The Visual Representation of Sound for the Hearing Impaired

By

Jonathan Brophy

Senior Project

Electrical Engineering Department

California Polytechnic State University

San Luis Obispo

2013

Table of Contents

Table of Figures

Table of Tables

Acknowledgments

I would like to thank my dad, mom, and sister who provide a constant stream of support. I would also like to thank Professor Smilkstein, of whom this project would not even exist if not for her brilliance and creativity.

I. Abstract

There are many difficulties that arise when one is faced with a hearing disability. New applications of advancing technology have the potential to enable designs capable of assisting the hearing impaired with the ability to see sound. This project seeks to provide a user with a visual representation of loud noises detected using small directional microphones. The microphones determine the location of the sound above a set decibel level and a microprocessor determines the approximate location of the sound source and displays a vertical colored bar on a video image in the direction of the noise using virtual reality glasses. Two sources of input are combined in the display: the sound that is measured with the microphones, and a camera video image of what the user sees. This information is processed using an ATmega 328P micro controller and the result is displayed on the screens of the user's virtual reality glasses.

II. Introduction

Technological advancements assisting people who are deaf have certainly come a long way, and this project aims to further these advancements by providing an improved visual representation of loud audio signals relative to the current sate of the art. A research group from the Korea Advanced Institute of Science and Technology (KAIST) in Daejeon has developed glasses with an array of LEDs on them that alert a deaf user where a loud noise is coming from, such as a car horn. This technology also allows the user to adjust the threshold level of the noise them selves. This senior project seeks to improve on this idea and to provide the user with an improved capability to detect and locate loud sounds. Instead of using glasses that only have LEDs on them, this project will use a camera that will capture what the user is looking at, and based upon the information from three small microphones, will overlay graphics onto the video signal indicating the approximate location of any loud noises. The user will see an image in their virtual reality glasses that gives them an accurate display of their surroundings, including the location of any noises above a threshold value. This method requires a computer to process the incoming information from the microphones, possibly resulting in a bigger, heavier design. A possible solution to this problem would be to wirelessly transmit all of the data from the microphones and camera to the computer and resend the information to the glasses, but such an enhancement is outside the scope of this project.

If successful, this project will create a system that uses a camera to display a video of what the user is looking at, and then alter this display of the world in real time to give the user an augmented reality in which the direction of loud noise sources are included. It is anticipated that people who are deaf would benefit from this piece of technology to enhance their health and overall lifestyle – seeing when and where a fire alarm is going off for example. In addition, even people with normal hearing could benefit from improved versions of this technology that would enhance their ability to determine precisely where a loud sound is coming from. For example, from a military standpoint a user could benefit from accurately seeing the location of enemy fire.

III. Background

This project exhibits the ability to locate the direction of sound above a certain threshold using three directional microphones and then sends this information to a computer. A simple surveillance camera is

used to take a video of exactly what the user is looking at. The video signal is processed and altered using a microcontroller based upon the input from the three directional microphones. A visual representation, such as a vertical colored bar on the user's screen, is displayed on the user's virtual reality glasses. This graphic is applied over the video signal unobtrusively to the user's view of their current surroundings. If the sound is persistent, the location of the bar will change as the user turns his head in the direction of the sound enabling the user to zero-in on the source of the sound.

IV. Requirements

The top-level requirements for this project are:

1. The information provided from the microphones shall determine a relative location of any sound greater than the set decibel level threshold.
2. The glasses shall display a vertical colored bar in the direction of the sound.
3. The information shall be processed within one second to display an accurate augmented reality.
4. This technology shall work where low levels of noise are present but do not trigger the glasses.

V. Project Description

A block diagram of the proposed concept can be found in Figure 1. As can be seen from this figure, microphones 2, 3, and 4 are pointed in roughly the same direction as the video camera (1). The side microphones (2 and 4) are pointed slightly to the left and right of the camera; all three microphones are needed to cover a wide physical area and to accurately pin point the location of the sound. In this drawing, item 5 represents the source of the sound. Item 6 is a high-level black box containing a Max7456 IC, three ADCs, and an ATmega 328P micro controller. The micro controller pre-programmed with C code written in AVR Studio 6.0 will process all of the video and sound information.

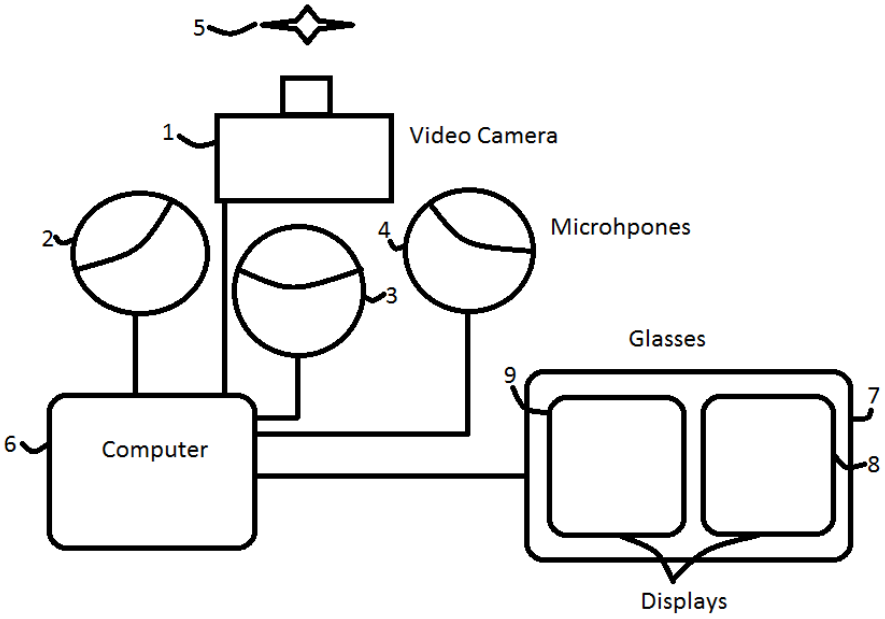

Figure 1 Graphic Patent Design

Items 7, 8, and 9 in Figure 1 represent the hardware that will display the re-constructed video signal after being processed by the micro controller. This display uses virtual reality glasses by Vuzix, which contain two small screens, one for each eye. Using a screwdriver, the user can adjust the positioning of the screens to create an image equivalent to one virtual 40" screen.

VI. Design

1. Overall Design

An overall system prototype design of this project is shown in Figure 2. The receiver circuit consists of the Max7456 IC to overlay the graphics onto the analog video signal. Three analog to digital converters, one for each of the three directional microphones take their analog inputs and turn them into digital data. The resolution of these ADCs must be good enough to determine relative location of the sound and fast enough to process the signal within a reasonable amount of time from when the sound occurred. Therefore, the built-in ADCs within the ATmega 328P micro controller have been chosen for this prototype design because they are 10-bit data converters, which give enough accuracy for the requirements of this project. The prescaler of these ADCs can be set to divide the clock of the ATmega 328P by 128. The micro controller has an internal clock speed of 16MHz, this value divided by the prescaler gives us a maximum sampling frequency of 125kHz. A typical ADC has a sampling frequency in the range of 50kHz to 200 kHz. These built-in ADCs will give us the speed and accuracy needed to quickly and precisely collect data from the external microphones. The fact that these ADCs are built-in to the micro controller also cuts down on the bulk of the hardware in this design.

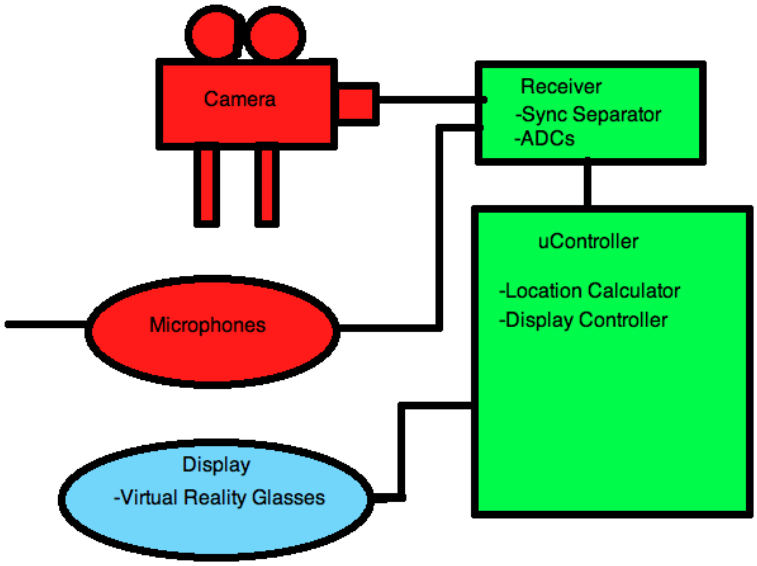

Figure 2 Overall Visualized Sound Block Diagram

The software logic programmed into the micro controller compares the amplitude of the signals from the microphones and uses this information to estimate the approximate location of the sound. The video signal coming from the surveillance camera is broken down into the horizontal and vertical syncs, all of this done inside of the Max7456 chip. Using this information, the hardware produces the original

video signal sent into it and inserts the desired graphics when told to, overlaying the graphics onto the video signal. The video signal is then sent to the glasses in composite video format.

2. Receiver/ Video Separator Design

The block diagram of the receiver/video separator circuit is given in Figure 3. The video camera output signal connects to the breakout circuit that houses the Max7456 IC. This signal will be manipulated by the graphics being overlaid onto it via the Max7456 IC from SPI commands via the Atmega328P micro controller. The output video signal that has the overlaid graphics will then go to the virtual reality glasses and get displayed on that venue.

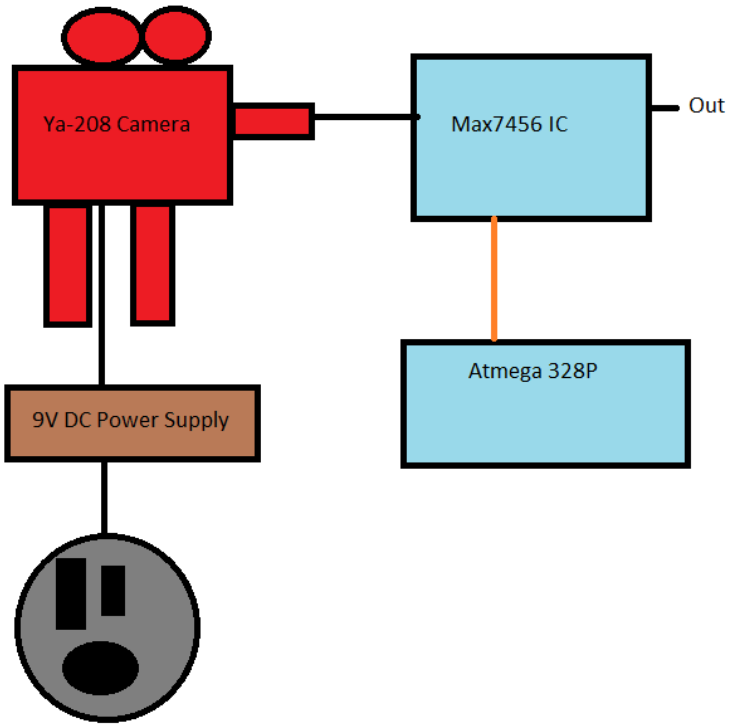

Figure 3 Receiver/Video Separator Design Block Diagram

The small Ya-208 surveillance camera needs in between +6V and +12V to operate and has a range of 200mA to 500mA in order to be operable. The camera outputs NTSC analog video, and the Max7456 IC can handle NTSC or PAL, and the virtual reality glasses have the same capability as well.

3. Receiver/Location Calculator

The block diagram of the receiver/microphone to ADC circuit can be found in Figure 4. Each microphone uses a separate analog to digital converter to process the incoming noise. The microcontroller calculates the position of the sound based upon the data from all three ADCs. The microphones will be in a fan-out position with the middle microphone stationed directly above the camera, and the two other microphones on each side of the camera facing outward at a forty-five degree angle.

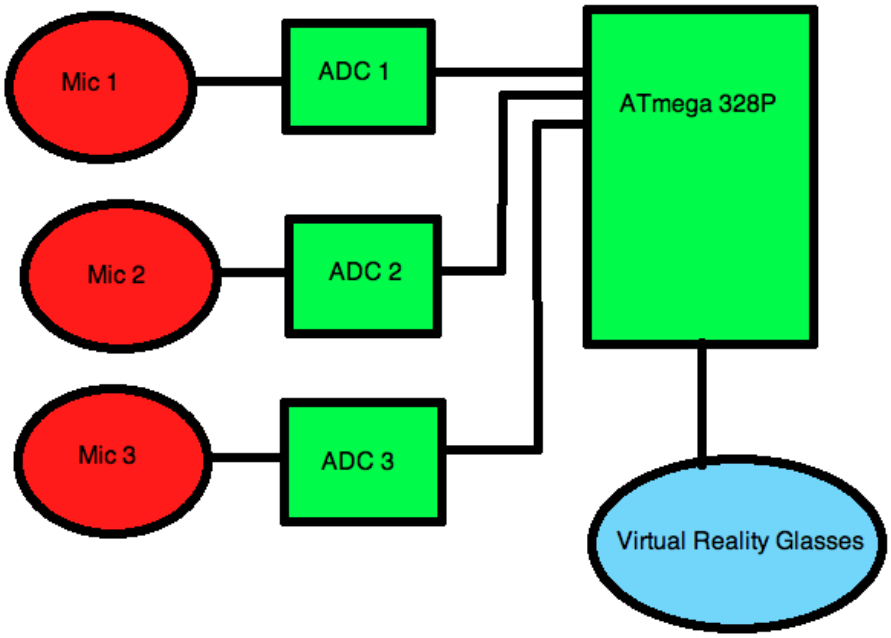

Figure 4 Receiver/Location Calculator Block Diagram

Audio cables are connected to each one of these microphones to measure the amplified sound from each microphone. The audio cable can be spliced open to measure the current or voltage generated when sound is picked up by the microphone. The voltages are converted to binary values using the ADCs and processed by the microcontroller. Logic in the C code software determines which microphone has the highest binary value and estimates where the sound is coming from in relation to the camera and microphone positions.

The location of the sound is then used to decide where to overlay the graphics in the video signal. Using the horizontal and vertical syncs as references, the software injects the sound information into the video signal using the Max7456 IC to display a vertical colored bar in the proper location on the screen. This location is based upon the information from the microphones. The new video signal is then sent to the virtual reality glasses display using another composite cable.

4. Software Code Design for Location Calculator

The general flow of the software code in determining the approximate location of the sound is shown in Figure 5. After the input and output ports are initialized, the level of steady state background noise is measured to act as a baseline. After 1000 samples have been taken (each sample takes about 15 clock cycles or about 1mS) using the microphones to find this baseline, the system is ready for use. Only noise above this baseline will be displayed on the screen, otherwise the user's screen would be constantly filled with unnecessary vertical bars. With this algorithm, the threshold can be set to any value. In practice only sounds that are excessively loud should trigger a display response.

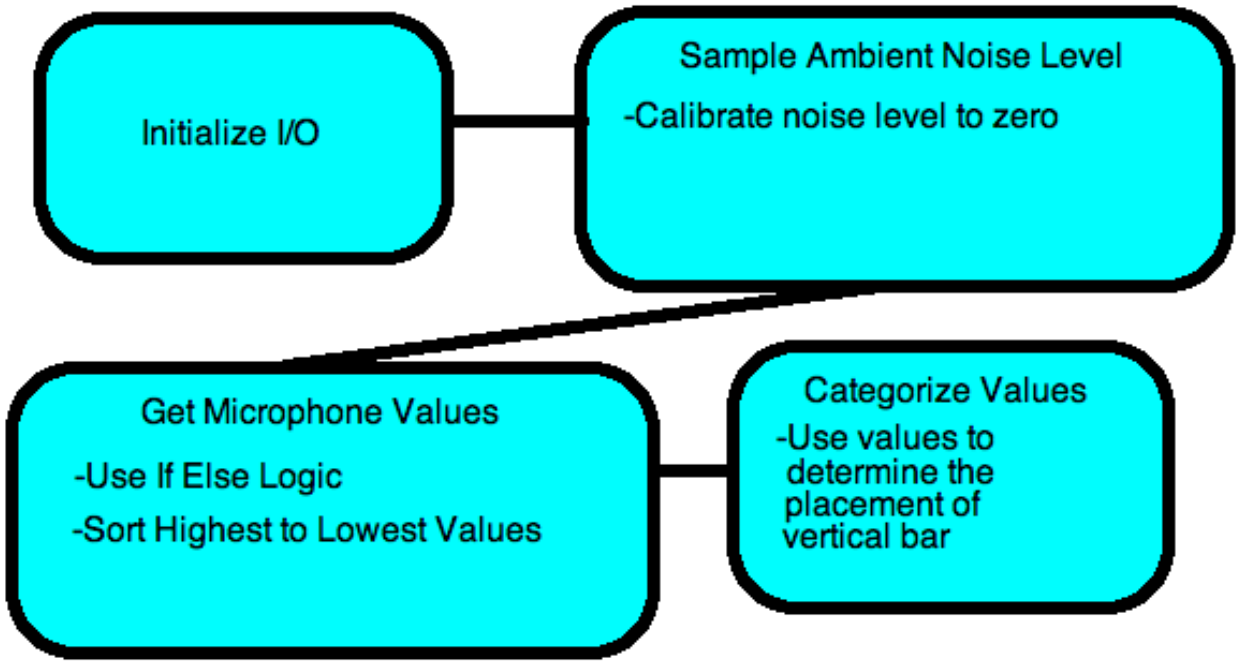

Figure 5 Software Code Design for Location Calculator

After setting the threshold voltage, the software will sort through the values that each microphone picks up from the depicted noise and determine the highest value. With this data, the software can determine if the sound is coming from straight ahead, to the left, or to the right. With some fine-tuning, the software can use the data from multiple microphones to get an even more accurate position of the sound, such as between the left and center microphone, or between the right and center microphone.

5. Software Code Design for Altered Video Signal

The general flow of the software code to overlays graphics onto the video signal is given in Figure 6.

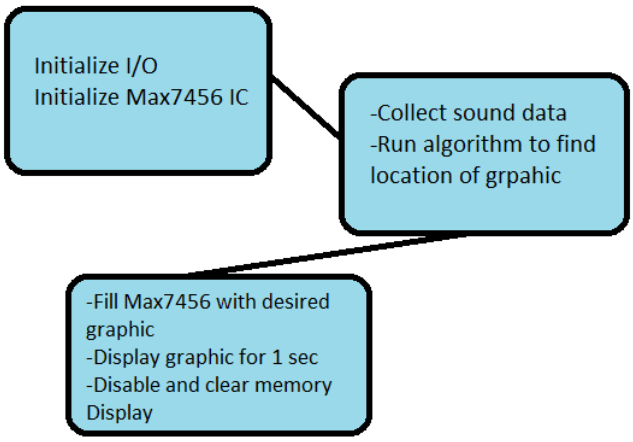

Figure 6 Software Code Design for Altered Video Signal

When the program executes, a few write and read operations will take place in order to initialize the Max7456 chip and the outputs needed for SPI communication to the IC via the micro controller. Once that is in order and the screen is adjusted to fit the virtual reality glasses nicely, data can be collected from the microphones. After the location algorithm takes place and the location of the white line has been decided, writes operations to the Max7456 IC will fill the display memory with code to display a white line in the decided area. Once this memory has been filled, the screen will be enabled for one second, and then disabled and the display memory will be cleared, ready for the next loud sound.

VII. Test Plans

Before trying to connect the entire circuit together, each component must be tested separately to ensure the functionality of each piece of the circuit. This method enables problems to be detected early and also helps the designer understand the function of each part.

1. Surveillance Video Camera Test

This is a simple test to ensure the camera is in working condition. After connecting a DC voltage in the range of +6V to +12v – a 9V adapter is used for this test – the composite video out cable is connected to an old cathode-ray-tube television monitor. The television monitor picks up the signal and outputs the video display. The next test is to connect the surveillance camera video output to the input cable connected to the Vuzix reality glasses. The glasses should pick up and display the signal just as the television did.

2. Amplifier Test

When each microphone detects a noise input it outputs an analog signal through an audio cable. The output voltage ranges from 0-300mV. In order to easily discern which microphone has a higher voltage level, the output is amplified using an LM741 chip. This integrated circuit contains a low power audio amplifier to boost the signal before it is sent to the analog-to-digital converter. To test the amplifier, a non-inverting configuration is applied: the negative input terminal to the op-amp is grounded, the positive input terminal carries the signal, and a feedback resistor network determines the gain. The gain is set at 1001 using Rf = 100kΩ and Ri = 100Ω; A (Gain) = 1+Rf/Ri = 1001. The supply rails are set at +5V and ground. For a 10mV signal on the input, the output should be about 10.01V.

3. ADC Test

The analog-to-digital converters on the ATmega 328P μ-controller each output 10 bits of binary data, corresponding to the resolution of the ADC. The range of values each of these ADCs are capable of producing are between 0 and 2^10. Two tests are conducted. The first test consists of hooking up ten LEDs to the microcontroller and outputting the result to the LEDs. The second test involves the use of an LCD screen where the result is displayed as an integer value on the screen from 0 to 1024.

4. Microphone Sensitivity Test

To test the sensitivity of the microphones, an LED is connected to the microcontroller, as well as the amplified input signal – amplified by the LM741 chip – from the microphone into one of the ADCs in the ATmega 328P. The microphone now receives sounds, which pass through the audio cable, get amplified

by the LM741, and are converted into digital data. These data bits allow the software to determine a baseline for steady background noise by taking an average of 1000 samples. Once the baseline is set, 1000 samples are taken and averaged, and this value is tested against the baseline value. If this value is greater than the baseline value + 10 then the LED will turn on, otherwise the LED remains off. Ten is equivalent to 1010 in binary and is chosen because we only want loud noises to turn the LED on. In this test, If 10 is added to the baseline value, this provides a higher threshold and the microphone must pick up a louder noise in order for the LED to turn on. This process of sampling the noise and comparing it to the threshold continues indefinitely while the baseline value is taken only during the beginning of execution.

5. Graphics Overlay Test

The video signal from the camera connects to the Max7456 chip. This chip takes in the analog video signal, overlays the graphics sent to it from the micro controller (there are preset characters in the Max7456 datasheet that can be communicated to the Max7456 chip through SPI), and the new video signal is sent to the virtual reality glasses . This test will comprise of writing my name to the Max7456 IC, overlaying the letters onto the signal and centering it onto the screen of my virtual reality glasses.

6. Microphone Algorithm Test

Three LEDs are setup, one corresponding to each microphone. The three microphones are setup in the positions suggested in Figure 1. The same technique described in Figure 5. is implemented in software to determine the location of the sound. In this case, the microphone receiving the loudest noise means it is closest to the source and its corresponding LED will turn on. This algorithm can be expanded upon to incorporate more LEDs, corresponding to the positions between microphones and possibly behind the microphones' field of 'vision'. In addition, different "viewing" angles for the microphones will be investigated to determine the orientation that produces the most reliable detection of the location of the sound source.

VII. Development and Construction

This section covers each component from a design perspective and the functionality of each piece.

1. Max7456 Breakout Circuit

Courtesy of Sparkfun electronics, this circuit provides an important service to the overall goal of this project. Included in this breakout circuit is the Maxim 7456 IC. This IC makes life much easier when overlaying graphics onto a video signal. The chip communicates using SPI, providing the ability to write commands to the chip to overlay the desired graphics onto the video signal. The breakout circuit takes a composite video signal in, overlays the graphics onto the signal, and outputs the resulting video signal in composite video format.

Table 1 Max7456 IC to System Pin Out

Max7456 IC	Micro Controller	YA-208 Camera	Vuzix Glasses
RESET	PORTB: PIN 0	-	-
CS	PORTB: PIN 2	-	-
MOSI	PORTB: PIN 3	-	-
MISO	PORTB: PIN 4	-	-
SCK	PORTB: PIN 5	-	-
+5V	+5V	-	-
GND	GND	-	-
Composite Video In	-	Composite Video Out	-
Composite Video Out	-	-	Composite Video In

The chip communicates via Master Slave with the micro controller, and the pin out for these connections and everything else the Max7456 is connected to can be found in Table 1 above.

2. Vuzix Virtual Reality Glasses

The display is one of the most essential parts of this project. As described above, these glasses contain two mini LCD screens, and when adjusted to the user, can produce a virtual screen analogous to a 40" inch screen sitting 10 feet away. The glasses take in a composite video format, in this case it is NTSC (National Television System Committee), but PAL (Phase Alternating Line) can also be run on these displays. The Max7456 chip can support NTSC as well as PAL, but NTSC was chosen for this project because our surveillance camera outputs video in NTSC.

3. YA-208 Surveillance NTSC Camera

This camera provides NTSC video output as well as audio, but the audio aspect of this camera is ignored for this project. Perhaps a future project could utilize the audio to incorporate into this design and could do away with the external microphones, though the microphones inside the camera would need to be able to sample different areas of the user's surroundings. The resolution of this camera is 628 pixels by 582 pixels.

4. Handheld Parabolic Microphone

The ability to see sound start with these microphones, they collect the necessary data to be processed in order to deliver the accurate location of sound. The parabolic dish helps to make the design unidirectional, theoretically taking the sound in from the direction the microphone is pointed in. The output of the microphone is an audio signal, generally listened to with headphones. This project aims to take a different route. Instead, the analog audio signal is amplified by an external circuit and passed through an ADC (analog to digital converter) where the digital amplitude can be compared to the digital amplitudes of the other microphones 2 and 3.

5. Amplifier

Output from the microphone is generally a very small audio signal, and is always amplified with headphones making it accessible for humans to hear. The amplifier used in this project to get manageable digital amplitude is an LM741 IC. Each microphone has an individual amplifying circuit. Each amplifying circuit has non-inverting bias and a feedback resistance of 100k and input resistance of 100.

The theoretical gain of this amplifier is thus: Gain = Rf/Ri = 100k/100 = 1001. The output of this circuit is passed to the ADC.

6. Analog to Digital Converter

Each microphone's analog data needs to be converted to digital values in order to compare them using the microcontroller. The ADC's used for this job are already integrated on the Arduino Uno Microcontroller. This micro controller contains five pins to use for analog to digital conversion. The pre-scalar set on these 10-bit converters is 125kHz. These ADC's are used to eliminate the need for more external circuitry. Also, they provide are easy to initialize and obtain values from since they are already integrated into hardware of the micro controller.

7. Atmega 328P Micro Controller (Arduino Uno)

This Atmel microcontroller was chosen because it provides a wide variety of functionality for this project. This project hinges on the delegation and retrieval of data from the micro controller. It is wired to just about everything in the project. It communicates via SPI to the max7456 IC, converts the microphones data to digital values, and sets the supply rails on the amplifying circuit. All of these functions can be controlled using the AVR Studio software written in C. C is a higher level language than assembly or VHDL, which makes writing and debugging easier. This accessibility makes this micro controller a valued choice for this particular project.

Table 2 Microcontroller to System Pin Out

Micro Controller	Max7456 IC	Amplifier Circuitry
PORTB: PIN 0	RESET	-
PORTB: PIN 2	CS	-
PORTB: PIN 3	MOSI	-
PORTB: PIN 4	MISO	-
PORTB: PIN 5	SCK	-
PORTC: PIN 0 (ADC 0)	-	Output 0
PORTC: Pin 1 (ADC 1)	-	Output 1
+5V	+5V	Supply Rail
GND	GND	GND

The connections from the microcontroller to ever y piece of the system it is connected to can be found in the pin out in Table 2 above.

VIII. Integration and Test Results

This section covers the integration of the entire system and the results produced from the various components of the total project.

1. NTSC Surveillance Video Camera Feed

The first step in integrating all of the components together starts with the surveillance camera. Once the microcontroller is connected to the PC and pre-programmed with C code that covers basic I/O in order to communicate via SPI to the Maxim 7456 IC, the camera output connects to the maxim chip input.

After powering the microcontroller and connecting the output of the maxim to the input of the glasses, the first step to integration is ready for testing. A few SPI writes to the maxim chip initializes the device and feeds the input video signal through, verification of a successful video feed with no overlay can be seen on the virtual reality glasses.

2. Maxim 7456 IC

Once a successful video signal stabilizes on the glasses, the next step involves writing to the screen and overlaying graphics to provide the user with pertinent information about the device they are using. Using the datasheet for the maxim 7456 IC, the chip provides the entire alphabet in upper and lower case to be written to the screen, as well as many other useful symbols to relay information to the user. In this project, text is overlaid to the screen to welcome the user and relay information about an initialization phase that needs to take place when the program first boots up. Writing and clearing graphics onto the screen is very easy and efficient using this IC and has been an excellent choice for this project.

3. Microphones

This step involves the microphones, amplifying circuitry, and ADC's. Implementation is done one microphone at a time, where the output from the parabolic microphone feeds into the LM741 audio amplifier, and the output of the amp into one of the ADCs of the microcontroller. When the program starts, 1000 samples are taken from the single microphone and averaged to represent a base voltage. A decimal value is added to this base voltage to give a threshold digital value. For the duration of the program, samples are constantly taken from the microphone and if, after converted to digital data, the amplitude of the signal exceeds the threshold value, a white bar is produced and overlaid onto the video signal in the middle of the screen. After calibration through trial and error, this step of the integration process is a success.

The next logical progression involves adding a second microphone. The second microphone's output runs into a second LM741 audio amplifier biased the exact same way as the first. When the program starts, 1000 samples are taken from each microphone and averaged together, giving the base level. For the duration of the program, samples are taken from each microphone and the amplitude of their digital values is compared to the threshold value and each other. Essentially, the algorithm is as follows: If one microphone is greater than the threshold, then the microphone's value is compared against the other microphone's value. Based on how much higher it is, the white line will appear farther to one side of the screen than the other. Calibration of this step is much more difficult as the analog values produced from the microphones are very small, which makes converting their values and comparing them accurately a great challenge. No testing with a third microphone has been conducted since two microphones acting at the same time were not very effective. Also, each microphone's trigger needs to manually be pulled, making testing with three microphones an even greater challenge.

4. Amplifier

A Non-Inverting bias connected the microphones to the ADCs through these LM741 audio amplifiers. The audio signals coming from microphones are in the arrange of 0-300mV, after amplification, the values shot up to a couple a of volts. The vast sensitivity of the microphones made it difficult to capture

an accurate estimate of their levels to be compared quickly enough to accurately portray the direction of sound on the screen with great and reliable quality.

5. ADCs

The built in 10-bit ADCs in the microcontroller have 5V as a reference voltage and are initialized to 125kHz sampling frequency by dividing the 16Mhz clock with a pre-scalar of 128. These ADCs are decent enough to convert the information to digital data, but the relatively sporadic data output from the microphones creates a large hurdle when attempting to capture, convert, and compare that data in real time, with a small time window to make the information available to the user in a short amount of time.

IX. Setup

1. Connect the power adapter of the surveillance camera to the AC outlet.
2. Connect the breakout circuit to the correct pins on the micro controller found on Table 1.
3. Connect the micro controller to the correct pins on the amplifying circuit found on Table 2.
4. Connect the output from the microphones to each corresponding amplifier circuit.
5. Connect the video cable of the surveillance camera to the input of the breakout circuit.
6. Connect the video cable of the breakout circuit to the input of the Vuzix glasses.
7. Connect the USB power cord of the micro controller to USB powered port on the computer.
8. Turn on Glasses.
9. Go into AVR Studio 6.0 CMD Prompt.
10. Change Directory (cd) into AVR Studio → Overlay → Overlay → Debug.
11. Type “prog Overlay.hex” into the command line.
12. Hold down triggers for all active microphones.
13. Check for accuracy by making loud sounds in different directions.

X. Improvements

This section covers improvements that could be made to this project to make it more efficient, accurate, and more robust; future projects can expand upon the ideas that follow.

1. NTSC Surveillance Camera

The resolution of the camera in this project is 628*582. A camera with higher resolution gives the user an even greater realistic picture of their surroundings. Also, a wireless Bluetooth camera makes this project more mobile and user friendly, a consumer implementation of this project needs this requirement.

2. Microphones

The microphones are some of the most important aspects of the design to achieve correct functionality of this project. The microphones used in this project had a combined (three microphones) cost of sixty dollars, and were made of cheap plastic. The accuracy of these instruments must be greatly improved to have a truly precise product. The new microphones still need uni-directionality, and must be wireless as well; discrete placement of these must also be considered to prevent any hindrance to the user.

3. Amplifiers

A different amplifier should be used to produce more accurate results from the microphones. The LM741 is a cheap audio amplifying IC that is not very reliable. A consumer product of this design needs a better amplifier, and this circuit needs to be manufactured into a PCB for a more efficient design.

4. Virtual Reality Glasses

The glasses used in this project uses analog video signals to create an image, future reality glasses should use all digital information for the whole system to be faster and more efficient. The resolution and size of the displays need upgrading as well. A more user friendly design of glasses that are ergonomic and comfortable should also be considered in later designs.

5. Entire System Integration

A consumer product version of this project should contain all circuitry on one PCB. If possible, all external circuitry would be transferred to the microcontroller, eliminating a clutter of connections around the system integration. All pieces in the system should be wireless to eliminate connections running around the user that could cause irritation or hindrance to the consumer. Future designs should be as lightweight and out of the user's way as possible to make this system as mobile and effective as possible. A rechargeable battery that supplies voltage to the entire system would also be an improvement to this design.

XI. Appendices

A. Budget of Materials

•	1x	Vuzix Virtual Reality Glasses	$350.00
•	3x	Parabolic Microphone	$25.00
•	1x	9V DC Power Adapter	$22.00
•	1x	Arduino Uno (ATmega 328P u-Controller)	$37.00
•	3x	100 Ohm Resistor	$0.15
•	3x	100k Ohm Resistor	$.015
•	2x	Composite Video Cable	$3.50
•	1x	Surveillance YA-208 NTSC Camera	$11.00
•	1x	LM1881 Sync Separator IC	$3.50
•	1x	RCA Coupler	$2.00
•	2x	0.1uF Capacitor	$0.25
•	Xx	Wires	$X.XX
•	3x	Audio Cable (Spliced)	$5.00

B. Information on Data Transmission

1. Serial Peripheral Interface

SPI allows for fast communication between one or more slave devices and the ability to communicate with them with a single data bus. Figure 8 encapsulates the connections between master and slave devices.

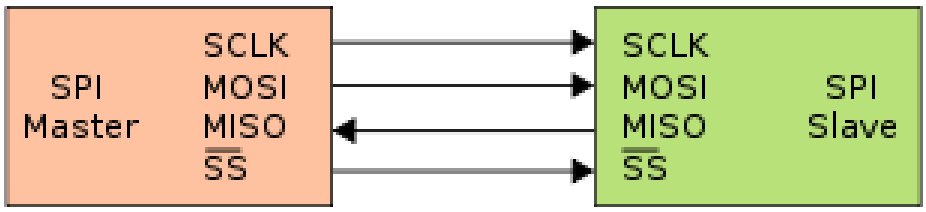

Figure 7 SPI Communications between Master and Slave Device

The whole process runs off a clock and is therefore synchronized with the slave device, and the speed of the clock cannot be faster than the clock of the slave device.

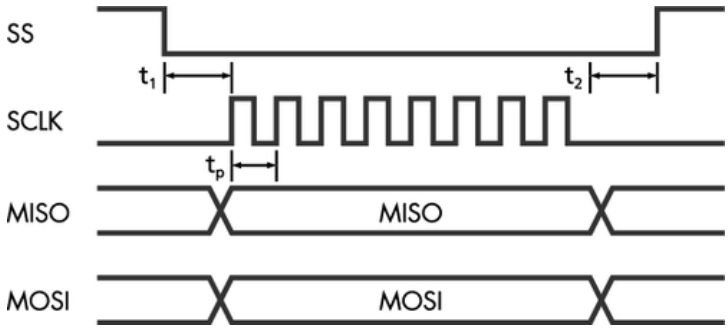

Figure 8 SPI Data Transfer Waveform

Figure 9 shows a typical waveform for a transmission of data between Master and Slave devices, and everything is transmitted while SS is de-asserted, and MISO MOSI start on a rising clock edge.

C. Information on Analog Signal Generation

On old black and white cathode ray tube televisions, there is a gun inside the television that shoots electrons onto the screen at different intensities; Figure 10 gives a visual representation of what is being described. The higher the intensity, the brighter and whiter the image was.

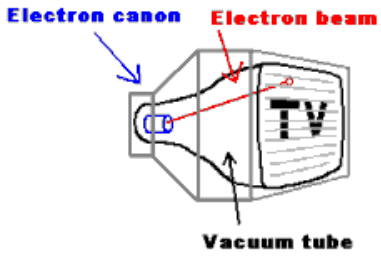

Figure 9 Signal Generation in a Cathode Ray Tube Television

The intensity scale ranges from 0.3 volts to 1 volt with 0.3 volts being black and 1 volt being white; everything in between is a shade of grey. When the gun reaches one end of the television, there is signal called the horizontal sync that tells the gun to move to the next line. When the gun reaches the bottom

of the screen, there is another signal called the vertical sync that tells the gun to move back to the top of the screen. Also, only every other horizontal line is filled in with voltages, so all of the odd lines are filled in first, and when the gun reaches the top, then the even lines, and back forth. The refresh rate of these televisions is around 50Hz, which is fast enough for your brain to put an image together. With this information, generating a black and white analog signal is easy and can be done using the Atmega 328P Micro controller and a screen that takes analog video input.

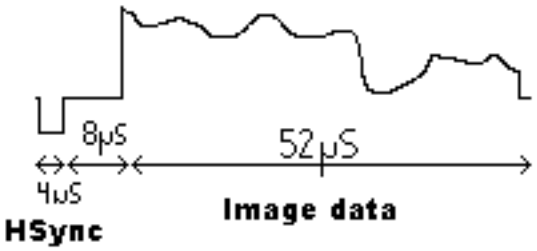

Figure 10 Typical Waveform for an Analog Video Signal

You can depict from Figure 11 that the actual data for the image is 52uS long; 4uS is used to tell the camera to go to the next line, and another 8uS is for the camera to move into position. In order to generate an analog signal using the micro controller, an analog to digital circuit is needed and the output of that circuit will connect to the input of the television or screen. Figure 12 describes the connections.

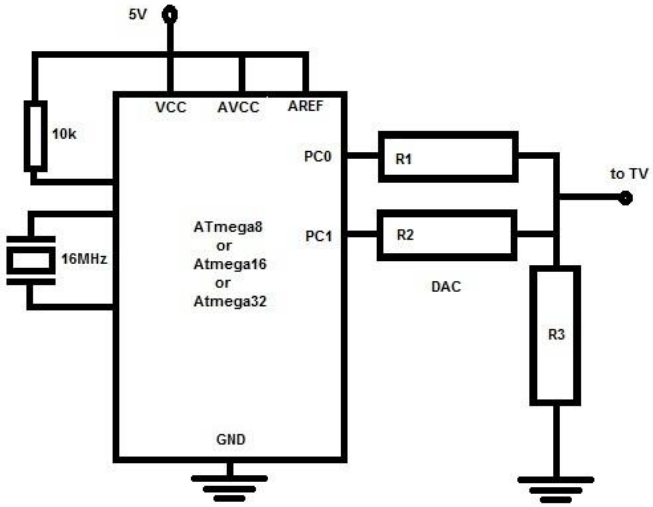

Figure 11 Connections to Generate an Analog Signal using Atmega 328P Micro Controller

For this test, three voltage levels are essential, 0 volts, ~0.3 volts, and 1 volts. In order to trigger a horizontal sync, 0 volts is used; 0.3 volts will produce a black pixel, and 1 volt will produce a white pixel.

First R1 has been chosen to be 450Ω, R2 to be 900Ω, and R3 is the internal resistance of the television which is around 75Ω. Also, Vin is the micro controller power of 5V. To achieve a black voltage level (0.3V), PORTC = 01, this will ground R1 and connect R2 in series with R3. Using simple voltage division:

Vin = 5V * (75Ω//450Ω) / (900Ω + (75Ω//450Ω)) = 0.33V (Black)

A more visual representation can be found in Figure 13, this time for a gray intensity value:

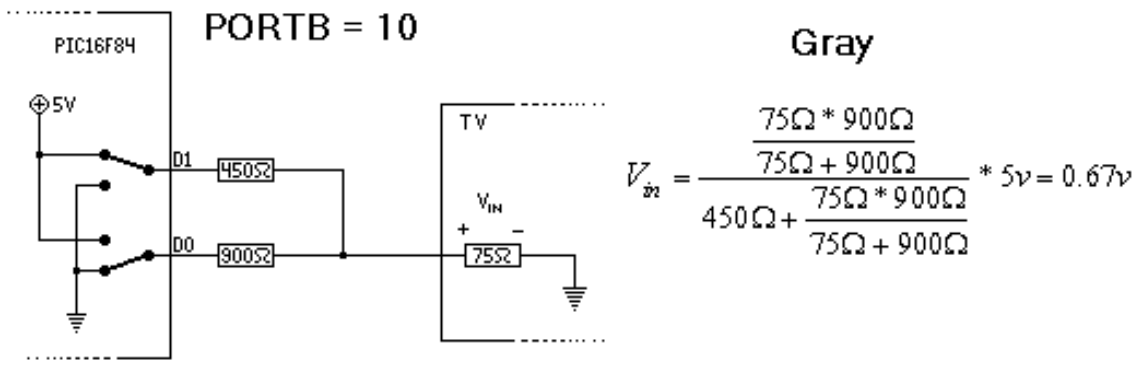

Figure 12 Voltage Division to achieve Gray Intensity Voltage Level

The C code in Figure 14 generates a white line 1uS thick in the middle of the screen. The code will generate a 0 volt value to start horizontal sync, and then black to delay for 8uS to put the gun into position; after that the image is printed onto the screen, black is printed for 25uS, then white for 1uS, and then 26uS for a total of 52uS of image data; this is repeated on every line creating a single white line on the television or screen.

```
1   #include<avr/io.h>
2   #define F_CPU 16450000
3   #include <util/delay.h>
4   #define ZERO PORTC=0
5   #define BLACK PORTC=1 //v out = .3v (black color)
6   #define WHITE PORTC=2 //v out = .8v (just white color)
7
8   void main()
9   {
10      DDRC = 1<<PC0|1<<PC1;
11      while(1) {
12          ZERO;_delay_us(4);BLACK;_delay_us(8);//horizontal synchronization
13          _delay_us(25);
14          WHITE;
15          _delay_us(1);
16          BLACK;
17          _delay_us(26);
18      }
19  }
```

D. Information on Micro Controllers

The Arduino Uno is a micro controller that is based on the Atmega 328P microprocessor. This board is extremely versatile and contains built-in GPIO's (General Purpose Input Output), ADC's (Analog to Digital Converters), and requires only a small voltage to become operable.

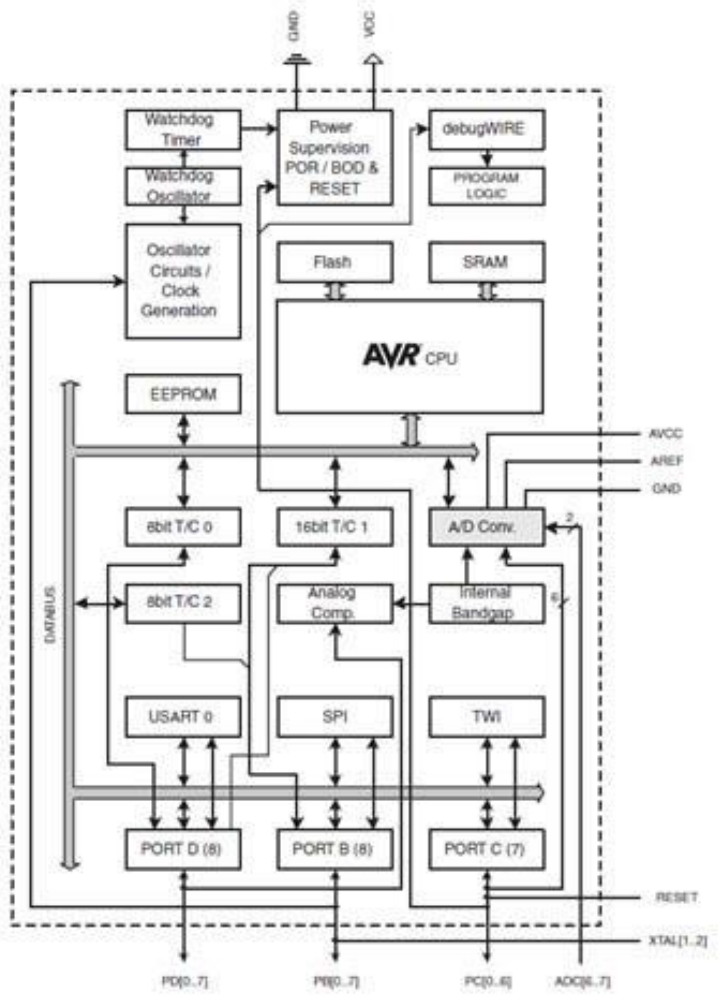

Figure 13 Atmega 328 Block Diagram

The block diagram in Figure 13 portrays an in-depth illustration of how data is transferred and processed in this microprocessor. The Arduino Uno can be programmed using AVR Studio, and all code is written in C, making the entire project easier to control and debug.

E. Scheduling and Testing

All of the essential parts needed to build this device have been ordered and almost all have arrived, save for the LM1881 chip, this should arrive by the end of November. The tentative scheduling of tests and reports are found in Table 3. Once all tests have been executed, then system integration will begin and the results can be tabulated. The schedule allocates time for design iteration to correct deficiencies identified in the initial testing.

Table 3 Description of milestones and their Completion Dates

Milestones	Tentative Completion Date	Completed	Remaining
Gather Components	12/14/2012	10	200
Begin Prelim Testing	1/13/2012	30	180
Design System	1/30/2012	35	175
Full System Integration	2/20/2013	70	140
Test Entire System	2/25/2012	85	125
Document Results	2/30/2012	100	110
Re-Design System	3/10/2012	110	100
Implement New Design	3/30/2102	130	80
Test Full System	4/15/2012	170	40
Document Final Results	4/27/2012	200	10
Senior Project Expo	5/30/2012	210	0

The goal of this project is to create a visual representation of the approximate location of loud sounds and present this in real-time to a user wearing virtual reality glasses. If time permits before the senior project expo, an enhancement of this project consists of improving the accuracy by building a more comprehensive sound location algorithm. In addition, more creative graphics such as different shaped objects or different colors corresponding to the amplitude of the sound could be displayed instead of just a vertical bar. Different shapes require custom characters to written to the Max7456 IC.

Significant preparation has already been done, but it is impossible to anticipate every potential problem that could be encountered. For this reason, a scrum management style will be in place. The design is broken up into pieces, which correspond to ‘sprints’ using scrum language. These ‘sprints’ focus on completing the test plans that are detailed in section VII of this report. Not only will these sprints serve as confidence boosters, but will also contribute to the system integration of the overall design and assist in debugging the circuit since every component is accounted for in the test procedures.

F. Code for Atmel 328P Microcontroller

```
/*
    MAX7456 overlay code combined with the single mic code
       Jonathan Brophy, Senior Project 2012-2013

       This test code creates a vertical white bar on the screen
       in the direction of where the user hears a sound

*/
#define F_CPU 16000000
#define SAMPLES 1000

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define RESET 0      //PB0
#define CS 2         //PB2
#define MOSI 3       //PB3
#define MISO 4       //PB4
#define SCK 5        //PB5

//Define functions
//======================
void ioinit(void);      //Initializes IO
void welcome();
void complete();
void tinaSmilkstein();
void tinaSmillksteinCascade();
void tinaShow();
void verticalBarShow();
void enableAndClear();
void vertical_bar(int upper, int lower);
void SPI_write(char address, char byte);
char SPI_read(char address);
void Initialize_ADC0(void);
void Initialize_ADC1(void);
int GetSound(int mic, int samples);
void reset();
//======================

int main (void)
{
       char x;

    ioinit(); //Setup IO pins and defaults

       _delay_ms(1000);

       //Reset
       //=========================================================================
       PORTB &= ~(1<<RESET);
       _delay_ms(1000);
       PORTB |= (1<<RESET);
       _delay_ms(1000);
       //=========================================================================
```

```
SPI_write(0,0x08);   //Bit 6:PAL Signal, Bit 3: Enable Display of OSD image
_delay_ms(1);

//automatic black level control, have to read, augment and rewrite
//The data sheet is rather specific about this
//============================================================================
x = SPI_read(0xEC);
_delay_ms(1);
x &= 0xEF;
SPI_write(0x6C,x);
_delay_ms(1);
//============================================================================

//Adjust The Screen to fit nicely into the Virtual Reality Glasses Display
//============================================================================
SPI_write(0x04,0);  //DMM set to 0
SPI_write(0x02, 0x1E);  //Horizontal Offset Left by -30 pixels
SPI_write(0x03, 0x14);  //Vertical Offset Down by -4 pixels
//============================================================================

/*Initialize_ADC0();
int base_noise = GetSound(0, 1000);
int threshold = base_noise + 30;*/
Initialize_ADC0();
Initialize_ADC1();
int base_noise = ((GetSound(0, 1000) + GetSound(1, 1000))/2); //Average
int threshold = base_noise + 15, mic0 = 0, mic1 = 0;

welcome();

while(1)
{
	//Tina Smilkstein
	//tinaShow();

	//Single Vertical Bar Testing
	/*vertical_bar(12, 26);
	SPI_write(0, 0x08); //Enable Display
	_delay_ms(200);
	SPI_write(0x04,0x06); //Clear Display on /VSync
	_delay_ms(50);

	vertical_bar(25, 9);
	SPI_write(0, 0x08); //Enable Display
	_delay_ms(200);
	SPI_write(0x04, 0x06); //Clear Display on /VSync
	_delay_ms(50); */

	//Single Mic Testing
	/*int mic0 = GetSound(0, 1000);

	if (mic0 > threshold)
	{
		vertical_bar(14, 28); //Middle
		SPI_write(0,0x08);  //Enable Display of OSD image
		_delay_ms(1000);
		SPI_write(0x04, 0x06); //Clear Display
		_delay_ms(500);
```

```
}*/

//Testing using two mics concurrently
mic0 = GetSound(0, 1000);
mic1 = GetSound(1, 1000);

if ((mic0 > threshold) && (mic0 > (mic1 + 30)))
{
	vertical_bar(0, 14); //Far Left
	SPI_write(0,0x08);  //Enable Display of OSD image
	_delay_ms(1000);
	SPI_write(0x04, 0x06); //Clear Display
	_delay_ms(500);
} else if ((mic1 > threshold) && (mic1 > (mic0 + 30))) {
	vertical_bar(29, 13); //Far Right
	SPI_write(0,0x08);  //Enable Display of OSD image
	_delay_ms(1000);
	SPI_write(0x04, 0x06); //Clear Display
	_delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 28))) {
	vertical_bar(1, 15); //Left Side
	SPI_write(0,0x08);  //Enable Display of OSD image
	_delay_ms(1000);
	SPI_write(0x04, 0x06); //Clear Display
	_delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 28))) {
	vertical_bar(28, 12); //Right Side
	SPI_write(0,0x08);  //Enable Display of OSD image
	_delay_ms(1000);
	SPI_write(0x04, 0x06); //Clear Display
	_delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 26))) {
	vertical_bar(2, 16); //Left Side
	SPI_write(0,0x08);  //Enable Display of OSD image
	_delay_ms(1000);
	SPI_write(0x04, 0x06); //Clear Display
	_delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 26))) {
	vertical_bar(27, 11); //Right Side
	SPI_write(0,0x08);  //Enable Display of OSD image
	_delay_ms(1000);
	SPI_write(0x04, 0x06); //Clear Display
	_delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 24))) {
	vertical_bar(3, 17); //Left Side
	SPI_write(0,0x08);  //Enable Display of OSD image
	_delay_ms(1000);
	SPI_write(0x04, 0x06); //Clear Display
	_delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 24))) {
	vertical_bar(26, 10); //Right Side
	SPI_write(0,0x08);  //Enable Display of OSD image
```

```
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 22))) {
        vertical_bar(4, 18); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 22))) {
        vertical_bar(25, 9); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 20))) {
        vertical_bar(5, 19); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 20))) {
        vertical_bar(24, 8); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 18))) {
        vertical_bar(6, 20); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 18))) {
        vertical_bar(23, 7); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 16))) {
        vertical_bar(7, 21); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 16))) {
        vertical_bar(22, 6); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
```

```
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 14))) {
        vertical_bar(8, 22); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 14))) {
        vertical_bar(21, 5); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 12))) {
        vertical_bar(9, 23); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 12))) {
        vertical_bar(20, 4); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 10))) {
        vertical_bar(10, 24); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 10))) {
        vertical_bar(19, 3); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic0 > threshold) && (mic0 > (mic1 + 8))) {
        vertical_bar(11, 25); //Left Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
else if ((mic1 > threshold) && (mic1 > (mic0 + 8))) {
        vertical_bar(18, 2); //Right Side
        SPI_write(0,0x08);  //Enable Display of OSD image
        _delay_ms(1000);
        SPI_write(0x04, 0x06); //Clear Display
        _delay_ms(500);
}
```

```
                else if ((mic0 > threshold) && (mic0 > (mic1 + 6))) {
                        vertical_bar(12, 26); //Left Side
                        SPI_write(0,0x08);  //Enable Display of OSD image
                        _delay_ms(1000);
                        SPI_write(0x04, 0x06); //Clear Display
                        _delay_ms(500);
                }
                else if ((mic1 > threshold) && (mic1 > (mic0 + 6))) {
                        vertical_bar(17, 1); //Right Side
                        SPI_write(0,0x08);  //Enable Display of OSD image
                        _delay_ms(1000);
                        SPI_write(0x04, 0x06); //Clear Display
                        _delay_ms(500);
                }
                else if ((mic0 > threshold) && (mic0 > (mic1 + 4))) {
                        vertical_bar(13, 27); //Left Side
                        SPI_write(0,0x08);  //Enable Display of OSD image
                        _delay_ms(1000);
                        SPI_write(0x04, 0x06); //Clear Display
                        _delay_ms(500);
                }
                else if ((mic1 > threshold) && (mic1 > (mic0 + 4))) {
                        vertical_bar(16, 0); //Right Side
                        SPI_write(0,0x08);  //Enable Display of OSD image
                        _delay_ms(1000);
                        SPI_write(0x04, 0x06); //Clear Display
                        _delay_ms(500);
                }
                else if ((mic0 > threshold) && (mic0 > (mic1 + 2))) {
                        vertical_bar(14, 28); //Left Side
                        SPI_write(0,0x08);  //Enable Display of OSD image
                        _delay_ms(1000);
                        SPI_write(0x04, 0x06); //Clear Display
                        _delay_ms(500);
                }
                else if ((mic1 > threshold) && (mic1 > (mic0 + 2))) {
                        vertical_bar(15, 29); //Right Side
                        SPI_write(0,0x08);  //Enable Display of OSD image
                        _delay_ms(1000);
                        SPI_write(0x04, 0x06); //Clear Display
                        _delay_ms(500);
                }
        }
}

void vertical_bar(int upper, int lower)
{
        SPI_write(0x05,0x00);//DMAH Top Half of Screen

        SPI_write(0x06,upper);//DMAL 1
        SPI_write(0x07,0xFF);

        SPI_write(0x06,upper+30);//DMAL 2
        SPI_write(0x07,0xFF);

        SPI_write(0x06,upper+60);//DMAL 3
        SPI_write(0x07,0xFF);
```

```
        SPI_write(0x06,upper+90);//DMAL 4
        SPI_write(0x07,0xFF);

        SPI_write(0x06,upper+120);//DMAL 5
        SPI_write(0x07,0xFF);

        SPI_write(0x06,upper+150);//DMAL 6
        SPI_write(0x07,0xFF);

        SPI_write(0x06,upper+180);//DMAL 7
        SPI_write(0x07,0xFF);

        SPI_write(0x06,upper+210);//DMAL 8
        SPI_write(0x07,0xFF);

        if (upper <= 15) {
           SPI_write(0x06,upper+240);//DMAL 9
           SPI_write(0x07,0xFF);
        }

        SPI_write(0x05,0x01);//DMAH Bottom Half of Screen

        SPI_write(0x06,lower); //DMAL 9
        SPI_write(0x07,0xFF);

        SPI_write(0x06,lower+30);  //DMAL 10
        SPI_write(0x07,0xFF);

        SPI_write(0x06,lower+60);  //DMAL 11
        SPI_write(0x07,0xFF);

        SPI_write(0x06,lower+90);  //DMAL 12
        SPI_write(0x07,0xFF);

        if (lower <= 13 || lower == 255) {
           SPI_write(0x06,lower+120);  //DMAL 13
           SPI_write(0x07,0xFF);
        }
}

void ioinit (void)
{
        UCSR0B = 0x00; //Disable Tx and Rx
        PORTB = 0xFF;
        DDRB = ((1<<CS) | (1<<MOSI) | (1<<SCK) | (1<<RESET));

    TCCR2B = (1<<CS21); //Set Prescaler to 8. CS21=1
}

void welcome()
{
        SPI_write(0x05,0x00);//DMAH Top Half of Screen

        //WELCOME
        SPI_write(0x06,161);//DMAL
        SPI_write(0x07,0x21); //W

        SPI_write(0x06,162);//DMAL
```

```
SPI_write(0x07,0x0F); //E

SPI_write(0x06,163);//DMAL
SPI_write(0x07,0x16); //L

SPI_write(0x06,164);//DMAL
SPI_write(0x07,0x0D); //C

SPI_write(0x06,165);//DMAL
SPI_write(0x07,0x19); //O

SPI_write(0x06,166);//DMAL
SPI_write(0x07,0x17); //M

SPI_write(0x06,167);//DMAL
SPI_write(0x07,0x0F); //E

SPI_write(0, 0x08); //Enable Display
_delay_ms(2000);
SPI_write(0x04,0x06); //Clear Display on /VSync
_delay_ms(50);

//INITIALIZING...
SPI_write(0x06,156);//DMAL
SPI_write(0x07,0x13); //I

SPI_write(0x06,157);//DMAL
SPI_write(0x07,0x18); //N

SPI_write(0x06,158);//DMAL
SPI_write(0x07,0x13); //I

SPI_write(0x06,159);//DMAL
SPI_write(0x07,0x1E); //T

SPI_write(0x06,160);//DMAL
SPI_write(0x07,0x13); //I

SPI_write(0x06,161);//DMAL
SPI_write(0x07,0x0B); //A

SPI_write(0x06,162);//DMAL
SPI_write(0x07,0x16); //L

SPI_write(0x06,163);//DMAL
SPI_write(0x07,0x13); //I

SPI_write(0x06,164);//DMAL
SPI_write(0x07,0x24); //Z

SPI_write(0x06,165);//DMAL
SPI_write(0x07,0x13); //I

SPI_write(0x06,166);//DMAL
SPI_write(0x07,0x18); //N

SPI_write(0x06,167);//DMAL
SPI_write(0x07,0x11); //G
```

```
        SPI_write(0x06,168);//DMAL
        SPI_write(0x07,0x41); //.

        SPI_write(0x06,169);//DMAL
        SPI_write(0x07,0x41); //.

        SPI_write(0x06,170); //DMAL
        SPI_write(0x07,0x41); //.

        SPI_write(0, 0x08); //Enable Display
        _delay_ms(1000);
        SPI_write(0x04,0x06); //Clear Display on /VSync
        _delay_ms(500);

        verticalBarShow();
        _delay_ms(500);
        complete();
}

void complete()
{
        SPI_write(0x05,0x00);//DMAH Top Half of Screen

        //COMPLETE
        SPI_write(0x06,160);//DMAL
        SPI_write(0x07,0x0D); //C

        SPI_write(0x06,161);//DMAL
        SPI_write(0x07,0x19); //O

        SPI_write(0x06,162);//DMAL
        SPI_write(0x07,0x17); //M

        SPI_write(0x06,163);//DMAL
        SPI_write(0x07,0x1A); //P

        SPI_write(0x06,164);//DMAL
        SPI_write(0x07,0x16); //L

        SPI_write(0x06,165);//DMAL
        SPI_write(0x07,0x0F); //E

        SPI_write(0x06,166);//DMAL
        SPI_write(0x07,0x1E); //T

        SPI_write(0x06,167); //DMAL
        SPI_write(0x07,0x0F); //E

        SPI_write(0, 0x08); //Enable Display
        _delay_ms(1500);
        SPI_write(0x04,0x06); //Clear Display on /VSync
        _delay_ms(500);

        //READY FOR USE
        SPI_write(0x06,158);//DMAL
        SPI_write(0x07,0x1C); //R
```

```
        SPI_write(0x06,159);//DMAL
        SPI_write(0x07,0x0F); //E

        SPI_write(0x06,160);//DMAL
        SPI_write(0x07,0x0B); //A

        SPI_write(0x06,161);//DMAL
        SPI_write(0x07,0x0E); //D

        SPI_write(0x06,162);//DMAL
        SPI_write(0x07,0x23); //Y

        SPI_write(0x06,164);//DMAL
        SPI_write(0x07,0x10); //F

        SPI_write(0x06,165);//DMAL
        SPI_write(0x07,0x19); //O

        SPI_write(0x06,166); //DMAL
        SPI_write(0x07,0x1C); //R

        SPI_write(0x06,168);//DMAL
        SPI_write(0x07,0x1F); //U

        SPI_write(0x06,169);//DMAL
        SPI_write(0x07,0x1D); //S

        SPI_write(0x06,170);//DMAL
        SPI_write(0x07,0x0F); //E

        SPI_write(0, 0x08); //Enable Display
        _delay_ms(1500);
        SPI_write(0x04,0x06); //Clear Display on /VSync
        _delay_ms(500);
}

void tinaSmilkstein()
{
        SPI_write(0x06,37);//DMAL 1
        SPI_write(0x07,0x1E); //T

        SPI_write(0x06,38);//DMAL 2
        SPI_write(0x07,0x2D); //i

        SPI_write(0x06,39);//DMAL 3
        SPI_write(0x07,0x32); //n

        SPI_write(0x06,40);//DMAL 4
        SPI_write(0x07,0x25); //a

        SPI_write(0x06,42);//DMAL 5
        SPI_write(0x07,0x1D); //S

        SPI_write(0x06,43);//DMAL 6
        SPI_write(0x07,0x31); //m

        SPI_write(0x06,44);//DMAL 7
        SPI_write(0x07,0x2D); //i
```

```
        SPI_write(0x06,45);//DMAL 7
        SPI_write(0x07,0x30); //l

        SPI_write(0x06,46);//DMAL 7
        SPI_write(0x07,0x2F); //k

        SPI_write(0x06,47);//DMAL 8
        SPI_write(0x07,0x37); //s

        SPI_write(0x06,48);//DMAL 9
        SPI_write(0x07,0x38); //t

        SPI_write(0x06,49);//DMAL 10
        SPI_write(0x07,0x29); //e

        SPI_write(0x06,50);//DMAL 11
        SPI_write(0x07,0x2D); //i

        SPI_write(0x06,51);//DMAL 12
        SPI_write(0x07,0x32); //n
}

void tinaSmillksteinCascade()
{
        SPI_write(0x06,37);//DMAL 1
        SPI_write(0x07,0x1E); //T
        enableAndClear();

        SPI_write(0x06,38);//DMAL 2
        SPI_write(0x07,0x2D); //i
        enableAndClear();

        SPI_write(0x06,39);//DMAL 3
        SPI_write(0x07,0x32); //n
        enableAndClear();

        SPI_write(0x06,40);//DMAL 4
        SPI_write(0x07,0x25); //a
        enableAndClear();

        SPI_write(0x06,42);//DMAL 5
        SPI_write(0x07,0x1D); //S
        enableAndClear();

        SPI_write(0x06,43);//DMAL 6
        SPI_write(0x07,0x31); //m
        enableAndClear();

        SPI_write(0x06,44);//DMAL 7
        SPI_write(0x07,0x2D); //i
        enableAndClear();

        SPI_write(0x06,45);//DMAL 7
        SPI_write(0x07,0x30); //l
        enableAndClear();

        SPI_write(0x06,46);//DMAL 7
```

```
        SPI_write(0x07,0x2F); //k
        enableAndClear();

        SPI_write(0x06,47);//DMAL 8
        SPI_write(0x07,0x37); //s
        enableAndClear();

        SPI_write(0x06,48);//DMAL 9
        SPI_write(0x07,0x38); //t
        enableAndClear();

        SPI_write(0x06,49);//DMAL 10
        SPI_write(0x07,0x29); //e
        enableAndClear();

        SPI_write(0x06,50);//DMAL 11
        SPI_write(0x07,0x2D); //i
        enableAndClear();

        SPI_write(0x06,51);//DMAL 12
        SPI_write(0x07,0x32); //n
        enableAndClear();
}

void tinaShow()
{
        tinaSmillksteinCascade(); //Flash letters
        tinaSmilkstein(); //Load entire name into max7456 memory

        SPI_write(0, 0x08); //Enable Display
        _delay_ms(3000);
        SPI_write(0x04,0x06); //Clear Display on /VSync
        _delay_ms(500);

        tinaSmilkstein();
        SPI_write(0, 0x08); //Enable Display
        _delay_ms(500);
        SPI_write(0x04,0x06); //Clear Display on /VSync
        _delay_ms(500);

        tinaSmilkstein();
        SPI_write(0, 0x08); //Enable Display
        _delay_ms(500);
        SPI_write(0x04,0x06); //Clear Display on /VSync
        _delay_ms(500);
}

void verticalBarShow()
{
    //Range of Vertical Bars from Left to Right:
    vertical_bar(0, 14); //Far Left
       enableAndClear();
    vertical_bar(1, 15);
       enableAndClear();
    vertical_bar(2, 16);
       enableAndClear();
    vertical_bar(3, 17);
       enableAndClear();
```

```
vertical_bar(4, 18);
   enableAndClear();
vertical_bar(5, 19);
   enableAndClear();
vertical_bar(6, 20);
enableAndClear();
   vertical_bar(7, 21);
   enableAndClear();
vertical_bar(8, 22);
enableAndClear();
   vertical_bar(9, 23);
   enableAndClear();
vertical_bar(10, 24);
   enableAndClear();
vertical_bar(11, 25);
   enableAndClear();
vertical_bar(12, 26);
   enableAndClear();
vertical_bar(13, 27); //Left
   enableAndClear();
vertical_bar(14, 28); //Middle
   enableAndClear();
vertical_bar(15, 29); //Right
   enableAndClear();
vertical_bar(16, 0);
   enableAndClear();
vertical_bar(17, 1);
   enableAndClear();
vertical_bar(18, 2);
   enableAndClear();
vertical_bar(19, 3);
   enableAndClear();
vertical_bar(20, 4);
   enableAndClear();
vertical_bar(21, 5);
   enableAndClear();
vertical_bar(22, 6);
   enableAndClear();
vertical_bar(23, 7);
   enableAndClear();
vertical_bar(24, 8);
   enableAndClear();
vertical_bar(25, 9);
   enableAndClear();
vertical_bar(26, 10);
   enableAndClear();
vertical_bar(27, 11);
   enableAndClear();
vertical_bar(28, 12);
   enableAndClear();
vertical_bar(29, 13); //Far Right */
enableAndClear();

   _delay_ms(250);

//Range of vertical bars from right to left:
vertical_bar(29, 13); //Far Right
enableAndClear();
```

```
vertical_bar(28, 12);
enableAndClear();
vertical_bar(27, 11);
enableAndClear();
vertical_bar(26, 10);
enableAndClear();
vertical_bar(25, 9);
enableAndClear();
vertical_bar(24, 8);
enableAndClear();
vertical_bar(23, 7);
enableAndClear();
vertical_bar(22, 6);
enableAndClear();
vertical_bar(21, 5);
enableAndClear();
vertical_bar(20, 4);
enableAndClear();
vertical_bar(19, 3);
enableAndClear();
vertical_bar(18, 2);
enableAndClear();
vertical_bar(17, 1);
enableAndClear();
vertical_bar(16, 0);
enableAndClear();
vertical_bar(15, 29); //Right
enableAndClear();
vertical_bar(14, 28); //Middle
enableAndClear();
vertical_bar(13, 27); //Left
enableAndClear();
vertical_bar(12, 26);
enableAndClear();
vertical_bar(11, 25);
enableAndClear();
vertical_bar(10, 24);
enableAndClear();
vertical_bar(9, 23);
enableAndClear();
vertical_bar(8, 22);
enableAndClear();
vertical_bar(7, 21);
enableAndClear();
vertical_bar(6, 20);
enableAndClear();
vertical_bar(5, 19);
enableAndClear();
vertical_bar(4, 18);
enableAndClear();
vertical_bar(3, 17);
enableAndClear();
vertical_bar(2, 16);
enableAndClear();
vertical_bar(1, 15);
enableAndClear();
vertical_bar(0, 14); //Far Left */
enableAndClear();
```

```
}
void enableAndClear()
{
	SPI_write(0, 0x08); //Enable Display
	_delay_ms(100);
	SPI_write(0x04,0x06); //Clear Display on /VSync
	_delay_ms(30);
}

void SPI_write(char address, char byte)
{
	unsigned char SPICount;			// Counter used to clock out the data

	unsigned char SPIData;		// Define a data structure for the SPI data

	PORTB |= (1<<CS);		// Make sure we start with active-low CS high
	PORTB &= ~(1<<SCK);				// and CK low

	SPIData = address;		// Preload the data to be sent with Address
	PORTB &= ~(1<<CS);		// Set active-low CS low to start the SPI cycle


	for (SPICount = 0; SPICount < 8; SPICount++) //Prepare to clock out address
	{
		if (SPIData & 0x80) PORTB |= (1<<MOSI); // Check 1 and set MOSI line

		else PORTB &= ~(1<<MOSI);

		PORTB |= (1<<SCK);		// Toggle the clock line
		PORTB &= ~(1<<SCK);

		SPIData <<= 1;			// Rotate to get the next bit
	}					// and loop back to send the next bit


	 // Repeat for the Data byte
	SPIData = byte;			// Preload the data to be sent with Data

	for (SPICount = 0; SPICount < 8; SPICount++)
	{
		if (SPIData & 0x80) PORTB |= (1<<MOSI);

		else PORTB &= ~(1<<MOSI);

		PORTB |= (1<<SCK);
		PORTB &= ~(1<<SCK);

		SPIData <<= 1;
	}

	PORTB |= (1<<CS);
	PORTB &= ~(1<<MOSI);

}

char SPI_read(char address)
{
```

```
	unsigned char SPICount;		// Counter used to clock out the data

	char SPIData;
	char temp;

	PORTB |= (1<<CS);		// Make sure we start with active-low CS high
	PORTB &= ~(1<<SCK);		   // and CK low

	SPIData = address;		 // Preload the data to be sent with Address
	PORTB &= ~(1<<CS);		 //Set active-low CS low to start the SPI cycle


	for (SPICount = 0; SPICount < 8; SPICount++) //Prepare to clock out address
	{
		if (SPIData & 0x80) PORTB |= (1<<MOSI); //Check 1 and set MOSI line

		else PORTB &= ~(1<<MOSI);

		PORTB |= (1<<SCK);			// Toggle the clock line
		PORTB &= ~(1<<SCK);

		SPIData <<= 1;			// Rotate to get the next bit
	}					// and loop back to send the next bit


	PORTB &= ~(1<<MOSI);			// Reset the MOSI data line

	SPIData = 0;

	for (SPICount = 0; SPICount < 8; SPICount++) // Prepare to clock in data
	{
		SPIData <<=1;				// Rotate the data
		PORTB |= (1<<SCK);    //Raise clock to clock data out of the MAX7456

		temp = PINB;
		if (temp & (1<<MISO)) SPIData |= 1;		// Read the data bit

		PORTB &= ~(1<<SCK);		// Drop the clock ready for the next bit
	}					// and loop back
	PORTB |= (1<<CS);			// Raise CS

	return SPIData;			// Finally return the read data
}

void Initialize_ADC0(void)
{
	ADCSRA = 0x87; //10000111: pre-scaler set to CLK/128 = 125kHz
	ADCSRB = 0x00; //Turn gain and auto-trigger off
	ADMUX = 0x00; //Set ADC channel ADC1 with 1x gain
}


void Initialize_ADC1(void)
{
	ADCSRA = 0x87; //10000111: pre-scaler set to CLK/128 = 125kHz
	ADCSRB = 0x00; //Turn gain and auto-trigger off
	ADMUX = 0x01; //Set ADC channel ADC1 with 1x gain
}
```

```
int GetSound(int mic, int samples)
{
       int i = 0;
       long sum = 0;

       for (i = 0; i < samples; i++) { //take 1000 samples of the sound
          if (mic == 1) //select analog input channel
             ADMUX = 0x01;
          else
             ADMUX = 0x00;
          ADCSRA = 0xC7; //11000111: pre-scalar set to CLK/128 = 125kHZ
         _delay_us(260);
          sum += ADC & 0x3FF; //Sum of the noise is turned into a long value
       }
    return (int)(sum/i); //return the average of 100 samples
}
```

G. References

[1] ANI, *Now, glasses that allow deaf people 'see' sounds*, 06 September 2012

[2] *Sync Separator IC LM1881* datasheet
URL: <http://nootropicdesign.com/ve/downloads/LM1881.pdf>

[3] Low Power Audio Amplifier IC LM741 datasheet
URL: <http://www.ti.com/lit/ds/symlink/lm741.pdf>

[4] Low Power Audio Amplifier IC LM386N datasheet
URL: <http://www.nari.ee.ethz.ch/wireless/education/PPS/PPS02/doc/LM386.pdf>

[5] ATmega 328P Micro Controller datasheet
URL: <http://www.atmel.com/Images/8271s.pdf>

[6] Video Signal Generation
URL: <http://blog.vinu.co.in/2012/04/avr-based-monochrome-signal-generation.html>

[7] Video Signal Overlay using an ATmega8
URL: <http://garydion.com/projects/videoverlay/>

[8] Eldon, J.A.; , "Genlocked digital overlay on a video signal," *Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on* , vol., no., pp.17-19 vol.1, 3-6 May 1993
doi: 10.1109/ISCAS.1993.393644
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=393644&isnumber=8956>

[9] Ferguson, S.; Moere, A.V.; Cabrera, D.; , "Seeing sound: real-time sound visualisation in visual feedback loops used for training musicians," *Information Visualisation, 2005. Proceedings. Ninth International Conference on* , vol., no., pp. 97- 102, 6-8 July 2005
doi: 10.1109/IV.2005.114
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1509065&isnumber=32319>

[10] Maxim 7546 IC datasheet
URL: <https://www.sparkfun.com/datasheets/BreakoutBoards/MAX7456.pdf>

[11] How to Generate Video Signals using PIC
URL: < http://www.rickard.gunee.com/projects/video/pic/howto.php>

[12] AVR Based Monochrome Signal Generation for a PAL TV
URL: < http://blog.vinu.co.in/2012/04/avr-based-monochrome-signal-generation.html>