**Final Project**

**Simple Chatting Room**

**(Pseudo Skype)**

Carlos Huezo

ECE 4220- Real Time Embedded Computing

**Abstract:**

The purpose of this project is to create a chatroom that functions like Skype but in this case a program that will only broadcast audio through socket programming. I will be utilizing the external component of a microphone and having it amplified using an ADC converter that I will make on a breadboard. The signal will then be streamed into a port that multiple users can access and at the same time they will be able to speak and load their audio into that socket in order to broadcast to other users. In order for the users to hear the streaming audio they will need to take the signal and convert it using a DAC. This is necessary because when we talk, the audio signal is in analogue but computers can’t understand that we need to convert it into machine language. The ADC does precisely that, it will take in the analogue signal and convert it into bits, 0s and 1s. In contrary, the DAC will do the opposite and help us hear it.

**Introduction:**

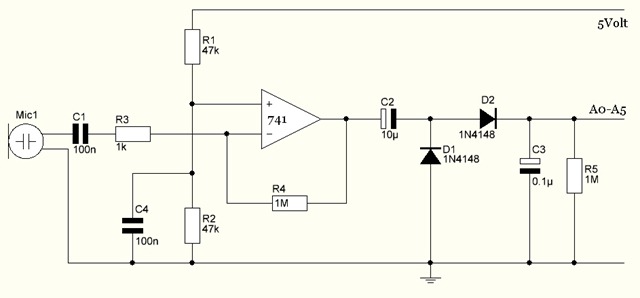
Social networking has become a large part of our society but sometimes a phone call requires us to wait on the phone and wait on the person to answer and if we don’t have any luck we will leave a voicemail. As an impatient person, sometimes I just want to leave a message without having to wait on the phone to ring. I thought of creating this Simple Chatting Room. The purpose of this project is to be able to load an audio message into a common stream and have other people respond to it. In a sense it will mimic the way a radio or walkie talkie functions. The following board, TS7250, will be used to take in the audio signal created by the microphone.

**Background:**

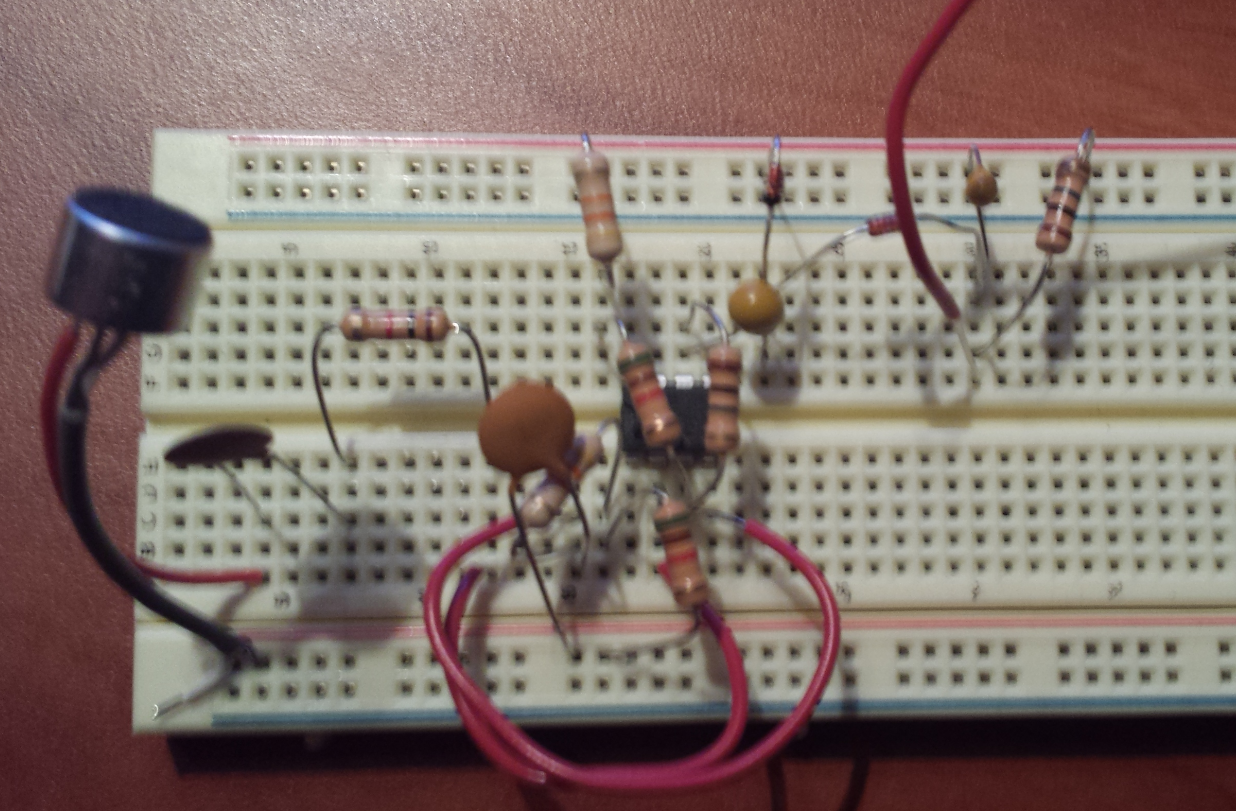
Some knowledge in circuitry will be utilized to build the ADC and DAC converters. The circuit will be used from an online source, so it won’t be generated from scratch. I have utilized the Raspberry Pi, so a basic idea of to use the ports and also the labs in our class will help me create this project. More specifically, I will try and reference Lab 5 and Lab6 because we deal with sockets and data streaming, not to the extent of audio data, but similar. Also I have learned and read by the course that TCP is the main way to approach the type of broadcasting because I want to have a constant streaming of data and if we get any cutoffs or distortion, then the server should still be able to stream the rest of the audio. If I utilized UDP broadcasting than there would be a higher chance of getting into streaming errors and higher chance of losing data because we may overwrite it and there wouldn’t be a way of retrieving it after its lost.

**Proposed Method:**

In order to start recording my audio into the system, I need to set up my microphone with an offset because the boards that we are using cannot read negative voltages that are coming from the signal. I first needed to do the offset using a simple circuit found online

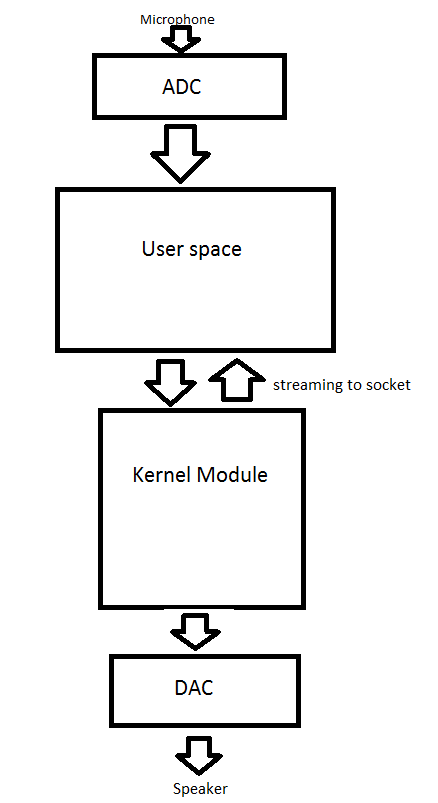


The following circuit is the actual circuit from above embedded in a breadboard.



In order to get a clean signal, I utilized the LM741 op amp. This helped me secure a clean and amplified wave to read into the board. This circuit allowed me to secure a voltage output centered at about 2.5 V.

I tried different ways to implement my code to read in the input data and I could not get the audio to stream into the board. I don’t know if I wasn’t offsetting the ports in the proper manner and the data was never reaching the socket. If I had received the data, my next step was to create my socket that will eventually stream the audio. The audio was going to be sent by a server programmed and within the server program then a kernel module was to take the data entered and broadcast it out through the output ports of the board and into the DAC converter.



**Diagram shows the flow of the signal**

**Results/Conclusion:**

Overall the project was halted an early stage because I could not get the signal to be read into the board correctly. I tried following a tutorial online of how to do it on a different board, Arduino omega. This was found in the following website: <http://www.instructables.com/id/Arduino-Audio-Input/>

Even though it seemed like a possible idea I ran short of time to try and implement it. I also tested the program and tried to run it in a UDP form to see if maybe my broadcasting was being affected by it being set up as TCP. It was interesting building the microphone circuit and seeing that it worked by plugging it into an oscilloscope and seeing the wave being offset enough for the board to take it in without having to deal with clipping. An alternative idea was to utilize the ADC port on the board to read in without having to serialize the analogue wave with a constructed ADC. Some sample code that I will attach will contain part of how the ADC is programmed. In the future think I will try and implement my project into a more mobile board that I can take around and be able to work on, instead of in between classes. Finding time to work on the board was very challenging because of the limited time that the linux lab is open, especially when we have a lab to work and to complete each week. This project has helped me have an idea and prepare myself with time management to be able to successful in Project and my Capstone. It also has given me a direction of an idea of what I want to build when those classes come around.

**Appendices:**

**USER SPACE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <time.h>

#include <ifaddrs.h>

#include <pthread.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <unistd.h>

//buffers storing address defined.

char addBuffer[INET\_ADDRSTRLEN];

char addBufferT[INET\_ADDRSTRLEN];

int sock, length, n;

int boolval = 1; // for a socket option

socklen\_t fromlen;

struct sockaddr\_in server;

struct sockaddr\_in addr;

char buffer[MSG\_SIZE]; // to store received messages or messages to be sent.

char bName[MSG\_SIZE];

char srandNum[10];

int flag=0;

int randNum=0;

char \*strToken;

int myBoard;

char \*boardNum;

int fd\_fifo;

int fd;

ulong \*ptr;

void myTask1(void);

void error(const char \*msg);

int main(int argc, char \*argv[]){

pthread\_t myThread1, myThread2;

//start FIFO

fd\_fifo=open("/dev/rtf/0",O\_RDWR);

if(fd\_fifo<0){

printf("FIFO can't be opened\n");

exit(1);

}

//soft interrupt starting

fd=open("/dev/mem",O\_RDWR);

ptr=mmap(NULL, getpagesize(), PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0x800C0000);

pthread\_create(&myThread1,NULL,(void\*)&myTask1,0);

//structure accessing for ip addressing

struct ifaddrs \* adStruct = NULL;

struct ifaddrs \*ptr = NULL;

void \* tmpAddrPtr = NULL;

//creates linked list of struct

//to access interface addresses

getifaddrs(&adStruct);

//points to structure of type if addrs

ptr = adStruct;

ptr = ptr->ifa\_next;

//IPv4 addressing commences through the structure sockaddr\_in

if (ptr ->ifa\_addr->sa\_family==AF\_INET){

//accessing the internet address sin\_addr

tmpAddrPtr=&((struct sockaddr\_in \*)ptr->ifa\_addr)->sin\_addr;

//takes address of family and returns a readable

//address of maximum address length [INET\_ADDRSTRLEN]

inet\_ntop(AF\_INET, tmpAddrPtr, addBuffer, INET\_ADDRSTRLEN);

printf("MY IP is %s\n", addBuffer);

}

sock = socket(AF\_INET, SOCK\_DGRAM, 0); // Creates socket. Connectionless.

if (sock < 0){

error("Error opening socket");

}

length = sizeof(server); // length of structure

bzero(&server,length); // sets all values to zero. memset() could be used

server.sin\_family = AF\_INET; // symbol constant for Internet domain

server.sin\_addr.s\_addr = INADDR\_ANY; // IP address of the machine on which

// the server is running

server.sin\_port = htons(atoi(argv[1])); // port number

// binds the socket to the address of the host and the port number

if (bind(sock, (struct sockaddr \*)&server, length) < 0){

error("binding");

}

// change socket permissions to allow broadcast

if (setsockopt(sock, SOL\_SOCKET, SO\_BROADCAST, &boolval, sizeof(boolval)) < 0){

printf("error setting socket options\n");

exit(-1);

}

// size of structure

fromlen = sizeof(struct sockaddr\_in);

pthread\_join(myThread1,NULL);

return 0;

}

void error(const char \*msg){

perror(msg);

exit(0);

}

void myTask1(void){

}

MODULE:

#ifndef MODULE

#define MODULE

#endif

#ifndef \_\_KERNEL\_\_

#define \_\_KERNEL\_\_

#endif

#include <linux/module.h>

#include <linux/kernel.h>

#include <asm/io.h>

#include <rtai.h>

#include <rtai\_lxrt.h>

#include <rtai\_sched.h>

#include <rtai\_fifos.h>

MODULE\_LICENSE("GPL");

RT\_TASK myTask;

RTIME period;

int init\_module(void){

//offset ports for ADC

//take in signal from socket

//broadcast the audio to the ADC to hear the sample

return 0;

}

//clean up function

void cleanup\_module(void){

//stop the rt\_timer and kill the task

stop\_rt\_timer();

rt\_task\_delete(&myTask);

}