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 © 2019.

Goals • Study ADC conversion and the Nyquist Theorem

• Characterize the performance of the DAC and ADC

• Encode information as sound output from DAC to speaker

• Decode information as sound input from microphone to ADC

• Develop an audio communication system

Review• Operation of the ADC system in the TM4C123GH6PM data sheet

• Valvano Section 6.2 on periodic interrupts using the timer

• Valvano Section 7.5 on SSI interfacing

• Valvano Section 8.4 on DAC parameters and waveform generation

Starter files • **PeriodicTimer0AInts\_4C123**

• **ADCT0ATrigger\_4C123**

• **ST7735\_4C123** LCD driver used in Labs 1-5

• Lab 5 software

## Team Size: 4

## Required Hardware (can build with parts or use Lab board)

EK-TM4C123GXL [http://www.ti.com](http://www.ti.com/tool/ek-tm4c123gxl?keyMatch=tm4c123g&tisearch=Search-EN-Everything)  $12.99

Sitronix ST7735 Color LCD <http://www.adafruit.com/products/358> $19.99

Speaker Checkout counter

Cell phone Headset (3.5mm) Use your own cellphone microphone

Electret Microphone Checkout counter

LM4041CILPR shunt diode Checkout counter

OP2350PA op amp Checkout counter

TLV5616 12-bit DAC From Lab 5

TPA731 or MC34119, Audio amp From Lab 5

Various resistors and capacitors Checkout counter

## Background Information

This lab introduces the student to the field of audio signal processing. According to [Wikipedia](https://en.wikipedia.org/wiki/Audio_signal_processing):

“Audio signal processing is a subfield of [signal processing](https://en.wikipedia.org/wiki/Signal_processing) that is concerned with the electronic manipulation of [audio signals](https://en.wikipedia.org/wiki/Audio_signal). Audio signals are electronic representations of [sound waves](https://en.wikipedia.org/wiki/Sound_wave)—[longitudinal waves](https://en.wikipedia.org/wiki/Longitudinal_wave) which travel through air, consisting of compressions and rarefactions. The energy contained in audio signals is typically measured in [decibels](https://en.wikipedia.org/wiki/Decibel). As audio signals may be represented in either [digital](https://en.wikipedia.org/wiki/Digital_signal_(signal_processing)) or [analog](https://en.wikipedia.org/wiki/Analog_signal) format, processing may occur in either domain. Analog processors operate directly on the electrical signal, while digital processors operate mathematically on its digital representation.”

These signals are in the range of 200 Hz to 16000 Hz and include speech and musical instruments. 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 in order 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. What is the Valvano Postulate?

## Communication System

Figure 9.1 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 transmit 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.



Figure 9.1. Data-flow graph of the audio communication system

**Input:** The input to the system can arrive whatever source you wish, as long as the human operator controls the values in some fashion. You could input characters from the PC keyboard and input them into the microcontroller using UART0. You could input data from switches. During testing, input will also be created with data of known values (to check for lost or changed values). The input task must run in the background using interrupts. Think about the appropriate priority level for this task.

**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) public functions the input module can call to affect communication.

**Encoder:** Each communication effort will involve a message, which includes data, 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 synchronize 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 particular 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.

**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.

**DAC:** Please use as much of your Lab-5 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.

------------------------------------------------------------------------------------------------------

**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. The first part of the main program should initialize all modules, data structures, and I/O devices. Once interrupts are enabled, the while-loop in the main will be restricted to just decoder and output tasks.

------------------------------------------------------------------------------------------------------

**ADC:** The audio input is filtered and then sampled by ADC0. 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 too is very simple. However, you should implement digital filtering here to improve SNR. A fast nonlinear digital filter to remove noise was presented in slides 7-9 in aLec49b\_SoundInput.pptx.

**ADC → Decoder:** The linkage between these modules also must include a data structure that supports streaming.

**Decoder:** You can run the decoder in the while-loop of 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, i.e., no lost data. 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. Cross correlation was presented in slides 6 and 11 in aLec49b\_SoundInput.pptx. FFT was presented in aLec49c\_FFT.pptx. 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.

**Decoder → Display:** The linkage between these modules can be a simple function call.

**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. You should not use UART0 to output if you used UART0 to input, because that couples the transmitter to the receiver. 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 complete in real time.

All I/O must be written in a style like the book, without calling any TivaWare driver code.

Figure 9.2 shows a possible call graph for the audio communication system that you will be building.



Figure 9.2. A possible call graph of the audio communication system. You are free to choose the input device (shown here as switches), the three streaming data structures (shown here as FIFOs) and the output device (shown here as LEDs). The initial part of main will initialize all modules.

One of the requirements for Lab 9 will be the equal distribution of tasks onto the various team members. You are allowed to use the Lab board. If you do use the Lab board, some of the hardware tasks are simple and hence do not contribute much to the distribution of effort. Basically we will expect a more sophisticated protocol and higher performance if you use the lab board. If you build the hardware from scratch, we will just expect it to work, sort of. Here is a partial list of tasks that you can divide amongst the team members. You can add other tasks as necessary to complete the system

**0) System design and integration:** Design of the encoding and decoding algorithm. This task will involve tweaking the size of the sinewave tables used by the DAC (changing the output frequency), the ADC sampling rate, and the overall system bandwidth so that everything is real time (all tasks complete on time).

**1) Input**: interface hardware, low-level software drivers, and a *main* program to test the input task.

**2)** **Input → Encoder:** streaming data structure, header file, code file, and a *main* program to test the streaming.

**3) Encoder:** Software to implement encoding. Initialization function, ISR, calls to 2 and 4. Implement a separate *main* program to test the encoder.

**4)** **Encoder → DAC:** streaming data structure, header file, code file, and a *main* program to test the streaming.

**5) DAC hardware:** Interface the DAC, amplifier and speaker. Essentially, this task is rebuilds and retests Lab 5. Implement a separate ISR and *main* program to test the DAC hardware. Use the TExaSdisplay spectrum analyzer to evaluate the signal to noise ratio of the DAC output of a single tone. Remember TExaSdisplay uses a TM4C123 ADC input so voltages must be between 0 and 3.3V.

**6) DAC software:** Write the ISR that reads from the appropriate data structure and writes to the DAC.

**7) Transmitter testing:** Write a separate *main* program to test steps 1 – 6. Create oscilloscope tracings that illustrate the behavior of the transmitter. Recall the TExaSdisplay scope samples at 10 kHz, so these tracings will be rough.

**8) Microphone amplifier hardware.** Design and build the sound input hardware. If you use the lab board, there are two options for sound input. You may use either option. Run your task 5 system and connect the TExaSdisplay spectrum analyzer to the ADC input and compare SNR at the ADC with SNR at the DAC.

**9) ADC software:** Write the ISR that reads from the ADC and writes to the appropriate data structure. Implement a separate main program to test the ADC input software. This task will include a digital filter to improve SNR. Implement a separate *main* program to test the ADC input software. Send typical data to the PC for plotting, and show the signal with and without the digital filter.

**10)** **ADC → Decoder:** streaming data structure, header file, code file, and a *main* program to test the streaming.

**11) Decoder:** Software to implement decoding. Initialization function, main loop, calls to 10 to collect data. This task will involve signal processing like cross correlation or FFT. Implement a separate *main* program to test the encoder. Use the collected data in task 9 to test the decoder.

**12) Display**: interface hardware, low-level software drivers, and a *main* program to test the display task.

**13) System performance measurements:** Final system performance testing (bandwidth, latency, reliability). Use a real logic analyzer to profile the percentage time required for each task. This profile will identify the bottlenecks.

## Preparation (do this before your lab period)

### 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.

### Choose a message format

Add something to the protocol to support error checking (like checksum). Add something to the protocol 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?

### Distribute tasks

Edit the above 14 tasks as needed and assign equal effort to each team member.

### Hardware Design

For the preparation you have to collect all hardware components and draw all circuit diagrams. If you are using the lab board, this step simply is to design which two audio input source to use: 1) Electret Microphone and 2) Cellphone headset as shown below in Figure 9.3.

![A close up of a map

Description automatically generated]()

Figure 9.3 Audio Signal Input circuit

### Software Design

For the preparation, design all software data structures, modules, and testing procedures. More specifically, write all header files and structs. Normally, preparation requires writing all software until it compiles. However for this preparation you need to complete all header files and test mains. There are ten test mains listed in the tasks. The code files can include empty functions (to be completed as part of the procedure), but the ten test mains should all compile without error.

## Procedure (do this during your lab period)

1. Each of the modules should be implemented and tested separately.
2. Test just the transmitter. Include tracings as described in Tasks 5 and 7. Tweak output parameters (sine wave table size, encoding scheme, time per bit) until the waveforms look appropriate on the scope connected to the DAC output.
3. Test the sound input. Include tracings as described in Task 8.
4. Test the ADC software (task 9). Plot waveforms on the PC and collect waveform samples to be used in the next step.
5. Test the decoder using collected waveform samples. Tweak system parameters to make it work most of the time when the room is quiet.
6. Modify the input module to create a known data stream at a known rate. Test the decoder using live data in real time. Tweak system parameters to make it work most of the time when the room is quiet.
7. With the known input data and input rate, profile the operation of the complete system using a real logic analyzer. Quantify system performance: 1) bandwidth (information bits/sec), 2) latency (time from input to display), and 3) reliability (percentage of bits properly communicated).
8. Repeat the performance measurements in presence of normal background sounds. For example, speak the numbers 1 to 10, in a normal voice, within five feet of the microphone. For example, sing a song, in a normal voice, within five feet of the microphone.
9. Final system test with human input.

## Deliverables (exact components of the lab report)

1. Objectives (1/2 page maximum)
2. Hardware Design (skip this for Lab board)
   1. Insert circuit diagrams input, DAC, speaker, microphone, and output
3. Software Design (a softcopy software printout is due at the time of demonstration)
   1. All low level modules (code and header files)
   2. All ten test mains
   3. Data flow graph and call graph if different from Figures 9.1 9.2
   4. Main program used to implement the communication
4. Performance Data
   1. DAC spectrum of a single tone (task 5); calculate SNR
   2. DAC scope of an encoded message (task 7)
   3. ADC input spectrum of a single tone (task 8) ; calculate SNR; compare to task 5
   4. Typical sampled ADC data of a message with and without filtering (task 9)
   5. Logic analyzer profile showing relative percentage time to execute all modules
   6. Bandwidth, latency, reliability (task 13) with and without background sounds
5. Analysis and Discussion (give short answers to these questions)
   1. What is the Nyquist theorem and how does it apply to this lab?
   2. How did you eliminate noise in the sampled audio?
   3. How does your protocol allow (or doesn’t allow) communication in the presence of background speech and singing?
   4. How could bandwidth have been improved?

## Checkout (show this to the TA)

1. Demonstrate the system with known input and known input rate, procedure 7. Show the scope signals on the DAC output and ADC input.
2. Demonstrate the system with user input, with quiet room and with background talking
3. Answer questions about sampling, ADCs, DACs, and analog filters.
4. Answer questions about encoding and decoding algorithms you used. In particular, how does your receiver synchronize to the transmitter?