KickSat: Bringing Space to the Masses

Zac Manchester, KD2BHC

Who hasn't dreamed of launching their own satellite? The opportunities afforded to scientists, hobbyists, and students by cheap and regular access to space could open up new areas of scientific research and enhance participation in science, technology, engineering, and math (STEM) education. The KickSat project, begun at Cornell in 2011, is trying to put space within reach of everyone by dramatically lowering the cost and technical expertise required to build and fly a satellite. This article will provide an overview of KickSat and its communication system, including information for setting up an amateur ground station.

The Sprite Spacecraft:

CubeSats have received a lot of well-deserved attention in the last few years. They've helped greatly expand the opportunities for students and HAMs to participate in space. Unfortunately, however, the barriers to entry remain high. The cost of building and launching a CubeSat is typically measured in hundreds of thousands of dollars and its development, integration, and testing usually requires a team with broad engineering expertise.

Thanks to rapid advances made in the semiconductor industry, it is now possible to integrate most of the features of a traditional spacecraft into a chip-scale device. At Cornell, we've leveraged the sort of tiny, low-cost, low-power integrated circuits used in modern consumer electronics to build the Sprite, an example of a new category of spacecraft known as a "ChipSat" or "femtosatellite." The Sprite includes solar cells, a Texas Instruments MSP430 microcontroller, a 70-centimeter band transceiver, and several sensors on a printed circuit board measuring 3.5 by 3.5 centimeters and is, to the author's best knowledge, the world's smallest purpose-built spacecraft.

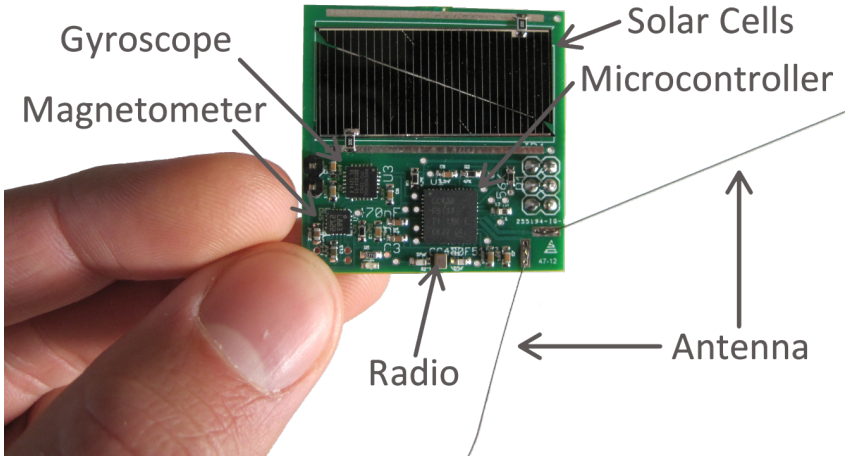

Figure 1. Sprite ChipSat

The Sprite is intended as a general-purpose platform for small experiments, serving as host to any of the numerous chip-scale sensors now commercially available. In the near future, it will be possible for a student or hobbyist with basic electronics skills to choose a sensor or two, write some microcontroller code, and put together a working satellite with a few hours' work. By shrinking the spacecraft and launching many together, we can realistically achieve per-Sprite launch costs of $1,000 or less at current prices.

KickSat:

The KickSat mission is a complete end-to-end demonstration of the Sprite, from launch and deployment to communication with ground stations and tracking. It has been made possible through the generous support of over 300 individual backers on the crowd-funding website Kickstarter. Over $74,000 was contributed in exchange for rewards such as having a name printed on a solar panel, receiving a souvenir Sprite, or getting to program the flight code on a Sprite.

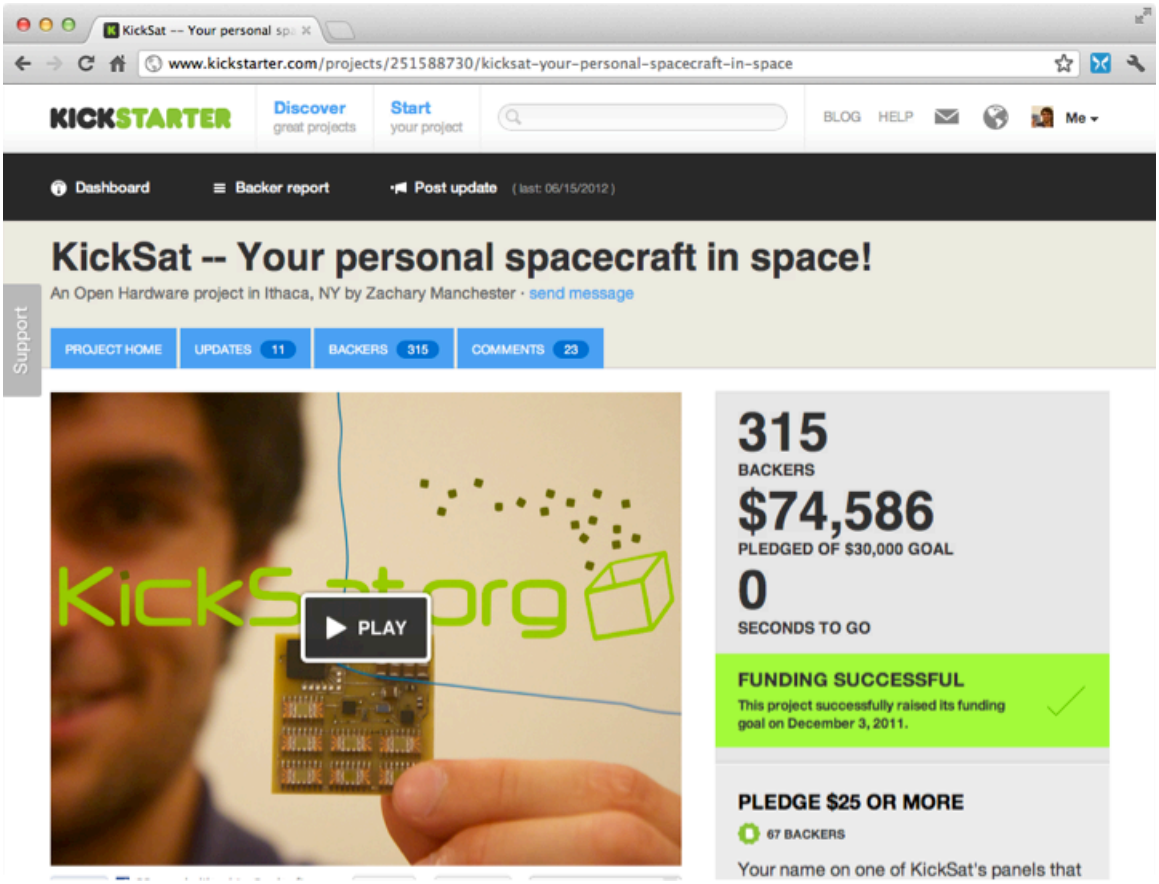

Figure 2: KickSat on Kickstarter

Kicksat's launch has been awarded through NASA's Educational Launch of Nanosatellites (ELaNa) program, which places university-built CubeSats as

secondary payloads on NASA missions. KickSat is currently manifested on CRS-3, a Space-X Falcon 9 set to launch in late 2013. CRS-3's primary mission is to bring supplies to the International Space Station, so KickSat will be placed in roughly the same orbit as the ISS – a 325 km altitude circular orbit with an inclination of 51.5°.

KickSat itself is a 3U CubeSat consisting of a 1U bus and 2U Sprite deployer. The bus is being built using a combination of commercial-off-the-shelf (COTS) CubeSat hardware and Cornell-built hardware to provide standard power, communication, and command and data handling functions. The deployer will house approximately 150 Sprites inside a spring-loaded mechanism actuated by a nichrome burn wire. A command from Cornell's ground station will trigger the burn wire, releasing the Sprites as free-flying spacecraft.

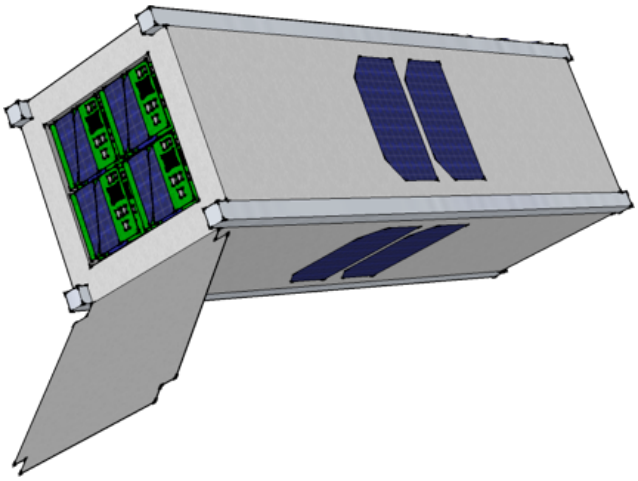

Figure 3: KickSat 3U CubeSat

After deployment, the Sprites will remain in orbit for a few weeks before reentering and burning up in the Earth's atmosphere. During that time, they will collect sensor measurements, perform calculations and, most importantly, communicate with amateur ground stations worldwide. The following sections will provide some technical background on the Sprite's communication system and the hardware and software required to set up a receiver. More information on Kicksat and updates on the project's status are available online at kicksat.net.

Sprite Communication Background:

One of the most difficult engineering challenges associated with the KickSat project is closing the communication link from orbiting Sprites to Earth stations. The Sprite's transmitter is limited to about 10 milliwatts of power. Additionally, a lack of attitude control (the ability to point or reorient the spacecraft) means a low-gain antenna with an omnidirectional gain pattern is required. Lastly, we need an efficient way for all of the Sprites on a given mission to share limited bandwidth. Closing link over several hundred kilometers with all of these constraints may, at first glance, seem impossible, but with some signal processing, it turns out to be

quite doable with relatively cheap hardware. This section will provide a conceptual overview of the techniques used in the Sprite receiver for the non-expert.

Let's start with some basic link budget calculations. We'll follow the link from end to end, working in decibels, to estimate the signal to noise ratio (SNR) at the receiver. A Sprite transmits with a power of 10 mW or 10 dBm. The Sprite's V-dipole antenna is approximately isotropic with a gain of about 0 dB. To account for downrange distance and allow for some margin on top of the Sprite's 325 km orbital altitude, we'll baseline a distance of 500 kilometers. The Friis equation gives us a free space path loss of $20\log\left(\frac{\lambda}{4\pi r}\right) = 20\log\left(\frac{.7}{4\pi\cdot 5\cdot 10^5}\right) \approx -139$ dB. We'll assume our receiver antenna has about 7 dB of gain, consistent with a small handheld Yagi. Adding these values up, we find that our received power should be in the neighborhood of -122 dBm.

The next thing we need to calculate is the noise power in the receiver. There are two main components to worry about – naturally present thermal noise and the noise introduced by the receiver components themselves. Thermal noise is given by $10\log(K_B T B)$, where K_B is Boltzmann's constant, T is temperature in Kelvin (assumed to be about 150 K for space-viewing applications), and B is bandwidth in Hertz. Plugging in the values for our particular case, we get $-177 + 10\log(64 \cdot 10^3) \approx -129$ dBm. The receiver's self-induced noise, expressed in dB as noise figure, has to be measured. For our purposes, we'll assume a noise figure of 9 dB, which is representative of what can be achieved with low-cost hardware. Adding these together, we get a noise power of about -120 dBm.

If we subtract the results of our two previous calculations, we find that our SNR is around -2 dB, the negative sign indicating that we have more noise than signal in the receiver. While this situation might more typically be resolved by increasing transmitter power or using higher-gain antennas, neither of those are viable options in our case. Instead, we'll make use of a trick used for decades by radar systems and the Global Positioning System (GPS) known as matched filtering.

For those who haven't studied signal processing, the basic idea behind matched filtering is to substitute each data bit with a long, specially chosen string of bits known as a pseudo-random number (PRN) code that is agreed upon by the transmitter and receiver beforehand. Rather than trying to lock onto the carrier or look for individual bits, the receiver looks for the PRN code by calculating a statistical correlation against the incoming signal. If the code is present, the correlation will be high, even in the presence of substantial noise, while if no code is present, the correlation will be low. The technique essentially allows the energy in the entire PRN code to be integrated up and treated as a single bit, providing a gain equal to the PRN length.

For the KickSat mission, we're using PRN codes that are 640 bits long, providing a "code gain" of about 28 dB. Adding this to our SNR puts us at a very respectable +26

dB. Keep in mind, however, that our data rate is now also lower by a factor of 640, so there's no free lunch. We've simply managed to trade data rate for gain.

Matched filtering helps us close our link, but it also helps us in another way. By assigning different PRN codes to each Sprite, we can implement code-division multiple access (CDMA), which you may be familiar with from the cellular telephone standard. Rather than assigning each Sprite its own frequency, they can all share one frequency and the receiver can "tune" to a particular Sprite by looking for its unique PRN code. This has several advantages, including reduced use of spectrum, simplified licensing, and the ability to record the signals from all the Sprites in a pass with one receiver.

The last piece of theoretical background we need is forward error correction (FEC). FEC is widely used in modern digital communication because it allows a receiver to correct noise-induced errors in a message without having to ask for a re-transmission. The idea is to pad the message with extra bits, known as parity check bits, based on some mathematical rule. In our case, a linear block code is used where a block of data is treated as a binary vector and encoded by simple matrix multiplication.

To encode a byte, the Sprite multiplies the corresponding 8-bit vector by an 8-by-16-bit matrix, known as the generator matrix of our code, to produce a 16-bit code word. The receiver can take advantage of the redundant bits in the code word and their mathematical relationship to the message bits to reproduce the original message byte, even if errors have been introduced in transmission. For those familiar with coding theory, our code is a (16,8,5) block code, and therefore can detect and correct up to 2 bit flips or 5 bit erasures.

Sprite Receiver:

With the goal of allowing as many people as possible to participate in the KickSat project, we've assembled and tested a reference design for a low-cost and portable KickSat receiving station. The hardware consists of a hand-held yagi antenna, an LNA, a DVB-T USB dongle, and a PC running the GNURadio software. The total hardware cost, not including the PC, is around $200. Full instructions for assembling a ground station will be available on the KickSat project wiki, accessible at www.kicksat.net.

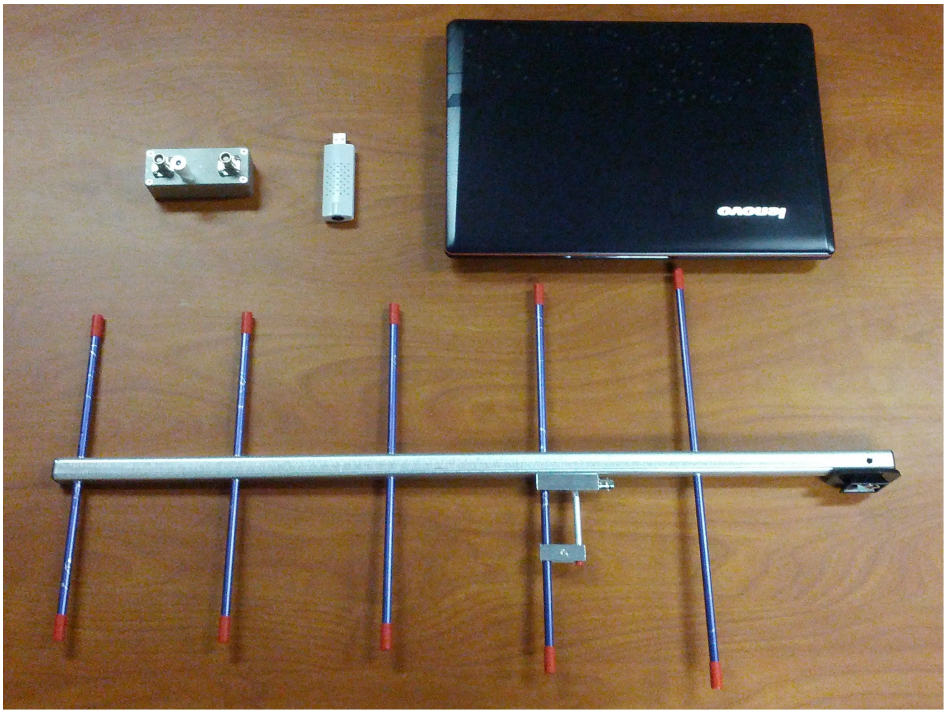

Figure 4: Ground Station Hardware

Because of the signal processing requirements inherent in our receiver design, a software defined radio (SDR) receiver is being used. The DVB-T dongle functions as a low-cost front end and analog to digital converter, bringing the raw baseband signal into the PC. From there, our receiver is written in C++ as a set of blocks for the GNURadio software framework. The block diagram in figure 5 shows the signal flow in the receiver.

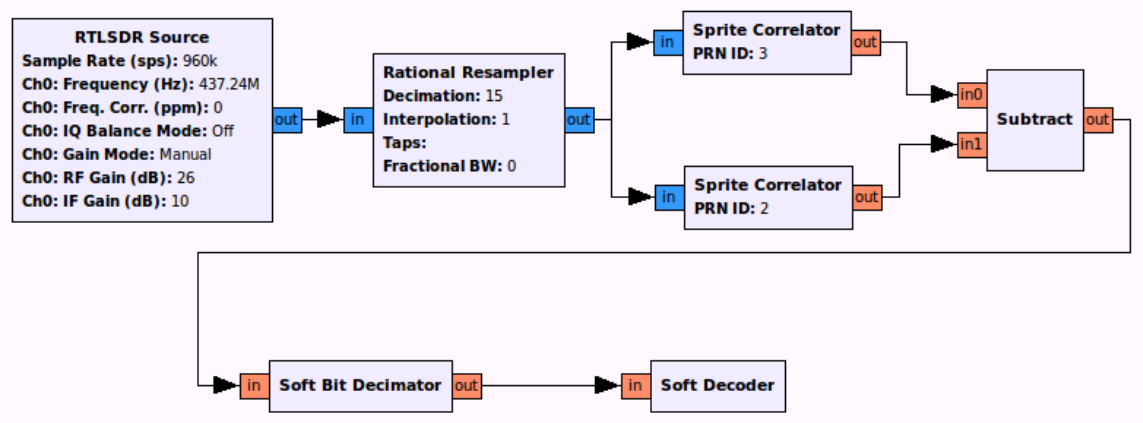

Figure 5: Receiver Block Diagram

Starting from the DVB-T dongle input on the left, the signal is decimated (low-pass filtered and down-sampled) to one sample per PRN chip, which is 64 kHz in our current implementation. From there, it passes through two PRN correlators, each of which performs matched filtering against a different PRN code. Figure 6 shows the

output of a correlator in which the spike corresponding to a PRN code is clearly visible.

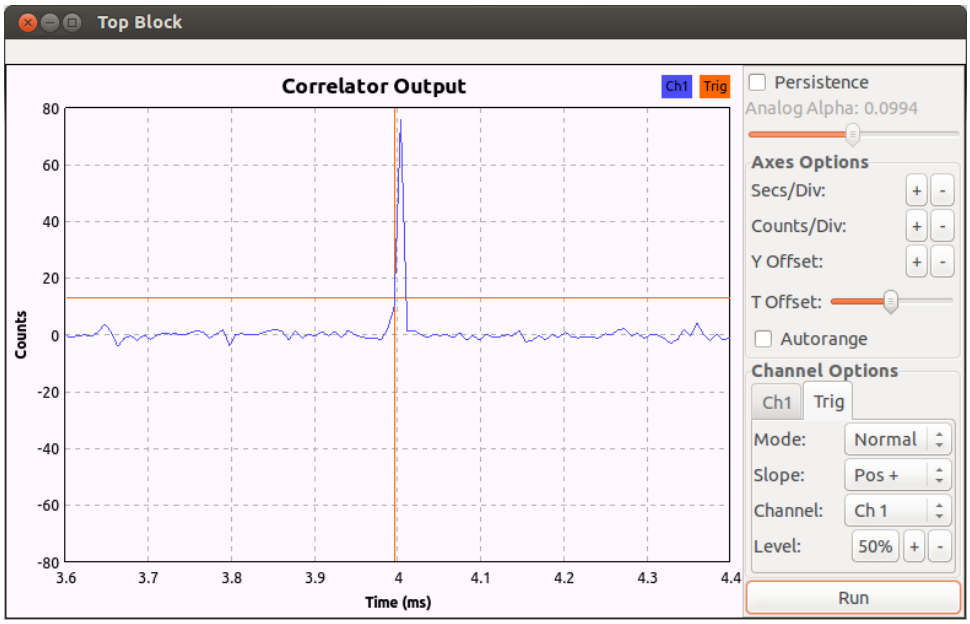

Figure 6: Correlator Output

Each Sprite is assigned two PRNs – one corresponding to a zero bit and one corresponding to a one bit. The correlator outputs are subtracted, giving a zero-mean signal where a one corresponds to a positive spike and a zero corresponds to a negative spike. The signal is then down-sampled again, this time to 200 Hz, by the Soft Bit Decimator block, before passing into the decoder.

The Sprite decoder is a maximum-likelihood soft decoder. It takes a group of spikes from the correlators and determines the byte that they most likely correspond to, taking into account the parity bits. The best match is then printed to the console. Figure 7 shows a screenshot of a running receiver with decoder output at the bottom of the window.

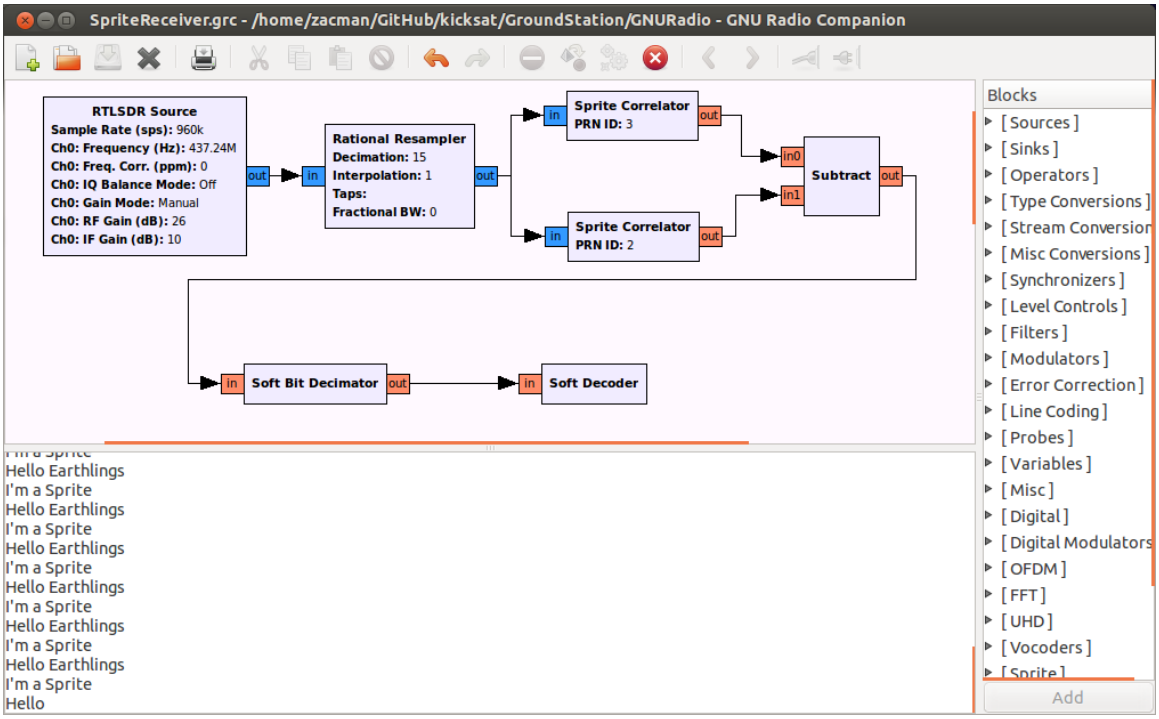

Figure 7: Receiver Output

The Sprite software receiver can run in real time on relatively recent PC hardware. It can also be used in a batch mode where data is recorded during a pass and fed through the receiver later. Both scenarios have been tested outdoors with Sprites and receiver separated by 25 miles and an additional 23 dB of attenuation inserted after the receiver antenna, roughly corresponding to the link conditions between LEO and Earth stations anticipated for the KickSat mission.

Conclusion:

KickSat represents a new way for people across the world to participate directly in spaceflight. With very modest hardware, amateurs can receive signals from Sprites in LEO during the KickSat mission. All of the design files and code for the Sprite and its software receiver are open source and available online for anyone to build their own or use as a starting point for new designs.

Acknowledgements:

The author would like to thank the many people who have supported KickSat financially through Kickstarter, as well as Andy Filo for many technical contributions.