two preceding paths must be broken. *Broken* in the testing conditions are either *missing*, *crashes*, or *stalls* within the wait period.

[6]Longer than the waiting period in the start-up script, which is 5 seconds.

In Table 4.2, the different scenarios are presented as the ”Adverse events”. The letters A-D represent the four paths mentioned at the end of Section 4.2.1. The resulting starting path is recorded when the OverlayFS is enabled and disabled and when the environmental variables are not set.

The data is acquired by studying the logs generated by `S99HypsoTotem`. An example is in Listing 4.13. The test case is when the OverlayFS is enabled, and all four (A, B, C, and D) paths are broken. The logs generated by `m6p-time-sync` and `sdr-services` are omitted in this example. All the test logs are in Appendix B.

Table 4.2: Startup paths resulting from testing different scenarios for the `S99HypsoTotem` startup script. The letters A-D represent the four startup paths discussed in Section 4.2.1. The single letters represent the successful startup paths. Broken means either non-existing, stalls, or crashes.

Adverse Event	OverlayFS Enabled	OverlayFS Disabled	Environment Variables Not Set
None	A	A	A
A Crashes	B	B	D
A Stalls	B	B	D
A Broken	B	B	D
A+B Broken	C	C	D
A+B+C Broken	D	D	D
A+B+C+D Broken	Disable Overlay	Loss of SDR	Disable Overlay/Loss of SDR

```
################################ HYPSO TOTEM STARTUP SCRIPT BELOW
↪ ################################
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
...
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
↪ location
[ERROR] /usr/bin/sdr-services does not exist. Trying next location
[ERROR] /usr/local/bin/sdr-services does not exist. Trying next
↪ location
[ERROR] /bin/sdr-services does not exist. Trying next location
[CRITICAL] sdr-services could not be found, disabling overlay
[CRITICAL] Overlay is disabled, please reboot.
```

Listing 4.13: Test results in the log from `S99HypsoTotem`, with logs from the executable removed.

The results from the tests are presented in Table 4.2. As can be seen, the correct paths are used for every ”Adverse Event”, and the `sdr-services` only fails to start when all four paths are broken, and the OverlayFS is disabled. Cases, when the environment variables are not set have the effect of removing paths B and C from the list of starting paths, as expected.

4.3 Utilizing the BR Toolchain for a flexible and easy-to-use SDR applications development environment

As the HYPSO mission is set to have at least 2 satellites, where one (HYPSO-2) will include an SDR, the need for an environment to develop and test applications is essential. While one SDR is in the satellite, a second SDR is installed in the lidsat at the NSSL. Both SDRs are of the TOTEM type, but as mentioned and discussed in Section 4.1, the toolchain, libraries, and FW have differences.

However, because of how the different FWs and toolchains are created and structured, discussed in Section 4.1, the SDR applications are equal. This means that all applications can be developed, then compiled to work with the different versions of the TOTEM SDR. The EM version of the totem, the `totem-em`, is the most critical as it tests the applications on the ground before they are transferred to the satellite.

Together with NSSL, the author has created a separate repository containing the SDR applications, named `sdr-applications`. This enables the usual git workflow with branches, commits, issues and PRs.

4.3.1 Optmizing the Structure for Agile and Easy Development

An easy-to-use and maintainable framework is necessary to develop and compile the applications for the different TOTEM versions. A standardized structure can be obtained by compiling the applications by utilizing `make` 's framework and features [67]. The compilation process can then be fully automated by standardizing the structure for each application. The suggested process flow diagram is shown in Figure 4.12.

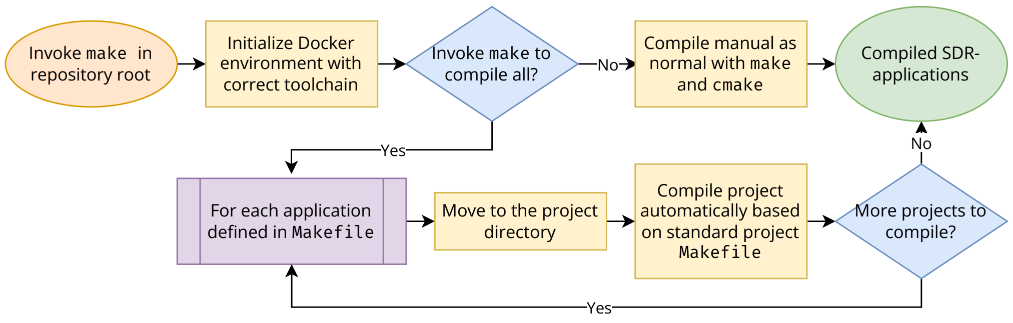

Invoke make in repository root
Initialize Docker environment with correct toolchain
Invoke make to compile all?
No
Compile manual as normal with make and cmake
Compiled SDR-applications
Yes
For each application defined in Makefile
Move to the project directory
Compile project automatically based on standard project Makefile
More projects to compile?
No
Yes

Figure 4.12: Flow diagram for how applications can be recursively, automatically built.

Each application is structured as individual projects[7]. The "recipes" for `make` (the `Makefile`) is central, as it enables each application to be compiled by just invoking `make` . The standardized project is not strict on the structure as long as a `Makefile` with the required "recipes" are included in the project's root.

Following the structure proposed in Figure 4.13 for every single application,

[7]Each project could be a separate repository and could be compiled isolated from the other applications.

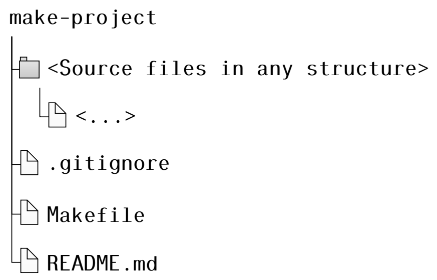


Figure 4.13: Structure for a single application. Note that this structure should be able to be standalone.

these can be recursively processed after each other. Each project `Makefile` is subject to including a set of standardized "recipes", as shown in Appendix A.5.

`make` The default "recipe" shall compile the final project. This includes setting all the correct compiler flags and including libraries.

`make clean` The `clean` "recipe" shall clean the project directory of output files and other helper files.

`make bundle` The `bundle` "recipe" shall copy and move the output files to a different directory to easily upload multiple applications simultaneously.

Besides these three, each project `Makefile` can contain other necessary recipes and variables. The story changes a bit for `CMake` projects. The original approach is not optimal as `CMake` is a tool to create `Makefiles` and will overwrite the standardized project `Makefile`. The author's solution is to modify the name of the `Makefile` to `Makefile.mk` and modify the logic to recursively invoke `make` to respect `CMake` projects. This is shown in Appendix A.6

An "apps" `Makefile` in `apps/Makefile` recursively goes through each application project directory and invokes the `make` command for each project, utilizing the standardized structure, specifically the standard "recipes". This file is shown in Appendix A.4

The main script can subsequently create a structure that includes all the applications in an output directory for mass transfer of the applications. This can then be uploaded to the development SDR for testing and eventually to the satellite(s) for operations.

By adhering to the standardized structure implemented in PR `sdr-applications#6` by the author, it is possible to ensure a well-organized, maintainable, and modularized SDR applications development environment. Because of the modularization of each project, multiple developers can develop projects in multiple branches. PRs are then created from the branches to be included in the "main" branch in the repository to increase the code quality.

```
sdr-applications
├── apps
│   ├── <project directories>
│   ├── <...>
│   └── Makefile
├── docs
│   ├── <Documentation structure>
│   ├── <...>
│   └── README.md
├── scripts
│   └── <Helper scripts>
├── .gitignore
├── Makefile
└── README.md
```

Figure 4.14: Structure for the repository containing all the applications. Each project is residing in `apps/`. Note the multiple layers of `Makefiles` from the root and towards the project.

4.3.2 Utilizing Multiple Toolchains for Compilation

Multiple toolchains are needed to compile SW for the different TOTEM SDRs, so a toolchain ”agnostic” Docker environment is desired. Since all versions of the `sdr-system` BR repositories follow the same structure, discussed in Section 4.1, this is trivial.

Each version of the `sdr-system` (`sdr-system`, `sdr-system-h2` etc.) contains (after building) a complete toolchain and the file `cross-compile-env`. The `cross-compile-env` file contains environment variables with paths to the tools in the toolchain. Linux usually has these variables set, but custom variables are used as the TOTEM SDRs requires a custom toolchain. `make` have a built-in feature to automatically use these variables where applicable, notably when compiling C/C++.

A custom Docker is used to separate the use of the custom toolchain from the host computer. The specific toolchain version is supplied to the Docker run command, which dynamically mounts the correct toolchain for compiling, making the Docker ”toolchain-agnostic”. This procedure is performed by the `scripts/run-docker.sh` script to instantiate and build Docker to the required specification.

Most of the development is done and tested on the EM TOTEM, so the default `sdr-system` (`sdr-system-em`) toolchain is used by default. The root `Makefile` handles the initialization of Docker and enables the change of toolchains. This

`Makefile` is shown in Appendix A.3. `make` allows for overwriting variables by supplying the variable and a new value to the command. To change the toolchain from `sdr-system` to the `sdr-system-h2`, invoke `make PLATFORM=h2`. This will be forwarded to the `scripts/run-docker.sh` script, and the correct toolchain is used. Any valid designator discussed in Section 4.1, like `h2` can be used. Each toolchain must be cloned/downloaded to the same root directory as `sdr-applications`, as the "root" `Makefile` checks for toolchains there.

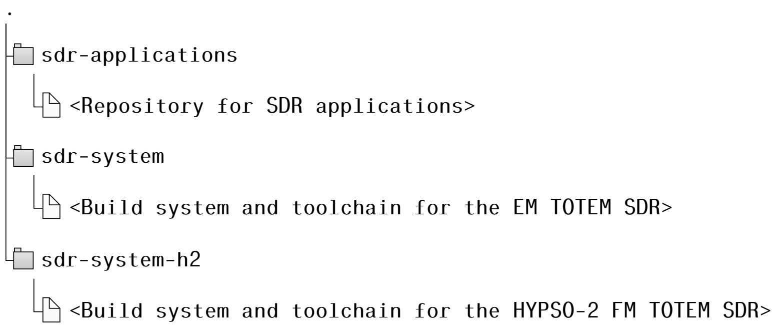

.
sdr-applications
<Repository for SDR applications>
sdr-system
<Build system and toolchain for the EM TOTEM SDR>
sdr-system-h2
<Build system and toolchain for the HYPSO-2 FM TOTEM SDR>

Figure 4.15: Organization of the different repositories containing the toolchains for the different TOTEM SDRs and the `sdr-applications` repository.

These essential contributions enable the NSSL to rapidly develop SDR applications for a range of different TOTEM SDRs. This is achieved by implementing a toolchain "agnostic" Docker environment to easy interchange the different toolchains with a simple command.

4.4 Designing a Safe Development Environment for the Flight Model SDR

Although the TOTEM is a space-proven and space-hardened HW designed to survive the harsh environment during launch and its lifecycle in space, it is still good practice to prevent anything undesirable from happening.

During testing and development, dust and grease from handling the TOTEM may distort or even damage the thermal and other properties. Electrostatic Discharge (ESD) is a hazard during handling of electronics HW that must be considered. Not touching the device directly is the best practice in this case.

Development and testing are necessary to maintain operations and add valuable new capabilities. According to the user manual [68] and the update procedure [41], the TOTEM should only be handled while wearing latex gloves and in an ESD-safe area. A safe environment is designed to minimize the risk of damaging the TOTEM during development and reduce the need to prepare and set up the TOTEM to be used.

The safe environment shall adhere to a set of requirements as listed below. These requirements were detected for a desirable and safe environment while still adhering to the maximum and minimum values specified in the TOTEM SDR datasheet and Interface Control Documents (ICD) [40, 41]

`GSE-ENV-010` **The enclosure shall be ESD safe.**
Removes the risk of being touched and causing an ESD.

`GSE-ENV-020` **The enclosure shall be airtight.**
Minimizes the settlement of dust and other contaminants from the air.

`GSE-ENV-030` **The enclosure shall include external power connection(s), for** +5 V **and ground.**
Reduces the risk of connecting the power wrong and reduces the handling of the FM.

`GSE-ENV-060` **The enclosure shall include an external CAN bus connection.**
Satellite bus that enables communications between devices in the satellite.

`GSE-ENV-050` **The enclosure shall include an external** 50 Ω **antenna connection.**
Connection for an external antenna. It can be an actual antenna, measurement device, or a dummy load.

`GSE-ENV-060` **The enclosure shall include an external Universal Asynchronous Receiver/Transmitter (UART) connection.**
Connections for communicating with the TOTEM for development purposes.

`GSE-ENV-070` **The enclosure should include an external ethernet connection.**
Connections for communicating with the TOTEM over IP for development purposes.

`GSE-ENV-080` **The enclosure should include an external JTAG connection.**
Connections for communicating and programming the SoC within the TOTEM for development purposes.

As the shipping of the TOTEM to NA is due in August 2023, the safe environment will be temporary. That does not allow for any deviations from the requirements but opens up for simplifying the solutions for cable management and routing.

The following items were chosen as the final components for a safe environment that meets requirements `GSE-ENV-010` through `GSE-ENV-060`.

- ESD safe plastic box with a latched lid.
- Metal standoffs with corresponding metal nuts.
- DBUS9 connector with flat-cable and hand-made connections to the CAN bus connectors on the TOTEM.
- Through-hole banana plugs for power, one red and one black.
- Through-hole antenna adapter provided by Alén Space.

- USB to UART with long cables with the correct connector for usage with the TOTEM.

Only the power and antenna have mounted connectors on the box's walls. CAN and UART have shorter cables fed through smaller holes, and friction fits in the box.

The development connectors specified in `GSE-ENV-070` and `GSE-ENV-080` were not prioritized as the components necessary were unavailable at the time of implementation.

The final assembly can be seen in Figure 4.16.

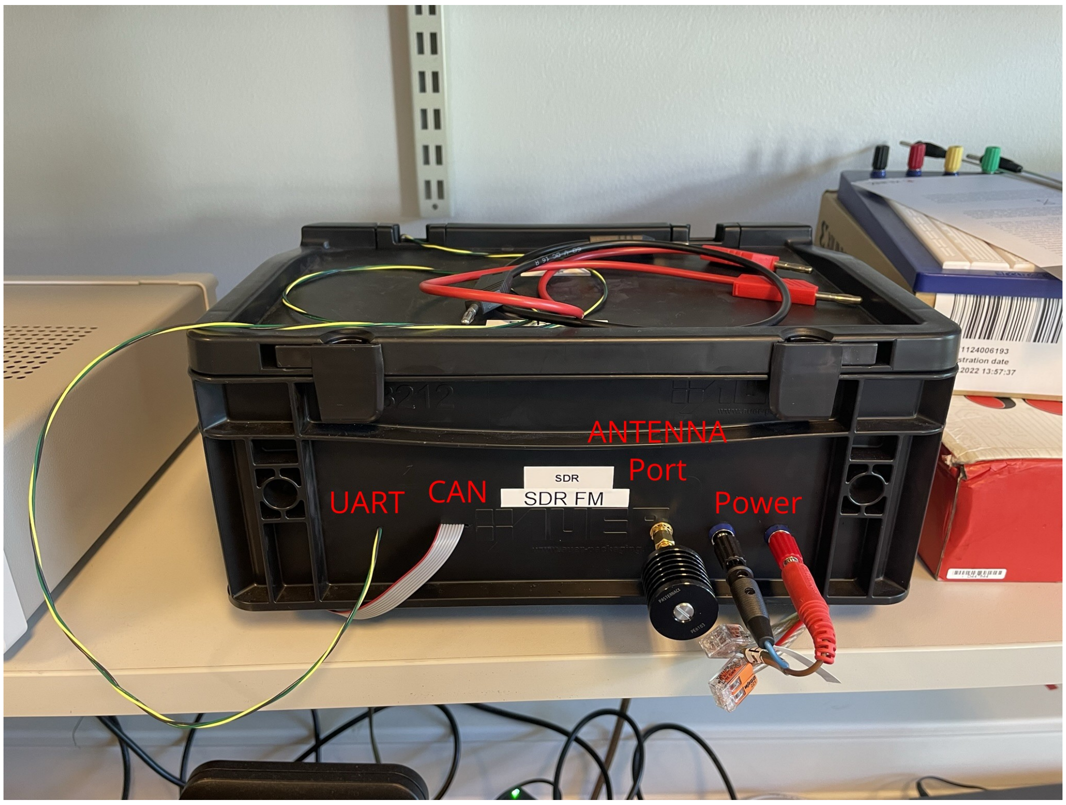


Figure 4.16: The finished safe environment for the FM version of the TOTEM. already connected to the HYPSO-2 LidSat.

These contributions are essential for the development of the FM TOTEM SDR. The safe environment enables the NSSL to develop rapidly and test SW on the model that is eventually to be sent into space without taking unnecessary risks regarding the handling of the FM.

Chapter 5

Conclusion

Don't adventures ever have an end? I suppose not. Someone else always has to carry on the story.

Bilbo Baggins

At the start of this project, the state of the development environments for the TOTEMs was functional but had a very steep learning curve and not a clear separation of what was done by NSSL and what was done by Alén Space. Maintaining two build systems for two different TOTEMs posed a risk of breaking during the project's life. Compilation of the SDR applications were also tricky to understand for less experienced system developers and programmers, thus prone to human errors. As most of the users and developers are Bachelor's and Master's students, their time on the project is limited. Hence, it is important that they can focus on, for example, their SDR application development instead of struggling and spending time setting up the build system and development environments.

The original built-in resilience of the payload to adverse events and conditions was insufficient to recover from all relevant adverse events and conditions. This could lead to losing the SDR in space, where repairs would be impossible.

5.1 Contributions and Impact

Utilizing features from BR, the author contributes to a clear separation of Alén Space's original configuration of the TOTEM, and the additions made by the author improved the system's maintainability and usability. External trees split the build system into three separate directories, one for the original from Alén Space and two from NSSL. Payload code adaptions made by NSSL and the author are split to separate between the adaptations for a unique TOTEM and adaptations that should be applied to all TOTEMs, making a reusable system and improving maintainability further. The shared external tree was distributed and version controlled using git submodules between all TOTEMs, which promotes reusability. These changes and removing dependencies completed in the specialization project [1] made the build systems highly portable, re-creatable, maintainable, and robust.

By creating a custom and automated development environment using Docker, the compilation of SDR-applications SW written in C/C++ enables rapid development and ease of use for time-constrained students. The toolchains used to compile the TOTEM SDR FW were also automated to be included and used to compile other SW that run on the TOTEM SDR. Switching between the specific hardware toolchains is done by specifying simple compiler flags.

System resilience in the harsh environment of space is essential. As the system configuration of the TOTEM was limited, multiple mission-critical SW locations were implemented to increase the passive resistance and improve the recovery and reaction capabilities to adverse events and conditions. The original start-up script was modified to handle the different events that can cause the SW not to start, which made the system more robust and will possibly extend the lifespan of the TOTEM SDR mission.

By combining the different changes and adaptations made to the build system and system resilience, the longevity of the HYPSO-2 satellite mission is ensured, as well as enabling developers to build applications and utilize the SDR for the entire lifetime of the satellite.

5.2 Future Work

During the later stages of the development of the TOTEM SDR system resilience, there were some concerns that the crucial `sdr-services` started but did nothing, like an infinite loop. If this happens during startup, the startup script should be able to handle it. However, because of the limitation of the TOTEM SDR's OS, it is a challenge to detect erroneous execution after this point.

The question is if it is possible to disable the overlayFS without having contact with the SDR. U-Boot [69] have mechanisms to detect failures during boot and change the boot process accordingly without interacting with Linux. U-Boot implements a "Boot Limit", that counts the amount of boot attempts before success. When reaching the boot limit, an alternative boot command is used instead. This command disables the overlay by setting the `bootargs` entry, mentioned in Listings 4.9 and 4.10, to the values of `bootargs_safe`.

As the documentation of U-Boot is hard to come by, these mechanisms are not fully understood. The options `bootlimit` and `altbootcmd` is set, indicating that the mechanism is active. During testing, the author could not get this to work properly. With this mechanism, it might be possible to rapidly power up and power down the SDR to force the boot procedure to use the `altbootcmd`, which disables the overlay. After the next reboot, the TOTEM SDR only loads the RO FS, which will put the system in a known working state as discussed in Section 4.2.3.

Bibliography

[1] Øyvind Paulsen Skaaden, "System integration, maintainability, and portability of an embedded linux system for use with a CubeSat SDR payload," Internal Document, Specialization Project in Embedded Systems, Norwegian University of Science and Technology, Trondheim, Dec. 2022, 33 pp.

[2] NTNU Small Satellite Lab. "HYPSO-project." (), [Online]. Available: http://hypso.space/ (visited on Apr. 26, 2023).

[3] HYPSO Project Team, *HYPSO-MRD-001 HYPSO-1 Mission Requirements Document*. HYPSO Internal Document, Dec. 4, 2020.

[4] HYPSO Project Team, *HYPSO-2-MRD-001 HYPSO-2 Mission Requirements Document*. HYPSO Internal Document, Aug. 29, 2022.

[5] M. E. Grotte, R. Birkeland, E. Honore-Livermore, S. Bakken, J. L. Garrett, E. F. Prentice, F. Sigernes, M. Orlandic, J. T. Gravdahl, and T. A. Johansen, "Ocean color hyperspectral remote sensing with high resolution and low latencythe HYPSO-1 CubeSat mission," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–19, 2022, ISSN: 0196-2892, 1558-0644. DOI: 10.1109/TGRS.2021.3080175. [Online]. Available: https://ieeexplore.ieee.org/document/9447150/ (visited on Nov. 16, 2022).

[6] Cal poly cubesat laboratory. "CubeSat," CubeSat. (2023), [Online]. Available: https://www.cubesat.org (visited on Jun. 5, 2023).

[7] Cal Poly CubeSat Laboratory, *CP-CDS-R14.1 CubeSat Design Specification* (CP-CDS-R14.1), 14.1. Cal Poly CubeSat Laboratory, Feb. 2022. [Online]. Available: https://www.cubesat.org/s/CDS-REV14_1-2022-02-09.pdf (visited on Dec. 15, 2022).

[8] NanoAvionics. "Nanosatellites & CubeSats," NanoAvionics. (), [Online]. Available: https://nanoavionics.com/ (visited on Apr. 29, 2023).

[9] Canadian Space Agency. "CubeSats in a nutshell," Canadian Space Agency. Last Modified: 2022-05-06. (Apr. 12, 2017), [Online]. Available: https://www.asc-csa.gc.ca/eng/satellites/cubesat/what-is-a-cubesat.asp (visited on Dec. 15, 2022).

[10] SpaceX. "SmallSat rideshare program," SpaceX - Rideshare. (Jun. 7, 2023), [Online]. Available: http://www.spacex.com/rideshare (visited on Jun. 7, 2023).

[11] M. Dillinger, K. Madani, and N. Alonistioti, Eds., *Software defined radio: architectures, systems, and functions*, Wiley series in software radio, Hoboken, NJ: Wiley, 2003, 416 pp., ISBN: 978-0-470-85164-7.

[12] A. Pini. "Learn the fundamentals of software-defined radio and how to use it with a low-cost module," Digi-Key Electronics. (Jun. 30, 2020), [Online]. Available: https://www.digikey.no/en/articles/learn-the-fundamentals-of-software-defined-radio (visited on Dec. 12, 2022).

[13] Alén Space. "Alén space | nanosatellites - CubeSats - small satellites," Alén Space. (), [Online]. Available: https://alen.space/ (visited on Dec. 6, 2022).

[14] A. Holt and C.-Y. Huang, *Embedded Operating Systems: A Practical Approach* (Undergraduate Topics in Computer Science), 2nd ed. 2018. Cham: Springer International Publishing : Imprint: Springer, 2018, 1 p., ISBN: 978-3-319-72977-0. DOI: 10.1007/978-3-319-72977-0.

[15] Wikipedia Collaborators, *Operating system*, in *Wikipedia*, Page Version ID: 1115693051, Oct. 12, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Operating_system&oldid=1115693051 (visited on Nov. 17, 2022).

[16] Wikipedia Collaborators, *Linux on embedded systems*, in *Wikipedia*, Page Version ID: 1110104595, Sep. 13, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Linux_on_embedded_systems&oldid=1110104595 (visited on Nov. 17, 2022).

[17] David A Rusling, *Processes*. The Linux Documentation Project, 1999. [Online]. Available: https://tldp.org/LDP/tlk/kernel/processes.html (visited on Jun. 26, 2023).

[18] International Business Machines (IBM) Corporation, *System/360 Model 67 Time Sharing System Preliminary Technical Summary* (S/360-00). 1966, vol. C20-1647-0.

[19] A. Bhattacharjee and D. Lustig, *Architectural and operating system support for virtual memory* (Synthesis lectures on computer architecture 42). San Rafael, California: Morgan & Claypool Publishers, 2018, 157 pp., ISBN: 978-1-62705-602-1. DOI: 10.2200/S00795ED1V01Y201708CAC042.

[20] *Filesystems*, in collab. with L. Wirzenius, Joanna Oja, Stephen Stafford, and Alex Weeks. The Linux Documentation Project. [Online]. Available: https://tldp.org/LDP/sag/html/filesystems.html (visited on Apr. 23, 2023).

[21] BlackBerry Limited. "Embedded OS, support and services | RTOS, hypervisor | BlackBerry QNX." (2022), [Online]. Available: https://blackberry.qnx.com/en.html (visited on Apr. 27, 2023).

[22] Real Time Engineers Ltd. "FreeRTOS - market leading RTOS (real time operating system) for embedded systems with internet of things extensions," FreeRTOS. (2023), [Online]. Available: https://www.freertos.org/index.html (visited on Apr. 27, 2023).

[23] Microsoft. "Windows for IoT." (2023), [Online]. Available: https://developer.microsoft.com/en-us/windows/iot/ (visited on Apr. 27, 2023).

[24] EE Times and Embedded, “2019 embedded markets study,” Mar. 2019. [Online]. Available: https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf (visited on Apr. 27, 2023).

[25] The kernel development community. “CPU architectures the linux kernel documentation.” (), [Online]. Available: https://docs.kernel.org/arch.html (visited on Nov. 25, 2022).

[26] *Computer data storage*, in *Wikipedia*, Page Version ID: 1151091968, Apr. 21, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Computer_data_storage&oldid=1151091968#cite_ref-3 (visited on Apr. 23, 2023).

[27] Lars Wirzenius, Joanna Oja, Stephen Stafford, and Alex Weeks, *The Linux System Administrator’s Guide*, 0.9. The Linux Documentation Project. [Online]. Available: https://tldp.org/LDP/sag/html/index.html (visited on Apr. 23, 2023).

[28] The kernel development community. “UBI file system the linux kernel documentation.” (), [Online]. Available: https://docs.kernel.org/filesystems/ubifs.html (visited on Nov. 25, 2022).

[29] Memory Technology Devices. “MTD - general documentation,” Memory Technology Device (MTD) Subsystem for Linux. (), [Online]. Available: http://www.linux-mtd.infradead.org/doc/general.html (visited on Nov. 29, 2022).

[30] Memory Technology Devices. “UBI - unsorted block images,” Memory Technology Device (MTD) Subsystem for Linux. (), [Online]. Available: http://www.linux-mtd.infradead.org/doc/ubi.html (visited on Nov. 29, 2022).

[31] Memory Technology Device. “NAND data,” Memory Technology Device (MTD) Subsystem for Linux. (), [Online]. Available: http://www.linux-mtd.infradead.org/doc/nand.html (visited on Nov. 29, 2022).

[32] X. Jimenez, D. Novo, and P. Ienne, “Wear unleveling: Improving NAND flash lifetime by balancing page endurance,” in *12th USENIX Conference on File and Storage Technologies (FAST 14)*, Santa Clara, CA: USENIX Association, Feb. 2014, pp. 47–59, ISBN: ISBN 978-1-931971-08-9. [Online]. Available: https://www.usenix.org/conference/fast14/technical-sessions/presentation/jimenez (visited on Dec. 1, 2022).

[33] Western Digital, *NAND Evolution and its Effects on Solid State Drive (SSD) Useable Life*, WP-001-01R. Nov. 12, 2011. [Online]. Available: https://web.archive.org/web/20111112000643/http://www.wdc.com/WDProducts/SSD/whitepapers/en/NAND_Evolution_0812.pdf (visited on Jun. 25, 2023).

[34] The kernel development community. “Overlay filesystem the linux kernel documentation.” (2023), [Online]. Available: https://docs.kernel.org/filesystems/overlayfs.html (visited on May 4, 2023).

[35] Buildroot Association. “Buildroot - making embedded linux easy.” (2023), [Online]. Available: https://buildroot.org/ (visited on Apr. 26, 2023).

[36] Yocto Project. “Yocto project it’s not an embedded linux distribution it creates a custom one for you.” (2023), [Online]. Available: `https://www.yoctoproject.org/` (visited on Apr. 26, 2023).

[37] The Buildroot developers, *The Buildroot User Manual*. Buildroot Association, Mar. 12, 2023. [Online]. Available: `https://buildroot.org/downloads/manual/manual.html` (visited on Apr. 26, 2023).

[38] NanoAvionics. “6u nanosatellite bus m6p,” NanoAvionics. (), [Online]. Available: `https://nanoavionics.com/small-satellite-buses/6u-nanosatellite-bus-m6p/` (visited on Apr. 29, 2023).

[39] R. Birkeland, G. Quintana-Diaz, E. Honore-Livermore, T. Ekman, F. A. Agelet, and T. A. Johansen, “Development of a multi-purpose SDR payload for the HYPSO-2 satellite,” in *2022 IEEE Aerospace Conference (AERO)*, Big Sky, MT, USA: IEEE, Mar. 5, 2022, pp. 1–11, ISBN: 978-1-66543-760-8. DOI: `10.1109/AERO53065.2022.9843447`. [Online]. Available: `https://ieeexplore.ieee.org/document/9843447/` (visited on Nov. 16, 2022).

[40] Alén Space, *ICD - TOTEM Motherboard & UHF Frontend*, 1.0. Alén Space, internal document, Dec. 7, 2022. (visited on Jun. 25, 2023).

[41] Alén Space, *TOTEM - Motherboard User Manual*, 1.1. Alén Space, internal document, Apr. 28, 2021, 34 pp. (visited on Jun. 18, 2023).

[42] Alén Space, *TOTEM - UHF Frontend - Datasheet*, 1.1. Alén Space, internal document, Oct. 8, 2018, 13 pp. (visited on Dec. 13, 2022).

[43] Advanced Micro Devices, Inc. “Zynq-7000 SoC,” AMD Xilinx. (2022), [Online]. Available: `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html` (visited on Dec. 6, 2022).

[44] Analog Devices, *AD9364 Datasheet*, C. Analog Devices, Jul. 2014, 32 pp. [Online]. Available: `https://www.analog.com/en/products/ad9364.html#product-overview` (visited on Dec. 6, 2022).

[45] Analog Devices, *Linux kernel - variant from analog devices, inc.* original-date: 2012-10-08T12:49:16Z, Nov. 17, 2022. [Online]. Available: `https://github.com/analogdevicesinc/linux` (visited on Nov. 23, 2022).

[46] libcsp, *The cubesat space protocol*, original-date: 2011-10-07T10:35:34Z, Dec. 11, 2022. [Online]. Available: `https://github.com/libcsp/libcsp` (visited on Dec. 14, 2022).

[47] pothosware, *Soapy SDR - vendor and platform neutral SDR support library*. original-date: 2014-10-04T19:40:28Z, Dec. 12, 2022. [Online]. Available: `https://github.com/pothosware/SoapySDR` (visited on Dec. 14, 2022).

[48] GNU Radio project. “GNU radio - the free & open source radio ecosystem,” GNU Radio. (2022), [Online]. Available: `https://www.gnuradio.org/` (visited on Dec. 14, 2022).

[49] J. Brtis and M. Mcevilley, *Systems Engineering for Resilience*, MP1909495. MITRE, Jul. 1, 2019. [Online]. Available: `https://www.researchgate.net/publication/334549424_Systems_Engineering_for_Resilience` (visited on May 8, 2023).

[50] D. Firesmith, *System Resilience: What Exactly is it?* Carnegie Mellon's Software Engineering Institute: Carnegie Mellon University, Software Engineering Institute's Insights (blog), Nov. 25, 2019. [Online]. Available: https://insights.sei.cmu.edu/blog/system-resilience-what-exactly-is-it/ (visited on May 8, 2023).

[51] National Aeronautics and Space Administration, *NASA Systems Engineering Handbook Revision 2*, NASA SP-2016-6105 Rev2. NASA, Jun. 20, 2017. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/nasa_systems_engineering_handbook_0.pdf (visited on Jun. 19, 2023).

[52] European Cooperation for Space Standardization, *ECSS-E-ST-10C Rev.1 System engineering general requirements (15 February 2017)*, ECSS-E-ST-10C Rev.1. ECSS, Feb. 15, 2017. [Online]. Available: https://ecss.nl/standard/ecss-e-st-10c-rev-1-system-engineering-general-requirements-15-february-2017/ (visited on Jan. 25, 2023).

[53] V. Cripps, "Phases of an ESA hardware project expained," Swedish Institute of Space Physics, Uppsala, May 23, 2018. [Online]. Available: https://www.space.irfu.se/seminars/20180523-Cripps-HW_Project.pdf (visited on Jun. 19, 2023).

[54] Stack Overflow. "Stack overflow developer survey 2022," Stack Overflow. (May 2022), [Online]. Available: https://survey.stackoverflow.co/2022/ (visited on Jun. 16, 2023).

[55] Git developers. "Git," Git. (2023), [Online]. Available: https://git-scm.com/ (visited on May 4, 2023).

[56] S. Chacon and B. Straub, *Pro Git*, 2.1.391-2-g2fbc35b8. Apress, Apr. 27, 2023. [Online]. Available: https://git-scm.com/book/en/v2 (visited on May 4, 2023).

[57] GitHub Inc. "GitHub - build software better, together," GitHub. (2023), [Online]. Available: https://github.com (visited on May 4, 2023).

[58] GitHub Inc. "About GitHub," GitHub. (2023), [Online]. Available: https://github.com/about (visited on May 4, 2023).

[59] GitHub Inc. "GitHub features: The right tools for the job," GitHub. (2023), [Online]. Available: https://github.com/features/ (visited on Jun. 17, 2023).

[60] A. Mundra, S. Misra, and C. A. Dhawale, "Practical scrum-scrum team: Way to produce successful and quality software," in *2013 13th International Conference on Computational Science and Its Applications*, Ho Chi Minh City, Vietnam: IEEE, Jun. 2013, pp. 119–123, ISBN: 978-0-7695-5045-9. DOI: 10.1109/ICCSA.2013.25. [Online]. Available: http://ieeexplore.ieee.org/document/6681108/ (visited on Jun. 19, 2023).

[61] GitHub Inc. "About forks," GitHub Docs. (2023), [Online]. Available: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/about-forks (visited on May 29, 2023).

[62] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, Jan. 20, 2015, ISSN: 0163-5980. DOI: 10.1145/2723872.2723882. [Online]. Available: https://doi.org/10.1145/2723872.2723882 (visited on Jun. 19, 2023).

[63] Docker Inc. "Docker overview," Docker Documentation. (Nov. 30, 2022), [Online]. Available: https://docs.docker.com/get-started/overview/ (visited on Dec. 1, 2022).

[64] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *International Journal of Computer Science and Network Security*, vol. 17, no. 3, p. 9, Mar. 2017. (visited on Jun. 19, 2023).

[65] C. Ramey, *The GNU bourne-again shell*, Sep. 26, 2022. [Online]. Available: https://tiswww.case.edu/php/chet/bash/bashtop.html (visited on May 8, 2023).

[66] The Open Group and IEEE. "Shell command language." (2018), [Online]. Available: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html (visited on Jun. 5, 2023).

[67] Free Software Foundation, Inc., *GNU make*, 0.77. Free Software Foundation, Inc., Feb. 26, 2023. [Online]. Available: https://www.gnu.org/software/make/manual/make.html (visited on Jun. 22, 2023).

[68] Alén Space, *TOTEM - Motherboard - Datasheet*, 1.5. Alén Space, internal document, Apr. 26, 2021, 21 pp. (visited on May 25, 2023).

[69] DEMX Software Engineering. "Das u-boot – the universal boot loader," DENX. (2023), [Online]. Available: https://www.denx.de/wiki/U-Boot/WebHome (visited on Jun. 18, 2023).

Appendix A

Selected Source Code Listings

A.1 `S99HypsoTotem` Initialization Script

```
#!/bin/sh
### BEGIN INIT INFO
# Should be placed in /etc/init.d/
# Provides:          S99HypsoTotem
# Required-Start: $ALL
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:   2 3 5
# Default-Stop:
# Description:     Linux Startup Script
### END INIT INFO

# Default time before poweroff
# 2500 seconds is around 42 minutes, a bit shorter than the EPS
 ↪  poweroff
SECONDS_BEFORE_POWEROFF=2500

# Function to kill all instances of sdr-services
kill_sdr_services ()
{
        killall sdr-services
}

# Function to check the existence of sdr-services and run it.
# Args: path to directory with the executable
# Returns: 0 if successful, 1 if fail
start_sdr_service ()
{
        # Check if existing
        FILE=$1/sdr-services
        if [ -f "$FILE" ]; then
```

```
                # As sdr-services could be found, execute
                echo "[MESSAGE] Starting sdr-services from $FILE".
                $FILE -m "$HYPSO_SDR_ROOT" 13 can0 &
        else
                # sdr-services could not be found, return and try the next location
                echo "[ERROR] $FILE does not exist. Trying next location"
                return 1
        fi

        # Allow the process to start and run for a couple of seconds
        sleep 5

        # Check if running
        if [ "$(ps | grep -c '[s]dr-services*')" -eq 0 ]; then
                echo "[ERROR] sdr-services failed to start, trying backup."
                return 1
        elif [ "$(ps | grep -c '[s]dr-services*')" -ne 2 ]; then
                echo "[ERROR] sdr-services failed to start properly, killing and trying backup."
                kill_sdr_services
                return 1
        fi

        echo "[MESSAGE] Started sdr-services from $FILE".

        return 0;
}

# Function to check the existence of m6p-time-sync and run it.
# Args: path to directory with the executable
# Returns: 0 if successful, 1 if fail
start_m6p_time_sync ()
{
        # Check if existing
        FILE=$1/m6p-time-sync
        if [ -f "$FILE" ]; then
                # As m6p-time-sync could be found, execute
                echo "[MESSAGE] Starting m6p-time-sync from $FILE".
                $FILE
        else
                # m6p-time-sync could not be found, return and try the next location
                echo "[ERROR] $FILE does not exist. Trying next location"
                return 1
        fi

        return 0;
}
```

```


# Function to enable the overlay at next reboot
# Args: NONE
# Returns: NONE
enable_overlay ()
{
        fw_setenv bootargs "$(fw_printenv -n bootargs_safe)"
        ↪  overlay=ubi0:data0
        echo "[CRITICAL] Overlay is enabled, please reboot."
}


# Function to disable the overlay at next reboot
# Args: NONE
# Returns: NONE
disable_overlay ()
{
        fw_setenv bootargs "$(fw_printenv -n bootargs_safe)"
        echo "[CRITICAL] Overlay is disabled, please reboot."
}


# Function to check if the overlay is enabled
# Args: NONE
# Returns: NONE
check_overlay ()
{
        fw_printenv | grep -c overlay=
}


# Function to mount the data partition if booted without overlay
# Args: NONE
# Returns: NONE
enable_data0 ()
{
        if [ "$(check_overlay)" = "1" ]; then
                echo "Overlay is enabled, already mounted at /tmp/rwfs"
                return
        fi
        MOUNTPOINT=/tmp/data0nand
        mkdir -p "${MOUNTPOINT}"
        mount -t ubifs ubi0:data0 "${MOUNTPOINT}"
        echo "[MESSAGE] data0 is mounted at ${MOUNTPOINT}"
}

# Main start script
start ()
{
        echo "################################ HYPSO TOTEM STARTUP
        ↪  SCRIPT BELOW ################################"

        # Check if overlay is disabled
        if ! check_overlay; then
                echo "[CRITICAL] Overlay is not present, no
                 ↪  non-volatile memory"
                echo "[CRITICAL] SDR-SERVICES could not be detected in
                 ↪  the overlay"
        fi

        # Shutdown automatically after 41 minutes and 40 seconds.
        # This is a shorter period than the power-off for the EPS
        # which is about 43 minutes.
        printf "sleep %s\npoweroff" "$SECONDS_BEFORE_POWEROFF" |
         ↪  /bin/sh &

        # Source the environment vars
        # This adds the HYPSO_SDR_ROOT, HYPSO_SDR_APPS and
         ↪  HYPSO_SDR_BACKUP_DIR env vars
        # If not found, use a default value
        ENV_FILE="/etc/profile.d/hypso_sdr.sh"
        if [ -f "$ENV_FILE" ]; then
                . $ENV_FILE
                echo "[MESSAGE] Set environment varialbes from
                 ↪  ${ENV_FILE}."
        else
                echo "[CRITICAL] Could not initialize the environment
                 ↪  variables. Using defaults"
                HYPSO_SDR_ROOT="/home/totem/hypso"
        fi

        # Locations for the sdr-services executable.
        # These are defined in /etc/profile.d/hypso_sdr.sh
        SDR_SYSTEM_LOC="${HYPSO_SDR_ROOT} ${HYPSO_SDR_BACKUP_DIR_USR}
         ↪  ${HYPSO_SDR_BACKUP_DIR_LOCAL} /bin"

        # Sync time with the EPS
        ## Start m6p-time-sync
        # Try the different m6p-time-sync locations (same as sdr),
        # if started correctly, break and continue
        for loc in ${SDR_SYSTEM_LOC}
        do
        if start_m6p_time_sync "$loc"; then
                break 1
        fi
        done

        sleep 5

        SDR_SERVICES_STARTED=0
        ## Start sdr-services
        # Try the different sdr-services locations,
        # if started correctly, break and continue
        # Store if it manages to start
        for loc in ${SDR_SYSTEM_LOC}
        do
        if start_sdr_service "$loc"; then
                SDR_SERVICES_STARTED=1
                break 1
        fi
        done

        # If sdr-services did not start, disable
        # overlay to boot without potential broken storage in the R/W
        ↪  part of NAND
        # If overlay is already disabled, do not reboot.
        if  [ "$SDR_SERVICES_STARTED" = "0" ] && [ "$(check_overlay)" =
        ↪  "1" ]; then
                echo "[CRITICAL] sdr-services could not be found,
                ↪  disabling overlay"
                disable_overlay

                echo "[CRITICAL] rebooting in 5 seconds to disable the
                ↪  overlay"
                sleep 5
                # After next reboot, overlay will be disabled.
                reboot
        elif [ "$SDR_SERVICES_STARTED" = "0" ] && [ "$(check_overlay)"
        ↪  = "0" ]; then
                echo "[ERROR] Overlay is disabled and could not start
                ↪  sdr-services from any location!"
        elif [ "$SDR_SERVICES_STARTED" = "1" ] && [ "$(check_overlay)"
        ↪  = "0" ]; then
                echo "[MESSAGE] Overlay is disabled!"
        fi

}

stop ()
{
        kill_sdr_services
        echo "Bye, bye Totem."
}

restart()
{
        stop
        start
}

case "$1" in
start)
        start; ;;
stop)
        stop; ;;
```

```
restart)
        restart; ;;
enable_overlay)
        enable_overlay; ;;
disable_overlay)
        disable_overlay; ;;
mount_data0)
        enable_data0; ;;
*)
        echo "Usage: $0
        ↪ {start|stop|restart|enable_overlay|disable_overlay|
        ↪ mount_data0}"
        exit 1
esac
exit $?
```

A.2 `build_config.sh` Configuration Combiner/Merger

```
#!/bin/bash

DIR="$(dirname "${BASH_SOURCE[0]}")"
MERGE_CFG=${DIR}/../buildroot/support/kconfig/merge_config.sh
MERGE_ARGS="-m"

HYPSOTOTEM_CONFIG_NAME="hypso_totem"

# Function to merge configuration fragments
# Args: string list of all configuration fragments
# Return: NONE
build_config() {
    echo "Writing defconfig to
     ↪  ${DIR}/${HYPSOTOTEM_CONFIG_NAME}_defconfig"
    KCONFIG_CONFIG=${DIR}/${HYPSOTOTEM_CONFIG_NAME}_defconfig
     ↪  CONFIG_="BR2_" ${MERGE_CFG} ${MERGE_ARGS} "$@"
}


# Check if configuration fragments is supplied
if [ $# -ne 0 ]; then
    build_config "$@"
    exit 0
else
    echo "Usage: $0 [configurations]"
    echo "No configs specified. Please specify configs. Output will be
     ↪  in '${DIR}/${HYPSOTOTEM_CONFIG_NAME}_defconfig'."
fi
```

A.3 `Makefile` Root Makefile in `sdr-applications`

```
SHELL                     := /bin/bash
APPS_DIR             := apps
PLATFORM         := em
SDR_SYSTEM_PATH := sdr-system

ifneq ($(PLATFORM),em)
        SDR_SYSTEM_PATH :=$(SDR_SYSTEM_PATH)-$(PLATFORM)
endif

all: docker

# Check for build system
../$(SDR_SYSTEM_PATH)/cross-compile-env:
        make -C ../$(SDR_SYSTEM_PATH)

# Force build the docker image
build-docker:
        scripts/run-docker.sh -f

# Run docker as normal, build if not exist
docker: ../$(SDR_SYSTEM_PATH)/cross-compile-env
        scripts/run-docker.sh -s $(SDR_SYSTEM_PATH)
```

A.4 Makefile Apps Makefile in sdr-applications

```

SHELL            := /bin/bash
MKDIR            := mkdir -p
RM               := rm -rf

MAKE_SUBDIRS     = example-make example-soapy
CMAKE_SUBDIRS    = example-cmake

PLATFORM         ?= sdr-system

#.PHONY: example-make
#example-make:
#        (cd example-make && make)

all: normal_make c_make

setup-env:
        source ../totem/firmware/cross-compile-env
        printenv



.PHONY: normal_make $(MAKE_SUBDIRS)
.PHONY: c_make $(CMAKE_SUBDIRS)

normal_make: $(MAKE_SUBDIRS)
c_make: $(CMAKE_SUBDIRS)

$(MAKE_SUBDIRS):
        (cd $@ && make)

$(CMAKE_SUBDIRS):
        (cd $@ && make -f Makefile.mk)



define bundle-command-make
(cd $(1) && make bundle)

endef

define bundle-command-cmake
(cd $(1) && make -f Makefile.mk bundle)

endef

.PHONY: bundle
bundle:
        @ $(MKDIR) build/$(PLATFORM)
        @ $(foreach dir,$(MAKE_SUBDIRS),$(call bundle-command-make,
         ↪  $(dir)))
        @ $(foreach dir,$(CMAKE_SUBDIRS),$(call bundle-command-cmake,
         ↪  $(dir)))
        @ sleep 2
        mv $(shell find build -maxdepth 1 -type f -print)
         ↪  build/$(PLATFORM)


define clean-command-make
(cd $(1) && make clean)

endef

define clean-command-cmake
(cd $(1) && make -f Makefile.mk clean)

endef

#@ $(RM) build/$(PLATFORM)
.PHONY: clean
clean:
        $(foreach dir,$(MAKE_SUBDIRS),$(call clean-command-make,
         ↪  $(dir)))
        $(foreach dir,$(CMAKE_SUBDIRS),$(call clean-command-cmake,
         ↪  $(dir)))
```

A.5 Makefile for make Applications in sdr-applications

```
# Name of output executable goes here
EXECUTABLE    = example-soapy

# Sourcefiles goes here
SOURCES      += example-soapy.c
# Linker flags goes here, like -llibiio or -lm
LDFLAGS             += -lSoapySDR -lfftw3f -lm

$(EXECUTABLE): $(SOURCES)

.PHONY: bundle
bundle: $(EXECUTABLE)
        cp $(EXECUTABLE) ../build

.PHONY: clean
clean:
        rm -f $(EXECUTABLE)
```

A.6 Makefile for cmake Applications in sdr-applications

```
hello:
        cmake helloworld/
        make


.PHONY: bundle
bundle: hello
        cp hello ../build

.PHONY: clean
clean:
        -make clean
```

Appendix B

Logs from Testing `S99HypsoTotem`

B.1 Overlay Enabled

Working `sdr-services` in `/home/totem/hypso/`

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[MESSAGE] Starting sdr-services from /home/totem/hypso/sdr-services.
[MESSAGE] Started sdr-services from /home/totem/hypso/sdr-services.
```

The `sdr-services` Crashes Immediately

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[MESSAGE] Starting sdr-services from /home/totem/hypso/sdr-services.
[ERROR] sdr-services failed to start, trying backup.
[MESSAGE] Starting sdr-services from /usr/bin/sdr-services.
[MESSAGE] Started sdr-services from /usr/bin/sdr-services.
```

The `sdr-services` Stalls Immediately

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[MESSAGE] Starting sdr-services from /home/totem/hypso/sdr-services.
[ERROR] sdr-services failed to start properly, killing and trying
↪  backup.
[MESSAGE] Starting sdr-services from /usr/bin/sdr-services.
[MESSAGE] Started sdr-services from /usr/bin/sdr-services.
```

Working sdr-services in /usr/bin/

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪  location
[MESSAGE] Starting sdr-services from /usr/bin/sdr-services.
[MESSAGE] Started sdr-services from /usr/bin/sdr-services.
```

Working sdr-services in /usr/local/bin/

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪  location
[ERROR] /usr/bin/sdr-services does not exist. Trying next location
[MESSAGE] Starting sdr-services from /usr/local/bin/sdr-services.
[MESSAGE] Started sdr-services from /usr/local/bin/sdr-services.
```

Working sdr-services in /bin/

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪  location
[ERROR] /usr/bin/sdr-services does not exist. Trying next location
[ERROR] /usr/local/bin/sdr-services does not exist. Trying next
 ↪  location
[MESSAGE] Starting sdr-services from /bin/sdr-services.
[MESSAGE] Started sdr-services from /bin/sdr-services.
```

No working sdr-services

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪  location
[ERROR] /usr/bin/sdr-services does not exist. Trying next location
[ERROR] /usr/local/bin/sdr-services does not exist. Trying next
 ↪  location
[ERROR] /bin/sdr-services does not exist. Trying next location
[CRITICAL] sdr-services could not be found, disabling overlay
[CRITICAL] Overlay is disabled, please reboot.
```

The m6p-time-sync Broken in /home/totem/hypso/

```
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[ERROR] /home/totem/hypso/m6p-time-sync does not exist. Trying next
↪  location
[MESSAGE] Starting m6p-time-sync from /usr/bin/m6p-time-sync.
[MESSAGE] Starting sdr-services from /home/totem/hypso/sdr-services.
[MESSAGE] Started sdr-services from /home/totem/hypso/sdr-services.
```

B.2 Overlay Disabled

Working sdr-services in /home/totem/hypso/

```
[CRITICAL] Overlay is not present, no non-volatile memory
[CRITICAL] SDR-SERVICES could not be detected in the overlay
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[MESSAGE] Starting sdr-services from /home/totem/hypso/sdr-services.
[MESSAGE] Started sdr-services from /home/totem/hypso/sdr-services.
[MESSAGE] Overlay is disabled!
```

Working sdr-services in /usr/bin/

```
[CRITICAL] Overlay is not present, no non-volatile memory
[CRITICAL] SDR-SERVICES could not be detected in the overlay
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪  location
[MESSAGE] Starting sdr-services from /usr/bin/sdr-services.
[MESSAGE] Started sdr-services from /usr/bin/sdr-services.
[MESSAGE] Overlay is disabled!
```

Working sdr-services in /usr/local/bin/

```
[CRITICAL] Overlay is not present, no non-volatile memory
[CRITICAL] SDR-SERVICES could not be detected in the overlay
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪  location
[ERROR] /usr/bin/sdr-services does not exist. Trying next location
[MESSAGE] Starting sdr-services from /usr/local/bin/sdr-services.
[MESSAGE] Started sdr-services from /usr/local/bin/sdr-services.
[MESSAGE] Overlay is disabled!
```

Working `sdr-services` in `/bin/`

```
[CRITICAL] Overlay is not present, no non-volatile memory
[CRITICAL] SDR-SERVICES could not be detected in the overlay
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪ location
[ERROR] /usr/bin/sdr-services does not exist. Trying next location
[ERROR] /usr/local/bin/sdr-services does not exist. Trying next
 ↪ location
[MESSAGE] Starting sdr-services from /bin/sdr-services.
[MESSAGE] Started sdr-services from /bin/sdr-services.
[MESSAGE] Overlay is disabled!
```

No working `sdr-services`

```
[CRITICAL] Overlay is not present, no non-volatile memory
[CRITICAL] SDR-SERVICES could not be detected in the overlay
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪ location
[ERROR] /usr/bin/sdr-services does not exist. Trying next location
[ERROR] /usr/local/bin/sdr-services does not exist. Trying next
 ↪ location
[ERROR] /bin/sdr-services does not exist. Trying next location
[ERROR] Overlay is disabled and could not start sdr-services from any
 ↪ location!
```

The `m6p-time-sync` Broken in `/home/totem/hypso/`

```
[CRITICAL] Overlay is not present, no non-volatile memory
[CRITICAL] SDR-SERVICES could not be detected in the overlay
[MESSAGE] Set environment varialbes from /etc/profile.d/hypso_sdr.sh.
[ERROR] /home/totem/hypso/m6p-time-sync does not exist. Trying next
 ↪ location
[MESSAGE] Starting m6p-time-sync from /usr/bin/m6p-time-sync.
[MESSAGE] Starting sdr-services from /home/totem/hypso/sdr-services.
[MESSAGE] Started sdr-services from /home/totem/hypso/sdr-services.
[MESSAGE] Overlay is disabled!
```

B.3 Missing Environment Variables

Working `sdr-services` in `/home/totem/hypso/`

```
[CRITICAL] Could not initialize the environment variables. Using
 ↪  defaults
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[MESSAGE] Starting sdr-services from /home/totem/hypso/sdr-services.
[MESSAGE] Started sdr-services from /home/totem/hypso/sdr-services.
```

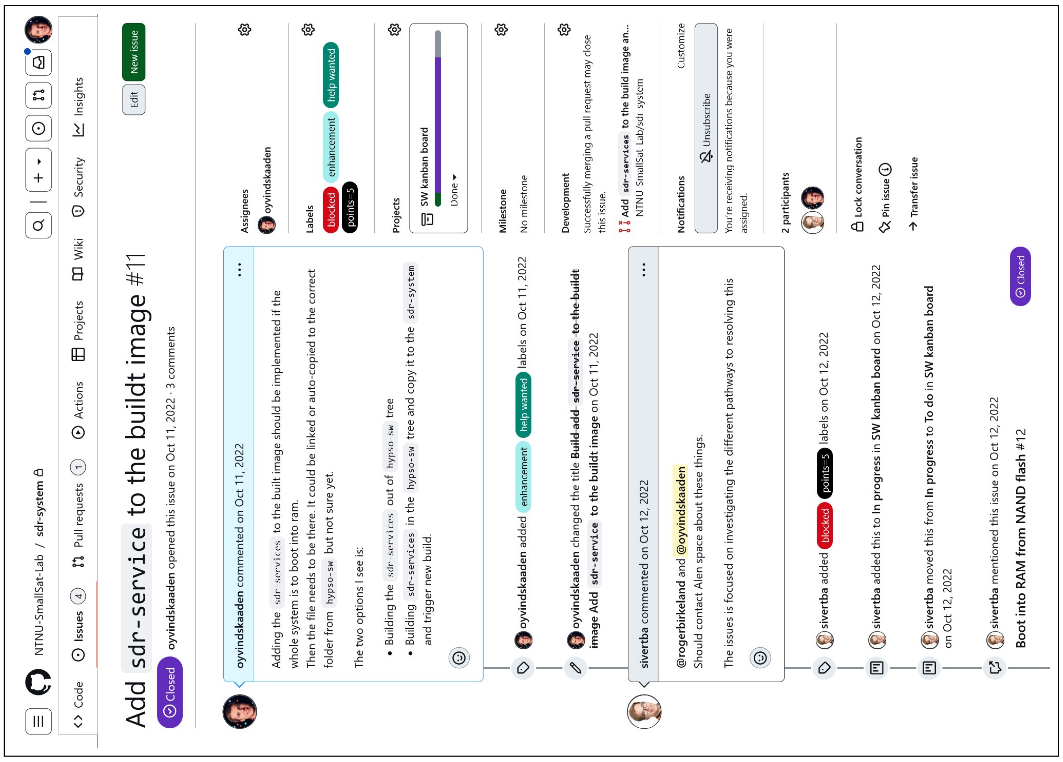
Working `sdr-services` in `/bin/`

```
[CRITICAL] Could not initialize the environment variables. Using
 ↪  defaults
[MESSAGE] Starting m6p-time-sync from /home/totem/hypso/m6p-time-sync.
[ERROR] /home/totem/hypso/sdr-services does not exist. Trying next
 ↪  location
[MESSAGE] Starting sdr-services from /bin/sdr-services.
[MESSAGE] Started sdr-services from /bin/sdr-services.
```

Appendix C

Issues

C.1 `sdr-system#11`



NTNU-SmallSat-Lab / **sdr-system**

Code · Issues 4 · Pull requests 1 · Actions · Projects · Wiki · Security · Insights

Add `sdr-service` to the buildt image #11

Edit · New issue

Closed · **oyvindskaaden** opened this issue on Oct 11, 2022 · 3 comments

oyvindskaaden commented on Oct 11, 2022

Adding the `sdr-services` to the built image should be implemented if the whole system is to boot into ram.
Then the file needs to be there. It could be linked or auto-copied to the correct folder from `hypso-sw` but not sure yet.

The two options I see is:

- Building the `sdr-services` out of `hypso-sw` tree
- Building `sdr-services` in the `hypso-sw` tree and copy it to the `sdr-system` and trigger new build.

oyvindskaaden added enhancement help wanted labels on Oct 11, 2022

oyvindskaaden changed the title ~~**Build add `sdr-service` to the buildt image**~~ **Add `sdr-service` to the buildt image** on Oct 11, 2022

sivertba commented on Oct 12, 2022

@rogerbirkeland and **@oyvindskaaden**
Should contact Alen space about these things.

The issues is focused on investigating the different pathways to resolving this

sivertba added blocked points=5 labels on Oct 12, 2022

sivertba added this to **In progress** in **SW kanban board** on Oct 12, 2022

sivertba moved this from **In progress** to **To do** in **SW kanban board** on Oct 12, 2022

sivertba mentioned this issue on Oct 12, 2022

Boot into RAM from NAND flash #12 · Closed

Assignees
oyvindskaaden

Labels
blocked · enhancement · help wanted · points=5

Projects
SW kanban board
Done

Milestone
No milestone

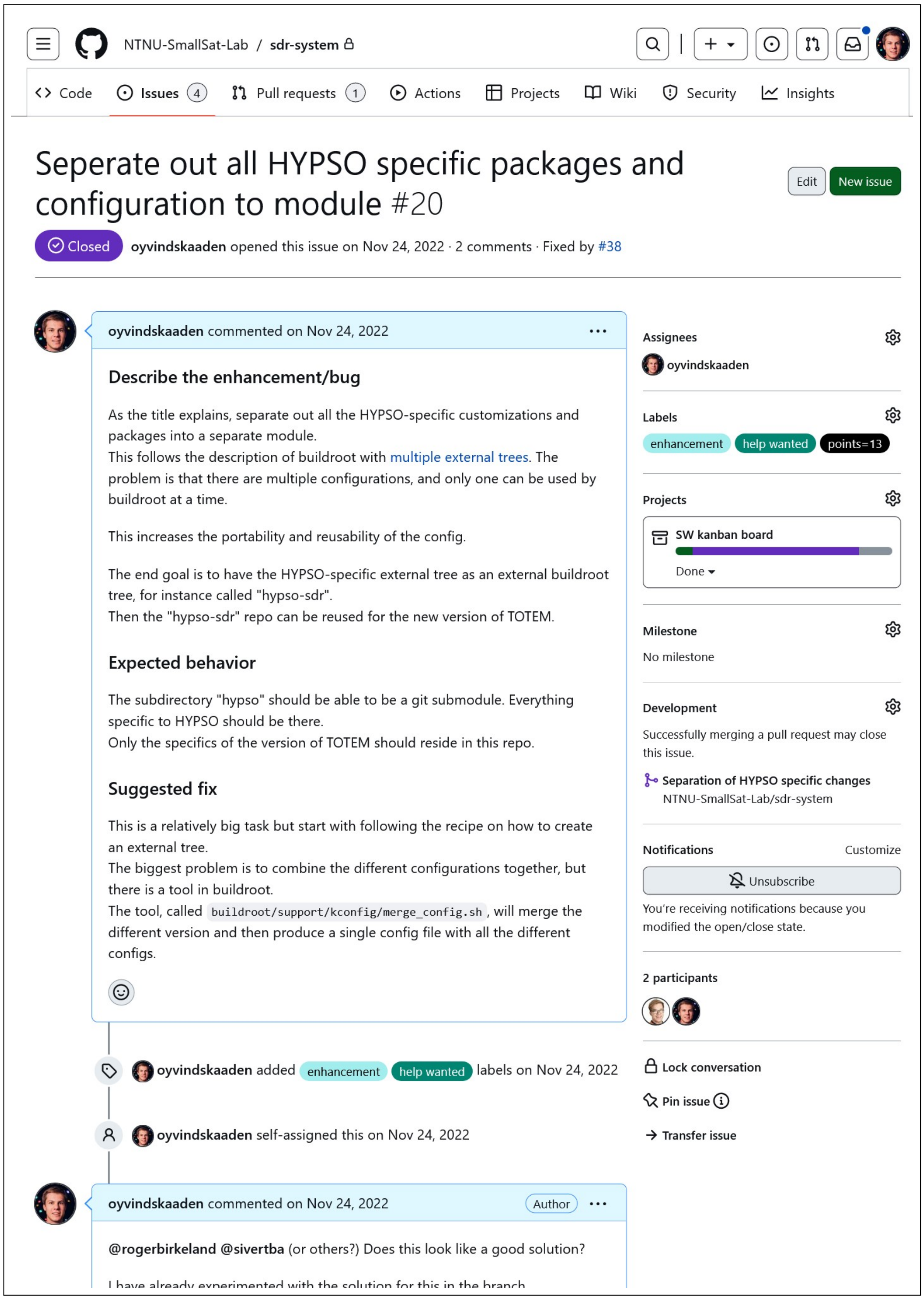
Development
Successfully merging a pull request may close this issue.
Add `sdr-services` to the build image an…
NTNU-SmallSat-Lab/sdr-system

Notifications · Customize
Unsubscribe
You're receiving notifications because you were assigned.

2 participants

Lock conversation
Pin issue
Transfer issue

C.2 `sdr-system#20`



NTNU-SmallSat-Lab / sdr-system

Code · Issues 4 · Pull requests 1 · Actions · Projects · Wiki · Security · Insights

Seperate out all HYPSO specific packages and configuration to module #20

Edit · New issue

Closed · **oyvindskaaden** opened this issue on Nov 24, 2022 · 2 comments · Fixed by #38

oyvindskaaden commented on Nov 24, 2022

Describe the enhancement/bug

As the title explains, separate out all the HYPSO-specific customizations and packages into a separate module.
This follows the description of buildroot with multiple external trees. The problem is that there are multiple configurations, and only one can be used by buildroot at a time.

This increases the portability and reusability of the config.

The end goal is to have the HYPSO-specific external tree as an external buildroot tree, for instance called "hypso-sdr".
Then the "hypso-sdr" repo can be reused for the new version of TOTEM.

Expected behavior

The subdirectory "hypso" should be able to be a git submodule. Everything specific to HYPSO should be there.
Only the specifics of the version of TOTEM should reside in this repo.

Suggested fix

This is a relatively big task but start with following the recipe on how to create an external tree.
The biggest problem is to combine the different configurations together, but there is a tool in buildroot.
The tool, called `buildroot/support/kconfig/merge_config.sh`, will merge the different version and then produce a single config file with all the different configs.

oyvindskaaden added enhancement help wanted labels on Nov 24, 2022

oyvindskaaden self-assigned this on Nov 24, 2022

oyvindskaaden commented on Nov 24, 2022 · Author

@rogerbirkeland **@sivertba** (or others?) Does this look like a good solution?

Assignees: oyvindskaaden

Labels: enhancement · help wanted · points=13

Projects: SW kanban board — Done

Milestone: No milestone

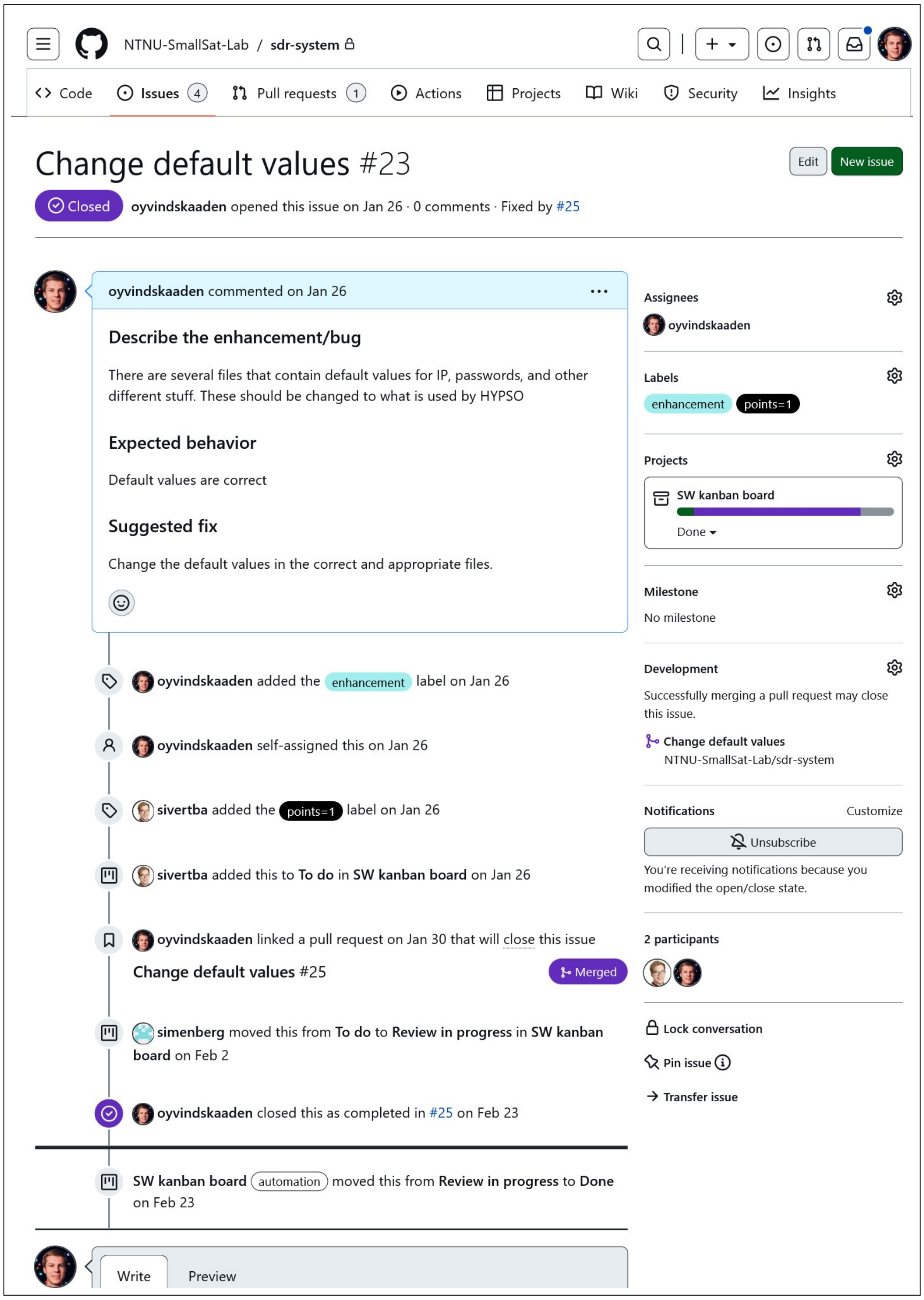
Development: Successfully merging a pull request may close this issue.
Separation of HYPSO specific changes — NTNU-SmallSat-Lab/sdr-system

Notifications · Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation
Pin issue
Transfer issue

C.3 sdr-system#23



NTNU-SmallSat-Lab / sdr-system

Code | Issues 4 | Pull requests 1 | Actions | Projects | Wiki | Security | Insights

Change default values #23

Edit | New issue

Closed **oyvindskaaden** opened this issue on Jan 26 · 0 comments · Fixed by #25

oyvindskaaden commented on Jan 26

Describe the enhancement/bug

There are several files that contain default values for IP, passwords, and other different stuff. These should be changed to what is used by HYPSO

Expected behavior

Default values are correct

Suggested fix

Change the default values in the correct and appropriate files.

oyvindskaaden added the enhancement label on Jan 26

oyvindskaaden self-assigned this on Jan 26

sivertba added the points=1 label on Jan 26

sivertba added this to **To do** in **SW kanban board** on Jan 26

oyvindskaaden linked a pull request on Jan 30 that will close this issue

Change default values #25 Merged

simenberg moved this from **To do** to **Review in progress** in **SW kanban board** on Feb 2

oyvindskaaden closed this as completed in #25 on Feb 23

SW kanban board (automation) moved this from **Review in progress** to **Done** on Feb 23

Write | Preview

Assignees: oyvindskaaden

Labels: enhancement, points=1

Projects: SW kanban board — Done

Milestone: No milestone

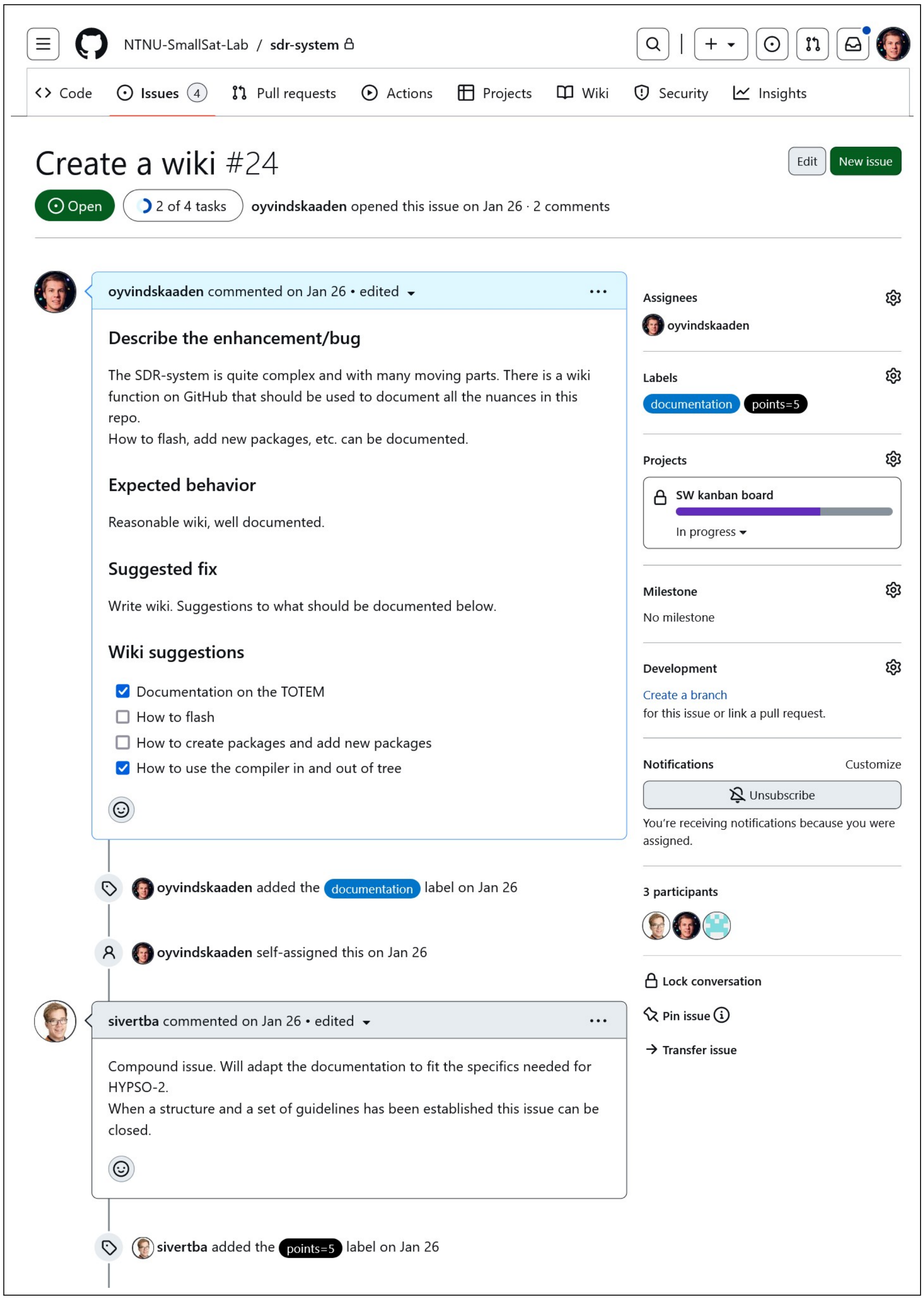
Development: Successfully merging a pull request may close this issue.
Change default values — NTNU-SmallSat-Lab/sdr-system

Notifications — Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation
Pin issue
Transfer issue

C.4 `sdr-system#24`



Create a wiki #24

Open · 2 of 4 tasks · **oyvindskaaden** opened this issue on Jan 26 · 2 comments

oyvindskaaden commented on Jan 26 • edited

Describe the enhancement/bug

The SDR-system is quite complex and with many moving parts. There is a wiki function on GitHub that should be used to document all the nuances in this repo.
How to flash, add new packages, etc. can be documented.

Expected behavior

Reasonable wiki, well documented.

Suggested fix

Write wiki. Suggestions to what should be documented below.

Wiki suggestions

- [x] Documentation on the TOTEM
- [ ] How to flash
- [ ] How to create packages and add new packages
- [x] How to use the compiler in and out of tree

oyvindskaaden added the documentation label on Jan 26

oyvindskaaden self-assigned this on Jan 26

sivertba commented on Jan 26 • edited

Compound issue. Will adapt the documentation to fit the specifics needed for HYPSO-2.
When a structure and a set of guidelines has been established this issue can be closed.

sivertba added the points=5 label on Jan 26

Assignees: oyvindskaaden

Labels: documentation, points=5

Projects: SW kanban board — In progress

Milestone: No milestone

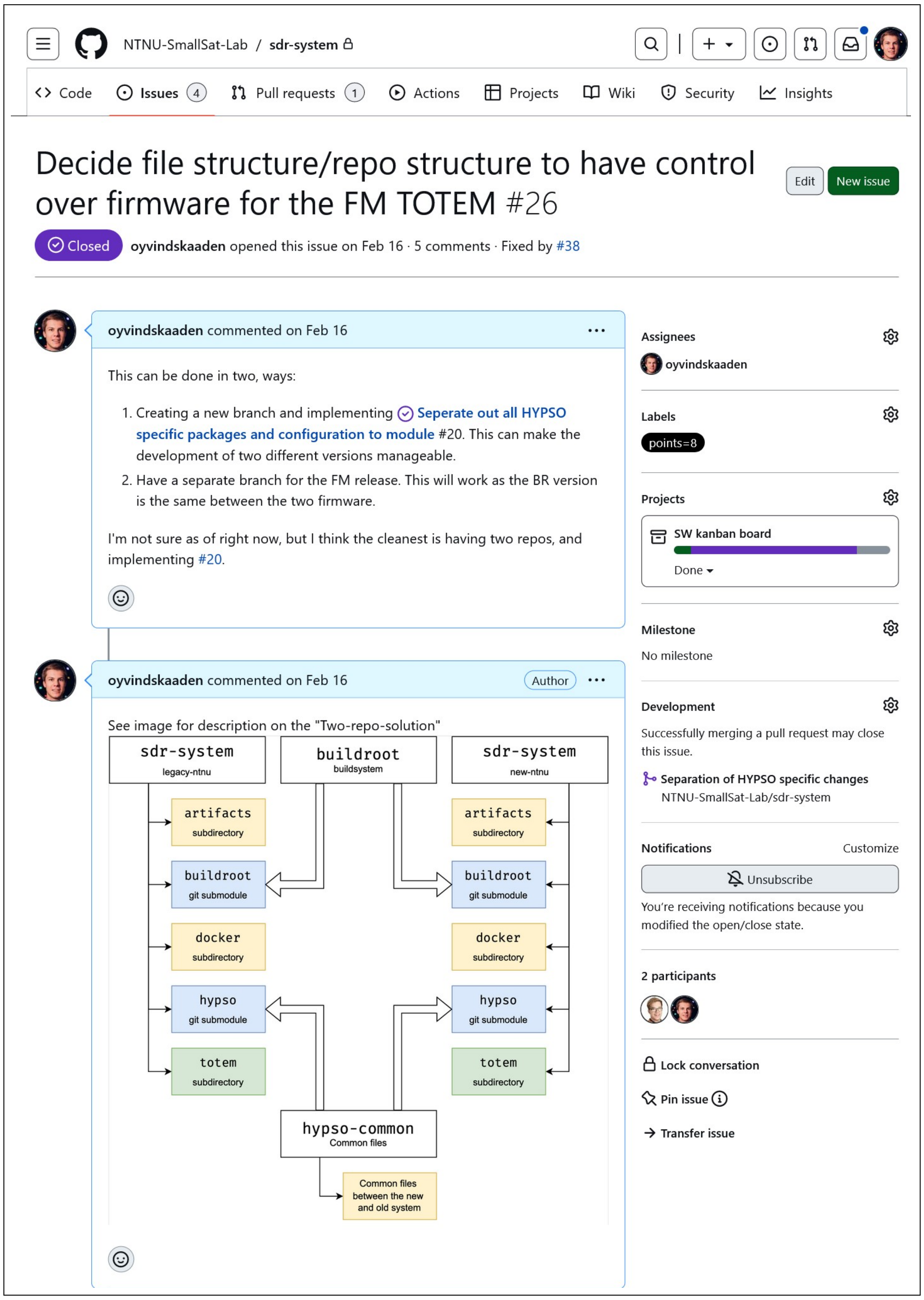
Development: Create a branch for this issue or link a pull request.

Notifications: Customize — Unsubscribe

You're receiving notifications because you were assigned.

3 participants

C.5 sdr-system#26



NTNU-SmallSat-Lab / sdr-system

Code · Issues 4 · Pull requests 1 · Actions · Projects · Wiki · Security · Insights

Decide file structure/repo structure to have control over firmware for the FM TOTEM #26

Edit · New issue

Closed · **oyvindskaaden** opened this issue on Feb 16 · 5 comments · Fixed by #38

oyvindskaaden commented on Feb 16

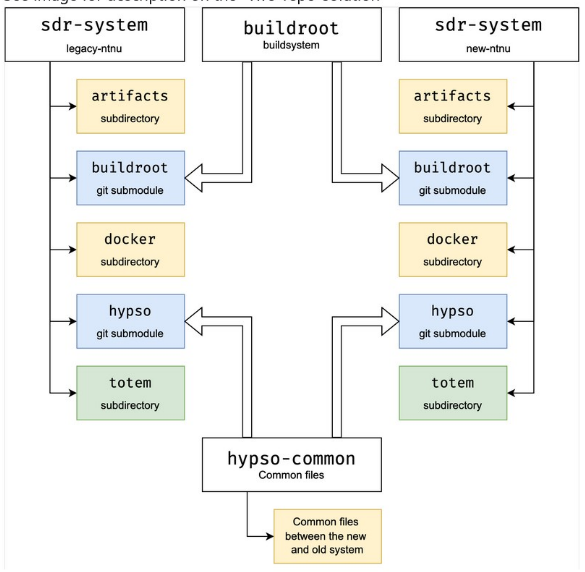
This can be done in two, ways:

1. Creating a new branch and implementing Seperate out all HYPSO specific packages and configuration to module #20. This can make the development of two different versions manageable.
2. Have a separate branch for the FM release. This will work as the BR version is the same between the two firmware.

I'm not sure as of right now, but I think the cleanest is having two repos, and implementing #20.

oyvindskaaden commented on Feb 16 · Author

See image for description on the "Two-repo-solution"



Assignees
oyvindskaaden

Labels
points=8

Projects
SW kanban board
Done

Milestone
No milestone

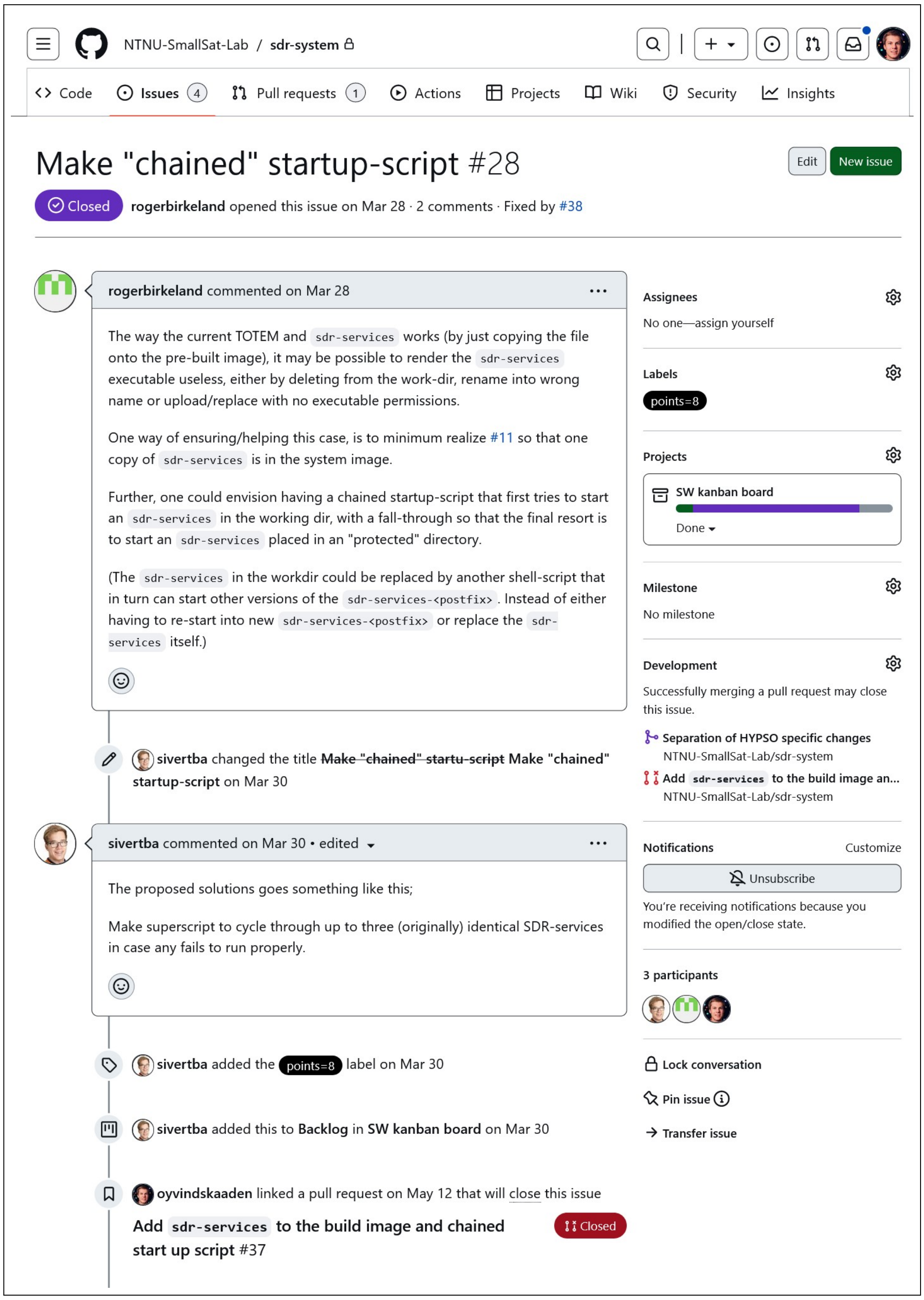
Development
Successfully merging a pull request may close this issue.
Separation of HYPSO specific changes
NTNU-SmallSat-Lab/sdr-system

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation
Pin issue
Transfer issue

C.6 `sdr-system#28`



Make "chained" startup-script #28

Closed **rogerbirkeland** opened this issue on Mar 28 · 2 comments · Fixed by #38

rogerbirkeland commented on Mar 28

The way the current TOTEM and `sdr-services` works (by just copying the file onto the pre-built image), it may be possible to render the `sdr-services` executable useless, either by deleting from the work-dir, rename into wrong name or upload/replace with no executable permissions.

One way of ensuring/helping this case, is to minimum realize #11 so that one copy of `sdr-services` is in the system image.

Further, one could envision having a chained startup-script that first tries to start an `sdr-services` in the working dir, with a fall-through so that the final resort is to start an `sdr-services` placed in an "protected" directory.

(The `sdr-services` in the workdir could be replaced by another shell-script that in turn can start other versions of the `sdr-services-<postfix>`. Instead of either having to re-start into new `sdr-services-<postfix>` or replace the `sdr-services` itself.)

sivertba changed the title ~~**Make "chained" startu-script**~~ **Make "chained" startup-script** on Mar 30

sivertba commented on Mar 30 • edited

The proposed solutions goes something like this;

Make superscript to cycle through up to three (originally) identical SDR-services in case any fails to run properly.

sivertba added the points=8 label on Mar 30

sivertba added this to **Backlog** in **SW kanban board** on Mar 30

oyvindskaaden linked a pull request on May 12 that will close this issue

Add `sdr-services` to the build image and chained start up script #37 Closed

Assignees
No one—assign yourself

Labels
points=8

Projects
SW kanban board
Done

Milestone
No milestone

Development
Successfully merging a pull request may close this issue.

Separation of HYPSO specific changes
NTNU-SmallSat-Lab/sdr-system

Add `sdr-services` to the build image an...
NTNU-SmallSat-Lab/sdr-system

Notifications Customize

Unsubscribe

You're receiving notifications because you modified the open/close state.

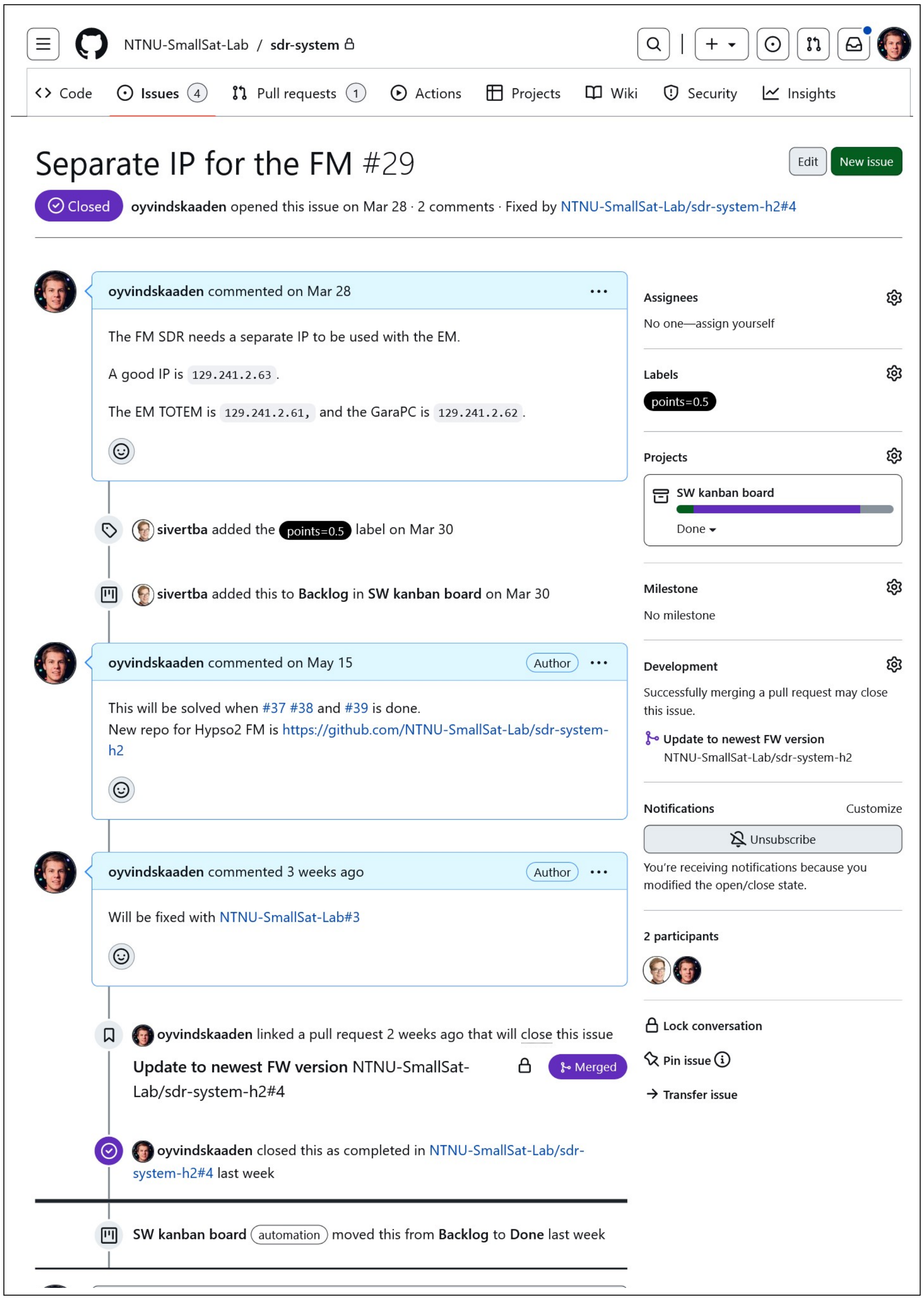
3 participants

Lock conversation

Pin issue

Transfer issue

C.7 sdr-system#29


NTNU-SmallSat-Lab / sdr-system

Code | Issues 4 | Pull requests 1 | Actions | Projects | Wiki | Security | Insights

Edit | New issue

Separate IP for the FM #29

Closed oyvindskaaden opened this issue on Mar 28 · 2 comments · Fixed by NTNU-SmallSat-Lab/sdr-system-h2#4

oyvindskaaden commented on Mar 28

The FM SDR needs a separate IP to be used with the EM.

A good IP is `129.241.2.63`.

The EM TOTEM is `129.241.2.61,` and the GaraPC is `129.241.2.62`.

sivertba added the points=0.5 label on Mar 30

sivertba added this to **Backlog** in **SW kanban board** on Mar 30

oyvindskaaden commented on May 15 (Author)

This will be solved when #37 #38 and #39 is done.
New repo for Hypso2 FM is https://github.com/NTNU-SmallSat-Lab/sdr-system-h2

oyvindskaaden commented 3 weeks ago (Author)

Will be fixed with NTNU-SmallSat-Lab#3

oyvindskaaden linked a pull request 2 weeks ago that will close this issue

Update to newest FW version NTNU-SmallSat-Lab/sdr-system-h2#4 Merged

oyvindskaaden closed this as completed in NTNU-SmallSat-Lab/sdr-system-h2#4 last week

SW kanban board (automation) moved this from **Backlog** to **Done** last week

Assignees
No one—assign yourself

Labels
points=0.5

Projects
SW kanban board
Done

Milestone
No milestone

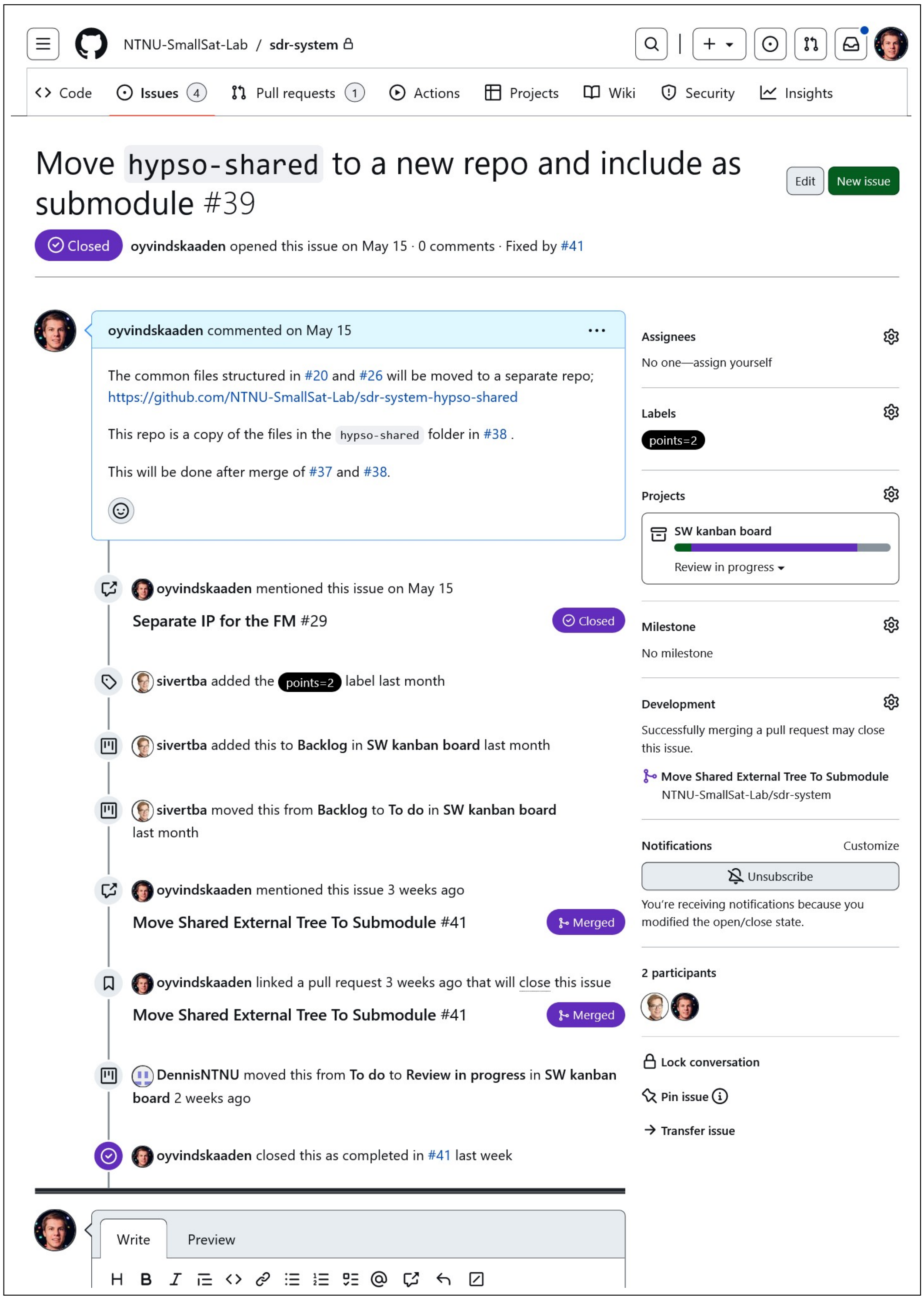
Development
Successfully merging a pull request may close this issue.
Update to newest FW version
NTNU-SmallSat-Lab/sdr-system-h2

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation
Pin issue
Transfer issue

C.8 sdr-system#39



NTNU-SmallSat-Lab / sdr-system

Code | Issues 4 | Pull requests 1 | Actions | Projects | Wiki | Security | Insights

Move `hypso-shared` to a new repo and include as submodule #39

Edit | New issue

Closed **oyvindskaaden** opened this issue on May 15 · 0 comments · Fixed by #41

oyvindskaaden commented on May 15

The common files structured in #20 and #26 will be moved to a separate repo; https://github.com/NTNU-SmallSat-Lab/sdr-system-hypso-shared

This repo is a copy of the files in the `hypso-shared` folder in #38 .

This will be done after merge of #37 and #38.

oyvindskaaden mentioned this issue on May 15

Separate IP for the FM #29 — Closed

sivertba added the points=2 label last month

sivertba added this to **Backlog** in **SW kanban board** last month

sivertba moved this from **Backlog** to **To do** in **SW kanban board** last month

oyvindskaaden mentioned this issue 3 weeks ago

Move Shared External Tree To Submodule #41 — Merged

oyvindskaaden linked a pull request 3 weeks ago that will close this issue

Move Shared External Tree To Submodule #41 — Merged

DennisNTNU moved this from **To do** to **Review in progress** in **SW kanban board** 2 weeks ago

oyvindskaaden closed this as completed in #41 last week

Write | Preview

Assignees
No one—assign yourself

Labels
points=2

Projects
SW kanban board
Review in progress

Milestone
No milestone

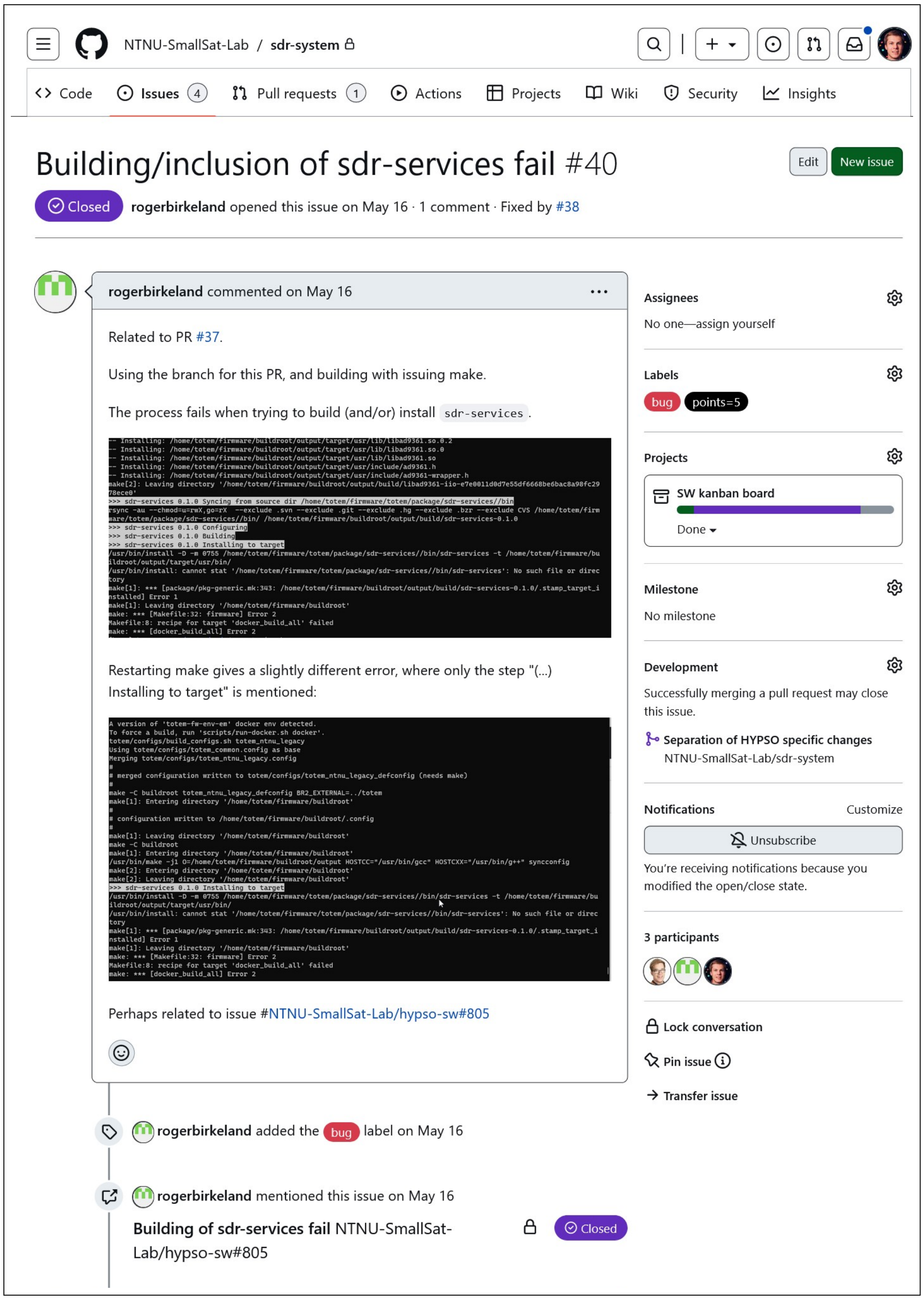
Development
Successfully merging a pull request may close this issue.
Move Shared External Tree To Submodule
NTNU-SmallSat-Lab/sdr-system

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation
Pin issue
Transfer issue

C.9 sdr-system#40



Building/inclusion of sdr-services fail #40

Closed rogerbirkeland opened this issue on May 16 · 1 comment · Fixed by #38

rogerbirkeland commented on May 16

Related to PR #37.

Using the branch for this PR, and building with issuing make.

The process fails when trying to build (and/or) install `sdr-services`.

```
-- Installing: /home/totem/firmware/buildroot/output/target/usr/lib/libad9361.so.0.2
-- Installing: /home/totem/firmware/buildroot/output/target/usr/lib/libad9361.so.0
-- Installing: /home/totem/firmware/buildroot/output/target/usr/lib/libad9361.so
-- Installing: /home/totem/firmware/buildroot/output/target/usr/include/ad9361.h
-- Installing: /home/totem/firmware/buildroot/output/target/usr/include/ad9361-wrapper.h
make[2]: Leaving directory '/home/totem/firmware/buildroot/output/build/libad9361-iio-e7e0011d0d7e55df6668be6bac8a98fc29
78ece0'
>>> sdr-services 0.1.0 Syncing from source dir /home/totem/firmware/totem/package/sdr-services//bin
rsync -au --chmod=u=rwX,go=rX  --exclude .svn --exclude .git --exclude .hg --exclude .bzr --exclude CVS /home/totem/firm
ware/totem/package/sdr-services//bin/ /home/totem/firmware/buildroot/output/build/sdr-services-0.1.0
>>> sdr-services 0.1.0 Configuring
>>> sdr-services 0.1.0 Building
>>> sdr-services 0.1.0 Installing to target
/usr/bin/install -D -m 0755 /home/totem/firmware/totem/package/sdr-services//bin/sdr-services -t /home/totem/firmware/bu
ildroot/output/target/usr/bin/
/usr/bin/install: cannot stat '/home/totem/firmware/totem/package/sdr-services//bin/sdr-services': No such file or direc
tory
make[1]: *** [package/pkg-generic.mk:343: /home/totem/firmware/buildroot/output/build/sdr-services-0.1.0/.stamp_target_i
nstalled] Error 1
make[1]: Leaving directory '/home/totem/firmware/buildroot'
make: *** [Makefile:32: firmware] Error 2
Makefile:8: recipe for target 'docker_build_all' failed
make: *** [docker_build_all] Error 2
```

Restarting make gives a slightly different error, where only the step "(...) Installing to target" is mentioned:

```
A version of 'totem-fw-env-em' docker env detected.
To force a build, run 'scripts/run-docker.sh docker'.
totem/configs/build_configs.sh totem_ntnu_legacy
Using totem/configs/totem_common.config as base
Merging totem/configs/totem_ntnu_legacy.config
#
# merged configuration written to totem/configs/totem_ntnu_legacy_defconfig (needs make)
#
make -C buildroot totem_ntnu_legacy_defconfig BR2_EXTERNAL=../totem
make[1]: Entering directory '/home/totem/firmware/buildroot'
#
# configuration written to /home/totem/firmware/buildroot/.config
#
make[1]: Leaving directory '/home/totem/firmware/buildroot'
make -C buildroot
make[1]: Entering directory '/home/totem/firmware/buildroot'
/usr/bin/make -j1 O=/home/totem/firmware/buildroot/output HOSTCC="/usr/bin/gcc" HOSTCXX="/usr/bin/g++" syncconfig
make[2]: Entering directory '/home/totem/firmware/buildroot'
make[2]: Leaving directory '/home/totem/firmware/buildroot'
>>> sdr-services 0.1.0 Installing to target
/usr/bin/install -D -m 0755 /home/totem/firmware/totem/package/sdr-services//bin/sdr-services -t /home/totem/firmware/bu
ildroot/output/target/usr/bin/
/usr/bin/install: cannot stat '/home/totem/firmware/totem/package/sdr-services//bin/sdr-services': No such file or direc
tory
make[1]: *** [package/pkg-generic.mk:343: /home/totem/firmware/buildroot/output/build/sdr-services-0.1.0/.stamp_target_i
nstalled] Error 1
make[1]: Leaving directory '/home/totem/firmware/buildroot'
make: *** [Makefile:32: firmware] Error 2
Makefile:8: recipe for target 'docker_build_all' failed
make: *** [docker_build_all] Error 2
```

Perhaps related to issue #NTNU-SmallSat-Lab/hypso-sw#805

rogerbirkeland added the bug label on May 16

rogerbirkeland mentioned this issue on May 16

Building of sdr-services fail NTNU-SmallSat-Lab/hypso-sw#805 Closed

Assignees
No one—assign yourself

Labels
bug points=5

Projects
SW kanban board
Done

Milestone
No milestone

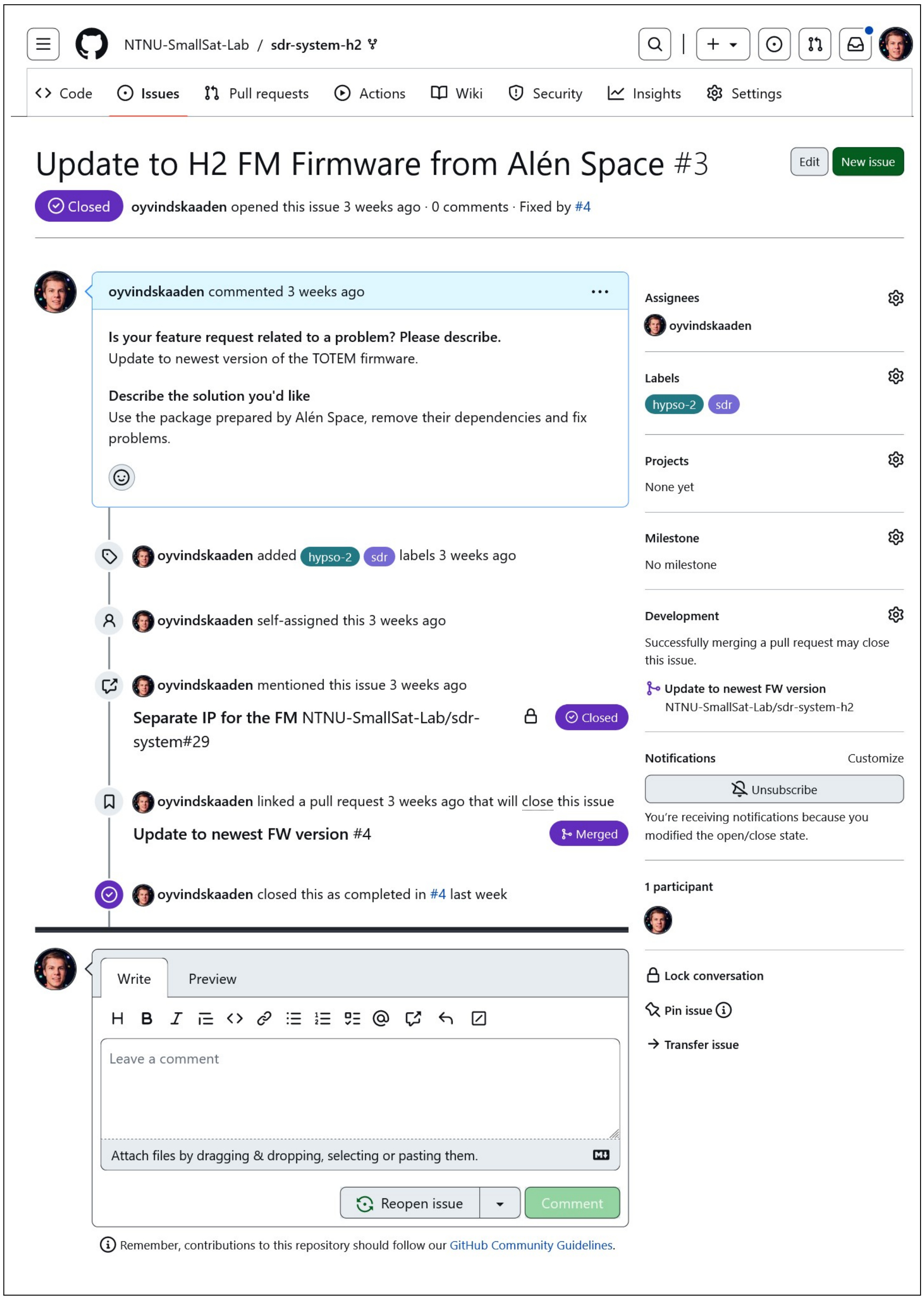
Development
Successfully merging a pull request may close this issue.
Separation of HYPSO specific changes
NTNU-SmallSat-Lab/sdr-system

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

3 participants

Lock conversation
Pin issue
Transfer issue

C.10 sdr-system-h2#3



Update to H2 FM Firmware from Alén Space #3

Closed oyvindskaaden opened this issue 3 weeks ago · 0 comments · Fixed by #4

oyvindskaaden commented 3 weeks ago

Is your feature request related to a problem? Please describe.
Update to newest version of the TOTEM firmware.

Describe the solution you'd like
Use the package prepared by Alén Space, remove their dependencies and fix problems.

oyvindskaaden added hypso-2 sdr labels 3 weeks ago

oyvindskaaden self-assigned this 3 weeks ago

oyvindskaaden mentioned this issue 3 weeks ago

Separate IP for the FM NTNU-SmallSat-Lab/sdr-system#29 Closed

oyvindskaaden linked a pull request 3 weeks ago that will close this issue

Update to newest FW version #4 Merged

oyvindskaaden closed this as completed in #4 last week

Assignees: oyvindskaaden

Labels: hypso-2 sdr

Projects: None yet

Milestone: No milestone

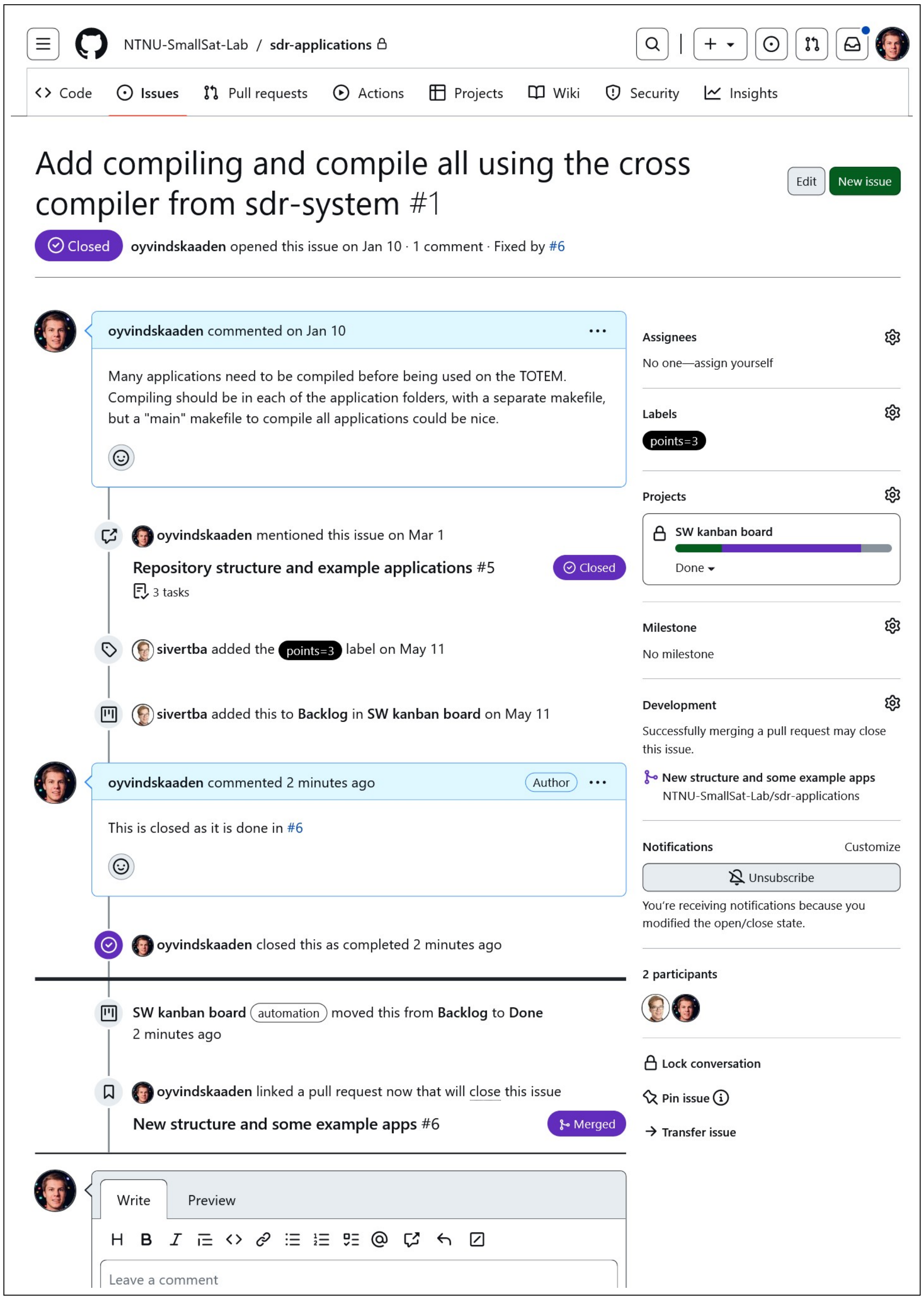
Development: Successfully merging a pull request may close this issue.
Update to newest FW version
NTNU-SmallSat-Lab/sdr-system-h2

Notifications: Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

1 participant

Remember, contributions to this repository should follow our GitHub Community Guidelines.

C.11 sdr-applications#1



NTNU-SmallSat-Lab / sdr-applications

Code Issues Pull requests Actions Projects Wiki Security Insights

Add compiling and compile all using the cross compiler from sdr-system #1

Edit New issue

Closed **oyvindskaaden** opened this issue on Jan 10 · 1 comment · Fixed by #6

oyvindskaaden commented on Jan 10

Many applications need to be compiled before being used on the TOTEM. Compiling should be in each of the application folders, with a separate makefile, but a "main" makefile to compile all applications could be nice.

oyvindskaaden mentioned this issue on Mar 1

Repository structure and example applications #5 Closed
3 tasks

sivertba added the points=3 label on May 11

sivertba added this to **Backlog** in **SW kanban board** on May 11

oyvindskaaden commented 2 minutes ago Author

This is closed as it is done in #6

oyvindskaaden closed this as completed 2 minutes ago

SW kanban board automation moved this from **Backlog** to **Done**
2 minutes ago

oyvindskaaden linked a pull request now that will close this issue

New structure and some example apps #6 Merged

Write Preview

Leave a comment

Assignees
No one—assign yourself

Labels
points=3

Projects
SW kanban board
Done

Milestone
No milestone

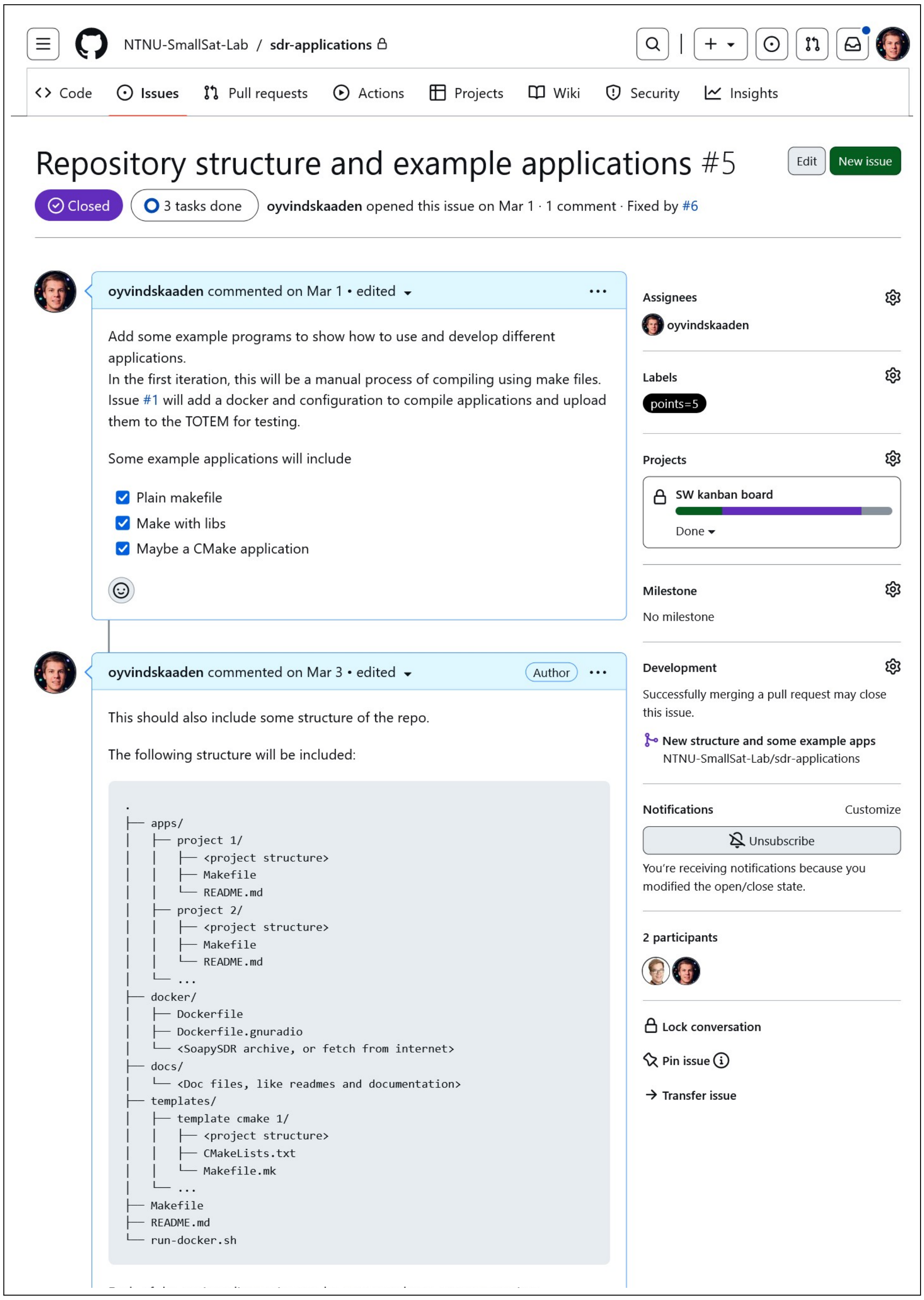
Development
Successfully merging a pull request may close this issue.
New structure and some example apps
NTNU-SmallSat-Lab/sdr-applications

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation
Pin issue
Transfer issue

C.12 `sdr-applications#5`



Repository structure and example applications #5

Closed · 3 tasks done · oyvindskaaden opened this issue on Mar 1 · 1 comment · Fixed by #6

oyvindskaaden commented on Mar 1 • edited

Add some example programs to show how to use and develop different applications.
In the first iteration, this will be a manual process of compiling using make files. Issue #1 will add a docker and configuration to compile applications and upload them to the TOTEM for testing.

Some example applications will include

- [x] Plain makefile
- [x] Make with libs
- [x] Maybe a CMake application

oyvindskaaden commented on Mar 3 • edited · Author

This should also include some structure of the repo.

The following structure will be included:

```
.
├── apps/
│   ├── project 1/
│   │   ├── <project structure>
│   │   ├── Makefile
│   │   └── README.md
│   ├── project 2/
│   │   ├── <project structure>
│   │   ├── Makefile
│   │   └── README.md
│   └── ...
├── docker/
│   ├── Dockerfile
│   ├── Dockerfile.gnuradio
│   └── <SoapySDR archive, or fetch from internet>
├── docs/
│   └── <Doc files, like readmes and documentation>
├── templates/
│   ├── template cmake 1/
│   │   ├── <project structure>
│   │   ├── CMakeLists.txt
│   │   └── Makefile.mk
│   └── ...
├── Makefile
├── README.md
└── run-docker.sh
```

Assignees: oyvindskaaden

Labels: points=5

Projects: SW kanban board — Done

Milestone: No milestone

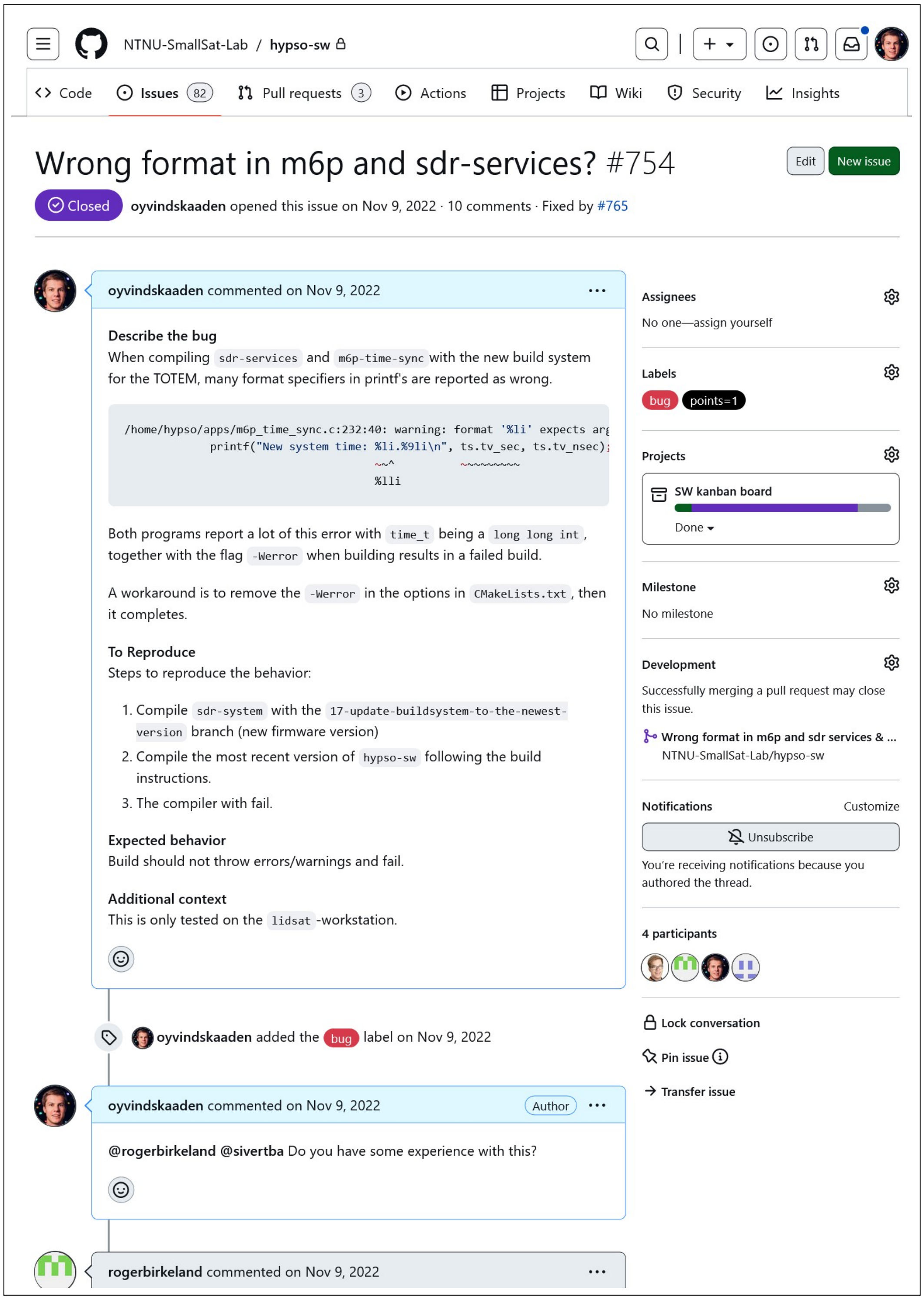
Development: Successfully merging a pull request may close this issue.
New structure and some example apps
NTNU-SmallSat-Lab/sdr-applications

Notifications — Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation
Pin issue
Transfer issue

C.13 hypso-sw#754



Wrong format in m6p and sdr-services? #754

Closed **oyvindskaaden** opened this issue on Nov 9, 2022 · 10 comments · Fixed by #765

oyvindskaaden commented on Nov 9, 2022

Describe the bug
When compiling `sdr-services` and `m6p-time-sync` with the new build system for the TOTEM, many format specifiers in printf's are reported as wrong.

```
/home/hypso/apps/m6p_time_sync.c:232:40: warning: format '%li' expects arg
          printf("New system time: %li.%9li\n", ts.tv_sec, ts.tv_nsec);
                                   ~~^          ~~~~~~~~~
                                   %lli
```

Both programs report a lot of this error with `time_t` being a `long long int`, together with the flag `-Werror` when building results in a failed build.

A workaround is to remove the `-Werror` in the options in `CMakeLists.txt`, then it completes.

To Reproduce
Steps to reproduce the behavior:

1. Compile `sdr-system` with the `17-update-buildsystem-to-the-newest-version` branch (new firmware version)
2. Compile the most recent version of `hypso-sw` following the build instructions.
3. The compiler with fail.

Expected behavior
Build should not throw errors/warnings and fail.

Additional context
This is only tested on the `lidsat`-workstation.

oyvindskaaden added the bug label on Nov 9, 2022

oyvindskaaden commented on Nov 9, 2022 Author

@rogerbirkeland **@sivertba** Do you have some experience with this?

rogerbirkeland commented on Nov 9, 2022

Assignees
No one—assign yourself

Labels
bug points=1

Projects
SW kanban board
Done

Milestone
No milestone

Development
Successfully merging a pull request may close this issue.

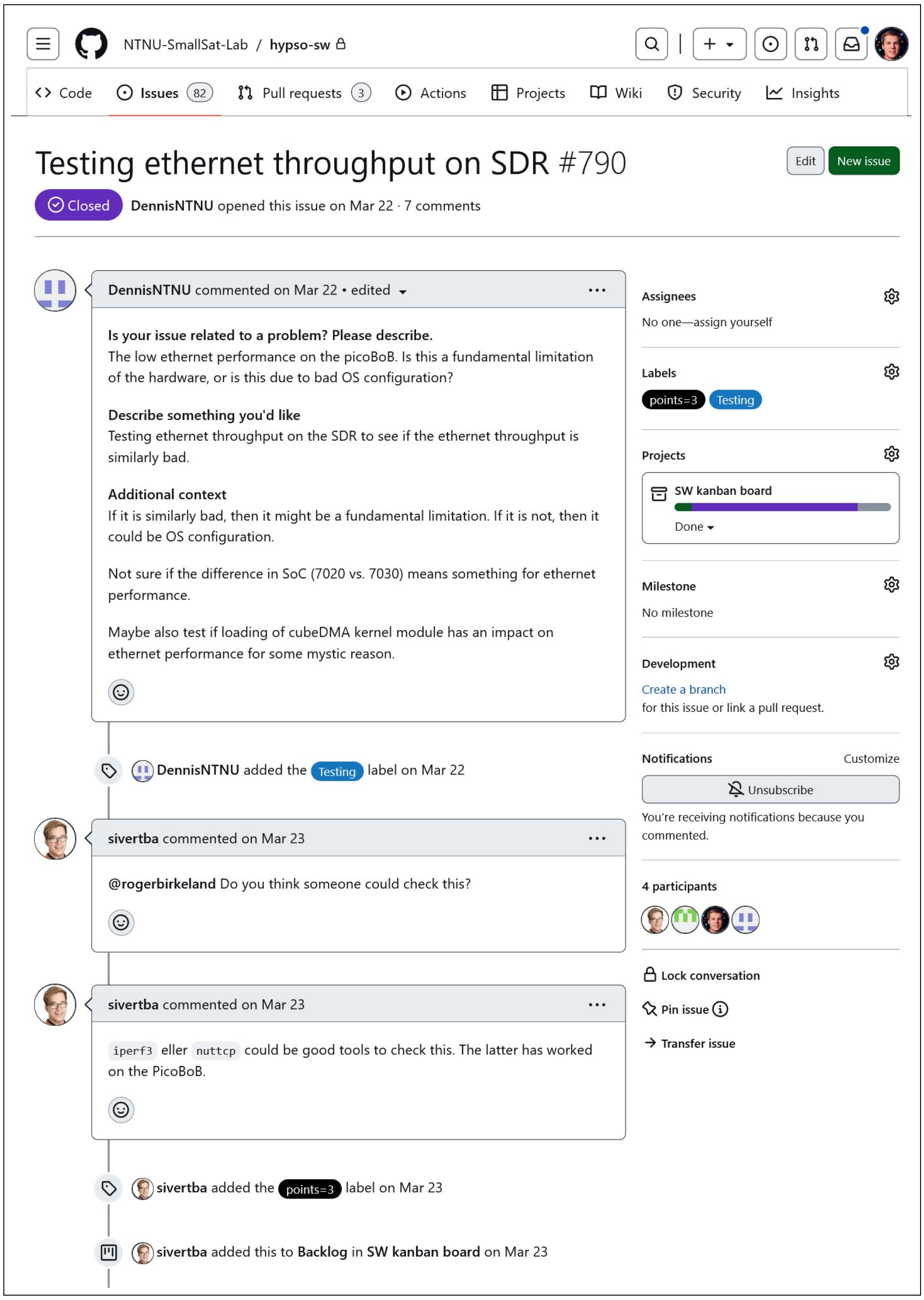
Wrong format in m6p and sdr services & ...
NTNU-SmallSat-Lab/hypso-sw

Notifications Customize
Unsubscribe
You're receiving notifications because you authored the thread.

4 participants

Lock conversation
Pin issue
Transfer issue

C.14 `hypso-sw#790`



Testing ethernet throughput on SDR #790

Closed **DennisNTNU** opened this issue on Mar 22 · 7 comments

DennisNTNU commented on Mar 22 • edited

Is your issue related to a problem? Please describe.
The low ethernet performance on the picoBoB. Is this a fundamental limitation of the hardware, or is this due to bad OS configuration?

Describe something you'd like
Testing ethernet throughput on the SDR to see if the ethernet throughput is similarly bad.

Additional context
If it is similarly bad, then it might be a fundamental limitation. If it is not, then it could be OS configuration.

Not sure if the difference in SoC (7020 vs. 7030) means something for ethernet performance.

Maybe also test if loading of cubeDMA kernel module has an impact on ethernet performance for some mystic reason.

DennisNTNU added the Testing label on Mar 22

sivertba commented on Mar 23

@rogerbirkeland Do you think someone could check this?

sivertba commented on Mar 23

`iperf3` eller `nuttcp` could be good tools to check this. The latter has worked on the PicoBoB.

sivertba added the points=3 label on Mar 23

sivertba added this to **Backlog** in **SW kanban board** on Mar 23

Assignees
No one—assign yourself

Labels
points=3 Testing

Projects
SW kanban board
Done

Milestone
No milestone

Development
Create a branch
for this issue or link a pull request.

Notifications Customize
Unsubscribe
You're receiving notifications because you commented.

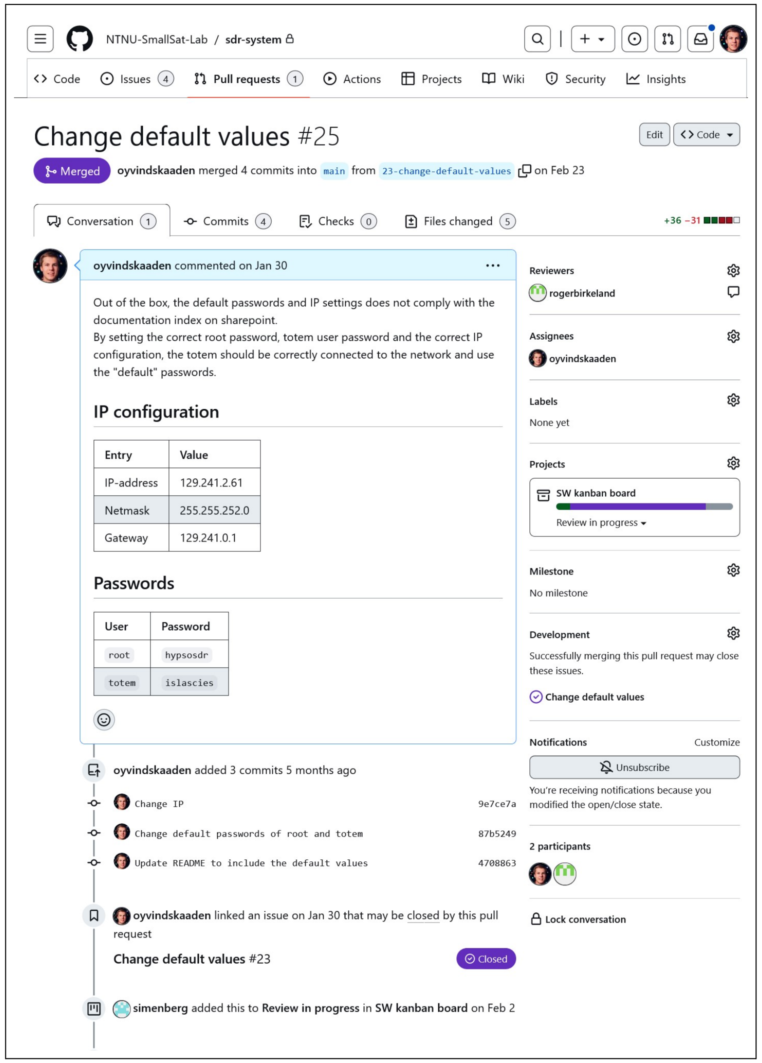
4 participants

Lock conversation
Pin issue
Transfer issue

Appendix D

Pull-Requests

D.1 `sdr-system#25`



Change default values #25

oyvindskaaden merged 4 commits into `main` from `23-change-default-values` on Feb 23

oyvindskaaden commented on Jan 30

Out of the box, the default passwords and IP settings does not comply with the documentation index on sharepoint.
By setting the correct root password, totem user password and the correct IP configuration, the totem should be correctly connected to the network and use the "default" passwords.

IP configuration

Entry	Value
IP-address	129.241.2.61
Netmask	255.255.252.0
Gateway	129.241.0.1

Passwords

User	Password
`root`	`hypsosdr`
`totem`	`islascies`

oyvindskaaden added 3 commits 5 months ago

```
Change IP                                        9e7ce7a
Change default passwords of root and totem       87b5249
Update README to include the default values      4708863
```

oyvindskaaden linked an issue on Jan 30 that may be closed by this pull request

Change default values #23 — Closed

simenberg added this to **Review in progress** in **SW kanban board** on Feb 2

Reviewers: rogerbirkeland

Assignees: oyvindskaaden

Labels: None yet

Projects: SW kanban board — Review in progress

Milestone: No milestone

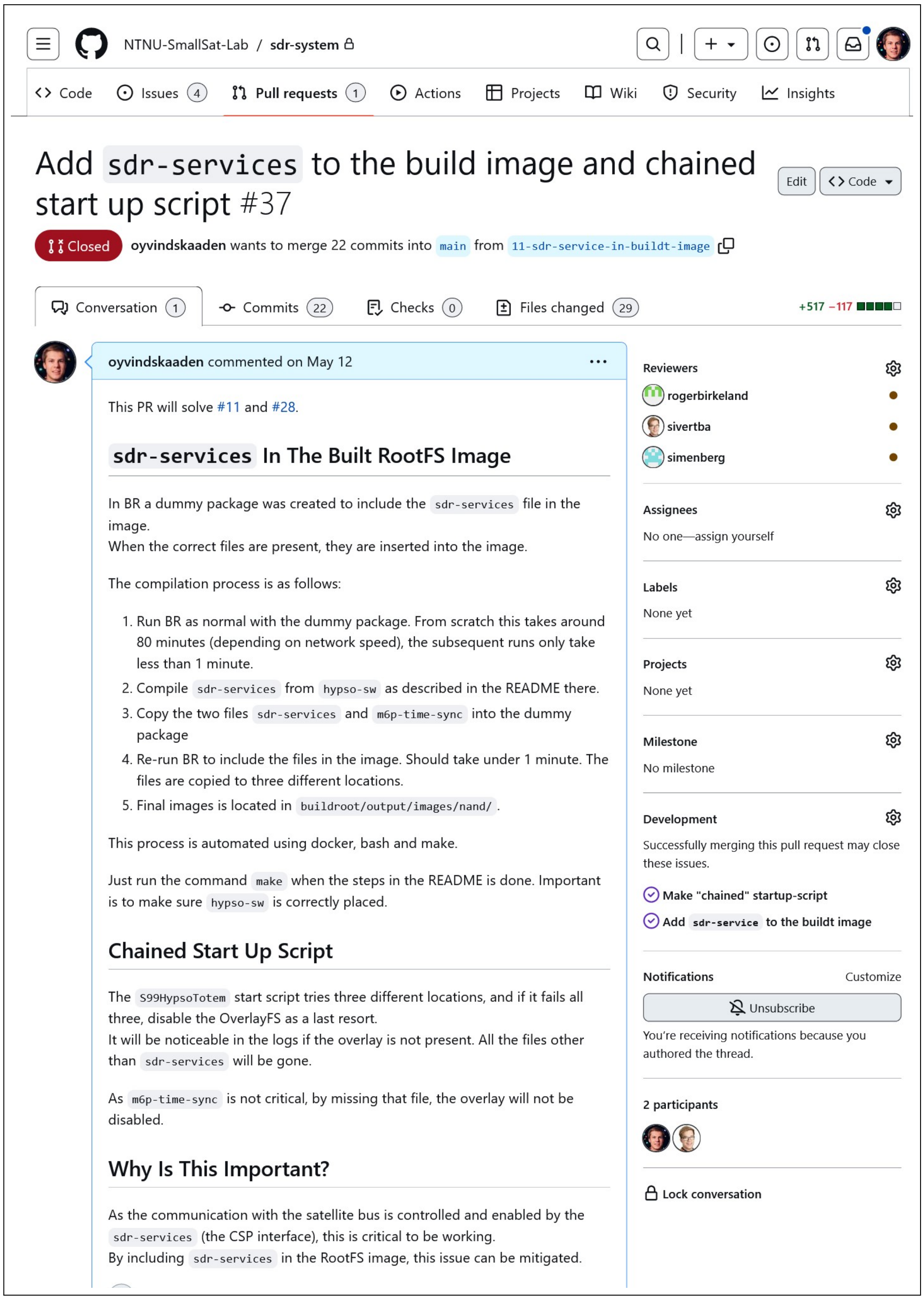
Development: Successfully merging this pull request may close these issues. Change default values

Notifications — Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.2 `sdr-system#37`



Add `sdr-services` to the build image and chained start up script #37

Closed oyvindskaaden wants to merge 22 commits into `main` from `11-sdr-service-in-buildt-image`

oyvindskaaden commented on May 12

This PR will solve #11 and #28.

`sdr-services` In The Built RootFS Image

In BR a dummy package was created to include the `sdr-services` file in the image.
When the correct files are present, they are inserted into the image.

The compilation process is as follows:

1. Run BR as normal with the dummy package. From scratch this takes around 80 minutes (depending on network speed), the subsequent runs only take less than 1 minute.
2. Compile `sdr-services` from `hypso-sw` as described in the README there.
3. Copy the two files `sdr-services` and `m6p-time-sync` into the dummy package
4. Re-run BR to include the files in the image. Should take under 1 minute. The files are copied to three different locations.
5. Final images is located in `buildroot/output/images/nand/`.

This process is automated using docker, bash and make.

Just run the command `make` when the steps in the README is done. Important is to make sure `hypso-sw` is correctly placed.

Chained Start Up Script

The `S99HypsoTotem` start script tries three different locations, and if it fails all three, disable the OverlayFS as a last resort.
It will be noticeable in the logs if the overlay is not present. All the files other than `sdr-services` will be gone.

As `m6p-time-sync` is not critical, by missing that file, the overlay will not be disabled.

Why Is This Important?

As the communication with the satellite bus is controlled and enabled by the `sdr-services` (the CSP interface), this is critical to be working.
By including `sdr-services` in the RootFS image, this issue can be mitigated.

Reviewers
- rogerbirkeland ●
- sivertba ●
- simenberg ●

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

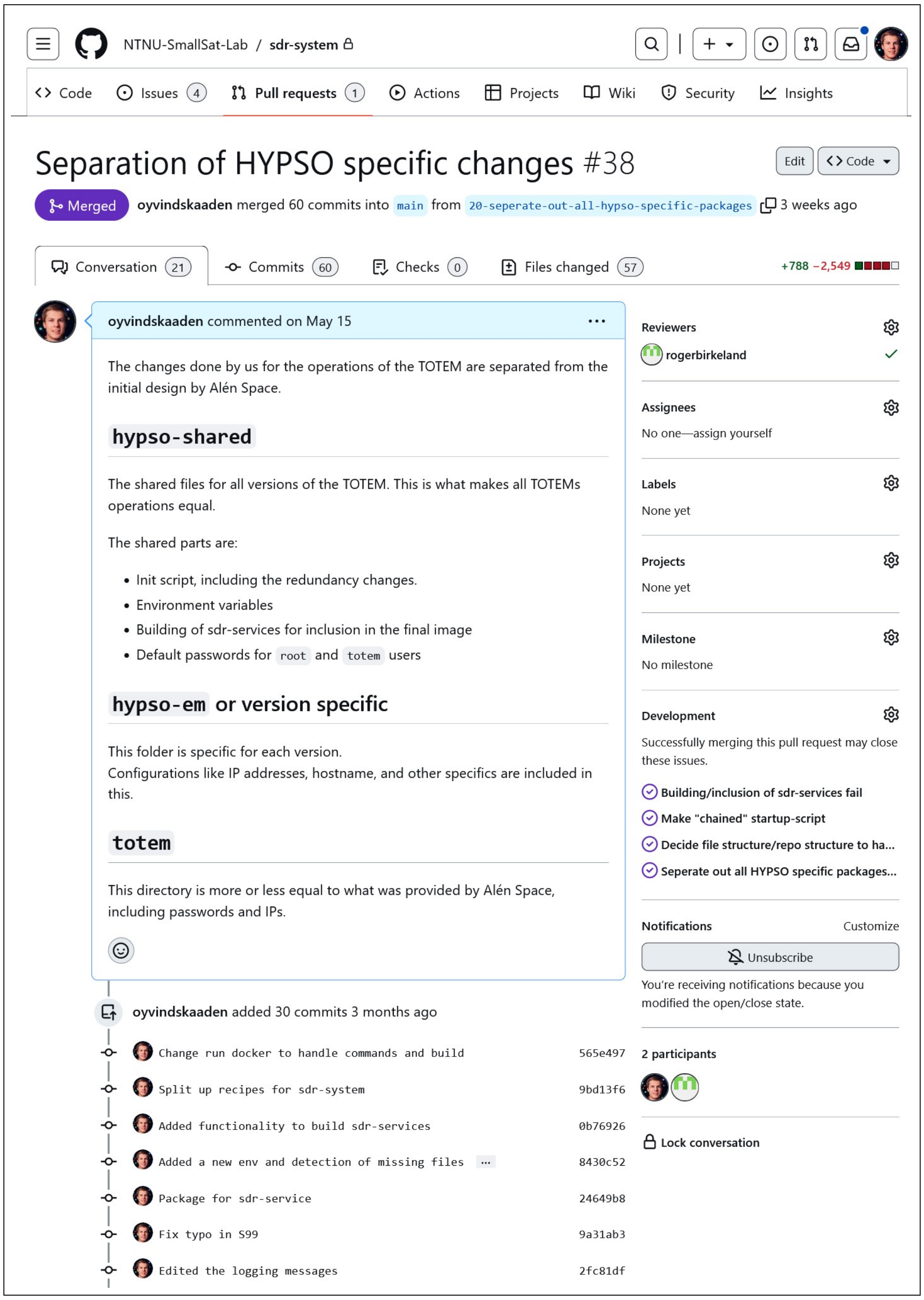
Development
Successfully merging this pull request may close these issues.
- Make "chained" startup-script
- Add `sdr-service` to the buildt image

Notifications Customize
Unsubscribe
You're receiving notifications because you authored the thread.

2 participants

Lock conversation

D.3 sdr-system#38



Separation of HYPSO specific changes #38

Merged oyvindskaaden merged 60 commits into `main` from `20-seperate-out-all-hypso-specific-packages` 3 weeks ago

oyvindskaaden commented on May 15

The changes done by us for the operations of the TOTEM are separated from the initial design by Alén Space.

`hypso-shared`

The shared files for all versions of the TOTEM. This is what makes all TOTEMs operations equal.

The shared parts are:

- Init script, including the redundancy changes.
- Environment variables
- Building of sdr-services for inclusion in the final image
- Default passwords for `root` and `totem` users

`hypso-em` or version specific

This folder is specific for each version.
Configurations like IP addresses, hostname, and other specifics are included in this.

`totem`

This directory is more or less equal to what was provided by Alén Space, including passwords and IPs.

oyvindskaaden added 30 commits 3 months ago

```
Change run docker to handle commands and build          565e497
Split up recipes for sdr-system                         9bd13f6
Added functionality to build sdr-services               0b76926
Added a new env and detection of missing files  …       8430c52
Package for sdr-service                                 24649b8
Fix typo in S99                                         9a31ab3
Edited the logging messages                             2fc81df
```

Reviewers
rogerbirkeland ✓

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

- Building/inclusion of sdr-services fail
- Make "chained" startup-script
- Decide file structure/repo structure to ha...
- Seperate out all HYPSO specific packages...

Notifications Customize

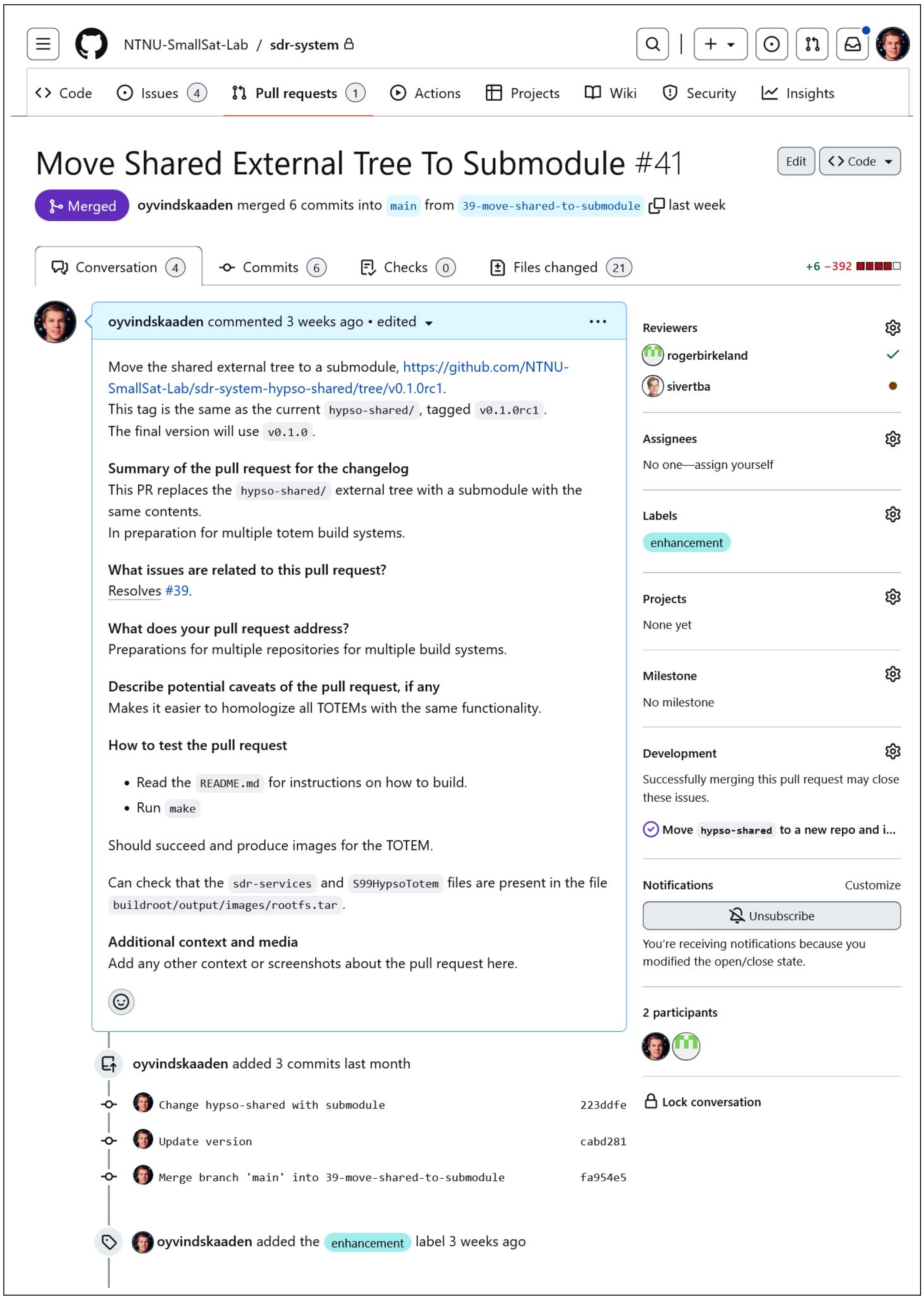
Unsubscribe

You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.4 sdr-system#41



Move Shared External Tree To Submodule #41

Merged oyvindskaaden merged 6 commits into `main` from `39-move-shared-to-submodule` last week

oyvindskaaden commented 3 weeks ago • edited

Move the shared external tree to a submodule, https://github.com/NTNU-SmallSat-Lab/sdr-system-hypso-shared/tree/v0.1.0rc1.
This tag is the same as the current `hypso-shared/`, tagged `v0.1.0rc1`.
The final version will use `v0.1.0`.

Summary of the pull request for the changelog
This PR replaces the `hypso-shared/` external tree with a submodule with the same contents.
In preparation for multiple totem build systems.

What issues are related to this pull request?
Resolves #39.

What does your pull request address?
Preparations for multiple repositories for multiple build systems.

Describe potential caveats of the pull request, if any
Makes it easier to homologize all TOTEMs with the same functionality.

How to test the pull request

- Read the `README.md` for instructions on how to build.
- Run `make`

Should succeed and produce images for the TOTEM.

Can check that the `sdr-services` and `S99HypsoTotem` files are present in the file `buildroot/output/images/rootfs.tar`.

Additional context and media
Add any other context or screenshots about the pull request here.

oyvindskaaden added 3 commits last month

```
Change hypso-shared with submodule                    223ddfe
Update version                                        cabd281
Merge branch 'main' into 39-move-shared-to-submodule  fa954e5
```

oyvindskaaden added the enhancement label 3 weeks ago

Reviewers
rogerbirkeland ✓
sivertba ●

Assignees
No one—assign yourself

Labels
enhancement

Projects
None yet

Milestone
No milestone

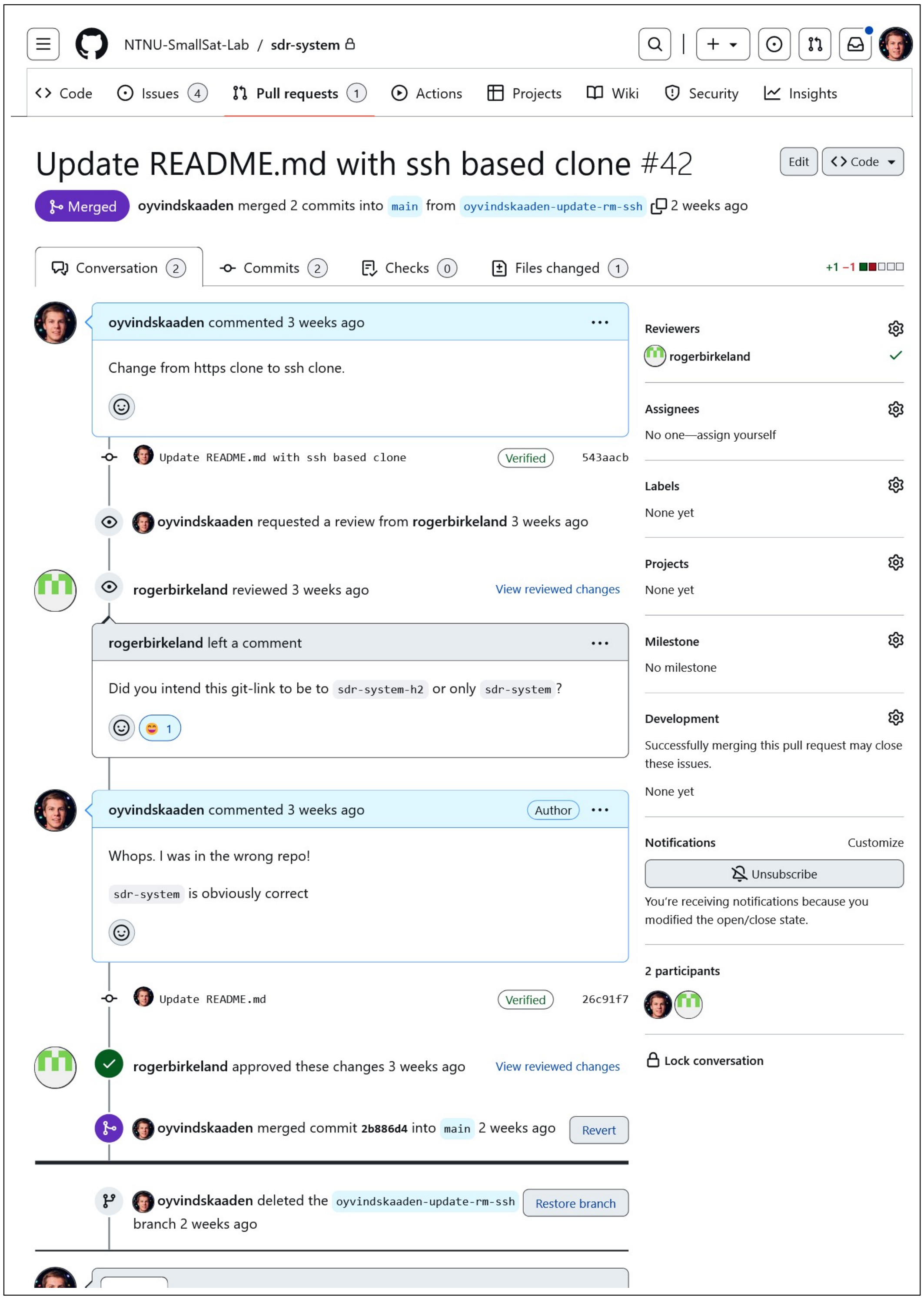
Development
Successfully merging this pull request may close these issues.
Move `hypso-shared` to a new repo and i...

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.5 sdr-system#42



NTNU-SmallSat-Lab / sdr-system

Code · Issues 4 · Pull requests 1 · Actions · Projects · Wiki · Security · Insights

Update README.md with ssh based clone #42

Edit · Code

Merged oyvindskaaden merged 2 commits into `main` from `oyvindskaaden-update-rm-ssh` 2 weeks ago

Conversation 2 · Commits 2 · Checks 0 · Files changed 1 · +1 −1

oyvindskaaden commented 3 weeks ago

Change from https clone to ssh clone.

`Update README.md with ssh based clone` Verified 543aacb

oyvindskaaden requested a review from **rogerbirkeland** 3 weeks ago

rogerbirkeland reviewed 3 weeks ago — View reviewed changes

rogerbirkeland left a comment

Did you intend this git-link to be to `sdr-system-h2` or only `sdr-system`?

😄 1

oyvindskaaden commented 3 weeks ago — Author

Whops. I was in the wrong repo!

`sdr-system` is obviously correct

`Update README.md` Verified 26c91f7

rogerbirkeland approved these changes 3 weeks ago — View reviewed changes

oyvindskaaden merged commit **2b886d4** into `main` 2 weeks ago — Revert

oyvindskaaden deleted the `oyvindskaaden-update-rm-ssh` branch 2 weeks ago — Restore branch

Reviewers: rogerbirkeland ✓

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

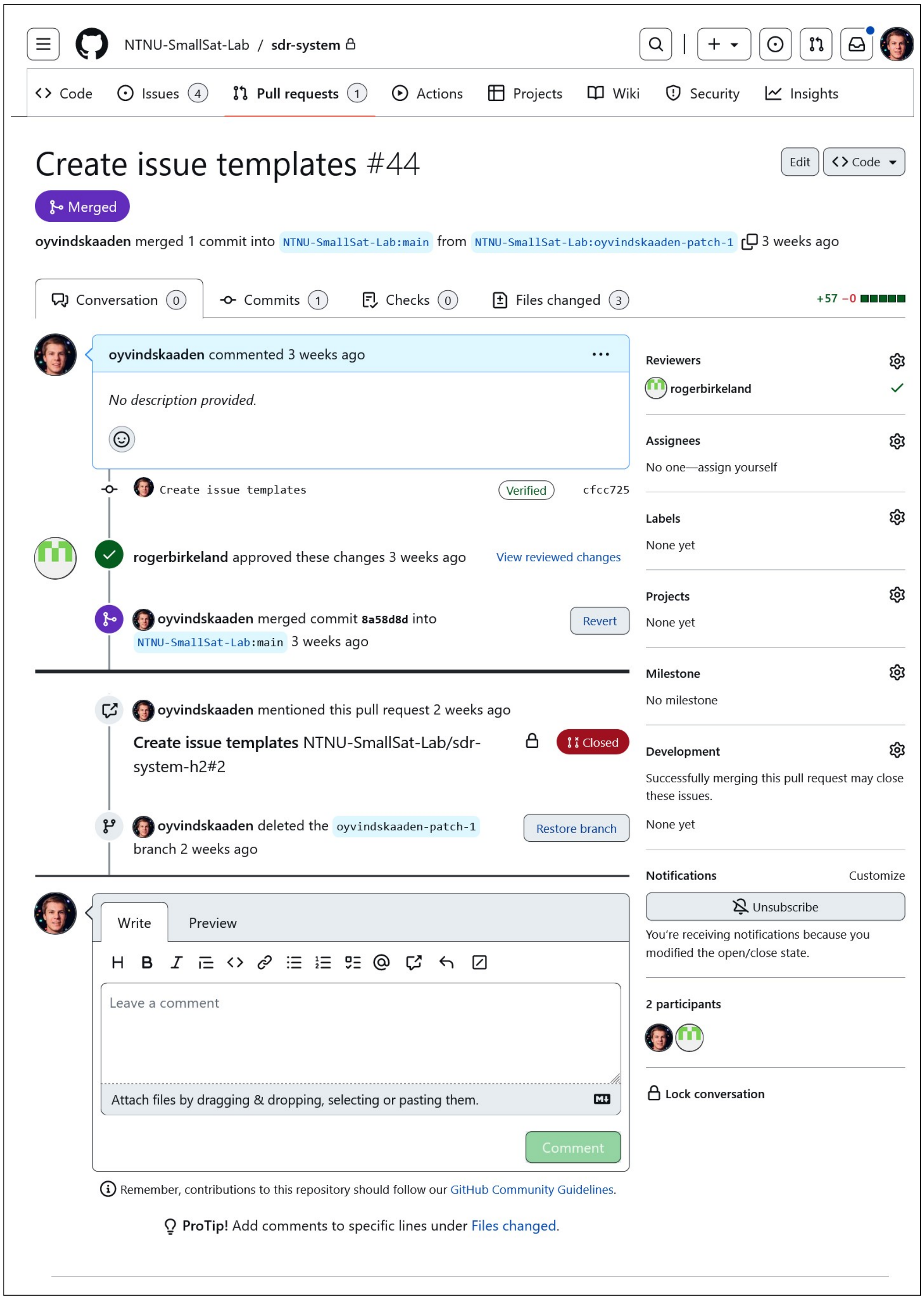
Development: Successfully merging this pull request may close these issues. None yet

Notifications: Customize — Unsubscribe — You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.6 `sdr-system#44`



NTNU-SmallSat-Lab / sdr-system

Create issue templates #44

Merged

oyvindskaaden merged 1 commit into `NTNU-SmallSat-Lab:main` from `NTNU-SmallSat-Lab:oyvindskaaden-patch-1` 3 weeks ago

oyvindskaaden commented 3 weeks ago

No description provided.

`Create issue templates` Verified `cfcc725`

rogerbirkeland approved these changes 3 weeks ago View reviewed changes

oyvindskaaden merged commit `8a58d8d` into `NTNU-SmallSat-Lab:main` 3 weeks ago Revert

oyvindskaaden mentioned this pull request 2 weeks ago

Create issue templates NTNU-SmallSat-Lab/sdr-system-h2#2 Closed

oyvindskaaden deleted the `oyvindskaaden-patch-1` branch 2 weeks ago Restore branch

Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

ProTip! Add comments to specific lines under Files changed.

Reviewers
rogerbirkeland ✓

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

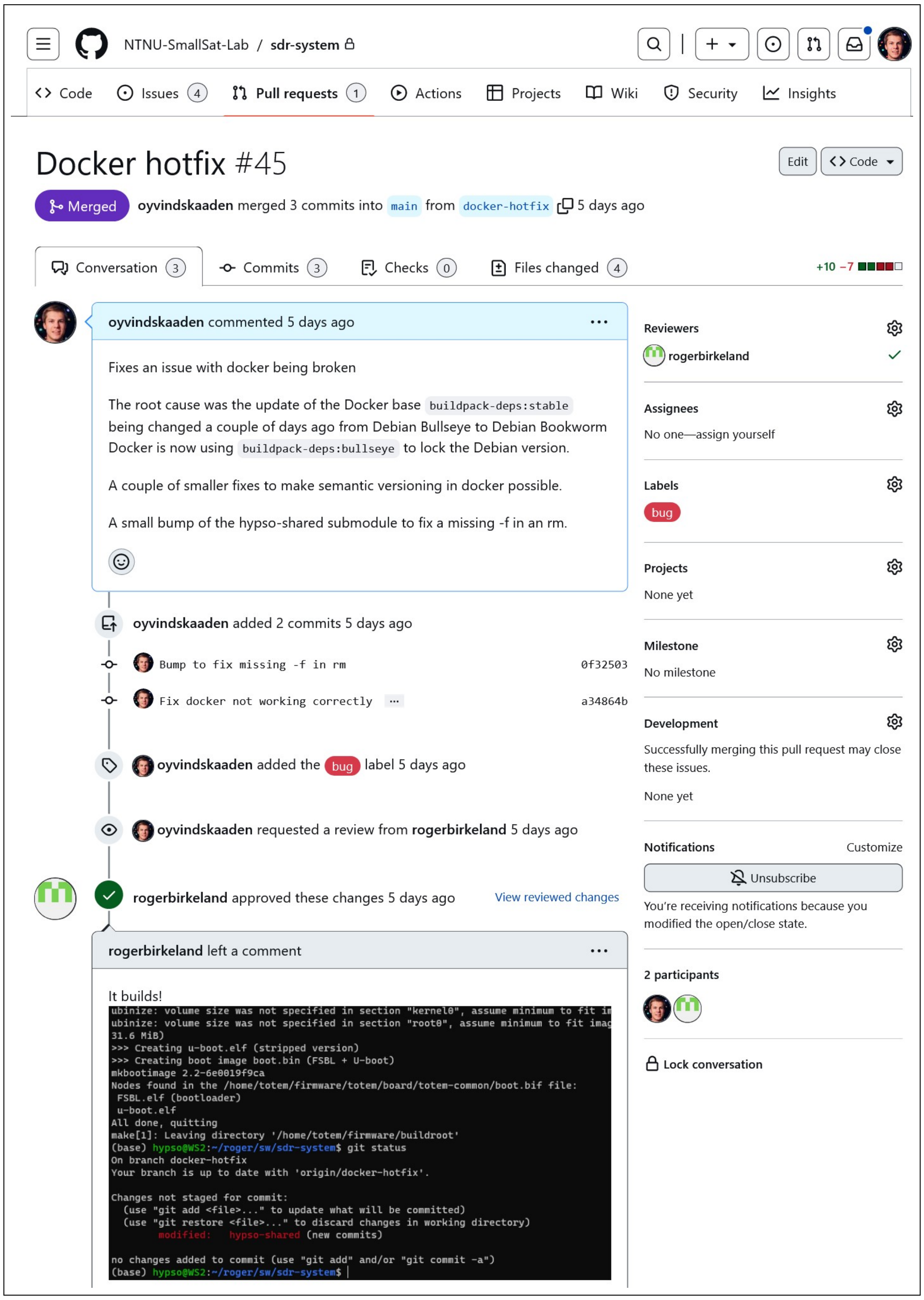
Development
Successfully merging this pull request may close these issues.
None yet

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.7 sdr-system#45



Docker hotfix #45

Merged oyvindskaaden merged 3 commits into `main` from `docker-hotfix` 5 days ago

oyvindskaaden commented 5 days ago

Fixes an issue with docker being broken

The root cause was the update of the Docker base `buildpack-deps:stable` being changed a couple of days ago from Debian Bullseye to Debian Bookworm Docker is now using `buildpack-deps:bullseye` to lock the Debian version.

A couple of smaller fixes to make semantic versioning in docker possible.

A small bump of the hypso-shared submodule to fix a missing -f in an rm.

oyvindskaaden added 2 commits 5 days ago

```
Bump to fix missing -f in rm                    0f32503
Fix docker not working correctly  …             a34864b
```

oyvindskaaden added the bug label 5 days ago

oyvindskaaden requested a review from **rogerbirkeland** 5 days ago

rogerbirkeland approved these changes 5 days ago — View reviewed changes

rogerbirkeland left a comment

It builds!

```
ubinize: volume size was not specified in section "kernel0", assume minimum to fit im
ubinize: volume size was not specified in section "root0", assume minimum to fit imag
31.6 MiB)
>>> Creating u-boot.elf (stripped version)
>>> Creating boot image boot.bin (FSBL + U-boot)
mkbootimage 2.2-6e0019f9ca
Nodes found in the /home/totem/firmware/totem/board/totem-common/boot.bif file:
 FSBL.elf (bootloader)
 u-boot.elf
All done, quitting
make[1]: Leaving directory '/home/totem/firmware/buildroot'
(base) hypso@WS2:~/roger/sw/sdr-system$ git status
On branch docker-hotfix
Your branch is up to date with 'origin/docker-hotfix'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hypso-shared (new commits)

no changes added to commit (use "git add" and/or "git commit -a")
(base) hypso@WS2:~/roger/sw/sdr-system$ |
```

Reviewers
rogerbirkeland ✓

Assignees
No one—assign yourself

Labels
bug

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

None yet

Notifications Customize

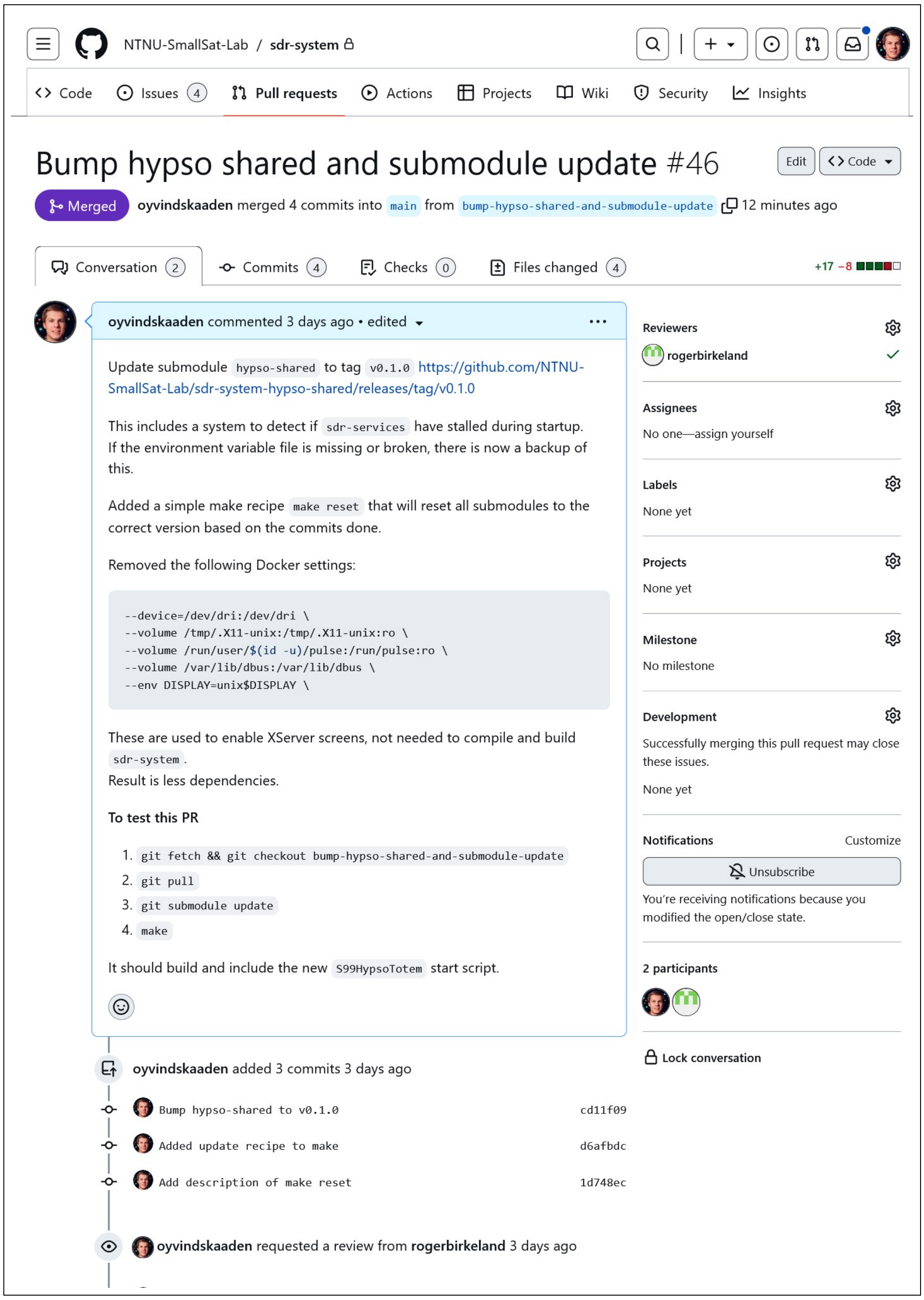
Unsubscribe

You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.8 `sdr-system#46`



NTNU-SmallSat-Lab / sdr-system

Code · Issues 4 · Pull requests · Actions · Projects · Wiki · Security · Insights

Bump hypso shared and submodule update #46

Merged — oyvindskaaden merged 4 commits into `main` from `bump-hypso-shared-and-submodule-update` 12 minutes ago

Conversation 2 · Commits 4 · Checks 0 · Files changed 4 — +17 −8

oyvindskaaden commented 3 days ago • edited

Update submodule `hypso-shared` to tag `v0.1.0` https://github.com/NTNU-SmallSat-Lab/sdr-system-hypso-shared/releases/tag/v0.1.0

This includes a system to detect if `sdr-services` have stalled during startup. If the environment variable file is missing or broken, there is now a backup of this.

Added a simple make recipe `make reset` that will reset all submodules to the correct version based on the commits done.

Removed the following Docker settings:

```
--device=/dev/dri:/dev/dri \
--volume /tmp/.X11-unix:/tmp/.X11-unix:ro \
--volume /run/user/$(id -u)/pulse:/run/pulse:ro \
--volume /var/lib/dbus:/var/lib/dbus \
--env DISPLAY=unix$DISPLAY \
```

These are used to enable XServer screens, not needed to compile and build `sdr-system`.
Result is less dependencies.

To test this PR

1. `git fetch && git checkout bump-hypso-shared-and-submodule-update`
2. `git pull`
3. `git submodule update`
4. `make`

It should build and include the new `S99HypsoTotem` start script.

oyvindskaaden added 3 commits 3 days ago

```
Bump hypso-shared to v0.1.0        cd11f09
Added update recipe to make        d6afbdc
Add description of make reset      1d748ec
```

oyvindskaaden requested a review from **rogerbirkeland** 3 days ago

Reviewers: rogerbirkeland ✓

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

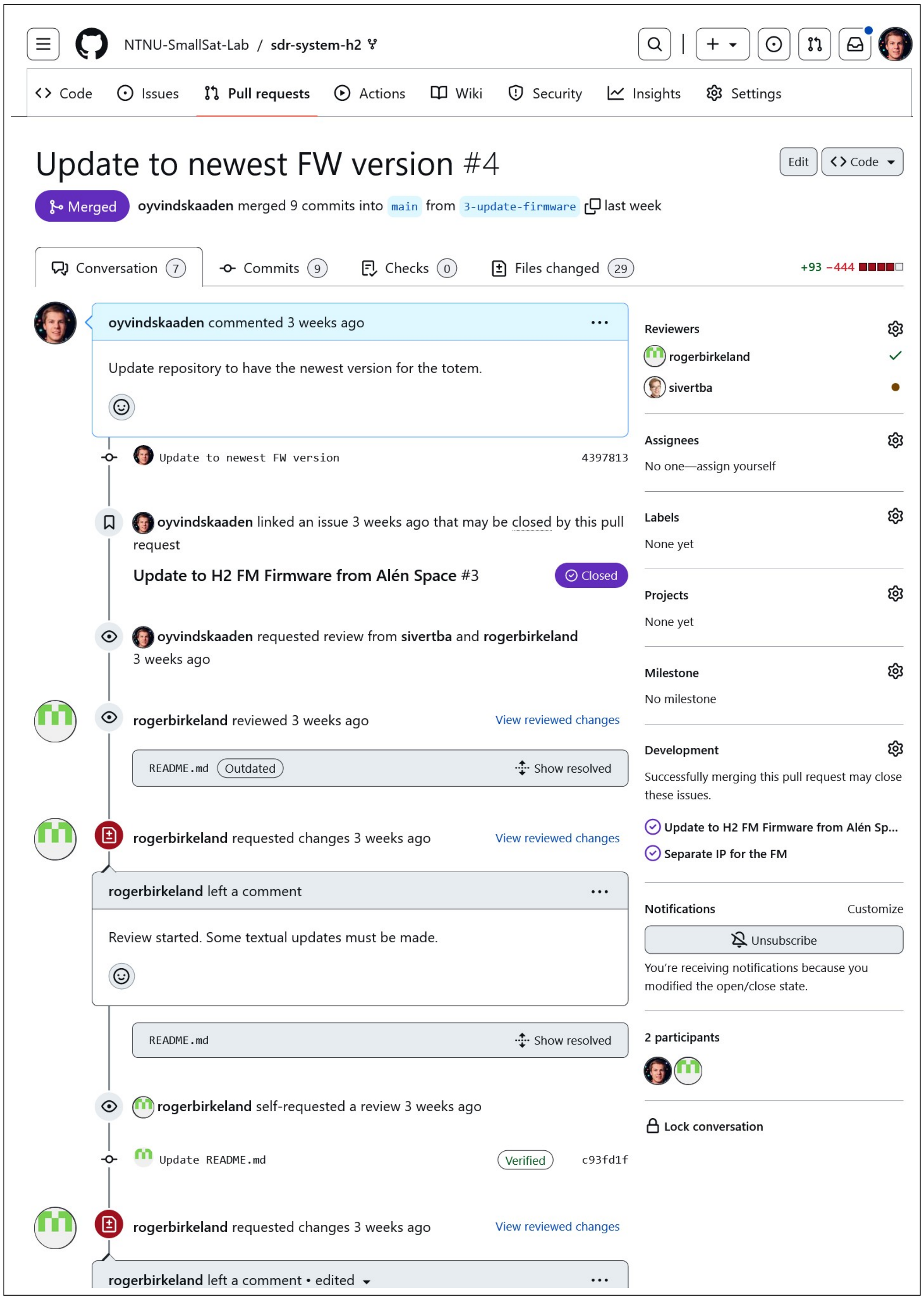
Development: Successfully merging this pull request may close these issues. None yet

Notifications Customize — Unsubscribe — You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.9 sdr-system-h2#4



Update to newest FW version #4

Merged oyvindskaaden merged 9 commits into `main` from `3-update-firmware` last week

oyvindskaaden commented 3 weeks ago

Update repository to have the newest version for the totem.

`Update to newest FW version` 4397813

oyvindskaaden linked an issue 3 weeks ago that may be closed by this pull request

Update to H2 FM Firmware from Alén Space #3 Closed

oyvindskaaden requested review from sivertba and rogerbirkeland 3 weeks ago

rogerbirkeland reviewed 3 weeks ago View reviewed changes

`README.md` Outdated Show resolved

rogerbirkeland requested changes 3 weeks ago View reviewed changes

rogerbirkeland left a comment

Review started. Some textual updates must be made.

`README.md` Show resolved

rogerbirkeland self-requested a review 3 weeks ago

`Update README.md` Verified c93fd1f

rogerbirkeland requested changes 3 weeks ago View reviewed changes

rogerbirkeland left a comment • edited

Reviewers
rogerbirkeland ✓
sivertba ●

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

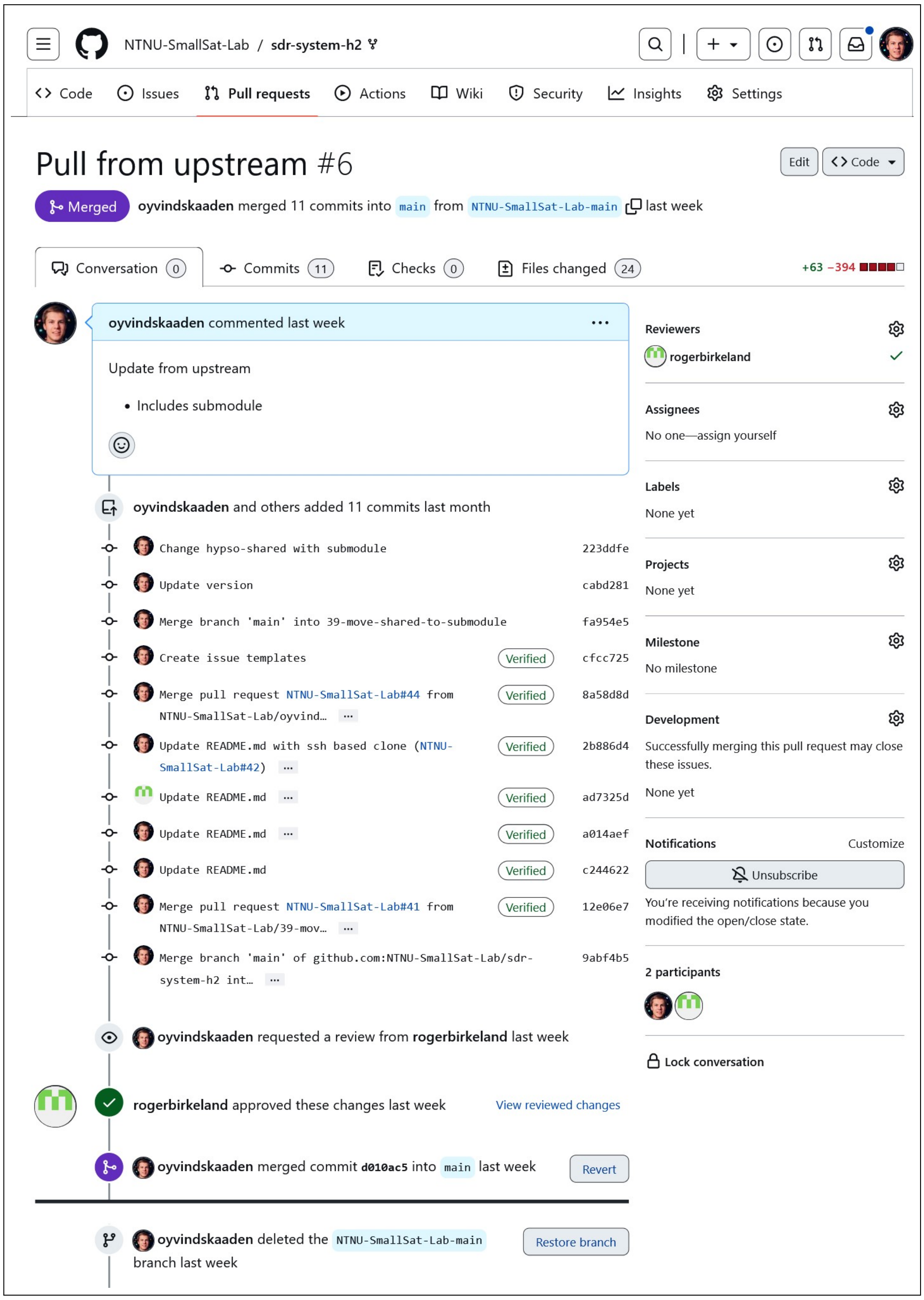
Development
Successfully merging this pull request may close these issues.
Update to H2 FM Firmware from Alén Sp...
Separate IP for the FM

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.10 `sdr-system-h2#6`



NTNU-SmallSat-Lab / sdr-system-h2

Code Issues Pull requests Actions Wiki Security Insights Settings

Pull from upstream #6

Edit | Code

Merged **oyvindskaaden** merged 11 commits into `main` from `NTNU-SmallSat-Lab-main` last week

Conversation 0 | Commits 11 | Checks 0 | Files changed 24 | +63 −394

oyvindskaaden commented last week

Update from upstream

- Includes submodule

oyvindskaaden and others added 11 commits last month

```
Change hypso-shared with submodule                                  223ddfe
Update version                                                      cabd281
Merge branch 'main' into 39-move-shared-to-submodule                fa954e5
Create issue templates                                   Verified   cfcc725
Merge pull request NTNU-SmallSat-Lab#44 from             Verified   8a58d8d
NTNU-SmallSat-Lab/oyvind… …
Update README.md with ssh based clone (NTNU-             Verified   2b886d4
SmallSat-Lab#42) …
Update README.md …                                       Verified   ad7325d
Update README.md …                                       Verified   a014aef
Update README.md                                         Verified   c244622
Merge pull request NTNU-SmallSat-Lab#41 from             Verified   12e06e7
NTNU-SmallSat-Lab/39-mov… …
Merge branch 'main' of github.com:NTNU-SmallSat-Lab/sdr-            9abf4b5
system-h2 int… …
```

oyvindskaaden requested a review from **rogerbirkeland** last week

rogerbirkeland approved these changes last week — View reviewed changes

oyvindskaaden merged commit **d010ac5** into `main` last week — Revert

oyvindskaaden deleted the `NTNU-SmallSat-Lab-main` branch last week — Restore branch

Reviewers
rogerbirkeland ✓

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

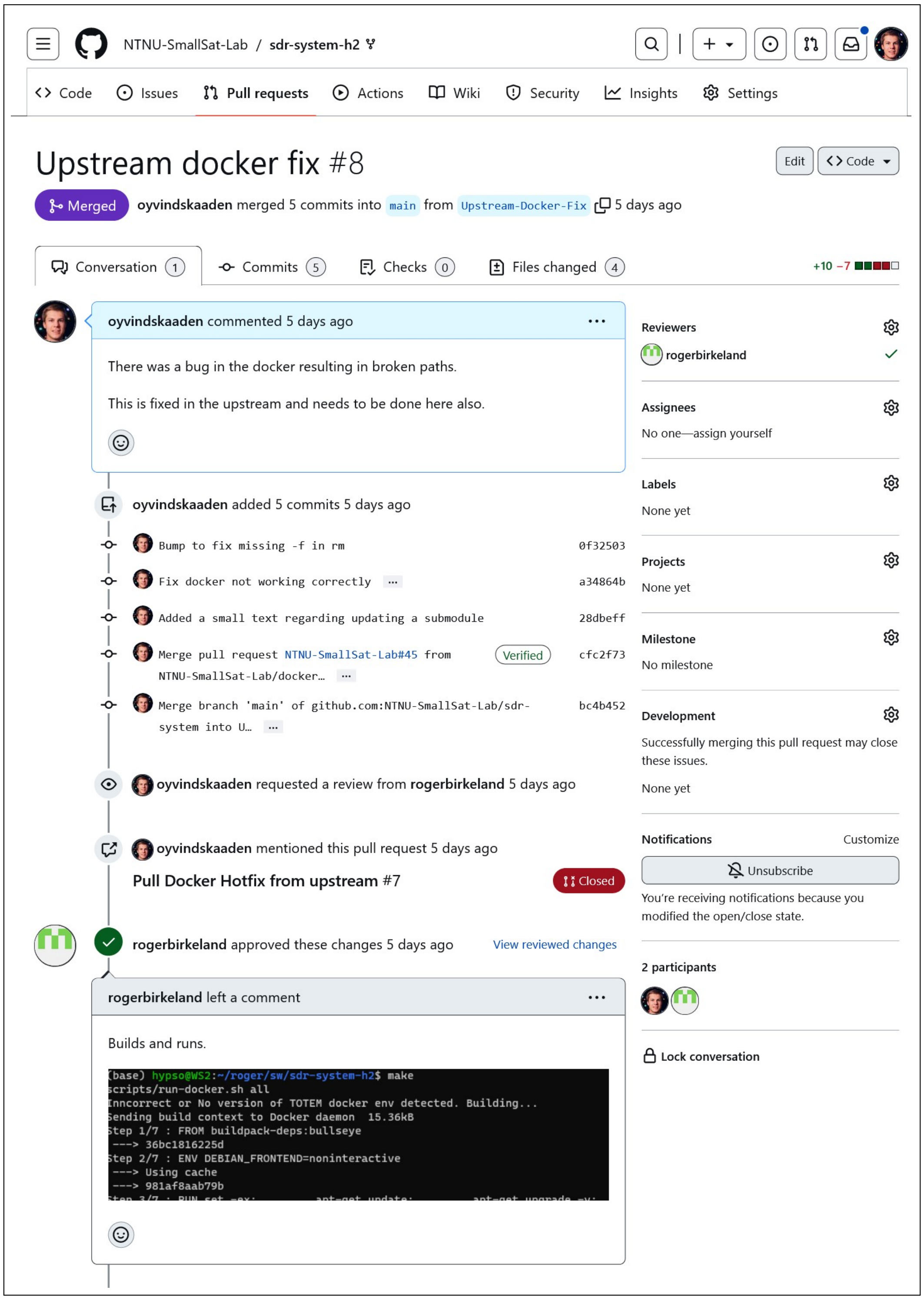
Development
Successfully merging this pull request may close these issues.
None yet

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.11 sdr-system-h2#8



Upstream docker fix #8

Merged **oyvindskaaden** merged 5 commits into `main` from `Upstream-Docker-Fix` 5 days ago

oyvindskaaden commented 5 days ago

There was a bug in the docker resulting in broken paths.

This is fixed in the upstream and needs to be done here also.

oyvindskaaden added 5 commits 5 days ago

```
Bump to fix missing -f in rm                                0f32503
Fix docker not working correctly …                          a34864b
Added a small text regarding updating a submodule           28dbeff
Merge pull request NTNU-SmallSat-Lab#45 from     Verified   cfc2f73
NTNU-SmallSat-Lab/docker… …
Merge branch 'main' of github.com:NTNU-SmallSat-Lab/sdr-    bc4b452
system into U… …
```

oyvindskaaden requested a review from **rogerbirkeland** 5 days ago

oyvindskaaden mentioned this pull request 5 days ago

Pull Docker Hotfix from upstream #7 Closed

rogerbirkeland approved these changes 5 days ago View reviewed changes

rogerbirkeland left a comment

Builds and runs.

```
(base) hypso@WS2:~/roger/sw/sdr-system-h2$ make
scripts/run-docker.sh all
Inncorrect or No version of TOTEM docker env detected. Building...
Sending build context to Docker daemon  15.36kB
Step 1/7 : FROM buildpack-deps:bullseye
 ---> 36bc1816225d
Step 2/7 : ENV DEBIAN_FRONTEND=noninteractive
 ---> Using cache
 ---> 981af8aab79b
Step 3/7 : RUN set -ex;          apt-get update;          apt-get upgrade -y;
```

Reviewers
rogerbirkeland ✓

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

None yet

Notifications Customize

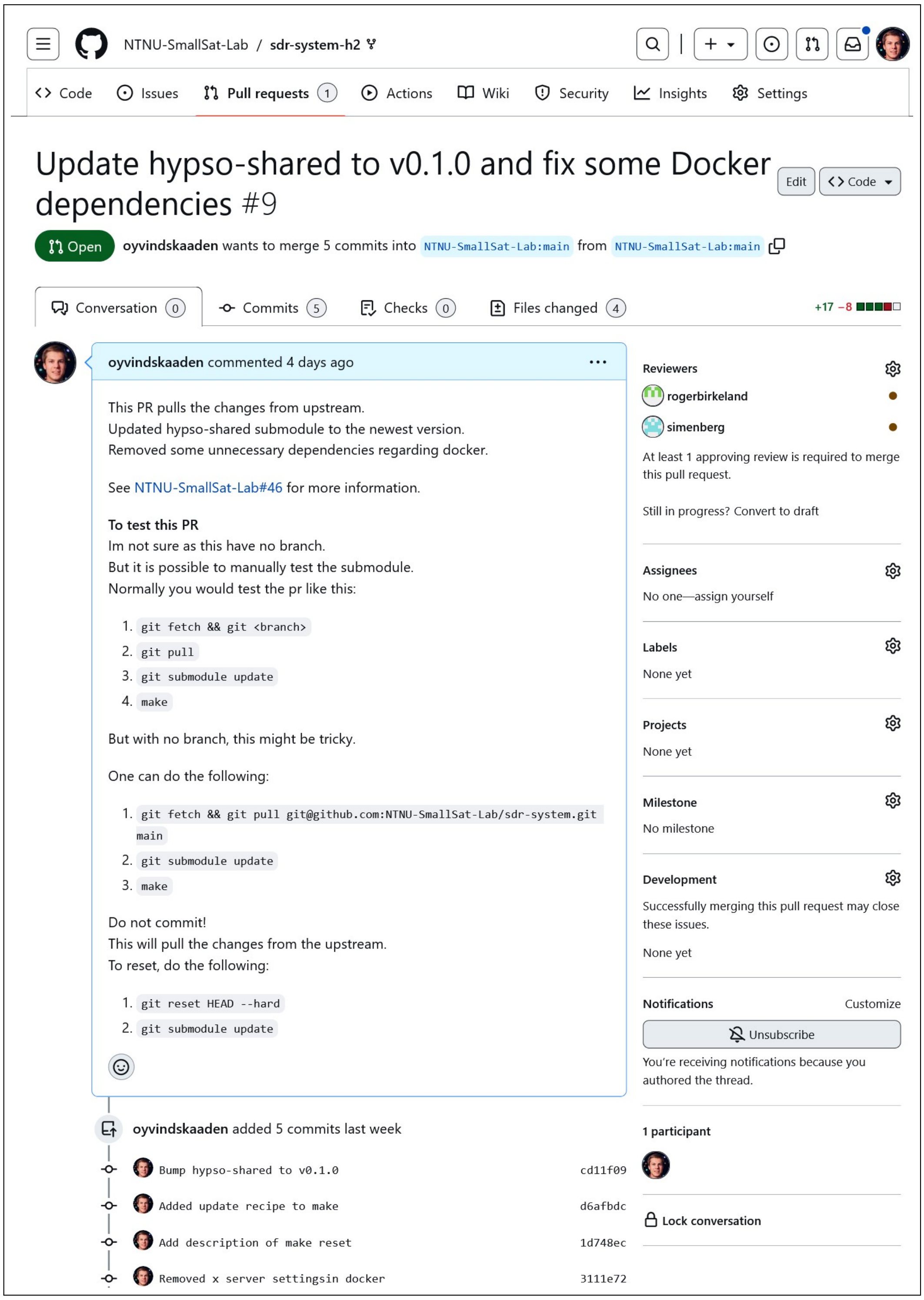
Unsubscribe

You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.12 `sdr-system-h2#9`



Update hypso-shared to v0.1.0 and fix some Docker dependencies #9

Open oyvindskaaden wants to merge 5 commits into `NTNU-SmallSat-Lab:main` from `NTNU-SmallSat-Lab:main`

oyvindskaaden commented 4 days ago

This PR pulls the changes from upstream.
Updated hypso-shared submodule to the newest version.
Removed some unnecessary dependencies regarding docker.

See NTNU-SmallSat-Lab#46 for more information.

To test this PR
Im not sure as this have no branch.
But it is possible to manually test the submodule.
Normally you would test the pr like this:

1. `git fetch && git <branch>`
2. `git pull`
3. `git submodule update`
4. `make`

But with no branch, this might be tricky.

One can do the following:

1. `git fetch && git pull git@github.com:NTNU-SmallSat-Lab/sdr-system.git main`
2. `git submodule update`
3. `make`

Do not commit!
This will pull the changes from the upstream.
To reset, do the following:

1. `git reset HEAD --hard`
2. `git submodule update`

oyvindskaaden added 5 commits last week

```
Bump hypso-shared to v0.1.0          cd11f09
Added update recipe to make          d6afbdc
Add description of make reset        1d748ec
Removed x server settingsin docker   3111e72
```

Reviewers
rogerbirkeland
simenberg
At least 1 approving review is required to merge this pull request.

Still in progress? Convert to draft

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

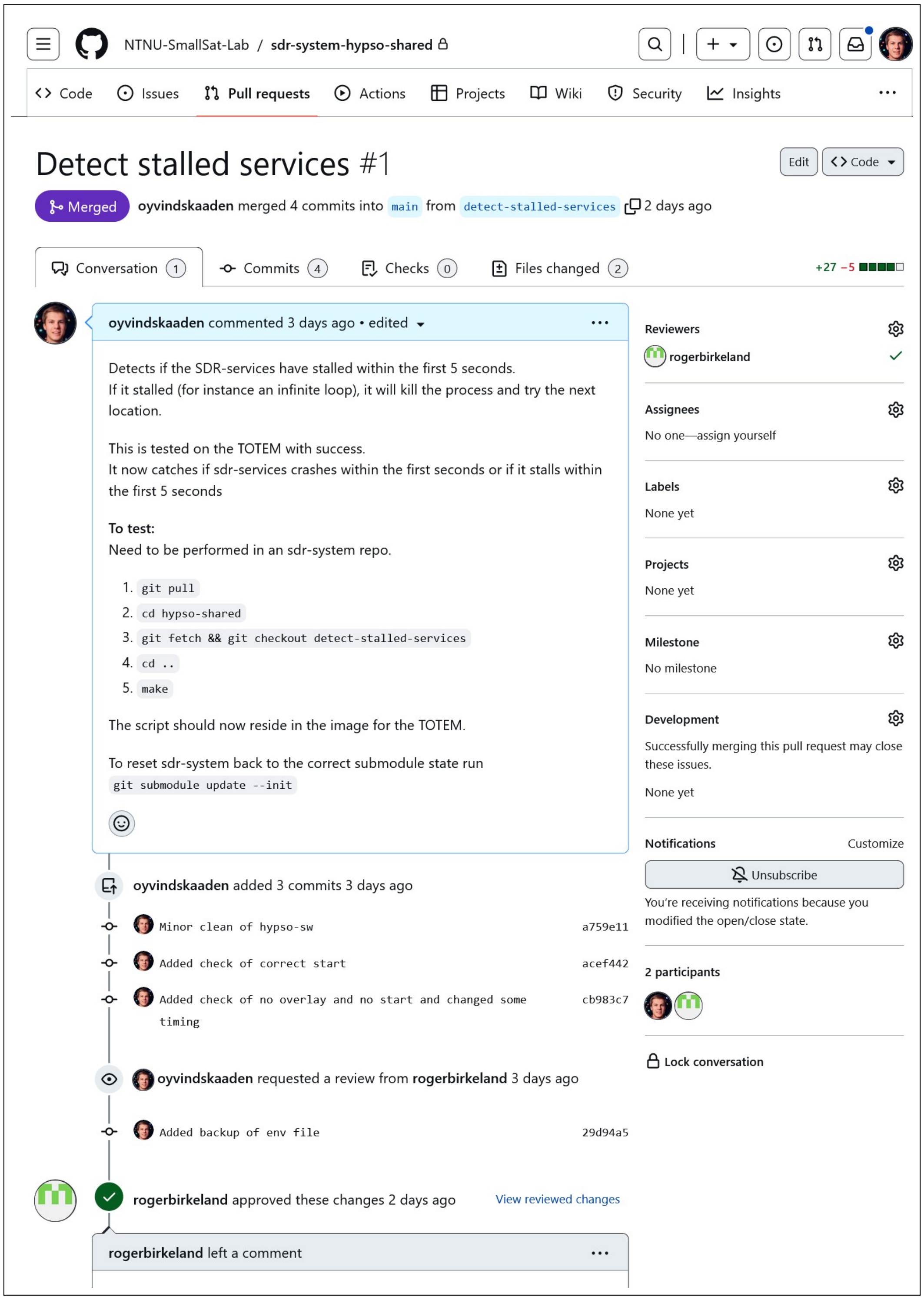
None yet

Notifications Customize
Unsubscribe
You're receiving notifications because you authored the thread.

1 participant

Lock conversation

D.13 sdr-system-hypso-shared#1



Detect stalled services #1

Merged oyvindskaaden merged 4 commits into `main` from `detect-stalled-services` 2 days ago

oyvindskaaden commented 3 days ago • edited

Detects if the SDR-services have stalled within the first 5 seconds.
If it stalled (for instance an infinite loop), it will kill the process and try the next location.

This is tested on the TOTEM with success.
It now catches if sdr-services crashes within the first seconds or if it stalls within the first 5 seconds

To test:
Need to be performed in an sdr-system repo.

1. `git pull`
2. `cd hypso-shared`
3. `git fetch && git checkout detect-stalled-services`
4. `cd ..`
5. `make`

The script should now reside in the image for the TOTEM.

To reset sdr-system back to the correct submodule state run `git submodule update --init`

oyvindskaaden added 3 commits 3 days ago

```
Minor clean of hypso-sw                                     a759e11
Added check of correct start                                acef442
Added check of no overlay and no start and changed some     cb983c7
timing
```

oyvindskaaden requested a review from **rogerbirkeland** 3 days ago

```
Added backup of env file                                    29d94a5
```

rogerbirkeland approved these changes 2 days ago View reviewed changes

rogerbirkeland left a comment

Reviewers
rogerbirkeland ✓

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.

None yet

Notifications Customize

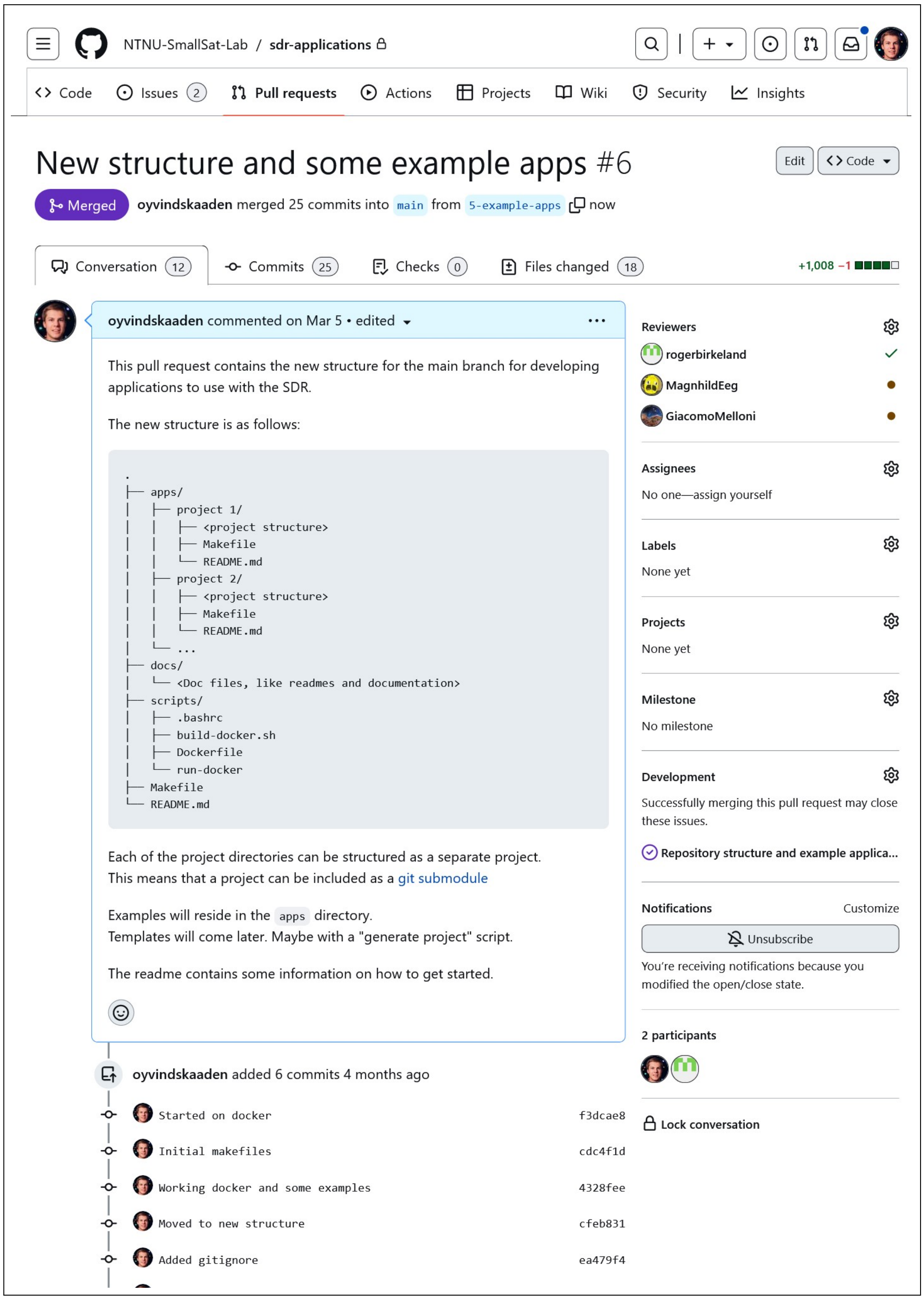
Unsubscribe

You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.14 sdr-applications#6



NTNU-SmallSat-Lab / sdr-applications

New structure and some example apps #6

Merged oyvindskaaden merged 25 commits into `main` from `5-example-apps` now

oyvindskaaden commented on Mar 5 • edited

This pull request contains the new structure for the main branch for developing applications to use with the SDR.

The new structure is as follows:

```
.
├── apps/
│   ├── project 1/
│   │   ├── <project structure>
│   │   ├── Makefile
│   │   └── README.md
│   ├── project 2/
│   │   ├── <project structure>
│   │   ├── Makefile
│   │   └── README.md
│   └── ...
├── docs/
│   └── <Doc files, like readmes and documentation>
├── scripts/
│   ├── .bashrc
│   ├── build-docker.sh
│   ├── Dockerfile
│   └── run-docker
├── Makefile
└── README.md
```

Each of the project directories can be structured as a separate project.
This means that a project can be included as a git submodule

Examples will reside in the `apps` directory.
Templates will come later. Maybe with a "generate project" script.

The readme contains some information on how to get started.

oyvindskaaden added 6 commits 4 months ago

```
Started on docker                    f3dcae8
Initial makefiles                    cdc4f1d
Working docker and some examples     4328fee
Moved to new structure               cfeb831
Added gitignore                      ea479f4
```

Reviewers
rogerbirkeland ✓
MagnhildEeg ●
GiacomoMelloni ●

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

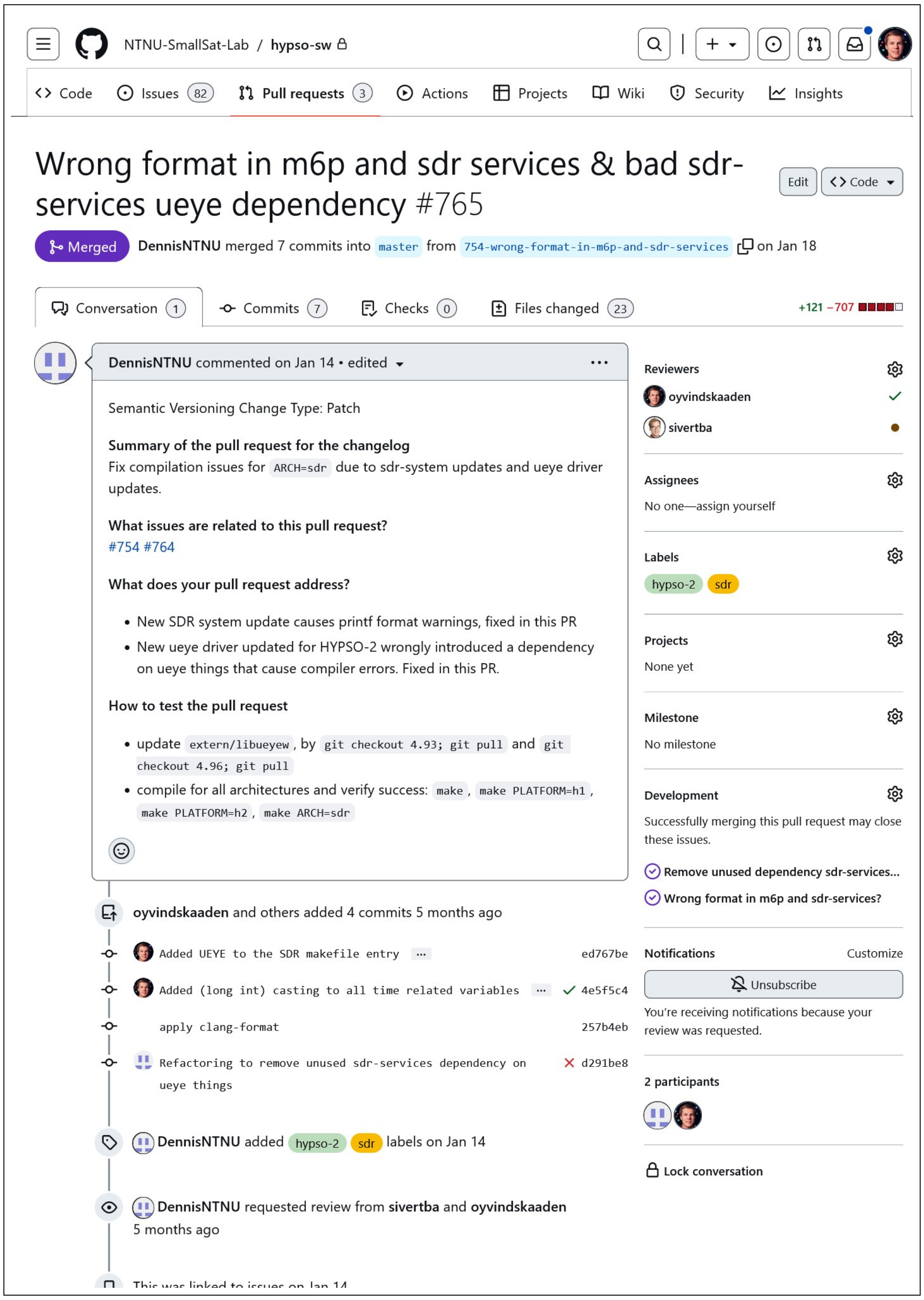
Development
Successfully merging this pull request may close these issues.
Repository structure and example applica...

Notifications Customize
Unsubscribe
You're receiving notifications because you modified the open/close state.

2 participants

Lock conversation

D.15 hypso-sw#765



Wrong format in m6p and sdr services & bad sdr-services ueye dependency #765

Merged **DennisNTNU** merged 7 commits into `master` from `754-wrong-format-in-m6p-and-sdr-services` on Jan 18

DennisNTNU commented on Jan 14 • edited

Semantic Versioning Change Type: Patch

Summary of the pull request for the changelog
Fix compilation issues for `ARCH=sdr` due to sdr-system updates and ueye driver updates.

What issues are related to this pull request?
#754 #764

What does your pull request address?

- New SDR system update causes printf format warnings, fixed in this PR
- New ueye driver updated for HYPSO-2 wrongly introduced a dependency on ueye things that cause compiler errors. Fixed in this PR.

How to test the pull request

- update `extern/libueyew`, by `git checkout 4.93; git pull` and `git checkout 4.96; git pull`
- compile for all architectures and verify success: `make`, `make PLATFORM=h1`, `make PLATFORM=h2`, `make ARCH=sdr`

oyvindskaaden and others added 4 commits 5 months ago

```
Added UEYE to the SDR makefile entry                          ed767be
Added (long int) casting to all time related variables   ✓   4e5f5c4
apply clang-format                                            257b4eb
Refactoring to remove unused sdr-services dependency on  ✗   d291be8
ueye things
```

DennisNTNU added hypso-2 sdr labels on Jan 14

DennisNTNU requested review from **sivertba** and **oyvindskaaden**
5 months ago

This was linked to issues on Jan 14

Reviewers
oyvindskaaden ✓
sivertba ●

Assignees
No one—assign yourself

Labels
hypso-2 sdr

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
Remove unused dependency sdr-services...
Wrong format in m6p and sdr-services?

Notifications
You're receiving notifications because your review was requested.

2 participants