SOFTWARE DEFINED RADIO FOR CARRIER AND SYMBOL TIMING EXTRACTION

Sergio Lara, Dr. Charles Creusere
Klipsch School of Electrical Engineering
New Mexico State University
Las Cruces, NM, 88011
laraser@nmsu.edu, ccreuser@nmsu.edu

ABSTRACT

Using LabVIEW, a software-defined radio receiver was developed to extract the phase of the carrier and the symbol timing interval for possible satellite diagnostics. The receiver consists of two main parts: a phase-locked loop and a symbol timing algorithm. Using a Weaver demodulator and a complex-to-complex mixer, the phase of the carrier is extracted from the input signal. A windows-based timing algorithm, compatible with non-return-to-zero and biphase coding, extracts the symbol arrival time interval. Lastly, a two-stage logging algorithm temporarily stores output data in a DRAM buffer before streaming the data to the hard drive for final storage.

INTRODUCTION

Communication systems depend on crystal oscillators to generate the carrier sinusoid and the symbol timing intervals. Over time, oscillator crystals wear out, causing the frequency of the sinusoids to deviate from the desired value. When the deviations are small, the change in frequency can be ignored or fixed through a software update. On the other hand, if the frequency deviations become large enough or the frequency deviations change sporadically, the communications system becomes unusable. Therefore, it is imperative to know the health of the crystal oscillator. Directly determining the health of the crystal oscillator of a satellite in orbit presents its challenges, however, so a system capable of tracking and logging the satellite's carrier frequency and symbol timing deviations from the earth's surface is desired. Through post-processing analysis, it may be possible to estimate the health of the crystal oscillator.

SOFTWARE-DEFINED RADIO

Software-defined radios are communications systems implemented in software instead of hardware. In previous times, digital computers and embedded systems, unlike hardware, were not fast enough to process digitized communication signals, so only a small number of processes were implemented in software. Due to advances in digital signal processing and embedded systems, the number of components implemented digitally has increased and will continue to increases over the

years. Using National Instruments' Vector Signal Transceiver (VST) and FlexRIO FPGA module, a software-defined radio receiver was developed within the LabVIEW framework.

PHASE-LOCKED LOOP (PLL)

A phase-locked loop is a control system that outputs a sinusoidal waveform whose phase matches the input signal's phase. In our system, the phase error between the incoming signal and the oscillator is calculated and then passed through an FIR filter. An IIR filter is then applied to smooth the phase error before using the output to adjust the oscillator, minimizing the phase error between the incoming signal and the oscillator. Both the IIR filter output and the oscillator outputs are logged for analysis.

A. WEAVER DEMODULATOR

The Weaver demodulator is a single sideband demodulator that works by first demodulating the RF signal to an intermediate frequency (IF) and then demodulating the intermediate signal to baseband. One of the advantages of a Weaver demodulator is that it does not require sharp cutoff filters or wideband 90° phase-difference networks [1]. Additionally, the weaver demodulator has the advantage of rejecting the undesired images in the transmitted signal. In our system, the VST demodulates the signals to an intermediate frequency and transfer the IF signal to the FPGA card, where the complex to complex mixer resides.

The process begins with the Vector Signal Transceiver (VST) receiving a signal from the antenna or external signal generator and demodulating the signal to an intermediate frequency before transferring the output I and Q channels to the VST's internal FPGA card. Matching the center frequency of the VST to the center frequency of the incoming signal demodulates to the intermediate frequency given by Equation (1), where f_c is the center frequency of the VST. However, exactly matching center frequencies limits the PLL's ability to extract the desired parameters. In cases where the symbol rate is higher than the intermediate center frequency, the PLL struggles to lock on to the carrier center frequency and instead attempts to lock onto the envelope created by the symbol modulation. Offsetting the center frequency of the VST by 1MHz corrects the issue by ensuring that the symbol rate is lower than the carrier frequency. The new DDS frequency is then calculated using sEquation (2), where f_c is the center frequency of the VST.

$$f = \frac{f_c}{200000} \tag{1}$$

$$f = \frac{f_c}{200000} + 6249800 \tag{2}$$

From the VST's internal FPGA card, the I and Q channels are transferred to the FlexRIO FPGA card through a P2P transfer protocol and passed through a complex-to-complex mixer. The mixer outputs two signals: a real channel and an imaginary channel. In the mixer, the signal's I channel and the oscillator's Q channel are multiplied, and the signal's Q channel and the oscillator's I channel are multiplied. The outputs of the multipliers are subtracted, resulting in the real channel. For the imaginary channel, the two I channels and the two Q channels are multiplied, and the

outputs are added, resulting in the imaginary channel. If small-angle approximations are assumed, and the frequency of the signal and the oscillator are close, then the resulting output signals have the forms shown in Equations (3) and (4) [2]. From the two outputs, the real channel was chosen due to the linearity of the phase error. Otherwise, an additional step would have been required to eliminate the quadratic term.

$$y\_Real \approx -(\psi - \theta) \tag{3}$$

$$y\_Im \approx 1 - \frac{(\psi - \theta)^2}{2} \tag{4}$$

In the complex-to-complex mixer, the real channel is calculated and passed to the FIR filter to remove symbol information while retaining the phase error. In parallel to the complex mixer, a basic demodulator demodulate the signal to baseband. The two I channels are multiplied, and the output is passed through a different FIR filter.

B. FINITE IMPULSE RESPONSE (FIR) FILTERS

The following MATLAB script was used to generate a 200th order Kaiser-window FIR filter:

```
n=200;
w0=.001;
beta=10;
b_fir=fir1(n,w0,kaiser(n+1,beta));
freqz(b_fir)
```

Three parameters control the frequency response of the filter: the order, cutoff frequency, and beta. The cutoff frequency is the point at which frequencies go from being passed to being attenuated. This value is application-dependent. Parameter β determines the trade-off between the main lobe and the side lobes of the filter. Lastly, the order of the filter determines the feasibility of its real-time implementation. The higher the order, the steeper the curve of the cutoff frequency and the closer the cutoff frequency of the filter will be to the specified cutoff frequency. Higher-order filters are more complex and offer better performance but required more FPGA resources.

For the SDR, the PLL must track frequencies $\pm 40kHz$ from the center frequency due to the orbital Doppler shift of the satellite. Frequencies higher than $40kHz$ contain information not needed for tracking the carrier frequency. The order of the filter was determined through experimentation. A filter order of 200 provides a good tradeoff between hardware resource consumption and performance. Setting parameter $\beta = 10$ creates a filter with good attenuation outside of the main lobe, though the width of the main lobe increases slightly. Figure 1 shows the frequency response of the filter used in the PLL. Studying the plot, we see that the FIR filter has a narrow passband range, which eliminates high-frequency components such as symbol modulation. The output of the low-pass filter is the phase error which is used to control the frequency of the oscillator, as described in Section D.

A second FIR filter was generated for the output of the basic demodulator. Using the output

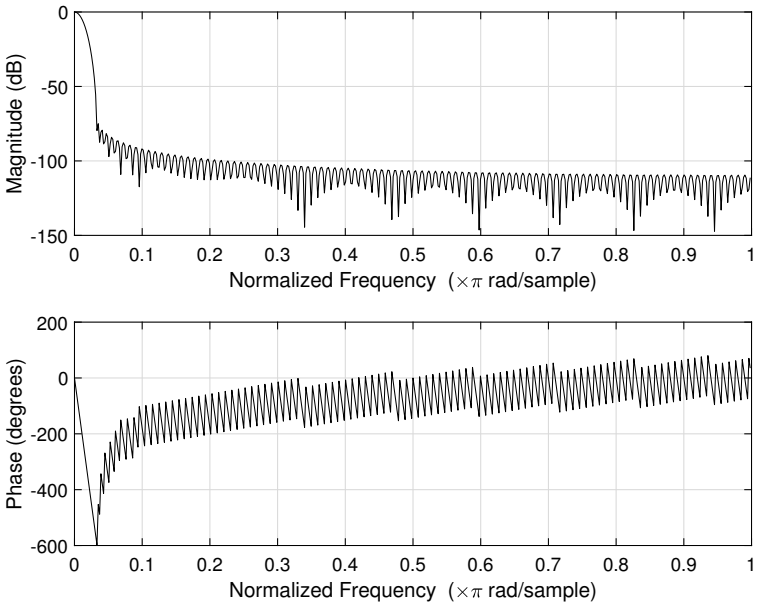

Figure 1: FIR Frequency Response

of the complex-to-complex mixer or the first FIR filter is not possible since that filter removes relevant information needed to detect symbols. Passing the output through the FIR filter removes the frequency images introduced by the multiplier. The resulting signal is a waveform with positive values representing one symbol and negative values representing a different symbol. The output of this filter is then passed to the symbol timing code, described in the Symbol Timing section, to calculate the timing deviations of the symbols.

C. INFINITE IMPULSE RESPONSE (IIR) FILTER

Before passing the phase error through the IIR filter, the signal is inverted by multiplying the data by -1, thus removing the negative term in Equation (3). Inverting the signal only affects the operation used in Section D.. Once the signal is inverted, it is passed through an IIR filter to smooth the phase error. A first-order IIR filter with a passband narrower than the FIR filter smooths the phase error, stabilizing the PLL by preventing the oscillator from rapidly changing frequencies. The parameters of the IIR filter are generated using Equations (6)-(7). Parameter α controls the cutoff frequency of the filter, while the K parameter ensures a 0dB gain at DC frequency [3]. The transfer function of the lowpass IIR filter is given by Equation (5). Input ω_c is the desired cutoff frequency of the filter. Note that α needs to be less than 1 for stability.

$$H_{LP}(z) = \frac{K(1+z^{-1})}{1-\alpha z^{-1}} \tag{5}$$

$$\alpha = \frac{1-\sin(\omega_c)}{\cos(\omega_c)} \tag{6}$$

$$K = \frac{1-\alpha}{2} \tag{7}$$

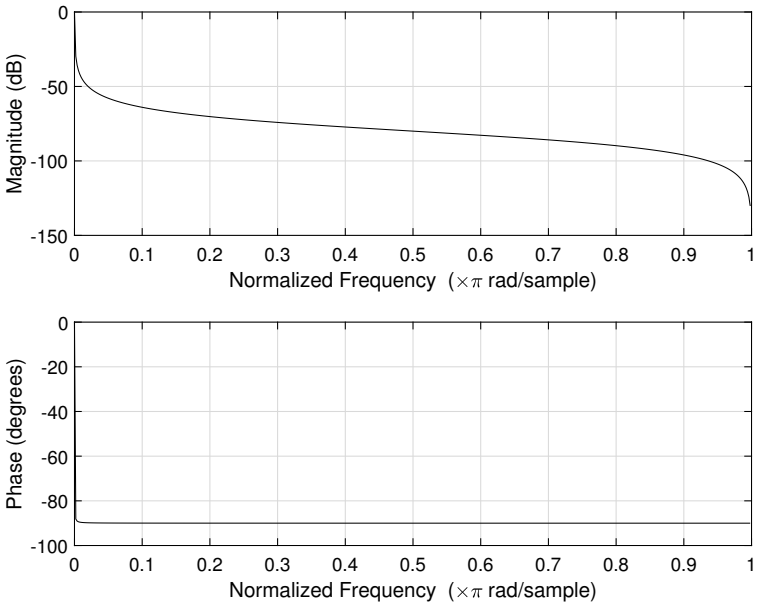

Figure 2: IIR Frequency Response

Analyzing the transfer function, we can calculate the amount of time it takes for the IIR filter to stabilize, known as the time constant. The time constant is dictated by the slowest converging pole of the transfer function, which is the largest pole of the filter [4]. Looking at Equation (8), as the time constant, η_{eff}, increases, $\rho^{\eta_{eff}}$ decreases to 0. We can then look at the number of samples needed for $\rho^{\eta_{eff}}$ to drop below a certain threshold ϵ. A common ϵ used is .01. Manipulating Equation (8), we calculate the time constant using Equation (9), where ϵ is the threshold and ρ is the filter pole with the largest magnitude. For the SDR, a cutoff frequency of $.001rads$ and a time constant of 4605 samples was used to design the IIR filter. Figure 2 shows the frequency response of the IIR filter.

$$\rho^{\eta_{eff}} = \epsilon \tag{8}$$

$$\eta_{eff} = \frac{\ln(\epsilon)}{\ln(\rho)} \tag{9}$$

The output of the IIR filter is passed through a conversion algorithm before going to the input of the oscillator. Furthermore, the output of the IIR filter is transferred to the host for visualization, and stored temporarily in a buffer for post-collection archiving.

D. FEEDBACK AND DDS CONVERSION CODE

The final step in the PLL is to control the frequency of the DDS based on the phase error output of the IIR filter. First, the output of the IIR filter is multiplied by a scaling factor specified by the user in the GUI. Selecting a large scaling factor increases the tracking range of the PLL; however, the PLL becomes unstable when the phase error is small. Choosing a small scaling factor improves the performance when the phase error is small; however, the tracking range of the PLL is reduced. Second, the carrier frequency of the signal coming from the VST, calculated as explained in Section

A., is added to the output. The resulting value is the frequency that the DDS must output to minimize the phase error. By minimizing the phase error, the sinusoids created by the oscillator are locked to the carrier of the incoming signal.

$$f_{out} = \frac{f_{clk}\Delta\theta}{2^{B_{\theta(n)}}} \tag{10}$$

$$\Delta\theta = \frac{f_{out}2^{48}}{250MHz} \tag{11}$$

$$\Delta\theta = f_{out} * 1125899 \tag{12}$$

Lastly, the frequency output is converted into the phase increment input required by the DDS. Using Equation (10), the frequency output of the DDS can be calculated, where f_{clk} is the clock of the DDS, $B_{\theta(n)}$ is phase width, and $\Delta\theta$ is the phase increment. During the configuration of the DDS, the clock and the phase width are specified and remain unchanged until the user manually alters the values. For our SDR, the clock was set to 250MHz, and the phase width was set to 48 bits. Manipulating Equation (10) and inserting the clock and the phase width yields Equation (11). Simplifying the equation results in Equation (12), where f_{out} is the desired frequency output of the DDS, and $\Delta\theta$ is the input required by the DDS to control its frequency. The outputs of the oscillator are transferred to the computer for visualization and stored in a buffer for archiving.

SYMBOL TIMING

The purpose of the Symbol Timing code is to measure the time delay between received symbols in the communication data stream. In theory, the symbol rates of the stream are specified and do not change. However, during actual transmission, jitter can be introduced for various reasons, such as hardware degradation. By measuring the delay between the received symbols, the jitter is calculated and potentially used for diagnostic purposes.

At the heart of the algorithm is a counter running at $40Mhz$. The counter increments every clock tick, only stopping when its input is invalid. When a symbol is detected, the time between the current symbol and the previous symbol is calculated. The unit of the reported time is in samples and is converted to seconds by dividing the samples by the known sampling rate of $40MHz$. For example, a symbol rate of $32ksps$ results in a time differential of $1250samples$ between symbols. Converting the samples to seconds yields $31.25\mu s$, which is the expected symbol period for this symbol rate.

E. SYMBOL TRANSITION EXTRACTION

The output of the basic demodulator is a sinusoid, where positive values represent one symbol and negative values represent the second symbol. Therefore, zero crossings are indicative of symbol transitions. The absolute value operation inverts the negative values, resulting in a signal without zero crossings. Instead, symbol transitions are depicted by areas where the signal drops to zero, followed by a rise to the maximum amplitude. Additionally, passing the signal through the absolute value eliminates the actual binary information. Since we are only interested in measuring the

symbol delay, the actual bits stream is irrelevant, and eliminating this information is inconsequential.

The output of the absolute value operation is passed through a threshold, which will output a stream of Boolean ones when the signal rises above the specified value. From the Boolean stream, only the first True is of importance since it indicates a symbol transition. By placing a rising edge detector after the threshold, our algorithm can detect the first Boolean 1 in the stream and output a Boolean one flag, indicating that a symbol transition has occurred. Otherwise, the rising edge detector outputs a Boolean zero flag.

F. WINDOW CREATION

A scanning range or window, centered at the estimated arrival time, is created. The window consists of three parameters: a lower time limit, the estimated arrival time, and an upper time limit. The lower time limit is used to initiate the scanning process, thus ensuring that every symbol is detected only once. When no symbol transitions are detected due to the transmission of consecutive identical symbols in non-return-to-zero (NRZ) coding, the estimated arrival time is used. Finally, the upper window limit stops the scanning process and is used to determine whether the actual arrival time or the estimated arrival time is passed to the Symbol Interval code, discussed in Section H.

There are two window types used at different times in the Symbol Timing code. The first window is used when the Symbol Timing algorithm is synchronized to the incoming symbol stream; the algorithm can detect symbol transitions when the counter is within the window limits. If the symbol transitions do not occur within the window interval for a specified period, then the algorithm is considered to be out of sync with the incoming data. A second window is then activated and used to resync the algorithm with the incoming stream. To calculate the estimated arrival time, the symbol period, calculated by dividing the sampling rate by the symbol rate, is added to the previous arrival time. Predefined values are then added and subtracted from the new estimated arrival time, resulting in the lower and upper limits of the window. For the first window type, the window should be large enough to allow for delays in the signal but not large enough to encompass two transitions. Therefore, the number of samples should not be greater than the symbol period. For the second window type, there are no restrictions on how large the windows should be. When the algorithm is again out of sync, the second window overwrites the parameters of the first window. Once the algorithm is determined to be synchronized, the algorithm switches back to the first window type.

G. SYMBOL DETECTOR/ESTIMATION

Once the counter reaches the lower limit of the window, the algorithm begins scanning for the symbol transition flag. When the symbol transition flag is detected, the algorithm outputs the current counter time, along with a flag indicating that a symbol transition was detected. If no symbol transitions appear within the window, the algorithm outputs the estimated arrival time with a flag indicating that the value is an estimate. Alongside the symbol detector/estimation code is a counter that keeps track of how many symbol estimations were performed. If the counter reaches 20, then the algorithm is considered out of sync, and the second window type is activated. Any

time a symbol is detected, the counter resets to zeros, and the second window type is deactivated.

H. SYMBOL INTERVAL

The last portion of the symbol timing algorithm is the symbol interval code. The previous arrival time is subtracted from the current symbol arrival time resulting in the symbol timing differential. Afterward, the output is transferred to the host computer for visualization, along with the flag indicating whether the arrival time was an actual arrival or an estimated arrival. Finally, the current arrival time is stored in a shift register and used to calculate the time differential of the next symbol.

ARCHIVING

At the end of the collection process, the output of the oscillator, the IIR filter, and the symbol time differential are stored on the hard drive. For post-collection analysis, $30\ minutes$ of output data is desired. However, streaming the data to the host computer and then to the hard disk is not possible since the transfer speed between the host and the FPGA card is faster than the write speed of the disk. Attempting to transfer the data directly to the disk resulted in data loss. Instead, a two-stage archiving algorithm was implemented. During the collection process, the output data is temporarily stored in separate buffers within the FPGA card. When the user terminates the program or one of the buffers fills up, the data is transferred from the buffers to the host computer and saved on the hard drive one buffer at a time. After the data is saved to the disk, the entire program terminates, and the user is free to analyze the data with third-party software offline.

RESULTS

Using a Keysight Agilent E4432B Digital RF Signal Generator, pure carrier signals, Binary Phase Shift Keying (BPSK) signals, and Binary Frequency Shift (BFSK) Keying signals were generated to test the SDR. Particular features of interest were the ability of the system to lock on to the carrier frequency, track the carrier frequency in the presence of the Doppler effects, and calculate the symbol timing differential.

In the first tests, a $2.2717GHZ$ BPSK signal with a symbol rate of $32ksps$ and an underlying sequence of 4 zeros followed by 4 ones was generated. The first plot in Figure 3 shows the envelope of the signal extracted by the demodulator, while the second plot shows the intermediate frequency signal overlapped with the output of the oscillator. The oscillator output was able to lock to the carrier frequency of the incoming signal while ignoring symbol modulation. The third plot shows the symbol timing differential. From the plot, we can see that symbols arrived at an interval of $1250samples \pm 4samples$, which was expected for the symbol rate. Lastly, the plot at the bottom indicates whether the symbol was detected or estimated. Since the underlying bit sequence was 4 zeros followed by 4 ones, we expected one symbol detection followed by three estimates.

In addition to locking on to and tracking BPSK signals, the PLL can lock on to a Binary Frequency Shift Keying (BFSK) signal. A BFSK signal at $2.2717GHZ$ with a symbol rate of $32ksps$ and

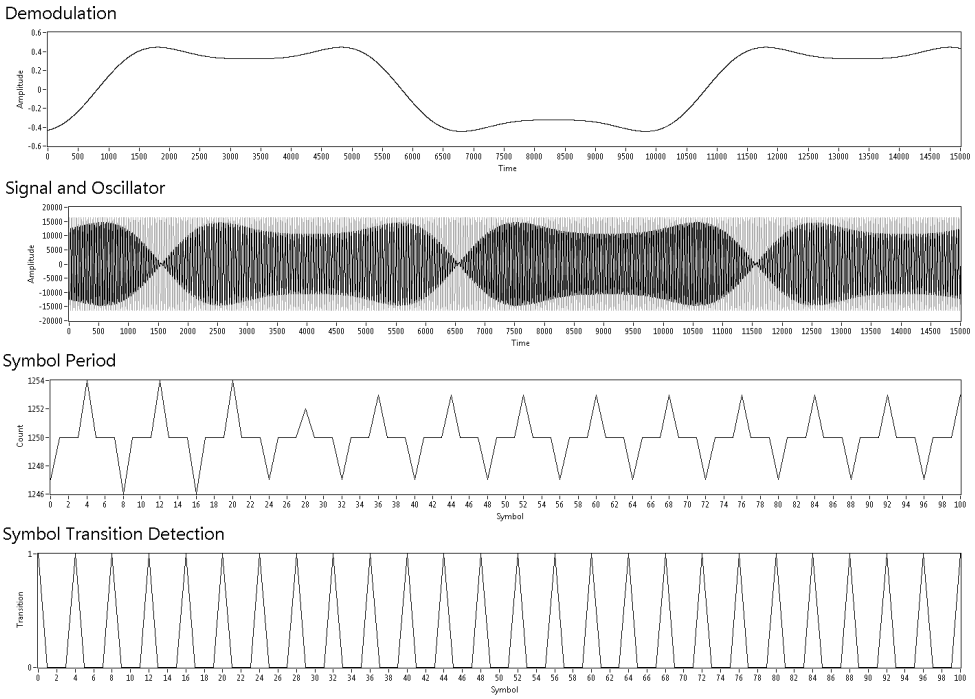

Figure 3: SDR Output for BPSK Signal at 64ksps

a frequency deviation of $1kHz$ was generated. With BFSK, the frequency of the signal changes with the symbol, where one symbol has a high frequency and the second symbol a low frequency. Studying the second plot in Figure 4, we see that the output of the oscillator is matched to the signal coming from the VST. By inspection, the two different frequencies are differentiated. The first plot in the figure is the output of the IIR filter and has an oscillatory pattern. Since the frequency of the signal changes from high to low and vice versa, the IIR filter must transition from high to low and vice versa to track the carrier frequency of the signal.

A BPSK signal centered at $2.2717GHz$ with a symbol rate of $256ksps$ was generated to test the tracking capabilities of the PLL. The frequency was slowly incremented in steps of $100Hz$, and the last frequency to sustain a lock was recorded. Once done, the process was repeated, but instead of incrementing the frequency, the frequency was decreased. The smallest frequency to sustain a lock was recorded. Based on this experiment, we determined that the PLL was able to stay locked to the carrier frequencies at $\pm 15kHz$ from the center frequency. Switching to BFSK, a $256ksps$ signal centered at $2.2717GHz$ was generated and used to test the tracking ability of the PLL. Using the same process as before, the PLL was able to track the carrier frequency at $\pm 12.5kHz$. Since the BFSK changes frequency for different symbols, the change in frequency puts additional stress on the PLL. The PLL has to track the center frequency of the signal as well as the changes in frequency from the modulation.

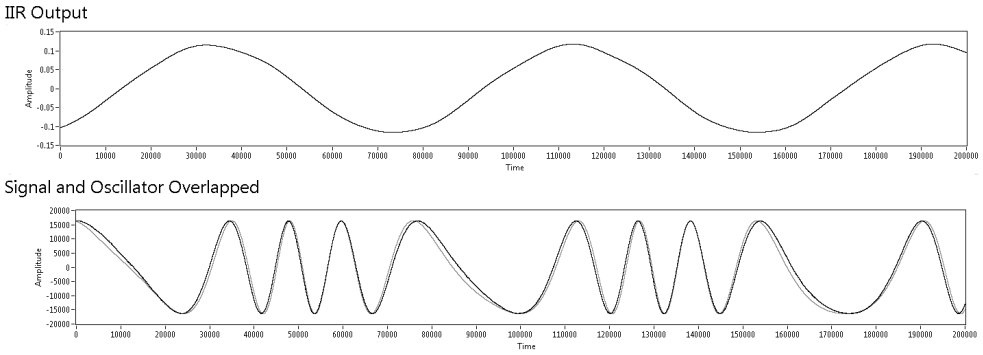

Figure 4: SDR Output for BFSK Signal at 32ksps

CONCLUSIONS

Using LabVIEW, coupled with a vector signal transceiver and an FPGA card, a software-defined radio capable of tracking the carrier frequency and symbol timing variations was developed. Based on the experiments performed, we conclude that the SDR can track the carrier frequency and symbol timing deviations of a satellite in orbit. Through analysis of the resulting output data, it may be possible for our sponsors to estimate the health of the crystal oscillators inside satellites in orbit without physical inspection. In addition, the programmable aspect of FPGA allows for further refinement of the SDR. Therefore, this SDR may become an essential diagnostic tool, assuming that an effective offline analysis algorithm is developed.

REFERENCES

[1] D. K. Weaver, "A third method of generation and detection of single-sideband signals," *Proceedings of the IRE*, vol. 44, pp. 1703–1705, Dec 1956.

[2] B. Sklar, *Digital communications: fundamentals and applications*. Prentice Hall PTR, 2001.

[3] S. K. Mitra, *Digital signal processing: a computer-based approach*. McGraw-Hill, 2011.

[4] S. J. Orfanidis, *Introduction to signal processing*. Prentice Hall, 1996.