

# FSK Exercises and Experiments

This document outlines the steps you should take to reach a real-time implementation of an FSK transmitter and receiver. First, you are asked to do some simulations using MATLAB to create a detailed model for the DSP code you will have to write to implement the FSK modulation and demodulation algorithms. Then, you are asked to implement the algorithms in real-time with the DSP boards. Initially, you will directly wire a transmitter board in one computer to a receiver board in another computer to check that the algorithms work in real-time. Then you will use the up and down-converters and transmit over the air.

## 1 Simulation and Theoretical Exercises

### 1.1 Simulating Binary Random Data with a Linear Feedback Shift Register

You will need to simulate a sequence of binary random digits to test your systems. Linear feedback shift registers can be used to generate pseudo-noise (PN) sequences that closely model binary random sequences. See Steven A. Treter, *Communication System Design Using DSP Algorithms*, Plenum Press, 1995, for a much more comprehensive discussion. We can also use bit-error rate test sets from the ENEE 428 lab that will connect to the RS-232 serial ports on the TIGER 549/PC boards.

A PN sequence with period  $2^{23} - 1$  can be generated by using a shift register with the connection polynomial

$$h(x) = 1 + x^{18} + x^{23} \quad (1)$$

This specifies the connections to a 23-stage shift register. Let  $d(n)$  be the binary sequence generated by this shift register and let the state of the register at time  $n$  be

$$\mathbf{S}(n) = [d(n-1), d(n-2), \dots, d(n-23)] \quad (2)$$

If the initial state is not zero, the desired PN sequence is generated by the recursion

$$d(n) = d(n-18) \oplus d(n-23) \quad (3)$$

where  $\oplus$  represents modulo two addition or an exclusive-or operation.

A sequence  $d(n)$  can be tested to see if it was generated by this shift register by rearranging (3) and checking that

$$d(n) \oplus d(n-18) \oplus d(n-23) = 0 \quad (4)$$

This result will be useful at a receiver for system testing.

### 1.2 Binary FSK Modulator Simulations

Do a MATLAB simulation of a binary FSK modulator with the following properties:

- $f_c$  = carrier frequency = 2400 Hz

- $f_s = \text{bit rate} = 300 \text{ bps}$ , Manchester encoded
- $\Delta f = \text{tone separation} = 480 \text{ Hz}$
- $f_0 = \text{sampling rate} = 9600 \text{ samples/sec}$

Use rectangular shaped message pulse chips at this point.

Once your modulator is working, do the following:

1. Plot a representative section of the modulated signal.
2. Use the PSD function of MATLAB to estimate and plot the power spectral density for the FSK signal.
3. Plot the theoretical spectrum given by (43) of the FSK notes and compare with the spectrum estimated by PSD.

### 1.3 Simulating an FSK Demodulator in MATLAB

1. First use the FM discriminator you designed for demodulating analog FM to demodulate the FSK signal and plot a representative segment of the output. A sampling rate of 9600 samples/sec should be adequate.
2. To detect the bits, you need to know where to sample the demodulated signal. Devise an algorithm to achieve this bit synch. Use the fact that the bits are Manchester encoded in your algorithm. The algorithm should be able to account for the fact that the transmit bit clock and receive bit clock may be at slightly different frequencies so you have to track the drift between the two clocks.
3. Devise a method to estimate the transmitted bit sequence from the demodulated signal. Your method should perform well even with somewhat noisy signals. An optimum approach is to use a matched filter or correlation receiver when the noise is white. See the standard texts for ENEE 420 Communication Systems like Simon Haykin, *Communication Systems*, John Wiley & Sons, 1994 for a discussion of matched filters. Basically, assuming bit synch has been achieved, you should multiply the demodulated samples in a bit interval, by samples of the Manchester pulse for a 1 and sum the products over the entire bit interval. Then, you should decide a 1 was transmitted if the sum is positive and a 0 was transmitted if the sum is negative. This operation is also called an integrate and dump detector.
4. Transmit a PN sequence. Check that your detector is working properly by making sure the received bit sequence satisfies (4).
5. Repeat the above FSK demodulator steps when the approximate demodulator discussed on page 14 of the Frequency Modulation notes is used. Also consider making the further approximation that  $\tan^{-1}(x) \simeq x$  when  $x$  is close to 0.

## 1.4 Bandwidth Reduction by Baseband Message Filtering

Remember that the bit rate for IS-54 is 10 kbps and that the bits are Manchester or split-phase encoded. That is a logical 1 is converted to a logical 0 followed by a 1 and a logical 0 is converted to a logical 1 followed by a 0. In generating an analog message, a logical 1 is converted to +1 and a logical 0 to -1. Each half of a bit is called a chip, so the chip rate is twice the bit rate or 20 k chips/sec. A basic split-phase pulse generated for a logical one input with rectangular shaping is

$$p_M(t) = \begin{cases} -1 & \text{for } 0 \leq t < T_s/2 \\ 1 & \text{for } T_s/2 \leq t < T_s \end{cases} \quad (5)$$

where  $T_s$  is the bit duration.

When rectangular pulses are used for the chips, the modulated FSK signal bandwidth exceeds the 30 kHz channel allocation. One way to reduce the FSK signal bandwidth is to filter the baseband message signal before applying it to the FM modulator. A colleague in industry has suggested to me that 50% excess bandwidth raised cosine spectral shaping based on the 20 kHz chip rate is adequate. See Section 11.2 of S.A. Treter, *Communication System Design Using DSP Algorithms*, Plenum Press, 1995 for a discussion of raised cosine filters. The frequency response of a raised cosine filter with excess bandwidth factor  $\alpha$  and chip period  $T = T_s/2$  is

$$G(\omega) = \begin{cases} T & \text{for } |\omega| \leq (1 - \alpha)\frac{\pi}{T} \\ \frac{T}{2} \left\{ 1 - \sin \left[ \frac{T}{2\alpha} \left( |\omega| - \frac{\pi}{T} \right) \right] \right\} & \text{for } (1 - \alpha)\frac{\pi}{T} \leq |\omega| \leq (1 + \alpha)\frac{\pi}{T} \\ 0 & \text{elsewhere} \end{cases} \quad (6)$$

with  $\alpha$  in the interval  $[0, 1]$ . The corresponding impulse response is

$$g(t) = \frac{\sin\left(\frac{\pi}{T}t\right)}{\frac{\pi}{T}t} \frac{\cos\left(\alpha\frac{\pi}{T}t\right)}{1 - 4(\alpha t/T)^2} \quad (7)$$

Notice that the impulse response is 1 for  $t = 0$  and is 0 at all other multiples of the chip period  $T$  and, consequently, introduces no intersymbol interference at the chip rate.

A program named `rascos.exe` for computing the raised cosine impulse response can be found in the directory `C:\DIGFIL`. This program also separates the impulse response samples into sets of subfilter coefficients to be used as an interpolation filter bank. See Section 11.3 of *Communication System Design* for a discussion of interpolation filter banks. They will also be presented in lecture notes on pulse amplitude modulation. In our case, a sampling rate of 9600 Hz was specified and the chip rate is 600 Hz. Therefore, the number of samples per chip that must be generated is  $9600/600 = 16$  and a filter bank with 16 subfilters is required. For  $\alpha = 0.5$ , truncating the impulse response to the interval  $[-2T_s, 2T_s]$  which amounts to  $\pm 4$  chip intervals about time 0 should be more than adequate.

The raised cosine filtered pulse generated for a Manchester encoded 1 occurring at time 0 is

$$h(t) = (-1) \times g(t) + (+1) \times g(t - T_s/2) \quad (8)$$

Let  $I(n)$  be the analog levels corresponding to the data bit sequence with the mapping of a logical 1 to  $I(n) = 1$  and a logical 0 to  $I(n) = -1$ . Then, the baseband filtered message is

$$m(t) = \sum_{k=0}^{\infty} I(n)h(t - nT_s) \quad (9)$$

Samples of  $m(t)$  at 9600 samples per second with a symbol rate of 300 bps can be generated by an interpolation filter bank with 32 subfilters. The coefficients for the subfilters can be easily computed from the output of `rascos.exe` and the subtraction indicated in (8).

Investigate the effects of raised cosine filtering by doing the following:

1. Plot the samples of  $h(t)$  generated by `rascos.exe` and the subtraction of (8) assuming  $\alpha = 0.5$ ,  $f_s = 300$  bps, and the sampling rate is 9600 Hz. Use truncation limits of  $\pm 4$  chips for  $g(t)$ .
2. Find the coefficients of the 32 subfilters. Generate and plot a representative segment of  $m(t)$ . Use MATLAB.
3. Apply the samples of  $m(t)$  to your simulated FM modulator. Plot a representative segment of the modulator output.
4. Use the MATLAB PSD function to compute and plot the power spectral density of the modulator output. Compare the spectra of the modulated signals for the cases where  $m(t)$  is unfiltered and filtered.
5. Demodulate the FSK signal for the filtered  $m(t)$  and detect the bits. You may have to modify your symbol clock tracking and detection schemes because the filtered signal does not have sharp transitions.

## 2 Real-Time DSP Implementation

Now that you have developed working FSK modulator and demodulator algorithms using the stable MATLAB environment, it is time to make them work in real-time on the DSP boards. Continue to use a carrier frequency of 2400 Hz, Manchester encoded bits with a bit rate of 300 bps, a sampling rate of 9600 Hz, and the same PN sequence generator. Do the following sequence of tasks:

1. Learn how to use the autobuffering feature of the C549 buffered serial ports by reading the TI manual, *TMS320C54x DSP CPU and Peripherals*. Also look at the TI publication, *TMS320C54x DSP Applications Guide*, Section 3.8 for more details, discussion, and examples. The TIGER 549/PC manual and software also show how to use the buffered serial port in Example 3. The autobuffering unit (ABU) automatically transfers words from memory to the serial port transmitter or to memory from the serial port receiver independently of the CPU. Autobuffering can be enabled or disabled for the transmitter and receiver individually. Autobuffering is the most efficient method

for sending or receiving data samples. It is better than using interrupts because interrupts require the overhead of saving and restoring the DSP state, and also monopolize the CPU while executing the service routine instructions.

The autobuffering unit transfers words from or to a memory buffer with a size set by the user. When a receive buffer becomes full or a transmit buffer becomes empty, the ABU can generate an interrupt to the CPU. The CPU can then process the received frame or generate a new frame and reset the buffer pointers. Typically “ping-pong” buffering is used. Consider the case of transmitting samples. Two buffers of equal size are used. While the ABU is transmitting samples from one buffer, the program is generating a new frame of samples and filling the second buffer. When the transmit buffer becomes empty and the ABU sends an interrupt to the CPU, the buffers are swapped and samples are read out of the buffer just filled by the program while the program fills the previous output buffer with a new frame of samples. Of course, the CPU must be able to fill the next output buffer before the current one is emptied. Processing frames of words rather than single words is the most efficient way of using the autobuffering features.

2. Implement an FSK modulator using raised cosine baseband message filtering. Use the autobuffering feature of the buffered serial port. Generate output frames containing one bit’s worth of samples. Remember that with a 9600 Hz sampling rate and 300 bps bit rate, your interpolation filter bank must generate  $9600/300 = 32$  samples per bit. You should fill one half of the buffer with the 32 samples for the current bit while the samples for the previous bit are sent to the serial transmit register by the autobuffering unit. Therefore, the buffer should have a block size of 64 words and should be aligned on a boundary that is a multiple of 128 which is the smallest power of 2 greater than 64. When the output segment of the buffer becomes empty and generates an interrupt, switch the output pointer to the start of the buffer you just filled and write a new set of 32 samples to the buffer half that was just emptied. Your filter bank should be implemented using assembly language.
3. Implement demodulators for both the real FSK signal on the 2400 Hz carrier and the I/Q signal pair. Again use the autobuffering feature and organize your program to process frames consisting of one bit’s worth of received samples.
4. Team up with an adjacent group. Send the real FSK signal with the 2400 Hz carrier to the A/D converter of the adjacent DSP board. Demodulate the signal and detect the bits. Check that the received bit stream satisfies (4).
5. Repeat the previous item but pipe the I and Q components between the codecs of the TIGER 549/PC boards. You will have to modify your buffers to account for the fact that I/Q pairs are now being transmitted.
6. Attach the I and Q components of the transmitter to the I/Q modulator block and WFE up-converter. Receive the signal at another station using the WFE down-converter and I/Q demodulator. Attach the I/Q signals to the stereo codec and run

your C549 demodulator and bit detector. Check that the detected bit stream satisfies (4).