CSE466 LAB 4 – Radio Fundamentals

AUTUMN 2011

OBJECTIVES

In this lab you will DO the following:

- Detect the presence or absence of a radio frequency signal.
- Transmit and receive data using software modulation and demodulation.

In this lab you will LEARN the following:

- How to use the Goertzel tone detection algorithm.
- Fundamentals of modulation and line coding.
- CRC implementation details.

DELIVERABLES

The following deliverables are due at the beginning of the next lab period:

- Demo of your wireless communication device.
- A PDF containing answers to all questions posed in this lab prompt.
- Your fully commented and neatly presented code, in a zipped format.

RESOURCES

These documents and web resources will be useful in completion of the lab and/or in answering the questions posed. Additional resources are listed in the text of the lab.

- MSP430F2012 Datasheet
- MSP430F20XX Family User's Guide
- MSP430 Code Examples
- MSP430 Software Coding Techniques
- MSP430 Optimizing C/C++ Compiler User's Guide
- Wikipedia: Line Coding (baseband signaling schemes)

PART 1: CONSTRUCTING THE CAPACITIVE RADIO

In this portion of the lab a simple one-way "radio" link will be constructed. The signal source for this radio link will be an MSP430F2013 with one output pin connected to a metal plate (similar to the E-field sensor plates constructed in Lab 3). A second transmitter plate will be connected to the common terminal of the transmitter. An identical set of metal plates will be the receiver for this radio, which will be connected to another MSP430 (your F2012) through a preamplifier circuit. The circuit used in Lab 3 for E-field sensing will be recycled here for the preamplification of the received signal.

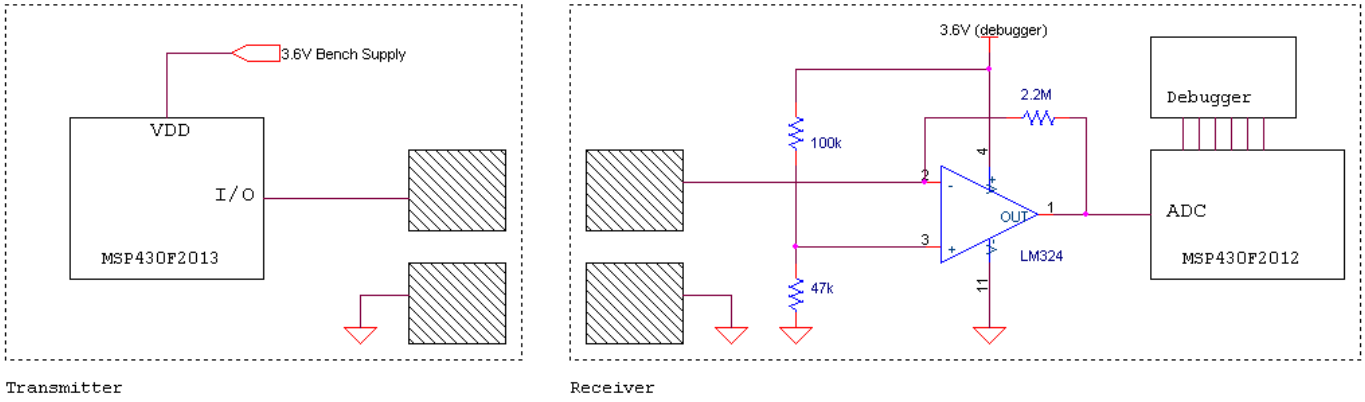

Figure 1. Transmitter and receiver for capacitive radio. Debugger connections and 100nF bypass capacitor omitted for clarity.

PROCEDURE:

1. Construct the receive circuit, using the preamplifier design shown.
2. Construct the transmit circuit. Use the bench supply to provide the F2013-based transmitter with **3.6V**.
3. Generate a 9.25 kHz square wave on the transmitter plate using the MSP430F2013.
4. Using the oscilloscope, view the received signal on the F2012 ADC input. Ensure that these signals appear as expected.

PART 2: TONE DETECTION WITH THE GOERTZEL ALGORITHM

In this portion of the lab, you will implement a software-defined tone detector using the Goertzel tone detection algorithm. This will be coded on the MSP430F2012, and used to detect the magnitude of a particular frequency in an analog input signal. **When the signal is present, an LED will be lit.**

WHAT'S A GOERTZEL?

The Goertzel algorithm is a method of measuring the magnitude of one particular frequency in a sample vector. This algorithm is far less mathematically intensive than the Fast Fourier Transform (FFT) commonly used for frequency analysis, making Goertzel superior to FFT for single-tone detection.

The form of the Goertzel algorithm used is described in a short article from EE Times, available here or at the following URL:

http://www.eetimes.com/design/embedded/4024443/The-Goertzel-Algorithm

You will use the method described in this article, with a few optimizations described below in the "Necessary Optimizations" section.

SAMPLING METHODS

To implement the Goertzel algorithm, you need to collect a set of evenly-spaced samples of the analog input waveform and process them according to the algorithm. This repeated sampling and processing may be done in one of two ways:

1. By placing the ADC10 in **Repeat Single-Channel** mode (User's Guide 22.2.6.3) and writing an ISR to process the samples individually as they arrive. Sampling must be triggered by an accurate timer (TIMER_A).
2. By using the ADC10's **Data Transfer Controller** (User's Guide 22.2.7) to collect all the samples with no software intervention, then processing them all at once after sample collection is complete. Sampling must be triggered by an accurate timer (TIMER_A).

NECESSARY OPTIMIZATIONS

The preliminary steps in the Goertzel algorithm (e.g., calculating values of "coeff" coefficient) **do not need to be performed during runtime** on the microcontroller. Because the detection and sampling frequencies are fixed, these coefficients will not change during runtime and therefore should be pre-calculated before writing the program.

Multiplication operations are not native to the MSP430F2012, and therefore take an extremely long time to compute (100+ cycles). Floating point operations are even worse (400+ cycles). In fact, **in order to meet the timing constraints for this system, you must avoid multiplication or floating point operations during processing of each sample. Select a value for the "coeff" parameter (the parameter used repeatedly during sampling) which is a power of two.** The recommended settings given here for sampling rate and detection frequency result in a "coeff" value of -1, which eliminates the need for any multiplication or bitwise shifting during the sample processing.

To find the magnitude of the detection frequency, use the "Optimized Goertzel" calculation from the article. It's OK to use multiplication when computing magnitude, because this step is only done once for each sample block. However, for performance reasons you should **OMIT the square root computation** required to find the true value of the magnitude. Your result will actually be the square of the magnitude, which will work fine for this application.

OTHER CONSIDERATIONS

In addition to the steps taken in the EE Times article, you will also need to **remove the DC offset** of the sampled signal. One possible method for determining the DC offset is to run an IIR low-pass filter (like that described in Lab 3) over the sample vector. You can then subtract the running average produced by the LPF from each sample prior to the Goertzel computation. If you are using the Data Transfer Controller method to collect samples, you can determine the DC offset by simply finding the mean value of the sample vector before processing.

RECOMMENDED OPERATING PARAMETERS

There are several parameters of this tone detector which must be carefully selected for proper operation. The table below lists some suggested values for each of these parameters.

Sampling frequency	**30 kHz**
Transmitted frequency	**9.25 kHz**
Block size (N)	**20**

PROCEDURE:

1. Carefully choose and implement one of the two ADC sampling methods described above.
2. Verify that your sampling implementation has the desired sampling frequency and is correctly sampling the input signal.
3. Add the computational steps of the Goertzel algorithm to your system.
4. Light an LED based on the value of magnitude^2 produced by the Optimized Goertzel computation. When you disable the 10 kHz transmitter, the LED should turn OFF.

Question 2.1: What are some costs and benefits of each of the sampling methods described? Which of the two methods did you choose?

Question 2.2: Why is it acceptable to use the square of the magnitude instead of true magnitude in this application?

PART 3: TRANSMITTING AND RECEIVING DATA

You will now implement a data link using a software-defined demodulator. You will be using a baseband signaling scheme known as Pulse Interval Encoding (PIE). The carrier wave will be modulated using the AM (Amplitude Modulation) method.

LINE CODING (BASEBAND SIGNALING) – PIE

Pulse Interval Encoding (PIE) is a relatively simple method of **Line Coding,** or encoding a string of **symbols** (typically 1's and 0's) into a pulsetrain. As the name suggests, the interval (length) of each HI pulse transmitted in a PIE system determines the symbol value.

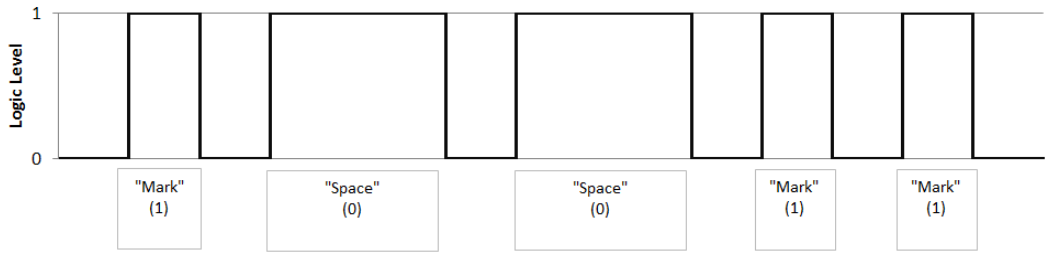

Figure 2. Pulse interval encoding of symbols "10011"

The PIE pulsetrain representing the bits you want to send will be used as the **baseband** signal for your transmitter, and will modulate the amplitude of your transmitter as shown in **Error! Reference source not found.**. At the receiver, this signal will need to be demodulated (converted back to the baseband PIE signal) before the symbols can be recovered. You need to carefully choose the width of each of your PIE symbols based on the measurement rate of the receiver.

MODULATION METHOD – AM

In Amplitude Modulation, a very simple modulation method used for over one hundred years, digital signaling is accomplished by switching ON or OFF a carrier wave. For the purposes of this lab, a logic HI may be represented by a carrier wave which is ON, while a LO may be represented by a carrier which is OFF.

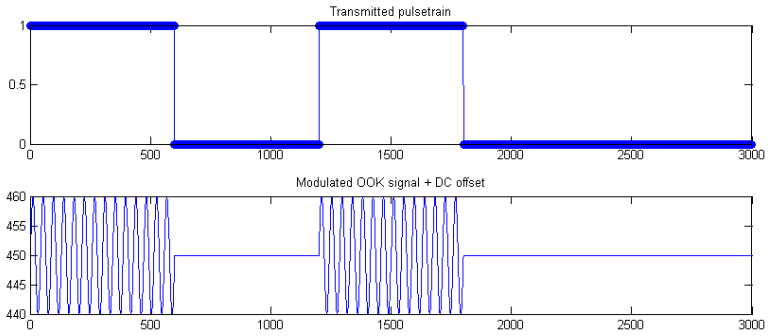

Figure 3. Carrier wave amplitude-modulated with a baseband signal

PROCEDURE:

1. Implement an AM demodulator by thresholding (comparing to a constant) the value produced by your tone detector from Part 2 (probably the same method used to light LED in Part 2). Choose the threshold value carefully based on your observations of the received magnitude.
2. Implement a Pulse Interval Encoding (PIE) transmit function for the MSP430F2013-based transmitter.
3. Implement a PIE symbol detection and data parsing function for your receiver.
4. Verify that your data link is working by transmitting some data and monitoring the resulting received values. Toggle an LED when the expected value is received.
5. Measure the average data rate by determining the number of valid bits received per second.
6. Measure the bit error rate by counting valid bits and invalid bits over a long period of time. Operate the device until at least one “bad” bit is received.

Question 3.1: What’s the average data rate of your radio link, in bits/second?

Question 3.2: What is your bit error rate (number of bad bits divided by total number of bits) ?

Question 3.3: What are some other methods of line coding that might be useful for this device (check out Wikipedia)?

PART 4: ERROR DETECTION

You will now implement the computation of a Cyclic Redundancy Check (CRC) polynomial to transmit alongside the payload data, which will be used to detect errors in the received signal.

REQUIREMENTS

The requirement for this portion of the lab is that you generate a CRC for a one-byte payload. Please be aware that this is not a typical CRC usage. CRC is most useful when computed over a large set of data (many bytes) which make up a data packet. However, for simplicity you need only compute a CRC on a single byte. Study the resources carefully, and you should find that computing a CRC for a single byte is a simple task given the 16 bit registers of the MSP430.

RESOURCES

Slides 16-19 of the Embedded Algorithms deck here list some common CRC polynomials and provide a description of how CRC can be implemented. In addition, pages 25 - 31 of this application note from Philips give some example multi-byte CRC generation code:

http://www.nxp.com/documents/data_sheet/SL080530.pdf

Both your transmitter and receiver must be capable of CRC generation in order to implement error checking. The CRC generated by the transmitter must match the one generated at the receiver in order for the data to be considered valid.

HINTS:

- You can use TIMER_A as a timing reference to determine how long a computation takes:

```
TACTL = TASSEL_2 | MC_2;      // Assuming SMCLK=MCLK=DCO. Start Timer_A in continuous mode.
timeBefore = TAR;             // Capture initial cycle count
CRC = myCRC(Data);            // Perform computation in question
timeAfter = TAR;              // Capture final cycle count
totalTime = timeAfter-timeBefore;// Compute number of cycles used during computation
```

PROCEDURE:

1. Select a CRC polynomial for your one-byte CRC generator. This polynomial should be 8 bits or fewer.
2. Build the CRC generator in a testbench program which is NOT associated with your radio, in order to test and optimize your implementation.
3. Measure the number of cycles your CRC generator requires to do its work.
4. Integrate the CRC generator with your transmitter and receiver.
5. Light an LED when a received data byte does NOT pass the CRC check.

Question 4.1: How many CPU cycles did your CRC calculation take?

OPTIONAL EXTRA CREDIT

There is a lot of room for improvement and elaboration over the methods described in this lab. Up to 2 extra credit points will be given to groups who:

- **Send some interesting data (such as the value measured by the internal temp. sensor) through your radio.**
- **Implement a line coding scheme other than Pulse Interval Encoding, and achieve an IMPROVED data rate.**
- **Implement the CRC on multiple-byte payloads (one CRC computation for multiple bytes).**
- **Integrate your radio receiver with a PC-side GUI.**
- **Achieve the highest average data rate in the class (must have fewer than 1 in 1000 bit errors)**
- **Implement a modulation scheme other than AM, and achieve an IMPROVED data rate (difficult).**