ECE 445L Lab 5

Audio Processing System: Amplification, Filtering, Signal Processing

This laboratory assignment accompanies the book, [*Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154*](https://www.amazon.com/Embedded-Systems-Real-Time-Interfacing-Microcontrollers/dp/1463590156), by Jonathan W. Valvano, copyright © 2024.

# Table of Contents

[Table of Contents 1](#_Toc173668086)

[Team Size 2](#_Toc173668087)

[Goals 2](#_Toc173668088)

[Review 2](#_Toc173668089)

[Starter Files 2](#_Toc173668090)

[Required Hardware 2](#_Toc173668091)

[Abbreviated Requirements Document 3](#_Toc173668092)

[Lab Overview 3](#_Toc173668093)

[Preparation 6](#_Toc173668094)

[Procedure 7](#_Toc173668095)

[Deliverable 1 8](#_Toc173668096)

[Deliverable 2 8](#_Toc173668097)

[Deliverable 3 8](#_Toc173668098)

[Deliverable 4 8](#_Toc173668099)

[Deliverable 5 8](#_Toc173668100)

[Lab Checkout 9](#_Toc173668101)

[Lab Report 9](#_Toc173668102)

[Hints 10](#_Toc173668103)

# Team Size

The team size for this lab is **2**.

# Goals

* Study ADC conversion and the Nyquist Theorem.
* Characterize the performance of the DAC and ADC.
* Encode information as sound output from the DAC to speaker.
* Decode information as sound input from the microphone to ADC.
* Develop an audio communication system.

# Review

* Data sheets for your microcontroller.
* Data sheets for your hardware components.
* Valvano Section 6.2 on periodic interrupts using the timer.
* Valvano Sections 7.5 and 8.4 on DAC interface and waveform generation
* Valvano Section 8.5.4 on hardware averaging and timer-triggered ADC
* Valvano Section 10.2.5, Example 10.4

# Starter Files

* Starter project:
  + Lab 5 template provided on GH classroom repo.
* Example projects:
  + PeriodicTimer0AInts\_4C123
  + ADCT0ATrigger\_4C123
  + ST7735\_4C123
  + FFT16.xlsx, presented in Lecture aLec49d.pptx

# Required Hardware

|  |  |  |  |
| --- | --- | --- | --- |
| Parts | Datasheet | Price | Source (**price source)** |
| EK-TM4C123GXL | [EK-TM4C123GXL datasheet](https://github.com/ECE445L/ECE445L-Lab5/blob/main/resources/TM4C_Datasheet.pdf) | $16.99 | **TI** |
| 8Ω or 32Ω speaker | N/A | N/A | EER Checkout Desk |
| Resistors and capacitors | N/A | N/A | EER Checkout Desk |
| Switches | N/A | N/A | EER Checkout Desk |
| TLV5616CP 12-bit DAC  or TLV5618 | [TLV5616 datasheet](https://github.com/ECE445L/ECE445L-Lab5/blob/main/resources/part_datasheets/tlv5616.pdf) | $9.61 | EER Checkout Desk  Or **Mouser**, Digikey |
| LM4041CILPR shunt diode  or LM4041C12 | [LM4041C datasheet](https://github.com/ECE445L/ECE445L-Lab5/blob/main/resources/part_datasheets/lm4041c.pdf) | $0.78 | EER Checkout Desk  Or **Mouser**, Digikey |
| TPA731D audio amp | [TPA731 datasheet](https://github.com/ECE445L/ECE445L-Lab5/blob/main/resources/part_datasheets/tpa731.pdf) | $2.54 | EER Checkout Desk  Or **Mouser**, Digikey |
| MC34119 (discontinued) | [MC34119 datasheet](https://github.com/ECE445L/ECE445L-Lab5/blob/main/resources/part_datasheets/mc34119.pdf) | N/A | EER Checkout Desk |
| Electret Microphone | [CMA-4544PF-W datasheet](https://cdn-shop.adafruit.com/datasheets/CMA-4544PF-W.pdf) | $1.50 | EER Checkout Desk  Or **Mouser**, Digikey |

# Abbreviated Requirements Document

**2. Function Description**

**2.1. Functionality: What will the system do precisely?**

The encoding system will have two switches, labeled SW1 and SW2. The minimum requirement is two different symbols. Holding down SW1 sends symbol1 over and over, and holding down SW2 sends symbol2 over and over. The encoding system is isolated from the decoding system by an air gap, and the physical channel is sound from the speaker in the encoding system to the microphone in the decoding system. The encoding system will interpret the sound and activate two LEDs, labeled LED1 and LED2, according to which symbol was sent. If neither switch is pressed, there should be no sound and both LEDs should be off.

What happens when both switches are pressed is up to you. I.e., you may ignore, send symbol1, send symbol2, or send a third symbol (turning on both LEDs).

# Lab Overview

Lab 5 introduces the students to the field of audio signal processing. According to Wikipedia:

“Audio signal processing is a subfield of signal processing that is concerned with the electronic manipulation of audio signals. Audio signals are electronic representations of sound waves—longitudinal waves which travel through air, consisting of compressions and rarefactions. The energy contained in audio signals is typically measured in decibels. As audio signals may be represented in either digital or analog format, processing may occur in either domain. Analog processors operate directly on the electrical signal, while digital processors operate mathematically on its digital representation.”

Audio signals discernable to the human ear are in the range of 200 Hz to 16000 Hz: they include speech and musical instruments. In this lab, we will be processing the signals in the digital domain which means we need to convert them from the analog domain to the digital domain. In Lecture 39 we covered the Nyquist Theorem which specifies that to reproduce an analog signal from a digital signal we need to use greater than a 2X sampling frequency during the analog to digital conversion process.

Figure 5.1 shows the data-flow graph for the audio communication system that you will be building. The human operator provides input data to the system. The input information is first encoded as digital waveforms. The waveforms are output to the DAC/amp/speaker to create sounds. The sounds are transmitted across the air as pressure waves. The electret/amp/ADC converts the sounds into a sequence of data. The decoder converts the data back to the information (hopefully). Lastly, the output displays that information to the human operator. Feel free to change how the pieces fit together if the data streams and there are no wasted software delays.

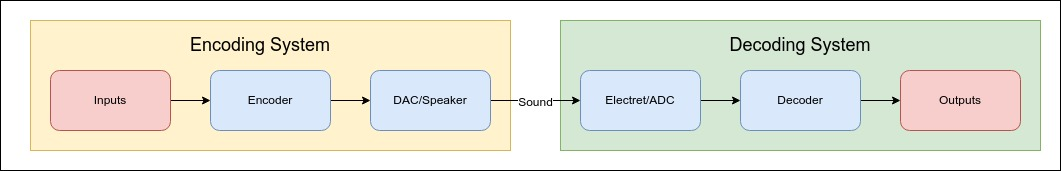


Figure 5.1. Data-Flow Graph of the Audio Communication System.

1. **Input**: The input to the system can arrive at whatever source you wish, if the human operator controls the values in some fashion. You can input characters from the PC keyboard and input them into the microcontroller using UART. You could input data from switches. The input task must run in the background using interrupts. Think about the appropriate priority level for this task.
2. **Input → Encoder**: The linkage between these modules must include a data structure that supports streaming, such as a FIFO or double buffer. A level of abstraction must be created by designing a set (two or more) of public functions the input module can call to affect communication.
3. **Encoder**: Each communication effort will involve a message, which must include data. Optional specifications include synchronization, and error checking. You are free to create whatever encoding mechanism you wish. However, consider encoding each n-bit symbol as a specific frequency output for a fixed time (frequency modulation). One of the challenges will be synchronizing the receiver to the transmitter. Even if the receiver knows you are sending a symbol encoding 2 bits of data every 10 ms, how does the receiver know where symbol ends and the next symbol starts? Consider a protocol where subsequent symbols always switch frequencies, even if the data were to remain constant. In Lab 5, recall that you set up a free running sine wave routine that can generate a wide range of sounds from 300 Hz to 1000 Hz. You will use that capability to encode data using sine waves (300 Hz and 1000 Hz). There are no requirements to maximize bandwidth (information/sec), minimize latency (time from input to output), or eliminate errors (sound noise causing bits to flip). However, you will be asked to quantify each of these performance measures and discuss in your report how each could have been improved. The encoder task also runs in the background using interrupts. Think about the appropriate priority level for this task. Remember there is a streaming data structure, so this interrupt task is different from the input task.
4. **Encoder → DAC**: The encoded message is sent to the DAC module in an appropriate manner that you will design. For example, it might be an array of frequencies. The linkage between these modules also must include a data structure that supports streaming. However, you may wish to place all transmission tasks (encoder and DAC) into the same software file.
5. **DAC**: Please use as much of your Lab4 as possible because the DAC does output to the speaker to send data across the communication channel. Since this interrupt will be high frequency, make sure it is very simple. Think about the appropriate priority level for this task.
6. **DAC → ADC**: There must be a software disconnect between the transmitter tasks (input, encoder, and DAC) from the receiver tasks (ADC, decode, display). There can be no shared data or function calls between them. The only linkage should be sound traveling as air pressure.
7. **ADC**: The audio input is filtered and then sampled by an ADC. The sampling rate is controlled by a periodic timer that triggers the ADC, and the ADC ISR executes when the conversion is complete. Because the timer starts the ADC, there is no sampling jitter. Think about the appropriate priority level for this task. How does the timer-triggered sampling affect the selection of priority for this ISR? Since this interrupt will also be high frequency, make sure it is very simple. However, you should implement digital filtering here to improve SNR. See the **NoiseReject** functions provided in the **/inc/DSP.c** file.
8. **ADC → Decoder**: The linkage between these modules also must include a data structure that supports streaming.
9. **Decoder**: You can run the decoder in the while-loop of the main. The goal is to extract the data from the sampled sound. The system is considered real time if the software is fast enough to keep up (no data is lost). If the decoder cannot keep up, you can slow down the transmission rate or use a simpler decoding algorithm. Possibilities include but are not limited to cross correlation and FFT. There are three very fast integer FFT functions in the inc folder of the starter projects: cr4\_fft\_64\_stm32.s, cr4\_fft\_256\_stm32.s cr4\_fft\_1024\_stm32.s. However, there is set of a very DFT functions in the **/inc/DSP.c** file
10. **Decoder → Display**: The linkage between these modules can be a simple function call.
11. **Display**: This task is run from the main program. It outputs the data to the human operator. The output can use whatever device you wish, as long as the human operator obverse the values in some fashion. You could output characters on the LCD. You could output data to LEDs. During testing, when the input is created with data of known values, the output can check for lost or changed values.

The max sampling rate on the TM4C123 is 125K samples/sec which is way above the Nyquist value (and the Valvano Postulate) for audio input signals. You should select sampling rates that allow all tasks to be completed in real time.

All I/O must be written in a style like the book, without calling any TivaWare driver code. It is acceptable but not necessary to use two microcontrollers, one for transmitting and the other for receiving.

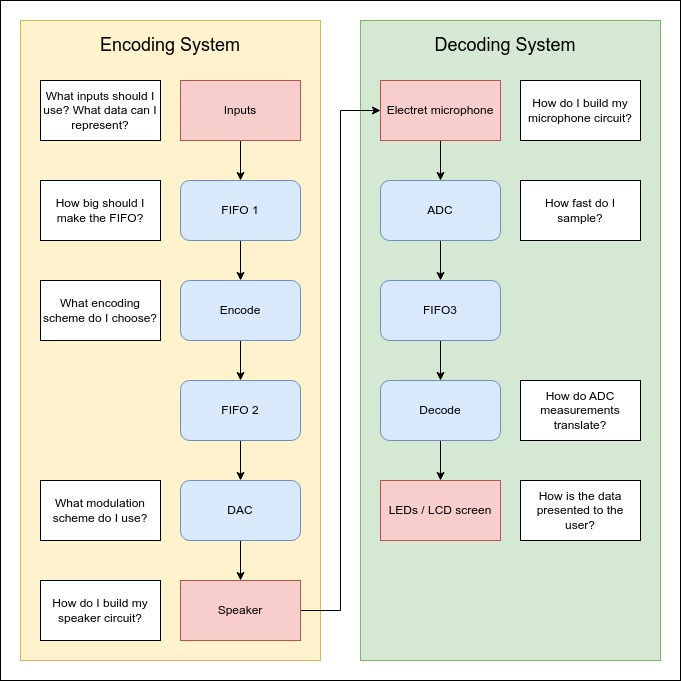


Figure 5.2. Possible Data-Flow Graph of the Communication System.

Figure 5.2 shows a possible expanded dataflow graph for the audio communication system that you will be building. Feel free to design your own implementation.

# Preparation

1. **Choose an encoding/decoding scheme.** Make some rough sketches of the sound waves you plan to use to transmit data. You are free to change this as you build it and see what works and what does not work.
2. **Choose a message format.** The message must include data.You may optionally add support for error checking (like [parity bits](https://en.wikipedia.org/wiki/Parity_bit) or [checksum](https://en.wikipedia.org/wiki/Checksum)). You may optionally add support that will allow the receiver to synchronize to the transmitter. How will the receiver be able to separate one bit from another? *Hint, how does the asynchronous UART protocol separate bits? What is a baud rate? Be able to provide an explanation of your message format.*
3. **Distribute tasks.** Provide a list of tasks of software to write, hardware to build, and tests to perform, equally distributed between team members. Ideally, everyone is working on something in parallel.
4. **Hardware Design.** Collect all components you may use and provide a schematic of the encoding and decoding systems. You may use a headset microphone and an audio jack connector in place of an electret microphone. A circuit for the electret microphone is provided for you in Figure 5.4. You may use an alternative design on the internet if that works better for you. Please document the source.
5. **Software Design.** Design header files for software modules and data structures. Write unit tests, potentially for each block in your call graph, demonstrating that each component works in isolation. You may then want to write integration tests to check that interactions between components work as well. The final main will likely be the combination of your integration tests.

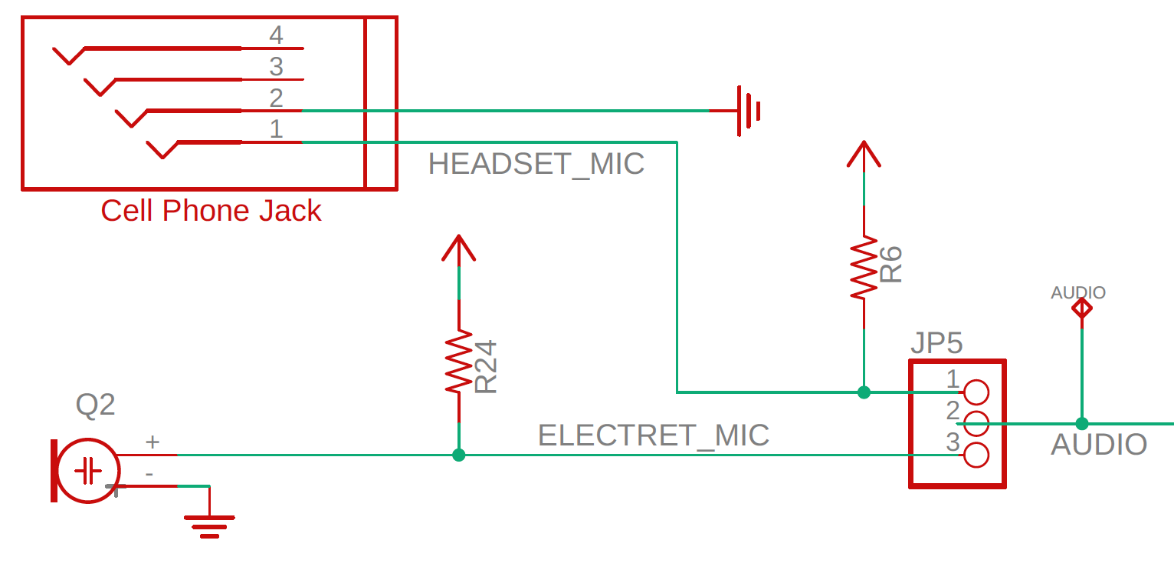


Figure 5.3. Audio Signal Input Circuit.

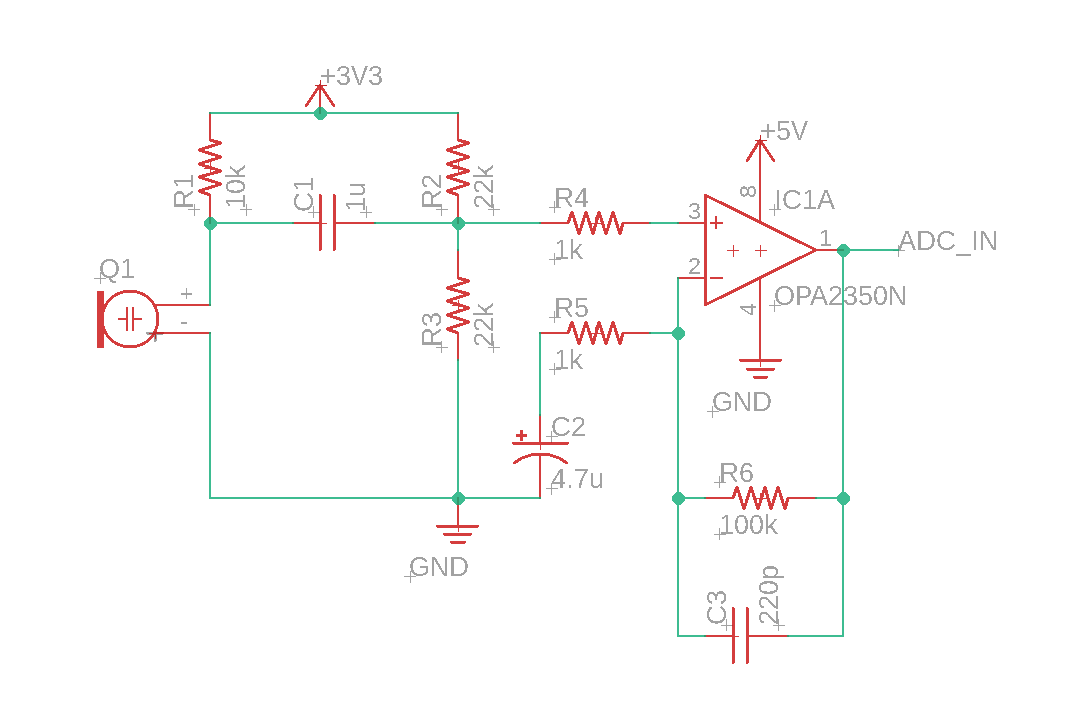


Figure 5.4. Possible Electret Circuit.

# Procedure

**Note: We assume that you have implemented and tested the modules separately. These are the test and performance validation procedures to be executed assuming that the system is implemented and works (relatively).**

For a single, fixed, arbitrary user input in a quiet room:

1. Encoder
   1. Tweak output parameters (sine wave table size, encoding scheme, symbol rate) of your encoder until the output waveform meets requirements of your designed message protocol.
   2. Dump the DAC output into a spreadsheet and visualize the quantized waveform.
   3. Provide the necessary context needed to understand the signal:
      1. What is the input message?
      2. What is the encoding scheme including baud rate?
      3. What is the DAC resolution (horizontal and vertical)?
2. Decoder
   1. Tweak the microphone circuit until the input waveform meets the requirements of your designed message protocol.
   2. Dump the ADC input into a spreadsheet and visualize the quantized waveform.
   3. Provide the necessary context needed to understand the signal:
      1. What is the distance between microphone and speaker?
      2. What is the gain and filtering of your microphone circuit?
   4. Measure the SNR of your input signal.
   5. Verify that the decoder can correctly interpret the input stream.
3. Profile the **encoder** and **decoder** system modules CPU utilization.
4. Quantify the system performance:
   1. **Baud rate of the system** – how much data can your system move?
   2. **Bandwidth of the system** – What range of frequencies does your system utilize?
   3. **Latency of the system** – how long does it take for a message to be sent and received from start to finish?
   4. **Reliability of the system** – sending the message N times, what is the rate of failure? N should be a statistically useful number.

## Deliverable 1

Create a schematic or figure showing all external components connected to the TM4C123 boards on KiCad. You do not need to show hardware components on the TM4C123 LaunchPad board. Your schematic should show your encoder and decoder circuits. The encoder should be like the circuit created in lab 4. The decoder circuit should include minimally, the TM4C, microphone, and the amplification circuit.

## Deliverable 2

Collect data on the TM4C and on the oscilloscope to show the performance of the encoder. Using some debugging technique, collect the quantized output the TM4C is attempting to output. Compare this output with the actual output of the TM4C. Did the DAC accurately create the signal? What Errors exist? Include the array of values you recorded and the scope trace.

## Deliverable 3

Collect data on the TM4C and on the oscilloscope to show the performance of the decoder. Using some debugging technique, collect the quantized input the TM4C is attempting parse. Compare the quantized input with the real analog signal produced by the microphone before any filtering. Compare this output with the actual output of the TM4C. What does adding a bandpass filter do to the output of the microphone? What Errors exist? Include the array of values you recorded and the scope trace.

## Deliverable 4

Use debugging techniques to determine the CPU utilization of your encoder and decoder. Specifically, determine how much time the encoder spends per second to output the encoded waveform. Additionally, determine how much time is spent by the decoder TM4C in both sampling and decoding of the input data.

Show any traces or debugging output used to calculate these values.

## Deliverable 5

Take measurements of your system performance to quantify its performance. Measure the total power consumption of the two TM4Cs. Measure the amount of time it takes to transmit one bit. Measure the distance at which the two TM4Cs can communicate. Play sound off your phone or computer while to see how noise tolerant your system is. You should record the following information in your report:

Maximum bits per second: (Total message transmission time)/(Data bits sent)

Energy per bit: (Supply voltage \* Current) / (Maximum bits per second)

Range: (Distance between TM4Cs)

Noise tolerance: Approximate noise in DB that the system can tolerate while working.

# Lab Checkout

The lab checkout is performed during the M/T lab session.

1. Show that the system works with an arbitrary fixed input, with DAC scope output and ADC quantized input plot (see *Procedure*).
2. Demonstrate that the system works with any user input.
3. Demonstrate that the system is robust (or best attempt robust) under noisy environments.
4. Explain the encoding/decoding scheme.
5. Explain any synchronization or robustness schemes implemented.

# Lab Report

The lab report shall be submitted by the Friday after the second lab section.

You should complete the Lab05Report.docx file with your data and answers then submit the completed file to Canvas.

# Hints

The figures in this section implement 8-bit data, parity error checking, and start bit synchronization. The minimum requirement for this lab is to send two distinct symbols, without error checking or synchronization.

The following figure is the analog signal at the DAC output showing a piece of an FM-modulated communication. This was measured running just the transmitter with TExaS initialized with the SCOPE\_PD2 setting. The higher frequency encodes a zero bit, and the lower frequency encodes a one bit. This symbol time is 10.24ms.

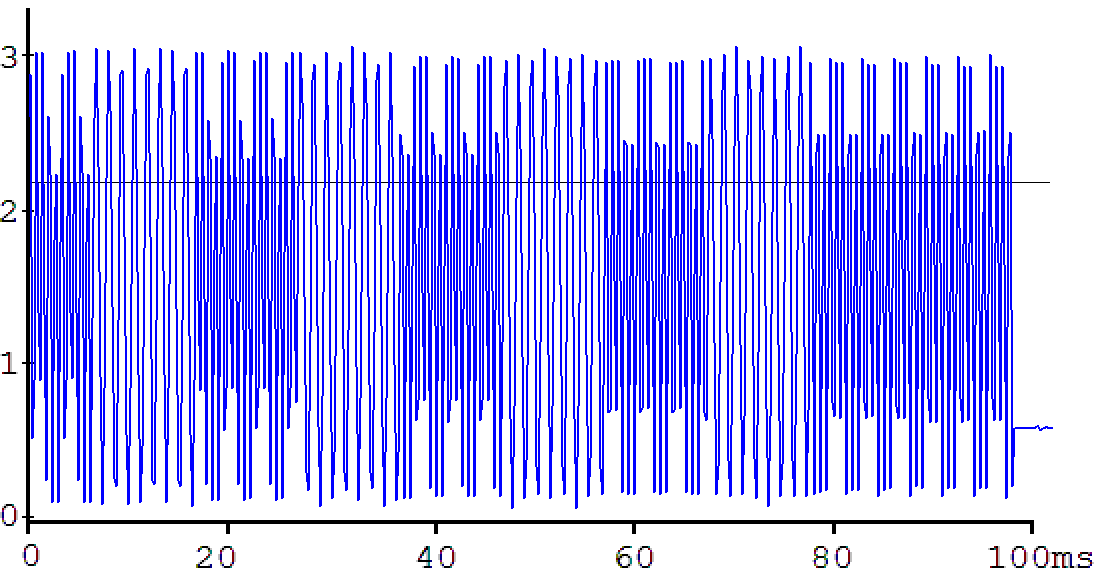


Figure 5.5. FM-modulated encoding of Audio Signal.

The following is a similar waveform of DAC out measured on a real scope. This format is start bit (0), 8 data bits (10101010), and even parity (0).

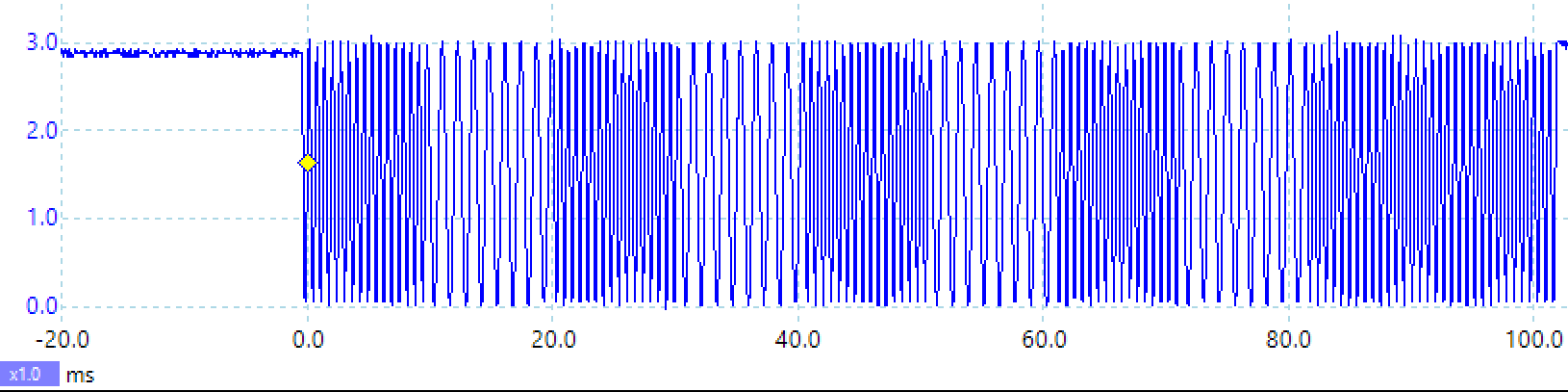


Figure 5.6. FM-modulated encoding of Audio Signal, one frame.

The following figure is the analog signal at the input of the receiver ADC showing a piece of FM-modulated communication. This was measured running just the transmitter with TExaS initialized with the SCOPE\_PE2 setting. The higher frequency encodes a zero bit, and the lower frequency encodes a one bit. The symbol time is 10.24ms.

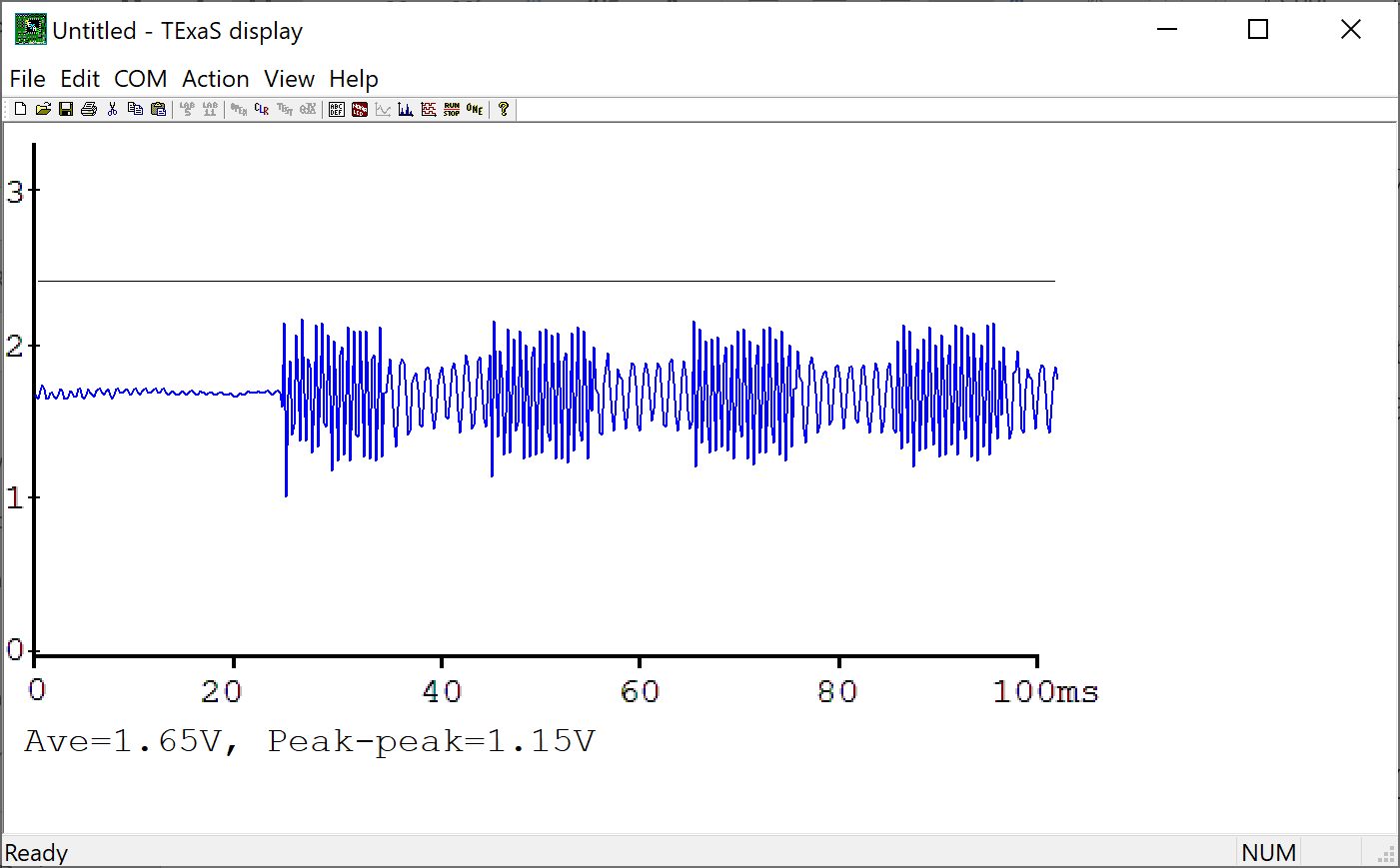


Figure 5.7. FM-modulated decoding of Audio Signal, some bits.

The following figure is the analog signal at the input of the receiver ADC showing the entire FM-modulated message, measured with a real scope. The time to transmit one message is 102.4ms.

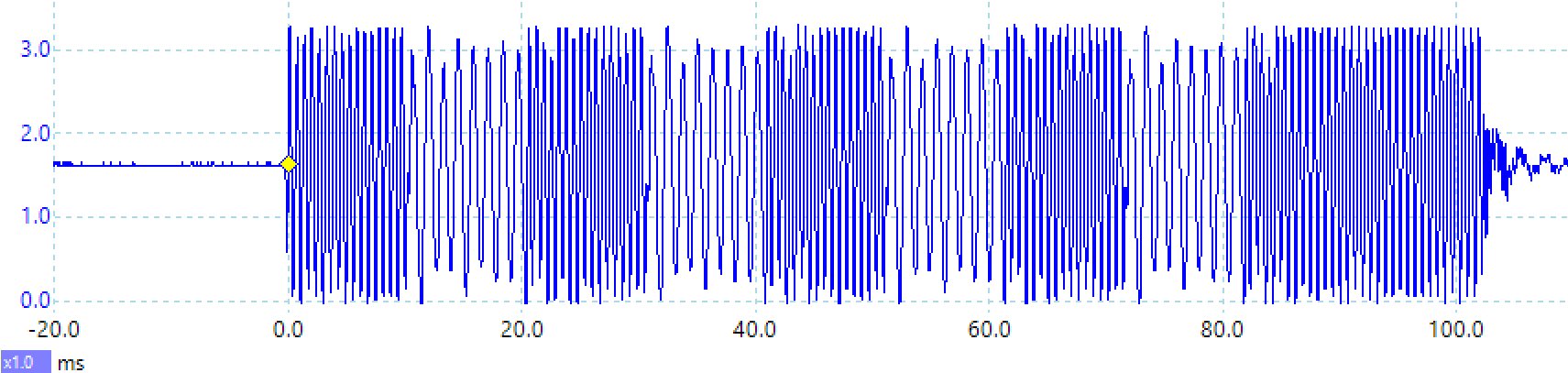


Figure 5.8. FM-modulated decoding of Audio Signal, one frame.

Let be the receiver sampling rate. DFT length will be N=16

Let be one frequency, which will be equal to

Let be the second frequency, which will be equal to

See the functions in **dsp.c dsp.h** in the **inc** folder

Usage for decoding FM modulation in receiver

- Call **DFT\_Init()** once

- at in real time using periodic interrupt

1) x=sample microphone using ADC

2) call **DFT(i,x)** once per interrupt, with index i going from 0 to 15

3) after 16 samples

a) call **Mag1()** for relative amplitude at

b) call **Mag2()** for relative amplitude at

The **Discrete Fourier Transform** (DFT) converts data in the time domain to data in the frequency domain. We can use the DFT to measure SNR, to identify noise type, and to design FIR digital filters. In fact, the spectrum analyzer is simply a high-speed data acquisition system followed by a DFT. The Fast Fourier Transform (FFT) is a technique to calculate the DFT with fewer additions and multiplications. There are four important parameters when employing the DFT. The first parameter is sampling rate, . While the DFT deals only with samples and bins with no concept of volts, seconds, and Hz, when applying it to real data, we assume the samples have units, are bound by physical limits, and are evenly spaced at time intervals . The second parameter is sequence length, N. The other two parameters are input resolution and range. In real systems, input data comes from the ADC or input capture, and the output data goes to the DAC or PWM. Therefore, the performance of the DFT will be affected by the range and resolution of the input. The input to the DFT will be N samples versus time, and the output will be N points in the frequency domain.

Input: {} = {}

Output: {} = {}

The definition of the DFT is

where and k=0,1,2,…,N-1

The DFT output Ak at index k represents the amplitude and phase of the input at frequency k\*/N (in Hz). The DFT resolution in Hz/bin is the reciprocal of the total time spent gathering time samples; i.e., 1/(N\*T).