INTRODUCTION

Software-defined radio (SDR) is already widely adopted in ground applications and currently present in many consumer devices such as mobile transceivers and car radios. Currently, also in aerospace applications, SDR is slowly finding its place, albeit a much lower adoption and integration rate can be observed compared to consumer devices.

The OPS-SAT spacecraft is a 3U CubeSat launched by the European Space Agency (ESA) on December 18, 2019, and is an open hardware/software innovation platform in low Earth orbit (LEO). It was launched into a 515-km Sun-synchronous orbit together with CHEOPS from Kourou on a Soyuz launcher operated by Arianespace. The platform consists of several advanced payloads, including a fine Attitude Determination and Control System, X-band transmitter, optical receiver, and SDR payload. The heart of the satellite is the Satellite Experimental Processing Platform (SEPP), which consists of two Cyclone V system-on-chip units in a cold redundancy configuration running Embedded Linux. A typical stack of embedded software applications is available on the SEPP, ranging from Java and Python 3, down to running low-level C applications together with an application programming interface (API) to interact with the various payloads [1]. Experimenters coming from European academia and industry write proposals for experiments on OPS-SAT, and get selected after which the development process begins [2]. The purpose of OPS-SAT is to break the “has not flown will not fly” boundary present in space operations. The spacecraft is illustrated in Figure 1.

The spacecraft bus, payloads, and structure are designed in a way such that experiments can be carried out safely. The latter includes the addition of power and data bus switches in order to completely isolate a unit from the satellite internally. In addition, the onboard software used to run the satellite bus and process telecommands (TC) and send telemetry (TM) is running on a completely separate dedicated unit. OPS-SAT is controlled from the European Space Operations Centre (ESOC), from which also all the experiments are coordinated. Commissioning of the satellite was completed in late 2020 and the mission is currently in the operational phase, having already realized various experiments for several partners, including the French Space Agency CNES and Airbus [3]. The purpose of the mission is to demonstrate reconfigurable payloads and validate new concepts in space operations. Hence, this in-orbit demonstration (IOD) was proposed to test and demonstrate the versatility of the OPS-SAT RF platform. The concept of reprogramming an RF subsystem or transponder to support a variety of operating modes can have many advantages. Combining SDR with open-source signal processing libraries such as GNU Radio results in a powerful space-borne RF experimentation platform. While the mission is already using GNU Radio routinely in the ground segment [4], this article covers the first experiment that uses the SDR payload together with in-orbit RF signal processing using GNU Radio. The goal of the demonstration is to develop a software-configurable search and rescue transponder on the satellite that is capable of receiving and decoding emergency transmissions from terrestrial distress beacons and relaying them to mission control (MC) at ESOC. The Special Mission Infrastructure Laboratory Environment (SMILE) facility at ESOC is used to operate the satellite. SMILE has a dedicated S/X-band tracking antenna and
several clusters of reconfigurable servers running baseline ESA ground operations software as well as custom application stacks.

**METHOD**

To demonstrate the use of the SDR payload of OPS-SAT with a real-life scenario, the COSPAS-SARSAT system was chosen, a global network of ground stations and transponder instruments on satellites that pick up and relay distress transmissions originating from terrestrial emergency beacons. The system allows detection and relay of emergency transmissions via satellites. Most ships and aircraft are required to carry such a beacon, also known as an emergency position indicating radio beacon (EPIRB). The aviation equivalent is referred to as an emergency location transmitter (ELT), often mounted in the tail of the aircraft. Distress beacons can be activated automatically by a variety of means, including prolonged contact with salt water for an EPIRB and large G-forces for ELTs. Manual activation is also always possible. Several protocols exist for the actual message encoding, but most schemes encode basic information such as a registration number and approximate location acquired via an internal GNSS receiver.

Many satellites in low, medium, and geostationary orbits (LEO, MEO, and GEO) carry repeater instruments that perform a frequency shift of the received beacon transmissions in UHF, and downlink them in the L-band (1.5 GHz) to the local user terminals for on-ground processing of the beacons. Once a beacon activation is verified, it is forwarded to the nearest rescue coordination center. If no GPS information is sent in the beacon, time difference of arrival (TDOA) is used to determine a coarse position. As opposed to the on-ground signal processing that is employed in the operational COSPAS-SARSAT system, this IOD is concerned with the direct in-orbit processing, and decoding of beacon transmissions.

**CONCEPT OF OPERATIONS (CONOPS)**

The SDR payload and UHF monopole antenna of OPS-SAT are used to receive the terrestrial RF transmissions from beacons in the 406-MHz band at the experiment preprogrammed start time \( t_0 \). The recordings are digitally processed and files with signal metadata containing the encoded beacon messages are generated onboard. Upon acquisition of signal at \( t_1 \), the files are synchronized with the ground. The mission operations are performed via the mission timeline, which is the result of the mission planning system and is a preprogrammed list of time-tagged commands that shall be executed onboard and on-ground. This consists of the uplink windows, when the SDR will be recording, and processing beacons, as well as the timing of the downlinks. The timeline is loaded as XML files into the mission automation system and consists of timestamps and references to procedures stored in a database. A procedure consists of one or more TC to be sent to the satellite, with intermediate TM checks. TCs are sent and TM is received using the satellite control and operation system (SCOS), the generic MC software infrastructure developed and used by ESA. SCOS handles many TM/TC aspects in space missions and connects using TCP to the ground station modem via a proxy to exchange packets on the RF space link. File transfers are performed using the Consultative Committee for Space Data Systems.
(CCSDS) File Delivery Protocol (CFDP). The resulting files from completed CFDP transfers during a contact window with the satellite are forwarded via secure file transfer protocol to another server and contain the encoded messages that originated from the EPIRB/ELTs. The experiment ConOps is illustrated in Figure 2.

**ONBOARD PROCESSING**

The onboard process is detailed in Figure 3. After setting the carrier frequency and sample rate of the SDR, in-phase and quadrature (IQ) samples representing the real and imaginary components of the digitized RF signal are acquired via a 12-bit parallel bus and stored in the field-programmable gate array random access memory (RAM).

The API reads the samples using direct memory access and transfers them to the Linux userspace and high-level scripts in order to store the samples on a temporary filesystem. The samples are then read by an executable binary (burst decoder), which is dynamically linked with GNU Radio libraries in order to perform the signal processing. Two types of files are written to the onboard embedded multimedia card permanent storage, JSON and IQ-files. The JSON products contain decoded metadata from the transmissions, such as encoded message, burst frequency, and errors, while the decimated IQ-files contain the raw transmission for offline analysis. The raw RF input files stored in RAM that were originally captured are automatically deleted since they are no longer required. Finally, upon a command from the ground, the files are downloaded using CFDP through a controller area network bus, which links the main interface of the SEPP to the ground via the S-band TMTC encoder/decoder.

The next sections will elaborate further on the technical details of the beacon transmitter standards and how they were used to test and design a GNU Radio signal processor to run on the satellite.

**BEACON TRANSMISSION CHARACTERISTICS**

In order to implement a suitable GNU Radio signal processor for the beacon transmissions, the beacon protocol was first investigated using the published standards. Modern EPIRB/ELT transmissions occur in the protected 406-MHz band and are phase-modulated at 400 bits/second with a phase offset of 1.1 radians. Manchester encoding is used to encode the beacon information, resulting in a modulation rate of 800 baud. This encoding is used to ensure the presence of periodic transitions in the case of many identical bits. The periodic transitions in turn allow for a better symbol clock recovery. The latter is illustrated in Figure 4. Due to the biphase nature of the linecode, a residual carrier will
be present in the transmitted signal. A residual carrier in this case is desirable as it allows for precise frequency offset determination for coarse position estimation.

A typical beacon transmits a distress signal with a power of 37 dBm or 5 watts into a monopole antenna and consists of several periods. First, an unmodulated carrier is sent for 160 ms. In case the signal is very weak, multiple satellites can still estimate a coarse beacon position via TDOA using only the carrier. A bit synchronization pattern follows, consisting of 15 “1”s, resulting in 30 Manchester symbols. The next 9 bits form the frame sync of the transmission and can have two values depending on the activation type. The frame sync will consist of the sequence 000101111 for a regular transmission and 011010000 for a transmission in self-test mode. When a user performs a monthly test of the beacon to check functionality, the transmission will use the self-test frame sync in order to not signal a genuine distress to the authorities. If the beacon, however, is activated in an emergency, the normal frame sync will be used. The bit and frame sync form the preamble of the transmission and are used by the receiver for synchronization purposes. After the frame synchronization, a format bit follows indicating if the message is a long or a short message. Finally, the data field is transmitted, which has a length of 87 bits for a short message and 119 bits for a long message [5]. The data field contains encoded user registration information, nature of the distress, and GPS-position if available. The complete sequence is illustrated in Figure 5.

**BEACON DECODER IMPLEMENTATION**

The beacon or burst decoder (highlighted red in Figure 3) processes the captured raw RF samples and performs demodulation of beacon transmissions (bursts) present in the recording. While the modulation and framing are known, the time of the transmission as well as the exact frequency are not known. The following requirements are defined for the burst decoder:

- IQ-sample input in format cs16;
- logging of burst carrier frequency;
- logging of time-of-burst;
- preamble detection (bit + frame sync);
- Manchester detection and output in JSON format.

The burst decoder is a Linux executable that can be invoked from command line with arguments to perform the desired signal processing steps on an IQ input file. The executable uses GNU Radio shared libraries that are dynamically linked during compilation as well as runtime. The necessary GNU Radio signal processing blocks are imported using the API by means of C++ header files [6] and connected together, forming the final flow graph. The burst decoder can be split up in three main steps:

1) sample preprocessor;
2) phase lock and demodulator;
3) symbol timing recovery and decoding.

The following sections cover the details of each processing step as well as the design choices. The complete overview of the different stages in the burst decoder is given in Figure 6.
SAMPLE PREPROCESSOR

The sample preprocessor is used to filter out the frequency range of interest as well as lower the sampling rate. The SDR interface on OPS-SAT currently does not support streaming of IQ-samples, and therefore, measurements have to be logged to disk and then read. The 12-bit I and Q samples acquired from the SDR are padded to 16 bits and stored as files with the GNU Radio cs16 format. The latter indicates that the samples are stored as 16-bit signed integers. To use this format as source in the GNU Radio flow graph, an interleaved_short_to_complex block is used to convert the samples to the cf32 format. This format is commonly used in GNU Radio and stores the I and Q samples as 4-byte IEEE-754 floating-point values.

Finally, to adjust for scaling offsets, the I and Q samples are multiplied with a constant according to (1) that ensures the full-scale 12-bit signed IQ input values map to -1.0 and 1.0 when converted to cf32.

\[
\tilde{S}_{\text{cf32}} = \tilde{S}_{\text{cs16}} \times \frac{1}{2^{12} - 1} = \tilde{S}_{\text{cs16}} \times \frac{1}{2047} \tag{1}
\]

An IQ imbalance is present in the RF front end, causing a dc-offset as indicated in Figure 7 by a carrier at 0 Hz. To ensure the signals of interest appear in a spurious free part of the baseband, the SDR center frequency is tuned to a lower frequency. The carrier frequency offset (CFO) is chosen such that the target passband still appears within the baseband so that it can be frequency shifted. This is shown in Figure 7(a) where the blue area indicates the target passband. A CFO of -125 kHz was selected such that the actual tuned SDR center frequency becomes 406.025 MHz - 125 kHz = 405.9 MHz. The GNU Radio block freq_xlating_fir_filter_ccf is used to revert the frequency shift of 125 kHz to bring the passband back to baseband, which is illustrated in Figure 7(b). The resulting signal is then low-pass filtered using a finite impulse response (FIR) filter with a cutoff frequency equal to the Nyquist frequency of the new sample rate as a decimation is also performed afterward to reduce the sample rate from \(F_{s1}\) to \(F_{s2}\). The low-pass filter in this case functions as an antialiasing filter. The final settings of the preprocessor are given in Table 1.

The resulting preprocessed stream of complex samples with reduced sample rate is also stored in a file onboard for later downlink in case a burst is detected.

PHASE LOCK AND DEMODULATOR

Due to the high relative velocity of the spacecraft toward the ground, there will be a significant Doppler shift that needs to be taken into account. In order to synchronize to the burst transmissions, a phase-locked loop (PLL) is used to lock on the initial carrier of the beacon and downconvert the burst transmission to baseband for further processing. More specifically, the GNU Radio block pll_carriertracking_cc is used to perform this operation. The phase-locked and downconverted burst transmission is filtered with a secondary FIR filter, which further decimates the incoming stream to a rate that is closer to the actual symbol rate. An FIR with a root-raised cosine shape is used. The majority of phase information will reside in the quadrature (Q) component of the complex signal, while the in-phase (I) component mostly contains the power of the residual carrier due to the biphase-L encoding of the bitstream. The contents of the I and Q branches after PLL lock and downconversion are illustrated in Figure 8. The PLL locking can clearly be seen as only the real part of the signal (the I or in-phase part) contains power, while the imaginary part does not. Constant IQ-values also indicate a phasor with a fixed phase and amplitude and, hence, are expected after the PLL obtains a lock. As soon as the phase modulation frequency

Table 1.

| Name                      | Value       |
|----------------------------|-------------|
| Input sample rate \(F_{s1}\) | 1.5 Ms/s    |
| Output sample rate \(F_{s2}\) | 37.5 ks/s   |
| Carrier frequency offset   | -125 kHz    |
| LPF cutoff                 | 15 kHz      |
| Decimation                 | 40          |

Figure 7.

(a) Various sample processing stages showing the target passband highlighted in blue. (b) Frequency shifting. (c) Finally, the low-pass filtering and decimation. The frequencies shown are baseband frequencies; to obtain the RF carrier frequency, the configured SDR center frequency of 405.9 MHz has to be added.
of the preamble begins, the power swings back and forth between the positive and negative imaginary components following the two Manchester phase levels according to the biphase-L encoding scheme. Only the samples from the PLL Q-branch are passed down further as these data contain the phase and, therefore, the encoded message.

**SYMBOL TIMING RECOVERY AND DECODING**

Symbol recovery is done by feeding phase symbols to a clock_recovery_mm block, which performs symbol timing recovery according to the Mueller and Müller timing synchronization algorithm [7]. This GNU Radio block is provisioned with the estimated symbol rate of 800 symbols/second. The latter is an estimated rate since the sampling rate of the SDR is not exact. After symbol recovery, the output is also phase symbols in the f32 format, but resampled to 1 sample/symbol, hereafter called “soft symbols.” The soft symbols are converted to hard symbols by means of a binary slicer. This block converts any value equal or less than 0 to a hard 0, and any positive value to a 1. The conversion from soft symbols to hard symbols in the preamble sequence is illustrated in Figure 9.

The obtained logic levels of the hard symbols are still representing the Manchester symbols and not the actual bits. A differential decoder is used to map the Manchester symbol pairs to bits. The bit synchronization pattern consisting of 15 “1”s (30 Manchester symbols) can be seen in Figure 9. The decoding is performed in the C++ code and is implemented as a tag correlator where the hard symbols are shifted continuously through a synchronization register. This register is checked at every bit shift to see how well it correlates with the expected preamble sequences. As the phase information is relative to the carrier that the PLL locks to, the absolute phase of the transmission is not known. Therefore, it is also necessary to correlate against the inverted preamble, resulting in the following four possibilities:

- (15x 1) + 000101111 (normal polarity and normal mode);
- (15x 0) + 111010000 (inverted polarity and normal mode);
- (15x 1) + 011010000 (normal polarity and self-test mode);
- (15x 0) + 100101111 (inverted polarity and self-test mode).

If there is a match between the contents of the synchronization register and one of the four possible preambles, then a transmission is detected, after which the subsequent symbols are shifted into a separate message register. Once the message register is filled, it is differentially decoded and the output logged.

**DEPLOYMENT**

Prior to code uplink to the satellite on-orbit, a validation is performed on the engineering model (EM). This consists of uplinking the application code with the same file transfer pipeline that will be used during a pass with the satellite. Time-tagged commands are sent to the EM, and the SEPP resource usage is monitored live using the MCS. This process is shown in Figure 10.

GNU Radio version v3.7.13.4 in a headless configuration is cross-compiled for the arm-linux-gnueabihf architecture, packaged into compressed archives, and is uploaded to the satellite together with the compiled GNU Radio flow graph. The archives containing the shared GNU Radio libraries are available to other future users as
well since they are placed in /usr/local/lib after installation on the satellite.

**ORBIT TEST CAMPAIGN AND RESULTS**

The COSPAS-SARSAT system comprises a worldwide network of beacon reference stations. The ground stations transmit periodic bursts with a fixed frequency and repetition rate. The reference beacons are used to monitor the system performance since the geographical location of the emitters is known. The reference beacons are used to test the onboard implementation since the encoded content is known in advance and can be cross-correlated afterward. For the orbital test campaign, four stations were selected due to their high transmission cadence of 30 seconds, i.e., a beacon is transmitted every 30 seconds. The reference station transmissions are identical in terms of power and modulation compared to a user EPIRB or ELT. The live operational status and additional details of each station can be consulted online [8]. The chosen reference sites are given in Table 2 and graphically represented in Figure 11.

The reference stations are located at extreme latitudes in order to maximize the total daily contact duration between transmitter and satellite and, hence, will be in LOS with the beacon for most of the orbits. In coordination with mission planning, the SDR application is started when the satellite is about to enter the visibility footprint of the various stations. The experiment is started via previously uplinked time-tagged TC stored in the onboard queue. Once started, the system will listen for beacon transmissions for 9 minutes and terminate afterward. A total of 11 passes were scheduled of which six decoded one or more beacon transmissions. This success rate is mainly due to the dead time of the SDR in between sample capturing runs since the payload interface does not yet support sample streaming. All decoded beacons are given in Table 3. The synchronization error count indicates the number of incorrect symbols detected in the preamble. The decode error shows the number of occurrences where there are three or more subsequent identical symbols. In Manchester encoding, only two subsequent identical symbols can occur. A possible cause for a good synchronization but high-decode error count is loss of PLL lock due to signal power fading.

The metadata for the highlighted beacon in Table 3 as generated onboard in a JSON product is illustrated in Listing 1. The full beacon message is contained in the msg_bits field. Since the most significant bit of the message (format bit in Figure 5) indicates the length of the data field (87 or 119 bits), the hexadecimal sequence needs to be left shifted by 1 bit after which the remaining bit sequence matches the station ID for the transmitter located at McMurdo given in Table 2.

| Site       | Latitude (°) | Longitude (°) | Station/Beacon ID        |
|------------|--------------|---------------|--------------------------|
| Longyearbyen | 78.228983    | 15.3955       | A0234BF8A7335D0          |
| Thule      | 77.46475     | -69.217217    | 9B62197CA703500          |
| Kerguelen  | -49.3515     | 70.256        | 9C7FEC2AADC3590          |
| McMurdo    | -77.846033   | 166.71178     | ADC389B8AA1B9D0          |
Upon detection of a beacon, the decimated IQ output file after the preprocessor is stored onboard for downlink to allow for further analysis on the ground. The filename of this file is also stored in the JSON product under the `filename` key to allow tracing of beacon detections across the various artifacts. The decimated IQ-files are in the `cf32` format and can be viewed directly in GNU Radio as well. Figure 12 illustrates the fast Fourier transform (FFT) of the burst transmission contained in the IQ-file that was downloaded from the satellite shows the residual carrier and the phase modulation power in the sidebands.

![Signal power](image)

**Figure 12.** Averaged FFT of the received beacon uplink, processed from the downloaded IQ-file from the satellite.

The bold entry in the table is used as an example in the text to further explain how its values relate to the code in Listing 1.

Upon detection of a beacon, the decimated IQ output file after the preprocessor is stored onboard for downlink to allow for further analysis on the ground. The filename of this file is also stored in the JSON product under the `filename` key to allow tracing of beacon detections across the various artifacts. The decimated IQ-files are in the `cf32` format and can be viewed directly in GNU Radio as well. Figure 12 illustrates the fast Fourier transform (FFT) of the burst transmission contained in the IQ-file that was downloaded from the satellite shows the residual carrier and the phase modulation power in the sidebands.

![Signal power](image)

**Figure 12.** Averaged FFT of the received beacon uplink, processed from the downloaded IQ-file from the satellite.

The bold entry in the table is used as an example in the text to further explain how its values relate to the code in Listing 1.

| UTC time | Burst frequency (Hz) | Message (format bit + data field) | Sync err | Decode err | Station LOS       |
|----------|----------------------|-----------------------------------|----------|------------|-------------------|
| 2021-06-15 00:42:25 | 406013952 | 1411A5FC5399AE857D5D8002002002C01 | 1         | 21         | Thule + Longyearbyen |
| 2021-06-15 01:37:50 | 406030144 | 4E3FF6155668AC86E79580 | 0         | 6          | Kerguelen         |
| 2021-06-15 01:39:20 | 406027552 | 4E3FF6155668AC86E79580 | 0         | 0          | Kerguelen         |
| 2021-06-15 02:13:11 | 406029632 | 1411A9FC5381AE857D5D80 | 2         | 5          | Thule + Longyearbyen |
| 2021-06-15 02:14:42 | 406021984 | 5011A5FC5399AE857D5D80 | 0         | 0          | Thule + Longyearbyen |
| 2021-06-15 02:15:21 | 406028288 | 1021F14B1878C81A2404E5A2ECAFO | 0         | 5          | Thule + Longyearbyen |
| 2021-06-15 02:17:18 | 406028384 | 4DB10CBF08FC07F800223E | 0         | 22         | Thule + Longyearbyen |
| 2021-06-15 03:12:49 | 406023424 | 4EBFF61556692C86E795A0 | 0         | 1          | Kerguelen         |
| 2021-06-15 04:32:53 | 406035424 | 56E1C4DC550DCE801D130F | 0         | 0          | McMurdo           |
| 2021-06-15 09:19:53 | 406028576 | 56E104DC550DCE801D130F | 0         | 5          | McMurdo           |
| 2021-06-15 09:24:25 | 406029248 | D66134FC0308C086A0478000000000000 | 2         | 12         | McMurdo           |

**Table 3.**

| UTC time | Burst frequency (Hz) | Message (format bit + data field) | Sync err | Decode err | Station LOS |
|----------|----------------------|-----------------------------------|----------|------------|-------------|
| 2021-06-15 00:42:25 | 406013952 | 1411A5FC5399AE857D5D8002002002C01 | 1         | 21         | Thule + Longyearbyen |
| 2021-06-15 01:37:50 | 406030144 | 4E3FF6155668AC86E79580 | 0         | 6          | Kerguelen |
| 2021-06-15 01:39:20 | 406027552 | 4E3FF6155668AC86E79580 | 0         | 0          | Kerguelen |
| 2021-06-15 02:13:11 | 406029632 | 1411A9FC5381AE857D5D80 | 2         | 5          | Thule + Longyearbyen |
| 2021-06-15 02:14:42 | 406021984 | 5011A5FC5399AE857D5D80 | 0         | 0          | Thule + Longyearbyen |
| 2021-06-15 02:15:21 | 406028288 | 1021F14B1878C81A2404E5A2ECAFO | 0         | 5          | Thule + Longyearbyen |
| 2021-06-15 02:17:18 | 406028384 | 4DB10CBF08FC07F800223E | 0         | 22         | Thule + Longyearbyen |
| 2021-06-15 03:12:49 | 406023424 | 4EBFF61556692C86E795A0 | 0         | 1          | Kerguelen |
| 2021-06-15 04:32:53 | 406035424 | 56E1C4DC550DCE801D130F | 0         | 0          | McMurdo |
| 2021-06-15 09:19:53 | 406028576 | 56E104DC550DCE801D130F | 0         | 5          | McMurdo |
| 2021-06-15 09:24:25 | 406029248 | D66134FC0308C086A0478000000000000 | 2         | 12         | McMurdo |

The bold entry in the table is used as an example in the text to further explain how its values relate to the code in Listing 1.

```json
{
  "beacon": {
    "filename": "dr_iq,20210615_043251_405900000_ 15000000_06_375000pS.cf32",
    "freq_hz": 406035424,
    "payload": {
      "error_n": 0,
      "msg_bits": "56E1C4DC550DCE801D130F",
      "msg_len": 88,
      "symbols": "6669A956A56A6A56665A6A995556A 6565A55AAB49C33DB369C3"}
  },
  "sample_n": 846,
  "sync": {
    "error_n": 0,
    "inverted_bit": true,
    "type": "beacon_normal_inverted"
  },
  "time": 1.057500008476*3716
}
```
CONCLUSION

We demonstrate the development of a GNU Radio application and the porting of it to an operational environment for onboard deployment on OPS-SAT. In-orbit processing of the RF samples significantly reduced the amount of data that needed to be downlinked, in turn increasing the mission efficiency. While the major limitation is currently the interface to the SDR, resulting in noncontinuous sample capturing, the results are nevertheless encouraging. The successful IOD activity indicated that the RF front end on OPS-SAT is suitable for experiments in the 406-MHz search and rescue band. This experiment has also shown the capability of OPS-SAT to serve as a flexible platform for SDR experimentation for other potential applications ranging from remote sensing to LoRa and other low-power Internet of Things (IoT) applications, bridging the gap between satellite operations and the IoT era. A multitude of experiments have already been submitted that aim to utilize this new in-orbit infrastructure for digital signal processing and RF experimentation in space.

REFERENCES

[1] D. Marszak et al., “OPS-SAT—Opening a satellite to the Internet,” in Proc. 12th IAA Symp. Small Satellites. International Academy of Astronautics, 2019, pp. 1–3.
[2] D. Evans, D. Marszak, T. Mladenov, G. Labrèche, V. Zelenevski, and V. Shiradhonkar, “OPS-SAT—The world’s first satellite accessible by the public over the Internet,” in Proc. 16th Int. Conf. Space Operations, 2021, pp. 6–8.
[3] European Space Agency, “Interplanetary Internet & cameras in space: ESA’s OPS-SAT first results,” 2021. Accessed: Feb. 24, 2021. [Online]. Available: https://www.esa.int/Enabling_Support/Operations/Interplanetary_internet_cameras_in_space_ESA_s-OPS-SAT_first_results
[4] T. Mladenov, B. Fischer, and D. Evans, “ESA’s OPS-SAT mission: Powered by GNU radio,” in Proc. GNU Radio Conf. 2020, vol. 5, no. 1. Accessed: Feb. 24, 2021. [Online]. Available: https://pubs.gnuradio.org/index.php/grccon/article/view/65
[5] “International COSPAS-SARSAT Programme Specification for COSPAS-SARSAT 406MHz distress beacons,” c/s T.001 issue 4-revision 8, 2021. Accessed: Jun. 9, 2021. [Online]. Available: https://cospas-sarsat.int/images/stories/SystemDocs/Current/T001-JUN-7-2021.pdf
[6] “GNU Radio project GNU radio manual and C API reference,” v3.7.13.4, 2021. Accessed: Mar. 18, 2021. [Online]. Available: https://www.gnuradio.org/doc/doxygen-v3.7.13.4/
[7] K. Mueller and M. Muller, “Timing recovery in digital synchronous data receivers,” IEEE Trans. Commun., vol. 24, no. 5, pp. 516–531, May 1976.
[8] “International COSPAS-SARSAT Programme MEOSAR reference beacon status,” 2021. Accessed: Jun. 3, 2021. [Online]. Available: https://cospas-sarsat.int/en/system/meosar-system-status/meosar-reference-beacon-status