DESIGN AND DEVELOPMENT OF WIRELESS TELEMEDICINE SYSTEMS

TEH CHEK MIN

A project report submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering (Hons.) Biomedical Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2011

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : TEH CHEK MIN

ID No. : 07UEB04649

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **DESIGN AND DEVELOPMENT OF WIRELESS TELEMEDICINE SYSTEMS** was prepared by **TEH CHEK MIN** has met the required standard for submission in partial fulfillment of the requirements for the award of Bachelor of Engineering (Hons.) Biomedical Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Mr. Teoh Chee Hooi

Date : _____

DESIGN AND DEVELOPMENT OF WIRELESS TELEMEDICINE SYSTEMS

ABSTRACT

As wireless technology advances, many applications went mobiles. It enabled medical processes to be done across geographic obstacles and distance. In the present time, home healthcare has been an important sector in telehealth. This involves telemonitoring system on the patient at home. In the project, a simple telemonitoring system is developed, consist of a temperature, humidity and audio sensor. The telemonitoring system is design for infant home care. This report is mainly for the audio sensor and its use in detecting infant's cry. In doing so, a Microelectromechanical system (MEMS) microphone is used. In the process of detecting crying sound, the amplitude and the frequency of the sound are compared to a threshold to determine the result. The result was 50 % efficacy. The wireless technology used is XBee, which is the most suitable method after reviewed along with Bluetooth and Wireless Local Area Network (WLAN) technology. A home server will be developed to receive and record the signal from the XBee transceiver installed. These component constituted a simple home telemonitoring system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATION

LIST OF SYMBOLS / ABBREVITATIONS

C	Capacitance, F
f	Frequency, Hz
F_{OSC}	Crystal frequency, Hz
Q	Charge, C
R	Resistance, Ω
T_{clock}	Period of one clock cycle, s
V	Voltage, V

API	Application Program Interface
ECM	Electret Condenser Microphone
GUI	Graphic User Interface
MCU	Microcontroller Unit
MEMS	Microelectromechanical System
PDA	Personal Digital Assistance
POSIX	Portable Operating System Interface for Unix
UART	Universal Asynchronous Receiver / Transmitter
USB	Universal Serial Bus
WLAN	Wireless Local Area Network

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

Telemedicine may be briefly defined as the healthcare carried out at a distance. At this period of time, the advancement of technology has enabled telemedicine implementation to able to dispose of cable. Further supported by many organisations, such as Intel / Cisco in their guide for wireless local area network (WLAN) healthcare deployment (Intel / Cisco, March 3, 2008), and also the ZigBee ® Alliance collaboration with the American Telemedicine Association (ATA) to promote the wireless health care (ZigBee ® Alliance, November 17, 2009), the telemedicine concept is no longer an innovation. In a study by Koch (2006), search terms often used to retrieve scientific literature from Medline database in January / February 2004 which related to home telehealth have recorded 5600 hits, and 72.2% (4045) of the searches were for home monitoring, and the related publication showed increment over time. Acknowledging the trend, this project is innovated to develop a prototype wireless homecare system for infant care and infant monitoring.

1.2 Aims and Objectives

The objective of this project is to develop a prototype wireless homecare system for infant care and infant monitoring. The whole system consist of three sensors, namely (1) temperature sensor, (2) humidity sensor, and (3) audio sensor. This project part is

focused on the audio sensor. The function of the temperature sensor is to sense the body temperature of an infant, while the humidity sensor is to sense the humidity level in the room.

The audio sensor is meant to detect the crying sound of an infant. The home server unit will periodically request the audio information of the sensor through the on site processing unit that physically connect to the sensor and done processing and monitoring on the home server unit. Once a crying noise is detected, the caregiver will be alarmed.

CHAPTER 2

LITERATURE REVIEW

2.1 Infant Crying

Infants cry the moment they were born as the sign of them taking over their own life. Through the first few months after born, infants have yet to develop speech-like voice (Hsu, Fogel, & Cooper, 2000) and thus unable to learn to speak. Crying is their only communication. In a study by Nikayama (2010), aside from biological reasons such as hunger, disturbance of sleep and discomfort, infants also cried for sociological reasons such getting attention and getting company. For instance, an infant cried more when the caregiver is out of the infant's visible and hearing range. This phenomena gave the idea to the present project as one of the method for a caregiver to monitor the crying status of the infant under care.

Many researches have been done on infant crying, most was about diagnosing the crying signal that may lead to a medical abnormalities. Várallyay, Illényi, Benyó, Farkas and Katona (2005) have been attempted to detect hearing disorder by acoustic features of the infant cry. In a study by Stifter (2005), Barr (2006), and Zeskind (2007), the crying behavior of an infant is related to the importance for psychosocial development in children. These research implied that the analysis of infant cries have the potential to diagnose the medical abnormalities. However, these studies were not reported as performed by remote analysis, nor by automated detection, recording, and analysis.

2.2 Wireless Technologies

There are many type of wireless technologies existed on the market nowadays. Complying with the project requirement, only a short range wireless communication area is required and this is used as the first concern in narrowing down the choices of the available wireless technologies in the market. The most used Bluetooth, ZigBee, and Wireless Local Area Network (WLAN) for homecare are reviewed.

2.2.1 Bluetooth

The “Bluetooth is a way for devices to wirelessly communicate over short distances” (Huang and Rudolph, 2007). The Bluetooth is regulated by Bluetooth Special Interest Group (SIG) and it is complied with IEEE 802.15.1 standards for Wireless Personal Area Network (WPAN), operate in 2.4 GHz, Industrial Scientific and Medical (ISM) band.

Relative to other versatile wireless technologies, Bluetooth is the lowest cost, simple and ubiquitous. The transceiver of Bluetooth is readily embedded in most gadgets in the market, such as PDAs, mobile phones and computers. The Bluetooth external transceiver, called a dongle, is widely available in market for years, for a relatively low price.

The connection between Bluetooth device is based on master-slave relationship, any device can be set to be a master or slave at any time (Kammer et al., 2002). The maximum communication one master device connects is seven slave devices at a time, forming a piconet. The transfer rate varies by their version:

Table 2.1: Transfer Rate of Various Version of Bluetooth

Version	Rate
1.2	1 Mbit/s
2.0 + EDR	3 Mbit/s
3.0 + HS	24 Mbit/s

The most conventional version used is bluetooth version 1.2. The 1 Mbps transfer rate is adequate for application such as document / audio / video transfers. The effective range of the communication will determine how far two device can communicate before they lost connection from the wireless system, while power system will determine the duration in which the device can operate before it's power source is recharged/replaced. The range and power output varies according to their class:

Table 2.2: Bluetooth Radio Power Classes

Power Class	Max Output Power	Range
Class 1	100 mW	100 meters+
Class 2	2.5 mW	10 meters
Class 3	1 mW	1 meter

For home-care application, Class 2 Bluetooth is preferable so that the communication valid around the range of the house. In an open area or place with less obstacle, the range could go more than the range indicated in Table 2.2.

All wireless communication systems nowadays are subjected to interference. With Bluetooth, interference may result of microwave (ovens, etc), thunderstorms, Wi-Fi or with other Bluetooth devices in that area, and hence Bluetooth technology should not be used for safety-critical applications where data absolutely must get through (Kammer et al., 2002). Nevertheless, a study made by Khoór, Nieberl, Fügedi and Kail (2001) used Bluetooth in Electrocardiography (ECG)-telemetry to compress and to convey the digital ECG data to their WEB-server. The performance of the implementation was not discussed, but the study successfully recorded various abnormalities in patients.

2.2.2 ZigBee

ZigBee is a wireless technology with low cost, low power, and short range specification. It comply with IEEE 802.15.4 standards for WPAN. The ZigBee

standard is published and maintained by a group of company called The ZigBee Alliance. In November 17, 2009, The ZigBee Alliance officially collaborated with American Telemedicine Association (ATA) to educate both health care professionals and consumers on wireless health care solutions. By this collaboration, wireless telemedicine, specifically with ZigBee, will be more popular among the medical practitioner and consumer. There is even ZigBee Health Care (ZHC) Public Application Profile free to download at http://www.zigbee.org/Products/DownloadZigBeeTechnicalDocuments.aspx to provide standard interfaces and device definitions to allow easy interoperability among telemedicine.

The ZigBee have transfer rate ranging of 20~250 Kbit/s. The communication range usually is between 10 and 75 meters, and may up to 1500 meters for ZigBee Pro, depending on surrounding condition. For normal ZigBee transceiver the output power of the radio can be as low as 1 mW (Cirronet, 2005). The low power consumption is the main advantage of the ZigBee in designing a battery-powered, embedded device. The network capabilities of ZigBee is self-organizing and self-healing dynamic mesh network based on ZigBee public standard, the network size may up to thousands of devices per network.

The ZigBee radiofrequency wave is also transmitted in the 2.4 GHz band, which means it will interfere with other users of this ISM band. However these consequences were expected by ZigBee Alliance and thus ZigBee is designed with the concept of coexistence in mind. The coexistence technique described in “ZigBee and Wireless Radio Frequency Coexistence (2007)” by the ZigBee Alliance is claimed to be effective in the coexistence. On the other hand, to get approved as an IEEE 802 wireless standards, the technology is required to attach with a Coexistence Assurance Document and implement it in a standards to ensure that all 802 wireless standards can operate and coexist in the same area.

A study by Vergari, Auteri, Corsi and Lamberti on ZigBee-based ECG telemonitoring showed the early research of the wireless technology in the field. The single lead ECG captured signals from a person, amplified and filtered in the embedded system before transmit it to a computer via ZigBee for further data processing. The signal was sampled at 500 *Hz,* encoded and sent through ZigBee

with the concern on the transfer rate because the transmission did not support high transfer rate. If too many bytes are sent to the radio module, the module will not be able to send all of them, resulting in data loss at the receiver end. This reflected to the project here that the audio captured from the infant will have difficulty to send through ZigBee to the base station for processing because the audio generally require much higher sampling rate and thus higher bitrate.

In another study by Srovnal and Penhaker (2006), ZigBee is used in homecare and health maintenance system using embedded system for the diagnostic and the maintenance planning in healthcare applications. In the study, ZigBee was chosen for their sensor network because it matched the main criteria for their project, which is bandwidth, range, energy, consumption, robustness, availability, usability and security. The sensor nets they built consist of ZigBee technology combined with ambient intelligence sensor technology.

The ZigBee have the lowest transfer rate among the wireless technologies reviewed, but due to its simplicity, low power consumption and fast respond time from the sleep mode (Andriana, Victor, & Ioan, 2008), it is more suitable for the wireless sensor network.

There are two types of ZigBee devices, (1) the full function device (FFD) and (2) the reduced function device (RFD) (Kinney, 2003). The advantage of FFD over RFD is the capabilities of the device to act as router to relay signal from other device to another, forming peer-to-peer and mesh networks, thus provide the ability to connect to devices at a greater extent. FFD device can also act as coordinator. RFD can only communicate with other device but cannot act as router. There are three types of topologies supported by ZigBee: star topology, peer-to-peer (mesh) topology, and cluster tree (Figure 2.1) (Kinney, 2003). RFD can only be used in the star topology.

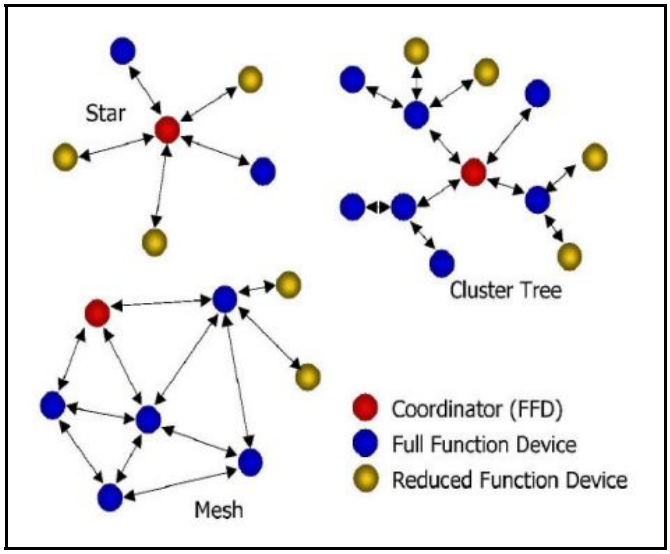


Figure 2.1: ZigBee Network Structure

The ZigBee device is not readily available in embedded form in daily life gadgets (phones, laptop, PDA etc) in the current time. They must be independently provided as a functional system for some applications. However there are ZigBee products and have sufficient amount of development tools readily for purchase in Malaysia; there are ZigBee resources available here for development.

2.2.3 WLAN

Wireless Local Area Network (WLAN) is widely used for conventional internet connection from devices to an access point. It comply with IEEE 802.11a/b/c/g/n standards and also operates in ISM bands. WLAN offers higher data transfer rate and may up to maximum 300 Mbit/s for 802.11n (the power consumption also high, in the range of a few Watts). The range usually goes around 100 meters outdoor and around 35 meters indoor, however, by forming access points, the range could be extended. WLAN establishes connection through IP address recognition, and security could be enhanced by MAC address recognition.

Like others RF that communicate in the ISM band, WLAN also subjected to the similar interference. However the 802.11a and 802.11n can operate in 5-GHz, providing high data throughput, reduced interference from non-802.11 wireless devices and generally provide improved performance in RF-reflective environments, which is a better option for high risk monitoring medical devices.

A design by Kugean, Krishnan, Chutatape et al. showed the possibility of implementing mobile telemedicine using WLAN. Their design was meant to provide mobility, reliable and cost-effective telemedicine network. The WLAN have high transfer rate and it was able to support all of the telemedicine requirement including videoconferencing. Noted in the paper, occasional interference was inevitable.

2.3 Radiation Concern

This project aims is to develop a device for infant application that involves Radio Frequency (RF) transmission. The RF is a type of electromagnetic (EM) wave and it contains energy. The energy of the wave is depend on the radio emitter. Bluetooth and ZigBee both have maximum radio power output around 50 mW. The duration of time of exposure to the power is equal to the energy delivered by the radio. In the project, it is important that the EM wave have no adverse effect on infants.

The wireless technologies reviewed is under the control of Code of Federal Regulation (United States), Title 47, Part 15 (47 CFR 15). The emission of the wireless technologies is within the tolerable range for human safety. However currently there was no adequate researches on the subject.

2.4 Microphones

Audio capturing is needed for infant crying detection and is done by a microphone. A microphone is a transducer that can convert the sound mechanical signal into

electrical signal. this project is aimed at designing a portable device and thus compactness is one f the design factors. The market existing compact microphone is Electret Condenser Microphone (ECM) (Figure 2.2) (Hosiden) and Microelectromechanical system (MEMS) microphone (Figure 2.3) (Justin, 2009). ECM have been used in many daily gadgets such as mobile phones, headset, digital camera and laptop (Nielsen and Fürst).

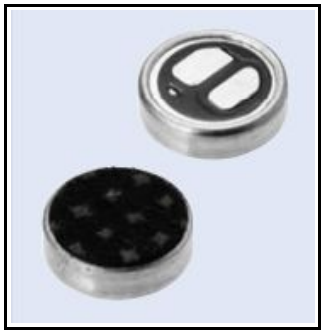


Figure 2.2: Electret Condenser Microphone (ECM)

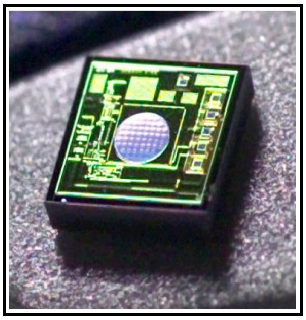


Figure 2.3: Microelectromechanical System (MEMS) Microphone

ECM is a capacitance microphone. Figure 2.4 shows the cross-section of a typical condenser microphone (MediaCollege.com). The front plate and the back plate forms a capacitance. The front plate, which is also a diaphragm, will vibrate upon receiving sound waves. The back plate is in fixed position and is charged by a battery. Upon vibrating occurs at the diaphragm, the capacitance of the capacitor will change, resulting in a change of voltage, which is the output audio signal. The ECM differ from normal condenser microphone in such as way that the battery does not exist, but being replaced by a fixed charge material that could discharge for years

(Nielsen and Fürst).

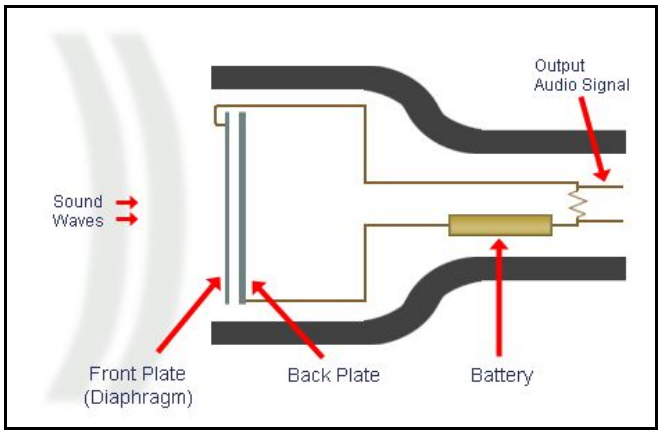


Figure 2.4: Cross-section of A Typical Condenser Microphone

The general relationship between charge, Q, voltage, V, and capacitance, C, is given in equation 2.1. The equation of the change of the V resulted from the change of C is given by equation 2.2. The ΔV is the audio output signal. Note that Q is constant, supplying a constant voltage over the plates. From equation 2.1, when C is small, then V will be higher. When the front plate vibrated, the capacitance changed.

Through a threshold at about 50 *Hz* (hardware dependent), the capacitance is almost constant and the voltage will also changed. Thus, The changes of voltage will reflects in the changes of the amplitude of the output signal. This can be use to detect the intensity of the sound, or how loud the sound was.

$$V = \frac{Q}{C} \tag{2.1}$$

$$\Delta V \simeq -V\left[\frac{\Delta C}{C}\right] \tag{2.2}$$

where

V = voltage, V

ΔV = change of voltage, V

C = capacitance, F

ΔC = change of capacitance, F

Q = charge, C

MEMS microphone works on the same concept as the ECM, except that the components are built onto a single chip using Complementary Metal-Oxide Semiconductor (CMOS) technology, while ECM is assembled from discrete parts (Justin, 2009). MEMS microphone is newer technology compare to ECM and have the same range of application to ECM. MEMS microphone can be found in two types: (1) Analogue output type, and (2) Digital output type. Digital output is in the form of Pulse-Density Modulation (PDM) and readily integrated into digital device. The digital MEMS microphone also immune to Radio Frequency (RF) interference. Figure 2.5 illustrate the cross-section of a typical MEMS microphone.

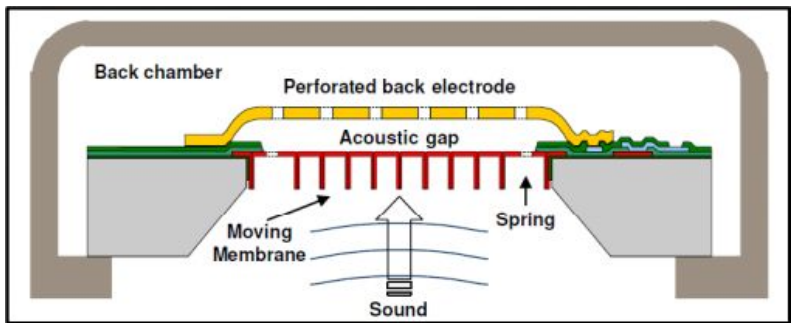


Figure 2.5: Cross-section View of A Typical MEMS Microphone

2.5 Crying Detection

There were several automated infant crying detection researches conducted until present. One of the study by Reyes-Galaviz and Reyes-Garcia (2004) used neural networks to automatically recognize infant cry and also classifies them into three kinds of cries, namely: normal, deaf, and asphyxiating infants cry. However to achieve that, a high end processor (and using Matlab®) is required to process the recorded signal.

The infant crying sound can be regarded as a form of music melody (Várallyay, 2007). In analysis of the crying sound, the audio sample is grouped in samples of 50 *ms* and Fast-Fourier Transform (FFT) is performed on the sample and the fundamental frequency is evaluated (Várallyay, 2007) (Manfredi, Bocchi, & Orlandi, 2009). Although the fundamental frequency changes during the crying audio detection can be use to recognise infant cry, however the processing algorithm is difficult in implementation for embedded system. This project uses the frequency threshold as the crying detection.

Infant crying frequency centered around 3500 *Hz*, differ between each individual and ranging from 1000 to 6000 *Hz* (Goodman, May 4, 2006). Frequency threshold method was implemented for the detection (Chau-Kai, 1992) in the commercial product on the market. The method is simple to implement and require relatively less processing, thus provides the real-time processing and embedded system application realisable.

CHAPTER 3

METHODOLOGY

3.1 Wireless Sensors Communication

3.1.1 Network Topology

This project used XBee (XBee 1mW Wire Antenna, Digi International) for the wireless communication. The peer-to-peer topology will be used to form a mesh network between the sensors and the home server, although there are only two nodes, this will increase the versatility of the node. The sensors will shares the common transceiver, while the other transceiver will be connected to a home server, which is a computer. The illustration is shown in Figure 3.1. The remote station that accumulate the data from the sensors are called base station. It performs ADC for the analog output from sensors and then send it to home server.

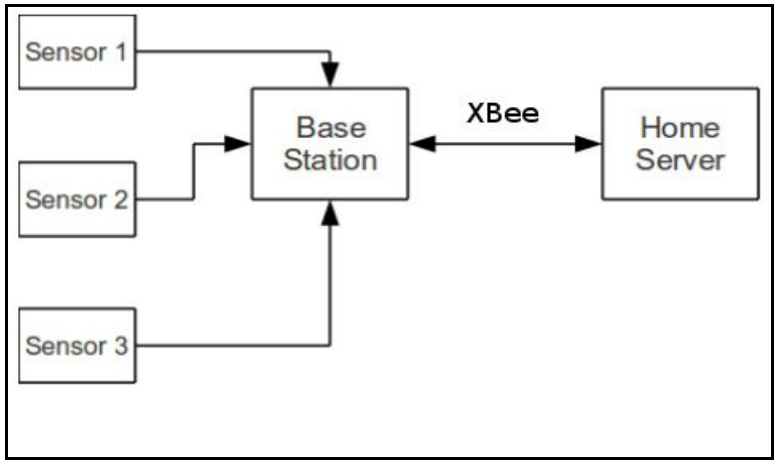


Figure 3.1: Project Network Topology

3.1.2 Communication Protocol

The XBee modules was able to communicate through 2 types of mode: 1) transparent mode, and 2) application program interface (API) mode. The transparent mode is the simplest way of communicating, including allowing the use of AT command to configure the properties of the modules. In transparent mode, the data are sent just as if they are sent through a serial port. However the destination address and source address of communicating devices must match and be configured prior to communicating. This mode however, only set for one pair of communicating devices because the destination address are fixed.

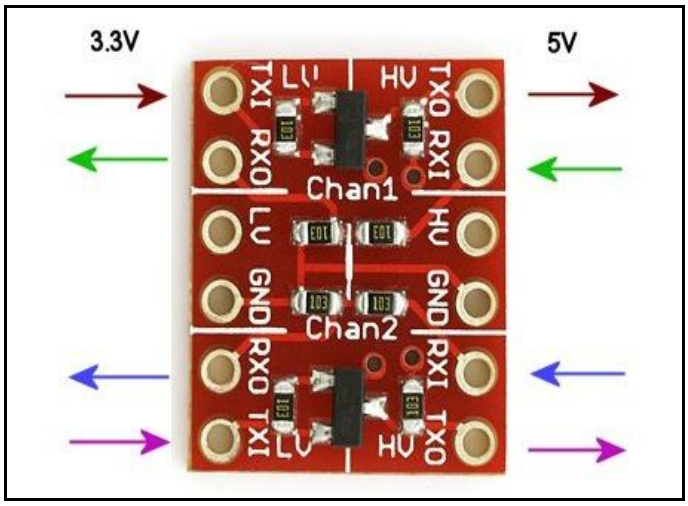
The API mode allows sending of data to destination with different addresses, depending on which destination address are put in the packet and regardless of the pre-configured destination address set in the register. Other than that, API packet also can be used to apply remote command on other XBee modem. To make use of this ability, the project has implemented the API communication mode for the XBee modules.

The communication with hosts (home server, microcontroller) was using Universal Asynchronous Receiver Transmitter (UART) protocol. The XBee USB dongle (Cytron SKXBEE, XBee Startup Kit) (Figure 3.2) is used to connect the with the home server. The dongle is based on UART protocol which converted to Universal Serial Bus (USB) data through an Integrated Circuit (IC) chip (FTDI FT232) which convert between RS232 and USB data. With this dongle, the data received from the XBee module will output the data to computer through a USB port which will act as a virtual serial port.

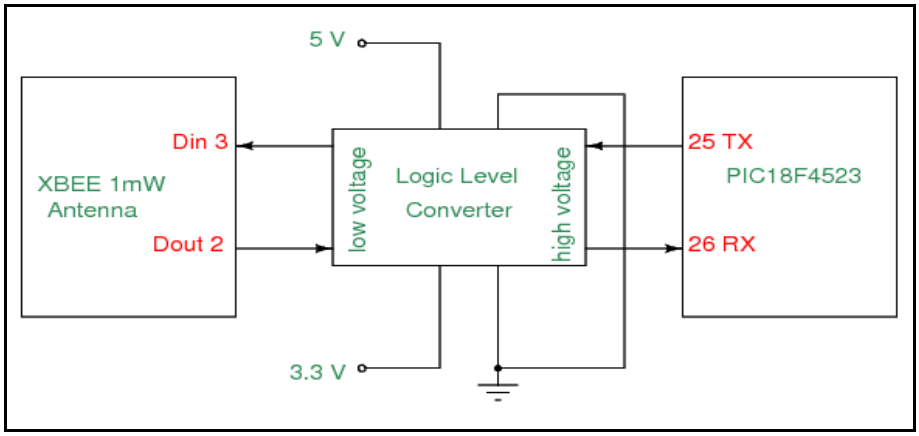
Figure 3.2: A SKXBee

The communication with the microcontroller require a logic level converter because the voltage required by these ICs are not same. The microcontroller used and transceive 5 Volt transistor-transistor logic (TTL) data while the XBee used 3.3 Volt. A logic level converter (Sparkfun Logic Level Converter) was used to convert between these voltage levels so the communication can be done without spoiling the IC. The communication between the XBee and the microcontroller used UART protocol. The connection between these IC and the logic level converter are as shown in Figure 3.3.

The UART protocol between XBee and hosts was configured to have 19200 baud, 8 bits data, no flow control and 1 stop bit. These parameter have to be matched between UART devices in order to communicate with their host.



(a)



(b)

Figure 3.3: (a) Sparkfun Logic Level Converter, (b) Connection Between XBee, Logic Level Converter and Microcontroller

3.1.3 API Packet Structure

The general API packet structure is shown in Figure 3.4 below. The start delimiter is a fixed symbol (with hex code 0x7E) for the start of the API packet. The MSB and LSB combined are a two-bytes number indicating the number of bytes in the frame data. In the frame data, there separated into two parts: 1) API identifier, and 2)

identifier-specific data. The API identifier will let the module know the purpose of the packet. For instance, API identifier with value 0x01 is for transmit request (16-bit address) packet, 0x89 is for transmit status packet, 0x81 is for receive packet (16-bit address) etc. The identifier-specific data, as it was named, is the data required based on the API identifier that was given. For example, if the packet is transmit request packet, the data will have destination address, frame identifier, option, and the data to be transmit out. The last byte is the checksum byte. It was an important byte to ensure there was no error in the sent data. For this module, the checksum is define by equation 3.1.

$$Checksum = 0xFF - sum\ of\ all\ bytes\ in\ the\ Frame\ Data \quad (3.1)$$

In order to request microphone information from the microcontroller, the packet as shown in Figure 3.6 is sent to the XBee module attached to the home server (host) which is a personal computer from the computer itself. The transmit request used 16-bits address and have the structure as shown in Figure 3.5. It can be seen that the request packet was a transmit request packet (16-bit address) with 0x52 as frame identifier, 0x1010 as the destination address and the character 'M' as the content. When the packet received and processed by the XBee module and when the checksum is correct, the modem then will send the data to the other XBee module attached to the microcontroller (another host), which will respond by performing the task required and then assemble a packet and then send the packet back to the home server following the similar path. The packet respond to the microphone information request packet will have the data structure shown in Figure 3.7 and Figure 3.8, total of 14 bytes. The receive packet was not of a constant value, because its content depends on the Receive Signal Strength Indicator (RSSI) which was 0x28 in the figure and the content of the data, and thus the checksum too will be varied. The 0x4D and the 4 consecutive 0x00 are the content of the data, containing the microphone information that requested by the home server.

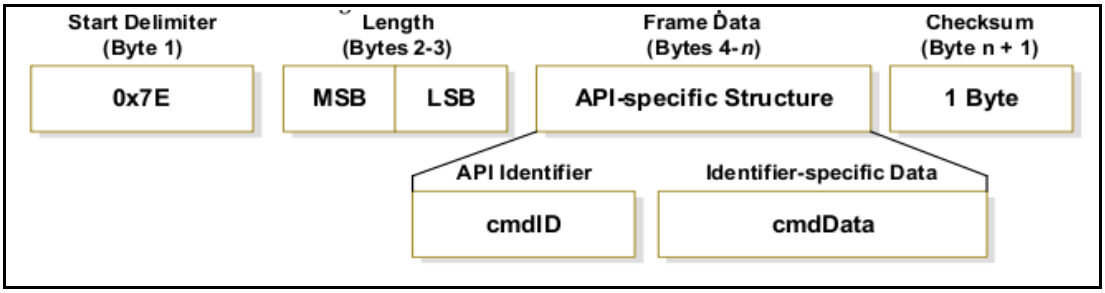


Figure 3.4: The General Structure of an API Packet

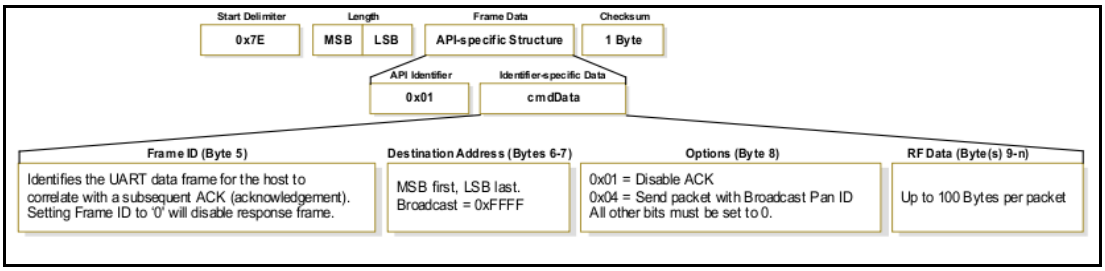


Figure 3.5: The API Structure of Transmit Request Packet (16-bit Address)

7E	00	06	01	52	10	10	04	4D	3B

Figure 3.6: The Packet to Request Microphone Information Sent by Home Server

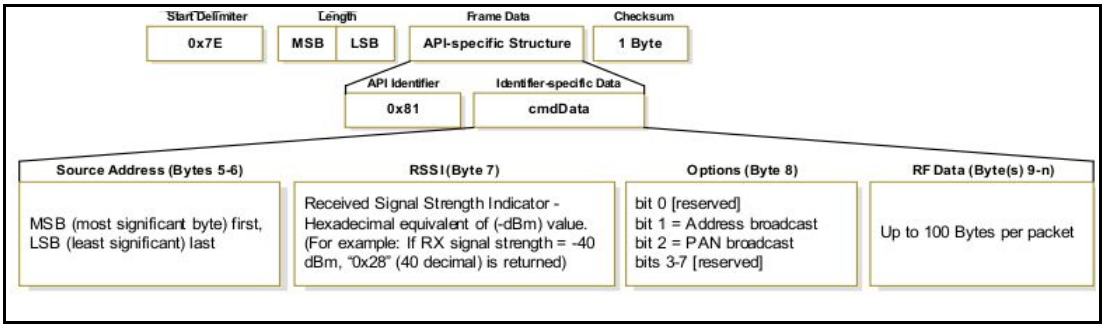


Figure 3.7: The API Structure of Received Packet (16-bit Address)

7E	00	0A	81	10	10	28	00	4D	00	00	00	00	E9

Figure 3.8: The Structure of Packet Reply by Microcontroller in Response to the Request

The request for microphone information will retrieve two parameters: 1) the amplitude of the microphone, and 2) the frequency of the microphone. These information were expressed in digital format as the result of analog-to-digital (ADC) conversion process. The interval of time for each request of 0.2 seconds, which means 5 inquiry per second. By unwrapping the API frame, the request basically send a character 'M', and the reply will send 'M' and followed by 4 information bytes. The first two bytes was the ADC result of the amplitude of noise, and the last two was the ADC result of the frequency.

When a transmit request data has been successfully sent out through the XBee, the XBee will reply the host with a transmit status packet, shown in Figure 3.9. This packet can be use to inform the host about the status of packet delivery. If the packet should not be missed, the host can use the status to trigger resend if the delivery was not success. However, the usage of this packet was no implemented in the project. The transmit status packet received upon a success or non-success delivery will just be ignored.

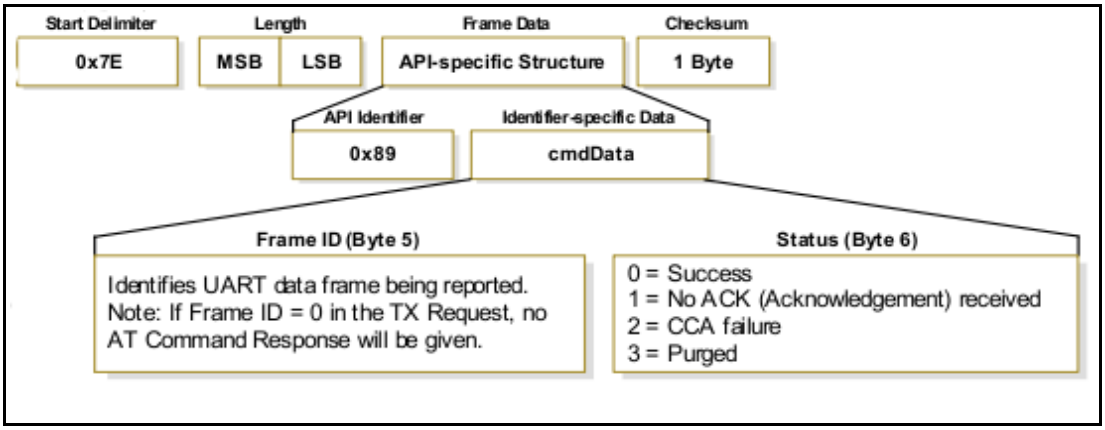


Figure 3.9: The API Structure of Transmit Status Packet

3.1.4 Utilised Programming Software & Tools

The firmware in the standalone XBee module and the XBee module on the USB dongle was upgraded before the XBee is used. The program used to configure and upgrade the XBee properties and firmware was Digi's X-CTU software. The

firmware upgrade and configuration steps for XBee USB dongle were easier than that of standalone XBee module. The reason was that XBee USB dongle have readily plug and play design so that upon plugged into the a USB port, the device will be detected and ready to be use.

Prior to use the X-CTU software with the attached XBee USB dongle, some configuration and additional software installation have to be done because the operating system used in this project was Linux (Ubuntu) operating system and the X-CTU software was made to run in Microsoft Windows operating system. First WINE Is Not an Emulator (WINE) software is installed so that the Linux can run some basic Windows application (Lizard43, 2008). The X-CTU installer was then used to install the software inside WINE. The X-CTU can now run through WINE, but the com port will not be detected. To make this works, a link of device file (where the XBee USB dongle is attached) has to be created in the *~/.wine/dosdevices* directory. These can be explained in a few steps:

i) The XBee USB dongle is inserted in a USB port.
ii) From terminal, run "dmesg | tail" and identify where the device is attached to. For example, */dev/ttyUSB0*.
iii) A soft link of */dev/ttyUSB0* is created in *~/.wine/dosdevices* directory using command:

ln -s /dev/ttyUSB0 ~/.wine/dosdevices/com8

In the steps the soft link created is named com8, but it can be any com number. After these steps, the XBee USB dongle will now be recognized by the X-CTU software. The X-CTU software is now switched on and the com number that was assigned is inserted in the text area within "Add User Com Port" group box (Figure 3.10). To communicate with the dongle, proper communication configuration need to be set. The default baud rate is 9600, flow control is NONE, data bits is 8, parity is NONE and stop bits is 1. The "Test / Query" button is then clicked and the software will try to establish a connection with the dongle. If the configuration was correct, the connection will success and the windows will be as shown in Figure 3.11.

To configure the modem, modem configuration tab in the X-CTU software is used (Figure 3.12). To know the current configuration, the “read” button is clicked. The parameters of the modem will be shown. To change the value of the parameter, the name of the parameter shown is clicked and the new value is entered. The “write” button is then clicked to write the new value into the register of the modem.

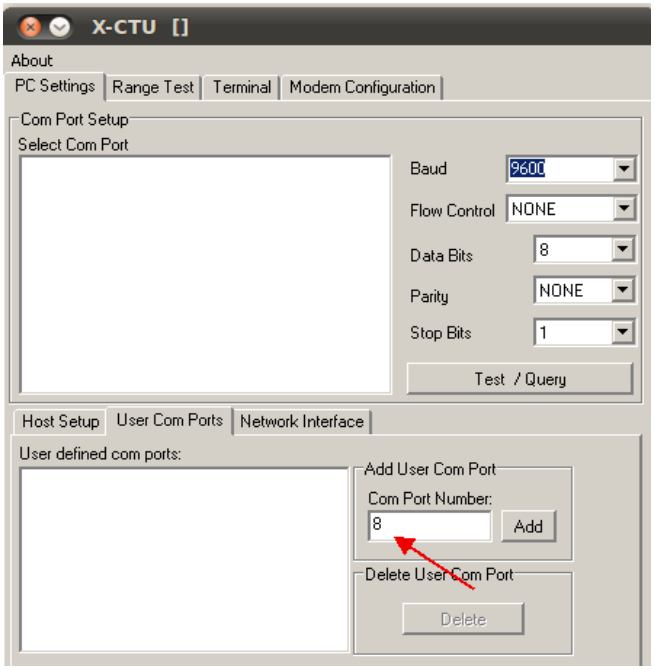


Figure 3.10: Opening a Com Port with X-CTU Software

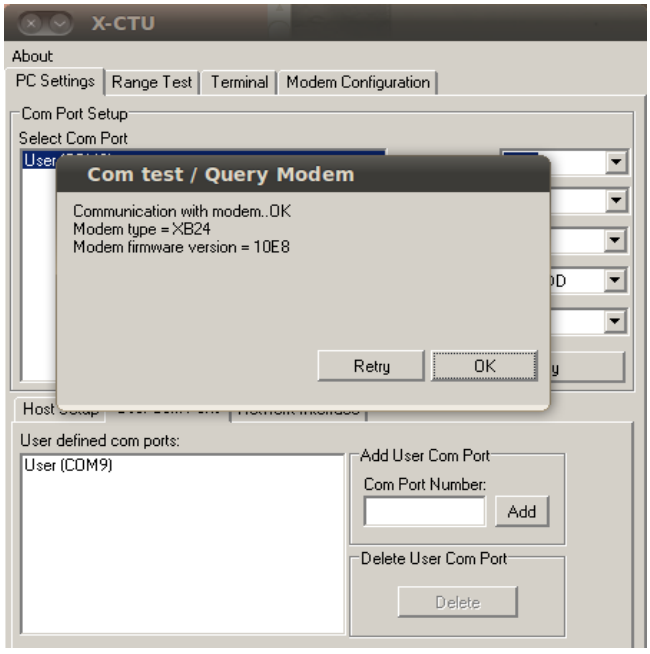


Figure 3.11: Success Test / Query Modem on Com Port

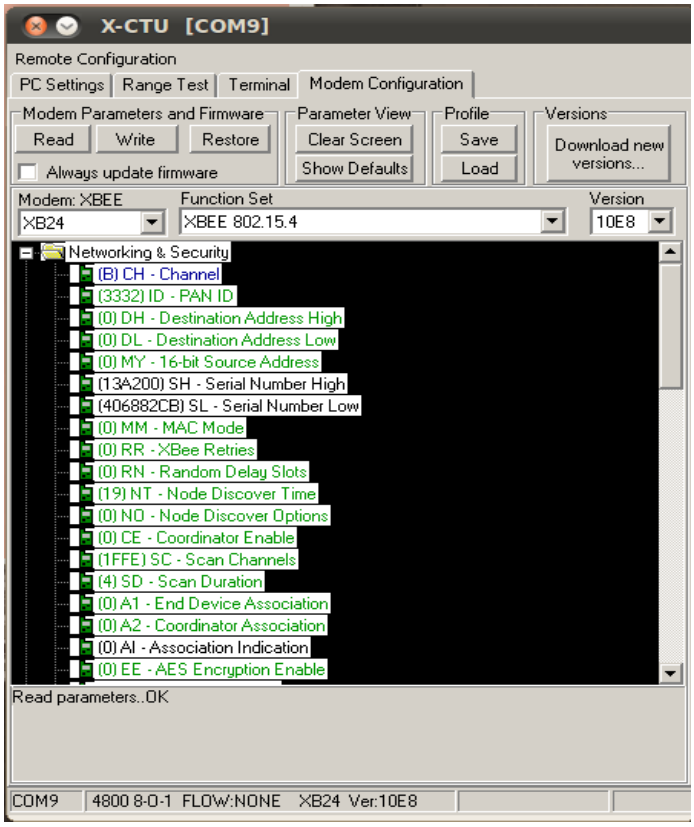


Figure 3.12: Modem Configuration Tab in X-CTU Software

The configuration set were the Channel (0x0B), Destination Address Low (0x1010), 16-bit Source Address (0x0101), Interface Data Rate (19200), Packetization Timeout (0) and API Enable (API ENABLED). After these changes are written into the modem, the connection need to be reestablished with 19200 baud rate.

The firmware upgrade is also done in the modem configuration tab. In the tab, in the "Version" group box, the "Download new versions..." button is clicked and then the new version firmware will be automatically downloaded and can be write into the modem. The firmware version used between two communicating modems must be the same to avoid unexpected error. The firmware version used here is version 10E8.

To upgrade and configure the XBee modem module, extra components are needed. Serial port communication was used to upgrade the firmware, through a hex inverter (HD74HC04, Hitachi) (Faludi, 2011). The connection is shown in Figure

3.13. The serial port communication was done using a FTDI chip based usbserial converter so that the protocol can work as virtual serial port over USB protocol. The hex inverter was used to invert the digital signal between 5.0 Volts and 3.3 Volts. Thus, the hex inverter was powered by 3.3 V supply. Then the similar steps used with configuring and upgrading modem firmware of XBee USB dongle with X-CTU can be applied to this standalone XBee module.

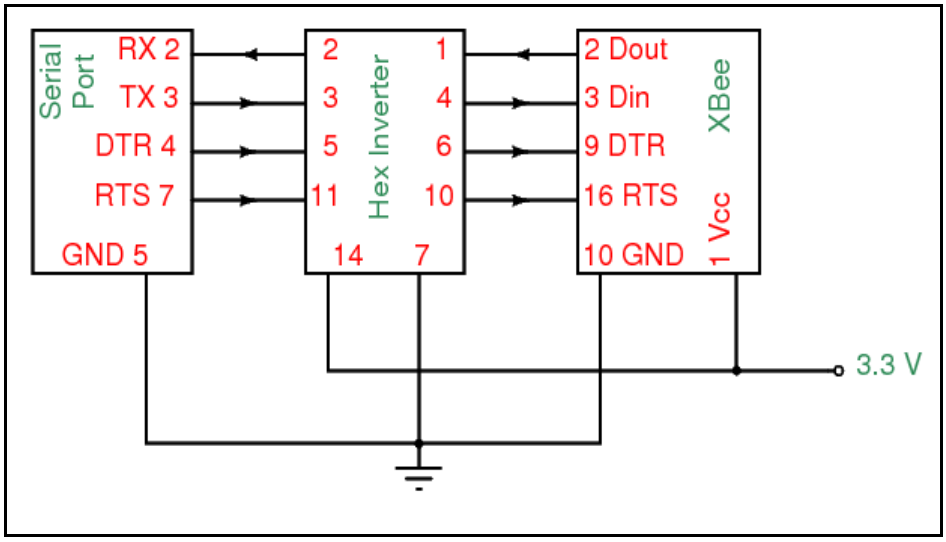


Figure 3.13: Connection to Configure and Upgrade Firmware on Standalone XBee Module

Other than using hex inverter to configure and upgrade the standalone XBee modem, the API packet can be sent by the XBee USB dongle to execute command on the standalone modem.

3.1.5 Modem Configuration

Several configuration were done on the XBee modems so that they can establish connection with their host. The XBee modem to modem communication did not affected by these configuration. The configuration were baud rate, data bits, parity bit, flow control, stop bits, channel, addresses, API enable, and packetization timeout. The configuration that were common to both modem is shown in Table 3.1 (a) while

the different one is shown in Table 3.1 (b).

Table 3.1: (a) Common Configuration on Both Modem, (b) Different Configuration on Both Modem

(a)

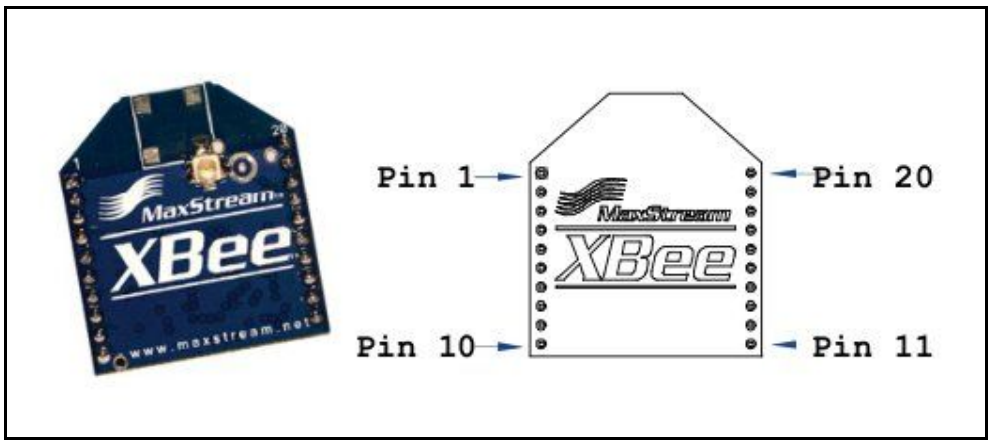
Parameter	Value
Baud rate	19200
Data bits	8 bits
Parity bits	None
Stop bits	1 bit
Channel	B
API Enable	API Enabled (no escape character)
Packetization timeout	0 second
Flow control	None

(b)

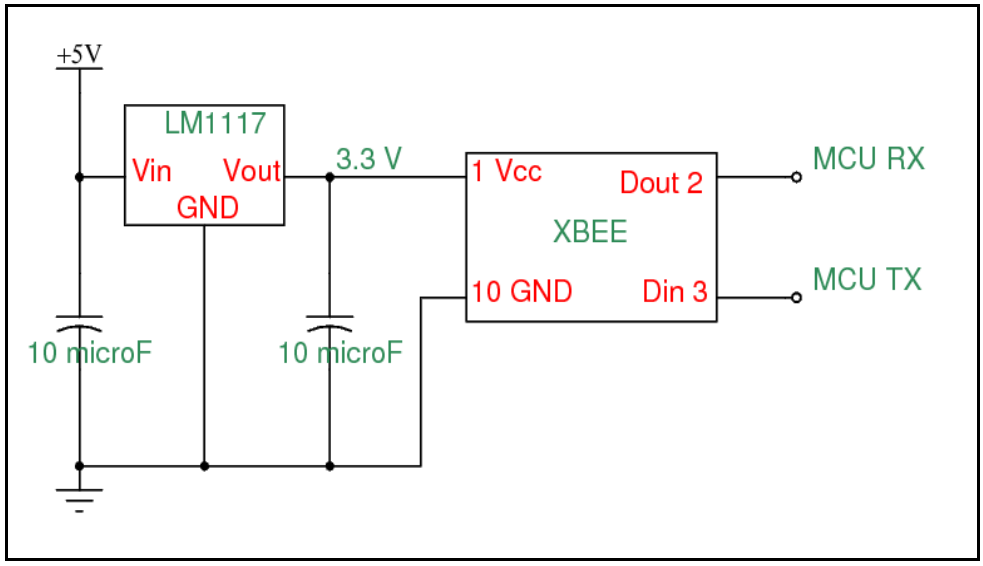
Home Server side		Base Station side	
Parameter	**Value**	**Parameter**	**Value**
DL	0x1010	DL	0x0101
MY	0x0101	MY	0x1010

3.1.6 Standalone XBee Module Connection

The XBee USB dongle did not need a manual circuit configuration to power up the modem because it was all prepared by the manufacturer. However the standalone XBee module need to be manually power up to make it function. The connection for the modem is shown in Figure 3.14 (b). The modem and its circuit works in 3.3 V range, so a LM1117 (LM1117, National Semiconductor Corp.) 3.3 V voltage regulator was used to regulate 3.3 V in the circuit. It can be seen that 5.0 V voltage is fed into the LM1117 to produce 3.3 V regulated voltage.



(a)



(b)

Figure 3.14: (a) The Standalone XBee Module and (b) The Connection for Standalone XBee Module

3.2 Audio Capture

Audio capture is done by a microphone. The microphone used in the project to detect infant cry is an analog MEMS microphone (ADMP401 (Analog Devices) breakout board, Sparkfun). Because the crying detection used the amplitude and the frequency

of the noise from which the microphone captured, the analog signal output from the microphone is first modified before feed into the ADC of the microcontroller.

The signal output from the microphone was an AC signal with an offset DC voltage. The amplitude of the peak-to-peak voltage (V_{pp}) varied with the intensity of the noise. Therefore, the V_{pp} is attached parallel with a capacitor to reduce the ripple (Figure 3.15). The rectifier is not used to rectify the voltage because the amplitude of the voltage drop across the diode is significantly big relative to the magnitude of the other signal in the circuit.

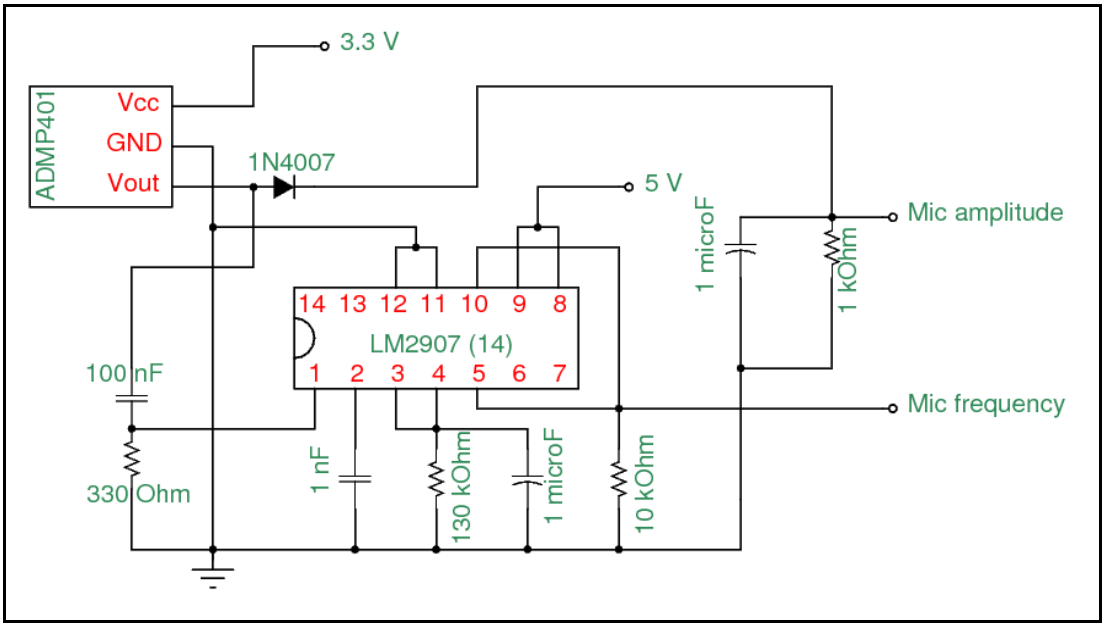


Figure 3.15: The Connection of the Microphone

To convert the frequency of the noise from the output of the microphone, an external IC is used. The LM2907 (LM2907, 14 pins, National Semiconductor) IC is a frequency to voltage (F / V) converter. The connection from microphone output to LM2907 is shown in Figure 3.15. A capacitor is used at the output of the microphone before feeding into LM2907 to block the DC offset voltage of the signal. The selection of capacitors and resistor is based on equation 3.2 (National Semiconductor, 2008).

$$V_{\text{out}} = f_{\text{in}} \times V_{\text{cc}} \times R_1 \times C_1 \qquad (3.2)$$

where

V_{out} = the output voltage from LM2907, in V

f_{in} = the frequency of the input, in Hz

V_{cc} = the supply voltage, in V

R_1 = the resistance of the resistor, in Ω

C_1 = the capacitance of the timing capacitor, in F

In Figure 3.12, the capacitor (1 nF) connected to pin 2 of the is C_1, the resistor (130 $kOhm$) connected to pin 3 and pin 4 is R_1. The supply voltage to this IC was 5 V, and the desired maximum detectable frequency and voltage output is 5 kHz and 3.3 V respectively. By using the LM2907, the frequency of the microphone output can be converted into voltage, which can be converted to digital signal by the ADC of the microcontroller.

3.3 Microcontroller Unit

The microcontroller unit or MCU chosen for the project is PIC18F4523 (PIC18F4523, Microchip Technology Inc.). The main reason for using the IC was because the 12-bit ADC function that it offers compares to other IC in the same family that offers 10-bit ADC. The 12-bit ADC provides higher resolution detection. The full range of the conversion have 4096 steps. Since the analog signal from the microphone that feed into the ADC have voltage range slightly lower than 3.3 V, the positive reference voltage, V_{ref}+ of the ADC was 3.3 V. The negative reference voltage, V_{ref}- was zero or ground. This allowed each step to correspond to 0.806 mV. The connection of the MCU is shown in Figure 3.16. The MCU also provided with 13 channels of the ADC, allowing multiple sensors to attach to a single MCU.

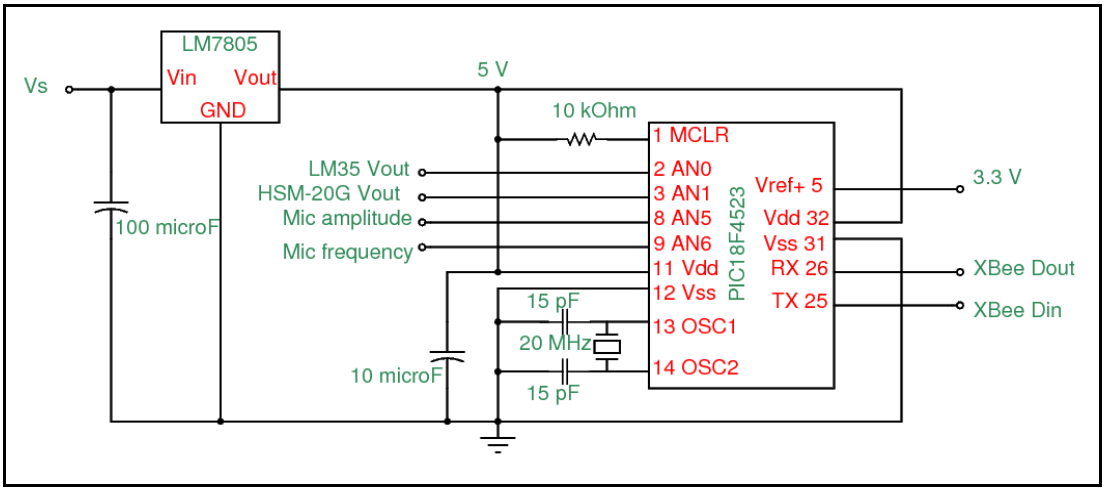


Figure 3.16: The Connection for PIC18F4523

From AC adapter, the voltage supply, V_s was feed into the LM7805 (LM7805, National Semicondutor Corp.) 5.0 V voltage regulator before supplying to the whole MCU circuit that uses 5.0 V (except the positive reference voltage at pin 5). A 20 *MHz* crystal was used to derive system clock, which is given by equation 3.3.

$$T_{\text{clock}} = \frac{4}{F_{\text{OSC}}} \qquad (3.3)$$

where

T_{clock} = period of clock cycle, in second

F_{osc} = frequency of the crystal, in *Hz*

Therefore one clock cycle took 2 x 10^{-7} seconds. The speed of the system clock determine how fast the instruction within the MCU can be execute. Using 20 *MHz* to derive system clock, the MCU is ensured to be able to perform well to manage the sensors attached to it and also the UART function. The analog output from microphone amplitude and microphone frequency are fed to ADC channel 5 (AN5, pin 8) and channel 6 (AN6, pin 9) respectively.

3.4 Power Supply

The full circuit consists of MCU circuit and XBee module circuit require proper design to manage the supply voltage. As mentioned in the previous section, XBee module used 3.3 V voltage supply and MCU used 5.0 V. Two voltage regulator involved were LM7805 (5 Volts regulator) and LM1117 (3.3 Volts regulator). LM7805 was connected directly to the AC adapter (Deluxe Universal AC/DC Adapter, NS) while LM1117 was powered by the 5.0 V voltage from LM7805 (Figure 3.17). This configuration is called cascade configuration. The purpose of the configuration was to reduce heat generated by the voltage regulators. The capacitors installed to the input and output of the voltage regulator were to function as filter to reduce ripple voltage from supply. The capacitor (100 μF) at the input supply have bigger value to prevent backflow voltage when the power supply was off and the other capacitors were discharging.

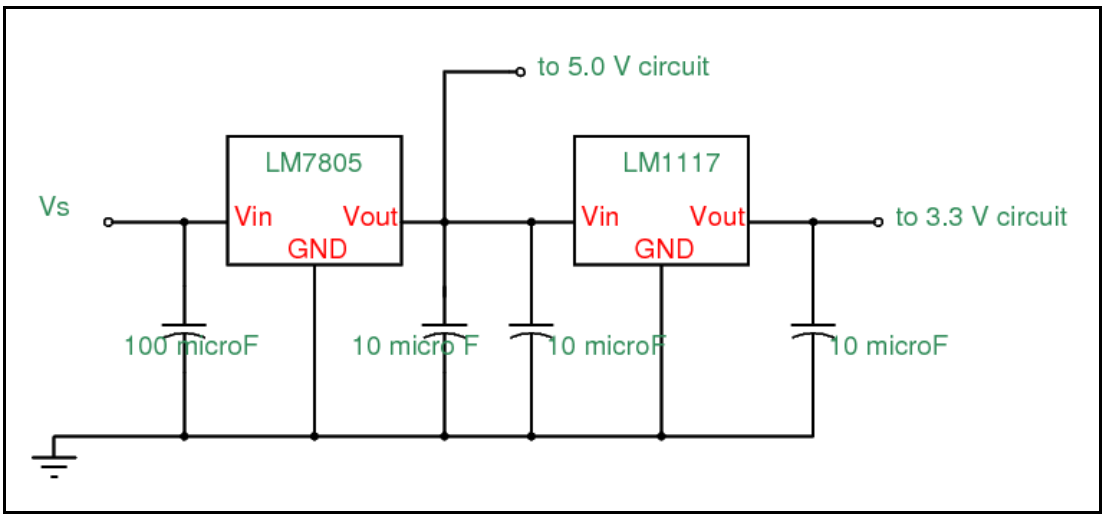


Figure 3.17: Cascade Configuration for Supply Voltage

The full circuit diagram is attached in Appendix C. The pictures of circuit connection are attached in Appendix D.

3.5 Software & Program Design

3.5.1 Programming for MCU

The program that was loaded into the MCU was written in C programming language and compiled with Microchip C18 Compiler (Student Version). The project was build within the Microchip MPLAB Integrated Development Environment (IDE). After obtaining the hex file, the program was loaded into the MCU using PK2CMD (Pickit 2 Command) software through a Pickit 2 compatible programmer (UIC00A Programmer, Cytron Technology Inc.). The MPLAB, C18 Compiler and PK2CMD software can be downloaded at http://www.microchip.com.

3.5.1.1 ADC Programming

To use the ADC peripheral function offered by the MCU, the registers to activate and setup the ADC were configured. Three registers to configure the ADC: 1) ADCON0, 2) ADCON1, and 3) ADCON2. ADCON0 was set to 0x00 to turn off the ADC module and disable the ADC before the program start. ADCON1 was set to 0x17 to enable the use of positive reference voltage at pin 5, and to enable analog input from ADC channel 0 to channel 7 (AN0 to AN7). ADCON2 was set to 0xBE to configure the right justified alignment of the conversion result in ADRESH and ADRESL register, set the ADC acquisition time to 20 T_{AD}, and clock derivation from $F_{OSC}/64$.

The 20 T_{AD} was needed to ensure the time is enough for complete analogue input level acquisition. This was because the start conversion instruction was given after enabling the ADC module, and at least 3 T_{AD} was needed to begin acquisition (Microchip, 2007). Each ADC bit require 1 T_{AD} for acquisition, so 12-bit ADC requires 12 T_{AD}. When the conversion ended, it requires another 1 T_{AD}. Summing up the minimum T_{AD} required was 16 T_{AD}. To be safe and avoid error, 20 T_{AD} was used instead. The part of the code:

```
ADCON0 = 0x00;
ADCON1 = 0x17;
ADCON2 = 0xBE;
```

The prescaler $F_{OSC}/64$ was needed to ensure that the T_{AD} that derived was bigger than the typical minimum T_{AD} needed which is 2.5 μs. This prescaler produced 3.2 μs for one T_{AD} and therefore acceptable. Smaller value, which is $F_{OSC}/32$ with 1.6 μs of T_{AD} was not acceptable because it was lesser then the minimum T_{AD} needed.

There was 4 channels of ADC used on the MCU, although only two channels (AN5 and AN6) were for this project. To change the channel, ADCON0 register must be changed. To set AN5, ADCON0 was set to 0x14, and 0x18 for AN6. After changing this register, the ADC is switched on, and the GO bits in the ADCON0 register is set so that the conversion is started. For example, to change to AN5 and start conversion, the code will look like:

```
ADCON0 = 0x14;      /* set the channel */
ADCON0bits.ADON = 1;      /* switch on the ADC module */
ADCON0bits.GO = 1;          /* start conversion */
```

The ADC conversion was waited until it finished before the result is collected. To wait the ADC conversion, a loop was polling on the DONE bit in the ADCON0 register. When the DONE bit is zero, the conversion is done and the result can be read from ADRESH and ADRESL register. The code for the routine:

```
while(ADCON0bits.DONE == 1);    /* loop here if DONE is 1 */
resultHighByte = ADRESH;            /* storing result once conversion done */
resultLowByte = ADRESL;
```

After the result was stored, the ADC module was then turned off. This was done was setting zero to ADON bit in ADCON0 register. The code:

```
ADCON0bits.ADON = 0;      /* turn off the ADC module */
```

3.5.1.2 UART & XBee Module Interface Programming

The UART peripheral on the MCU communicate with XBee modem through a logic level converter. The MCU acted as a host while the modem was a slave. The UART setting was set to on both devices to allow them communicate successfully. The setting were baud rate (19200 baud), and data bits (8 bits). The configuration of the modem was discussed earlier. Four registers on the MCU were needed to configure the UART: 1) TXSTA, 2) RCSTA, 3) BAUDCON, and 4) SPBRG. The TXSTA was set with 0x20, to switch on the UART transmitter in asynchronous mode, and RCSTA was set with 0x90, to switch on the UART receiver and the master serial port enable bit. The BRG16 and ABDEN bits in BAUDCON was switched off by assigning zero to the bit to set the baud rate generator (BRG) to be 8 bits, and disable the auto baud rate detection function. Then the SPBRG was set to decimal value 15 for desired baud rate of 19200. The equation for calculating the SPBRG value was given in equation 3.4.

$$SPBRG = ((F_{OSC} / Desired\ Baud\ Rate)/64) - 1 \quad (3.4)$$

The error percentage for the transmission can be calculated by substitute back the SPBRG in the equation and calculate the baud rate generated, and then use the baud rate to compare with the desired baud rate to calculate the error percentage. For the UART with SPBRG of 15, the baud rate generated was 19531.25, which have 1.73 % of error compared to the original 19200 baud.

To send a byte of data, the byte just need to be assigned into TXREG register of the MCU, and to read a received byte, the data is read from the RCREG register of

the MCU. To ensure all sent byte was sent, the TXIF flag in PIR1 register was checked with an infinite polling loop. The code for sending a byte of data:

```
while(PIR1bits.TXIF == 1);  /* waiting for TXREG to be empty */
TXREG = data;          /* put a byte of data to be sent out */
```

The UART receiving part was similar, but made use of interrupt. When a byte of data has filled the RCREG, the interrupt flag (RCIF) will be raised, and the interrupt service routine (ISR) will be executed to check the source of the interrupt. The following code demonstrate how the low priority interrupt was used to receive a byte of data and store the byte of data in a buffer:

```
#pragma  code lowPriorityInterrupt = 0x18  /*  address of low priority
                                                interrupt  at 0x18 */
void lowPriorityInterrupt(void)
{
        _asm
        GOTO          chk_isr_low    /* go there when interrupt triggered */
        _endasm
}
#pragma code

#pragma interrupt chk_isr_low
void chk_isr_low(void)          /* interrupt service routine */
{
        if(PIR1bits.RCIF) receiving_function();
}

void receiving_function(void)
{
        char buffer = RCREG;          /* stored the receive data in a buffer */
}
```

The receiving function used in the project will start a timer everytime a byte was received. This is because to interface with XBee modem which used API packet structure, the receiving function must know when to start to process the data stored in the buffer. The timer was set 10 *msec* to timeout. Every time a byte was received, the timer is stopped and reseted, and turned on again with 10 *msec* to timeout. Once timeout event occurred, it triggers the high priority interrupt. During that time, receiving function will be disabled and the process data function will be called to process the data in the buffer.

The process data function was to identify XBee API packet, verify the checksum and to read the content of the data sent from home server. The method was to loop through every byte in the buffer and search for data with value 0x7E, which is the constant starting value of the XBee API packet. Then the next byte will be MSB, LSB, and so forth. All the bytes from receive packet were stored in a XBee API packet structure defined in file xbee_def.h. The bytes then being arithmetically processed to verify the checksum and then the content data will be read. A character 'M' in the data will ask the MCU to send back the information of microphone amplitude and frequency at that time.

To send back a packet from MCU to home server, the MCU assembled a transmit request packet (also 16-bits address) with the data requested and calculate the checksum to send to the modem. The sending to modem was byte by byte data, and upon finish sending, the modem will verify the checksum before sending back to home server. If the checksum was incorrect, the packet will be dropped silently. The full coding for the program in the MCU is attached in Appendix A.

3.5.2 Programming for Home Server

The program used on the home server was written in C++ programming language. The compiler used was GNU GCC compiler version 4.4.3. To program the graphic user interface (GUI), Qt4 C++ library version 4.6.2 was used.

3.5.2.1 Serial Port Programming

The XBee USB dongle that attached to home server uses virtual serial port for communication. When the dongle is inserted, it will mount to a device file in /dev directory in the Linux. Depending on the number of the mounted devices to the computer at that time, it will mounted to /dev/ttyUSBx file in the directory where the x is an integer indicating which file it was attached to. The first attached USB serial device will be ttyUSB0 and then the next one would be ttyUSB1 and so forth.

The serial port programming in Linux uses Portable Operating System Interface for Unix (POSIX) standard. The standard included with the definitions and function calls which are common to system that apply them. The language used was C programming language combined with C++ programming language for Object Oriented Programming (OOP).

To make use of the device file, the file was opened for read and write with system call. The stdin and stdout were duplicated for the use with the serial port program. The function of the serial program was to receive data from the an opened device file and convey the data to the stdout. The program will exit when an escape character (ESC or ^[or 0x1B) was sent to stdin of the program.

3.5.2.2 Graphic User Interface (GUI) Programming

The Qt4 C++ GUI library was used to program the GUI for the visual presentation of the sensors and control. Other than that, the back end of the GUI also contains the algorithm to process the data from the sensors. The GUI programmed and used for the project is shown in Figure 3.18.

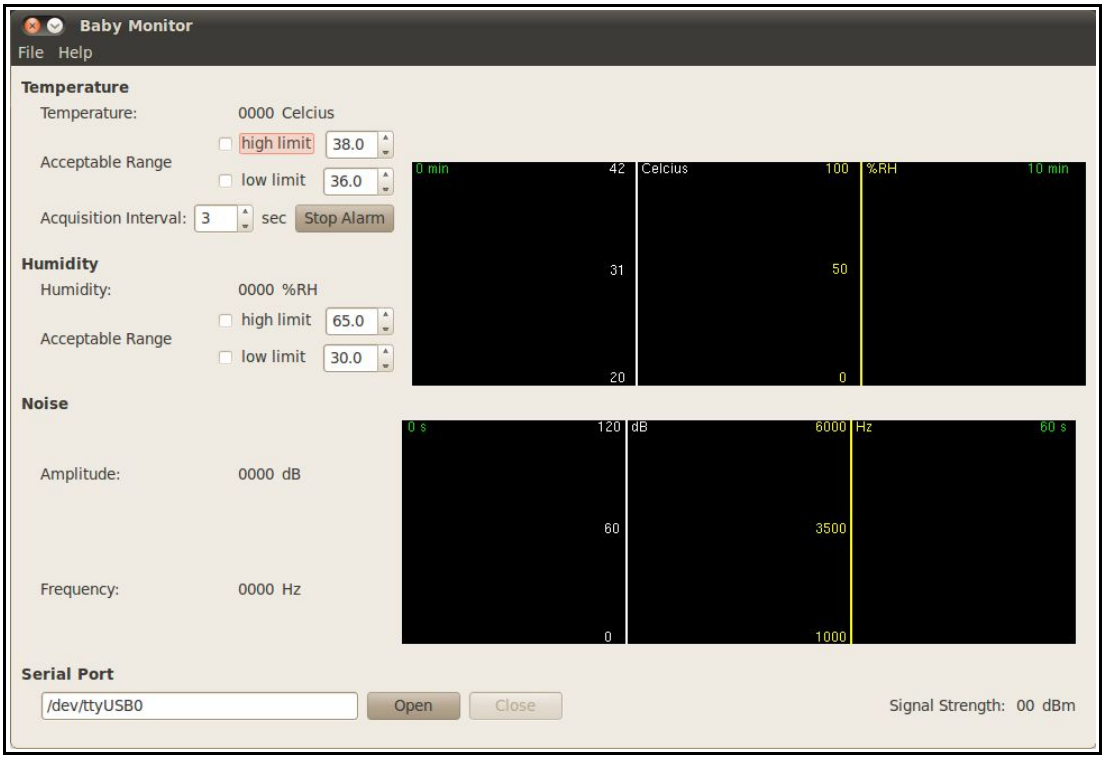


Figure 3.18: Baby Monitor GUI Program

For the microphone part, the GUI presents the amplitude and the frequency of the noise the microphone received at 5 samples per second, in real time. The graph widget on the lower will presents the changes in both amplitude and frequency in time range of 60 seconds. User can type in the path for serial port device file and open the serial port. The alarm system also commonly shared with the temperature and humidity sensing features. Therefore user can control the on / off of the alarm system. Other than that, signal strength of the received packet also will be shown at lower right corner of the GUI.

The back end of the GUI did the conversion for the amplitude and frequency of the microphone. The received data from base station was in binary format containing the actual value of the number of the steps on the ADC channels measured. After the conversion, the label on the GUI will be updated with the newest values obtained.

To interface with the XBee USB dongle attached to the home server, the serial port program was used. When opening the serial port from the GUI, a new process

was created which runs the serial port program. This ensure that both GUI and the serial port were independent of each other and communicate only through the stdin and stdout of the serial port program.

The alarm system also used another external program called “mplayer”, which function to run the alarm sound file. The alarm sound playing will run as another process, so that both alarm playing and the GUI will also be independent of each other.

The graph widget was used to plot graph with painting method. As the label for amplitude and frequency of the noise are updated, the graph will also be updated. By using graph to show the data, the user can visually track the recent changes (up to 60 seconds) and the pattern of amplitude and frequency of the noise. The size of the graph widget was fixed at 600x200 resolution. The program also used in the project to analyse the amplitude and frequency of the infant crying noise.

Upon successfully opened a serial port, the back end program will start to send pre-assembled data request packet to the modem through the serial port program 5 times per second to request microphone information, resulted in 5 samples per second of data acquisition. The request packet content is shown in Figure 3.6. Because the request packet content was the same throughout the whole program, the packet only needed to be assembled once during the initialisation of the GUI program.

3.5.2.3 Program Flow Chart

The flow chart in Figure 3.19 shows how the GUI program flow and how the program interacts with the XBee modem through the serial port program.

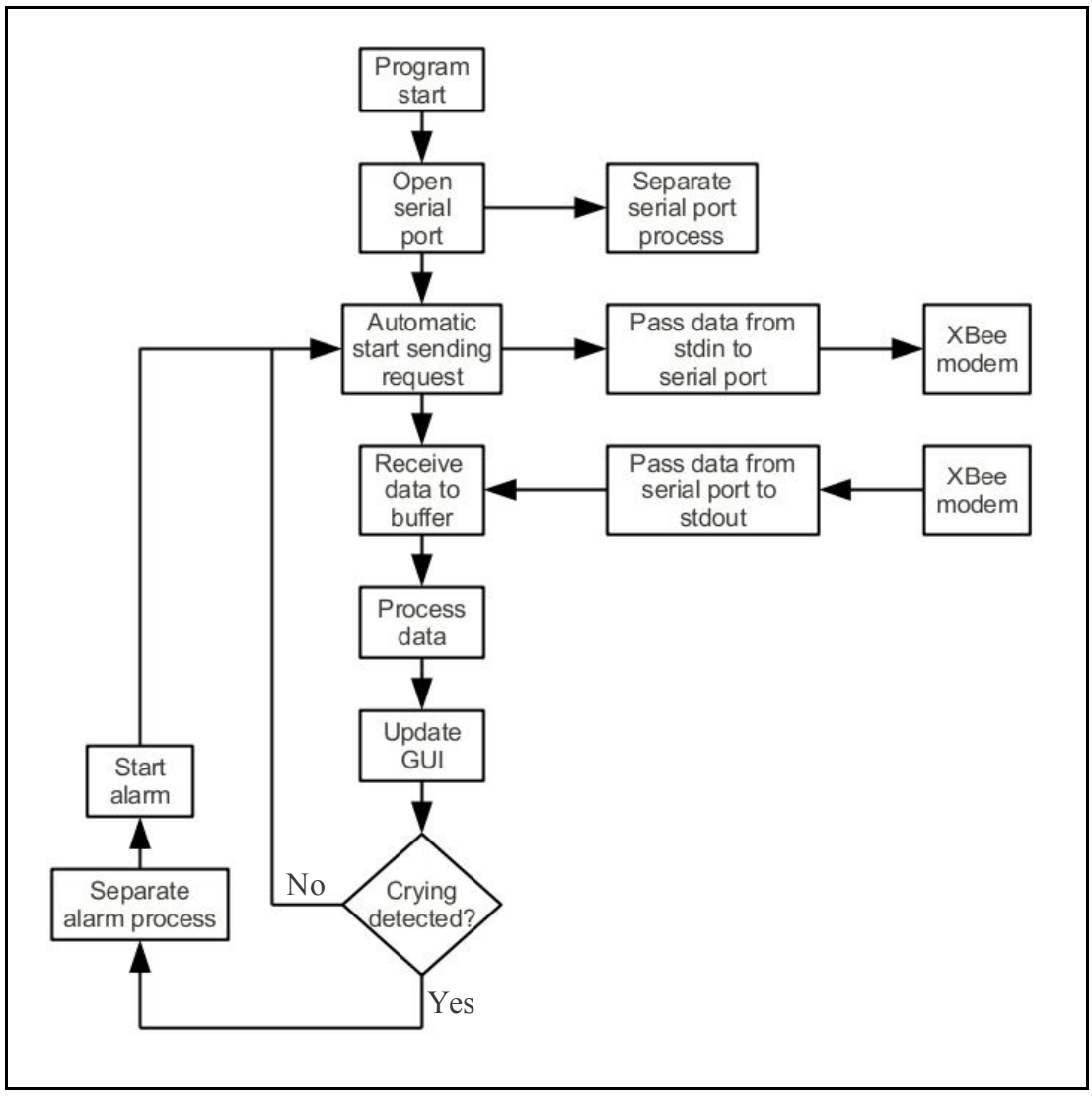


Figure 3.19: Program Flow Chart

3.6 Crying Detection

Based on the analysis of infant crying noise discussed in section 4.1.1 and section 4.1.2, the method to detect infant crying was designed. Three parameters was used in the detection: 1) intensity (amplitude), 2) frequency, and 3) duration of noise.

The intensity was used for crying detection using a threshold at 19 *dB SPL*. This is because the frequency detection only occurred at over this threshold. When the noise intensity reached the threshold, the frequency value approximately at the moment will be read. Then the consecutive readings will be recorded. When three consecutive readings have frequency detected, a counter will be increase by one, indicating one pulse of crying noise. The consecutive readings with frequency detected will not be taken into account because the threshold for duration was three readings (0.6 seconds), until a reading with no frequency detected was measured. When five pulse was recorded with the count, the infant crying is said to be detected. To prevent false positive detection, a timer was added in to limit the time for the occurrence of the pulse. If no pulse was detected in 10 seconds, the pulse count will be reseted to zero.

CHAPTER 4

RESULT & DISCUSSION

4.1 Result and Discussion

4.1.1 Microphone Output

The microphone output centered around 1.70 Volts and oscillate as noise was detected. By measuring the microphone amplitude generated, it was found that the voltage was centered at 1.08 V, and went up to 2.25 V at the maximum. Maximum acoustic input to the microphone is 120 deciBel Sound Pressure Level (dB SPL) (Analog Devices, 2010). So it can be concluded that at 0 *dB SPL*, the voltage output was 1.08 *V* and 120 *dB SPL* at 2.25 *V*. Therefore the linear gradient is calculated as in equation 4.1. The offset is calculated as -110.769.

$$gradient\ ,\ m = \frac{120 - 0}{2.25 - 1.08} = 102.564\ dB/V \tag{4.1}$$

Because the frequency response at the range that desired (1 *kHz* to 6 *kHz*) was approximately constant, it was assumed that the amplitude was proportional to the SPL. From 12-bit ADC, there was 2^12 steps. The conversion to *dB SPL* is given by equation 4.2.

$$dB\ SPL = 102.564 \cdot \left(\frac{ADC\ result}{4096} \times 3.3\right) - 110.769 \tag{4.2}$$

For the microphone frequency, the output from the LM2907 was measured against the frequency of noise applied with a buzzer powered by adjustable signal generator. The data was tabulated (Table 4.1) and plotted (Figure 4.1).

Table 4.1: Voltage Output of LM2907 Corresponding to Frequency of Noise

Frequency *kHz*	**Voltage** *V*
1.0	0.57
1.5	0.83
2.0	1.11
2.5	1.38
3.0	1.71
3.5	1.95
4.0	2.24
4.5	2.48
5.0	2.74
5.5	2.94
6.0	3.04

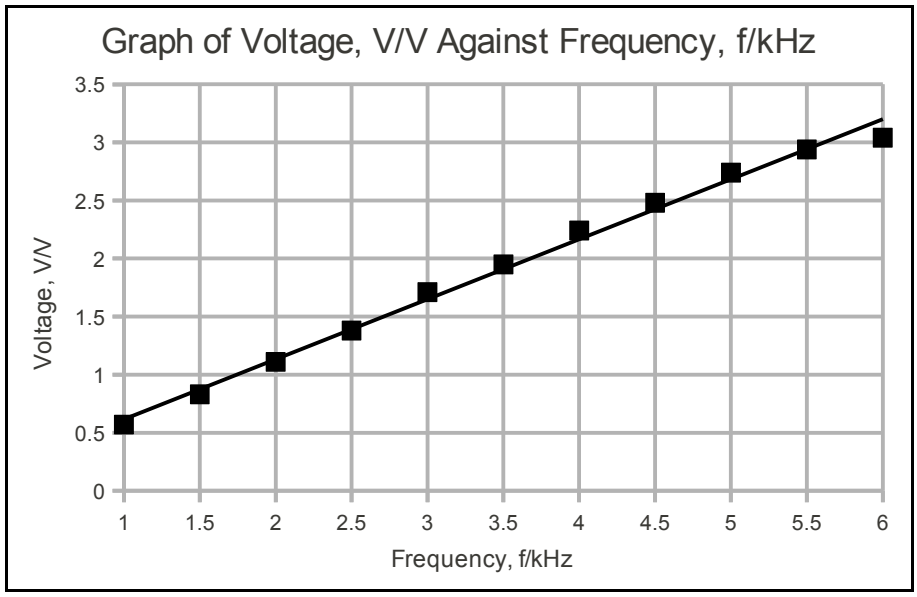


Figure 4.1: Graph of Voltage, *V/V* Against Frequency, *f/kHz*

The equation of the trend line in the plot is given in equation 4.3. The conversion equation from ADC result is given in equation 4.4.

$$Frequency\ ,\ f = 1.923 \cdot V - 0.154 \tag{4.3}$$

$$Frequency\ ,\ f = 1.923 \cdot \left(\frac{ADC\ result}{4096} \times 3.3\right) - 0.154 \tag{4.4}$$

where

f = Frequency of the noise in *kHz*.

V = Voltage output produced by the LM2907 in *V*.

The ADC results was sent to home server upon request and the unit conversion was done in the home server program.

It is noted that the center voltage of the microphone was stepped down from 1.70 *V* to 1.08 *V*. This was due to the voltage drop across the diode (1N4007) (Figure 3.15) that used to prevent back flow of current when the capacitor was discharging. Because the AC signal from the microphone was of relatively high frequency (desired frequency was 1 *kHz* and above), the signal was not filtered with a rectifier, instead, the discharging capacitor will smooth out the signal. However this did not always true, when low frequency and high amplitude signal were fed, the microphone output could went below 1.08 *V*. On the other hand, the capacitor value will determine the charging and discharging time and therefore cannot be too slow or too fast. The 1 μF capacitor was chosen for the best suitability.

For the LM2907, a graph need to be plotted to identify the linearity and corresponding voltage output to the frequency. Theoretically these can be calculated by equation 3.2. But in practical the result was not the same. However, linearity achieved for frequency range of 1 *kHz* to 5 *kHz*, which was used in equation 3.2 to calculate the resistance of the resistor and the capacitance of the capacitor used. The offset voltage to response by the LM2907 was important in the design, because the output of the microphone ranged approximately + / - 1.17 Volts. It was required 250 *mV* $V_{p\text{-}p}$ to trigger the LM2907 output. Theoretically, frequency only be detected when the intensity is at least 25.6 *dB SPL*. However, this was used as an advantage to the design because it can filter out low amplitude noise within the frequency range. Referring to Figure 3.15, the capacitor between 3, 4 and ground was for filtering. By

setting the large capacitance value, the ripple voltage will reduced, but it also slow down the charging and discharging time. Smaller capacitance has fast charging and discharging time but the ripple voltage will be greater. Therefore it was found that 1 μF capacitor did the job just fine.

The resolution of the noise intensity was calculated to be 0.0826 *dB SPL* and the minimum frequency detectable was 1.55 *Hz* in the linear range.

4.1.2 Infant Crying Noise

Ten samples of infant crying sound were used to identify the pattern and features of the sound. Because the crying detection will be done in the GUI program, the sampling will be done using the GUI program too. The sampling was done by running the digital recorded infant crying sound through the speaker and put the microphone directly facing the sound. One result of the crying noise sampling is shown in Figure 4.2. The white colour painted graph was for the amplitude of the noise, using the white colour scale, and the yellow colour painted graph was for the frequency of the noise, using the yellow colour scale. The full sampling result is in Appendix B.

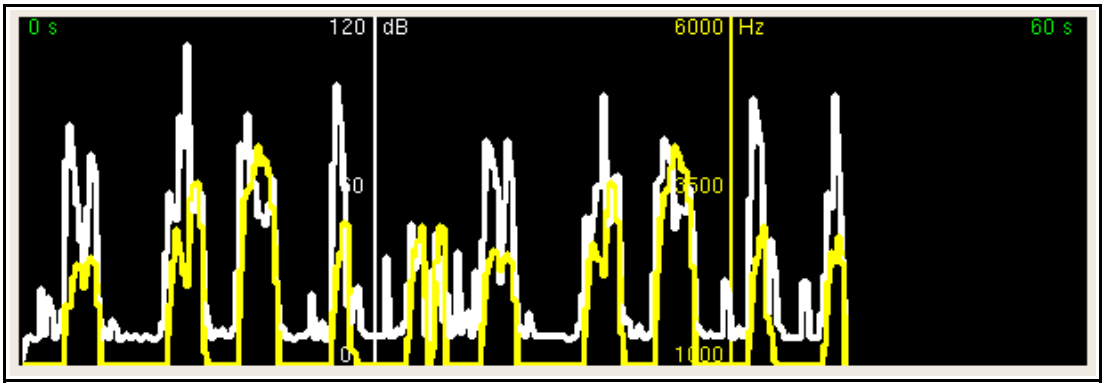


Figure 4.2: Real Time Plot of Infant Crying Sound

From those samples, it can be recognized that the crying pattern of infant was of irregular pulsatile noise. The frequency of the crying noise went up in almost synchronous with the intensity of the noise. Because the frequency detection threshold is 1 *kHz*, the graph for the frequency only detect the frequency above the threshold.

From the log file produced for these values, it can be seen that the frequency can be detected frequently at SPL as low as around 19 *dB SPL* compared to the theoretically calculated value at 25.6 *dB SPL*. Although the frequency was also read at as low as 10 – 11 *dB SPL*, but these were seldom occurred and therefore could be explained as that the time to read ADC channel for microphone amplitude and frequency was differ by a small amount of time which the the amplitude might have increase over the threshold to allow the detection of frequency after lower amplitude was converted by ADC. However another explanation for the low intensity frequency read was probably because of the heuristic characteristic of the LM2907 IC. This can be showed by checking the position of the low-intensity-frequency read log, which was all occurred at the end of each crying noise pulse.

The log file also revealed the duration of each irregular crying noise pulse. The typical crying pulse last for around 0.8 seconds, although in some of the samples the duration went down to 0.2 seconds, or went up to 3.0 seconds. However, these duration was not in high resolution enough to tell the exact duration because the sampling rate was 5 samples per seconds. Therefore crying detection methods could be done on the duration parameter, taking 0.6 seconds as the threshold for the occurrence of the crying pulse.

The crying noise samples from analysis only produces noise that hardly went over 3500 *Hz* of frequency, contrary to the infant crying noise centering at 3500 *Hz* as stated in the review. The first explanation would be the fact that most infant crying noise was below this value. The second explanation would be the fact that those higher frequency (>3500 *kHz*) noise were short and therefore hardly be detected by the sampling setup (5 samples per seconds). The third explanation would be the fact that the samples used in the sampling did not contain the crying noise with those frequencies.

Without running the samples, the recording of the ambient noise centered at 10 *dB SPL* and never reaches the threshold of frequency detection. After running several crowd noise sound effects, the sampling graph produced is as shown in Figure 4.3. The sound effects mainly consist of noise of crowd chatting in the background. This showed that although the noise of people chatting might be loud, but the frequency does not gone over 1 *kHz*, and therefore the detection will not be done the type of noise.

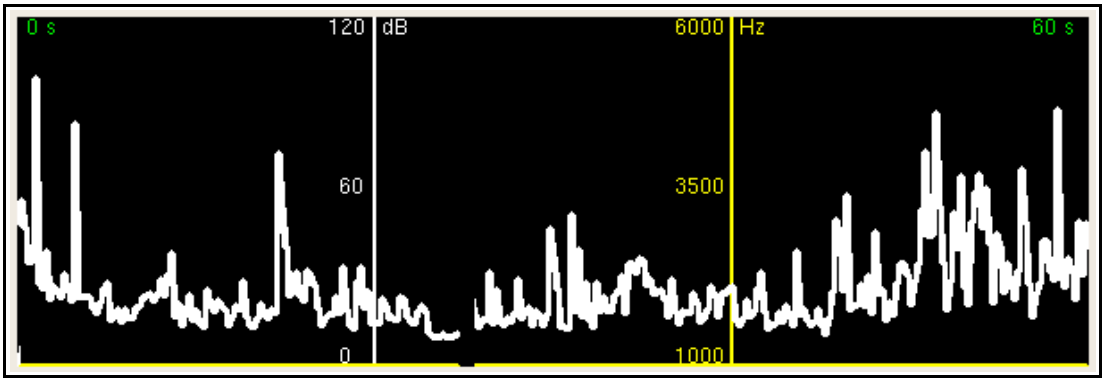


Figure 4.3: Sampling Graph of Crowd Noise

4.1.3 Crying Detection

After implementing the crying detection method as describe in section 3.6, the samples were used to test the performance of the algorithm. Out of ten samples used, five of them were able to be detected, giving 50 % efficacy. Some of the samples with short crying pulses were not detected, otherwise the detectable pulses were too far apart in time. The counting of pulses was reseted when the timeout of 10 seconds signaled. The sample with graph as shown in Figure 4.3 was able to be detected because the pulse occurred before the timeout.

The example audio log that produced by the sample is shown in Figure 4.4. The first column (number 183 to 214) is the line number, the second column is the time (hh:mm:ss), the third column is the amplitude value, and the fourth column is the frequency value. Three consecutive positive detections constitute a start of crying pulse and an negative detection following constitute the end of the crying pulse. If

crying pulses detection within 10 seconds of the pulse before, the pulses count will accumulate and when 5 counts are detected, the alarm will sound.

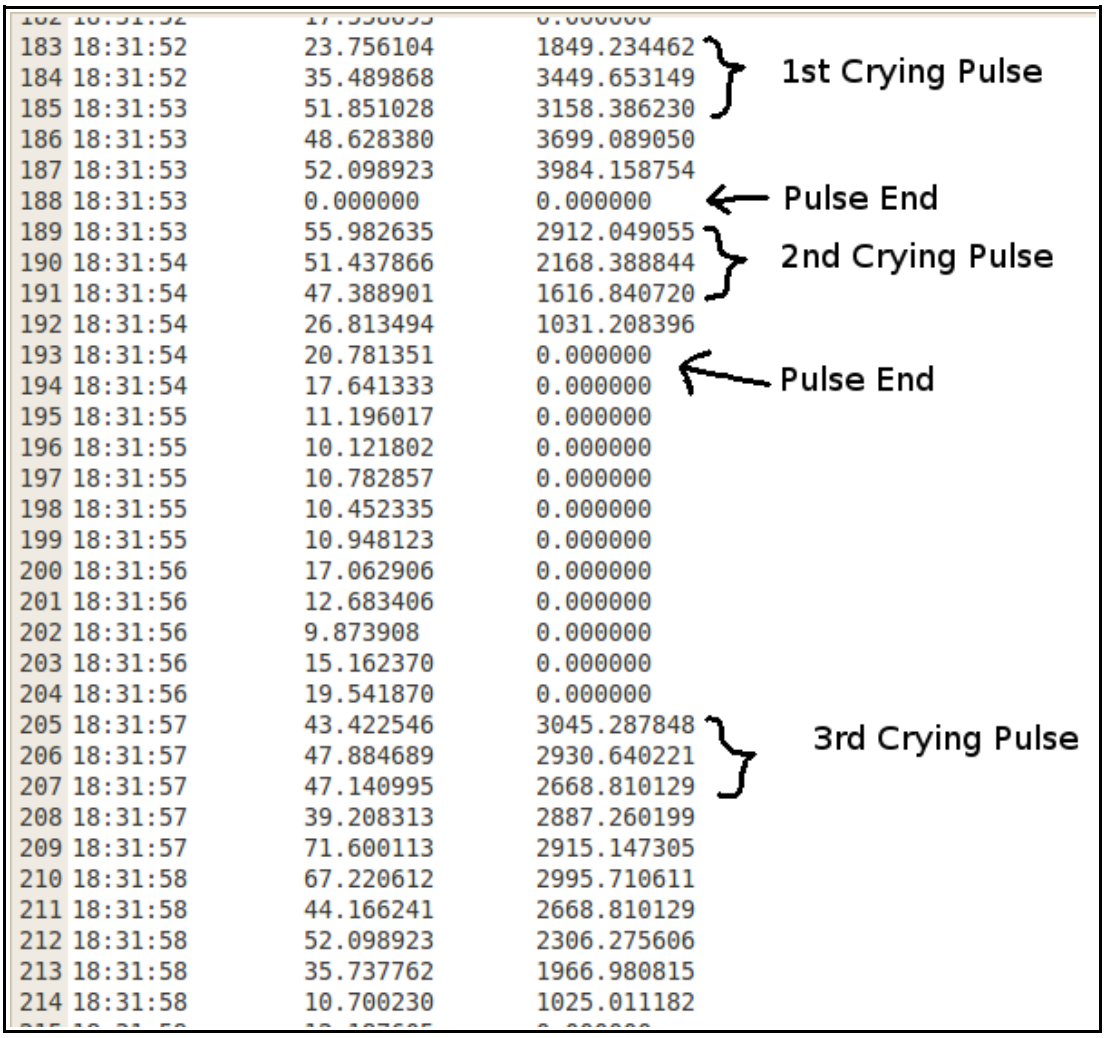


Figure 4.4: Sample Audio Log Showing the Event of Crying Pulses

Three of the samples were too short (7 – 12 sec) for the detection to perform and therefore the algorithm was failed to detect it. The intensity of the noise also one factor that affect the detection. The sampling was done with the microphone directly facing the speaker output of the digital crying sound clips with distance approximately and therefore the intensity was intense enough for the frequency to reach the threshold. When the microphone was placed approximately at distance 15 *cm*, the intensity measured was relatively lower and the frequency was not detected at all. Therefore, for the detection to work, the microphone must be placed adjacent to the sound source.

4.1.4 XBee Modem to Modem Communication

For a XBee to establish a communication from power up, it was estimated that the time before the first data sent in was 7 seconds. The reason was that the modem was searching for the target destination based on the destination address.

The signal strength indication of received packet (RSSI) shown at the GUI allows the user to know whether the modems were in range, so that the it could be use to determine the location and the distance suitable for the placing of the modems.

4.1.5 Power Supply & Power Consumption

The cascade configuration used for the power supply greatly reduced the heat generated at LM1117 3.3 V voltage regulator. The amount of heat generated from voltage regulator depends on how much voltage was stepped down during the regulation. Since there was only one AC adapter used, and at least 7.0 Volts are required to power up the LM7805, to minimize the heat generated, only LM7805 was connected directly to the AC adapter while LM1117 was powered by the 5.0 V voltage from LM7805. A multimeter measurement of the voltage from the power supply was around 8.79 *V,* although the configuration at the adapter was 6 *V*. This was due to the low current consumption from the circuit compared to the maximum current drawable from the adapter itself. In other words, the load used was considerably small compared to the maximum load that the adapter can support. If LM1117 was also connected to the power supply, the voltage gap that needed to be stepped down is 5.49 *V.* Compared to voltage that needed to be stepped down by LM7805, the LM1117 will generate relatively more heat. The cascade configuration let LM1117 to step down from 5.0 *V* to 3.3 *V*, with gap of only 1.7 *V*. Therefore, by using the cascade configuration the heat generated was reduced by minimizing the voltage gap that needed to be stepped down. Therefore, only LM7805 was felt to heat up when powered by AC adapter.

The current drew by the circuit measured steadily at 92.1 *mA* with one LED in the circuit that constantly on, and 70.8 *mA* when the LED was removed. If the circuit was to power up with a battery that can be purchase from the grocery store instead of AC adapter, the most suitable type of battery would be C-type battery. This is because the typical drain of the battery is 100 *mA*. Using battery with lower typical drain than the application required would damage the battery. With C-type alkaline battery, the capacity could be up to 8000 *mAh*. Which means it will give roughly 86 hours (3.5 days) of application with the LED attached, and roughly 113 hours (4.7 days) of application without the LED.

The voltage dropped across the whole circuit was 8.79 *V*. From this, the power consumption can be calculated from multiplying the voltage dropped and the current sank together, which gave 810 *mW* with the LED and 622 *mW* without the LED. Therefore, to minimize the power consumption, the LED should be removed.

4.1.6 Challenges in Infant Crying Detection

Audio is an analogue signal that requires high processing speed to process it. In the project, embedded system that being used was not having high processing unit. Therefore the crying detection method used was to measure the noise intensity and frequency. However, these parameters also varies very quickly in crying sound.

Theoretically, high sampling rate was required to capture as much changes in those parameters as possible. But due to the design utilized XBee API packet structure, which requiring additional processing time, the sampling rate was reduced to 5 samples per second. Then the microphone processed output was attached with capacitor, which allows discharge when the output voltage drop below the voltage of the charged capacitor. These made the noise intensity and frequency to be read approximately from the peak noise intensity and frequency that detectable before the reading done by the ADC.

To recognize the crying, a simple algorithm was used instead of implementing advance artificial intelligence algorithm. This allowed the detection to be done on low speed processor and therefore it can be done in real time measure. Although the chances of false positive detection was high if different kind of noise was used to test the efficacy of the algorithm, but since to limit the usage of the detection to a quiet room with an infant placed in the room, the most probably false positive detection was due to infant's laughing noise.

The detection also depend on the quality of the crying noise. If the crying pulse did not last at last 0.6 seconds and recurrence within 10 seconds, the crying will not be detected. The intensity of the crying sound also need to be loud, otherwise the microphone need to be placed near to the sound source so that the intensity threshold is reached to enable frequency detection.

4.1.7 Challenges of XBee API Packet Implementation

Using XBee API packet for transmitting and receiving, the structure of the data sent and received was longer. The microphone information request packet used in the project consist of 10 bytes (Figure 3.6), but the actually data that need to be sent is just a character 'M' (one byte). When receiving the packet from the MCU respond, the RX packet consisted of 14 bytes (Figure 3.8), the actual requested data was only 5 bytes (a character 'M', and the following 4 bytes of ADC results for microphone amplitude and frequency). It can be seen that the API frame was only negligible if the data content is consisted of many bytes. Otherwise, the bytes were just wasted on the frame and more time will be consumed to send the bytes.

The extra instruction cycles also needed to process the API packet. The API identifier need to be checked to determine the type of packet received, and the checksum need to be calculated to verify the validity of the data. For home server computer, these processing was not a problem, but for the MCU, the processing will took significance processing time. To reduce the processing time, the API packet used in the project was reduced to just 16-bit address transmit request packet and 16-

bit address receive packet. The other packet that received will be discarded.

By limiting the type of API packet used in the project, many functions of the XBee cannot be applied, such as to execute command on attached XBee modem and also to apply remote command. However, these functions were just for the configuration and will not be used when the modems were properly configured.

The project implement the API mode to demonstrate the feature of the XBee API packet. The advantages gained were checksum ability, destination address targeting, and also the received signal strength indicator (RSSI) value.

CHAPTER 5

CONCLUSION & RECOMMENDATION

5.1 Conclusion

The telemonitoring as a branch of telemedicine is a potential field for expansion and development. Although most homecare telemonitoring under research nowadays is focused on diseased adults and elderly people, infant monitoring is also an important field. By using the method described, the crying of an infant will be able to be detected. Even though the equivalent infant crying detection product is already available in market, but the aim of the present project is focus on multiple sensors network and healthcare, partly shows the realisation of self-designed crying detection device using XBee wireless transmission.

5.2 Recommendation

There were still improvement that can be done on the infant crying detector. The processor used can be replaced with high performance processor and better ADC IC can be used to increase the resolution and ADC time. The sampling rate can then be increased. A digital signal processing artificial intelligence algorithm also can be developed and implemented to recognise the event of crying. By doing this the false positive detection could be reduced. When having better data processing, one may develop a crying recognition algorithm that can recognize the cause of the crying, such as hunger, lonely or uncomfortable.

A medical server can also be developed in the future work to implement the bigger range of telemonitoring. The medical server at the remote area can use to log the event of crying and record it in the server data base.

The XBee Pro can be used to improve the distance range of communication, which allows the use of the device in a wide area.

An amplifier can be added to the microphone output to improve the signal amplitude. The amplified signal will have larger range of output and therefore easier to be detected by ADC. The threshold of the LM2907 will also can be reached easily when the signal is amplified, allowing the frequency of low intensity sound to be detected. With amplifier, the microphone will not need to strictly placed near the source of the sound, instead it can be placed at a distance away.

REFERENCE

Analog Devices (2010). *Omnidirectional Microphone with Bottom Port and Analog Output ADMP401*. Analog Devices, Inc. 2010.

Andriana, N., Victor, A.V.D., Ioan, I. (2008). A ZigBee solution for telemedicine applications. *Acta Technica Napocensis: Electronics and Telecommunications* (2008), Vol. 49-3.

Barr, R.G. (2006). Crying Behaviour and its importance for psychosocial development in children. *Encyclopedia on Early Childhood Development.* Published online April 13, 2006, at http://www.child-encyclopedia.com/en-ca/child-crying-behaviour/according-to-experts.html

Chau-Kai, H. (1992). Baby cry recognizer. *Patern Storm*. Retrieved August 31, 2010, from http://www.patentstorm.us/patents/5668780/claims.html

Cirronet, Inc. (2005). *RF power options in ZigBee™ solutions.* Cirronet, Inc. (Fall 2005).

Faludi, R. (2011). *XBee Firmware Upgrade.* Retrieved February 16, 2011, at http://www.faludi.com/itp_coursework/meshnetworking/XBee/XBee_firmware_upgrade.html

Goodman, B. (May 4, 2006). I hear ringing and there's no one there. I wonder why. *The New York Times* (May 4, 2006). Retrieved August 31, 2010, from http://www.sounddogs.com/htm/article24.htm

Hosiden. *Guide for electret condenser microphones*. Retrieved August 30, 2010 from http://www.hosiden.co.jp/web/english/web/products/pdf/e_on07_mems.pdf

Hsu, H-C., Fogel, A., & Cooper, R.B. (2000). Infant vocal development during the first 6 months: speech quality and melodic complexity. *Infant and Child Development* (2000), 9. 1-16.

Huang, A.S., & Rudolph, L. (2007). *Bluetooth essentials for programmers.*

Massachusetts Institue of Technology. Cambridge University Press, UK.

Intel ®, CISCO™ (2010). *Intel and Cisco WLAN deployment guide for healthcare.*

Justin, L. (October 19, 2009). MEMS microphones. *MECH207.* Retrieved August 31, from http://mech207.engr.scu.edu/SensorPresentations/Lee%20-%20MEMS_Microphone%20Combined.pdf

Kammer, D., McNutt, G., Senese, B., & Bray, J. (2002). *Bluetooth, Application developer's guide: the short range interconnect solution.* Rockland: Syngress Publishing, Inc. (2002).

Khoór, S., Nieberl, J., Fügedi, K., & Kail, E. (2001). Telemedicine ECG-Telemetry with Bluetooth Technology. *Computers in Cardiology* (2001) 28. 585-588.

Kinney, P. (October 2, 2003). ZigBee technology: Wireless control that simply works. *Communications Design Conference* (2003).

Koch, S. (2006). Home telehealth-current state and fuure trends. *International Journal of Medical Informatics* (2006) 75. 565-576.

Kugean, C., Krishnan, S.M., Chutatpe, O., Swaminathan, S., Srinivasan, N., & Wang, P. (2002). *Design of a mobile telemedicine system with wireless LAN.* IEEE © 2002, 0-7803-7690-0/02.

Lizard43 (2008). *X-CTU with Linux.* Retrieved on February 15, 2011, at http://lizard43.blogspot.com/2008/10/x-ctu-with-linux.html

Manfredi, C., Bocchi, L., Orlandi, S., Spaccaterra, L., & Donzelli, G.P. (2009). High resolution cry analysis in preterm newborn infants. *Medical Engineering & Physics* (2009), 31. 528-532.

MediaCollege.com. *Condenser microphones.* Retrieved August 30, 2010, from http://www.mediacollege.com/audio/microphones/condenser.html

Microchip (2007). *PIC18F2423/2523/4423/4523 Data Sheet.* Microchip Technology Inc. (2007).

Nakayama, H. (2010). Development of infant crying behavior: A longitudinal case study. *Infant Behavior and Development* (2010), doi: 10.1016/j.infbeh.2010.05.002.

National Semiconductor (2008). *LM2907/LM2917 Frequency to Voltage Converter.*

National Semiconductor Corporation (2008).

Nielsen, J.H., & Fürst, C. (2007). Toward more-compact digital microphones. *Analog Dialogue*, Vol. 41 (September 2007).

Reyes-Galaviz, O.F., & Reyes-Garcia, C.A. (2004). A system for the processing of infant cry to recognize pathologies in recently born babies with neural networks. *SPECOM 2004: 9th Conference Speech and Computer* (September 20-22, 2004).

Srovnal, V., & Panhaker, M. (2006). *Home care and health maintenance systems.* Retrieved July 10, 2010, from http://medlab.cs.uoi.gr/itab2006/proceedings/eHealth.htm

Stifter, C.A. (2005). Crying Behaviour and its impact on psycholsocial child development. *Encyclopedia on Early Childhood Development.* Published online April 4, 2005, at http://www.child-encyclopedia.com/en-ca/child-crying-behaviour/according-to-experts.html

Várallyay, G. (2007). The melody of crying. *International Journal of Pediatric Otorhinolaryngology* (2007), 71. 1699-1708.

Várallyay, G., Illényi, A., Benyó, Z., Farkas, Z., & Katona, G. (2005). *An attempt to detect hearing disorder by acoutic features of the infant cry.* Proceedings of the Forum Acusticum 2005 Congress, Budapest. 526/1-6. ISBN 963 8241 68 3.

Vergari, F., Auteri, V., Corsi, C., Lamberti, C.. A ZigBee-based ECG transmission for a low cost solution in home care services delivery. *Mediterranean Journal of Pacing and Electrophysiology.*

Zeskind, T.S. (2007). Impact of the cry of the infant at risk on psychosocial development (Revised August 24, 2007). *Encyclopedia on Early Childhood Development.* Published online April 4, 2005, at http://www.child-encyclopedia.com/en-ca/child-crying-behaviour/according-to-experts.html

ZigBee ® Alliance, American Telemedicine Association (November 17, 2009). *The ZigBee Alliance and the American Telemedicine Association to collaborate on advancing use of telehealth solution.*

ZigBee ® Alliance. ZigBee and wireless radio frequency coexistence. *ZigBee White Paper* – June 2007. ZigBee ® Alliance (2009).

APPENDICES

APPENDIX A: Source Codes for Program in MCU

SOURCE CODES FOR PROGRAM IN MCU

File name: test18.c

```
/* receiving rx packet from xbee, and blink LED on PB7 upon success checksum */

/** C O N F I G U R A T I O N   B I T S
************************************/

#include <p18f4523.h>
#include <string.h>  /* to use memset */
#include "mytype.h"
#include "xbee_def.h"

#pragma config IESO = OFF, FCMEN = OFF, OSC = HS
#pragma config PWRT = ON, BOREN = OFF, BORV = 3
#pragma config WDT = ON, WDTPS = 256
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = OFF, CCP2MX =
PORTC
#pragma config STVREN = ON, LVP = OFF, XINST = OFF, DEBUG = OFF
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB = OFF, CPD = OFF
#pragma config WRT0 = ON, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF

#pragma config WRTB = ON, WRTC = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF

#pragma config EBTRB = OFF

#define PB6    LATBbits.LATB6
#define PB7    LATBbits.LATB7
#define RXINT          PIR1bits.RCIF
#define TXINT          PIR1bits.TXIF
#define TMR0INT      INTCONbits.TMR0IF
#define RX_BUFFER_SIZE 32
```

```
#define DELIMITER 0x7E

volatile static uchar rxbuffer[RX_BUFFER_SIZE];
volatile static uchar rxbuffer_count;
volatile static uchar timeout_flag;
volatile static uchar process_data_flag;
volatile static uchar send_TH_flag;
volatile static uchar send_M_flag;
volatile static RX_PKT_16 rx_sample;
volatile static TX_PKT_16 tx_sample;
volatile static uchar tempbuffer[4];
volatile static uchar mbuffer[4];

void init_tx_pkt(void);
void rx_func(void);
void tx_func(uchar);
void timeout_func(void);
void process_data_func(void);
void send_TH_func(void);
void send_M_func(void);
void getTH(void);
void getM(void);

void chk_isr_high(void);
void chk_isr_low(void);


#pragma code highPrioINT = 0x0008
void highPrioINT (void)
{
        _asm
        GOTO chk_isr_high
        _endasm
}
#pragma code

#pragma code lowPrioINT = 0x00018
void lowPrioINT (void)
{
        _asm
        GOTO chk_isr_low
        _endasm
}
#pragma code

#pragma interrupt chk_isr_high
void chk_isr_high(void)
{
        /* check our high priority interrupt flag(s) here */
        if(TMR0INT) timeout_func();
```

```
}

#pragma interrupt chk_isr_low
void chk_isr_low(void)
{
	/* check our low priority interrupt flag(s) here */
	if(RXINT) rx_func();
}

void main()
{
	TRISBbits.TRISB6 = 0;
	TRISBbits.TRISB7 = 0;
	TRISCbits.TRISC6 = 0; /* set tx = output */
	TRISCbits.TRISC7 = 1; /* set rx = input */
	TRISAbits.TRISA0 = 1;

	/* interrupt enables */
	INTCONbits.GIE = 1;
	INTCONbits.PEIE = 1;
	RCONbits.IPEN = 1;  /* enable interrupt priorities */
	PIE1bits.RCIE = 1;  /* receive interrupt */
	IPR1bits.RCIP = 0; /* set low priority for rx INT */
	INTCONbits.TMR0IE = 1; /* timer interrupt */
	INTCON2bits.TMR0IP = 1; /* set high priority for timer INT */

	/* ADC stuff */
	ADCON1 = 0x17;	/* up to AN7 */
	ADCON0 = 0x00;	/* channel 0, disable conv. first */
	ADCON2 = 0xBE;	/* right justified, 20 TAD, Fosc/64 */

	/* serial port stuff */
	TXSTA = 0x20;
	RCSTAbits.RX9 = 0;
	BAUDCONbits.BRG16 = 0; /* 8 bits BRG */
	BAUDCONbits.ABDEN = 0;		/* disable the auto baud rate detection */
	SPBRG = 15;  /* 19200 with 1.73 error */
	RCSTAbits.SPEN = 1; /* enable corresponding pins to be serial port */
	TXSTAbits.TXEN = 1;
	RCSTAbits.CREN = 1;
	OSCCONbits.IDLEN = 1; /* idle when "SLEEP" is issued */

	/* timer setup stuff */
	T0CON = 0x07;  /* prescale 256 */

	/* set the initial condition of register & variables */
	memset((char*)rxbuffer, '\0', RX_BUFFER_SIZE);
	memset((char*)tempbuffer, '\0', 4);
	memset((char*)mbuffer, '\0', 4);
	rxbuffer_count = 0;
```

```
	timeout_flag = 0;
	process_data_flag = 0;
	send_TH_flag = 0;
	send_M_flag = 0;

	init_tx_pkt();

	PB6 = 1;
	PB7 = 1;

	while(1)
	{
		Sleep();

		PB6 = ~PB6;
		if(process_data_flag) process_data_func();

		if(send_M_flag){
			getM();
			send_M_func();
		}
		else if(send_TH_flag)
		{
			getTH(); /* do adc read & convert and put in global static var
*/
			send_TH_func();
		}
		RCSTAbits.CREN = 1;
		INTCONbits.GIE = 1;
	}
}

void init_tx_pkt(void)		// initiate fixed values of tx packet
{
	tx_sample.DELIM = 0x7E;
	tx_sample.APIID = 0x01;
	tx_sample.FrameID = 0x52;
	tx_sample.AddrH = 0x01;
	tx_sample.AddrL = 0x01;
	tx_sample.opt = 0x04;

	tx_sample.checksum = 0;
	tx_sample.dataSize = 0;
}

void rx_func(void)
{
	/* stop timer, receive data, reset time */
	T0CONbits.TMR0ON = 0;
	rxbuffer[rxbuffer_count] = RCREG;
```

```
        if((!timeout_flag)&&(rxbuffer_count < RX_BUFFER_SIZE - 1))
        {
                ++rxbuffer_count;
                TMR0H = 0xFF;    /* 0.01 second to timeout */
                TMR0L = 0x3D;
                T0CONbits.TMR0ON = 1;
        }
}

void timeout_func(void)
{
        INTCONbits.GIE = 0;
        RCSTAbits.CREN = 0;
        TMR0INT = 0;
        T0CONbits.TMR0ON = 0;
        timeout_flag = 1;
        process_data_flag = 1;
}

void process_data_func(void)
{
        uchar i, j, k;
        uchar rx_checksum= 0, cal_checksum = 0, data_sum = 0;
        rx_sample.dataSize = 0;

        for(i = 0; i < rxbuffer_count; i++)
        {
                if(rxbuffer[i] == DELIMITER)
                {
                        rx_sample.DELIM = rxbuffer[i];
                        i++;
                        rx_sample.MSB = rxbuffer[i];
                        i++;
                        rx_sample.LSB = rxbuffer[i];
                        i++;
                        rx_sample.APIID = rxbuffer[i];
                        i++;
                        if(rx_sample.APIID == 0x89) break; /* if tx status */

                        rx_sample.AddrH = rxbuffer[i];
                        i++;
                        rx_sample.AddrL = rxbuffer[i];
                        i++;
                        rx_sample.RSSI = rxbuffer[i];
                        i++;
                        rx_sample.opt = rxbuffer[i];
                        i++;

                        k = 0;
```

```
                        for(j = i; j < (i + rx_sample.LSB); j++)
                        {
                                rx_sample.data[k] = rxbuffer[j];
                                data_sum += rx_sample.data[k];
                                rx_sample.dataSize++;
                                k++;
                        }
                        rx_checksum = rxbuffer[j];

                        cal_checksum = 0xFF - (rx_sample.APIID + rx_sample.AddrH
                                        + rx_sample.AddrL + rx_sample.RSSI +
                                        rx_sample.opt + data_sum);

                        if(cal_checksum == rx_checksum)
                        {
                                PB7 = ~PB7;
                                for(i=0;i<rx_sample.dataSize;i++)
                                {
                                        if(rx_sample.data[i] == 'M'){
                                                send_M_flag = 1;
                                                break;
                                        }
                                        else if(rx_sample.data[i]=='T')
                                        {
                                                send_TH_flag = 1;
                                                break;
                                        }
                                }
                        }

                        break;

                } /* finish process one packet if exist */

        } /* finish looping & detecting delimiter */


        /* clear the rxbuffer */
        memset((char*)rxbuffer, '\0', RX_BUFFER_SIZE);

        /* reset counter */
        rxbuffer_count = 0;

        /* restore the flags */
        process_data_flag = 0;
        timeout_flag = 0;

}

void send_TH_func(void)
```

```
{
        uchar i;

        tx_sample.dataSize = 11; /* T__H__M____ */
        tx_sample.MSB = 0x00;
        tx_sample.LSB = tx_sample.dataSize + 5;

        tx_sample.data[0] = 'T';
        tx_sample.data[1] = tempbuffer[0];
        tx_sample.data[2] = tempbuffer[1];
        tx_sample.data[3] = 'H';
        tx_sample.data[4] = tempbuffer[2];
        tx_sample.data[5] = tempbuffer[3];
        tx_sample.data[6] = 'M';
        tx_sample.data[7] = mbuffer[0];
        tx_sample.data[8] = mbuffer[1];
        tx_sample.data[9] = mbuffer[2];
        tx_sample.data[10] = mbuffer[3];

        tx_sample.checksum = tx_sample.APIID + tx_sample.FrameID +
                        tx_sample.AddrH + tx_sample.AddrL + tx_sample.opt;

        for(i = 0; i< tx_sample.dataSize; i++)
        {
                tx_sample.checksum += tx_sample.data[i];
        }

        tx_sample.checksum = 0xFF - tx_sample.checksum;

        /* sending of tx pkt */
        tx_func(tx_sample.DELIM);
        tx_func(tx_sample.MSB);
        tx_func(tx_sample.LSB);
        tx_func(tx_sample.APIID);
        tx_func(tx_sample.FrameID);
        tx_func(tx_sample.AddrH);
        tx_func(tx_sample.AddrL);
        tx_func(tx_sample.opt);
        for(i = 0; i< tx_sample.dataSize; i++)
        {
                tx_func(tx_sample.data[i]);
        }
        tx_func(tx_sample.checksum);
        /* end packet */

        send_TH_flag = 0; /* reset the flag */

        tx_sample.checksum = 0;
        tx_sample.dataSize = 0;
}
```

```
void getTH(void)
{
	ADCON0 = 0x00; // channel A0
	ADCON0bits.ADON = 1;
	ADCON0bits.GO = 1;

	while(ADCON0bits.DONE == 1);
	tempbuffer[0] = ADRESH;
	tempbuffer[1] = ADRESL;

	ADCON0 = 0x04; // channel A1
	ADCON0bits.ADON = 1;
	ADCON0bits.GO = 1;

	while(ADCON0bits.DONE == 1);
	tempbuffer[2] = ADRESH;
	tempbuffer[3] = ADRESL;

	ADCON0bits.ADON = 0;

	getM();
}

void send_M_func(void)
{
	uchar i;

	tx_sample.dataSize = 5; /* M____ */
	tx_sample.MSB = 0x00;
	tx_sample.LSB = tx_sample.dataSize + 5;

	tx_sample.data[0] = 'M';
	tx_sample.data[1] = mbuffer[0];
	tx_sample.data[2] = mbuffer[1];
	tx_sample.data[3] = mbuffer[2];
	tx_sample.data[4] = mbuffer[3];

	tx_sample.checksum = tx_sample.APIID + tx_sample.FrameID +
			tx_sample.AddrH + tx_sample.AddrL + tx_sample.opt;

	for(i = 0; i< tx_sample.dataSize; i++)
	{
		tx_sample.checksum += tx_sample.data[i];
	}

	tx_sample.checksum = 0xFF - tx_sample.checksum;

	/* sending of tx pkt */
	tx_func(tx_sample.DELIM);
```

```
        tx_func(tx_sample.MSB);
        tx_func(tx_sample.LSB);
        tx_func(tx_sample.APIID);
        tx_func(tx_sample.FrameID);
        tx_func(tx_sample.AddrH);
        tx_func(tx_sample.AddrL);
        tx_func(tx_sample.opt);
        for(i = 0; i< tx_sample.dataSize; i++)
        {
                tx_func(tx_sample.data[i]);
        }
        tx_func(tx_sample.checksum);
        /* end packet */

        send_M_flag = 0; /* reset the flag */

        tx_sample.checksum = 0;
        tx_sample.dataSize = 0;
}

void getM(void)
{
        ADCON0 = 0x14; // channel A5
        ADCON0bits.ADON = 1;
        ADCON0bits.GO = 1;

        while(ADCON0bits.DONE == 1);
        mbuffer[0] = ADRESH;
        mbuffer[1] = ADRESL;

        ADCON0 = 0x18; // channel A6
        ADCON0bits.ADON = 1;
        ADCON0bits.GO = 1;

        while(ADCON0bits.DONE == 1);
        mbuffer[2] = ADRESH;
        mbuffer[3] = ADRESL;

        ADCON0bits.ADON = 0;
}

void tx_func(uchar bdata)
{
        while(!TXINT);
        TXREG = bdata;
}
```

File name: xbee_def.h

```
#ifndef XBEE_DEF_H
#define XBEE_DEF_H

#include"mytype.h"

typedef struct rx_pkt_16
{
	uchar DELIM;
	uchar MSB;
	uchar LSB;
	uchar APIID;
	uchar AddrH;
	uchar AddrL;
	uchar RSSI;
	uchar opt;
	uchar data[32];
	uchar dataSize;
	uchar checksum;

} RX_PKT_16;

typedef struct tx_pkt_16
{
	uchar DELIM;
	uchar MSB;
	uchar LSB;
	uchar APIID;
	uchar FrameID;
	uchar AddrH;
	uchar AddrL;
	uchar opt;
	uchar data[32];
	uchar dataSize;
	uchar checksum;
} TX_PKT_16;

#endif
```

File name: mytype.h

```
#ifndef MYTYPE_H
#define MYTYPE_H

typedef unsigned char uchar;

#endif
```

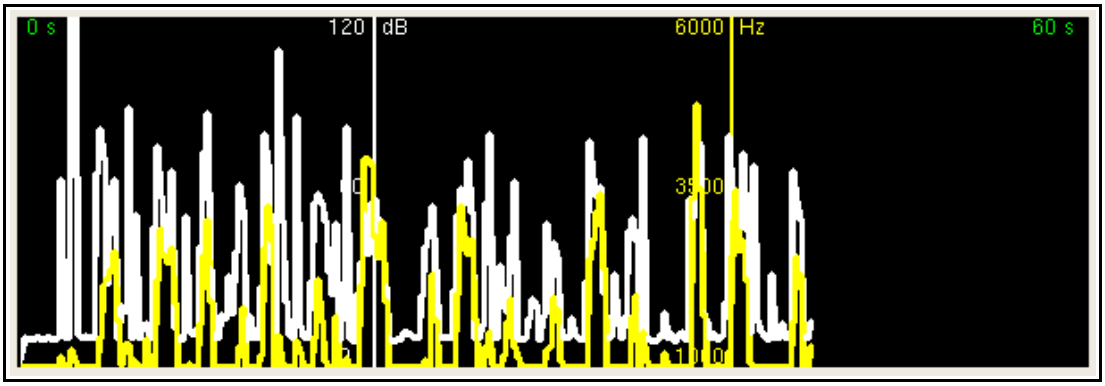
Appendix B: Sampling Result of Infant Crying Sample

Sampling Results

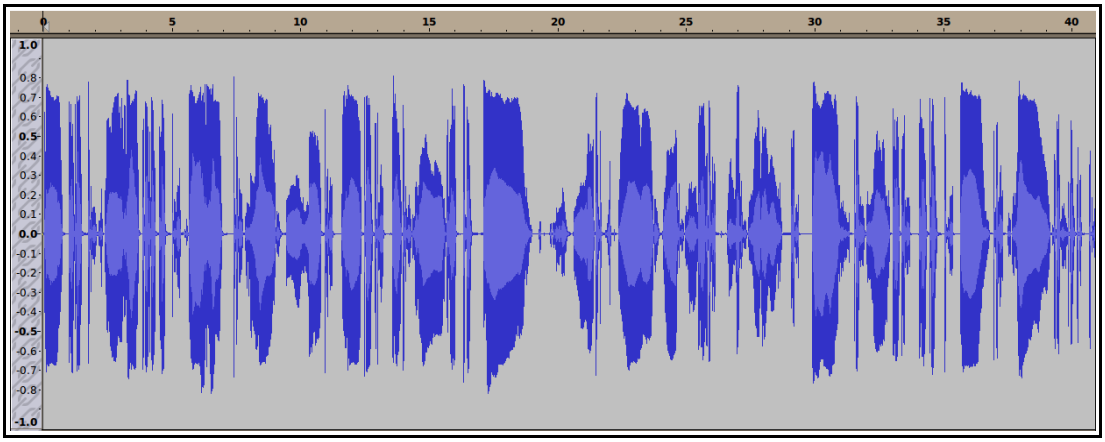
The sampling was done using BabyMonitor4 GUI program. And the comparison of the waveform was done with Audacity program (Audacity ® 1.3.12-beta (Unicode)).

File name: 28101__neonaeon__babycry2.flac

Graph



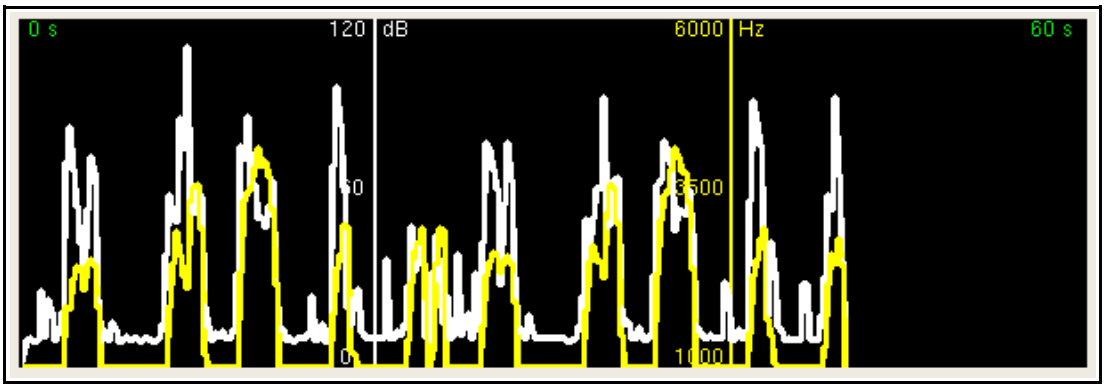
Audacity Output



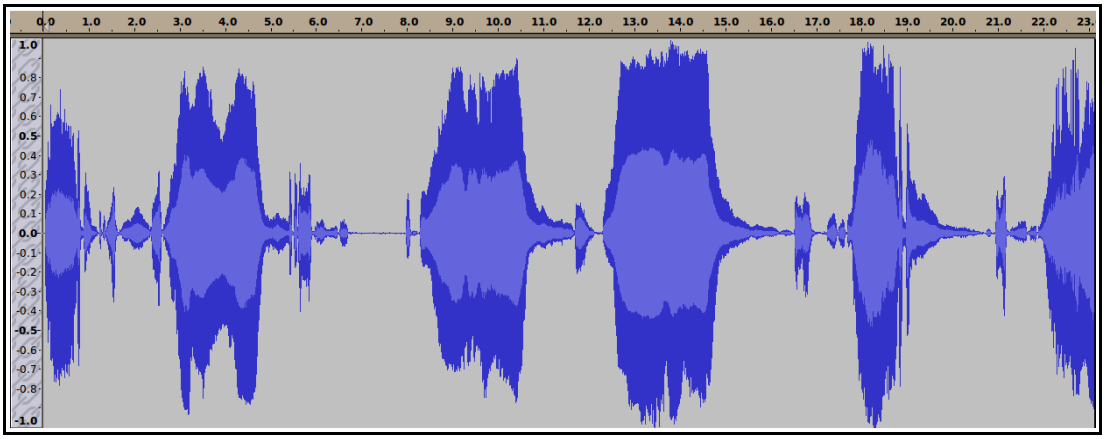
File name: 59578__morgantj__babycrying.mp3

Graph

Description: the audio repeated once during sampling



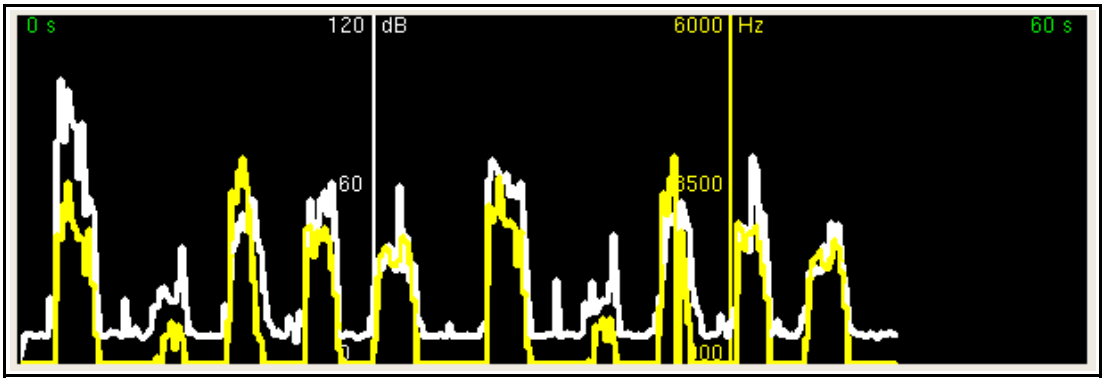
Audacity Output



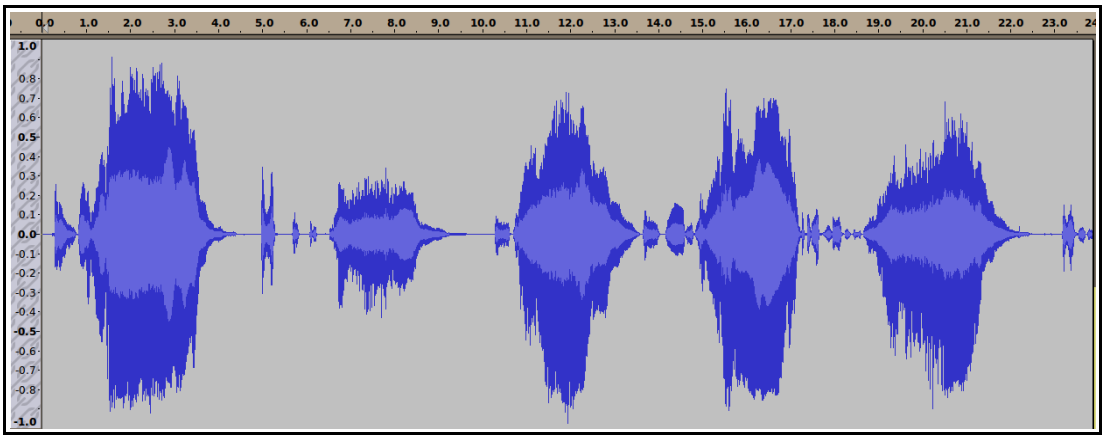
File name: 59579__morgantj__babycrying2.mp3

Graph

Description: the audio repeated once during sampling



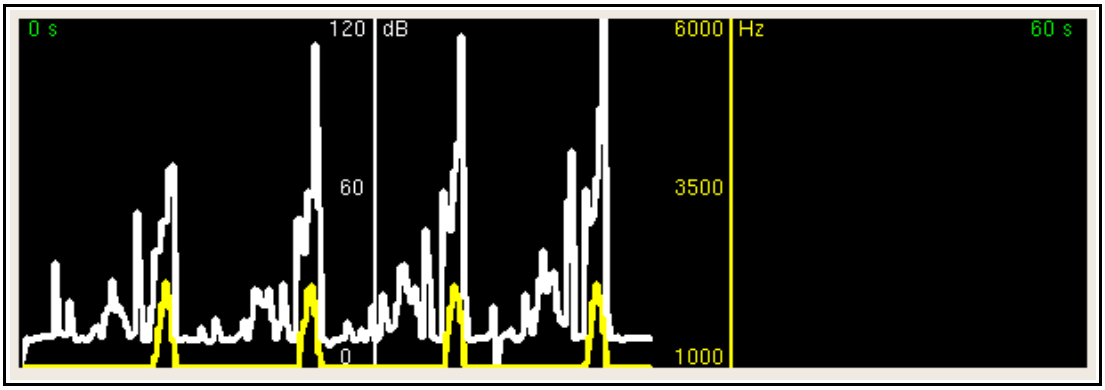
Audacity Output



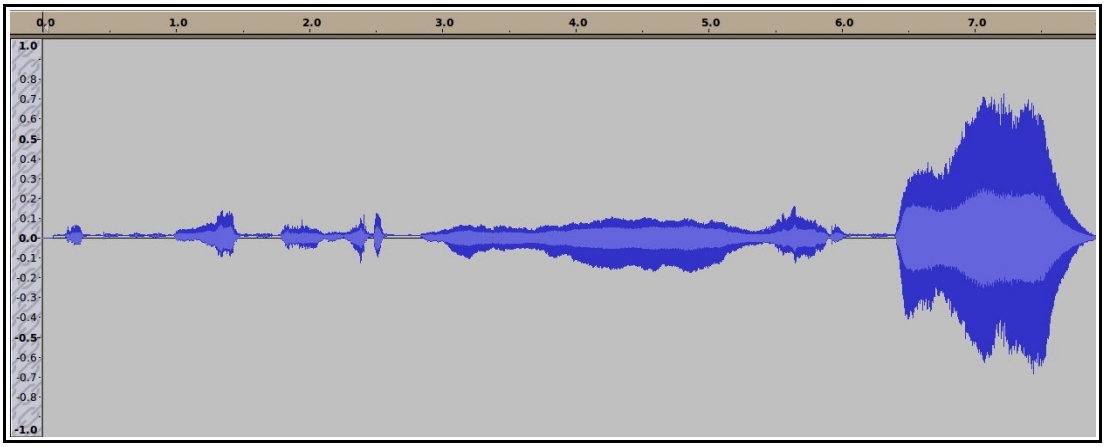
File name: baby_cry_1_(simplythebest).mp3

Graph

Description: the audio repeated three times during sampling

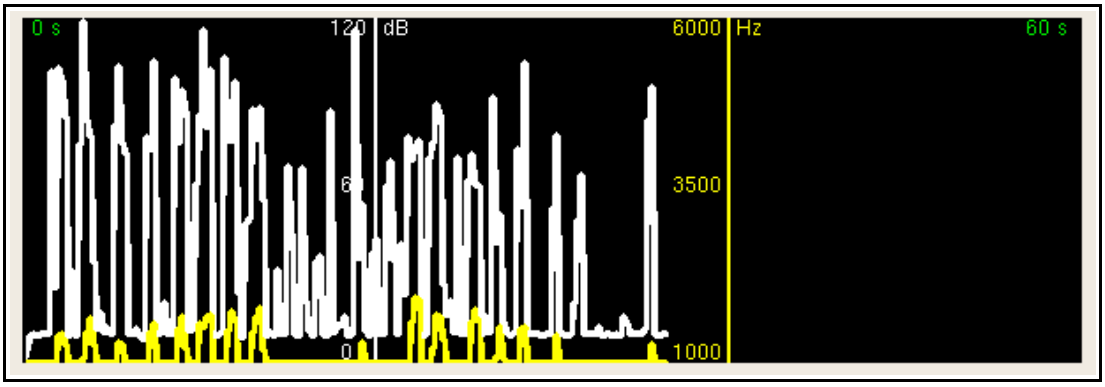


Audacity Output

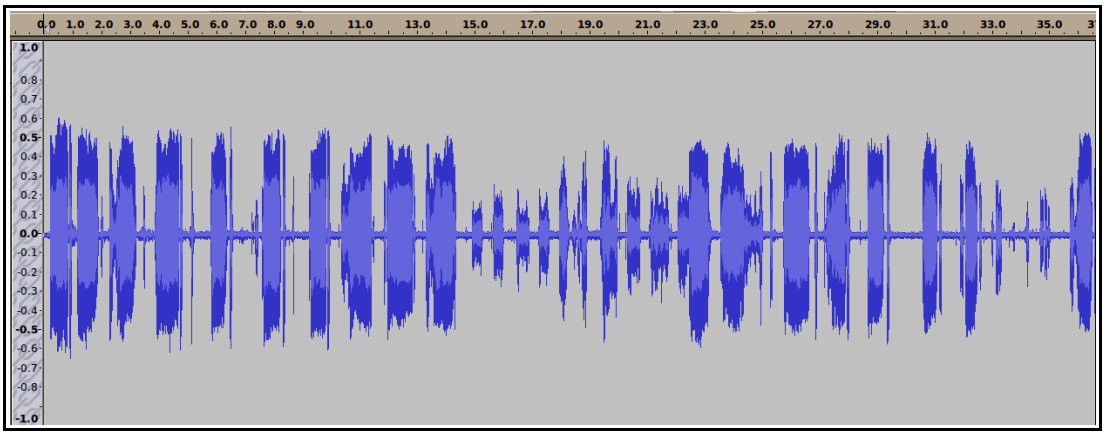


File name: Baby crying 38 sec_(dramatic_publishing).mp3

Graph



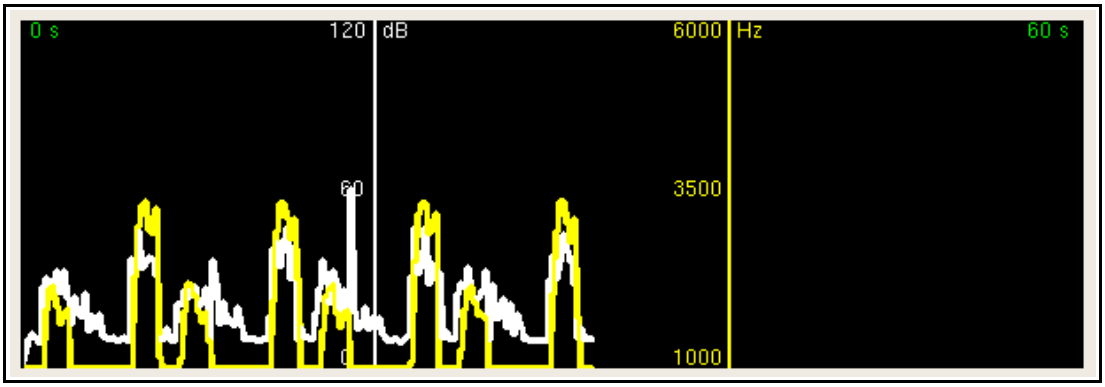
Audacity Output



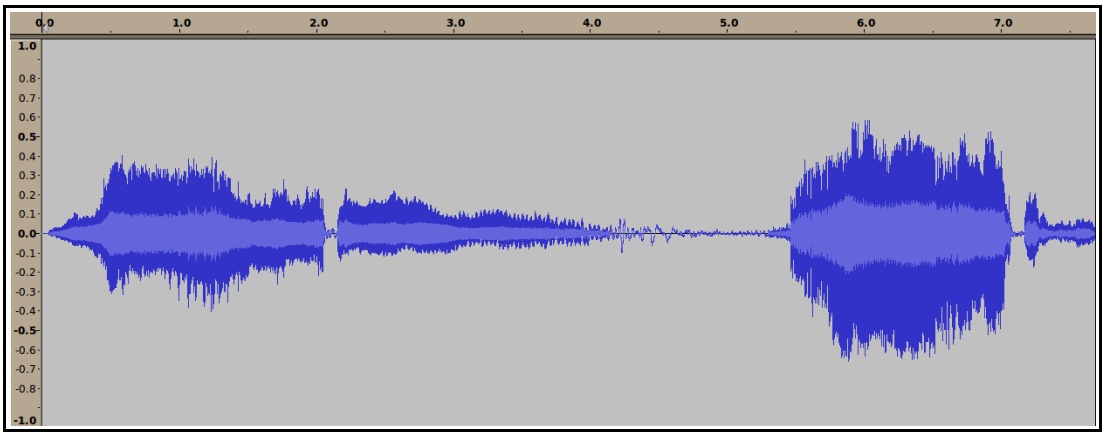
File name: Baby Crying-SoundBible.com-1143552027.mp3

Graph

Description: the audio was repeated three times during sampling



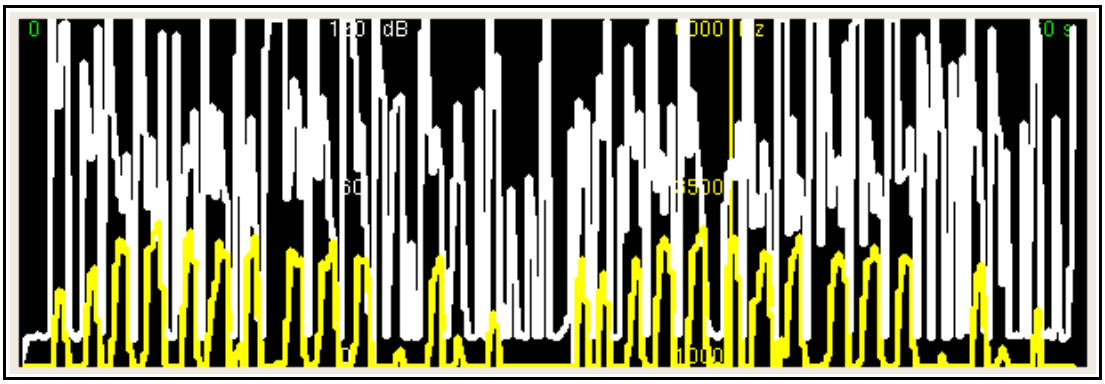
Audacity Output



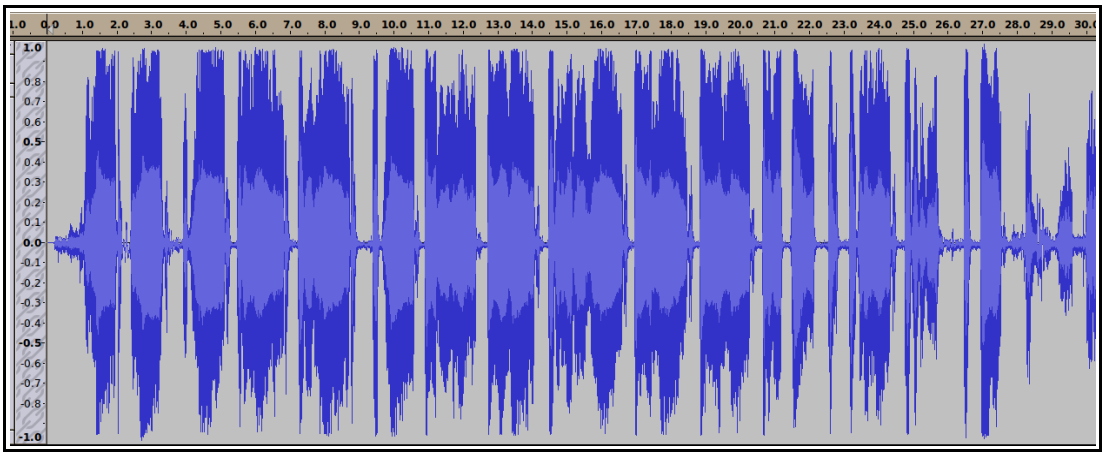
File name: baby_cry_(simplythebest).mp3

Graph

Description: the audio was repeated once during sampling

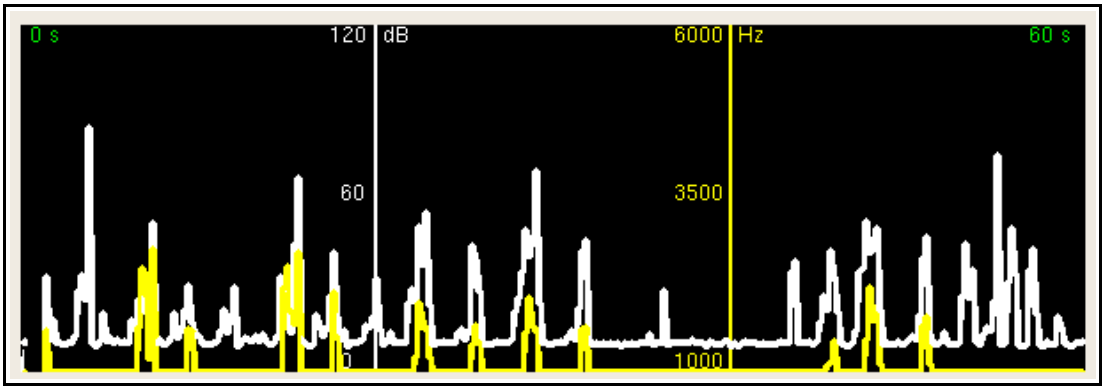


Audacity Output

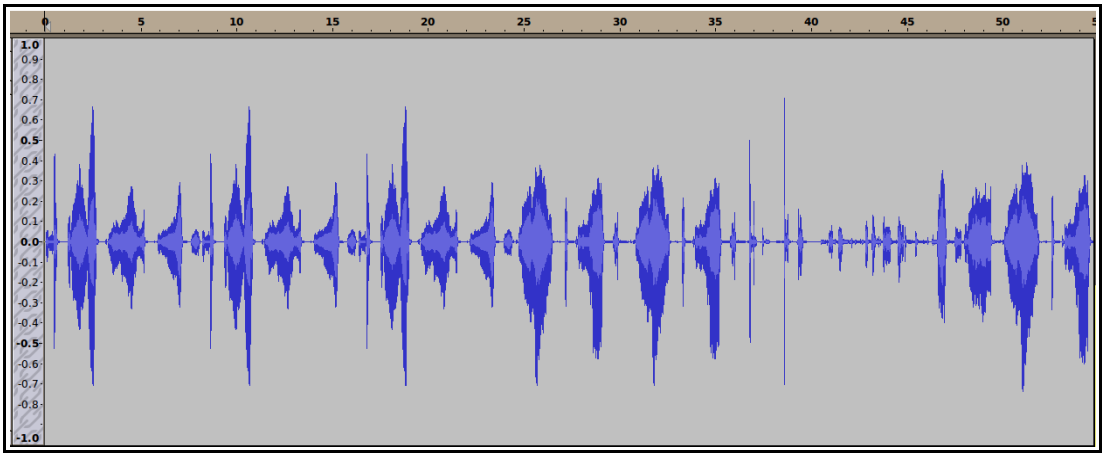


File name: 26760__zerolagtime__baby_crying1.wav

Graph



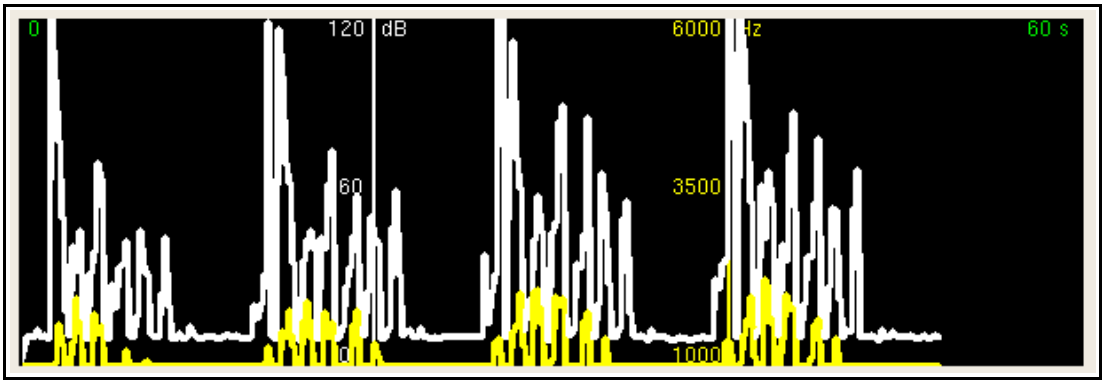
Audacity Output



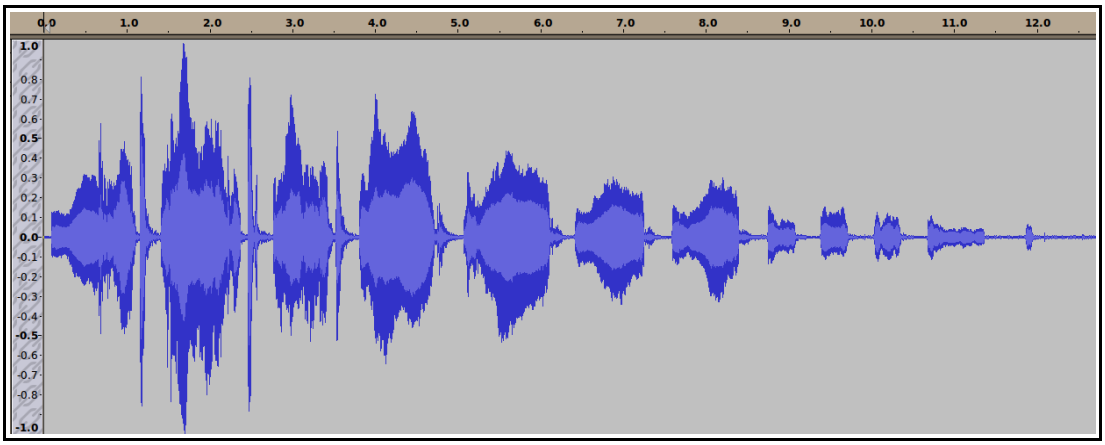
File name: 58178__Robinhood76__00235_baby_newborn_first_voice.wav

Graph

Description: the audio was repeated three times during sampling

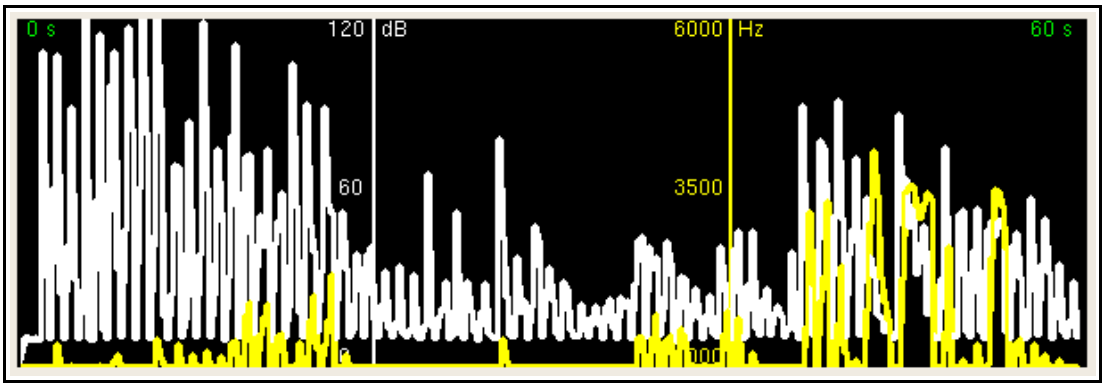


Audacity Output

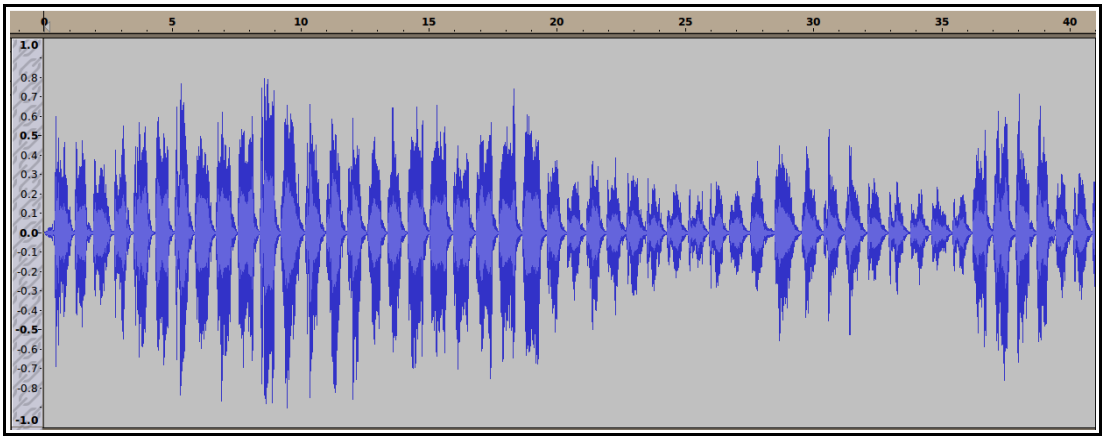


File name: 58741__Robinhood76__00255_baby_crying_long_1.wav

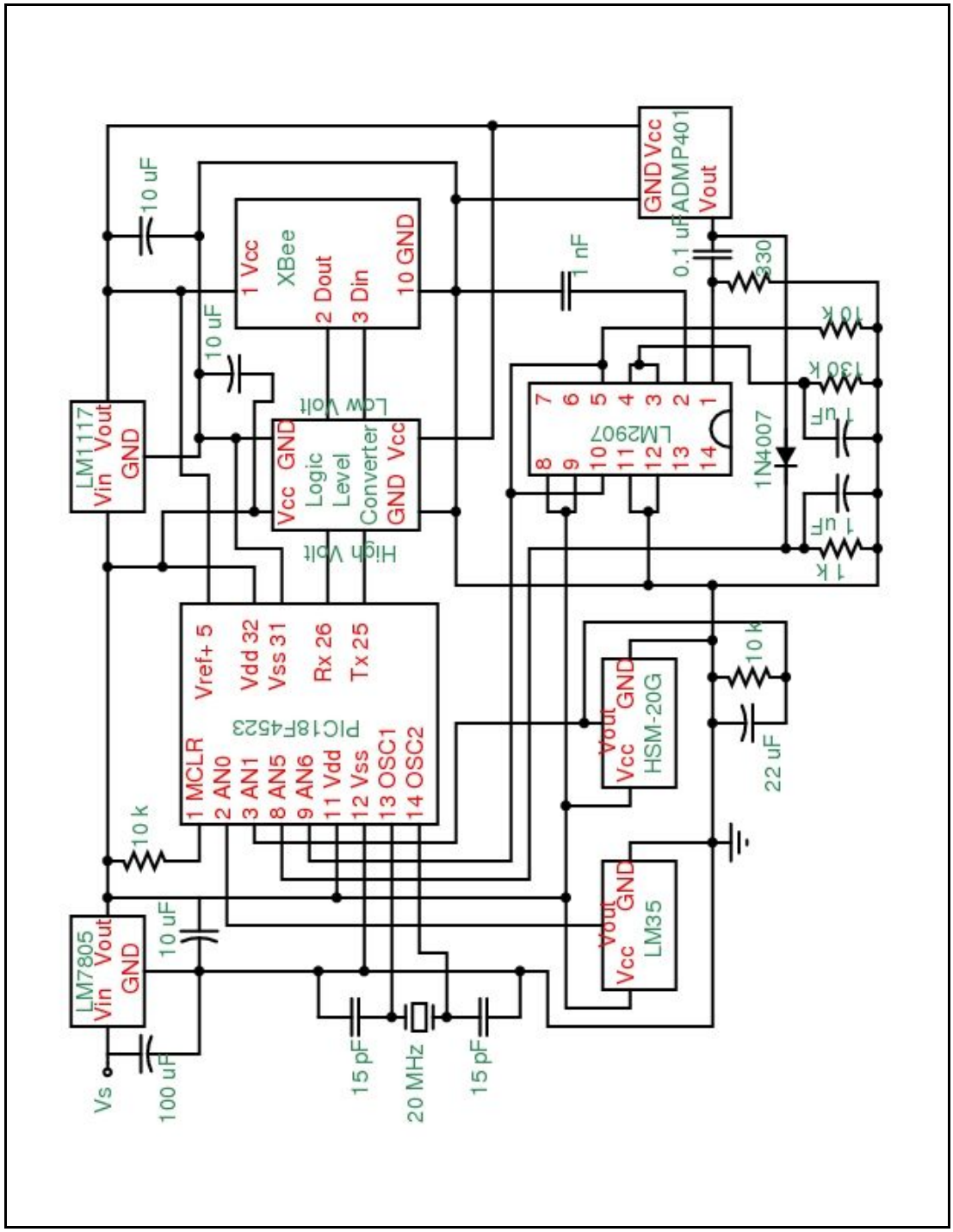
Graph



Audacity Output

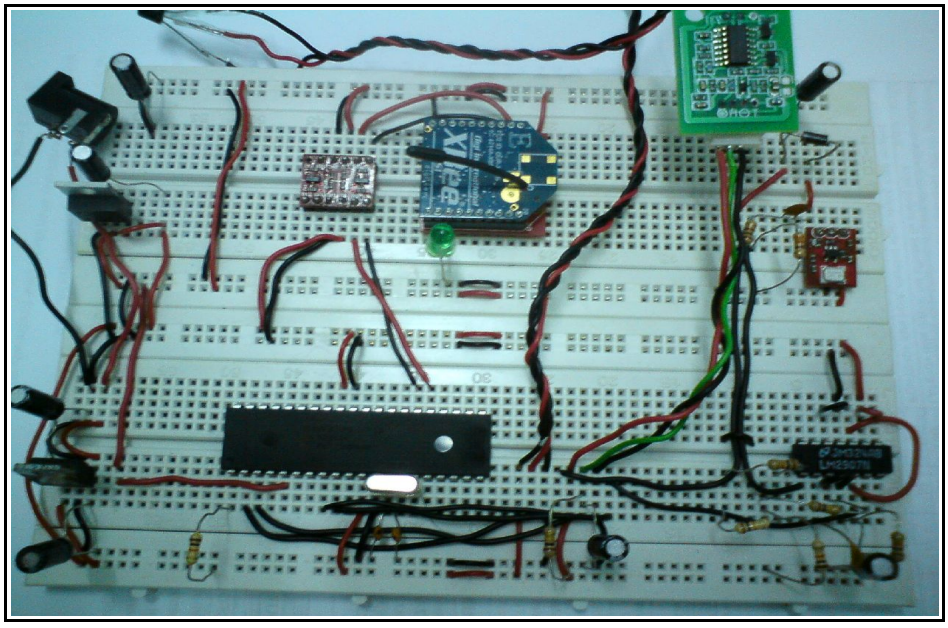


Appendix C: Full Circuit Diagram

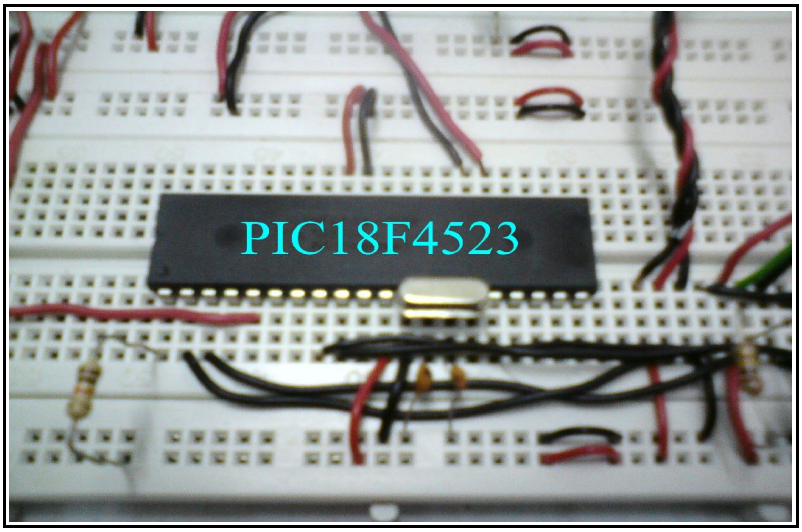


Appendix D: The Pictures of Circuit Connections

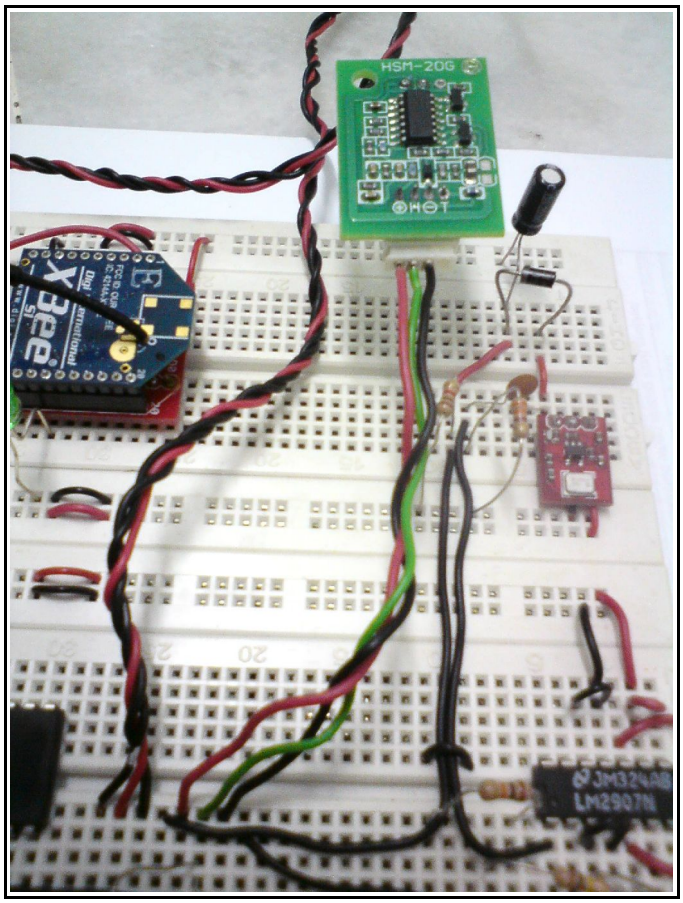
Full Circuit Connection



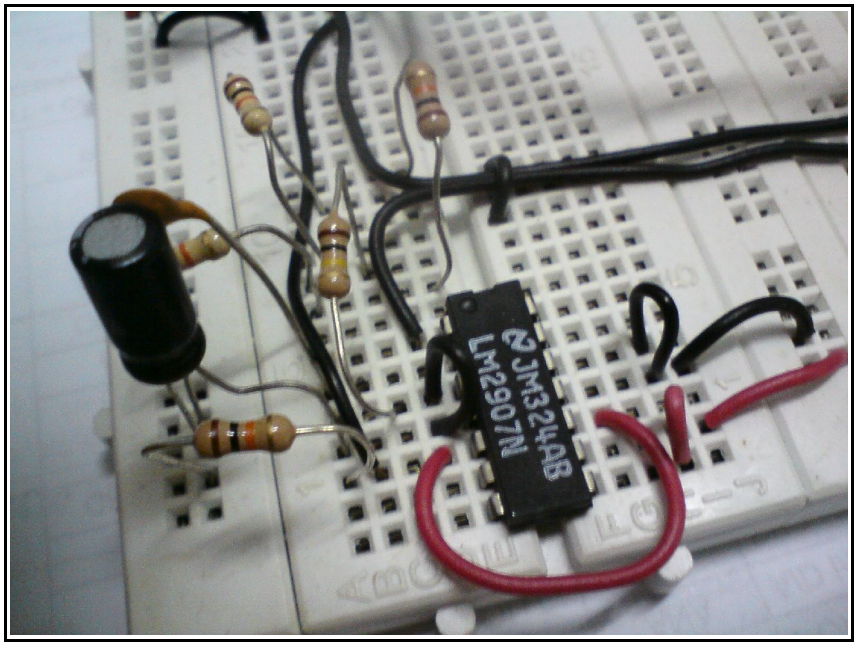
Microcontroller Connection (PIC18F4523)



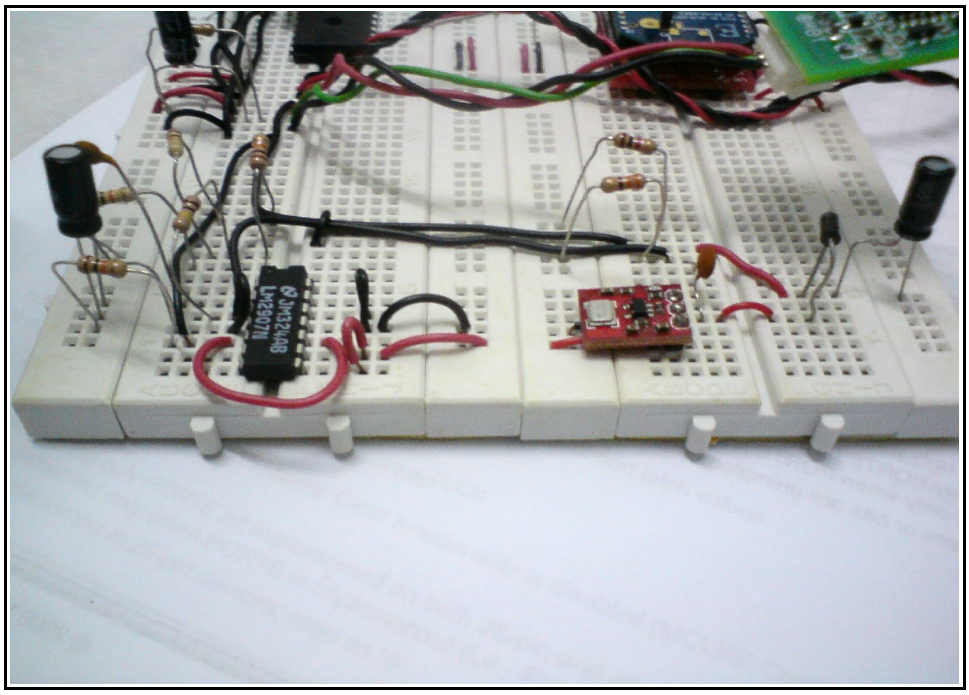
Humidity Sensor Connection (HSM-20G)



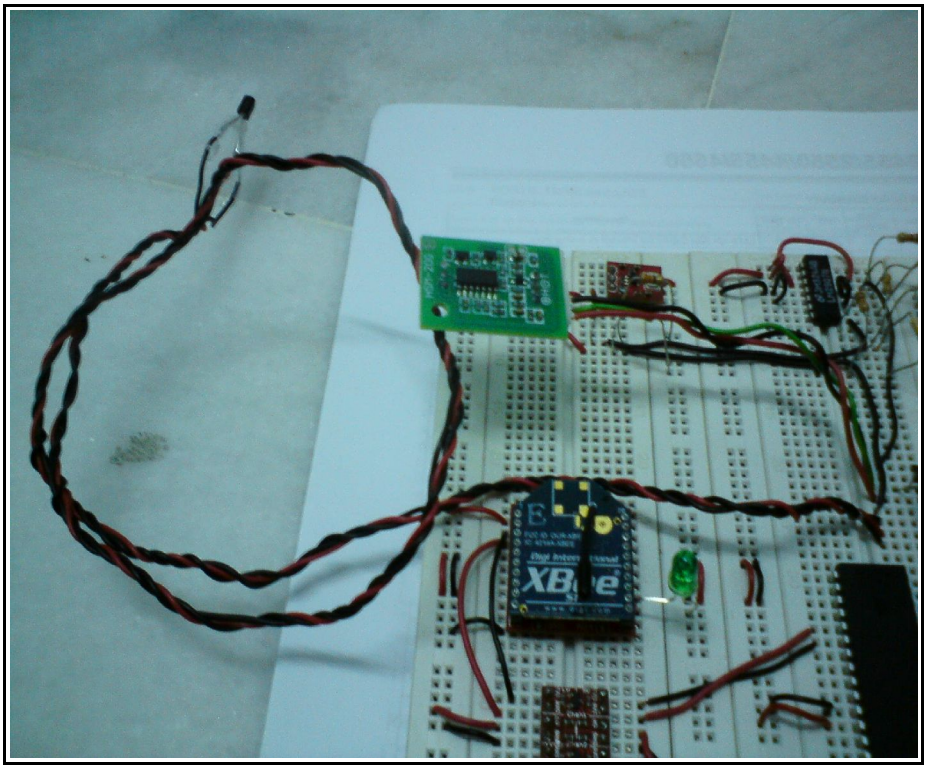
LM2907 F / V Converter Connection



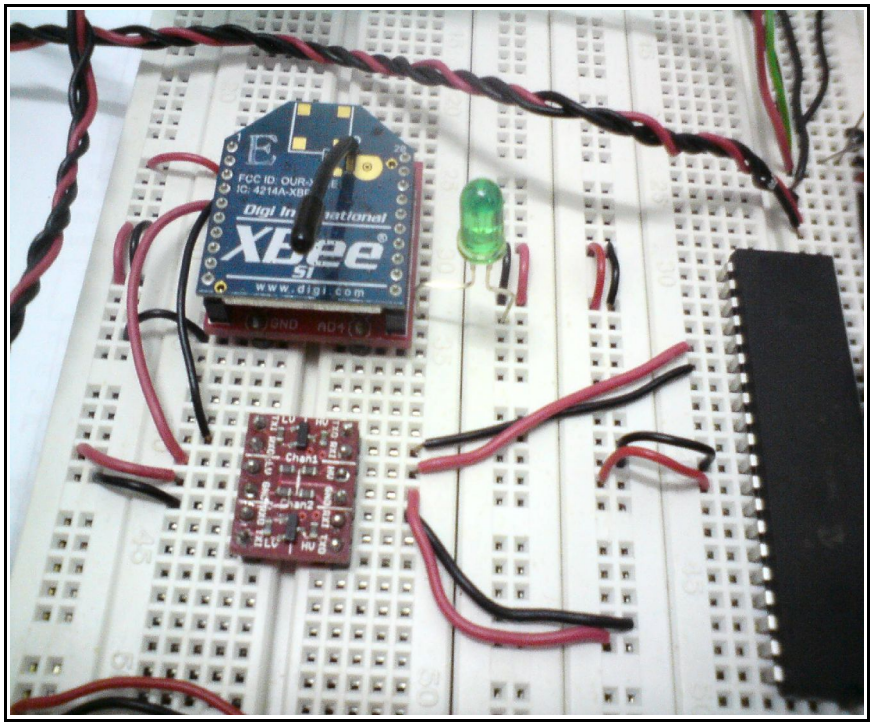
Microphone Connection (ADMP401)



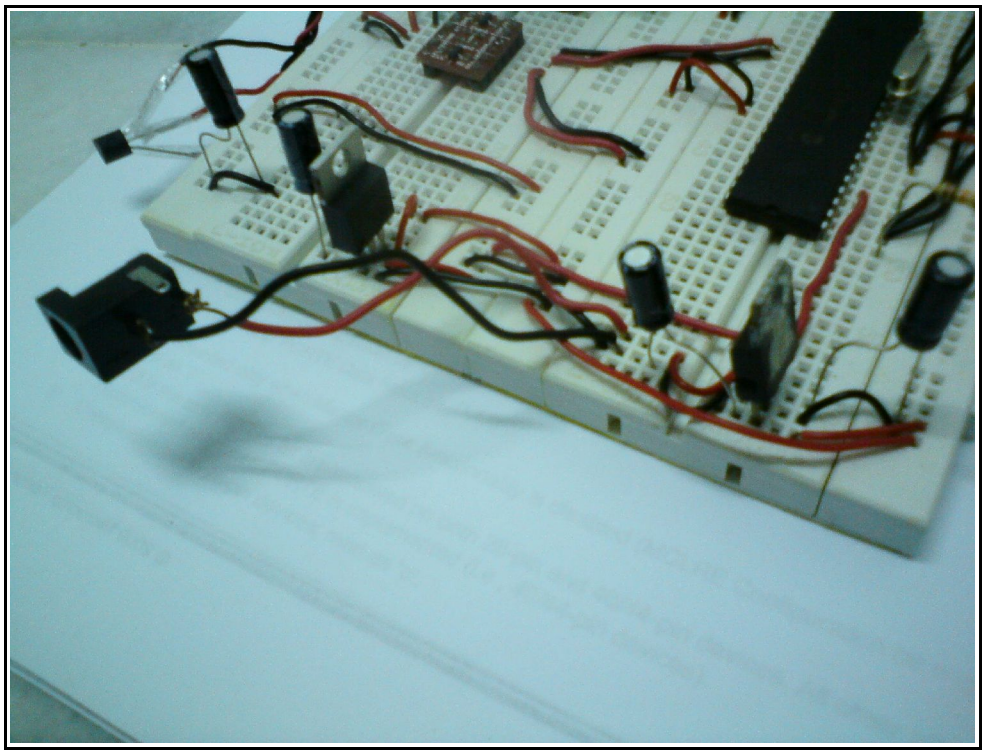
LM35 Temperature Sensor Connection



Logic Level Converter



Power Supply (LM7805 & LM1117)



(This page is intentionally left blank)