Fox-1 Satellite Telemetry Part 2: FoxTelem

Chris Thompson • AC2CZ

FoxTelem is the AMSAT ground station software for the Fox-1 series of satellites that supports decoding of the Data Under Voice (DUV) and high speed telemetry. This article will discuss some of the advanced features of FoxTelem and will preview some of the interesting things that will be available as future satellites are launched.

How FoxTelem Works

At its simplest, FoxTelem reads audio from a source, samples that audio into binary digits, then decodes the binary format. You can see this flow in Figure 1. The output is a frame of telemetry data. The frame is then unpacked into telemetry values. We call them "raw" values because they are the original source readings from the instruments on the spacecraft. They are shown when you click Display Raw Values on one of the FoxTelem screens. We apply conversion routines to give us human-readable data when we display them on the screens and graphs.

Let's follow the flow in Figure 1 where FoxTelem is reading audio from the soundcard and decoding 200 bps DUV. Let's also assume the sample rate is 48000 samples per second. One bit will be stored in 48000/200 = 240 samples. FoxTelem then reads 70 bits worth of audio from the soundcard, or 16800 samples.

The audio chunk contains the telemetry and any voice transmissions. FoxTelem runs the 70 bits through a digital filter to remove all of the audio above 200 Hz. The user can configure the filter, but in most cases the standard 200 Hz Raised Cosine Filter with 512 coefficients works well. We have broken the audio into chunks, which upsets the Digital Filter, giving us pops and crackles as the audio cycles through. We use the overlap add method to eliminate this. In summary, we take the excess audio that is created by the digital filter at the end of a chunk and add it to the start of the next chunk.

FoxTelem knows how wide one bit is by counting the samples -- 240 samples for 200 bps when the sample rate is 48000. The samples can be analyzed quickly and tagged as a one or a zero. But that rapidly fails because the first sample is very unlikely to be the start of the first bit. It is probably somewhere in the middle of a bit. Step 1 in decoding the bits, therefore, is recovering the clock.

Clock Recovery

Consider the situation in Figure 2 where we examine the first five bits. Each bit has 200 samples of audio. We find that the middle of the bit has values of something like 0.5, 0.5, 0, 0.5, 1. The clock is clearly misaligned. FoxTelem calculates when each bit starts to transition, on average, and works out how much the audio stream needs to be shifted. It reads some additional bits from the audio channel, effectively pulling the audio stream forward slightly. This will be a fraction of a bit, and we end up with the situation in Figure 3. If the clock stays aligned, then we won't need to adjust the clock again, but that is never the case. The clock on the spacecraft, in the hostile environment of space, and the clock in your computer, safe in your shack, do not stay aligned.

With the clock recovered, FoxTelem samples each bit in the middle of the sample period. At 9600 bps we only have 5 bits for each sample, so we average the middle 3 bits to try to compensate for noise. For 200 bps, we use a more sophisticated algorithm that measures the distance from the last bit. This better compensates for sloping bits that have

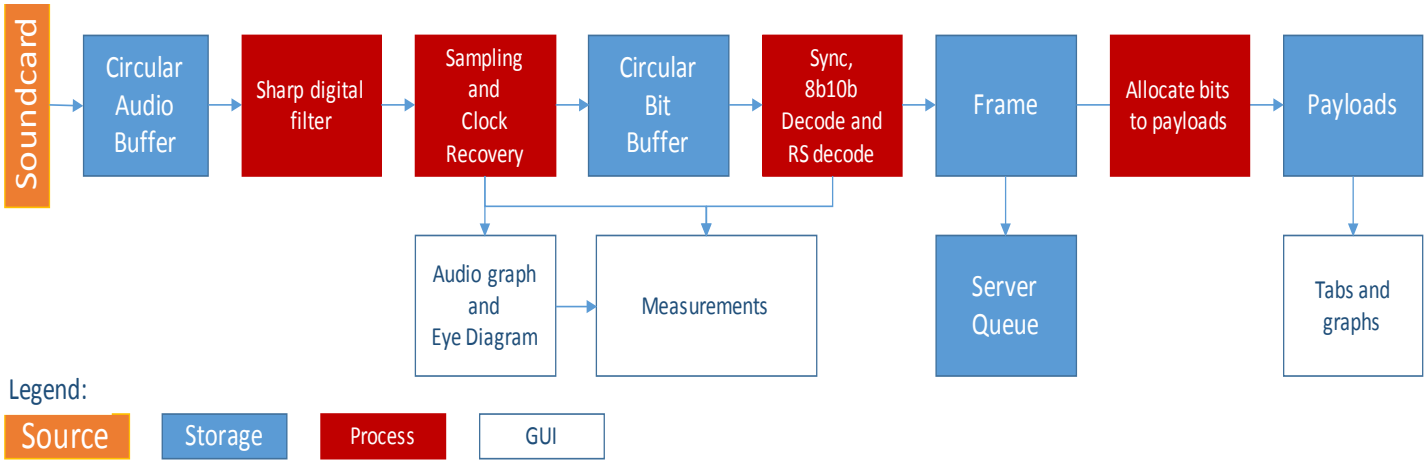

Figure 1 - FoxTelem decoding from a soundcard

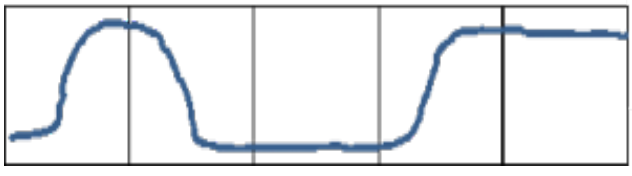

Figure 2 - Mismatched sample periods

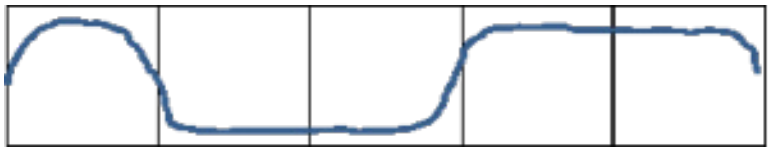

Figure 3 - After we have read an extra third of a bit

been distorted by high pass filtering, such as those shown in Figure 4. You can see that bits start at their full value then steeply slope. If we have several bits in a row representing 1, then the value can drop below the centerline, and we can incorrectly sample the last bit as a zero. The "distance to last bit" algorithm compensates for this and allows radios such as the Yaesu FT-817 to be used as the audio source.

Once the bits are sampled, they are written into a large circular bit buffer, shown in the middle of the flow in Figure 1. After each chunk of audio is sampled, we check to see if the SYNC word has been received, which is 1100000101 or its compliment 0011111010. If we find a SYNC word, then FoxTelem notes its position in the circular bit buffer. A DUV frame has 96 bytes (64 bytes of data and 32 check bytes). Each byte is sent from the spacecraft as a 10-bit word. So, if we have two SYNC words that are 960 bits apart, then we might have a valid frame. We find lots of SYNC words in random noise, so we ignore SYNC words, waiting for a valid frame length before trying to decode further.

With two sync words the right distance apart we sample at the next level of detail. It's like unwrapping a parcel with many layers of paper. Taking 10 bits at a time and decoding them into an 8-bit byte using a lookup table decodes this next layer. The bytes have been 8b10b encoded (see **en.wikipedia.org/wiki/8b/10b_encoding**) to minimize the number of 1's and 0's that we get in a row. This minimizes the slope of the bits and helps with decoding when high pass filtering is present.

Sometimes the 10-bit word is not in the lookup table, so FoxTelem notes the byte as corrupt. The 96 bytes are then passed to the Reed Solomon Decoder (see **www.ka9q.net/code/fec/**), together with the list of corrupted bytes, called "erasures." The RS Decoder uses the 32 check bytes to fix any errors in the received data. If it succeeds, then it reports the number of errors corrected. If it fails, then the frame is corrupt and was either too damaged or was not a real frame in the first place.

FoxTelem displays the number of errors and erasures on the eye diagram (Figure 5). This is the number from the last successfully decoded frame. You can also plot the Errors and Erasures on a graph from the Measurements Tab. This gives you a feel for how corrupt the data stream is.

Unpacking the Data

We now have 64 bytes of valid data -- a DUV frame -- from the spacecraft. The raw data values are packed into the 64 bytes as tightly as possible. FoxTelem uses a set of files in the spacecraft directory to unpack the data. First, we decode the header to work out what type of payload we have inside the frame. The header is laid out as follows:

| 1 0 1 1 0} [0 0 1]<- [FoxID] (3 bits) = 001
| 0 0 0 0 0 0 0 1 <- {Reset count} (16 bits = 000 00000001 10110 = 54)
| 0 1 0 1 1] {0 0 0
| 0 0 0 0 0 0 0 1 <- [Uptime] (25 bits = 0000 00000000 00000001 01011 = 43
| 0 0 0 0 0 0 0 0
|{0 0 0 1}[0 0 0 0 <- {Frame type} (4 bits = 0001)

We might have hoped for a simpler layout, but it turns out that this *is* rocket science, and apparently, even when you try to make it simple, somehow it becomes

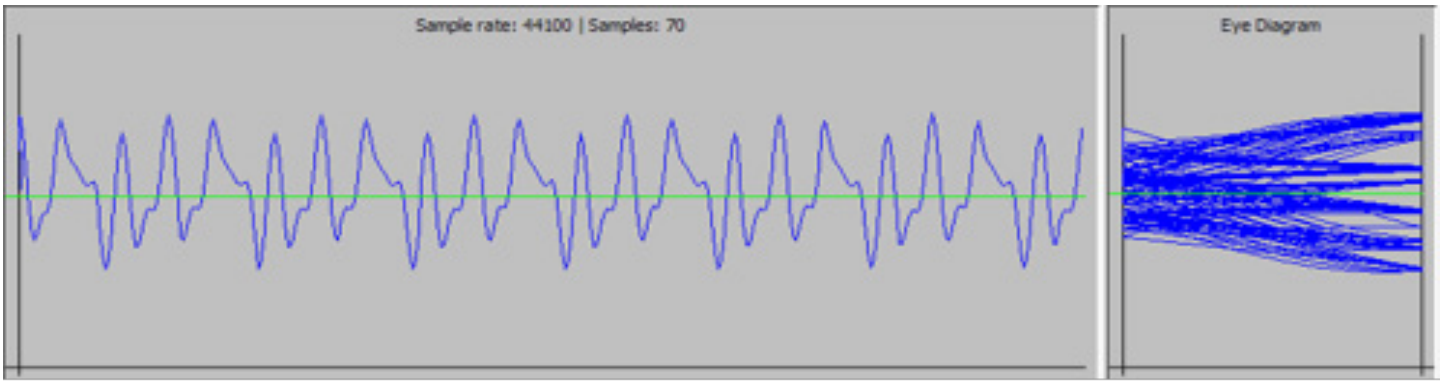

Figure 4 - Bits distorted by high pass filtering

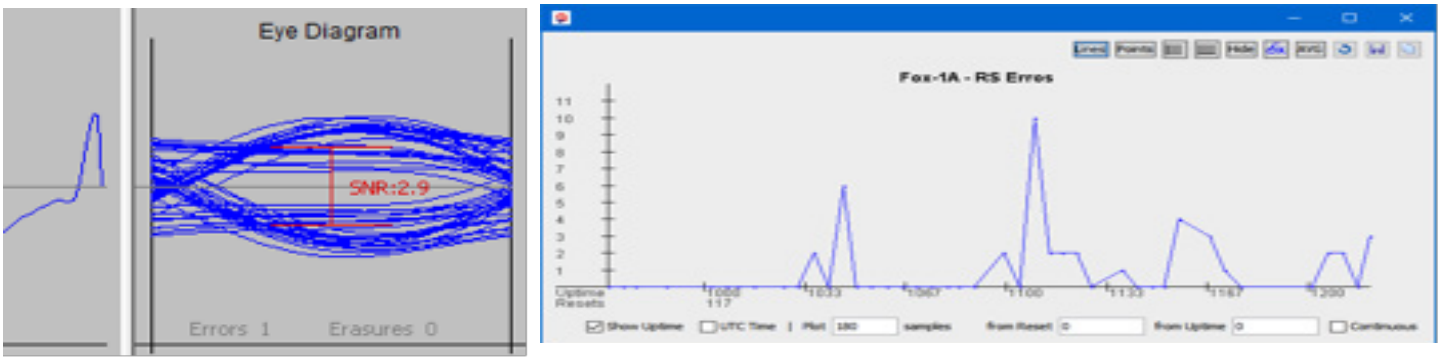

Figure 5 - Errors and erasures

complex. The complexity results from the spacecraft sending the most significant bit first (as we can see with the FoxID), but Fox's computer is a little-endian system; therefore, we get the least significant byte first. Decoding the data is full of fun little challenges like this.

Having sorted out the bit and byte order, we next decode the header and recover the Fox ID, Reset Count, Uptime and Payload Type. The FoxID and Payload Type tell us which layout we are going to decode. Types 1, 2 and 3 are telemetry for the spacecraft. Type 4 is experiment data.

We use the same bit and byte order logic as the layouts in the spacecraft directory to unpack the remaining 60 bytes into raw values. The raw values are then written to disk. If the user has selected Display Raw Values, then these are shown on the screen. Otherwise, each value is converted to a human-readable form using a conversion routine specified in the layout file. The FoxTelem manual specifies these conversions.

High Speed

High speed data is decoded in the same way as DUV data. Each Frame is 52720 bits long. We search for SYNC words in exactly the same way and attempt to decode a frame when we find SYNC words are the right distance apart.

Instead of running the Reed Solomon Decoder for the whole frame, we break the frame into 21 Reed Solomon words and run the RS Decoder for each one. The data is sent from the spacecraft with the bytes allocated round robin to the RS words. The check bytes are all sent at the end of the frame in a block. This pattern allows us to better cope with fading and interference.

The high speed frame also contains a header, so FoxTelem first decodes it and identifies the spacecraft. Each high speed frame then contains real time, max and min telemetry payloads. The layout of the rest of the high speed frame is then determined by the experiments onboard the spacecraft: Fox-1A contains Type 4 experiment payloads; Fox-1Cliff will contain Type 5 camera lines; and Fox-1D will contain either University of Iowa HERCI data or Type 5 camera payloads. We will talk more about these payloads in a future article.

UPDATE: A Full-Duplex VHF-UHF Satellite System Using Flex SDR

Ronald G. Parsons • W5RKN
w5rkn@w5rkn.com

The article, "A Full-Duplex VHF-UHF Satellite System Using SDR," that appeared in the July/August 2013 issue of *The AMSAT Journal* – **www.flexradio.com/downloads/w5rkn_julyaugust2013withcover-pdf** -- described a fixed ground station for working satellites using two FlexRadio Systems® radios. This update describes a number of modifications and improvements to the original station configuration. The two transmit and receive radios were replaced with a single Flex-6500, a tuning knob was added, and a software program, FlexSATPC, written by David Beumer, W0DHB, was incorporated to tie all the functions together. [See "Using SatPC32 with FlexRadio Systems Radios," p. 13.]

Implementation

Three major changes were made to the system. The first was the addition of a software program, FlexSATPC, which acts as the system central control point. FlexSATPC provides an interface between SatPC32, the tracking and Doppler control software, and SmartSDR, the FlexRadio control and display software because SatPC32 does not support SmartSDR. FlexSATPC also includes a logging interface that enables all required logging by just entering the callsign. All other fields required for logging are automatically entered. FlexSATPC also facilitates uplink frequency calibration to match the satellite's transponder.

The second change was to improve the full-duplex operation, made possible with version 1.5.0 of FlexRadio's SmartSDR control program. The third was adding a FlexControl tuning and control knob. The knob controls the frequency of the active slice and other programmable functions. If the operator is using the FlexControl to tune the radio and a separate logging operator is entering data in other programs, the FlexControl only controls SmartSDR, leaving other programs under mouse control. I made good use of this feature when operating as W1AW/5 and W7O.

The original design used a FLEX-1500 for transmit and a FLEX-3000 for receive, but the required switching between various bands and antennas was very complex. The single antenna port on the FLEX-3000 made it difficult to operate HF or satellites which use HF such as AO-7 Mode A.

When FlexRadio introduced its newest Signature Series model in May 2014, the FLEX-6300, the SmartSDR software that controls the new radio did not support use of transverters. In August 2014, FlexRadio added transverter support, so I replaced the two radios in the original system with one FLEX-6300. Effecting the change was simple, requiring adjusting the CAT COM port and moving the coax from the receive transverter coax switch for the FLEX-3000 to the FLEX-6300 ANT2 port and moving the coax from the transmit transverter coax switch on the FLEX-1500 to the FLEX-6300 XVTR port. That left the FLEX-6300 ANT1 output free for 6-meter use.

FLEX-6300 users must be very careful with implementing and operating a system that allows the physical hardware to transmit into the RX IF port of the transverters. I tried to minimize this risk by automating the signal routing and other software settings. FlexSATPC also contains checks to help avoid this potential problem. Still, the operator must be aware of this risk and operate accordingly.

To completely eliminate this risk, I traded in my FLEX-6300 for a FLEX-6500, which has a separate receive-only port RX A that is connected to the RX IF ports of the transverters. The XVTR port is connected to the TX IF ports of the transverters for transmitting.

Major Features

Full Duplex

The addition of full-duplex capability in FlexRadio's Signature Series radios – FLEX-6300, 6500, and 6700 – enables the audio from the radio's receive slice to pass through to the speakers/headphones even when another slice is transmitting.

"Slice" is the term used by FlexRadio for a segment of the entire spectrum sampled by