Comms. System Final Report

December 18, 2015

Nicholas Kizy
Joshua McCready
Reid Pinkham
Brian Chang

Contents :

Abstract

This is a final report of an EECS 330 Communications Project. The goal of this project was to send digital audio data between two systems. We decided to make walkie-talkies using XBee Radios. Our original intent was to send data back and forth between the two boards; however, due to limited time, we were only able to configure the boards to send data one way.

I. Design

The overall concept and flow of data through the system is explained. We give rationale behind how choices were made to accomplish each stage of the design with hardware. PCB design using altium is discussed. Our design was guided by the Friis transmission formula,the Nyquist sampling theorem and by the frequency content of voice. Finally a detailed budget is presented.

System concept

Figure 1 details the operation of the major components of the system. The audio input breakout produces a continuous audio voltage signal. The CPU then samples the continuous audio voltage at a variable rate with an 12-bit-wide ADC, which is then reduced to 8 for ease of transmission. The ADC uses a reference voltage of 2.5 V, so the signal becomes full scale at 2.5 rather than the power supply voltage of 3.3 V. The data stream representing the audio signal is then transmitted over a serial interface to a radio which communicates that data to another identical system. The other system's CPU then receives the serial data and outputs it as a discrete voltage signal over an identical range of 2.5 V. The discrete signal from the DAC is then amplified and output to a speaker. The push button provides digital input to the CPU to initiate the communication process.

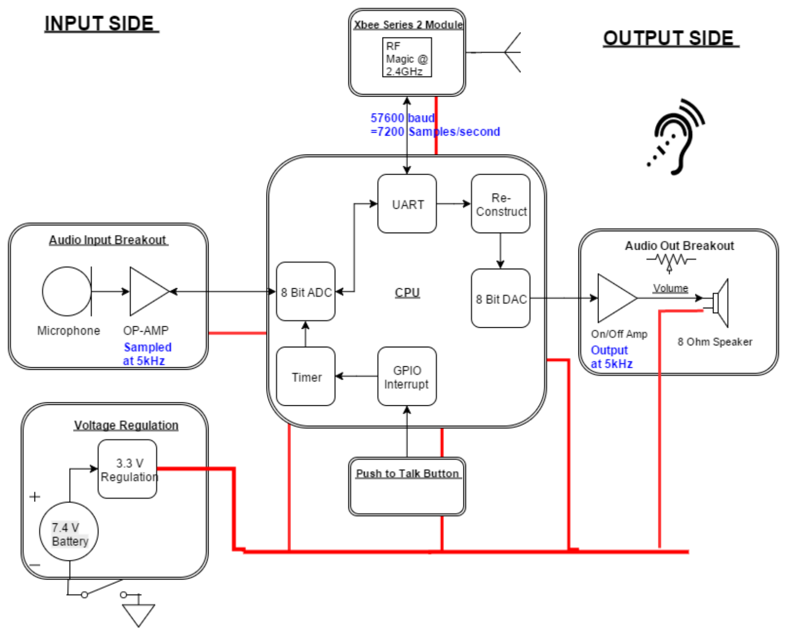


Figure 1: Major Systems Block Diagram: The above block diagram displays each major element of the system within double lined blocks.

Part selection

The hardware will be contained on a custom printed circuit board (PCB). Major components of the system include the battery, power regulators, microcontroller, microphone, speaker amplifier, speaker, and XBee transmit/receive device.

The batteries are 18650 lithium ion batteries on loan from the Michigan eXploration Lab. Two of these in series will supply enough voltage and current to drive both the 5V and 3.3V systems. The batteries come in pairs in a shrinkwrapped case with integrated wire leads. These leads are connected to the circuit board with a small power connector.

Two power regulators are designed to be used on the board. An LM317 is used for the 5V power, and an LT1763 is used to provide 3.3V low-noise power. We are choosing to use a LM317 for the 5V power instead of a second LT1763 because of power requirements for a potential long-range XTend module. Driving both a speaker amplifier and an XTend module requires more current than the LT1763 is rated to deliver. Although this will not affect the performance of the two way radio aspect, should an XTend module be used in the future, this board will require greater 5V power capabilities than that of the LT1763. The LT1763 regulator is specifically designed to produce a very low-noise supply for use with microcontrollers. The noise can be further reduced by placing a large power capacitor, which runs off of the 3.3V power produced by the LT1763, close to the microcontroller.

A 64-pin MSP430F2618 microcontroller from Texas Instruments is used. This microcontroller has 64 pins, and has enough input and output pins available to drive the UART controlled XBee. The DAC's and ADC's are built in to interface directly with the microphone pre amp and the speaker audio amplifier. The microcontroller also has a relatively high internal clock frequency capable of performing the computations required to process the microphone and speaker data.

A pre-built microphone and preamp module is chosen for use in the system. The microphone module comes pre-assembled from SparkFun and only requires 3.3V power. The output from the microphone preamp is sent into an 8-bit ADC in the microcontroller.

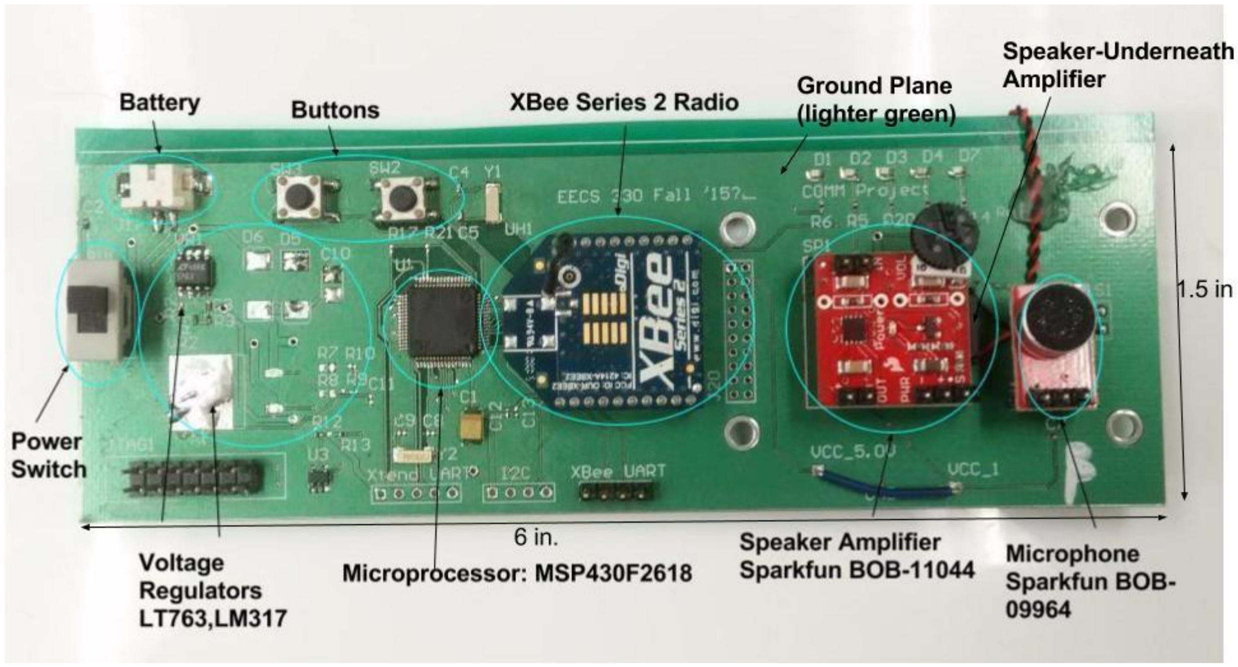
A pre-built mono speaker amplifier from SparkFun is also used. This eliminated concerns of designing a speaker amplifier capable of accepting the range of input which the microcontroller can output. The amplifier accepts analog input from the DAC in the microcontroller and produces an amplified audio signal which can drive a 1.4W 8-Ohm speaker.

The speaker chosen isa high power micro-speaker produced by CUI Inc. This speaker is similar to ones used in modern smartphones. This speaker will provide us with the high range of output frequencies required to replicate voice, while providing a compact footprint.

The XBee Series 2 module is used as the radios. These modules have a range of up to 300 ft indoors, which is enough to allow our radios to work inside the laboratory classroom. One XBee is configured as a Coordinator AT and the other as a Router AT (Wall RT). The coordinator is used to establish the network, and the router is used to extend the network. X-CTU, a Windows-based application by Digi, was used to perform the configurations.

PCB design
Altium is used to design the circuit boards. The boards are 1.5 inches by 6 inches and contain two layers. The board contains the speaker, microphone, headers, microprocessor, and the XBee Series 2. The PCB is also configured to account for the long-range XBee Pro, even though this is outside of the scope of the project. A layout of the PCB is shown below in Figure 2. Bay Area Circuits is the circuit board manufacturer.

The board features a ground pour. The ground pour is on both the top and bottom layer of the board. It fills in all of the empty space between traces and connects that space to ground. This effect creates a ground plane on each side of the board to reduce noise and interference.


Battery
Buttons
XBee Series 2 Radio
Ground Plane (lighter green)
Speaker-Underneath Amplifier
1.5 in
Power Switch
6 in.
Voltage Regulators LT763,LM317
Microprocessor: MSP430F2618
Speaker Amplifier Sparkfun BOB-11044
Microphone Sparkfun BOB-09964
Figure 2: Printed Circuit Board

Signal analysis
The typical adult human voice has the most essential frequency components ranging from 300 Hz to 3400 Hz. While this means that our aimed-for sampling frequency should be at least 6800 Hz, the greater the frequency we can sample at, the better chance we have at reconstructing the data. Theoretically, consistent with the sampling theorem, we can expect to be able reconstruct a signal if we sample at twice the bandwidth, according to the Nyquist sampling rate. However, realistically, errors can cause flawed data reconstruction. To resolve the problem, we hope that sampling more can allow us to reconstruct the data with better quality. 8 kHz is minimally sufficient if we only wish to satisfy the Nyquist Shannon sampling criterion. However, the voice consists of many peaks and jitters that require greater sampling if we want those to be part of the reconstructed data. To capture these finer elements of speech, the ideal voice sampling rate will be above 16 kHz.

Figure 3 displays a diagram of the entire communication system. It includes each of the factors associated with achieving our ideal sampling rate.

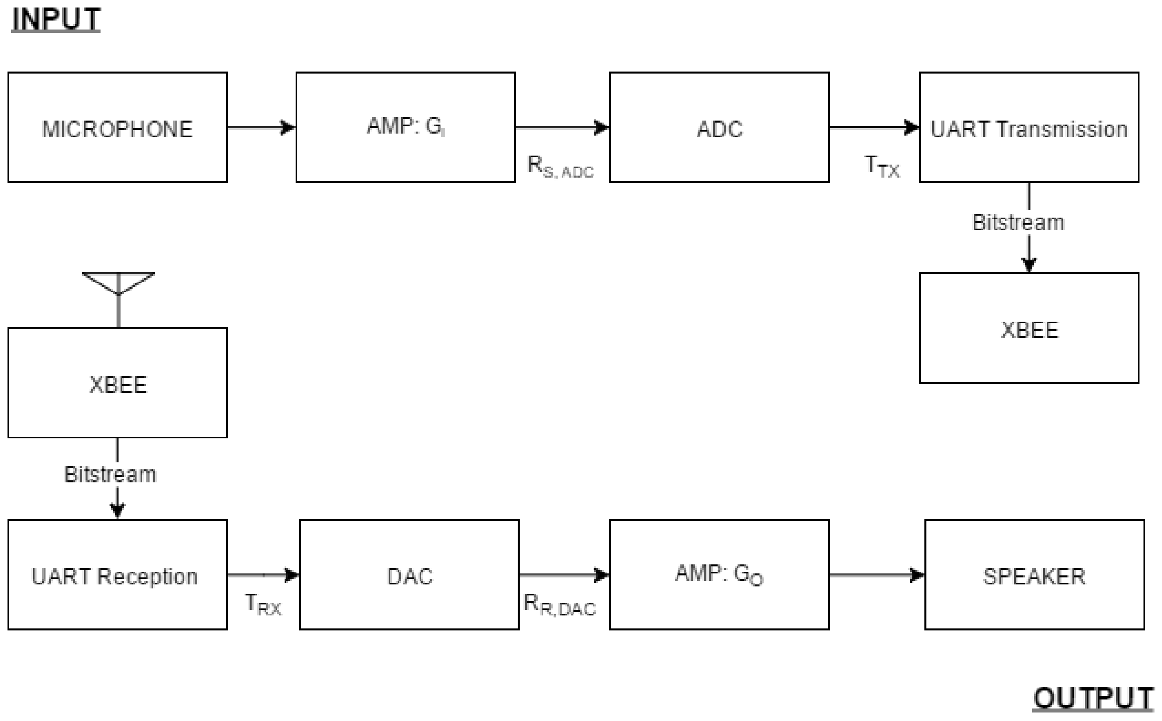


Figure 3: Voice Signal Diagram:
Block diagram describing the parameters which will impact the sampling rate.

On the input side we want to determine the ADC's required sampling rate, $R_{S,\,ADC}$as well as the UART channel's baud rate. On the output side we want to determine the UART channel's required baud rate and the DAC's required rate of reconstruction, $R_{R,DAC}$. $T_{TX} = T_{RX} = T_{U}$ is the time it takes UART to transmit and receive one sample from the 8-bit ADC. Let F be the safety factor, which accounts for the activity the microprocessor must do. First, let us consider everything in terms of time.

$$F * (T_{S,\,ADC} + T_{U}) = T_{in}$$
$$F * (T_{S,\,DAC} + T_{U}) = T_{out}$$
$$T_{S,ADC} + T_{U} = T_{R,\,DAC} + T_{U} \leq T_{sampling}/F$$

We know $T_{S,\,ideal} = 62.5\ \mu s$ and $T_{S,\,necessary} = 125\ \mu s$ based on the sampling criterion. Additionally, there is some constraint on how fast UART is able to function adequately. Let us assume that the maximum baud rate (Bits/s) is 230400 baud. Given that each sample is 8 bits, this means the frequency UART will likely work at is 28.2 kHz, so $T_{U} = 34.7\ \mu s$. It will clearly be difficult to achieve the ideal sampling frequency. Let us assume that $F = 1.5$. We can now solve for the required sample rates for the DAC and ADC.

$$T_{S,ADC} = T_{R,\,DAC} = T_{Sampling}/F - T_{U}$$

This is the equation governing our system and will help guide how fast the DAC and ADC are setup.

Antenna analysis

The XBee Pro 2 modules use quarter wavelength monopoles to transmit the digital data at 2.4 GHz. In these calculations, it is assumed that antenna's impedance is matched with the driving circuit, and the ground plane on the circuit board is sufficiently infinite.

First, the theoretical efficiency of the monopole is calculated. A quarter wavelength monopole antenna has a radiation resistance $R_{rad} = 36.5\ \Omega$. Using a 1 mm diameter wire, the resistive loss can be calculated using the following formula.

$$R_{loss} = \sqrt{\frac{\omega\mu}{(2\sigma)}}\frac{l}{2\pi b} = 12.8 \cdot 10^{-3}\Omega$$

This value for $R_{loss}$is much smaller than R_{rad}, thus the expected efficiency is very high. This is indeed the case, with the calculated radiating efficiency being $\xi = 0.998$.

Next, the effective area of the monopole is calculated. The directivity of the monopole antenna is found to be 3 dB higher than an equivalent dipole antenna. Thus, $D = 3.635$. From this, the effective area, A, can be calculated.

$$A = \frac{\lambda^2 D}{4\pi} = 4.52 \cdot 10^{-3}\ m^2$$

This value of $A$is the same for the transmitting and receiving antenna. Using this and the radiating efficiency, the power received at a given distance is calculated using the Friis transmission formula. The XBee modules transmit at $P_t = 63\ mW$.

$$P_{rec} = \frac{A_1 A_2 \xi_1 \xi_2}{\lambda^2 R^2} P_t = 824\ nW$$

Here, a distance of $R = 10\ m$ is used. The minimum receive power is quoted from the spec sheet as $P_{rec} =- 103\ dBm = 0.0794\ pW$. This is roughly ten times the thermal noise at 300 K. At this low level, the receiving power is roughly one million times greater than needed. This means there should be no issues with signal quality across the classroom.

Again, using the Friis transmission formula, a maximum theoretical range of 1.27 km should be possible with the small monopoles operating in a line of sight environment.

Budget and additional resources

The \$200 budget was adhered to. Many parts required for the project were sampled or provided by the Michigan eXploration Lab. The sampled and provided items are listed in Table 1.

Item	Vendor	Amount
Header, 7-pin	Samtec	4

UART Header	Samtec	8
I2C Header	Samtec	4
MSP430F2618	Texas Instruments	4
LM317	On Semiconductor	4
XBee Series 2	eXploration Lab	2
XTend Header	Samtec	4
Mic Header	Samtec	4
Speaker Header	Samtec	4

Table 1: Summary of Sampled Parts

Table 2 lists the purchases which were made for the project. We were $0.38 under budget.

Item	**Unit Cost**	**Number**	**total**
Speaker Amp	7.95	4	31.8
Microphone	7.95	4	31.8
Shipping	4.53	1	4.53
Circuit Boards	81.95	1	81.95
Digi Shipping	3.54	1	3.54
LT1763	4.15	3	12.45
Speaker	4.09	3	12.27
Oscillator	0.82	6	4.92
Switch	2.21	3	6.63
Blocking Diode	0.3	6	1.8
LED	0.24	15	3.6
Power Capacitor	0.4333	10	4.333
		Total:	**199.62**

Table 2: Summary of Purchases and Costs

II. Fabrication

Procedures

Four circuit boards were ordered from Bay Area Circuits, a PCB manufacturer in the San Francisco area. Bay Area Circuits produced our PCBs at a lower cost than the alternative, Advanced Circuits. Our boards arrived three days late due to the Thanksgiving holiday. Because of manufacturing procedures, we received six boards in total. The boards came in pairs of two, so each individual board was separated using a bandsaw. We were expecting to do this, thus there was a keepout region designed between the two boards.

Each board primarily contains surface mount components. These were soldered by hand, using both conventional solder and solder paste. A hot air rework station was used when removing the microprocessor when necessary.

Challenges faced
The board had many issues which had to be overcome for the board to work.

The first and most significant issue was the size of the component pads. The design intended for 0603 sized components to be used as many of the miscellaneous capacitors and resistors. However, all except two of these components were actually made with 0402 sized pads, which are less than half the size of the 0603 size pads. This error was in the design. There were some 0402 parts on hand, however not all of the components were in that size. This meant that most of the parts being soldered to the 0402 sized pads were actually much larger in size (0603 or 0804 sized).

The second major issue with the fabrication was the power connector. The connector was connected in reverse of the typical way. The meant with initial testing, power was being supplied with the incorrect polarity.

Another major issue with the power was the voltage regulators themselves. The LT1763 voltage regulator for the 3.3V bus did not function when initially assembled. The output of the regulator was a constant offset of the input voltage, but not 3.3V. Additionally, the LM317 regulator did not function as expected. Initially, it output zero voltage, but debugging revealed it was actually outputting a similar value as the LT1763 to the 5V rail. Figure 4 displays one of the boards during the debugging.

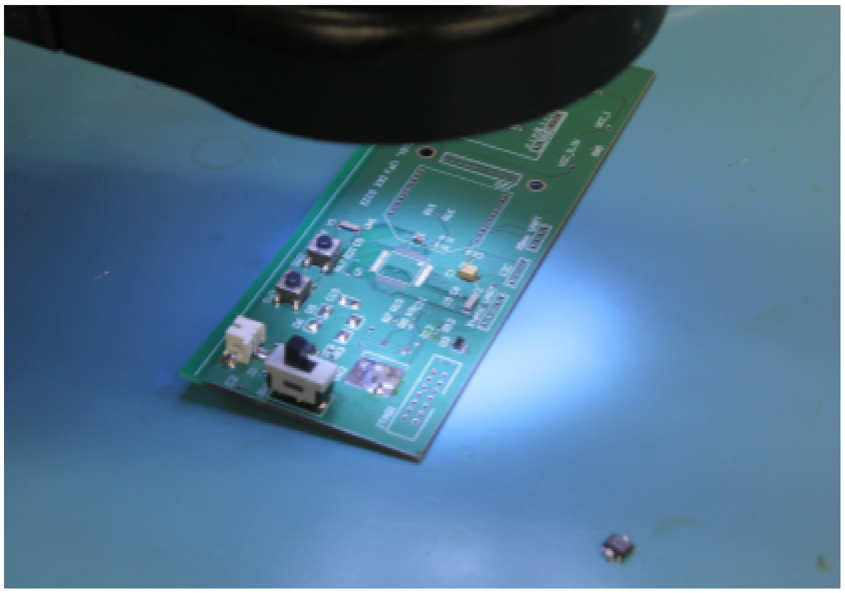


Figure 4: Debugging the Voltage Regulation
One of the circuit boards is shown under microscope. To the right of the board is the a voltage regulator that had just been de-soldered

One general design issue with the board was the ground plane, which was supposed to provide thermal relief when soldering. The ground plane was poured in too closely to many of the through hole pads. When components were soldered to these parts, it was very easy for them to be shorted directly to ground, significantly increasing fabrication time.

There were a few minor errors in the silkscreen on the board. First, the location of the "COMM Project" description was placed in the incorrect location. Second, only Reid Pinkham's name appeared on the board. This was determined to be a compatibility issue between the manufacturing files produced by Altium and the manufacturer.

Challenge mitigations

The issue of the component size was overcome in multiple ways. Some nonessential components to the design were left out. For example, C6 was removed because the only parts with the correct capacitance on hand had over 10 times the footprint area of the pads. For the essential components, 0603 parts were used in place of the 0402 parts. These were attached to the pads by rotating them by 90 degrees, and adding a jumper wire. An example of one of these fixes is displayed in Figure 5 below. The width of the 0603 part covers almost the entire 0402 pad.

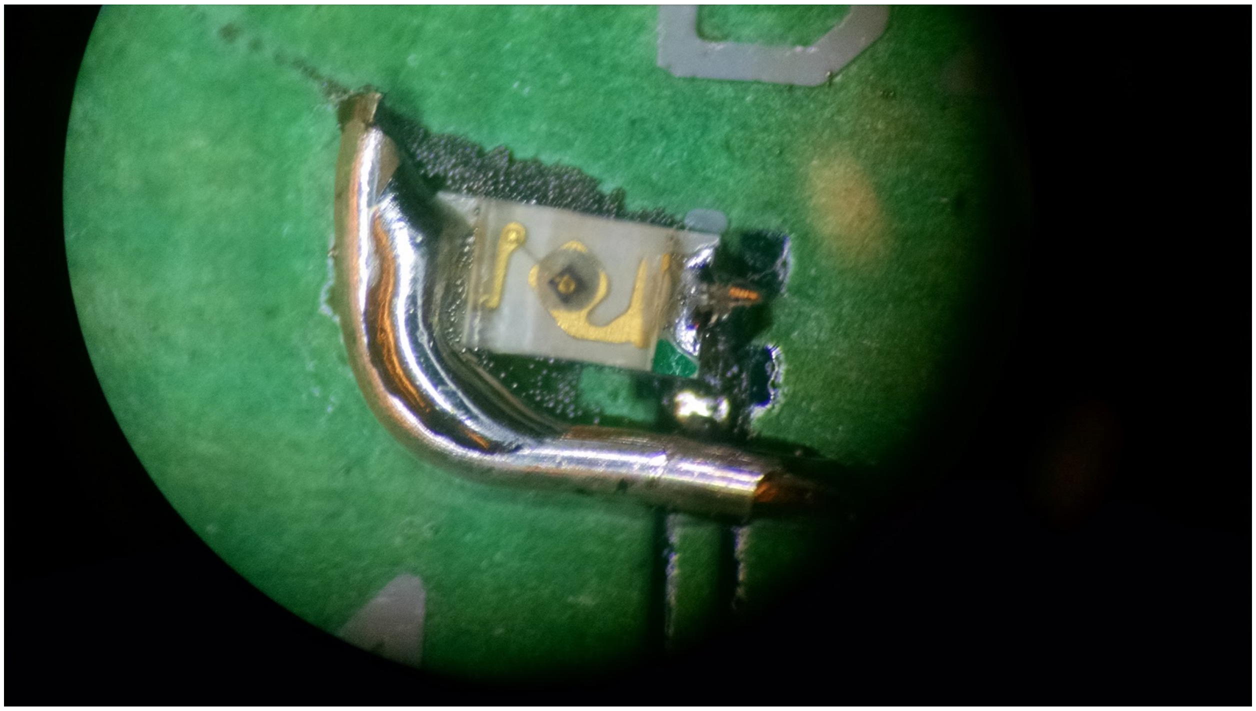


Figure 5: 0603 Parts: Image of an 0603 part with jumper on an 0402 pad. For scale, the size of the LED is 1.5 mm long by 0.8 mm wide.

The power connector issue was resolved by switching the polarity of the battery connectors. This was not a major issue because the batteries required a connector replacement anyways. This will be an issue in the future if these boards are used elsewhere in the MXL lab.

After extensive trial and error, it was determined that the issue of the voltage regulators was in the parts themselves. The LT1763, which was originally used, had the part number of LT1763-33. This part was intended for use only with 3.3V output. Because the design was not working with this part, a standard

LT1763 was tried and found to work perfectly. The MXL donated two of these regulators for use in the project, resolving the first voltage regulation issue.

The second voltage regulator, the LM317, was not resolved. This regulator was intended to supply high power to the speaker amplifier, and possibly the long range antenna if used in the future. Due to time constraints, we decided to omit the LM317 from the board and to use the LT1763 to provide power to the whole board. The limitation is that it can only supply 500mA maximum. Because of this, the speaker power was very limited. The jumper connecting the 5V and 3.3V rails can be seen in Figure 2.

The issue of clearances on the ground plane was mitigated by inspecting every through hole solder joint many times. If there was a short between the joint and the ground plane (Figure 6), the component was removed and resoldered with a minimal amount of solder to prevent any grounding issues. This method worked well because many of the components had at least one ground connection, which allowed for at least one strong solder joint. In the case of many of the headers, this single pin served to pin the connector down mechanically.

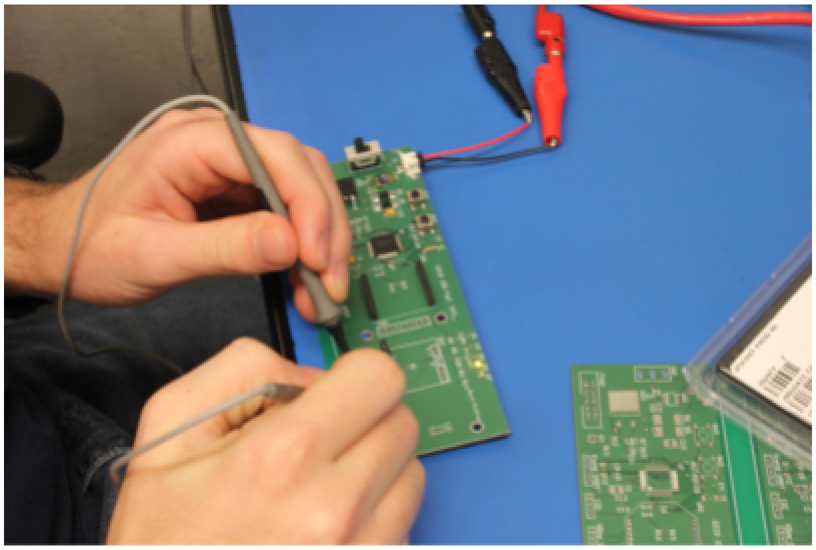
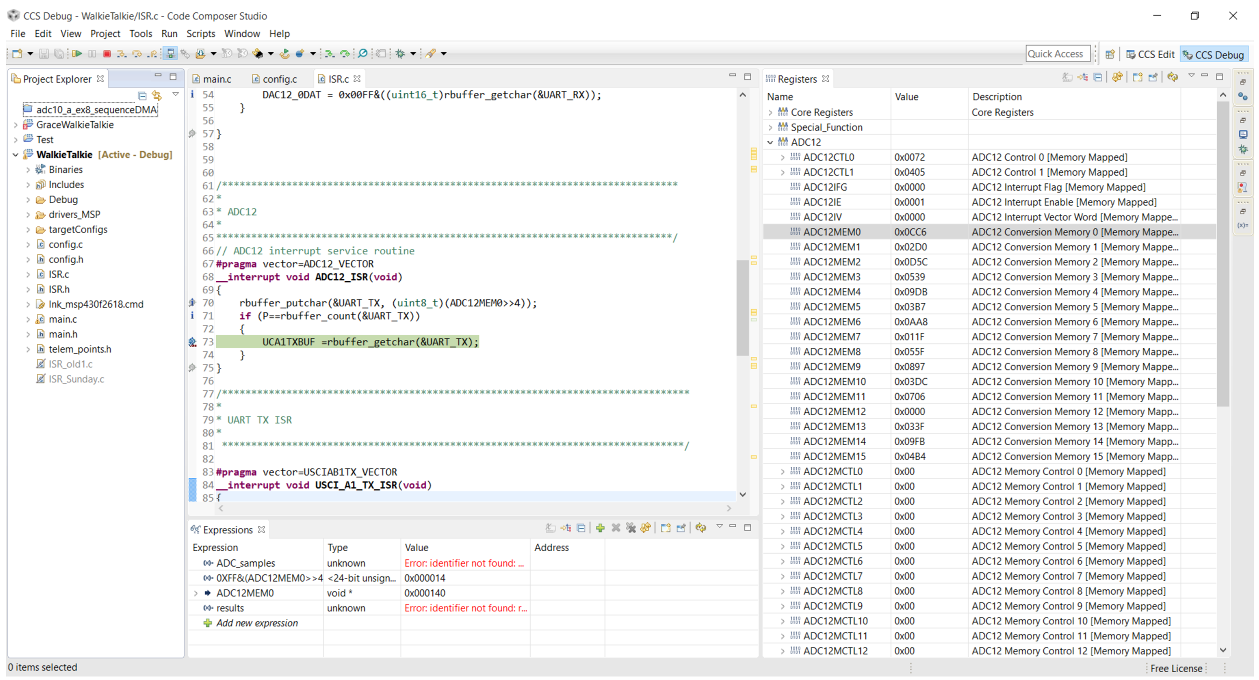


Figure 6: Continuity Check Between Ground and Power
Skillful hands carefully probe the continuity between the ground plane and the power rails. The LED shows that at this point in fabrication they were shorted.

III. Software

MSP430 development tools

There were several development tools that were invaluable to producing a functional code base to refine the walkie talkies' functionality. The first tool at our disposal was the integrated design environment (IDE) Texas Instruments Code Composer Studio (CCS). This tool allowed for the programming of the MSP430F2618 processor through a Joint Test Action Group (JTAG) programming interface. CCS allowed for easy development of complex projects without the need for makefiles or running the compiler for the MSP430F2618 from command line. Beyond simply programming a processor, JTAG and CCS provided for real time debugging. The code could be paused, breakpoints could be set, and both variables and memory could be inspected. This feature is crucial to any embedded system's development. The layout of the design environment CCS is displayed in Figure 7.



adc10_a_ex8_sequenceDMA
GraceWalkieTalkie
Test
WalkieTalkie [Active - Debug]
- Binaries
- Includes
- Debug
- drivers_MSP
- targetConfigs
- config.c
- config.h
- ISR.c
- ISR.h
- lnk_msp430f2618.cmd
- main.c
- main.h
- telem_points.h
- ISR_old1.c
- ISR_Sunday.c

```c
        DAC12_0DAT = 0x00FF&((uint16_t)rbuffer_getchar(&UART_RX));
    }

}



/*****************************************************************************
*
* ADC12
*
*****************************************************************************/
// ADC12 interrupt service routine
#pragma vector=ADC12_VECTOR
__interrupt void ADC12_ISR(void)
{
    rbuffer_putchar(&UART_TX, (uint8_t)(ADC12MEM0>>4));
    if (P==rbuffer_count(&UART_TX))
    {
        UCA1TXBUF =rbuffer_getchar(&UART_TX);
    }
}

/*****************************************************************************
*
* UART TX ISR
*
 *****************************************************************************/

#pragma vector=USCIAB1TX_VECTOR
__interrupt void USCI_A1_TX_ISR(void)
{
```

Expression	Type	Value	Address
ADC_samples	unknown	Error: identifier not found: ...	
0XFF&(ADC12MEM0>>4	<24-bit unsign...	0x000014	
ADC12MEM0	void *	0x000140	
results	unknown	Error: identifier not found: r...	
Add new expression			

Name	Value	Description
Core Registers		Core Registers
Special_Function		
ADC12		
ADC12CTL0	0x0072	ADC12 Control 0 [Memory Mapped]
ADC12CTL1	0x0405	ADC12 Control 1 [Memory Mapped]
ADC12IFG	0x0000	ADC12 Interrupt Flag [Memory Mapped]
ADC12IE	0x0001	ADC12 Interrupt Enable [Memory Mapped]
ADC12IV	0x0000	ADC12 Interrupt Vector Word [Memory Mappe...
ADC12MEM0	0x0CC6	ADC12 Conversion Memory 0 [Memory Mappe...
ADC12MEM1	0x02D0	ADC12 Conversion Memory 1 [Memory Mappe...
ADC12MEM2	0x0D5C	ADC12 Conversion Memory 2 [Memory Mappe...
ADC12MEM3	0x0539	ADC12 Conversion Memory 3 [Memory Mappe...
ADC12MEM4	0x09DB	ADC12 Conversion Memory 4 [Memory Mappe...
ADC12MEM5	0x03B7	ADC12 Conversion Memory 5 [Memory Mappe...
ADC12MEM6	0x0AA8	ADC12 Conversion Memory 6 [Memory Mappe...
ADC12MEM7	0x011F	ADC12 Conversion Memory 7 [Memory Mappe...
ADC12MEM8	0x055F	ADC12 Conversion Memory 8 [Memory Mappe...
ADC12MEM9	0x0897	ADC12 Conversion Memory 9 [Memory Mappe...
ADC12MEM10	0x03DC	ADC12 Conversion Memory 10 [Memory Mapp...
ADC12MEM11	0x0706	ADC12 Conversion Memory 11 [Memory Mapp...
ADC12MEM12	0x0000	ADC12 Conversion Memory 12 [Memory Mapp...
ADC12MEM13	0x033F	ADC12 Conversion Memory 13 [Memory Mapp...
ADC12MEM14	0x09FB	ADC12 Conversion Memory 14 [Memory Mapp...
ADC12MEM15	0x04B4	ADC12 Conversion Memory 15 [Memory Mapp...
ADC12MCTL0	0x00	ADC12 Memory Control 0 [Memory Mapped]
ADC12MCTL1	0x00	ADC12 Memory Control 1 [Memory Mapped]
ADC12MCTL2	0x00	ADC12 Memory Control 2 [Memory Mapped]
ADC12MCTL3	0x00	ADC12 Memory Control 3 [Memory Mapped]
ADC12MCTL4	0x00	ADC12 Memory Control 4 [Memory Mapped]
ADC12MCTL5	0x00	ADC12 Memory Control 5 [Memory Mapped]
ADC12MCTL6	0x00	ADC12 Memory Control 6 [Memory Mapped]
ADC12MCTL7	0x00	ADC12 Memory Control 7 [Memory Mapped]
ADC12MCTL8	0x00	ADC12 Memory Control 8 [Memory Mapped]
ADC12MCTL9	0x00	ADC12 Memory Control 9 [Memory Mapped]
ADC12MCTL10	0x00	ADC12 Memory Control 10 [Memory Mapped]
ADC12MCTL11	0x00	ADC12 Memory Control 11 [Memory Mapped]
ADC12MCTL12	0x00	ADC12 Memory Control 12 [Memory Mapped]

Figure 7: Texas Instruments Code Composer Studio Workspace

Above is a screen capture featuring the most important aspects of the CCS development environment. The left section is the project explorer, which allows for easy navigation of the code in a project. The upper middle section is the code editor, where breakpoints can be placed. The code in the screen capture is stopped at the breakpoint highlighted in green. On the left, the registers of the microprocessor are shown. The ADC12 module reads 0x0CC6 which corresponds to 2.0 V

The JTAG programmer we used was MSP-FET430UIF, which was available for our use in both the EECS 330 lab room and the Michigan Exploration Lab. A photo of the programmer is displayed in Figure 8.

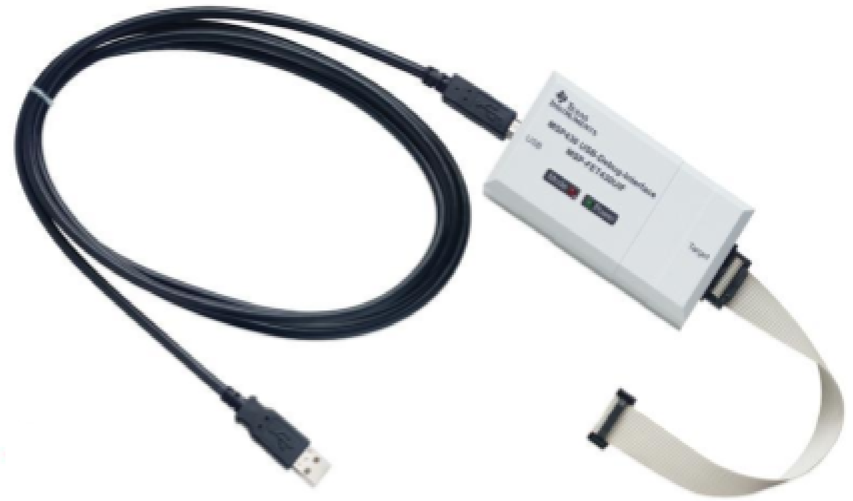
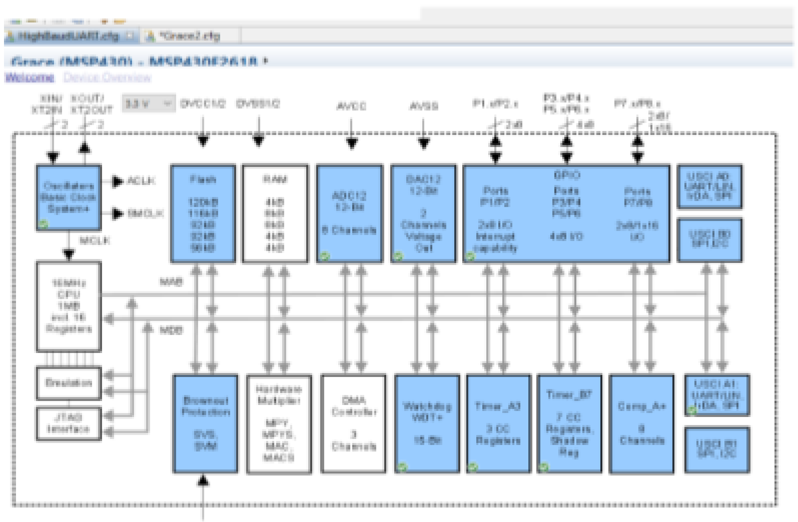


Figure 8 : MSP-FET430UIF
The MSP430-FETUIF interfaces with the microprocessor using the connector in the left and connects to a computer via USB. Image courtesy of www.TI.com

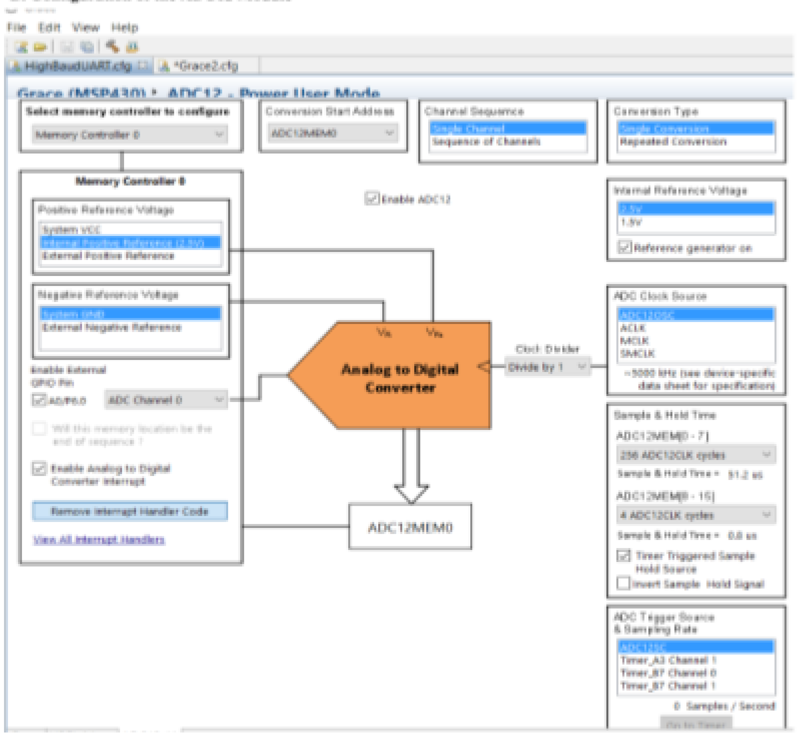
Grace is a graphical configuration tool for the MSP430 series of processors provided by Texas Instruments. With the help of Grace, the MSP430's ADC, DAC, UART, and timer peripheral modules were configured properly in minutes rather than hours. Without Grace configuring a peripheral module, one would need to reference the MSP430x2xx Family User Guide (Appendix A) and read how each register of the processor must be configured for the task at hand. Understanding of the modules was still required, but configuration went much more quickly using Grace. Once a particular peripheral module of the microcontroller was selected, drop-down menus allowed particular features and configurations to be selected. Then the configuration was compiled and code was automatically generated. The configuration of the DAC module of the microprocessor in this fashion is detailed in Figure 9.

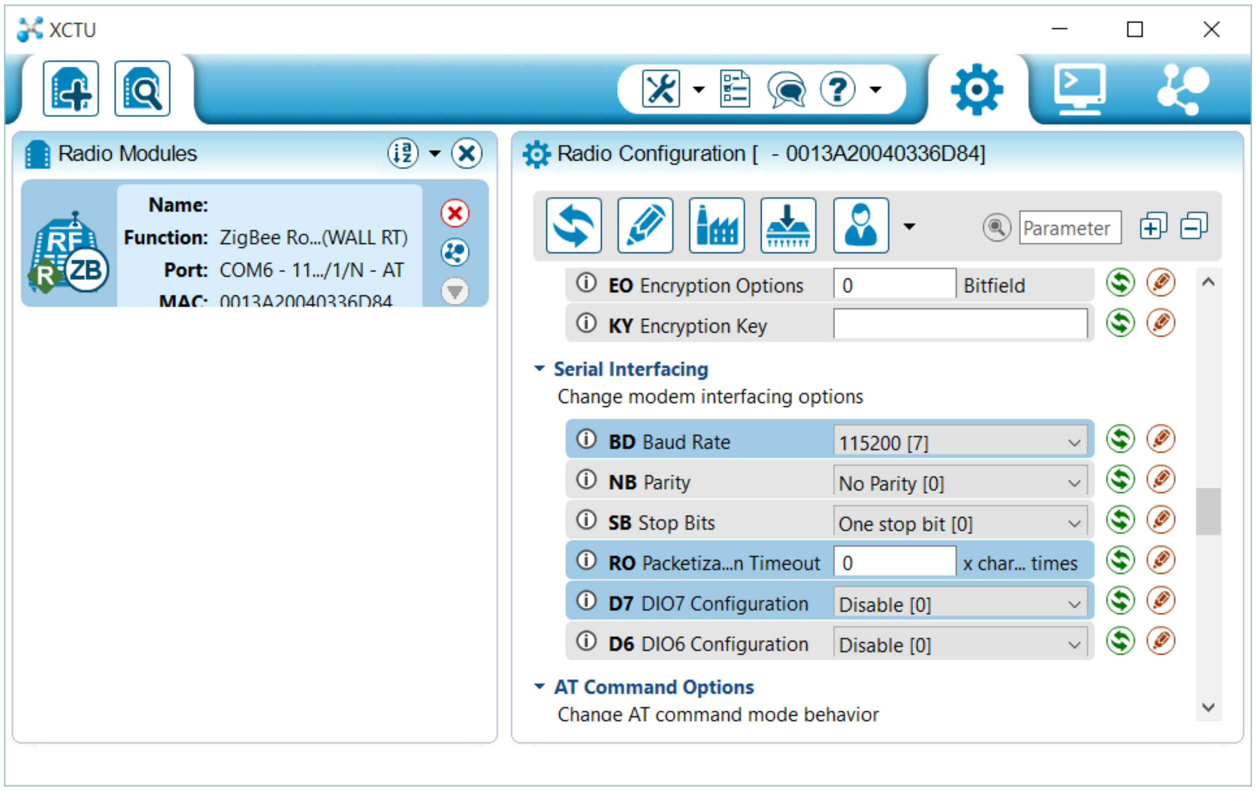
Figure 9: Grace - Graphical Peripheral Configuration Tool
Below two screen captures of Grace are shown, In part A the all the modules of the MSP430 that can be configured with Grace are shown. In part B, the ADC12 module has been selected and it's configuration is shown. This graphical tool takes using many separate tables of shortened keywords to compose the correct configuration for a given register of the MSP430. Once compiled the project in Grace auto-generates code which is included in Appendix C.

A: Overview of MSP430F2618 peripheral modules



B: Configuration of the ADC12 Module





Name:
Function: ZigBee Ro...(WALL RT)
Port: COM6 - 11.../1/N - AT
MAC: 0013A20040336D84

EO Encryption Options	0	Bitfield
KY Encryption Key		

Serial Interfacing
Change modem interfacing options

BD Baud Rate	115200 [7]	
NB Parity	No Parity [0]	
SB Stop Bits	One stop bit [0]	
RO Packetiza...n Timeout	0	x char... times
D7 DIO7 Configuration	Disable [0]	
D6 DIO6 Configuration	Disable [0]	

AT Command Options
Change AT command mode behavior

Figure 10: XCTU XBee configuration tool. The right window shows that one ZigBee radio is connected. On the left, it's configuration is shown.

XBee radio development tools
The XBee Series 2 Pro radio modules used in the project were configured using XCTU, a Windows based graphical user interface provided by the Digi International, the radio module's manufacturer. It allowed us to set up the radios in the proper network configuration and see the data they received in real time using a built in terminal. See Figure 10 for an example of the configuration tool that was used.

Software Discussion
The program running on the microcontroller had three important parts: namely main.c, config.c, and ISR.c. main.c was the main program from which the buttons were polled for input. config.c contained the functions for configuring the clock system, ADC, DAC, UART, I/O ports, and timers. ISR.c contained the interrupts that diverted the main function to process the UART data, the DAC, and ADC functionality.

The ADC12 module was part of the MSP430 and allowed it to read in the microphone input. It was configured to use an internal voltage reference of 2.5 V, and sample on the rising edge of the output bit of the timer A0 module. Timer A0 operated in tandem with the ADC12 module and counted up to make the ADC12's sampling frequency the desired value. For the configuration used in the demonstration, the timer's frequency was 3.33 kHz. This configuration was based on example code from TI.

The DAC12 module was part of the MSP430 and was used to output a voltage to the audio amplifier. It was configured with a gain of 1, and to update its output voltage when a new value was assigned to the

DAC12_0DAT the memory block. The DAC12's output was updated using timer B0 at a frequency of 3.33 kHz in the demonstration. The configuration was obtained using Grace.

UART (serial communication) was used to communicate digital data to and from the XBee radio modules. The baud rate (bits/s) for UART was configured to be 115200 for the demo, but many other configurations were obtained from Grace. Interrupts handled the reception of the data going to and coming from the XBees.

Once everything was configured, the code functioned as shown in Figure 11. As shown in the state diagram, the button controlled transmission of data. It was clear that interrupts did much of the work in both obtaining and outputting the samples. This interrupt method optimized the speed of the code because everything happened without the need for polling.

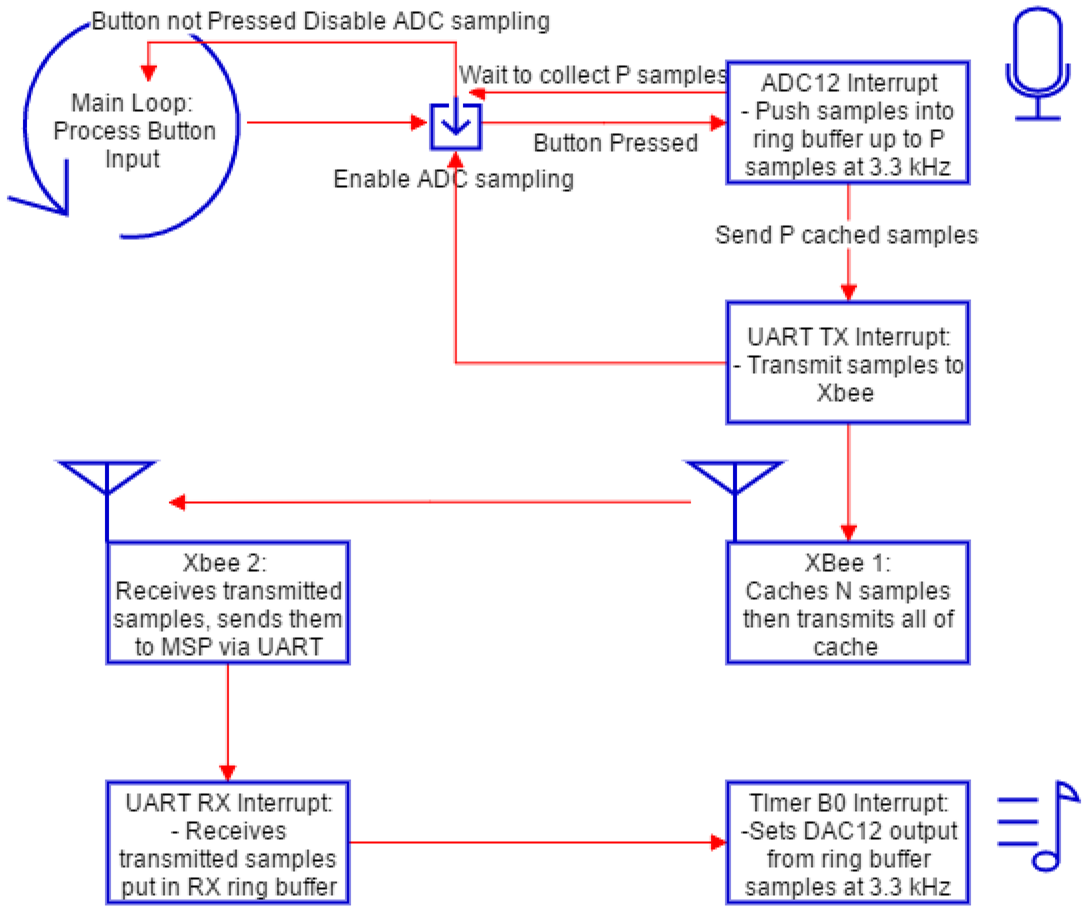

Button not Pressed Disable ADC sampling
Main Loop: Process Button Input
Wait to collect P samples
Button Pressed
Enable ADC sampling
ADC12 Interrupt - Push samples into ring buffer up to P samples at 3.3 kHz
Send P cached samples
UART TX Interrupt: - Transmit samples to Xbee
Xbee 2: Receives transmitted samples, sends them to MSP via UART
XBee 1: Caches N samples then transmits all of cache
UART RX Interrupt: - Receives transmitted samples put in RX ring buffer
TImer B0 Interrupt: -Sets DAC12 output from ring buffer samples at 3.3 kHz

Figure 11: Functionality of Microcontroller Code
Shown above is a pseudo-state machine which describes how the the software works to transmit voice from a microphone to the other board, and how it then outputs the voice to the speaker.

IV. System Testing and Demonstration

Testing

Once the hardware state had settled and parts were no longer being soldered and desoldered, system testing began. We examined how the UART input compared with the UART output and how the audio input and outputs behaved. Figure 12 and 13 explain why the signal quality was poorer than desired.

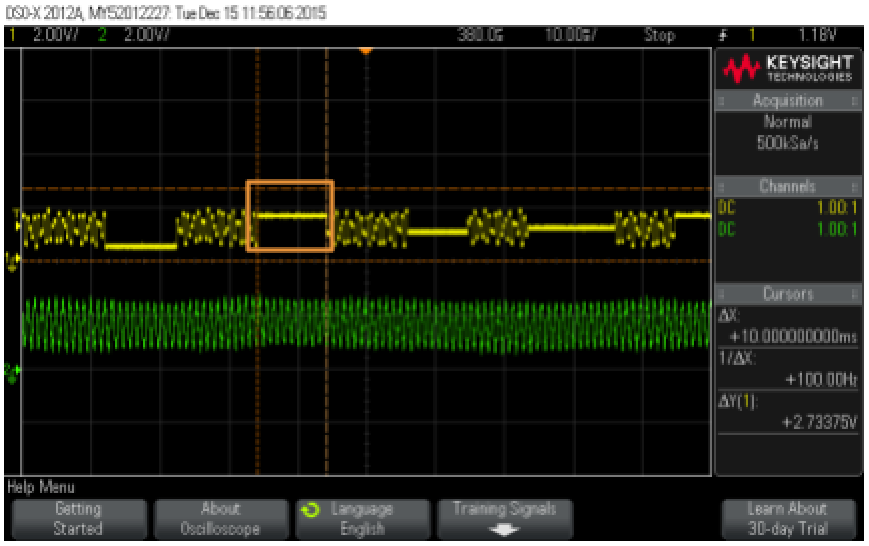


Figure 12: Audio input (green) and audio output (yellow)

The audio signal input, shown in green, was generated by whistling. The input was sampled by the ADC at 3.3 kHz and sent over UART using the XBees to produce the output audio signal shown in yellow. The output signal was reconstructed at the same 3.3 kHz rate. There were consistent periods where the output signal was constant, referred to as "dead zones." One dead zone is boxed in orange on the oscilloscope capture above. They likely resulted from a combination of UART data buffering by the transmitting XBee, as well as by the code which handled the UART interrupts on the MSP430. It was likely that no new data was being read in by the ADC while UART was being sent, and that no new data was being output when the UART was being read in. The dead zones seemed to occur regularly for uniform periods. Above, one dead zone lasted exactly 10 ms.

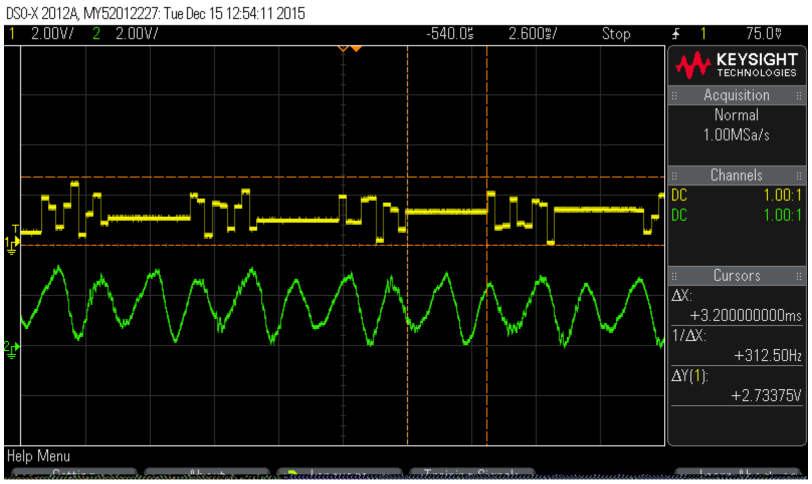


Figure 13: Direct UART connection without Radios
In this test, the XBees were taken off the PCBs, and the two UART channels were jumpered together so that the two systems could still communicate. In addition, their grounds were connected out of necessity. The continuous signal, shown in green, was the audio input from one PCB. The sampled signal, shown in yellow, was the audio output on the other board. There were still dead zones, even without XBees in the loop. This showed that there was a broader problem with the transmission of the UART data from one board to the next. It was likely that the when the UART either transmitted or received, it blocked the ADC from reading in data and blocked the DAC from outputting data.

Demonstration
Our demonstration consisted of the use of the walkie talkies to transmit the ABCs from one radio to the other. During this demonstration, one user was sent outside into the hallway to sing the ABCs at a far enough distance such that the song was inaudible to the audience. The audience listened to the song through the receiving radio inside of the EECS 330 lab. This demonstration used our two PCBs, as well as batteries to supply them with power. The output of the radio inside the lab was of poor quality because of the issues presented in the testing section. However, the song was still distinct and identifiable without prior knowledge of what was being heard.

After the initial proof of concept was presented, the reasons for the poor audio quality was shown. The sampling frequency was made to be much lower than the desired 20 kHz for the demonstration. Debugging was made easier with a lower sampling frequency, although the lowered rate contributed to poor audio quality. We briefly discussed and showed that the output audio signal had periodic dead zones when compared with the audio input from the other board. These dead zones were attributable to the buffering of the UART data by the XBees, as well as other undetermined flaws likely relating to how UART was set up. The theory was that while UART was active, nothing was being recorded from the ADC on the transmitting end, and nothing was being output by the DAC on the receiving end, resulting in the periodic dead zones shown in Figures 12 and 13. We used an oscilloscope to demonstrate some of these issues.

V. Recommended Improvements

Prior to the demonstration, we were only able to get the radio to work one way. That is, we were able to speak into one of the radios and the receive from the other, however, not vice versa. If we had more time for this project, we could have possibly debugged the issue and had the radios working both ways.

Additionally, with more time, the quality of the communication could be improved. As noted in the testing section, there was deadtime in the transmitted audio. This most likely resulted from software issues. Given more time, this deadtime problem could be solved. After that, the limits of the XBee module could be explored by increasing the data rate of the system. This would also increase the quality of the transmitted audio signal. If those problems could be addressed, then our walkie-talkies would have a greatly improved audio quality and overall functionality.

VI. Appendices

The following appendices were frequently referenced during the design and fabrication of this project, and have been included for the reader's reference. These are provided as soft copies only due to their length. They have been included as a zipped folder in the final submission.

Appendix A: Texas Instrument MSP430Fx2xx Family User Guide

Appendix B: XBee User Guide

Appendix C: Complete PCB project schematics

Appendix D: Sparkfun Microphone and Pre-Amp Module Schematic

Appendix E: Sparkfun Mono Speaker Amplifier Module Schematic

Appendix F: WalkieTalkie project code