Project Report
Sonic Location System

Mark Stacey
Matthew Dixon
Jason Brinton

1.1

Introduction

Sound is a reliable resource that can be used in many applications such as music, visualizing tissue, medical procedures and sonar. The goal of our project was to use sound data and process it in a way so as to find the location of a sound's source.

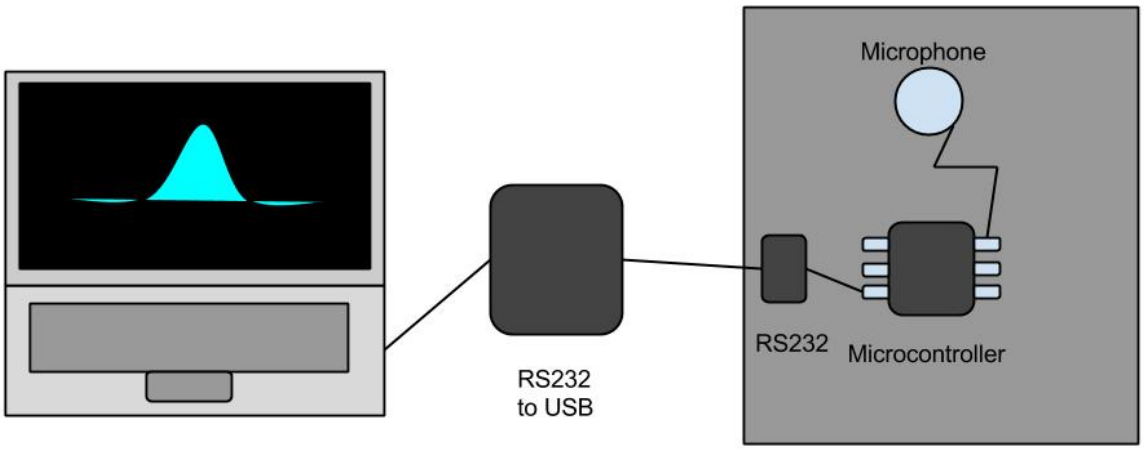
In this project, we show how a PC can control a microcontroller to capture sound via microphones. The sound data would then be transferred over UART and processed on a PC. The data was processed to show which where the sound is strongest in reference to the boards.

Originally our design was to use 3 of these boards connected together as nodes to record sound. Then using a computer, compute the data together in order to use them as a sound locator. Using a cross correlation method we would take the difference in magnitude of the sound level between the 3 nodes and determine where a sound is coming from on a plane. However, because of time and resource constraints, and a broken microphone we simplified the project to just one board and used it as a sound level graph.

1.2

High Level Design

We have a board with a microphone that communicates with a computer over UART.



We gather the sound data by triggering the microphone with the I2S protocol. We trigger the WS line for each byte of data that the microphone picks up. Then the data is processed on the microcontroller. Then we send it to the RS232 chip. The RS232 chip sends to the RS232 to USB converter which sends the data to the computer. At the computer, we process the data and display it as a graph using the matplotlib python library.

1.3

Member Task Distribution

Matt was in charge of the GUI, which was programmed in python. Jason was in charge of the computer side of communicating with the board over UART. Mark did the microcontroller programming, which consisted of I2S and internal commands. Also plotting the data on the computer.

1.4

Hardware Design

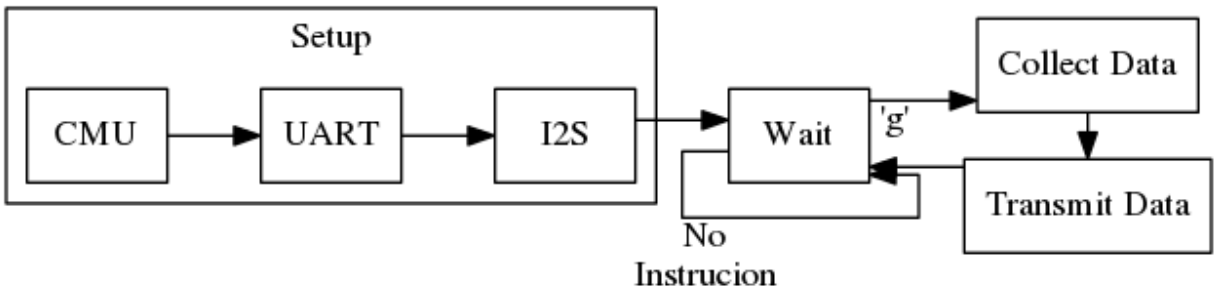
To implement our project we decided to use a prototype printed circuit board (PCB) design from Engimusing LLC. The board comprises a microcontroller (MCU), dual RS232 ports, a 24-bit MEMS microphone with built in analog to digital Converter (ADC), and 5V to 3.3V linear regulator. The MCU used in the design was the EFM32TG110F32 from silicon labs. It includes the ARM cortex-M3 microcontroller, a clock management unit (CMU), and multiple peripherals such as, timers, general purpose I/O (GPIO) pins, and multiple serial communication modules. This provided a good foundation to begin our project.

The data is collected by the INMP441 MEMS microphone. The microphone has a built in 24-bit ADC which we can communicate over the inter integrated sound (I2S) protocol. The e protocol allowed us to control the sampling frequency of the microphone by controlling the serial clock. Controlling the sampling frequency is essential for data analysis on the computer.

Two of the MCU serial communications modules are connected to a dual RS232 transceiver. This allows the PCB to have two RS232 communications ports. With two RS232 ports, we would be able to connect multiple nodes together to form a sensor network. One of the RS232 ports is connected to an RS232-to-USB conversion board, so that we can transmit instructions to the nodes and data back from the nodes to the computer. With this setup we have to capacity to set up a network of MCU controlled microphones and transmit the collected data back to a computer for data processing and display.

1.5

Software Design



The firmware on the microcontroller (MCU) was designed to facilitate the sampling sound from the microphone while giving control over when to sample to a host computer. Giving control to the host computer allowed us to simplify the design of the firmware and make the functionality modular. That way the system could be expanded to include multiple nodes. Collecting sound from multiple nodes is essential for determining the origin of sound impulses.

The software has the functionality to communicate with the microphone using the I2S protocol. This protocol was implemented using one of the serial communications modules on the MCU. In I2S data transmission is controlled by sending a continuous serial clock to the microphone. The rate of the serial clock determines the sampling frequency of the microphone in addition to the data transfer rate. The microphone was designed to collect stereo sound, so additional control signals were required. One control line is used to weather the microphone is the left or right microphone. A word-select (WS) line is used to control when data is transferred from the left and right microphones. Even, though the PCB only had one microphone, for correct operation it had to be controlled as if there were two microphones present. Data from the 24-bit ADC on the microphone was transferred in 4 bytes or 32-bits because of the I2S operation. The 32-bit data received from the microphone had to be shifted by 4-bits, after reception in the MCU, to represent the 24-bit value from the ADC.

Once the data collection functionality was completed we integrated it with the RS232 communications from MCU to the host computer. This communication worked simply by waiting for an instruction from the host computer. If a ‘g’ is received from the host computer, the MCU collects data from the microcontroller and subsequently transmits the data back to the host computer for processing. Once the transmission is completed the MCU then waits for another instruction from the host computer before collecting data again. This allowed the control and timing of the data collection to be controlled from the host computer.

Data is received on the host computer using a serial communications library in python. The serial communications are configured with a timer to collect the data every half second. We determined the controlling the data flow in this way makes the most sense because it gives the user of the host computer the ability to control the data flow and timing. Controlling the data flow may be important when determining the location of a sonic impulse. When the host computer receives the data from the MCU, The data is stored into an integer array. The software then calls the matplotlib python library to display the data using and animation class inside of matplotlib. The matplotlib library processes the data and displays a plot showing the audio frequency spectrum of the data from the microphone.

1.6

Results of the Design

One of the problems we ran into was the INMP441 microphone footprint was oriented incorrectly on the circuit board. To resolve this issue a hot air gun was used to desolder the microphone from the board, and then a hand soldering iron was used to connect the microphone to correct locations on board. This wouldn't normally be a problem, but as we got towards the end of the project and we were beginning to graph the output from the microphone we noticed that the waveform resembled random noise, and didn't change no matter what sounds we made around the microphone. This lead us to believe that the microphone was malfunctioning. To confirm this we performed and plotted the FFT of the signal, and we saw every frequency was present in the samples, confirming the microphone was detecting random noise.

We believe the microphone was damaged during the hand soldering process. It is a very small IC microphone, and the extreme heat of the soldering tip likely cooked the internal circuitry.

Another problem we encountered was the python serial reader function did not see a "0" if it was part of the message. this is because a "0" is interpreted as a null character. This caused a problem because the way we wrote the serial reader function required a certain number of bytes to be read into the read buffer before the data would be returned. After the read buffer was filled and data returned the software would ask the MCU for another set of data. But because the serial reader function did not recognize "0"s sometimes the read buffer would never fill up and then because the read buffer never filled the software never asked for the MCU for more data. To overcome this problem we converted all "0"s to an asterisk "*" on the MCU before sending data, and then converted all "*"s back to "0"s.

The serial reader function was written this way to ensure none of the samples were lost, and also because with the function written this way it is possible to control the flow and timing of when data is requested from the computer running the software. While this was all fine and good, in the end we re-wrote the serial reading function to be able to read all the data in the buffer when it is full or to read back all the data that was received after a certain amount of time, whether or not the read buffer was full.

The last big problem we ran into was related the the malfunctioning INMP441 microphone. Because this microphone was not working correctly, we built an analog microphone circuit and used the EFM32WG-STK3800 on board ADC to digitize the output from the microphone circuit and replace the INMP441 microphone. While we were in the process of calibrating the onboard ADC we fried the CortexM4 MCU on the EFM32WG-STK3800. We think we plugged the ADC input directly into the 15V rail voltage of the microphone circuit (it is only one row away from the microphone output on the breadboard)

1.7

Conclusions

Our original project was to design a sonic impulse location system. The system required a network of multiple audio collection nodes. Each node would be controlled through a host computer that could control the operation of each node through a series of one or two byte commands. However, the development of the nodes proved to be a large task in itself. So, due to time constraints our project was limited to the development of the sonic location node.

We were able to develop a versatile node capable of gathering data over a set period of time and then transmit the data back to a host computer for processing. On the node, we were able to program a microcontroller to communicate with MEMS microphone using the I2S protocol. Using the same protocol, we were able to directly control the sampling frequency of the microphone in addition to collecting data from the microphones ADC. The microcontroller was able to communicate with the host computer through an RS232-to-USB conversion. Through this channel the microcontroller was able to receive instructions from the host computer and the return audio data collected from the MEMS microphone. Finally, we were able to receive the audio data on the host computer using a python GUI. The GUI was able to convert the data into a usable format and display the audio data.

In conclusion, we were able to develop the necessary system components for an audio data collection node which could be used inside of a larger sonic impulse location system. The node was able to collect data using a predetermined sampling frequency and return the data back to a host computer. The data was then processed and the audio spectrum of the sound was displayed for a user. We were able to create a node that could be used as an audio frequency spectrum analyzer. The creation of the node showed that, given time, similar nodes could be combined to form a sonic location system.

1.8

Media

We created a video demonstrating the GUI and it recieving information from the microcontroller. It also demonstrates the graphing of the information coming from the microcontroller.

1.9

References

INMP441: http://www.invensense.com/mems/microphone/documents/INMP441.pdf
EFM32QFN24: http://www.silabs.com/Support%20Documents/TechnicalDocs/EFM32TG110.pdf