An Open Source Implementation of the ECSS PUS-C Services in Rust

Author: Selman Özleyen
Co-authors: Jose Feiteirinha, Filip Geib

INTRODUCTION

We will talk about;

- A system programming language called **Rust**.
- ECSS PUS-C which is a standard.
- An open source ECSS PUS-C implementation in Rust which is called **Prust.**
- **Prust**'s use cases in **VST104,** which was where Prust was deployed first.

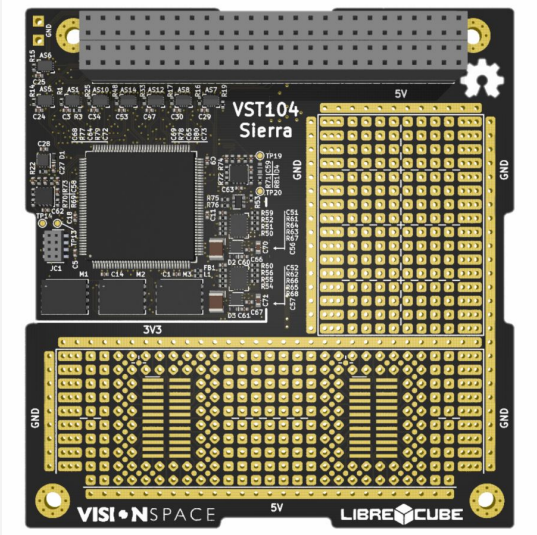
PUS-C (ECSS-E-ST-70-41C)

- ECSS is a cooperative effort for the purpose of developing and maintaining common standards.
- Its recent iteration PUS-C was published in 2016 and it's relatively new.
- Prust satisfies some of the service requirements of PUS-C
- PUS is the standard of choice by ESA, and only one spacecraft operated by ESA doesn't use PUS (OPS-SAT).

VST104 board_sierra

This board hosts a single redundant onboard computer designed to fulfill space industry requirements. The main processing unit is STM32L496 microprocessor.

Runs the Prust software. Will be presented tomorrow by Filip Geib (An Open-Source on-board computer platform for CubeSats).



Rust Language

- It is a programming language focused on performance and safety, especially safe concurrency.
- Rust provides memory safety without using garbage collection, but instead through the use of a borrow checking system.
- "most loved programming language" in the Stack Overflow Developer Survey every year since 2016.

Why Rust ?

- C remained the only alternative for a long time because it was faster than other programming languages.
- Rust offered a cost-free way of ensuring memory safety.
- It also offered usage of High-Level programming features with System-Level performance.

Prusty

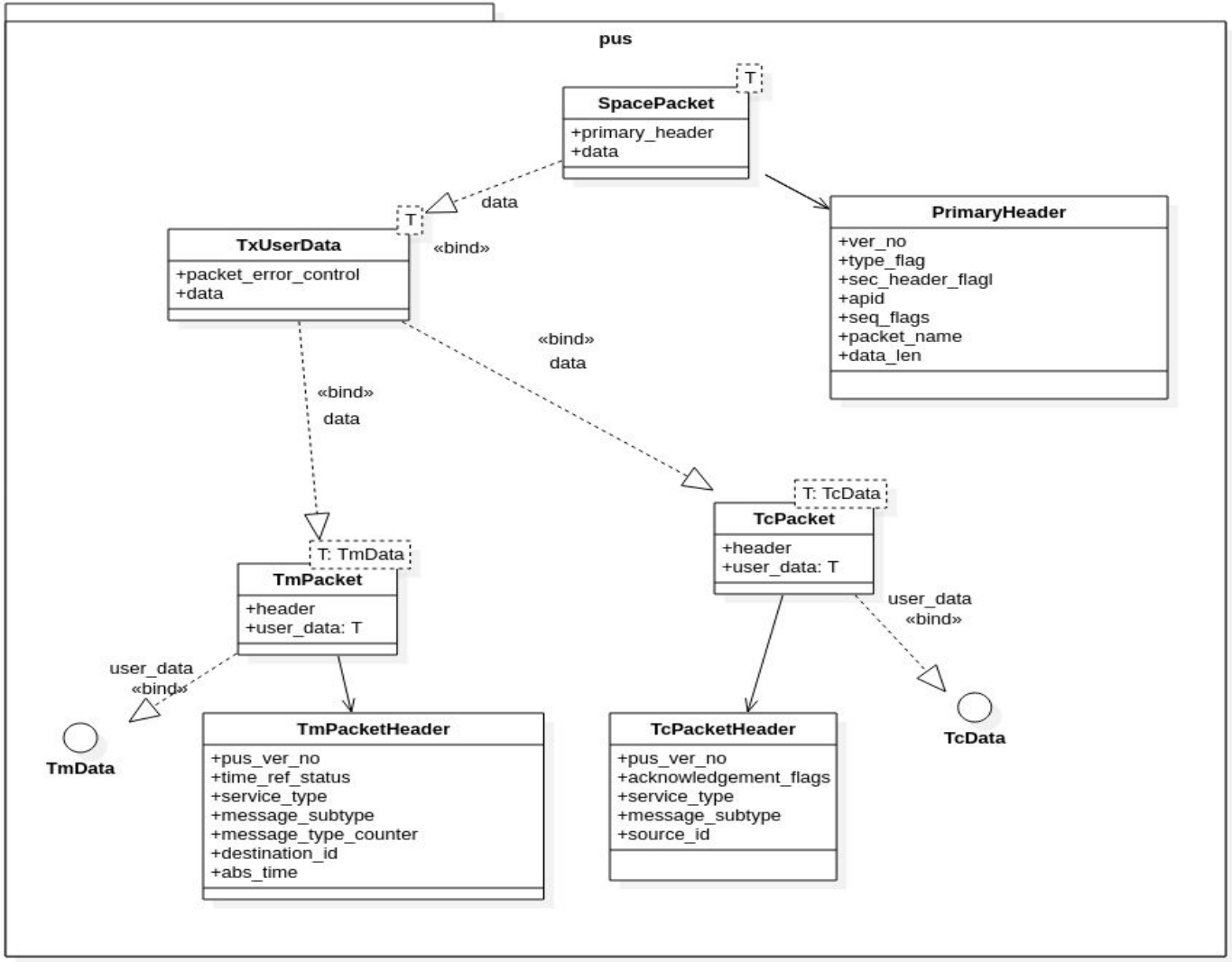
- Is a software implemented with Rust language
- It complies to the PUS-C.
- Tested on VST104 project.
- Aims to be reliable, fast and maintenance cost efficient.

PUS-C packets in Prust

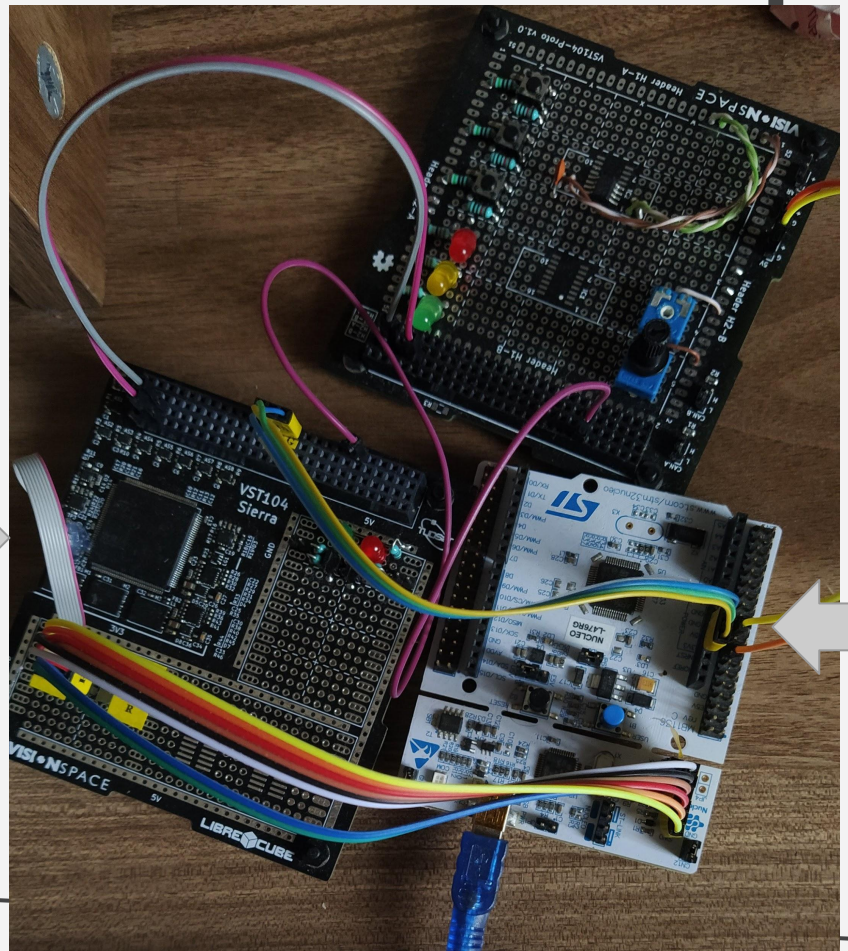
- It implements the data structures to send, receive, and interpret PUS space packets
- Has Service 1,3 and 8 data structures in it from PUS-C. (The definitions on next slide)
- Basically a representation of the PUS-C packets in Rust.

Implemented Services

Service Implemented	Summary
Function Management (Service 8)	It has one request (TC) and it executes a function defined by the user by giving the name of the function
Request Verification (Service 1)	It has 9 response (TM) types which every one of them indicates different states for the send subservices (for example failure,success etc..)
Housekeeping (Service 3)	It has plenty of TM and TC packets in it. It helps to have reports of the peripherals connected to the device. It is implemented partially in Prust


pus
T
SpacePacket
+primary_header
+data
data
T
TxUserData
+packet_error_control
+data
«bind»
PrimaryHeader
+ver_no
+type_flag
+sec_header_flagl
+apid
+seq_flags
+packet_name
+data_len
«bind»
data
«bind»
data
T: TcData
TcPacket
+header
+user_data: T
T: TmData
TmPacket
+header
+user_data: T
user_data
«bind»
user_data
«bind»
TmData
TmPacketHeader
+pus_ver_no
+time_ref_status
+service_type
+message_subtype
+message_type_counter
+destination_id
+abs_time
TcPacketHeader
+pus_ver_no
+acknowledgement_flags
+service_type
+message_subtype
+source_id
TcData

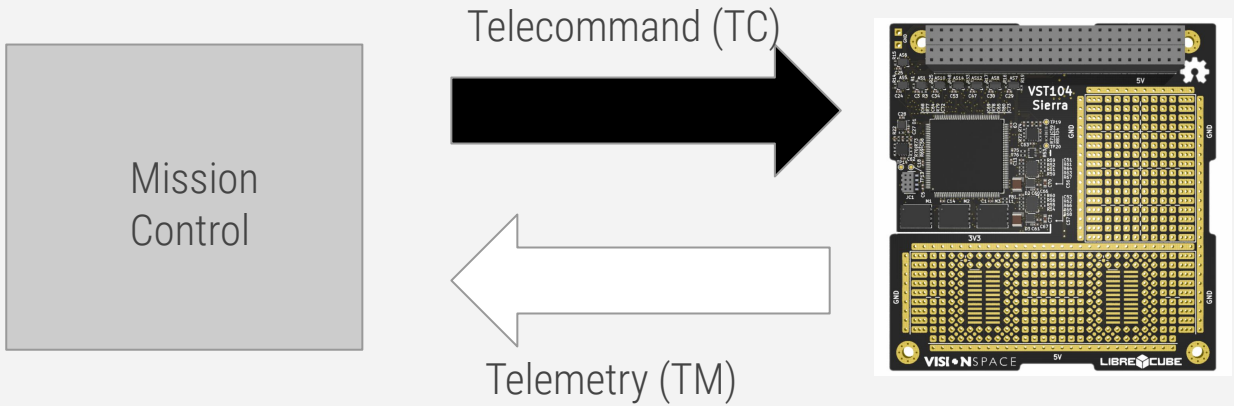
Test Case: The Setup



board_zero

board_sierra

Programmer
(to deploy software to
the VST104 board)



Sending a command and getting a response works like this here.

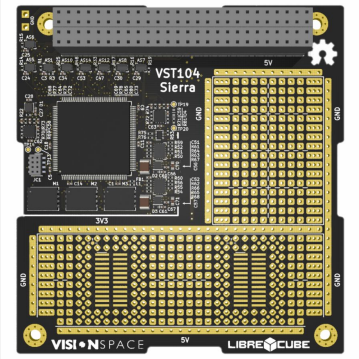
Example 1:
Function management and request verification and report (Service 1 and 8)

STEP 1-
TC[8,1] for example can mean;
Execute function "*turn_led*" from functions.rs file with argument: *true*

Function Table (functions.rs file)
-say_hi_to_aliens()
-turn_led(bool)
-set_led(int,bool)
-sing_happy_bday_to_curiosity(age)

Mission Control (TC)

TC[8,1]



Mission Control (TM)

TM[1,7]

STEP 2-Evaluate Packet;
Find the function from ***Function Table*** and execute it and generate a ***TM*** to report

STEP 3-
TM[1,7] for example can mean;
Recent TC[8,1] request did finish it's execution successfully

Example 2:
Housekeeping (Service 3)

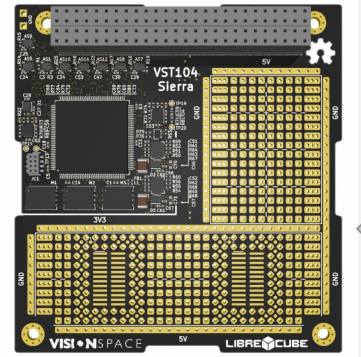
STEP 1-
TC[3,1] for example can mean;
Create a structure of parameters which includes number 1 and 2 and set this structure id to 0

Parameters

1. Temperature (32 bit integer)
2. Potentiometer (16 bit integer)
3. Internal Voltage (32 bit float)

Mission Control (TC)

TC[3,1]



Mission Control (TM)

TM[1,7]

STEP 2-Evaluate Packet;
Create a structure with id 0 and 1 and 2 parameters inside.

STEP 3-
TM[1,7] for example can mean;
Recent TC[3,1] request did finish it's execution successfully.

Structures
(empty)

CONT..

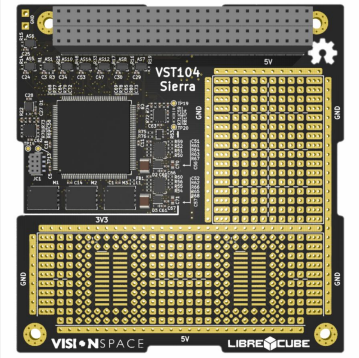
STEP 4-
TC[3,5] for example can mean;
Enable periodic collection of Structure Id 0 and report it periodically.

Parameters

1. Temperature (32 bit integer)
2. Potentiometer (16 bit integer)
3. Internal Voltage (32 bit float)

Mission Control (TC)

TC[3,5]



Mission Control (TM)

TM[3,25]

STEP 5-Evaluate Packet;
Enable timer and periodically report parameters of Structure Id 0.

STEP 6- (This is send *PERIODICALLY*)
TM[3,25] for example can mean;
A packet containing parameter values of Structure Id 0.

Structures
0. Includes parameter 1 and 2

Next Steps

Extend the Prust source code with the following functionality;

- Add check by APID
- Read flash and F-Ram device types and UUID - to be provided via service 8 request
- Read temperature from on-board sensors - to be provided via service 3
- Set clock speed
- Add new test cases for each new feature
- Look into Rust based RTOS for Embedded development.
- Propose and implement the migration of existing code onto a RTOS kernel

If you are interested in contributing
you can contact this email address: jose.feiteirinha@visionspace.com

Here is the public repository: https://github.com/visionspacetec/Prust

QUESTIONS?

I'd like to hear your questions and thoughts.
It might be about Rust or the VST104 board maybe? Or anything really...

THANKS!

Also thanks to Jose Feiteirinha, Fatih Erten, Filip Geib and VisionSpace for their support.

github:SelmanOzleyen
email:syozleyen@gmail.com
linkedin:selman_oz