Cadet Thomas Dean, KB1JIJ

5 Mohegan Rd, Groton, CT 06340; thomas.dean@usma.edu

A Digital Frequency Hopping Spread Spectrum Transmitter

What can you program an FPGA to do? A digital logic class project leads to this FHSS transmitter.

Software-defined radio is quickly transforming the communications field. Software-defined radio uses software to perform functions such as modulation, demodulation and filtering, all in the digital realm. The advantage to this is that it allows for systems to be built with fewer components than a traditional analog radio architecture, and it allows for an incredible amount of flexibility within the system, even after it has already been produced. There are several common approaches to implementing a software-defined system. One way that most hams may be familiar with can be done on a computer and converted to or from the analog realm through either a sound card or a peripheral such as a Universal Software Radio Peripheral. It is also possible to do the processing on a programmable chip such as a field programmable gate array (FPGA) or through the use of an application specific integrated circuit (ASIC), microprocessors, or DSP chips. In this article I would like to present an example of a project that creates waveforms by using only two components: an FPGA and a video digital to analog converter (DAC). This article will also present an introduction to spread spectrum (SS) systems, which are not commonly seen on the amateur bands.

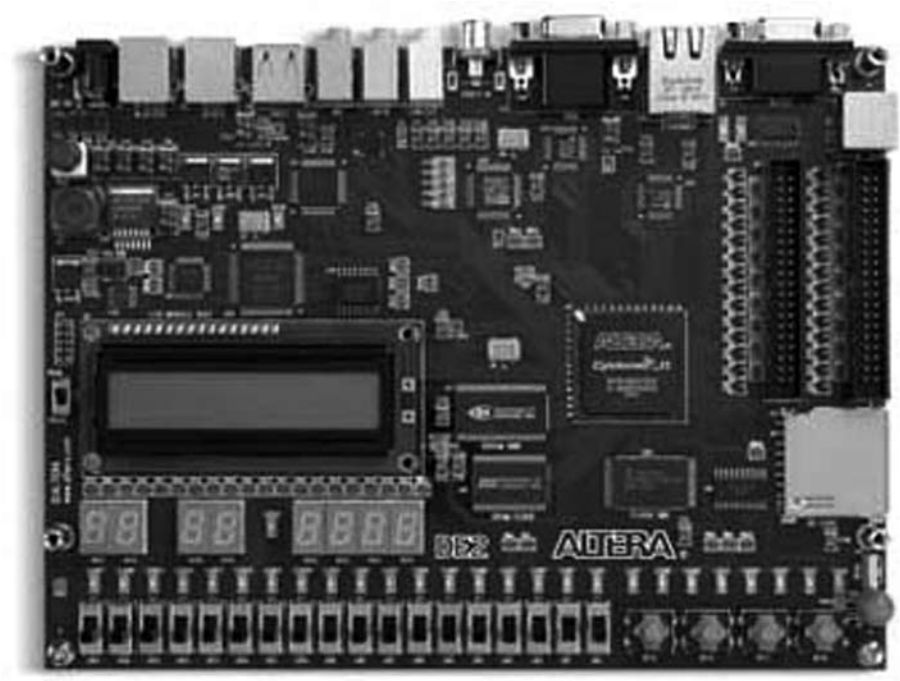

Figure 1 — The Altera DE2 Development and Education Board.

Introduction

This was a project that I designed for a digital logic class. The project was required to be designed on the Altera DE2 Development and Education board, using an Altera Cyclone II FPGA. The DE2 board is designed to allow students to gain experience programming digital systems, minimizing the hassle of fabrication. The board is also useful for prototyping systems before production. This board contains many built in I/O devices and interfaces, as well as on board memory and displays. See Figure 1. For my project, I chose to design a software-defined frequency hopping spread spectrum transmitter. The output of the board contains a signal in the range of 24 kHz to 6.2 MHz. This signal would have to be converted to an amateur band to be used on the air by an Amateur Radio operator. That is an operation that is possible, but somewhat difficult to do. Rather than being presented as a fully working project, this paper presents a demonstration of the sort of system that could be contained in a single chip and how it was created.

The Cyclone II FPGA generates samples of the waveform and then sends them to an ADV7123 Digital to Analog Converter (DAC) to create the IF output of the transmitter. The data rate, frequency range and hopping rate of the transmitter are easily adjusted within the source code. It would not be too difficult to program an interface to allow the user to configure these values. I used a 100 kilobit per second data rate with a 400 kHz hopping rate. There are 16 possible code division multiple access (CDMA) patterns associated with this design, each

[1]Notes appear on page 16

of which has 255 channels that are spaced approximately 24 kHz apart. The total bandwidth of the transmitter is 6.25 MHz.

I developed the transmitter portion of this project, while my project partner, Cadet Adam Royal worked on interfacing it with a hex keypad over the expansion header and the 2 × 16 digit LCD display on the DE2 board. The keypad allowed selection of the CDMA sequence and the LCD displayed the sequence number. In this article, I will present the design and theory behind the transmitter.

Frequency hopping spread spectrum (FHSS) was chosen in part due to the relative simplicity of creating the waveform. In addition, I chose a spread spectrum mode to help me gain a better understanding of spread spectrum systems, which are of great importance in military and many commercial communication systems.

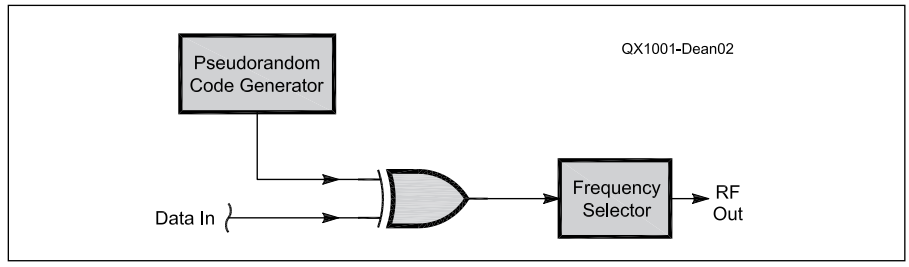

Figure 2 — Here is a simple block diagram of a frequency hopping spread spectrum (FHSS) system.

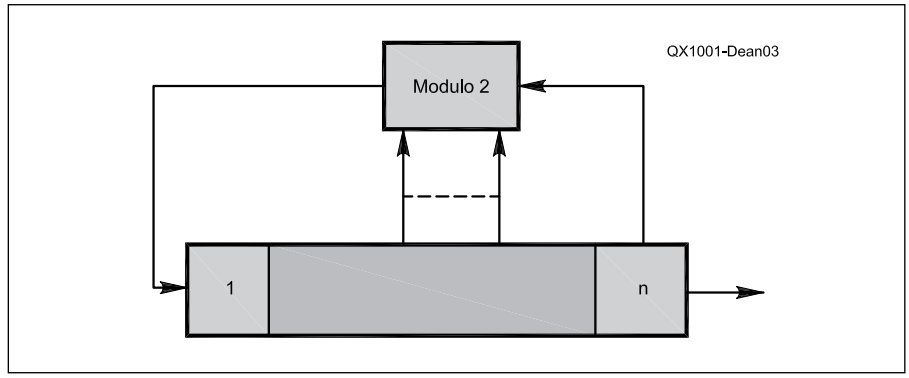

Figure 3 — This is a simple block diagram of a linear sequence generator.

Spread Spectrum Systems

Spread spectrum emissions are not commonly seen on the amateur bands; however, they offer several advantages over traditional narrowband modes. From the point of view of military communications, they make it much more difficult for a signal to be jammed. For example, consider a typical 5 kHz FM voice signal. This could be jammed by simply transmitting on top of it. If the signal was instead hopped over a bandwidth of 30 MHz, it would become nearly impossible to jam unless the jammer had knowledge of the spreading pattern. Commonly used spread spectrum modes in the military include the Single Channel Ground-Airborne Radio System (SINCGARS) and HAVEQUICK system. The use of a spread spectrum system for this purpose is often referred to as an electronic counter-counter measure (ECCM).

Spread spectrum systems are also of great advantage outside of the military. They offer resilience to fading and can often be hidden below the noise floor. By spreading a signal across a large portion of the spectrum, a communications system will have what is known as a process gain. Process gain is the ratio of the spread signal bandwidth to the unspread (baseband) signal bandwidth, and is usually expressed in decibels. For example, if a 1 kHz signal is spread over a 100 kHz bandwidth, the ratio will be 100, or a 20 dB process gain. (For more information on process gain, see **http://en.wikipedia.org/wiki/Process_gain**.)

Depending on the amount of spreading, a system can also gain additional rejection of interference. SS systems can offer a way for multiple users to send data over the same set of frequencies, even though it is a large frequency range. Users must simply choose spreading codes that have low levels of correlation to each other. This is referred to as code division multiple access (CDMA). Most protocols under IEEE 802.11 use spread spectrum systems to allow multiple users on the same frequency range. In addition, the Bluetooth protocol as well as many cellular telephone networks and GPS systems employ a spread spectrum modulation method.

Sequence Number:	Tap positions:
1	8,4,3,2
2	8,6,5,3
3	8,6,5,2
4	8,5,3,1
5	8,6,5,1
6	8,7,6,1
7	8,7,6,5,2,1
8	8,6,4,3,2,1

Figure 4 — This table shows the tap positions for maximal codes.

Programming FGPAs

Before I get into the details of the transmitter design, I will introduce the main component of the design — the FPGA — for those who aren't familiar with it. Programming an FGPA is slightly different from writing code for a computer or microprocessor. An FPGA is a very large array of logic cells. Each cell contains a number of inputs and outputs, and can be programmed to perform a certain logic function. Cells can also function as counters, adders, or in some cases multipliers. Most FPGAs also have built in SDRAM and sometimes incorporate PLLs or DSP units into their design. When you write a program for an FPGA, rather than defining a set of instructions for a processor to follow, you are describing how the cells within the FPGA are connected.

FPGAs are programmed with a hardware description language (HDL). This project was coded using *VHDL* (Very High Speed Integrated Circuit HDL), which is similar in syntax to *ADA*. Programming an FPGA can be very tricky at first. This may be especially true for someone who is familiar with computer programming. The main difference is that your code is executed in parallel rather than sequentially. Once the code is written, it can be synthesized to the Register Transfer Logic (RTL) which is a description of how registers and logic elements are connected. From this, the circuit's actual wiring can be derived. This is useful for purposes of simulation, which can be used to determine if your circuit will execute properly before it is programmed. From here, the circuit is fitted to the chip and complied into a line code that can be programmed on the chip. This is generally done using a program that is provided by the chip manufacturer. The software package used in this project is *Quartus II*, a trial version of this package is freely

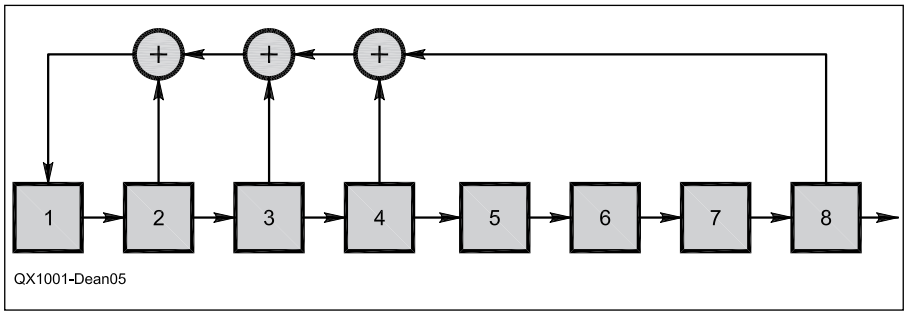

Figure 5 — This block diagram shows an 8-bit maximal linear code generator. The circles with a plus inside represent 1 bit adders (with no carry out), but can also be considered as exclusive OR (XOR) gates.

	1	2	3	4	5	6	7	8
1	-	6	7	6	5	4	4	5
2	6	-	13	6	10	4	4	7
3	7	13	-	5	9	4	4	6
4	6	6	5	-	8	6	5	6
5	5	10	9	8	-	8	5	10
6	4	4	4	6	8	-	7	5
7	4	4	4	5	5	7	-	6
8	5	7	6	6	10	5	6	-

Figure 6 — This table gives the cross-correlation values for 8-bit sequences.

available from Altera. The DE2 board is programmed from a computer USB port using a JTAG interface. The programming files can also be stored on non-volatile EEPROM, which is loaded onto the FPGA each time it is powered up.

Design Theory

Frequency Hopping Spread Spectrum Communications

There are many different methods for spreading data across a wide bandwidth.[1] Two of the most common methods are direct sequence spread spectrum (DSSS) and frequency hopping spread spectrum (FHSS). DSSS takes a signal and multiplies it by a pseudorandom noise signal. As a result, the signal resembles white noise and simultaneously occupies a large bandwidth. In contrast, FHSS systems use a pseudorandom sequence (known to both the transmitter and receiver) to change the carrier frequency.[2] In my transmitter, the frequency changes at a rate faster than the data transmission. FHSS is essentially a form of frequency shift keying.

The data is added to the pseudorandom sequence by an exclusive OR (XOR) digital logic operation. Then the data can be recovered by XORing the sequence again, since $A \oplus B \oplus A = B$. A block diagram of a typical FHSS transmitter is shown in Figure 2.

Generating Pseudonoise

One component that is critical to designing a spread spectrum system is the pseudorandom code generator. The code that is generated must be deterministic so that it can be decoded by the receiver. Generally, the use of a spread spectrum signal is not to protect against eavesdropping; this is accomplished more effectively through encryption. The two important features that must be considered in the code are called cross-correlation and autocorrelation.

Autocorrelation is a measure of similarity between the code and a phase-shifted version of itself. It can be defined as the number of times that the values of a phase shifted version of the code are equal to the values of the original code. This property is of extreme importance to our system, due to its coherence. For a signal to be decoded, the receiver must be able to synchronize its code sequence to the transmitted code sequence (a process referred to as correlation). If a sequence has a high autocorrelation then it may become difficult to determine the correct point to correlate the codes.

Another important property of the code is cross-correlation. This property is very similar to autocorrelation, but measures the amount of correlation of one sequence to a different sequence. This property is important to our system if it is to allow for CDMA. Two communicators may share a single set of frequencies only if there is a low level of correlation between the set of codes that they use.

I first chose a linear code for our system. A linear code can be generated by a shift register with a configuration similar to the one shown in Figure 3.

Consider a code that is generated by a sequence generator that has n stages, then it would have a length of 2^n–1 (a value of zero is not possible). Dixon defines a code to be a "maximal sequence" if the following properties apply to this code:[3]

1) Every possible state exists at some point.

2.) No values repeat themselves within one period of length 2^n–1.

3.) The number of ones in a sequence equals the number of zeros in the sequence plus one.

4.) The statistical distribution of ones and zeros is well defined and always the same.

A maximal code would be the ideal code to use for our system since it has the lowest possible autocorrelation and is easy to generate. Generally, the larger the number of states in a system, the better. It will be difficult, however, to generate a waveform at a precise frequency using strictly digital logic. Therefore, it is best not to choose too long of a sequence. A length of eight bits seemed like a reasonable medium. There are sixteen possible sets of maximal codes that can be made from an eight stage generator. The table in Figure 4 shows half of these possible codes, the remaining eight are the mirror images of these codes.[4] Figure 5 will clarify how these sequences are defined.

This generator is a shift register that feeds back into itself. With each clock cycle, the bits are shifted to the right one register. The register is tapped at various locations as shown, which allows the values on the register to be changed in a cyclic process with a period of 255 clock cycles. Since this code will be used to select frequencies, we will want the output to be in parallel. Using the values from each register at each clock cycle, we will end up with 255 states, corresponding to 255 different frequencies.

Each of the sixteen different possible sequences will correspond to different channels that can be used in the CDMA scheme. We are therefore interested in the cross-correlation between each sequence, as this will show how often the channels will interfere with each other.

The table in Figure 6 shows an index of cross-correlation between half of the codes, generated in-phase with each other. It was calculated by generating every possible code in a Microsoft *Excel* spreadsheet, and then comparing the values of the codes. The index represents the number of times that the two

codes shared the same value. With the exception of approximately five occurrences, this only occurred at the beginning and ending of the codes.

This data only shows a very small portion of the possible cross-correlation values since it is very rare that two codes will be exactly in phase. Studies have determined that a maximal sequence of length eight will overlap the remaining fifteen other maximal sequences between 31 and 95 times.[5] While this may not result in any gain in spectral efficiency, it would still be possible to achieve error free communication using this scheme with sufficient forward error correction.

Convolutional Coding

The purpose of forward error correction (FEC) is to add a level of redundancy to a transmission to allow the receiver to correct some errors.[6] These errors can come from many sources including thermal noise, jammers or communicators on other channels. There are many ways to implement this redundancy. One form of forward error correction that is very simple to implement and provides us with a relatively high coding gain is convolutional coding. While this code is very simple to encode, it is rather complicated to decode. This can be accomplished through what is known as a Viterbiti decoder;[7] however, that is beyond the scope of this article since this project simply focuses on the transmitter. Convolutional codes are usually specified by three parameters:

n = number of output bits

k = number of input bits

m = number of registers

Not all forms of convolutional code create good codes. One commonly used form of this code[8] is (2,1,7), with tap positions defined as:

P1: 1111001

P2: 1011011

This can be done with the encoder shown in Figure 7.

This encoder is a shift register, similar to the pseudonoise generator described earlier. Instead of XORing bits and feeding them back into the register, they are instead passed as two separate lines. This provides an amount of redundancy within the system. In order to use the output of this encoder, it is converted into an 8 bit parallel line at a lower clock rate, where it can be XORed with the pseudonoise bit for bit.

This simple coding technique can allow signals that are up to 5 dB weaker to be received with the same error rate.[9] The disadvantage to using this coding technique is that it doubles the amount of data that is sent. This means that twice the energy is being sent by the radio, resulting in a coding gain of 2 dB (5 dB – 3 dB). It is also highly effective to add Reed-Solomon coding before performing convolutional coding. Since Reed-Solomon coding is a form of block error correction, these two forms of error correction complement each other very well and result in a significant code gain. Reed-Solomon coding works by creating a polynomial out of the data and then adding redundancy by oversampling the polynomial. Creating a Reed-Solomon encoder on the register level is not as hard as it might seem, but the theory behind how it is done lies in abstract algebra. These two forms of error correction are used together in most deep space and satellite communications.[10]

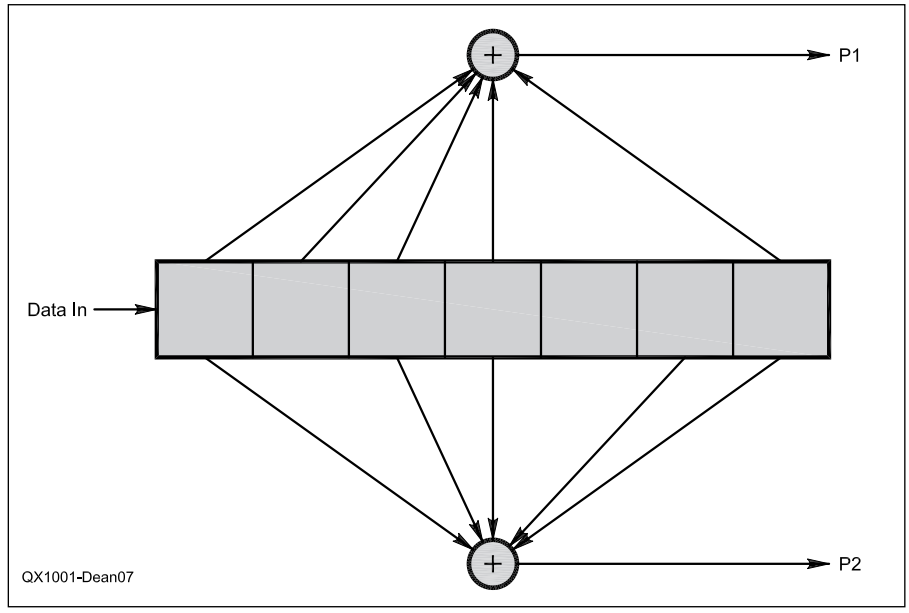

Figure 7 — Here is a block diagram of a convolutional encoder.

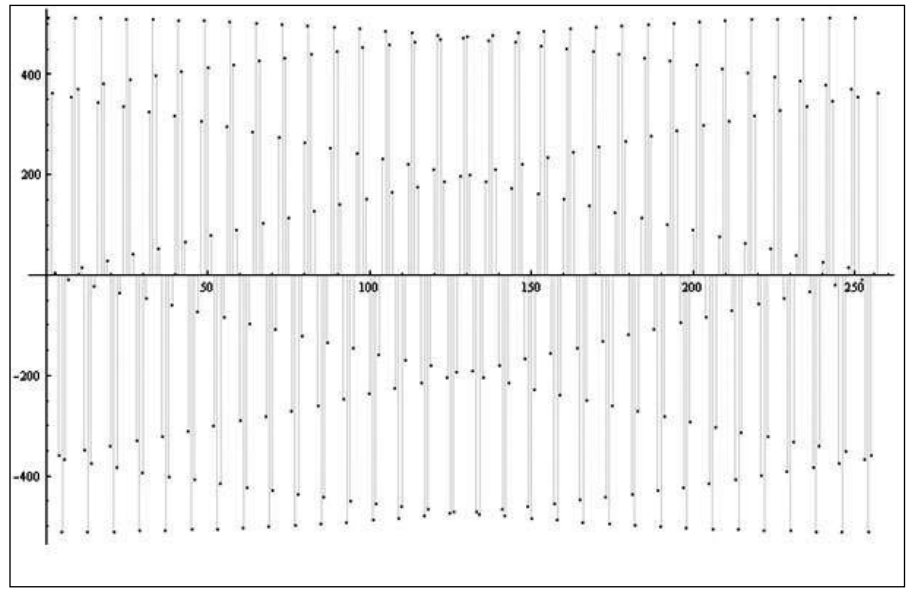

Figure 8 — This waveform results from sampling the look up table once every 255 values.

Generating the waveform

After the pseudorandom code is modified by the data, we are left with a series of eight bit values, which can be directly corresponded with a frequency. Determining the best way to digitally synthesize a wide range of frequencies was perhaps the most difficult aspect of this project. The initial method that I investigated involved generating clock signals and then converting these to a sinusoidal wave through tunable digital filters. This method would be complex from the design perspective, but very simple when it actually came to implementing in hardware. This method has been used in the design of wideband CDMA systems before, and is referred to as pulse shaping.[11] Eventually I decided not to use this method, and chose to use a

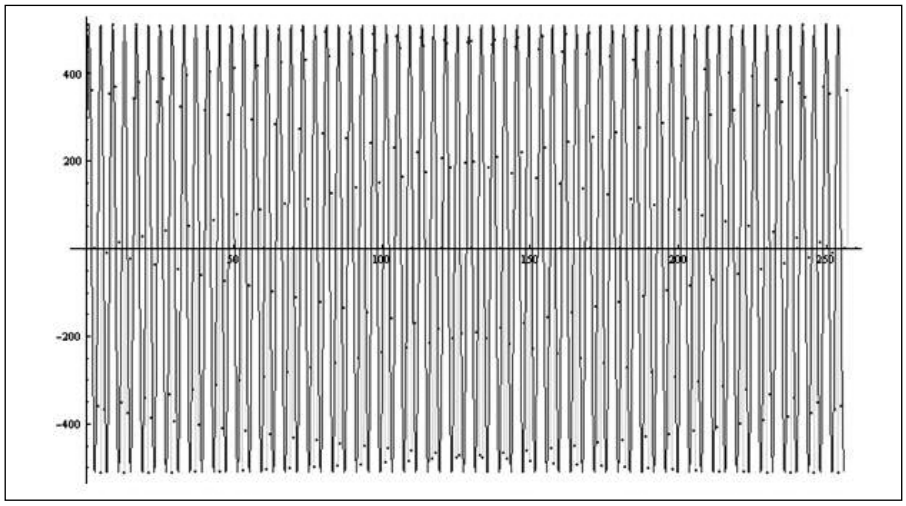

Figure 9 — Here, the signal resulting from sampling the look up table every 255 values is superimposed on the ideal waveform.

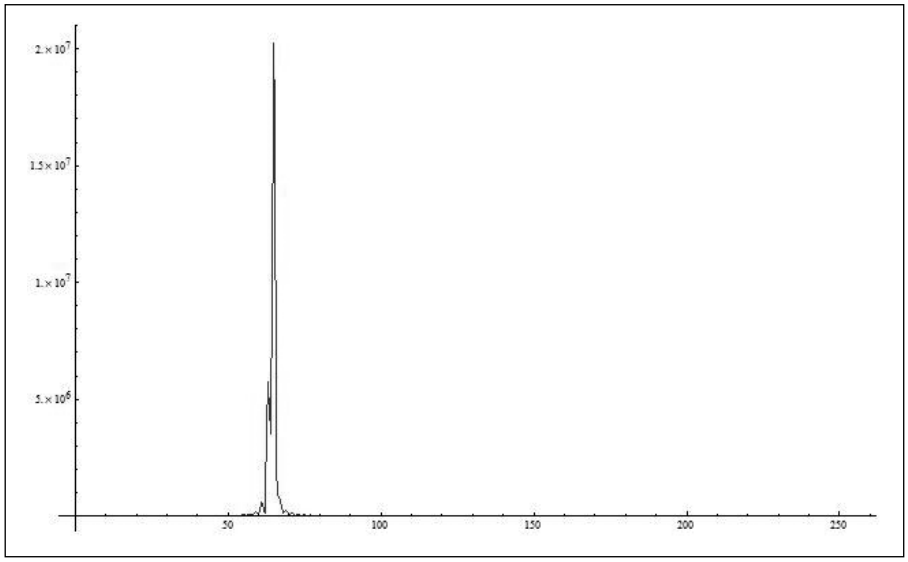

Figure 10 — This is the spectral display of the waveform that results from sampling the look up table once every 255 values.

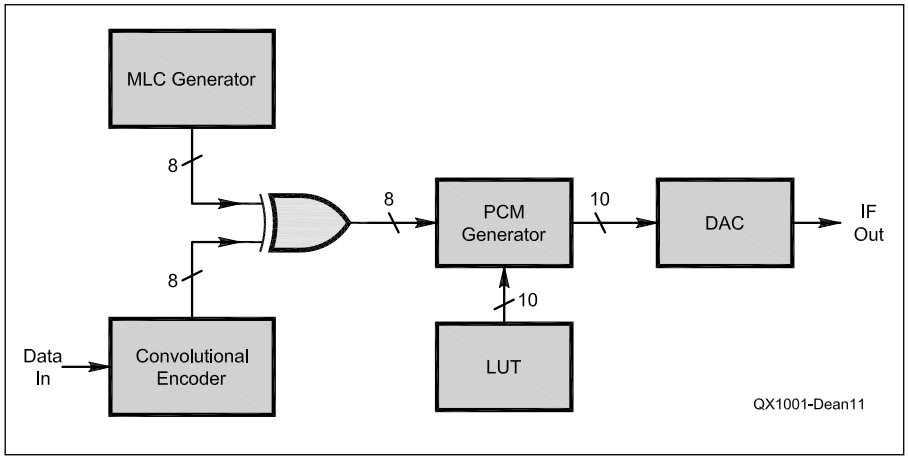

Figure 11 — This is the final block diagram of the transmitter.

large look-up table to generate the wave, once I discovered how easily both *Quartus II* and the Cyclone II handle internal memory.

The DAC that is on the Altera DE2 board uses a process known as zero-order hold to convert a set of digital samples to a piecewise analog function. For this project, samples will be sent to the DAC at a rate of 50 MHz. Each sample will be a ten bit value that corresponds to the amplitude of the resulting waveform. Sine waves will be generated by taking values from a look-up table. To change the frequency, I change the number of times that the table is sampled. For example, if every other value of the table is taken, then the resulting waveform will be twice the frequency if all values had been sampled.

Look-up Table

The look-up table must have, at a minimum, twice the number of possible frequencies because of the Nyquist criteria. Since there are 256 possible values for the frequency, there must be a minimum of 512 values in the table. Since storage inside the chip is not an issue, I chose to double this to improve the quality of the waveform generated. Then, because of the symmetry of the sine wave, the number of entries can be reduced to a quarter of a sine wave. This results in 256 values in the table. The word size within the look-up table must be equal to 10, since we are using a 10-bit DAC. The final amount of memory needed is $2^9 \times 10$ or 5,120 bits. A *Perl* script was found on-line to automate the process of making this look-up table.[12] It came with VHDL code that could be used to access the table to generate a sine wave. The code had to be modified in order to allow for the sampling interval of the table to be modified with input. In addition, the code used two's-complement in order to make the values of the wave signed. The code had to be modified in order to make the values unsigned.

Spectra

In order to determine the frequencies that result from sampling our table we must consider that the table will be accessed at a rate of 50 MHz or 5×10^7 times per second. If n is equal to the output of the modified pseudonoise generator (a value from 1 to 255), then the resulting frequency will be:

$$f(n) = 5\times10^7 \frac{n}{2048} \qquad \text{[Eq 1]}$$

where 2048 is equal to the period of the table. This means that the system will have outputs between 24.41 kHz and 6.225 MHz.

When we sample the table at intervals that are not factors of 2048, the resulting waveform will not be purely sinusoidal. Consider when the table is sampled at an interval of 255. The resulting signal can be

expressed as:

$$e^{\frac{-255\pi\ in}{1024}} \qquad \text{[Eq 2]}$$

where n is a positive integer. Figure 8 displays the waveform graphically. Figure 9 compares this signal to the desired waveform, Sin ($255\ \pi\ n$ / 1024).

It is evident that the waveform is simply not sampled at values corresponding to peaks of the wave. To ensure that this imperfection has no negative impact on the system, it can be converted into the frequency domain using a discrete Fourier transform. The spectrum of the wave is then equal to the square of the transform, which is shown in Figure 10.

The desired signal is about 6 dB higher than any unwanted products. While this is not ideal, it should be small enough not to cause unwanted effects on the receiver. The waveform generated at this frequency is in fact the worst possible waveform. The amplitude of the output signal in Figure 8 appears to show a low beat frequency. Normally, the output signal would be put though a reconstruction filter to remove any alias products. If we filtered out any alias products, we would be left with a constant amplitude sine wave.

Imperfections in the Waveform

One issue that is presented when converting a digital signal to an analog is the result of imaging occurring at higher frequencies, as well as other imperfections that end up in the resulting waveform. Due to its zero-order sample-and-hold operation, a digital to analog converter (DAC) will produce a distortion in the output spectrum that follows the function sin(x) / x.[13] At 80% of the Nyquist frequency, this attenuation will be around –12.6 dB.[14] The maximum frequency reached by the system is 25% of the Nyquist frequency. While this may cause some roll-off, it should not be large enough to have a significant impact on the performance of the system. The roll-off could be countered by using a digital filter prior to the DAC to counter the effects.[15] Another major issue in the resulting signal comes from aliasing that occurs. Replicas of this signal will occur about multiples of the sampling frequency. In some cases, it is possible that these replicas will occur within the radio passband. Thanks to the sample-and-hold nature of the DAC, aliases that fall close to multiples of the sampling frequency will be attenuated; generally, the greater the sampling frequency, the smaller the amplitude of the aliases. One additional form of error that will be present in the waveform will be noise in the output called glitch energy, which is caused by a voltage error in the DAC. Analog smoothing and low pass filters could be used at the output of the DAC in order to improve the final signal. This would help reduce all forms of error mentioned above.

Interfacing with the DAC

The actual interface to the ADV7134V DAC is very straightforward. All of the wiring was already done on the DE2 board. All that had to be done was to properly select the output pins on the FPGA to drive the chip. The DAC was designed to drive a VGA monitor and therefore has separate DACs on chip. The samples were sent to the red input in this project. In order to get output from the DAC, high signals had to be input to the $\overline{BLANK}$ and $\overline{SYNC}$. In addition, the DAC has an output impedance of 75 Ω. Therefore the signal had to be passed through an RF transformer to match the impedance to the commonly used 50 Ω output. The signal was captured by connecting a VGA cable to the VGA port on the board. I cut off the other end of the cable and installed a BNC connector to the red signal in the cable.

Final Block Diagram

The block diagram of the final system is shown in Figure 11. The output is intended to be the IF signal of a radio. Bands which would be good choices would include the 70 cm Amateur band and the 2.4 GHz ISM band.

Simulation and Test Plan

The component that created the digital signal was first simulated on *Active-HDL*, which is an FPGA design and simulation tool. The resulting waveform from the pulse code modulator (PCM) is shown in Figure 12. You can see that the output values increase and decrease in periodic fashion.

Next, the maximum length code (MLC) generator was added to control the input of the PCM generator. Figure 13 shows the waveform from this simulation.

While this waveform seems chaotic,

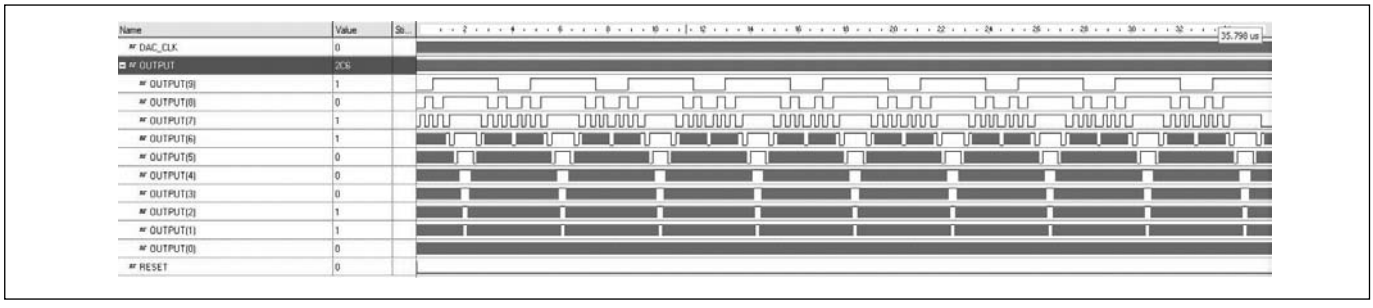

Figure12 — This display is the output of the pulse code modulator (PCM) generator, as given by the *Active-HDL* simulation.

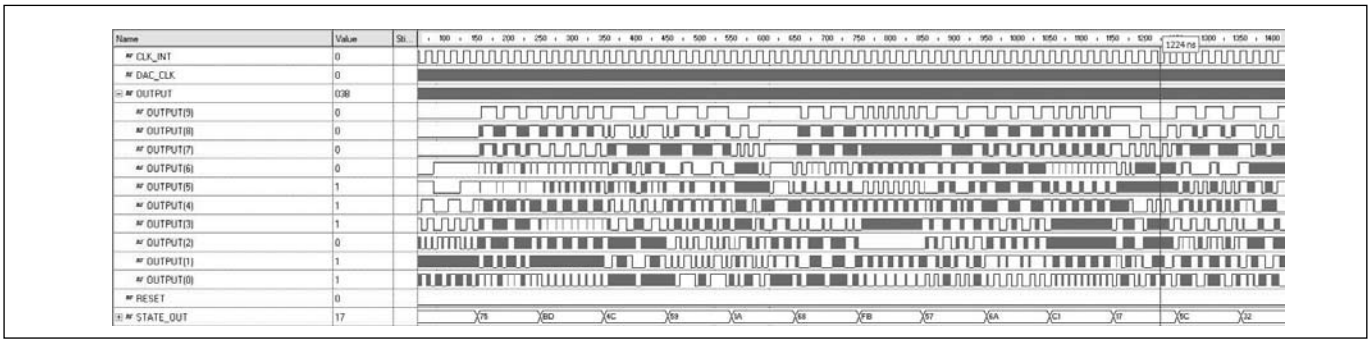

Figure13 — The *Active-HDL* simulation output of the PCM generator tied to the maximum length code (MLC) generator.

you can see that it is functioning properly by simply looking at the fourth line on the graph, just below the two black lines (labeled as Output [9] on the left edge of the graph) and the bottom (State Out) line. The fourth line represents the most significant bit of the PCM. It clearly changes its period when the bottom line, which represents the states from the PCM generator, changes state.

The test bench code that was used was identical in both of these simulations. What made them different was that the MLC Generator was connected on the second one.

The next step in the testing process was to test the output of the VGA DAC. This test served to ensure that all components of the radio were working correctly. It is very difficult to test whether or not the forward error correction is working without a working receiver, due to the pseudorandom nature of the transmitter.

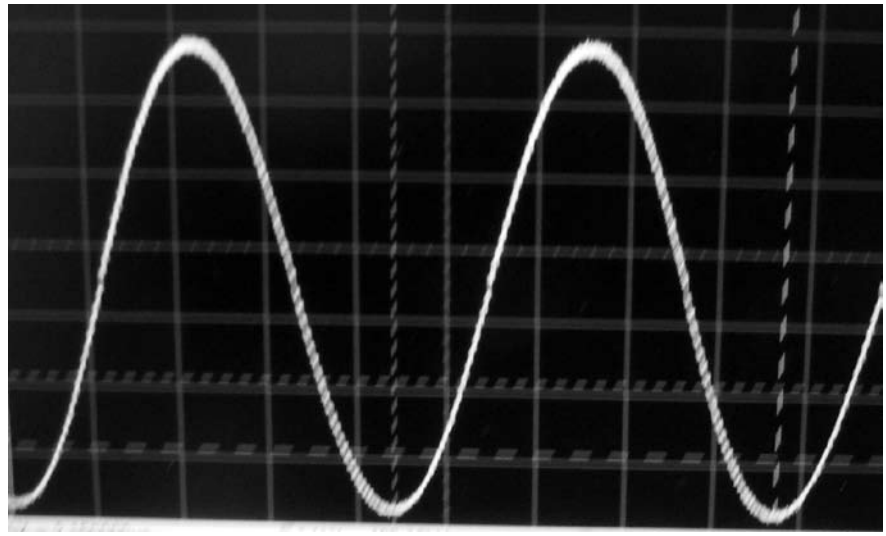

Figure14 — This oscilloscope photo shows the output waveform with the transmitter operating at 480 kHz.

Results

The waveforms that the DAC produced exceeded my expectations. In the time domain, the most evident distortion that exists is caused by the sample-and-hold nature of the DAC. This distortion increases as the frequency of the output increases. In addition, a small amount of glitch energy is present, but this would not have a significant impact on the system. Figure 14 shows the output waveform at 480 kHz. Figure 15 is the output waveform with the transmitter operating at 5.8 MHz.

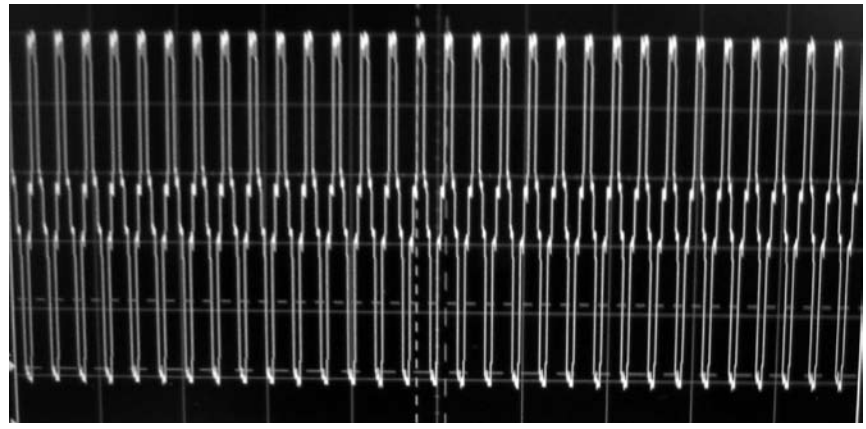

Figure 15 — This oscilloscope photo shows the output waveform with the transmitter operating at 5.8 MHz. Note that this should correspond to the "worst case" waveform, as discussed in the text.

Looking at the spectrum of the waveforms can give a little more insight into the performance of the radio. Bandwidth measurements may not be highly accurate because the spectrum was created by having a 200 million samples per second oscilloscope take fast Fourier transforms (FFT) of the waveform. It should be sufficient, however, to provide a good idea of the functioning of the transmitter. Figure 16 shows the 66 kHz output in both time and frequency domains.

Figure 17 shows perhaps the worst case output of the radio. A product with significant amplitude lies within the 6 MHz bandwidth of the radio. The primary product is 1.36 MHz. The first major product that is produced after that is 30 dB lower than the intended product and at about three times the original frequency. This is an alias of the original signal. Figure 18 shows the spectrum of the highest frequency generated by the transmitter, at about 6.2 MHz.

Figure 19 shows a product that occurs outside of the 6 MHz passband of the radio. This image occurs at three times the fundamental frequency and is 20 dB lower than the intended product. Since this lies outside the passband of the radio, it would be easy to

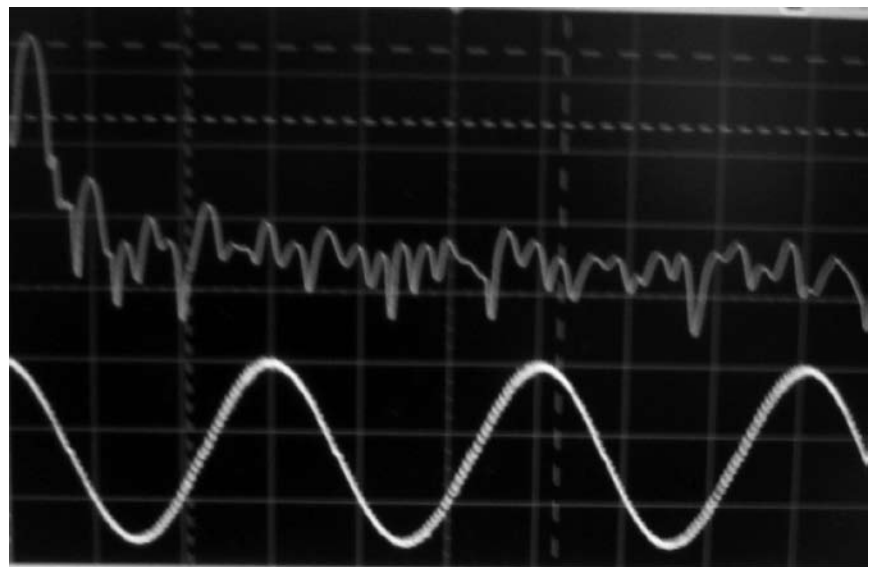

Figure 16 — Here is an oscilloscope photo of a 66 kHz wave and its corresponding spectrum. The peak of the spectrum is over 60 dB greater than other products.

remove from the final signal.

Conclusion

This project was intended to explore how to develop an FPGA-based transmitter, as well as spread spectrum systems. The waveforms that we created show that it is very possible to use simply an FPGA and a DAC to create direct RF signals, or to serve as an IF signal in a transmitter. The waveform that was created here could easily be improved. One significant improvement would be to increase the clock speed within the system. Altera claims that the DE2 board can be driven as fast as 150 MHz. At this speed, timing would become a big issue. Increasing the clock speed would allow a much faster sample rate, which would in turn reduce aliasing and increase the maximum frequency that can be created on the board. Additionally, the size of the look-up table that was used could be significantly increased since the Cyclone II chip has just under 500 Kbits of internal SDRAM. With both of these factors in mind, it should be possible to directly synthesize signals in the entire HF band on the DE2 board. Working with the DE2 board makes experimentation very simple since no soldering is required. Creating a receiver would be slightly more difficult since the board does not contain a high-speed ADC. It is possible that one could be interfaced via the GPIO expansion port. Alternatively, a narrowband signal could be down-converted and then processed with the audio ADC on the board.

Spread spectrum systems are not commonly heard within the amateur bands. This is primarily because most of the advantages of spread spectrum come from their resistance to jamming and their stealth nature. Neither of these are particular concerns in the amateur community. FCC Rules do provide for experimentation with spread spectrum in the 70 cm and shorter wavelength amateur bands, however.

Tom Dean, KB1JIJ, has been licensed since 2003. He currently attends the United States Military Academy at West Point where he is studying electrical engineering. Tom serves as the president of the Cadet Amateur Radio Club, W2KGY. He has taught Amateur Radio license classes and serves as the Liaison VE for exam sessions given by the club. In his free time, besides being on HF, he enjoys running, swimming and reading. His academic interests include software defined radio, satellite communications and complex variable methods in partial differential equations.

Notes

[1]Robert C. Dixon, *Spread Spectrum Systems,* New York: Wiley International, 1984, p 71.
[2]Ibid, p 72.

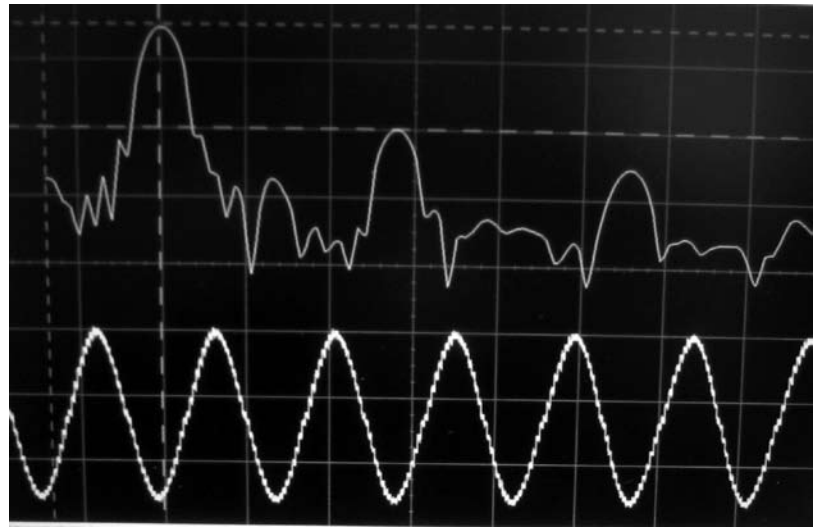

Figure 17 — This photo is of a 1.36 MHz output. The first unwanted product is 30 dB lower than the output of the fundamental frequency.

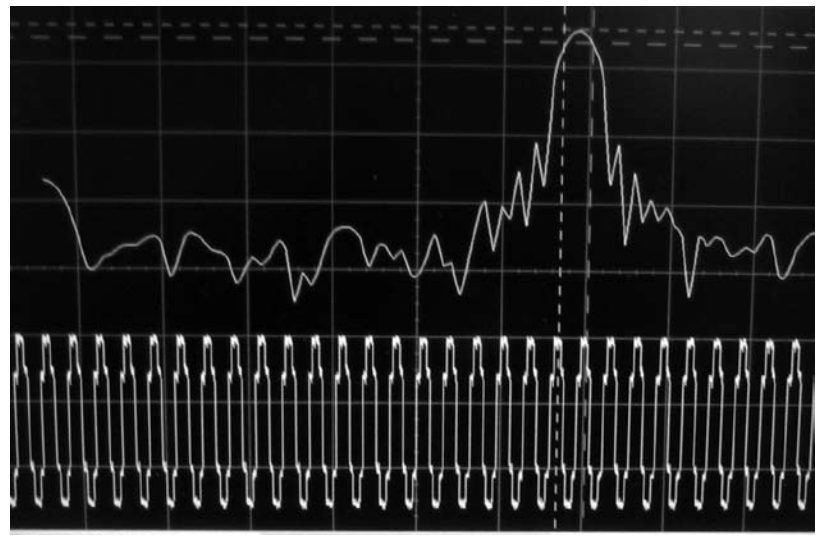

Figure 18 — Here is the spectrum of the highest frequency generated by the transmitter. The 3 dB bandwidth of this signal is on the order of 300 kHz.

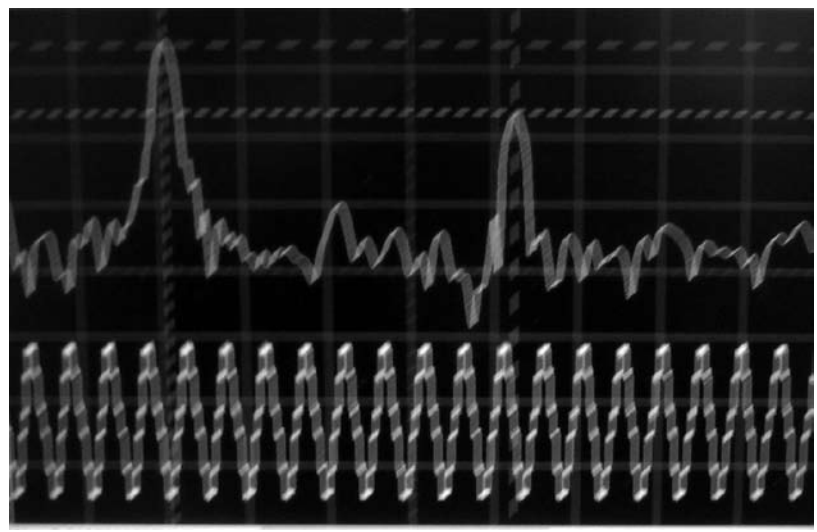

Figure 19 — This oscilloscope display shows the transmitter operating at 4.16 MHz, with a 12 MHz image at –20 dB.

[3]Ibid, p 58.

[4]Ibid, p 87.

[5]Anatol Tirkel, *Cross-Correlation of m-Sequences,* Australia: Monash University.

[6]Mark Wilson, K1RO, ed, *The ARRL Handbook for Radio Communications,* Newington: ARRL, 2009, p 16.2.

[7]Charan Langton, ed, *Coding and Decoding with Convolutional Codes,* Complextoreal, 1999.

[8]Ibid.

[9]Chip Flemming, *A Tutorial on Convolutional Coding with Viterbiti Decoding,* Spectrum Applications, 2006. Accessed from: **home.netcom.com/~chip.f/viterbi/tutorial.html**.

[10]Intel Corporation. *Intel ECC Application Note 108,* San Jose: Intel, 1999.

[11]George Aliftiras, *CDMA Waveform Generation for Digital Radios*, Filtronic Sigtek, p 1.

[12]*Synthesizable Sine Wave Generator,* Doulos. Accessed from: **www.doulos.com/knowhow/vhdl_designers_guide/models/sine_wave_generator/**.

[13]George Aliftiras, *CDMA Waveform Generation for Digital Radios*, Filtronic Sigtek, p 4.

[14]Ken Yang, *Flatten DAC frequency response,* Maxim Integrated Products, 2006. Accessed from: **www.edn.com/article/CA6321533.html**.

[15]George Aliftiras, *CDMA Waveform Generation for Digital Radios*, Filtronic Sigtek, p 4.