The Sapphire Telemetry System

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.1	September 2013		A

Contents

1 Overview

The Sapphire Telemetry System is a telemetry downlink system for the Sapphire test rocket by Copenhagen Suborbitals. Its purpose is to support downlink of high rate telemetry from the GNC and AAU units allowing detailed analysis of the flight.

Figure 1 shows the telemetry downlink system in context of the complete data infrastructure during the Sapphire mission. Figure 2 shows an overview of the telemetry system itself.

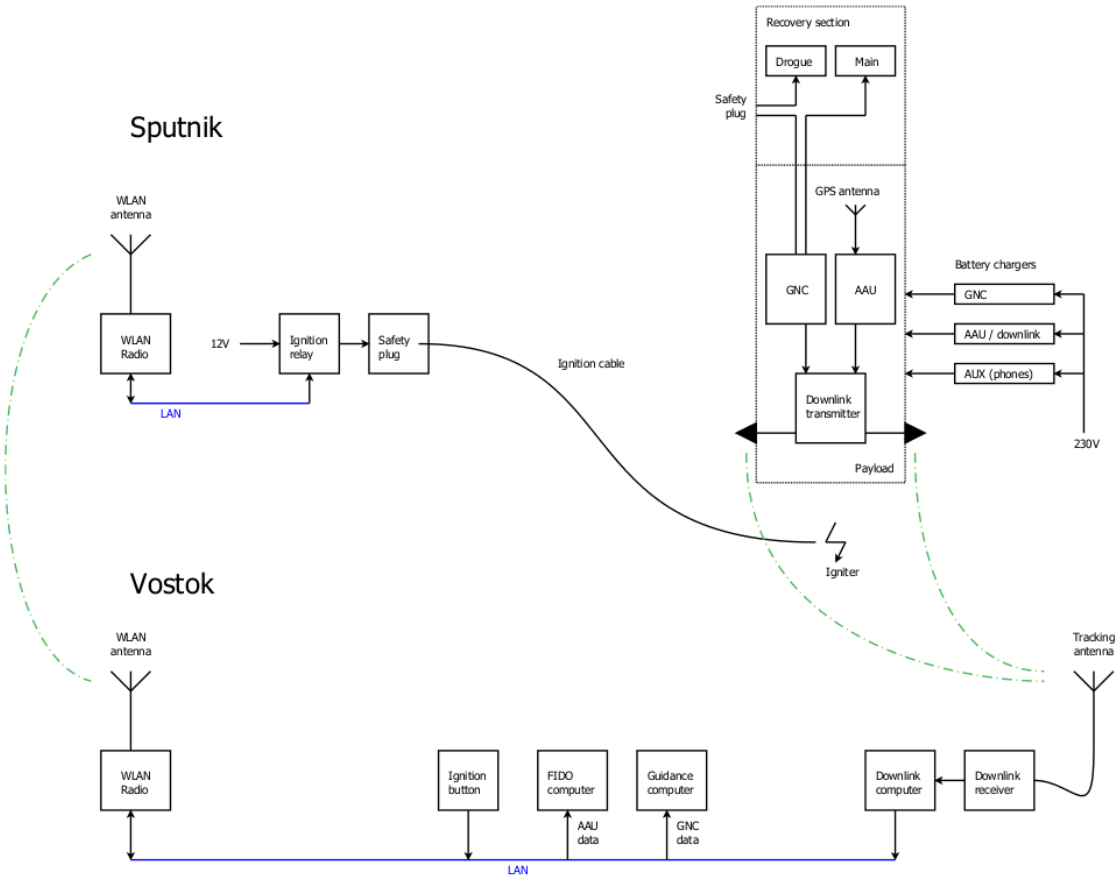

Figure 1: Sapphire data infrastructure

The telemetry system is based on a design originally created for the Euroluna Romit 2 cubesat and has the following characteristics:

- Two downlink channels working simultaneously.
- Works in the 13 cm amateur / ISM band.
- 1 watt RF output per channel (adjustable in firmware)
- Up to 2 Mbps bit rate per channel (including all headers, FEC, etc.)
- GFSK modulation
- CRC and FEC (rate 1/2)

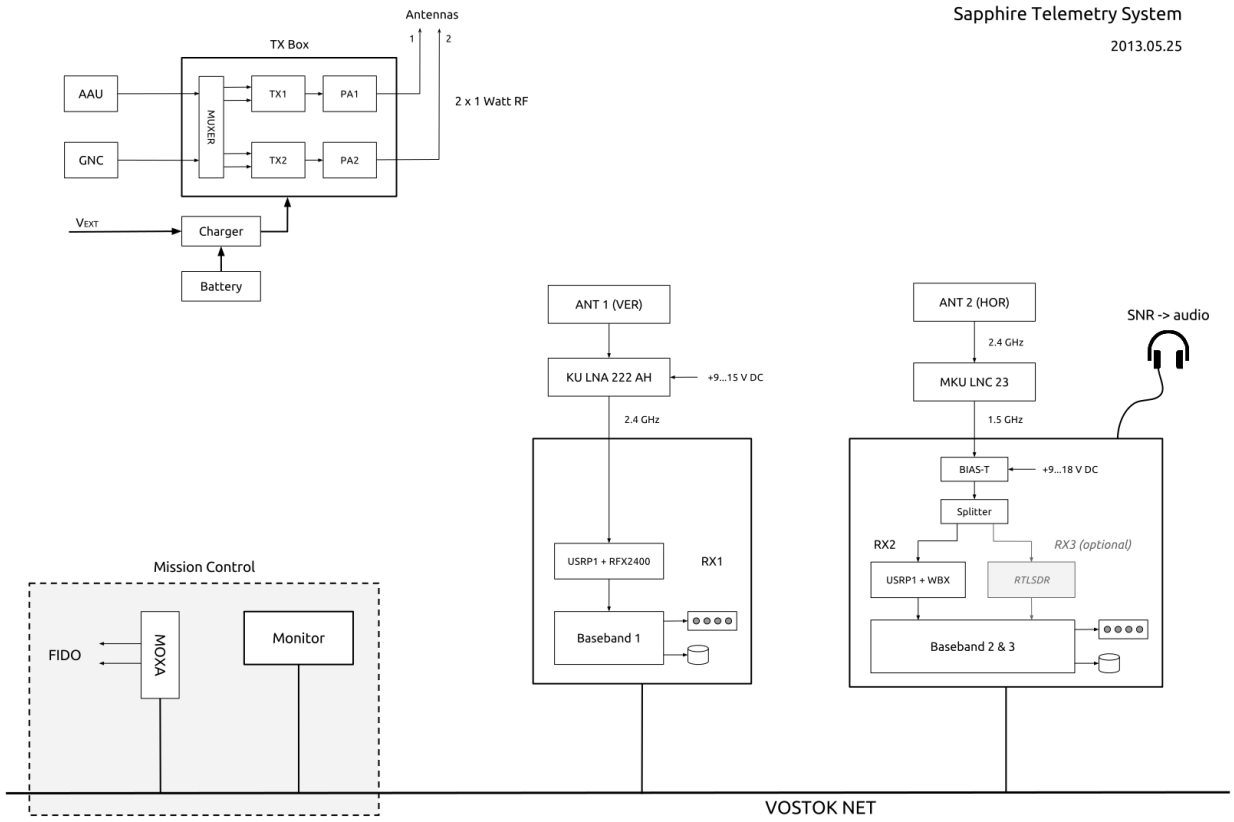

Figure 2: Telemetry system overview

The telemetry system contains the following components:

Transmitter
The transmitter is based on the ADF7242 transceiver chip running in raw mode. The digital data interface to the payload goes through a PIC24, which is responsible for converting the potentially variable rate data to fixed rate. The transmitter is described in more details in Section 2.

Receiver
The receiver is a software defined radio using USRP hardware and GNU Radio based software receiver. It runs on a Linux PC and delivers the telemetry data via Ethernet or RS232 interface. The receiver is set up to run autonomously and does not require any user intervention; however, key parameters are exported via a control port interface. This allows remote clients to connect to the receiver, monitor the system performance, and modify some key parameters. The receiver is described in details in Section 3.

Control panel
The monitoring and control panel is a client that connects to the receiver and displays system performance (spectrum, SNR, data and error rates, etc.) to mission control crew. It also allows modifying some basic settings in the receiver chain such as frequency, active channel, recording on/off, etc. The control panel is described in more details together with the receiver in Section 3.

Tracking tower
An AZ/EL mount built specifically for tracking fast flying objects. It can carry many antennas of different shapes and sizes as long as the construction stays balanced. For the Sapphire mission it carries two 2.4 GHz yagi antennas, one vertically

and horizontally polarized. Each antenna is connected to a receiver that receives both downlink channels; however, only one channel of the horizontal receiver is monitored in real time. The other channels are recorded for reference. The tracking tower has been described in TBD.

In addition to the unit documentation listed above, following system level documentation exists:

Packet format
Describes the packet format used by the telemetry system, see Section 4.

Forward error correction
Describes the error correction code used by the system, see Section 5.

2 Telemetry transmitter

The transmitter is based around the ADF7242 FSK transceiver chip by Analog Devices. The transceiver is used in raw FSK mode as opposed to any of the pre-programmed modes in the chip. Raw FSK mode allows up to 2 Mbps data rate.

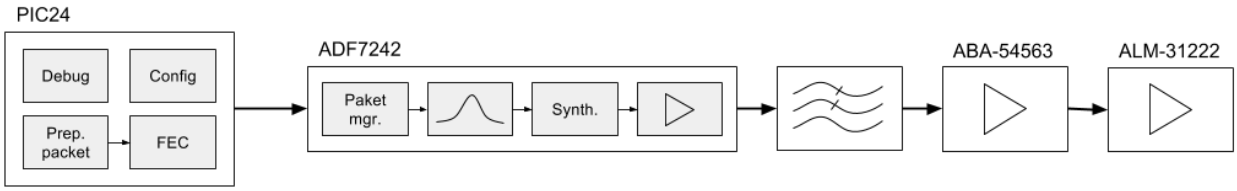

Figure 3: Functional diagram of the telemetry transmitter.

The transmitter is connected to the GNC and AAU unit using a serial interface. This interface is provided by a PIC24 microcontroller which is in charge of all the data handling, including preparing the packets, calculating CRC and adding FEC. Moreover, the PIC24 is also responsible for configuring and controlling the ADF7242 transceiver chip.

For the Sapphire mission we were using 250 kbps bit rate with rate 1/2 FEC, i.e. 125 kbps data throughput. The 125 kbps data includes packet headers and other bookkeeping data. See Section 4 for details about the packet format.

The ADF7242 transmitter is running in GFSK mode. This eliminates the minimum shift restriction of FSK and reduces the occupied bandwidth (the transmitter uses 260 kHz shift in our case). On the other hand this also introduces inter-symbol interference that must be compensated for at the receiver end.

The ADF7242 transceiver chip is followed by a low pass filter, an ABA-54563 buffer amplifier and finally an ALM-31222 power amplifier capable of delivering 1W RF into the antenna.

3 Telemetry receiver

The Sapphire telemetry receiver is based on a Universal Software Radio Peripheral (USRP) hardware from Ettus Research and the GNU Radio software defined radio framework.

As indicated on Figure 2 in Section 1 we designed the system to use three receiver chains running parallel. One for receiving each telemetry channel using a USRP and a third one for testing and RTL-SDR receiver. Due to time constraints, we ended up using one USRP-based receiver for real time reception and one USRP-based receiver for recording the I/Q data to disk. The RTL-SDR experiment was not performed.

Figure 4 below shows an overview of the receiver used for real time reception.

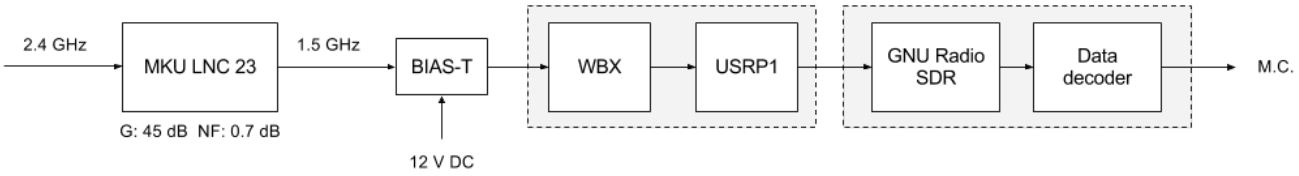

Figure 4: Sapphire telemetry receiver

It uses an MKU LNC 23 down-converter from Kuhne Electronic, a USRP1 equipped with a WBX transceiver daughter card and connected to a PC running software receiver and data decoder. Each of these components are described in the following sections.

3.1 MKU LNC 23 down-converter

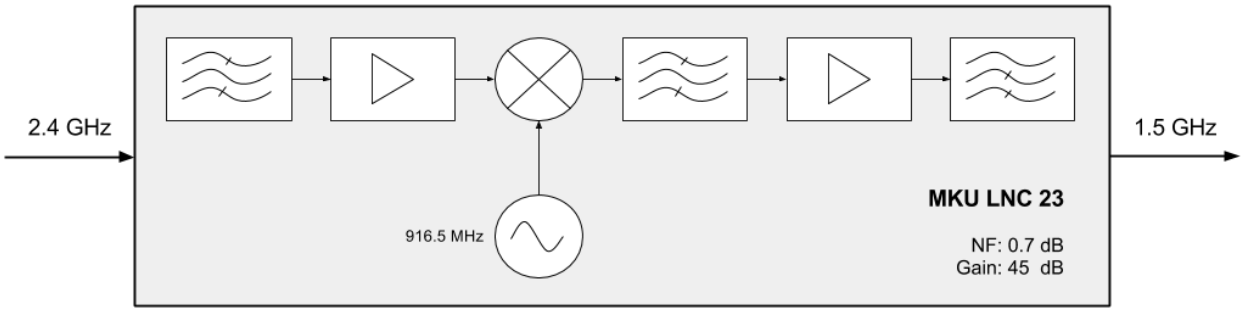

Figure 5: Kuhne MKU LNC 23 down-converter

The MKU LNC 23 down-converter from Kuhne Electronic is mounted directly on the antenna and connected using a very short coaxial cable. This gives the primary receiver chain a low noise figure and provides sufficient gain to run a long coax cable down to the receiver, which is located below deck.

The down-converter has built-in bias tee and is supplied through the coax. The supply voltage is injected into the coax using a bias tee at the USRP end below the deck.

3.2 USRP with WBX front end

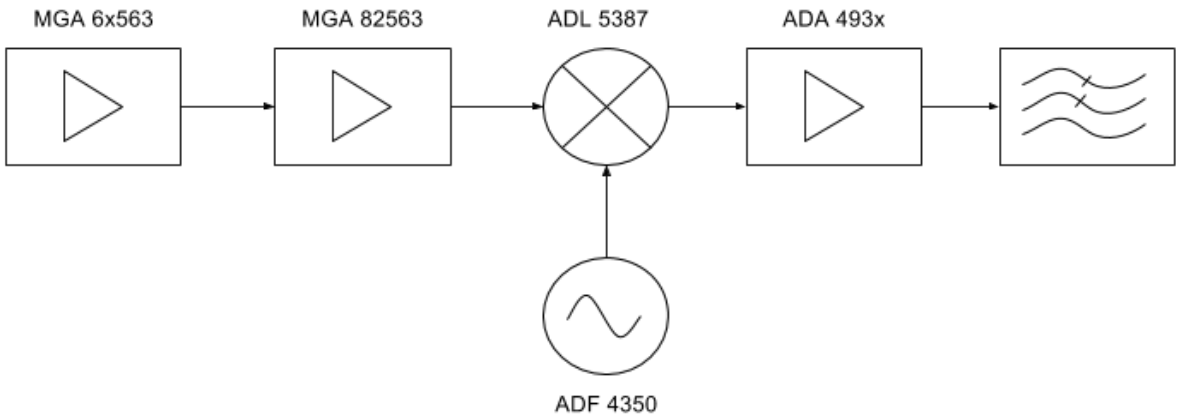

Figure 6: WBX receiver

The WBX transceiver board covers 50 MHz to 2.2 GHz and it has one of the best performing receivers for the USRPs. Figure 6 shows a high level diagram of the receiver part.

The board has two amplifier stages (MGA62563 and MGA82563) followed by the ADL5387 quadrature demodulator. The analog I/Q goes through an ADC driver and an anti alias filter with cut-off at around 50 MHz before it is routed to the ADC on the USRP main board.

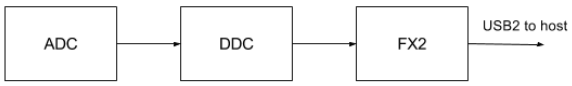

Figure 7: USRP1 receiver

The ADC on the USRP1 runs at 64 MHz and could theoretically provide 64 MHz of bandwidth. However, since the USRP1 is connected to the host PC through USB2, the usable real-time bandwidth is limited to 8 MHz at 16 bit resolution. To that end, the ADC is followed by an FPGA running a digital down converter (DDC) providing a means to convert the 64 MHz to a lower rate. The USRP1 supports a wide variety of sample rates between 250 kHz and 8 MHz. For the Sapphire mission we have been using 4 MHz bandwidth allowing us to monitor both transmitters at the same time.

Schematics of the USRP1 and the WBX RF board are included in the receiver/hardware/ directory.

3.3 Software receiver

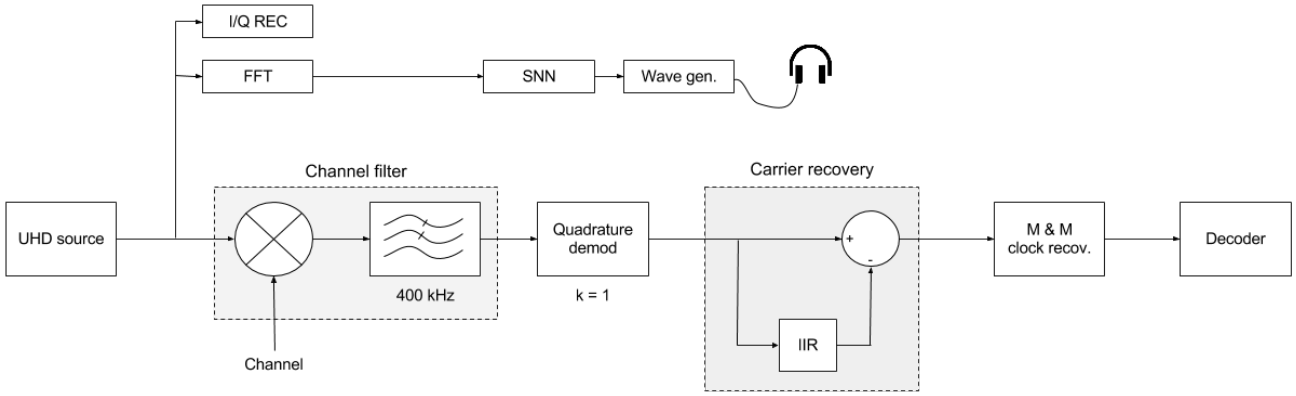

Figure 8: GNU Radio software receiver

The software receiver is a C++ application built using the GNU Radio SDR framework. The receiver uses both DSP blocks and the run-time framework provided by GNU Radio. In addition to that, the telemetry receiver also takes advantage of the control port interfaces, which is one of the latest additions to GNU Radio.

Most of the signal processing is executed by the GNU Radio scheduler and runs within that context. There are a few exceptions, namely the FFT block and the SNN calculations. These blocks are run in a separate boost thread scheduled periodically at 25-50 Hz. Since these are user interface related processing, there is no need to run them at higher rates.

3.3.1 Signal processing blocks

UHD source

The UHD source block provides the interface to the USRP hardware. We chose to use gr-uhd as input source for the Sapphire mission because we only needed USRP interface. A gr-osmosdr source could be used instead with minor modifications.

FFT

This blocks performs complex FFT on the input spectrum. The FFT is performed in three stages: 1. The first stage is a block in the GNU Radio flow graph that stores the incoming samples in a circular buffer. 2. The second stage fetches the last N samples from the circular buffer (N is the FFT size), performs the FFT calculations and prepares the data for presentation (scales to dBFS and translates the spectrum). This function is executed periodically at 25-50 Hz outside of the GNU Radio scheduling. 3. The third stage delivers the latest FFT data to external clients through the gnuradio-controlport interface.

I/Q recorder

This block is a simple file sink that dumps the complex I/Q samples to a file.

We found that it is rather problematic to save I/Q files at high rate and perform real time signal processing at the same time on the same computer. When recording is enabled, the operating system will start caching until it runs out of available RAM. Once it's out of RAM it will start writing data to the disk periodically. The amount of data that needs to be written is huge and SDR processing is suspended during the write. Although the dropouts are very brief in duration, they are quite frequent and lead to loss of samples. Consequently, the correctly received packet rate dropped from 99% to 60%.

There are several things one can do to improve the situation:

1. Use faster hardware optimized for continuous disk I/O.
2. Get more RAM and use a RAM disk if possible. This can't be used for long duration recordings.
3. Tune the virtual memory parameters of the kernel. We found that following parameters allowed for continuous recording on a low end Dell server:

```
sysctl -w vm.swappiness=0
sysctl -w vm.dirty_background_bytes=1048576
sysctl -w vm.dirty_writeback_centisecs=20
sysctl -w vm.dirty_expire_centisecs=30
```

The above settings pretty much disables caching and are not recommended for daily use on desktop systems. Use "vmstat 1" for monitoring how the system is caching. Documentation for the Linux kernel VM settings is available in the sysctl/vm document.

SNN & tracking

The SNN block calculates the (S+N)/N ratio where S stand for signal power and N for noise power in dB. The SNN is converted to an audio tone by the wave generator block and is used as an audio aid for tracking. The higher the tone the stronger the signal.

The SNN calculation is based on the FFT. Therefore, the SNN and wave generator blocks are scheduled by the same thread as the FFT processing outside of the GNU Radio main loop.

Frequency translating FIR filter

The frequency translating FIR filter has three functions:

1. Translate the spectrum so that the signal of interest is at 0 Hz.
2. Perform low pass filtering.
3. Decimate the data to a lower rate.

The frequency translation allows us to perform software-based tuning within the received spectrum. We use this functionality to select which downlink channel to decode i real time, see e.g. Figure 11.

The cut-off frequency of the low pass filter is initially set to 400 kHz but it can be adjusted during run time. The transition width is set to be equal to the cut-off frequency. Note that a 400 kHz cut-off corresponds to an 800 kHz wide channel when we are filtering at complex baseband.

The decimation factor is set to 2 so the spectrum after the filter is 2 MHz wide.

Demodulator

The transmitter uses frequency shift keying followed by a Gaussian filter to eliminate the minimum shift requirement and reduce the bandwidth. On the receiver end we can use a standard quadrature demodulator followed by a matching filter to compensate for some of the inter-symbol interference introduced by the Gaussian shaping of the signal.

The quadrature demodulator block takes a parameter, which determines the modulation factor. Sometimes it's called sensitivity, sometimes it's called gain — we just call it k. In an analog system k is the ratio between the quadrature rate and the maximum deviation (and reversed for the modulator). In a digital system this translates to samples per symbol assuming that the minimum shift condition is met.

For the Sapphire mission we have used $k = 1$, which is not optimal but sufficient thanks to the large margin we had in the link budget.

FIXME: Gaussian filter is missing in the Sapphire receiver.

Carrier recovery

Under optimal conditions the output from the quadrature demodulator will be ± 1 corresponding to 0 and 1. One of these optimal conditions is that the receiver is tuned to the exact same frequency as the transmitter. In practice this is rather difficult and expensive to accomplish and it is better to tolerate slight frequency offsets.

When using frequency modulation such an offset will cause the output levels from the demodulator to shift up or down by a constant and is therefore easy to compensate. We use a single pole IIR filter to calculate the running average of the output samples and correct for the offset before sending the samples to the clock recovery block. This has the advantage of being able to correct even for variable frequency drifts.

For the Sapphire mission we used alpha = 1e-3, corresponding to a time constant 0.5 ms.

Clock recovery

Our sample rate after the channel filter is reduced to 2 Msps. With a symbol rate of 250k this gives us 8 samples per symbol to work with. However, clock skew between the transmitter and the receiver may lead to this rate being different in practice, e.g. 7.998 or 8.01. The clock recovery block helps us determine this clock skew and select the "best" sample to be sent to the decoder.

The Mueller & Muller clock recovery algorithm requires only one sample per symbol and finds the correct timing by calculating an error term using the present and the previous symbol:

$e_n = (y'_n * y_{n-1}) - (y_n * y'_{n-1})$

where y refers to the symbol (+1 or -1) and y' refers to the actual sample (any float).

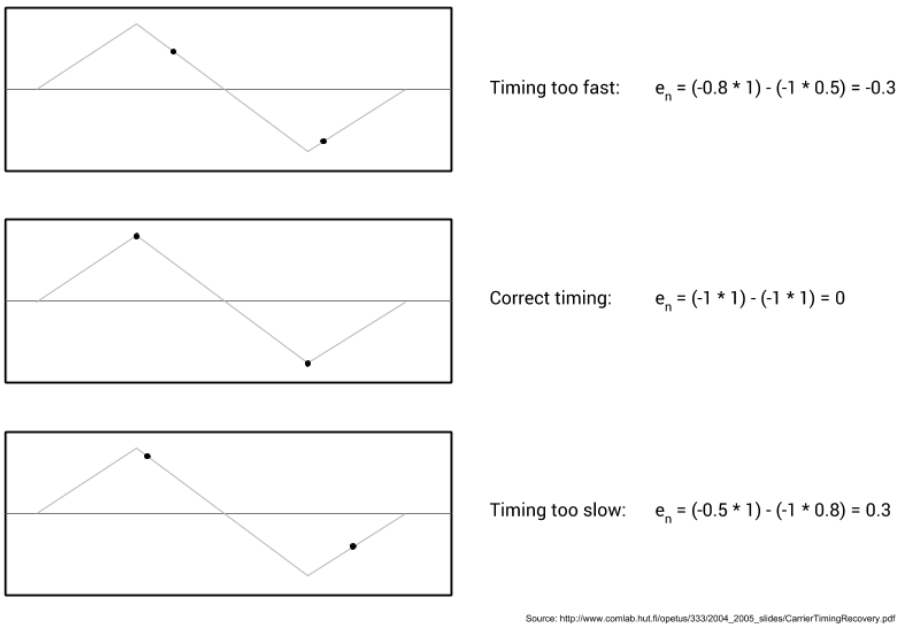

Figure 9: Mueller & Muller clock recovery

The Mueller & Mullerr clock recovery algorithm is sensitive to carrier offsets which is why we perform the carrier recovery described in the previous section.

3.3.2 Command line interface

The software receiver is a command line application with the following command line options:

```
alc@atlantis:~/sdr/stlm.git/receiver/build$ ./strx -h
linux; GNU C++ version 4.7.3; Boost_105300; UHD_003.005.003-140-gc099f2b5

Sapphire telemetry receiver 0.3
Command line options:
  -h [ --help ]         This help message
  -n [ --name ] arg     Receiver name (used for ctrlport)
  -i [ --input ] arg    USRP sub device or I/Q file (use file:/path/to/file)
  -a [ --ant ] arg      Select USRP antenna (e.g. RX2)
  -f [ --freq ] arg     RF frequency in Hz or using G, M, k suffix
  -g [ --gain ] arg     RF/IF gain in dB
  -l [ --lnb ] arg      LNB LO frequency in Hz or using G, M, k suffix
  -o [ --output ] arg   Output file (use stdout if omitted)
  --audio arg (=none)   Audio output device (e.g. pulse, none)
```

3.4 Data decoder

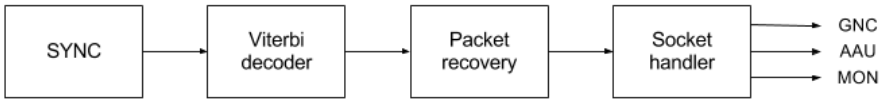

Figure 10: Data decoder

The data decoder runs as a separate process

First, the decoder looks for the sync bytes that each packet begins with c.f. Figure 12 in Section 4. Once sync is obtained the decoder begins running the bytes through the Viterbi decoder. Recall that we are using convolutional FEC and all bytes in a FEC frame are encoded.

The final step in the decoding process is the packet recovery, which consists of detecting the packet boundary, checking the packet length, the CRC, extracting the packet source and finally forwarding it to the respective user.

3.5 Monitoring and control

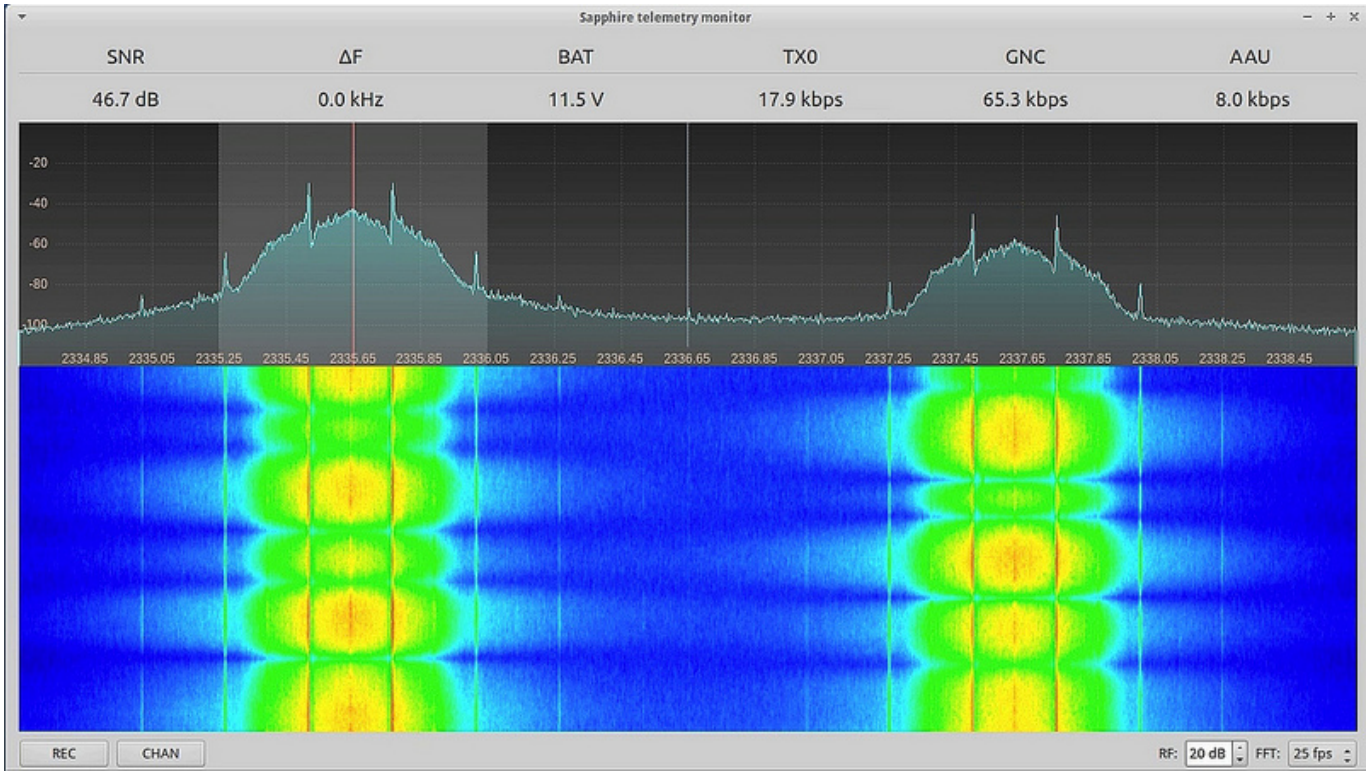

Figure 11: Sapphire telemetry monitor

Monitoring and control of the receiver is carried out using the Sapphire telemetry monitor. It is a simple Qt-based application called strx-mon, which connects to the receiver and the data decoder using network sockets.

Connection to the receiver is done through the gnuradio-controlport interface. Following interfaces are implemented for the Sapphire mission:

- FFT and waterfall plot of the receiver spectrum (4 MHz).
- Change FFT rate.
- Toggle between downlink channels.
- Signal to noise ratio (SNN actually) of the selected channel.
- Show and adjust filter bandwidth.
- Show and adjust USRP gain.
- Start and stop I/Q recording.

Connection to the data decoder is done through a raw TCP connection. Whenever a character is sent over the connection, the decoder will reply with:

- Decoded AAU telemetry in bytes.
- Decoded GNC telemetry in bytes.
- Decoded transmitter telemetry in bytes.

- Current transmitter ID.
- Current battery voltage.
- Transmitter uptime.

The telemetry monitor polls the decoder periodically and translates the received status into data throughput in kilobits per second.

You can watch the Sapphire telemetry monitor i action in this YouTube video showing a replay of the data downlink during the flight. The video is also a good demonstration of the telemetry system performance under harsh flight conditions (strong vibrations and tumbling).

4 Packet format and bitrate budget

The Sapphire telemetry is a continuous stream of packetized data. The data comes from the GNC and AAU units and packed into the frame structure described here. When neither GNC nor AAU are transmitting data, the transmitter will fill out the empty slots using own telemetry. This will ensure that the receiver can stay locked and synchronized to the signal even when there is no telemetry from the payload.

The Sapphire telemetry system uses a very simple packet format shown on the figure below:

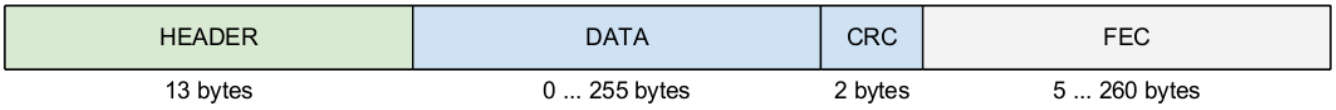

Figure 12: The Sapphire packet structure

A packet consists of the following parts:

- A fixed length packet header (11 bytes)
- Variable length data field (up to 255 bytes)
- CRC for the data field (2 bytes)
- FEC field that applies to parts of the header (last 5 bytes), the data field and the CRC

4.1 Packet header

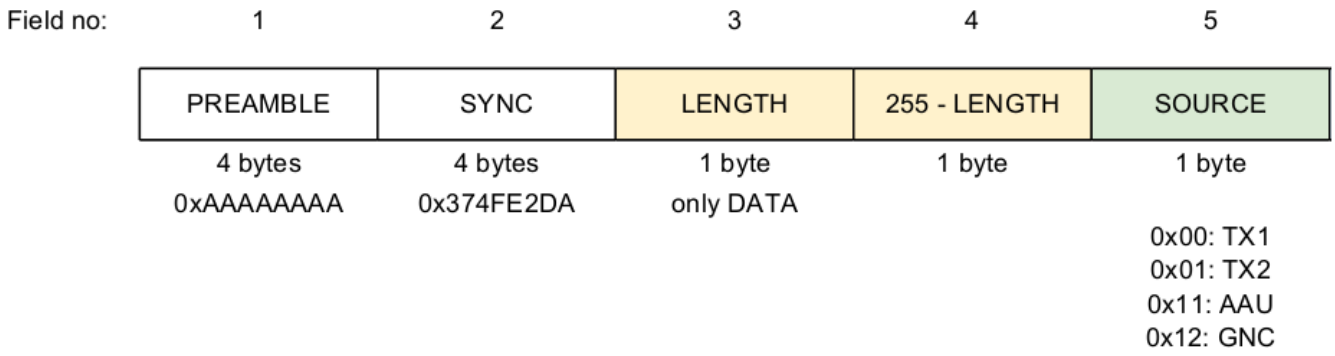

Figure 13: Sapphire telemetry packet header

The packet header contains the following fields:

1. A 4 byte PREAMBLE field consisting of alternating 0 and 1. The preamble is used for training the clock recovery in the receiver.
2. A 4 byte SYNC field containing the pattern 0x374FE2DA. The SYNC field is used by the correlator in the receiver to identify a packet.
3. A 1 byte LENGTH field specifying the number of bytes in the data field.
4. Another 1 byte length field but this time the "inverse" (255-LENGTH).
5. A one byte SOURCE field specifying the origin of the packet. This information is used by the data distribution system in mission control to route the packets to their respective users. Following values are valid:
   - 0x00: TX1
   - 0x01: TX2
   - 0x11: AAU
   - 0x12: GNC

4.2 Data field

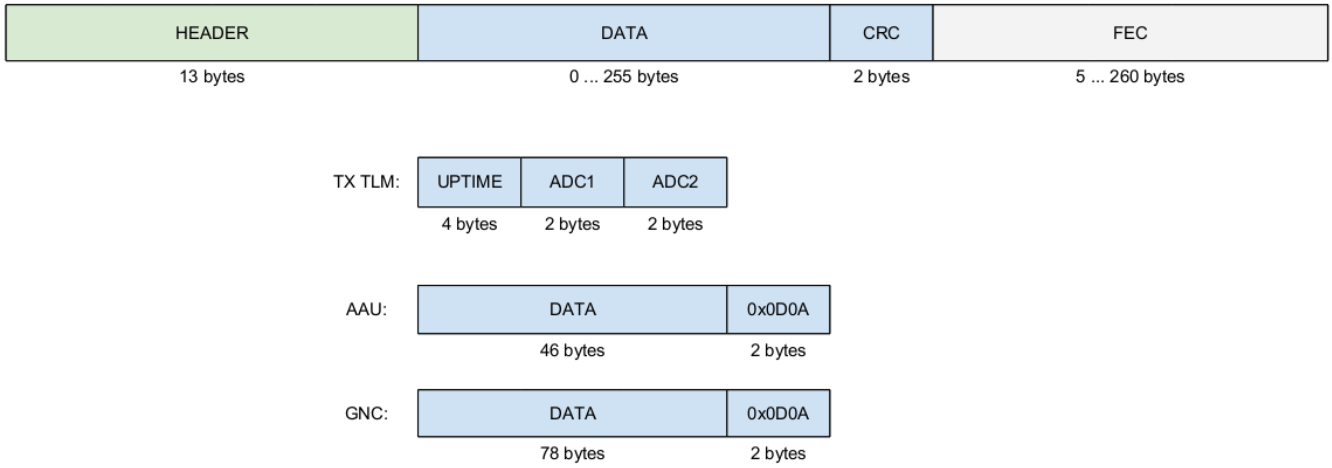

Figure 14: Sapphire telemetry data field

The data field contains the data as received from the payload. There are two types of payload data as indicated on the figure above. They are terminated with carriage return and linefeed characters 0x0D0A indicating packet end. The transmitter interprets this as a "flush the buffer" command even if 0x0D0A occurs inside the data. In practice this may lead to fragmentation of payload packets; however, this is not expected to cause any problems at the receiving end.

A third type of packet contains telemetry from the transmitter itself. These packets are used to fill out empty slots that are not used by the payload. See Section 4.5 for details.

The transmitter packets contain the transmitter uptime in 1/10th of a second and two raw ADC readings corresponding to the supply voltage and the temperature inside the PIC24.

4.3 CRC field

A 16-bit CRC is added to each packet in order to allow verifying correct packet reception. The CRC is generated using the programmable CRC generator in the PIC24 using the polynomial $x^{16}+x^{12}+x^5+1$.

4.4 FEC field

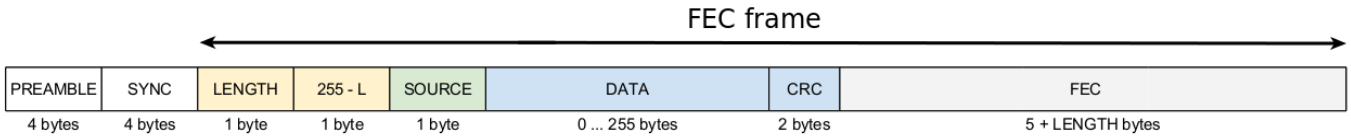

Figure 15: FEC frame

FEC is applied to all fields except the PREAMBLE and SYNC fields in the header. A FEC frame is thus 5 + N bytes long, where N is the length of the data field. Since we are using convolutional code we do not really have separate data and FEC fields as suggested by the figure. Instead, each byte is encoded as soon as it arrives and the coded data is put into the output buffer.

4.5 Downlink budget

In the present configuration we have 250 kbps at our disposal. We apply rate 1/2 FEC to the data, thus we have 125 kbps at our disposal. Table 1 below shows the downlink budget under these circumstances. As can be seen from the table, almost half of the traffic is transmitter telemetry to fill out empty slots, thus there is plenty of room for future increase in data rate.

Table 1: Bitrate budget

Source	**Packet size [bytes/pkt]**	**Packet overhead [bytes/pkt]**	**Packet length [bits/pkt]**	**Packet rate [pkt/sec]**	**Bitrate [bps]**
GNC	80	14	752	102.25	76789.62
AAU	48	14	496	20.00	9920.00
TX	10	14	192	198.00	38016.00
Total					125.83 k

5 Forward error correction

To be written

As writen in the Packet Format section, we use both CRC and rate 1/2 convolutional error correction code. While the convolutional code is used to correct bit errors in the received data, the CRC is used to verify that the received data is correct.

6 Glossary

AAU	Autonomous Abort Unit
ADC	Analog to digital converter
AZ	Azimuth
CRC	Cyclic redundancy check

EL	Elevation
FEC	Forward error correction
FSK	Frequency shift keying
GNC	Guidance, navigation and control
ISM	Unlicensed frequency bands used by industrial, scientific and medical devices
RF	Radio frequency
SDR	Software defined radio
SNN	Signal+noise to noise ratio
SNR	Signal to noise ratio
USRP	Universal Software Radio Peripheral