GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 2001
Lab #11: Digital Communication: FSK Signals

Date: 14 Nov.–20 Nov. 2001

This is *the official* Lab #11 description.

You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section before your assigned lab time. You **MUST** complete the online Pre-Post-Lab exercise on Web-CT at the beginning of your scheduled lab session. You can use MATLAB and also consult your lab report or any notes you might have, but you cannot discuss the exercises with any other students. You will have approximately 25 minutes at the beginning of your lab session to complete the online Pre-Post-Lab exercise. The Pre-Post-Lab exercise for this lab includes some questions about concepts from the previous Lab report as well as questions on the Pre-Lab section of this lab.

The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. After completing the warm-up section, turn in the verification sheet to your TA.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports but the submitted work should be original and it should be your own work.

The lab report for this lab will be **informal.** Discuss your results from section 4.

The lab report will be **due during the week of 26–29 Nov. at the beginning of your lab**.

---

1 Introduction

Perhaps an apt title for this quick introduction to Frequency Shift Keying (FSK) Modems would be everything you wanted to know about FSK Modems but were afraid to ask!

The goal of this lab (and the next one) is to understand a simple modem, the Frequency Shift Keying (FSK) Modem, referred to by the International Telecommunications Union (I.T.U.) as V.21. The V.21 modem communicates 1's and 0's by sending either a 1650 Hz tone or a 1850 Hz tone, respectively, for 1/300 second. Thus the overall data rate is 300 bits/second (one bit is sent in 1/300-th of a second). Even though 300 bps is quite slow compared with the theoretical maximum of 56 kilobits per second over a phone line, the V.21 format is still used in almost every modem call. This is due to the fact that receiving and decoding it is so simple. A V.21 modem call can be received without using difficult techniques such as equalizers, cancellers and matched filters. Furthermore, it can be received accurately even in the presence of a significant amount of noise. For these reasons, V.21 is used as an initial *handshake* between two modems, meaning that V.21 is a way to communicate some basic startup/control information between the two modems. You can hear the V.21 modem tones at home when your V34, V.90, V.92 phone line modem or fax machine starts a phone call. V.21 is also used to transmit caller ID information over the phone line.

2 Pre-Lab

2.1 Generating FSK tones

The FSK signal is the concatenation of sinusoidal tones at two different frequencies, 1650 Hz and 1850 Hz. There is no silence in between the tones, as shown in the example below:

```
fs = 15000;
tn = 0:1/fs:0.005;
phi2 = 0;
xx = [cos(2*pi*1650*tn), cos(2*pi*1850*tn + phi2)];
```

If you plot the signal `xx`, you should notice (in the middle) that the first sinusoid does not smoothly transition to the second sinusoid. This happens because the "phase history" of the concatenated signal is discontinuous when the frequency shifts. A little bit of thought will suggest that you could make the concatenated signal look smoother by choosing a different value for the phase of the second sinusoid, `phi2`. Determine the best value for the phase `phi2`.

2.2 Converting from ASCII to FSK

An FSK modem is a digital communication system, so it transmits zeros and ones. In order to transmit messages in an alphabet, there must be a binary representation for each character in the alphabet. The standard for this representation is ASCII where eight bits are used to represent characters, numbers and special punctuation. For example, the upper-case character `'A'` is represented in ASCII with the number 65, which has an 8-bit form as `01000001`; lower-case `'b'` is 98 in ASCII, or `01100010`.

Therefore, if we want to encode a message, such as `'Hello World'`, for the FSK modem, we must turn the eleven characters of the message into 0's and 1's (blanks are counted as characters). We would end up with 88 bits. Given a vector of bits, we could generate the appropriate sinusoids by agreeing to the convention that 1650 Hz is used for a "1", and 1850 Hz is used for "0".

MATLAB has some useful functions for doing ASCII conversion and also for turning decimal numbers into bits. Check out help on the following functions:

```
DEC2BIN Convert decimal integer to a binary string.
CHAR Create character array (string)
BIN2DEC Convert binary string to decimal integer
```

In addition, try the following MATLAB expression `abs(['ABC';'b01'])`, and note that the ASCII equivalents are returned in a matrix.[1] To test your knowledge at this point, write a MATLAB expression that will give the binary representation for the lower-case character `'b'`. Use the optional argument in `dec2bin` to get an 8-bit answer (with leading zeros).

2.3 Synch Bits and Prefix

In order to send a message, the modem needs additional bits to indicate the beginning and end of the message, and also some bits to initialize the modem so that the bits are synchronized. This is accomplished by sending a pre-ordained pattern of bits. For the implementation in this lab, the preamble sent prior to the message bits will be a sequence of six "01" pairs followed by eight ones.

```
0101010101011111111
```

[1] In MATLAB, a string is actually an array of characters, so `['ABC';'b01']` is a 2×3 array. Use `size` to verify this fact.

The alternating 0's and 1's are used for synchronization, but we will postpone a discussion of this topic until the demodulator and decoder have to be designed.

The choice of eight ones comes from the fact that 255 in ASCII is not normal character, so the decoder can look for eight consecutive ones, knowing that this is not part of any valid message. More consecutive ones could be used if a more conservative design strategy were taken.

For the end of the message, another sequence of ones is used to indicate the end. For this lab, the message will be terminated with a sequence of 16 ones. In between the preamble and the termination, all the "message bits" will be placed.

3 Warmup

The objective of the warm-up is twofold: implement parts of the FSK encoder, and then explore techniques for FIR filter design of high quality filters based on new methods that include using a MATLAB GUI.

3.1 Generate an FSK Signal

One part of the FSK encoder must take a bit stream and create a sinusoid for each bit. The duration of each sinusoid will determine the bit rate, e.g., 300 bits/second (bps) requires a duration of 1/300 sec. In this part, you must complete the function below and use it to generate the FSK signal for the preamble (see Section 2.3). Use a duration of 50 millisec, even though that is only 20 bps, because then it will be easy to identify the tones on a spectrogram.

```
function  xx = fsk_gen( bitstr, fs, bdur )
%FSK_GEN  generate the FSK sinusoids at 1650 and 1850 Hz
%         1650 encodes a "1", 1850 encodes a "0"
%
%    bitstr = STRING of zeros and ones.
%        fs = sampling rate
%     bdur = duration of the sinusoid for one bit
%                  (this determine the "bit rate")
%        xx = synthesized FSK signal
%
inbits = abs(bitstr) - abs('0');   %- convert bit string to numbers
%%
%%%%%%%%%%  put your code here %%%%%%%%%%%%%%%%%%
```

Use a sampling rate of 8000 samples/sec which corresponds to the sampling rate used in POTS (the plain old telephone system). Make a spectrogram of the generated FSK signal in order to show how the individual bits correspond to sinusoids. Use a window length that is the default of 256, or shorter (perhaps 128 or 64 will give a better result).

Instructor Verification (separate page)

3.2 Filter Design

When you implement the FSK decoder, it will be necessary to have lowpass and (possibly) bandpass filters. We had previously encountered a similar filter design problem in the DTMF system, where several bandpass

filters were needed. In that case, we used FIR filters that had acceptable passbands, but were actually quite poor in their stopband behavior. The objective of this part is to investigate two ways to get better filters.[2]

1. Use the `filtdemo` GUI in MATLAB's Signal Processing Toolbox (see Section 3.2.1 below).

2. Perform the FIR filter design with a "window." One easy method of filter design is based on using specially designed functions called windows, such as the Hamming window described in Section 3.2.3.

(a) Design an FIR lowpass filter with the following specifications: $f_{\rm samp} = 8000$, $f_p = 1650$ Hz, $f_s = 1850$ Hz, stopband ripples less than -40 dB, and passband ripples less than 0.1 dB. If the use of dB is confusing, these specs can be restated as having stopband ripples less than 0.01 and passband ripples that lie between 1 ± 0.011. These are very tight specs, but they will guarantee that the response in the pass and stop bands is nearly ideal. Use either of the methods from Sections 3.2.1 and 3.2.3, but in the ECE lab (VL-252) it will be simpler if you use the GUI.

(b) Plot the frequency response magnitude using `freqz()` on the coefficients obtained in part (a).

Instructor Verification (separate page)

(c) Use the lowpass FIR filter to process the generated FSK signal from Section 3.1. Make a plot of the output signal to verify that all the 1850 Hz tones are removed.

Instructor Verification (separate page)

3.2.1 GUI for Filter Design

The process of filter design involves a trade off between the filter length L, the bandedges, and the stopband and passband ripples. If we use a GUI we can manipulate these trade-offs directly on the screen and see immediately how the magnitude response changes as we try to satisfy the desired filter specs. Figure 1 shows the magnitude response of a digital FIR filter obtained with the MATLAB GUI called `filtdemo`.

> *Decibels:* One feature of the magnitude plot in Fig. 1 is that the vertical axis is *logarithmic* in units called decibels (dB), i.e., the quantity plotted is $20 \log_{10} |H(e^{j\hat{\omega}})|$. When using a dB-scale, the passband, which ideally has a magnitude of one, should be 0 dB. In the stopband, we might have a value likes $|H(e^{j\hat{\omega}})| = 0.01$ which is -40 dB.

If we consider the case of a lowpass filter (LPF) such as Fig. 1, then we can describe the passband as extending from $f = 0$ to a cutoff frequency, $f = f_p$ Hz; and the stopband is the region $f \geq f_s$ Hz. Even though the frequencies are given in hertz, the filter design GUI yields a digital filter. The sampling frequency $f_{\rm samp}$ controls the relationship between the given analog cutoff frequencies, f_p and f_s, and the filter's frequency response $H(e^{j\hat{\omega}})$ which is actually a function of $\hat{\omega}$.
Question: Determine the values of $\hat{\omega}$ for the passband and stopband edges in Fig. 1.

3.2.2 Using the GUI

The filter type is controlled by the drop-down menu in the upper right-hand corner of the GUI. There are several choices, but the only ones of interest for FIR design in this lab are `REMEZ` or `KAISER`. For given ripple specs, the `REMEZ` design will give a shorter filter, so it might be prefered, but either method is acceptable for this lab.

[2] The Signal Processing Toolbox in MATLAB provides numerous functions that will perform filter design, but often this toolbox is not available. It is installed and available in the ECE-2025 lab in VL-252.

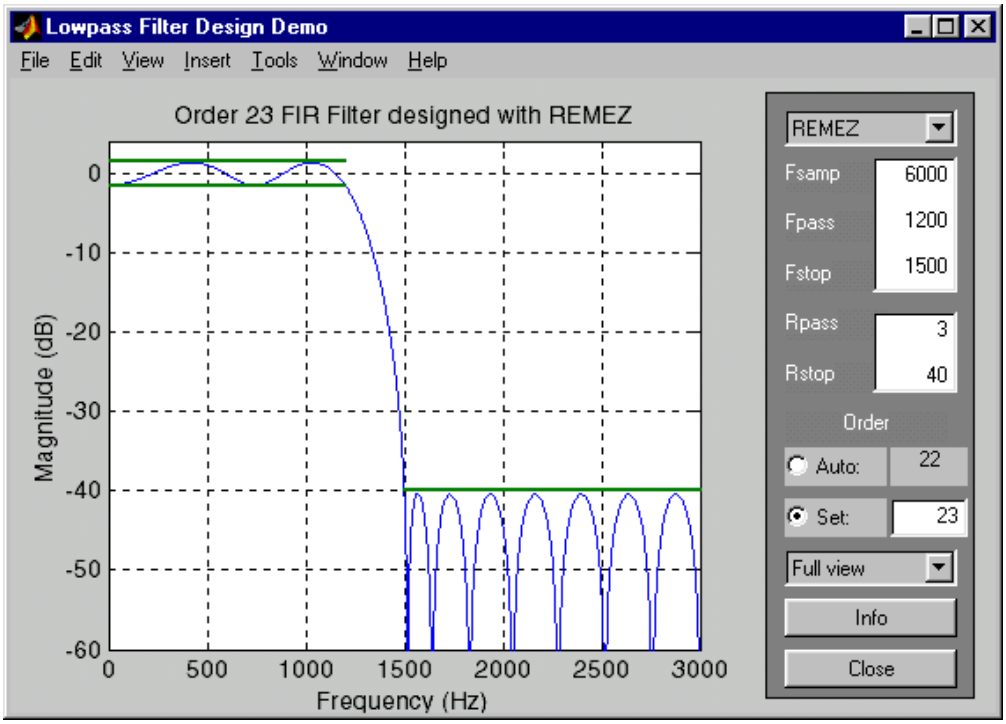

Figure 1: The `filtdemo` MATLAB GUI for digital filter design. The magnitude is given is dB (decibels). The frequency axis is labeled in hertz from DC to half the sampling frequency.

The filter specifications can be entered into the GUI in two ways: either in the text boxes on the upper right part of the window, or by dragging the green bars that indicate the desired passband and stopband regions. The bars can be moved up and down to change the desired ripple sizes, but the ends of the bars can also be moved horizontally to change the desired bandedges.

The filter order (which is $L-1$) can be calculated automatically by the GUI, or the user can enter a specific value. This option is controlled by the radio buttons on the right side of the GUI window. In Fig. 1, the value of $M = 23$ was entered by hand.

Obviously, the stopband cutoff frequency f_s must be greater than the passband cutoff frequency f_p. If we try to make f_s and f_p nearly equal, then the filter length L would have to be very large, so it is customary to allow a *transition region* between the passband and stopband.

Once the magnitude response has been manipulated into a desirable form, the filter coefficients can be extracted from the GUI by running the following in the command window.

```
[bb,aa] = filtdemo('getfilt');
```

Since this is an FIR filter, only the vector `bb` is needed. It will contain all of the filter coefficients for the FIR filter.

3.2.3 Filter Design via Windows

Previously, we have created LPFs by using the filter coefficients of a running averager, and BPFs with a cosine formula. It is possible to get better performance in the frequency response by modifying the filter coefficients to be tapered. One easy way is to use a Hamming window.[3] In this case, the impulse response

[3] A special M-file, called `hammfilt.m`, for filter design with the Hamming window is available for download from the WebCT page.

for a length-L lowpass filter would be given as:

$$h[n] = (0.54 - 0.46\cos(2\pi n/(L-1)))\,\frac{\sin(\hat{\omega}_n(n-\frac{1}{2}(L-1)))}{\pi(n-\frac{1}{2}(L-1))} \qquad \text{for } n = 0, 1, 2, \ldots L-1 \qquad (1)$$

where $\hat{\omega}_n$ is the nominal cutoff frequency of the lowpass filter. Usually $\hat{\omega}_n$ is chosen to be $\hat{\omega}_n = \frac{1}{2}(\hat{\omega}_p + \hat{\omega}_s)$ where $\hat{\omega}_p$ and $\hat{\omega}_s$ are the desired passband and stopband cutoff frequencies, respectively. The first term is the Hamming window; the second term is a "sinc" function which is used because it is the impulse response of an ideal lowpass filter.

The design process for the Hamming window filter involves finding the filter length L to meet the given design specs. In order to get the minimum value for L, it is customary to perform the following "trial and error" steps:

1. Initialize L as the integer closest to $2\pi/(\hat{\omega}_s - \hat{\omega}_p)$. The quantity $\Delta\hat{\omega} = (\hat{\omega}_s - \hat{\omega}_p)$ is called the *transition width* of the filter.

2. Using the present value of L, generate the filter coefficients via (1).

3. Compute the frequency response and plot the magnitude.

4. Compare the values of $|H(e^{j\hat{\omega}})|$ at the bandedges, $\hat{\omega}_p$ and $\hat{\omega}_s$. If the frequency response satisfies the specs for this value of L, but not for $L-1$, then the design is complete.

5. If the frequency response does *not* meet the specs, then increase the length L and return to step (ii); otherwise, decrease the length L and return to step (ii) to see if a smaller value will work.

This iterative "trial and error" can be accelerated if the changes in L are done by doubling and halving as in a binary search. For example, assume you start at $L = 50$, but the specs are not met; then try $L = 100$; if the specs are met, try $L = 75$ next; if that fails, then try $L = 87$, and so on.

The big advantage of the Hamming window design is that its stopband has ripples that are very small (usually less than 0.01). You can check this by zooming in to figure out the height of the ripples in the stopband whenever you plot the frequency response. The Hamming window design has two notable limitations.[4] First, the size of the ripples in the stopband and passband will be the same; second, the maximum ripple height is usually less than -53 dB, which is ± 0.0022. Therefore, all specs given in this lab are set up to conform to these constraints.

4 Lab Exercises

4.1 FSK Encoder

One of the simplest modems is FSK (V.21) which works by sending either a 1650 Hz tone (a 1 bit) or a 1850 Hz tone (a 0 bit) for 1/300 second (the bit rate or symbol interval). At the beginning of every symbol interval the cosine wave being synthesized can conceivably have its frequency changed from 1650 to 1850 Hz (or vice versa), the phase however should be smooth at the boundary to avoid a jump (also referred to as phase reversal).

It is possible to produce an FSK signal that is "phase smooth" by using the following trick. Write the synthesized signal as $x[n] = \cos(\psi[n])$ and concentrate on defining $\psi[n]$, the "argument" of the cosine. When you want the synthesized $x[n]$ to have a frequency of $\hat{\omega}_1$, the signal will be $x[n] = \cos(\hat{\omega}_1 n + \phi_1)$. In other words, $\psi[n] = \hat{\omega}_1 n + \phi_1$. If you then examine the first difference of $\psi[n]$, the result will be

[4] There are many filter design methods based on windows that do not suffer the same limitations.

$\psi[n] - \psi[n-1] = \hat{\omega}_1$. Since we know which frequency we want in each bit interval, we can define the values of the first difference of $\psi[n]$ for all n.

How do we get a smooth $\psi[n]$ from its first difference? Answer: use the following identity:

$$\psi[n] = u[n] * ((\delta[n] - \delta[n-1]) * \psi[n]) = u[n] * (\psi[n] - \psi[n-1])$$

where "*" denotes convolution. The convolution with the unit-step signal $u[n]$ gives a "running sum." Consult help on `cumsum` for a MATLAB function that will efficiently compute a running sum.

Write a complete FSK encoding system that will take a text message and produce the correct sequence of sinusoidal signals as would be generated in an FSK modem. The message has to be converted from text characters to a stream of bits. In addition, the standard preamble bits must be placed at the beginning of the bit stream, and 16 ones must be placed at the end to indicate termination of the message.

Give the MATLAB code for your encoding function. Make the bit rate one of the inputs to the encoder. Then encode the message "ECE-2025" at 50 bps, and then show part of the spectrogram (with a window length of 128) to illustrate that the encoder is functioning properly.

4.2 FIR Filter Design

In this section, you must demonstrate that it is possible to filter out one of the individual tones of the FSK signal with a bandpass filter (BPFs). It is easy to create a bandpass filter from a lowpass filter — all that is needed is to multiply the filter coefficients by a sinusoid whose frequency is the center frequency of the passband.

$$h_{\rm bp}[n] = 2h_{\rm lp}[n]\cos(\hat{\omega}_c(n - \tfrac{1}{2}(L-1))) \qquad \text{for } n = 0, 1, 2, \ldots, L-1$$

where L is the filter length. Thus you can use a LPF from the `filtdemo` GUI, or from a Hamming window design to create a high quality BPF. If the passband of the BPF extends from $\hat{\omega}_1$ to $\hat{\omega}_2$, then the LPF should have a cutoff frequency at $\frac{1}{2}(\hat{\omega}_2 - \hat{\omega}_1)$.

(a) Design a BPF that will pass the 1850 Hz FSK tone and reject the 1650 Hz tone. Make sure that you have at least -40 dB in the stopband to reject the 1650 Hz sinusoid. Try to minimize the filter length L. The filter's passband should extend from 1850 Hz to 2250 Hz, so as to allow for the widest possible transition band. State which type of filter you have designed: Hamming, Kaiser or Remez.

Show a plot of the frequency response magnitude in the region 1400 Hz to 2600 Hz to confirm that the filter satisfies the specs.

(b) Generate an FSK signal with a bit rate of 10 bps, and then process a one-second interval of the FSK signal to see if the filter works correctly. On a plot of the time signal show how the amplitude of the output changes for different bits.

(c) Now change the bit rate to 300 bps, and process a short section of the signal containing betwee 10 and 15 bits. Comment on how good or bad the BPF filter works at this bit rate. For example, on a plot of the processed output indicate where the 1850 Hz sections should be found. Compare the filter's length to the length of one bit interval. The length of the filter's transient (or start-up) region should be an important factor in your answer.

4.3 Comment

It is tempting to think that this sort of BPF implementation would make a useful FSK decoder. However, you should have discovered that the filter length L of this BPF is extremely long, meaning a hardware implementation would require an very large of multiply-adds for each output. A much simpler decoding implementation will be pursued in the next lab.

Lab #11
ECE-2025
Fall-2001
INSTRUCTOR VERIFICATION PAGE

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or professors might ask. Turn this page in at the end of your lab period.

Name: _____ Date of Lab: _____

Part 3.1 Complete the FSK signal generator. Show a spectrogram of the signal generated for the standard preamble.

Verified:_____ Date/Time:_____

Part 3.2(b) Design a lowpass FIR filter. Plot the frequency response magnitude to verify that it meets the given specifications.

Verified:_____ Date/Time:_____

Part 3.2(c) Use the lowpass FIR filter to filter the generated FSK signal. Make a plot of the output signal to verify that the 1850 Hz tones are removed.

Verified:_____ Date/Time:_____