Using Real Signals with Simulated Systems

Joseph P. Hoffbeck
University of Portland

Abstract

Students often find real systems more interesting than simulations, but using real systems in a course can be impractical. One compromise between using real systems and pure simulation is to capture signals from real systems and have the students process them using simulated systems. The advantages of this approach include exposing the students to deviations from the ideal such as noise and timing imperfections, and allowing them to experiment with different solutions in software. An example of this method is presented where a caller identification signal is captured from the telephone system, and is demodulated using the numeric computation package MATLAB.

Introduction

It is often necessary to rely on simulations of complex systems in order to demonstrate their behavior to a class since access to real systems can be limited due to cost, space, and time constraints. While simulations are sometimes the only practical approach, they can be too far removed from real systems to be convincing to the students or to really capture the imagination of the students. Furthermore, simulations often produce results that are too good in that they often do not include the imperfections associated with real systems such as noise, distortion, and timing imperfections.

Capturing signals from real systems and processing them using simulated systems has the advantage of using real world signals recorded directly from actual systems, and at the same time retaining the flexibility and convenience of using simulated systems. Only the instructor needs to have access to the actual system to record the signals, and the students can process the results using appropriate software, experimenting with different methods simply by making changes in software.

To demonstrate this teaching method, a project is described that captures the caller identification (CID) signal that is used to transmit the name and number of a telephone caller, and demodulates the signal using the numeric computation package MATLAB. A recording of a CID signal can be made with a telephone coupler that converts the telephone line signals to line level, or alternatively a recording can be obtained from the author (by sending email to hoffbeck@up.edu). The CID signal is demodulated with the routines in the Communications Toolbox, which is an optional package for MATLAB. Decoding the signal allows the students to discover the name and number that is encoded in the CID signal, gives the students exposure to a real communications protocol, and gives them an interesting introduction to the simulation capabilities of MATLAB. The CID signal is a good candidate for this exercise because it is easily recorded, is relatively simple to decode, is short enough to be examined manually, and is a signal that many of the students have actually used. Furthermore, the students can examine a graph of the signal and can listen to the CID signal by sending it to the PC's sound card which should appeal especially to the students who prefer sensory information[1]. Many other systems could be studied in a similar way, although recording radio frequency (RF) signals would require special equipment.

The Caller ID Signal

The CID signal, which is used to transmit the time and date, the telephone number, and in some cases the name of the calling party, is sent between the first ring and second ring[2,3]. Since the CID signal is terminated if the user picks up the phone, telephone users never hear the CID signal. The signal consists of several sections as shown in Figure 1.

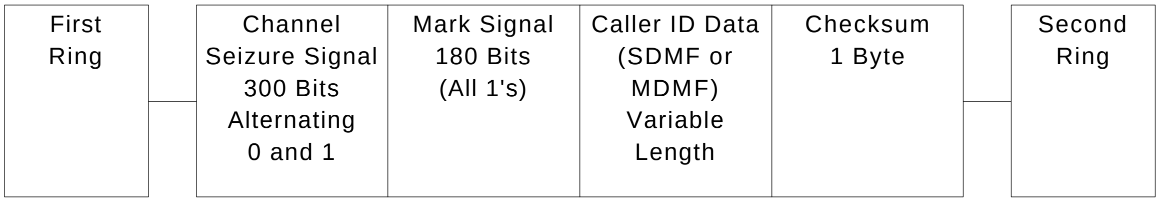
First Ring
Channel Seizure Signal 300 Bits Alternating 0 and 1
Mark Signal 180 Bits (All 1's)
Caller ID Data (SDMF or MDMF) Variable Length
Checksum 1 Byte
Second Ring

Figure 1. The Format of the Caller ID Signal

The channel seizure signal consists of 300 bits of alternating zeros and ones, starting with zero and ending with one. It is used to alert the user's CID equipment that a CID signal is about to be transmitted. The mark signal is 180 bits of all ones and allows the CID equipment to prepare for the CID data. Next is the CID data which can be sent in one of two different formats: Single Data Message Frame (SDMF) which encodes the time, date, and number of the calling party, or Multiple Data Message Frame (MDMF) which allows the name of the calling party to be sent along with the time, date, and number.

The SDMF consists of a header and a body as shown in Figure 2. The SDMF header contains a message type byte whose value is 4 in decimal and a message length byte whose value is the number of bytes in the body.

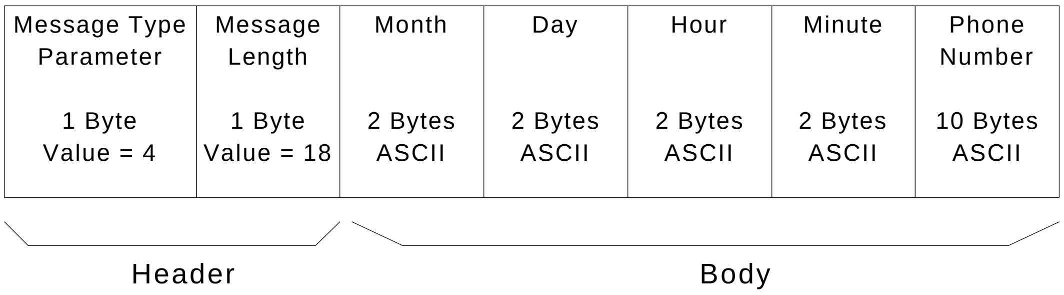
Message Type Parameter 1 Byte Value = 4
Message Length 1 Byte Value = 18
Month 2 Bytes ASCII
Day 2 Bytes ASCII
Hour 2 Bytes ASCII
Minute 2 Bytes ASCII
Phone Number 10 Bytes ASCII
Header
Body

Figure 2. The SDMF Format

The message length byte is followed by the date field (two bytes that represent the month and two bytes that represent the day), the time field (two bytes that represent the hour and two bytes that represent the minute), and the number field (ten bytes that represent the telephone number of the calling party). The date, time, and number are all represented by ASCII characters.

Like the SDMF, the MDMF also consists of a header and a body as shown in Figure 3, but the format of the MDMF is more flexible and allows the name to be transmitted along with the date and number. The MDMF header contains a message type byte with a value of 128 in decimal and a message length byte whose value is the number of bytes following in the body (i.e. the message length value does not include the message type byte or the message length byte). Typically the body of the MDMF format contains three parameter messages, one that represents the time/date, one that contains the telephone number, and one that contains the name.

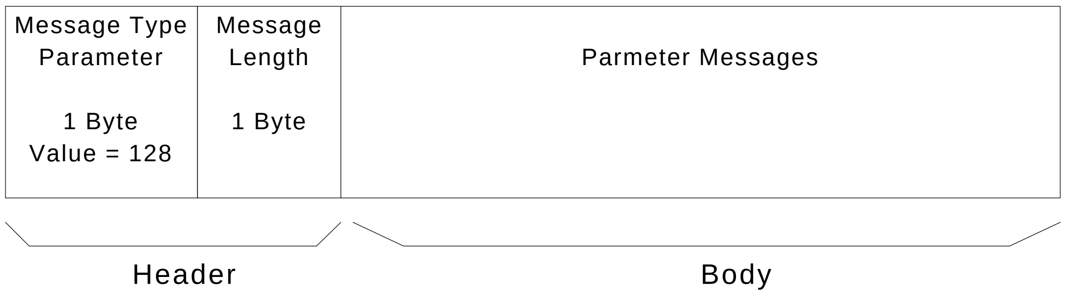

Figure 3. The MDMF Format

As shown in Figure 4, the time/date parameter message begins with a parameter type byte with the value 1 which is followed by the parameter message length byte which indicates the number of bytes in the body. The body of the parameter message consists of two bytes that represent the month, two bytes that represent the day, two bytes that represent the hour (in 24 hour format), and finally two bytes that represent the minute. The month, day, hour, and minutes are represented by ASCII characters.

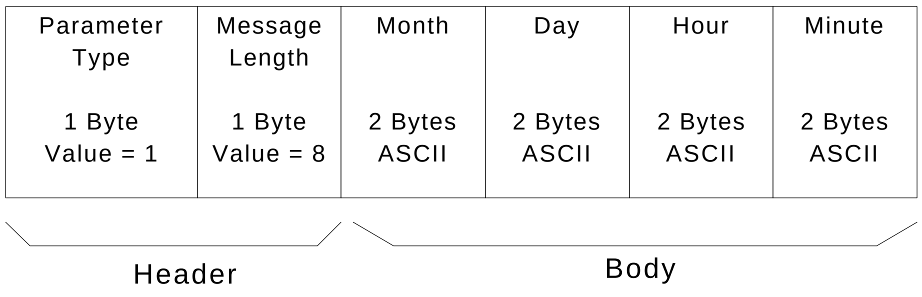

Figure 4. Time/Date Parameter Message Format

As shown in Figure 5, the number parameter message begins with a parameter type byte with a value of 2 which is followed by the parameter message length byte. The body of the parameter message consists of the ten-digit telephone number in ASCII.

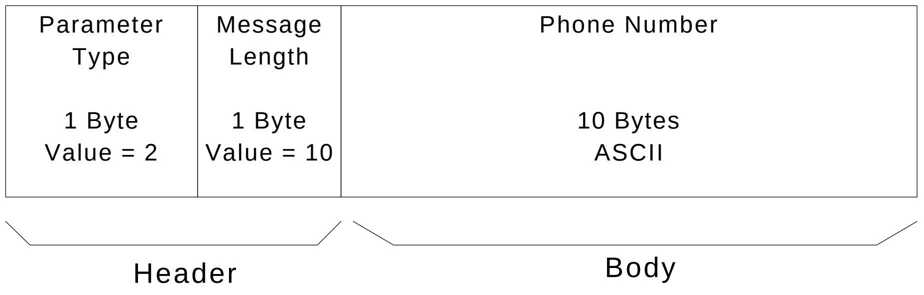

Figure 5. Number Parameter Message Format

The name parameter message begins with a parameter type byte of 7 which is followed by the parameter message length byte that indicates the number of bytes in the body (see Figure 6). The body of the parameter message consists of the name of the calling party in ASCII characters.

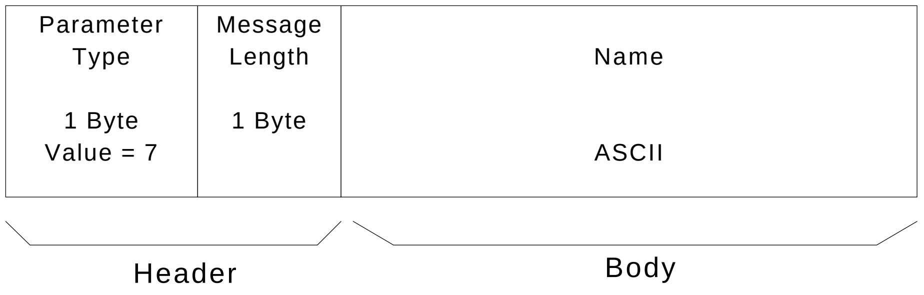

Figure 6. Name Parameter Message Format

The SDMF or MDMF message is followed by a checksum byte (see Figure 1) which is computed by adding, with MOD256 addition, all the bytes in the SDMF or MDMF format. The checksum includes the message type byte, message length byte, and all the bytes in the Caller ID data, but not the checksum byte itself or the channel seizure signal or the mark signal. Then the 2's complement (complement each bit and add one) of the sum is taken. The user's CID equipment computes the checksum on the received data, and if it does not match the checksum byte transmitted by the telephone system, there was an error in transmission. There is no mechanism to correct errors, and not all errors will be detected. For example say the least significant bit (LSB) of a byte was supposed to be a one, but was incorrectly received as zero, and the LSB of another byte was supposed to be a zero, but was incorrectly received as a one, these two errors offset each other and they will not be detected.

The CID signal is sent as an asynchronous serial bit stream using continuous-phase binary frequency-shift-keying (BFSK) modulation with logic 0 represented by a 2200 Hz tone and logic 1 represented by a 1200 Hz tone. The baud rate is 1200 Hz, which means that the duration of each symbol (either a 0 or a 1) is 1/1200 = 833.3 μs. The bits in each byte are sent LSB first, and a start bit (logic 0) is sent before each byte and a stop bit (logic 1) is sent after each byte. Additionally, up to 10 mark bits (logic 1) may be inserted between the message bytes if the telephone system is loaded heavily, and up to 10 mark bits may be transmitted after the checksum.

Recording the Caller ID Signal

A CID signal can be obtained from the author or can be recorded from a phone line with the CID feature available from the local phone company. CID signals are relatively easy to record since they are low frequency audio signals that can be recorded using a standard PC sound card and standard audio processing programs such as the Sound Recorder program in Windows or CoolEdit (www.syntrillium.com). The recording can be saved to a sound file in .wav format which can be read by MATLAB (www.mathworks.com). Alternatively MATLAB can also be used to directly record the signal with the Data Acquisition Toolbox.

It is important to note, however, that only equipment specifically designed to be connected to the telephone system should ever be connected to a phone line. The phone line carries fairly high voltages, such as the ring signal and noise spikes, that could destroy unprotected circuitry. Also, it is important to avoid connecting equipment to the phone line that could damage the telephone company's equipment or present a hazard to people working on the telephone lines.

A telephone coupler is a device designed to connect to a phone line and convert the signal to line level that can be recorded using a standard PC sound card. One manufacturer of telephone couplers is Getner Communications Corporation (www.gentner.com). The CID signal can then be recorded using the system shown in Figure 7.

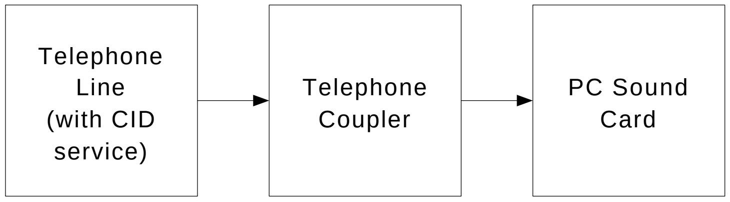

Figure 7. System Used to Record the CID Signal

Since the highest frequency that the telephone line transmits is less than 4 kHz, the sampling frequency used to record the signal can be as low as 8 kHz[4]. The CID signal only lasts about a second, so recording it requires very little memory or disk space.

Decoding the CID Signal

Once the CID signal is recorded and saved in a sound file, the following MATLAB commands can be used to plot the signal (see Figure 8) and send the signal to the speakers so the students can listen to it:

```
[x,fs] = wavread('filename.wav');     % load file (replace "filename" with the name of the file)
plot(x)                               % plot signal
soundsc(x,fs)                         % send signal to speakers
```

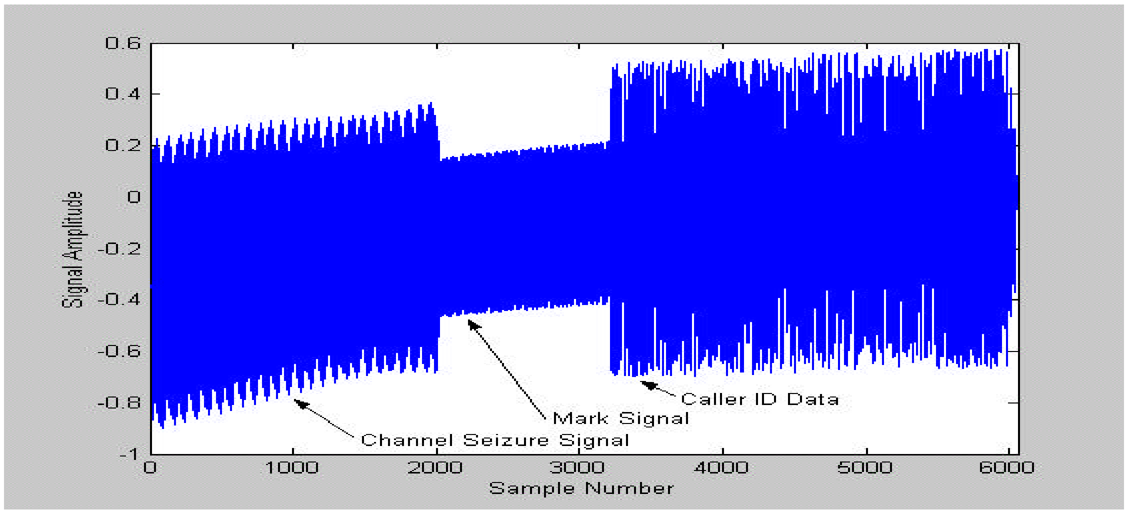

Figure 8. Graph of a CID Signal

The MATLAB routines used to demodulate the signal require that the sampling frequency be an integer multiple of the baud rate, which for the CID signal is 1200 Hz. Therefore the recording must be converted to a sampling frequency such as 9600 Hz (which is 8 times the baud rate and means there will be 8 samples in the signal for each bit). The following MATLAB commands illustrate how to convert a signal that was recorded at 8000 Hz to a signal having a sampling frequency of 9600 Hz:

```
y=resample(x,6,5);          % change sampling rate from 8000 Hz to 8000*6/5 = 9600 Hz
fs = fs*6/5                 % compute new sampling rate and print result
```

The following commands set up some constants that are used by the demodulation routines:

```
fc = (2200 + 1200)/2;       % carrier (center) frequency (Hz)
fd = 1200;                  % baud rate (Hz)
m = 2;                      % M-ary value (m = 2 for binary)
tone = 2200 - 1200;         % frequency separation (Hz)
N = fs/fd;                  % number of samples in each bit (must be an integer)
```

In order to properly demodulate the signal, the location where the bits start and stop must be determined. A routine called findoffset.m is available from the author that computes the correlation for all possible offsets (since there are 8 samples per bit, there are 8 possible offsets), and finds the one with the best correlation. The eye diagram is another method used to find the best offset. Finding the bit boundaries is an example of a required function for real communication systems that is often ignored in simulations.

```
offset = findoffset(y);     % find best offset (this routine available from the author)
y(1:offset) = [];           % delete first few samples to make new offset = 0
y(ceil(length(y)/N)*N)=0;   % pad the vector so its length is an integer number times N
                            % which is required by the routine ddemod
```

Now we are ready to demodulate the signal using the ddemod routine that is part of the Communications Toolbox. This routine determines whether the signal has a higher correlation to a 0 bit or a 1 bit for each bit interval, and returns the result. Non-coherent FSK demodulation is employed since we do not know the phase of the carrier. (In simulations, coherent demodulation is sometimes used because the phase of the carrier is known.) The ddemod routine returns a 0 if the signal matched the lowest frequency and a 1 if it matched the higher frequency, which is opposite of how the signals are defined in the CID signal (logic 0 is represented by a 2200 Hz tone and logic 1 is represented by a 1200 Hz tone). Therefore the ones and zeros must be swapped using the MATLAB symbol for logical NOT "~". Also for convenience the transpose is taken by placing an apostrophe after the variable.

```
z = ddemod(y,fc,fd,fs,'fsk/noncoherence',m,tone);    % demodulate the CID signal
z = ~z'                                              % swap ones and zeros, take transpose
                                                     % and print out bits
```

Now the vector z contains the demodulated signal where each element of z contains a single bit, and it can be examined to extract the CID information. If the recording begins before the beginning of the CID signal, the first few bits will be random ones and zeros based on the noise in the recording before the first bit. Next there should be 300 bits that alternate between zero and one from the channel seizure signal followed by 180 ones from the mark signal. Then the caller ID data begins with the message type byte of 4 for SDMF or 128 for the MDMF format. Note that a start bit (i.e. a zero) will be sent first, then the byte with the LSB first, followed by the stop bit (i.e. a one). It is convenient to have MATLAB convert the 8 bits that make up the byte to decimal and to ASCII as follows:

```
i = 481         % set the variable i to the index of the first start bit (i.e. 0) following the
                % mark signal  in YOUR vector z.  This value may vary.

bits = z(i:i+9),byte = bin2dec(char(bits(9:-1:2)+48)),ascii = char(byte),i=i+10;
                        % Print the bits to make sure the start bit is 0, and the stop bit is 1.
                        % Get the byte and flip the bits around since LSB is sent first,
                        % then convert the bits to decimal and print the
                        % decimal number and the corresponding ASCII character
                        % Then advance the pointer to the next start bit.
```

The command above can be executed repeatedly to convert all the bytes in the message. To repeat a command in MATLAB, simply press the up arrow and then press the return key. If the value of the start bit is not zero, then there may be some extra stop bits between bytes. In this case, the pointer needs to be advanced to the next start bit before the byte is converted.

In order to check for bit errors, the checksum can be calculated as follows:

```
temp = sum([128, 39, 1, 8, etc.])      % sum all the bytes except the checksum byte
                                       % replace the numbers (and the etc.) in the square brackets
                                       % with YOUR converted decimal values
checksum2 = bitcmp(mod(temp,256),8) + 1  % compute MOD256, and the 2's complement
```

If the transmitted checksum byte does not equal the computed value of checksum2, one or more errors have occurred.

A MATLAB program called CallerID_Decode1.m is available from the author which automatically performs all the steps above, extracts the CID information, and performs error checking.

Discussion and Conclusion

Decoding an actual CID signal exposes the students to a real communication protocol with features like the channel seizure signal, mark signal, and checksum. Also, some of the required functions of real communication systems can be illustrated that are often ignored in simulation, such as finding the start and end of bits and words, and the need for non-coherent demodulation. Demodulating a real signal makes it much more clear why these functions are required. The students can demodulate the signal using the MATLAB Communication Toolbox, or can write their own demodulation algorithms and experiment with variations.

Bibliography

[1] Richard Felder and Linda Silverman, "Learning and Teaching Styles in Engineering Education," Engineering Education, Vol. 78, April 1988, pp. 674-681.

[2] LSSGR: Voiceband Data Transmission Interface (FSD 05-01-0100), GR-30-CORE, Issue 2, Telcordia Technologies Generic Requirements, www.telcordia.com, December 1998.

[3] Implementing Caller ID Functionality in MC68HC(7)05 Applications, Motorola Application Note AN1733, Derrick Forte and Hai Nguyen, www.motorola.com, Austin, Texas, 1998.

[4] Modern Digital and Analog Communication Systems, 3rd ed., B.P. Lathi, Oxford University Press, New York, 1998, p. 251.

Joseph P. Hoffbeck

Joseph P. Hoffbeck is currently an Assistant Professor of Electrical Engineering at the University of Portland, Oregon, and had previously worked on digital cellular and PCS telephone systems at Lucent Technologies, New Jersey. He received his Ph.D. from Purdue University and is a member of ASEE and IEEE. His technical interests include communication systems, digital signal processing, and remote sensing. Email: hoffbeck@up.edu