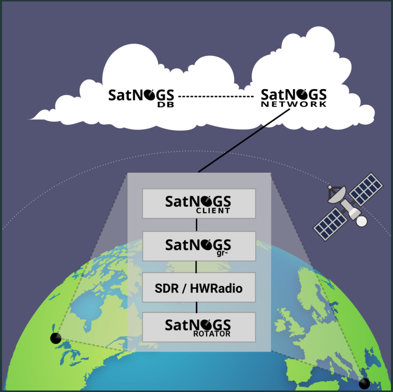
SatNOGS-COMMS: An Open-Source Communication Subsystem for CubeSats

FOSDEM 2025

Manolis Surligas manolis@libre.space
Libre Space Foundation

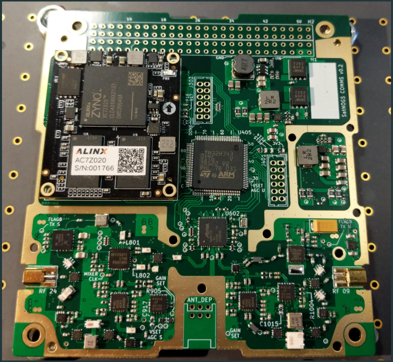
SatNOGS in a nutshell

- Ground Stations Network
- Modular setup
- SDR based RF
- Complete open source stack
- VHF/UHF, L-Band, S-Band (expanding to X-band)

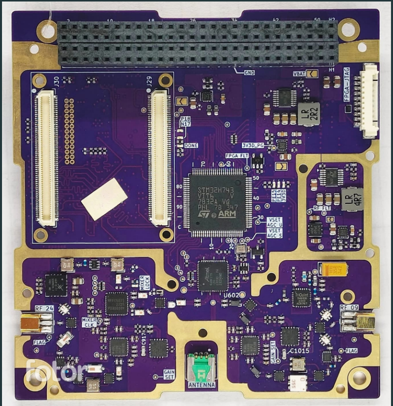


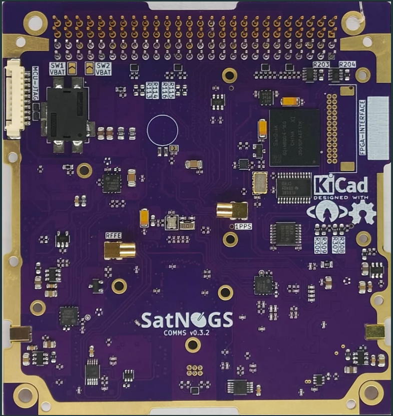
SatNOGS-COMMS transceiver

- Co-funded by LSF and ESA
- UHF and S-Band dedicated transceivers
- STM32H7 main MCU
- ZYNQ-7020 FPGA
- Suitable for LEO up to 600 km
- Fully open software and hardware
- Seamless SatNOGS Network integration
- Suitable for a wide range of Cubesat missions



SatNOGS-COMMS transceiver

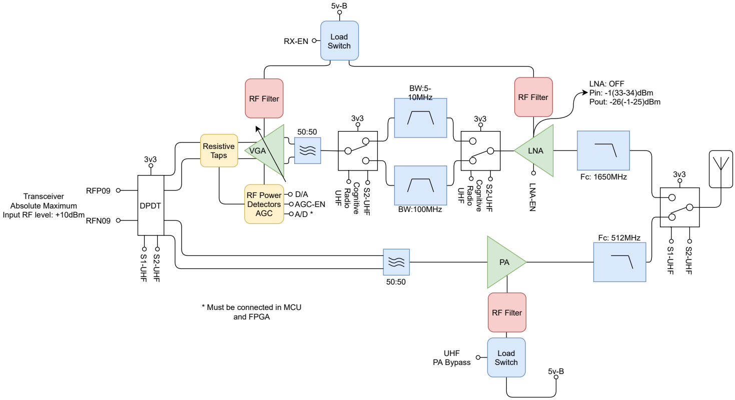




RF Frontend

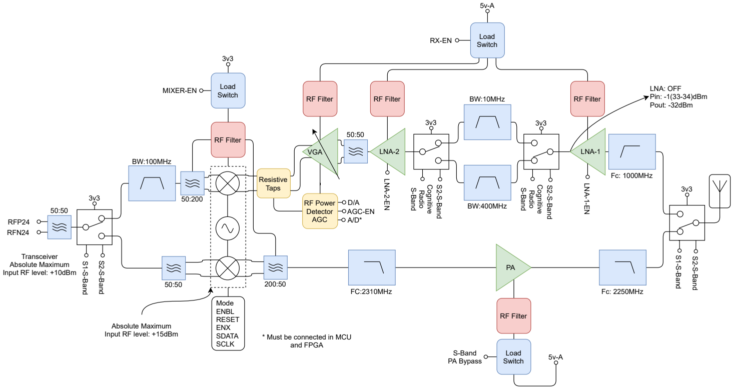
- **UHF:** 395 – 450 MHz
- **S-Band:**
  - Rx: 2025 – 2110 MHz
  - Tx: 2200 – 2290 MHz
  - Radio amateur bands support upon request
- **Tx Power:** 26 - 32 dBm (1 dB step)
- **SFCG 21–2R4** compliant emissions
- Low noise figure (1.4 dB)

RF Frontend: UHF



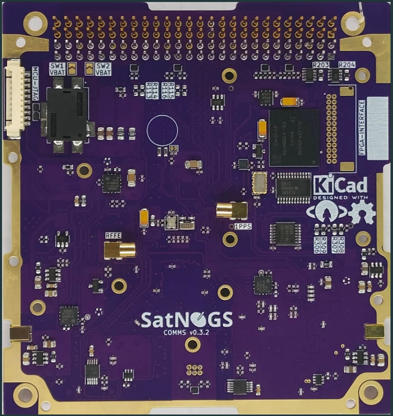
* Must be connected in MCU and FPGA

RF Frontend: S-Band

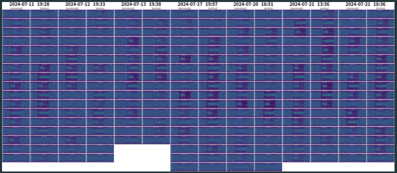


IO Interfaces

- 2× CAN-2.0
- 1× SPI up to 8 Mbps
- 1× I2C
- 3× UART
- 1× RGMII Ethernet
- 2× antenna deployment interfaces
- Reference clock and PPS inputs
- PC/104



Missions: Curium-1 on Ariane 6!

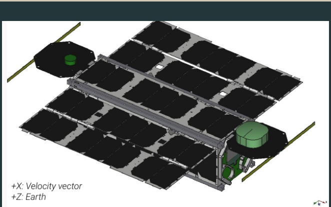


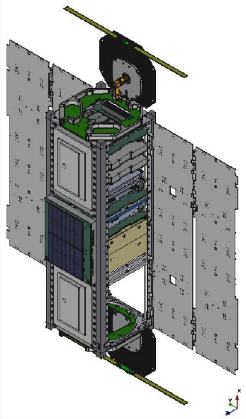
1

[1] www.camras.nl/blog/2024/satelliet-curium-one-gezien-vanuit-dwingeloo

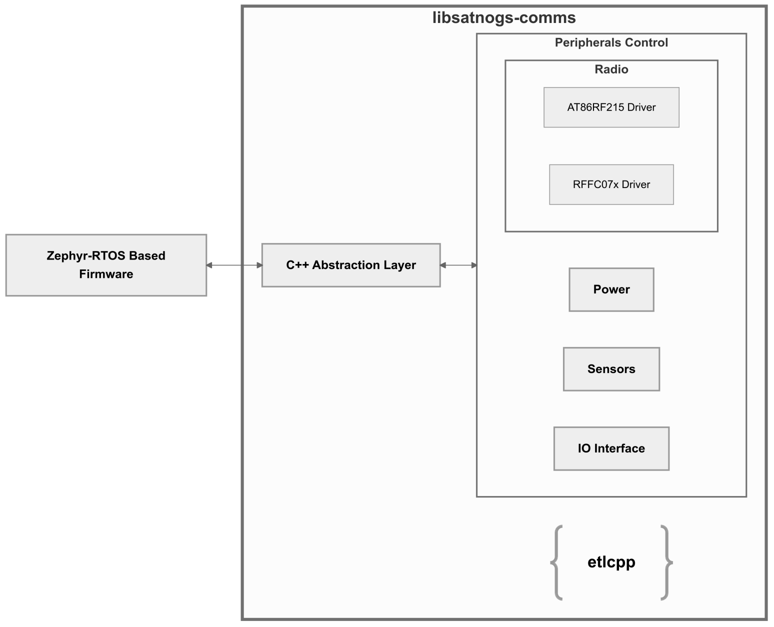
Missions: PHASMA LAMARR & DIRAC

- SatNOGS-COMMS will be used in PHASMA , a 2× 3U Cubesat mission for spectrum monitoring
- One board for OBC/TC&C, another for spectrum monitoring
- Q3 2025, on Transporter-15



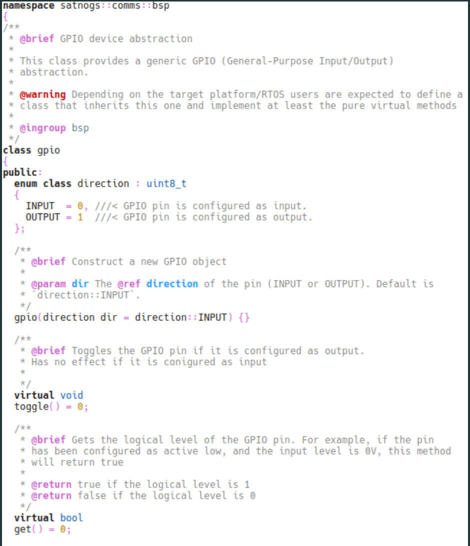


Onboard Software



Onboard Software: libsatnogs-comms

- Platform-agnostic
- Available as CMake interface library
- C++17 everywhere!
- Abstract interface based on pure virtual methods for platform specific operations



```
namespace satnogs::comms::bsp
{
/**
 * @brief GPIO device abstraction
 *
 * This class provides a generic GPIO (General-Purpose Input/Output)
 * abstraction.
 *
 * @warning Depending on the target platform/RTOS users are expected to define a
 * class that inherits this one and implement at least the pure virtual methods
 *
 * @ingroup bsp
 */
class gpio
{
public:
  enum class direction : uint8_t
  {
    INPUT  = 0, ///< GPIO pin is configured as input.
    OUTPUT = 1  ///< GPIO pin is configured as output.
  };

  /**
   * @brief Construct a new GPIO object
   *
   * @param dir The @ref direction of the pin (INPUT or OUTPUT). Default is
   * `direction::INPUT`.
   */
  gpio(direction dir = direction::INPUT) {}

  /**
   * @brief Toggles the GPIO pin if it is configured as output.
   * Has no effect if it is conigured as input
   *
   */
  virtual void
  toggle() = 0;

  /**
   * @brief Gets the logical level of the GPIO pin. For example, if the pin
   * has been configured as active low, and the input level is 0V, this method
   * will return true
   *
   * @return true if the logical level is 1
   * @return false if the logical level is 0
   */
  virtual bool
  get() = 0;
```

Onboard Software: Zephyr-RTOS

But why?

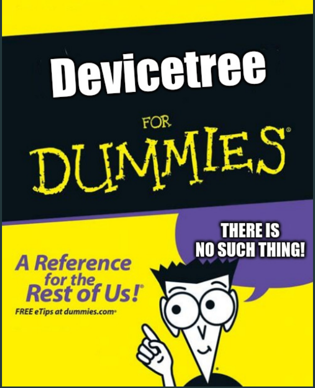
- Modern
- Huge community
- Actively developed
- Modular
- Large number of modules
- CMake
- Devicetree (please don't shoot me!)

Onboard Software: Zephyr-RTOS

Zephyr-RTOS Components in use		
ADC	DAC	GPIO
UART Async	SPI	i2c
retention	sensors	emmc & disk access
GNSS	settings	RTC
hwinfo	console	nanopb
LittleFS	Task Watchdog	CAN & ISOTP
sysbuild	MCUBoot with XIP	twister
And many more!		

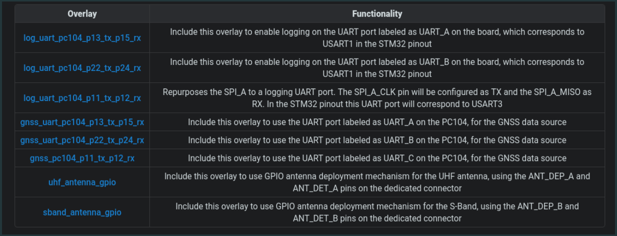
Devicetree

- Support multiple hardware versions as development progresses
- Customization options for different missions though overlays
- Together with the libsatnogs-comms abstraction layer, provides a bulletproof code base even if the SoC changes



Devicetree

Currently we support more than 30 different configurations for various IO interfaces and subsystems that a satellite mission may require with ZERO code modifications!



Overlay	Functionality
log_uart_pc104_p13_tx_p15_rx	Include this overlay to enable logging on the UART port labeled as UART_A on the board, which corresponds to USART1 in the STM32 pinout
log_uart_pc104_p22_tx_p24_rx	Include this overlay to enable logging on the UART port labeled as UART_B on the board, which corresponds to USART1 in the STM32 pinout
log_uart_pc104_p11_tx_p12_rx	Repurposes the SPI_A to a logging UART port. The SPI_A_CLK pin will be configured as TX and the SPI_A_MISO as RX. In the STM32 pinout this UART port will correspond to USART3
gnss_uart_pc104_p13_tx_p15_rx	Include this overlay to use the UART port labeled as UART_A on the PC104, for the GNSS data source
gnss_uart_pc104_p22_tx_p24_rx	Include this overlay to use the UART port labeled as UART_B on the PC104, for the GNSS data source
gnss_pc104_p11_tx_p12_rx	Include this overlay to use the UART port labeled as UART_C on the PC104, for the GNSS data source
uhf_antenna_gpio	Include this overlay to use GPIO antenna deployment mechanism for the UHF antenna, using the ANT_DEP_A and ANT_DET_A pins on the dedicated connector
sband_antenna_gpio	Include this overlay to use GPIO antenna deployment mechanism for the S-Band, using the ANT_DEP_B and ANT_DET_B pins on the dedicated connector

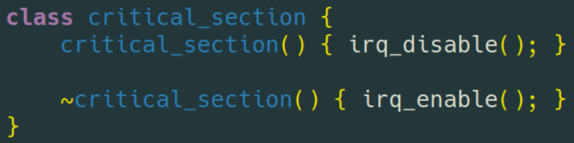
[2]

[2] https://librespacefoundation.gitlab.io/satnogs-comms/satnogs-comms-software-mcu/group__customization.html

C++. This is the way!

The use of C++ contributes significantly towards a reliable system that must operate unattended

- Better code organization through polymorphism
- Safer abstraction layers (no need for weak or function pointers)
- RAII (Resource allocation is initialization) idiom



```
class critical_section {
    critical_section() { irq_disable(); }

    ~critical_section() { irq_enable(); }
}
```

C++. This is the way!

- References instead of pointers
- Template metaprogramming FTW!
- Readable and maintainable compile time checks through **constexpr**
- Exceptions instead of error codes

C++. This is the way?

Challenges?

- STL and dynamic memory allocation -> **No go for space!**
- RTTI is not an option for the majority of embedded devices
- Even exceptions in not an option for flash limited devices

What about the STL?

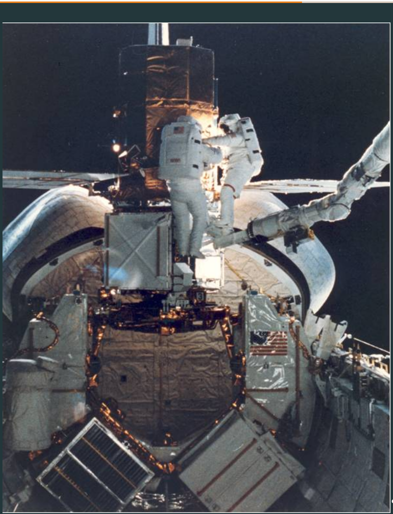
SOLUTIONS FOR THE STL PROBLEM?
WE CAN AVOID STL!
C-LIKE C++!
A HEAP MANAGER!

etlcpp to the rescue!

- etlcpp is an STL-like library that makes 0 dynamic memory allocation
- Maximum memory is known at compile time
- No RTTI
- Fully templated
- STL API compatible
- Multiple available approaches for error handling
  - Exceptions
  - Error codes
  - ghosting
- `https://www.etlcpp.com`

Error handling

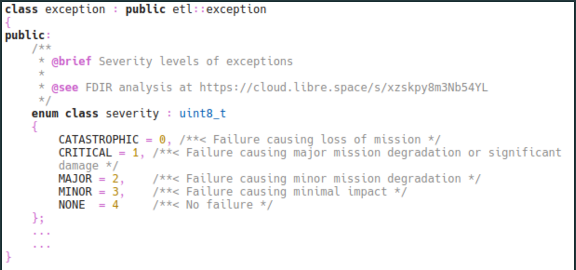
- Error identification and recovery is one of the most critical aspects of a satellite software
- Errors should be also logged for the operator to be able to troubleshoot from ground

[a]

[a] STC-41C repair mission of Solar Max satellite, 1984

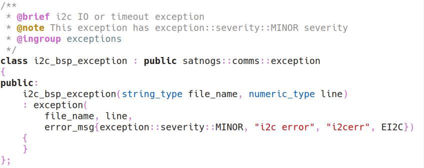
Error handling

- **std::exception**
- Unified error/logging system
- 4 different backends:
  - SWO
  - Ring buffer
  - eMMC storage
  - BACKUP_SRAM
- Exceptions of different severity level

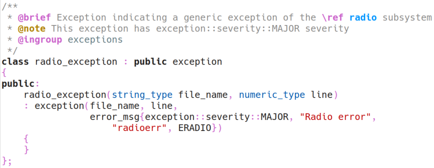


```
class exception : public etl::exception
{
public:
    /**
     * @brief Severity levels of exceptions
     *
     * @see FDIR analysis at https://cloud.libre.space/s/xzskpy8m3Nb54YL
     */
    enum class severity : uint8_t
    {
        CATASTROPHIC = 0, /**< Failure causing loss of mission */
        CRITICAL = 1, /**< Failure causing major mission degradation or significant
        damage */
        MAJOR = 2,    /**< Failure causing minor mission degradation */
        MINOR = 3,    /**< Failure causing minimal impact */
        NONE  = 4     /**< No failure */
    };
    ...
    ...
}
```

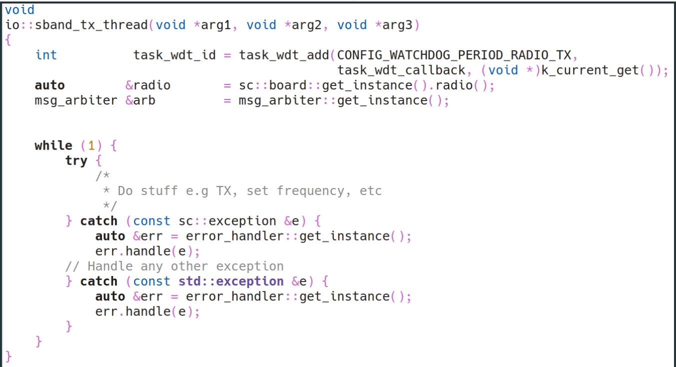
Error handling



```
/**
 * @brief i2c IO or timeout exception
 * @note This exception has exception::severity::MINOR severity
 * @ingroup exceptions
 */
class i2c_bsp_exception : public satnogs::comms::exception
{
public:
    i2c_bsp_exception(string_type file_name, numeric_type line)
    : exception(
        file_name, line,
        error_msg{exception::severity::MINOR, "i2c error", "i2cerr", EI2C})
    {
    }
};
```

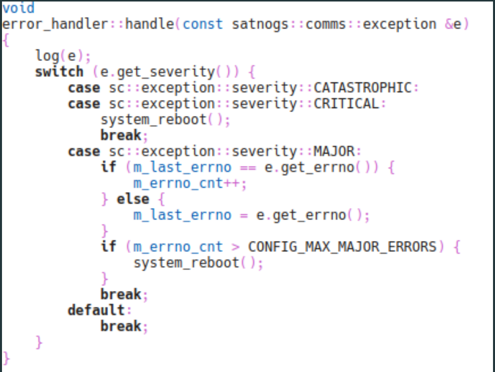


```
/**
 * @brief Exception indicating a generic exception of the \ref radio subsystem
 * @note This exception has exception::severity::MAJOR severity
 * @ingroup exceptions
 */
class radio_exception : public exception
{
public:
    radio_exception(string_type file_name, numeric_type line)
    : exception(file_name, line,
                error_msg{exception::severity::MAJOR, "Radio error",
                    "radioerr", ERADIO})
    {
    }
};
```



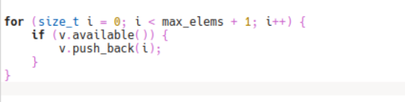
```
void
io::sband_tx_thread(void *arg1, void *arg2, void *arg3)
{
    int          task_wdt_id = task_wdt_add(CONFIG_WATCHDOG_PERIOD_RADIO_TX,
                                            task_wdt_callback, (void *)k_current_get());
    auto         &radio      = sc::board::get_instance().radio();
    msg_arbiter &arb         = msg_arbiter::get_instance();


    while (1) {
        try {
            /*
             * Do stuff e.g TX, set frequency, etc
             */
        } catch (const sc::exception &e) {
            auto &err = error_handler::get_instance();
            err.handle(e);
        // Handle any other exception
        } catch (const std::exception &e) {
            auto &err = error_handler::get_instance();
            err.handle(e);
        }
    }
}
```

Error handling



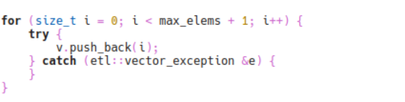
```
void
error_handler::handle(const satnogs::comms::exception &e)
{
    log(e);
    switch (e.get_severity()) {
        case sc::exception::severity::CATASTROPHIC:
        case sc::exception::severity::CRITICAL:
            system_reboot();
            break;
        case sc::exception::severity::MAJOR:
            if (m_last_errno == e.get_errno()) {
                m_errno_cnt++;
            } else {
                m_last_errno = e.get_errno();
            }
            if (m_errno_cnt > CONFIG_MAX_MAJOR_ERRORS) {
                system_reboot();
            }
            break;
        default:
            break;
    }
}
```

But exceptions makes the code slow right?



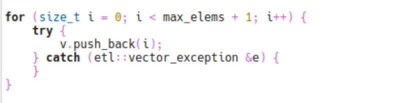
```
for (size_t i = 0; i < max_elems + 1; i++) {
    if (v.available()) {
        v.push_back(i);
    }
}
```

Took 25 ticks



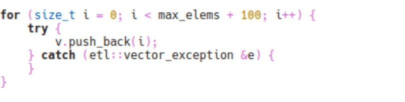
```
for (size_t i = 0; i < max_elems + 1; i++) {
    try {
        v.push_back(i);
    } catch (etl::vector_exception &e) {
    }
}
```

Took 24 ticks!



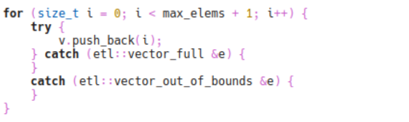
```
for (size_t i = 0; i < max_elems + 1; i++) {
    try {
        v.push_back(i);
    } catch (etl::vector_exception &e) {
    }
}
```

Took 24 ticks



```
for (size_t i = 0; i < max_elems + 100; i++) {
    try {
        v.push_back(i);
    } catch (etl::vector_exception &e) {
    }
}
```

Took 94 ticks!



```
for (size_t i = 0; i < max_elems + 1; i++) {
    try {
        v.push_back(i);
    } catch (etl::vector_full &e) {
    }
    catch (etl::vector_out_of_bounds &e) {
    }
}
```

Took 24 ticks!

More info

- #satnogs-comms:matrix.org
- gitlab.com/librespacefoundation/satnogs-comms
- https://libre.space
- info@libre.space

See you at the booth!

Come and visit our booth at K Level 2!

Swag available!