Microprocessor and Microcontrollers (EE306)

MTE Project Report

Implementing a Sound Detection System using Arduino Uno and a KY-037 Sound Sensor

Submitted By:

Tushar Choudhary
2K18/EE/225

INDEX

1. Introduction

Sounds are vibrations that flow in the form of a sound wave through a medium (air, water, steel, etc.). The sound wave is made up of air pressure changes caused by the movement of air molecules back and forth. These variations in air pressure are referred to as vibrations, and they can be detected using a number of sensors. These variations in air pressure are referred to as vibrations, and they can be detected using a number of sensors.

The sensor era has arrived. Dr. Janusz Bryzek, former VP Development of MEMS and Sensing Solutions at Fairchild Semiconductor, has stated that by 2022, there would be one trillion sensors in use.[1] According to Gartner, there will be 20 billion connected things in operation worldwide by 2020, with the majority of them containing several sensors.[2]

Sensors are significant because they collect a large amount of data that can be analyzed and interpreted to improve productivity, security, predict risks, and improve people's lives. In simple words, sensors can be used to impart intelligence into machines and systems and give birth to countless applications in various fields.

When this ability to analyze and interpret data in real-time is amalgamated with microcontrollers and microprocessors, we get intelligent systems with wide-ranging applications in various parts of life. In this project, I have implemented two rudimentary sound detection systems which use the Arduino microcontroller board and the KY-037 sound sensing module.

2. Theory

The model prepared in the project for creating a sound detection model uses the Arduino Uno. The Arduino UNO microcontroller board is based on the ATmega328 microcontroller. Before implementing the sound detection system through the Arduino, I have studied both Arduino Uno and ATmega328 from a theoretical and practical standpoint. We will also take a look at the details of the sound sensor that we'll be using.

a) Arduino UNO

The Arduino Uno is a microcontroller board based on the ATmega328 microcontroller. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHzcrystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.[3] It is perfectly designed and contains everything needed to support the microcontroller.

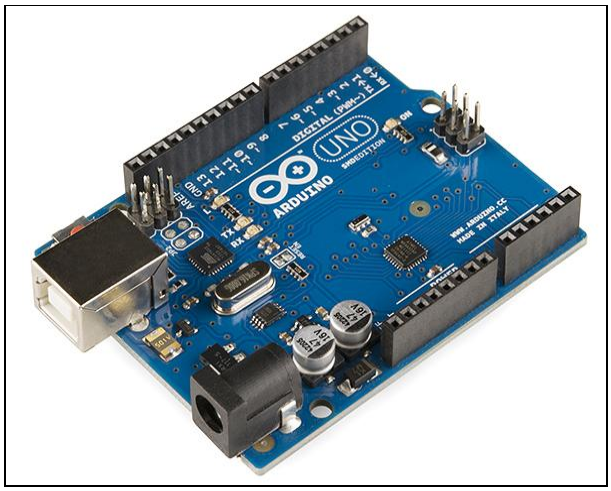

Fig. Arduino Uno

Users simply need to connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards as it uses Atmega8U2 programmed as a USB-to-serial converter rather than FTDI USB-to-serial driver chip. [4]

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, as they move forward. The Uno is the latest in a series of USB Arduino boards. It is also the reference model for the Arduino platform which is used for a comparison with previous versions.

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-9V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) (0.5 KB used by bootloader)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Fig. Arduino Uno Overview

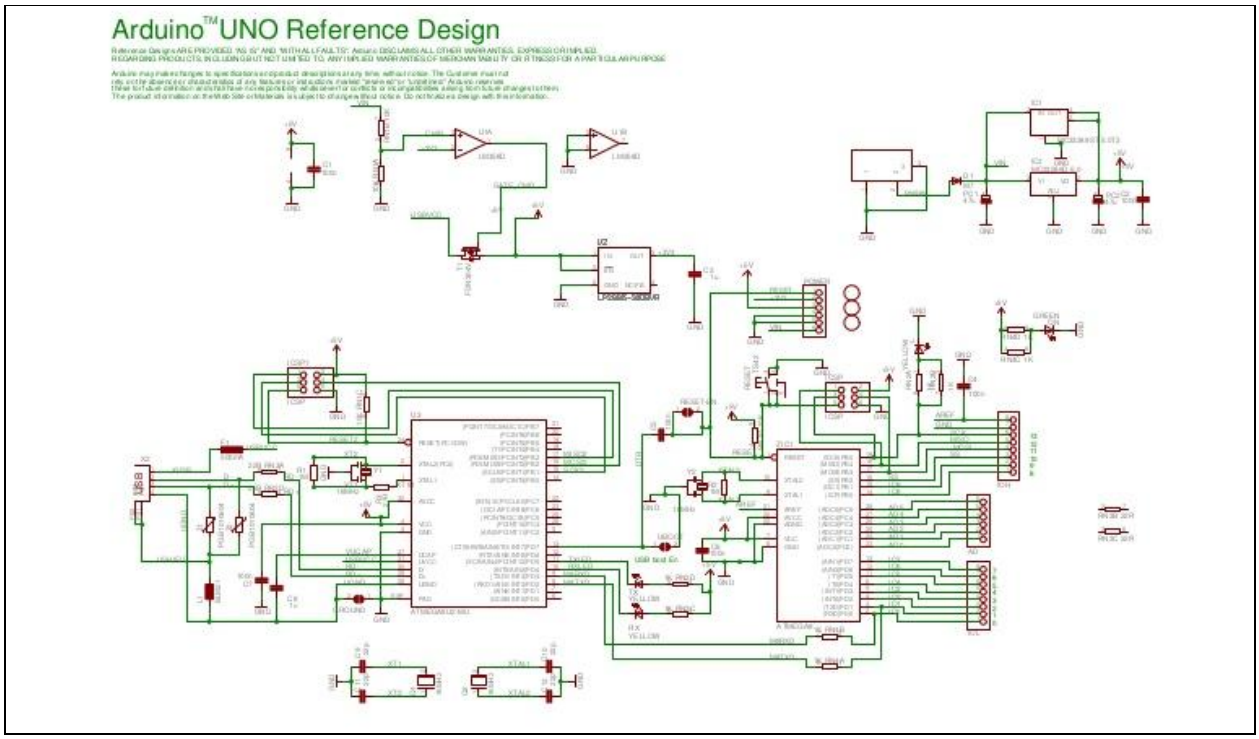

Fig. Arduino Uno Reference Design

Arduino Uno: Power

The Arduino Uno can be powered using the USB connection or with an external power supply. The power source is selected automatically by the board.

External or non-USB power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected to the board by plugging a 2.1mm centre-positive plug into the power jack. Leads from a battery can be inserted in the Ground pin and Vin pin headers of the POWER connector.

The board needs an external supply of 6 to 20 volts to operate. If supplied with less than 7V, the 5V pin may supply less than five volts and the board might be unstable. Using more than 12V can lead to the voltage regulator getting overheated and damaging the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN:** The input voltage to the Arduino board when it's using an external power source, as opposed to 5 volts from the USB connection or other regulated power source. You can supply voltage or access it through this pin (if supplying voltage via the power jack).

- **5V:** The regulated power supply used to power the components on the board. This can be either from VIN via an on-board regulator, or be supplied by USB or even another regulated 5V supply.
- **3V3:** A 3.3 volt supply is generated by the on-board regulator. Maximum current draw for the pin is 50 mA.
- **GND:** Ground pins.

Arduino Uno: Memory

The ATmega328 has 32 KB of which 0.5 KB is to be used for the bootloader. It also has 2 KB of SRAM and 1 KB of EEPROM.

Arduino Uno: Input and Output

Each of the 14 digital pins on the Arduino Uno can be used as an input or output, using the functions pinMode(), digitalWrite(), and digitalRead(). All of them operate at 5 volts. Each pin can provide or receive a maximum current of 40 mA and has an internal pull-up resistor, which is disconnected by default, of 20-50 kOhms. In addition to this, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** These pins are connected to the corresponding pins of the ATmega8U2USB-to-TTL Serial chip and are used to receive (RX) and transmit (TX) TTL serial data.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interruption when the value is LOW, a rising or falling edge, or a change in value.
- **PWM: 3, 5, 6, 9, 10, and 11.** You can use these pins to provide 8-bit PWM output with the analogWrite()function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins are used to support the SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. The LED is on when the pin is HIGH value and off when the pin is LOW value.

The Arduino Uno has 6 analog inputs, which are labeled A0 through A5, each of which provide 10 bits of resolution i.e. 1024 different values. Although they measure from ground to 5 volts by

default, it is possible to change the upper end of their range using the AREF pin and the analogReference() function.

In addition to that, some of the pins have specialized functions:

- **I2C: 4 (SDA) and 5 (SCL).** Used to support I2C (TWI) communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** This pin is the reference voltage for the analog inputs. It is used with the function analogReference().
- **Reset.** If you bring this line LOW, the microcontroller will reset. Typically used to add a reset button to shields which block the one on the board.

b) ATmega328 Microcontroller

ATmega328 is an Advanced Virtual RISC (AVR) microcontroller. It can support up to eight (8) bits of data. It has 32KB of internal builtin memory. Advanced Virtual RISC (AVR) is the family of microcontrollers which was developed by ATMEL in 1996. These microcontrollers are the modified Harvard Architecture 8-bit RISC Single Microcontroller Chip. [5]

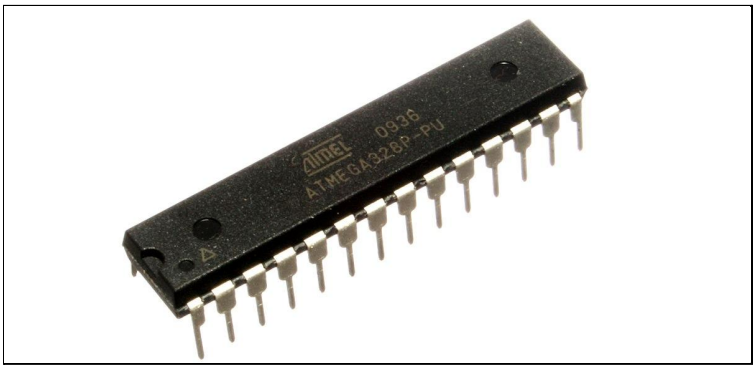

Fig. ATmega328 Microcontroller

ATmega328 also has 1KB Electrically Erasable Programmable Read Only Memory (EEPROM). This means that if the electric supply supplied to the microcontroller is removed, it can still store the data and can provide results after providing it with the electric supply. Furthermore, ATmega328 has 2KB Static Random Access Memory (SRAM).

ATmega328 has several other characteristics which make it the most popular device in today's market. Some of these features are advanced RISC architecture, good performance,

low power consumption, real timer counter having separate oscillator, 6 PWM pins, programmable Serial USART, programming lock for software security, throughput up to 20 MIPS etc. ATmega328 is most popularly used in Arduino.

ATmega328 Features	
No. of Pins	28
CPU	RISC 8-Bit AVR
Operating Voltage	1.8 to 5.5 V
Program Memory	32KB
Program Memory Type	Flash
SRAM	2048 Bytes
EEPROM	1024 Bytes
ADC	10-Bit
Number of ADC Channels	8
PWM Pins	6
Comparator	1
Packages (4)	8-pin PDIP32-lead TQFP28-pad QFN/MLF32-pad QFN/MLF
Oscillator	up to 20 MHz
Timer (3)	8-Bit x 2 & 16-Bit x 1
Enhanced Power on Reset	Yes
Power Up Timer	Yes
I/O Pins	23
Manufacturer	Microchip
SPI	Yes
I2C	Yes
Watchdog Timer	Yes
Brown out detect (BOD)	Yes
Reset	Yes
USI (Universal Serial Interface)	Yes
Minimum Operating Temperature	-40 C to +85 C

Fig. ATmega328 Overview

ATmega328 Pins:

- ➔ ATmega328 is an AVR Microcontroller that has twenty eight (28) pins in total.
- ➔ All of the pins are placed in a chronological order, and are listed in the table.

ATmega328 Pins			
Pin Number	**Pin Name**	**Pin Number**	**Pin Name**
1	PC6	15	PB1
2	PD0	16	PB2
3	PD1	17	PB3
4	PD2	18	PB4
5	PD3	19	PB5
6	PD4	20	AVCC
7	Vcc	21	AREF
8	GND	22	GND
9	PB6	23	PC0
10	PB7	24	PC1
11	PD5	25	PC2
12	PD6	26	PC3
13	PD7	27	PC4
14	PB0	28	PC5

Fig. Pin Number and Pin Name

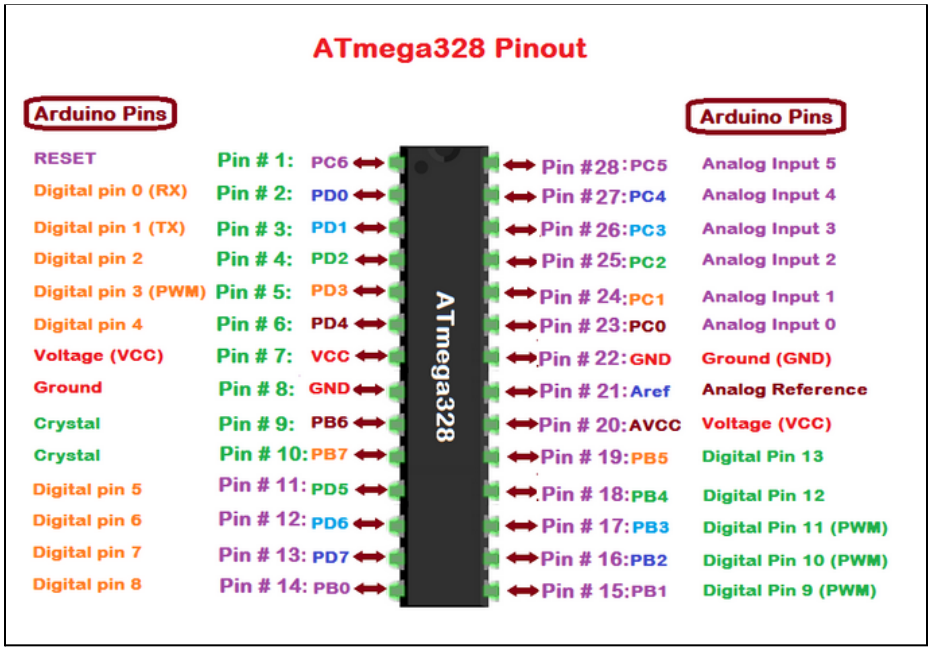

Fig. Pinout Diagram

The Pinout Diagram of ATmega328 helps us get a clear understanding of its configuration.

ATmega-328 pins are divided into various ports which are given in detail below:

- **VCC** denotes the digital voltage supply.
- **AVCC** is a supply voltage pin for the analog to digital converter.
- **GND** denotes Ground and it has a voltage of 0V.
- **Port A** consists of the pins from **PA0** to **PA7** which serve as analog input to analog to digital converters. If analog to digital converter is not used or required, **Port A** acts as an 8-bit bidirectional input/output port.
- **Port B** consists of the pins from **PB0** to **PB7** which is an 8 bit bidirectional port having an internal pull-up resistor.
- **Port C** consists of the pins from **PC0** to **PC7.** The output buffers of **port C** have symmetrical drive characteristics with source capability and high sink.
- **Port D** consists of the pins from **PD0** to **PD7** which is also an 8 bit input/output port having an internal pull-up resistor.
- **AREF** is an analog reference pin for analog to digital converter.

ATmega328 Architecture

The architecture of a device presents each information about the particular device. The architecture of ATmega328 is shown in the figure given below.

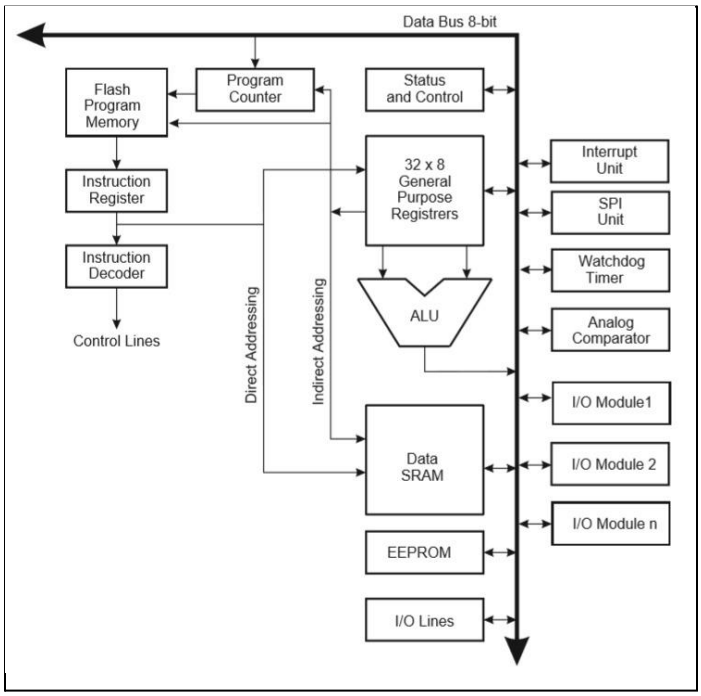

Fig. ATmega328 Architecture

ATmega328 Memory

ATmega 328 has three different types of memories e.g. EEPROM, SRAM etc. Flash Memory has a capacity of 32KB. It has an address of 15 bits. It is a non-volatile, Programmable Read Only Memory (ROM). SRAM stands for Static Random Access Memory which is a volatile memory i.e. data will be removed after removing the power supply. EEPROM is the abbreviation for Electrically Erasable Programmable Read Only Memory. It has long-term data.

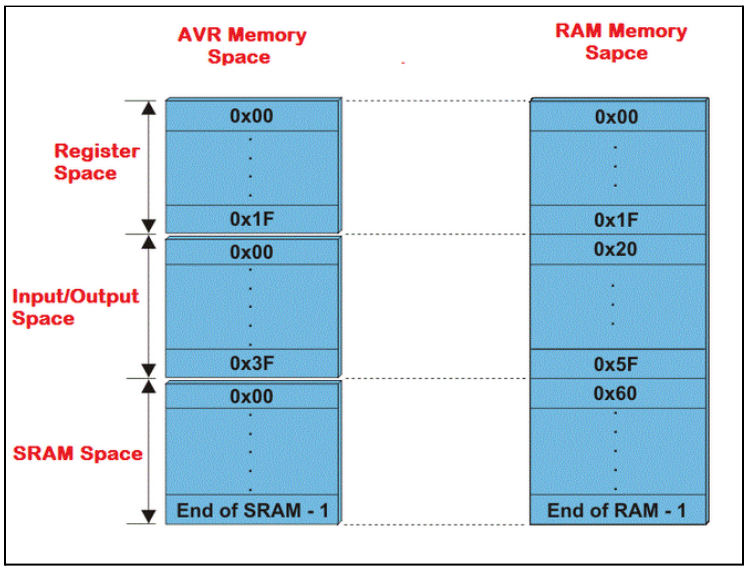

Fig. AVR Memory Spaces

ATmega328 Registers

ATmega-328 has 32 General Purpose registers. All of these registers are part of Static Random Access Memory (SRAM).

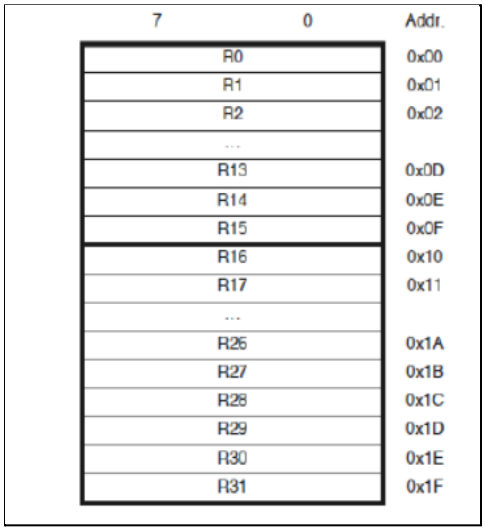

Fig. AVR General Purpose Registers

KY-037 Sound Sensing Module

KY-037 sound sensing module consists of a sensitive capacitance microphone for detecting sound and an amplifier circuit. The output of this module is both analog and digital.

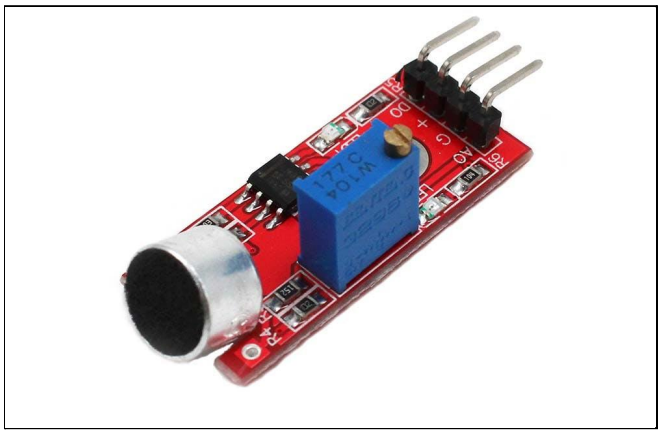

Fig. KY-037 Sound Sensing Module

KY-038 Sound Sensor having 4 Pins:

- ➔ AO – Analog Output
- ➔ G – Ground
- ➔ + – VCC (3.3V DC to 5V DC)
- ➔ DO – Digital Output

The digital output acts as a key and is activated when sound intensity has reached a certain threshold. The sensitivity threshold can be adjusted using the potentiometer on the sensor.[6]

The analog output voltage changes with the intensity of sound that is received by the microphone. Both these outputs can be connected to the Arduino module and output can be generated to show the detection of sound.

How a Sound Sensing Module Works?

Sound detection sensor works similarly to our Ears, having a diaphragm which converts the mechanical vibrations of sound into signals. However, what's different is that a sound sensor consists of an in-built capacitive microphone, peak detector and an amplifier that's highly sensitive to sound.[7]

With these components, it allows for the sensing module to work:

1. Sound waves propagate through the air molecules.
2. Such sound waves cause the diaphragm in the microphone to vibrate, which results in capacitive change.
3. This capacitance change is then amplified and digitalized for processing of sound intensity.

Project Workflow

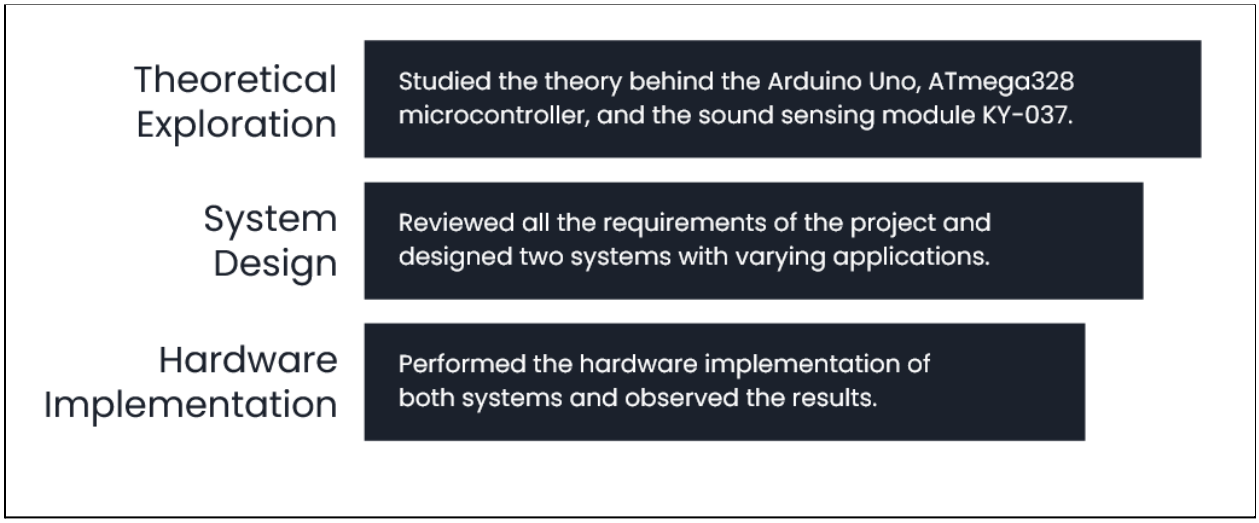

Fig. Project Workflow

Implementation

After studying each of the hardware components to be used in the project, the hardware implementation was performed. In this project, two types of sound detection systems were implemented using Arduino.

One was designed in a way just to detect if the sound disturbance caused the response of the sound sensing module to go smaller than the lower threshold or higher than the upper threshold. In the report, I have named it the Absolute Sound Detection System.

On the other hand, the second model was designed to produce a different response depending upon the bracket of response the sound disturbance was in. In the report, I have named it the Sequential Sound Detection System.

Note: The response when a disturbance was created was not necessarily positive. Hence, both upper and lower thresholds need to be taken for the system to be accurate.

But, before setting the intensity thresholds for both the systems we need to assess the steady state response using the Serial Monitor and Serial Plotter in the arduino programming the interface.

a) Finding the Steady State Response of the Module

We will use the Serial Monitor from the arduino programming interface to find the steady state response of the sound sensing module. We will also observe how it reacts to a sound disturbance using a Serial Plotter.

Circuit:

For finding the steady state, we will make the following connections:

- Connect the 5V power pin of the Arduino board to the +ve power pin of the sound sensor.
- Connect the GND power pin of the Arduino board to the GND pin of the sound sensor.
- Connect the A0 analog pin of the Arduino board to the Analog Output (AO) pin of the sound sensor.

Code:

The code for finding the steady state response of the sound sensing module is given below.

```
void setup() {
Serial.begin(9600);
pinMode(A0,INPUT);
pinMode(13,OUTPUT);

}

void loop() {
Serial.println(analogRead(A0));
delay(0.01);
}
```

In the setup block, A0 analog pin is set as INPUT, since it is connected to the sound sensor's response, and the digital pin 13 is set as OUTPUT. We don't need to connect an output device to the digital pin 13 because we're just concerned with finding out the steady state response in this section.

In the loop block, the function analogRead() is used to get the response of the sound sensor in loop. The code prints the periodic response of the sound sensor on the Serial Monitor.

Result:

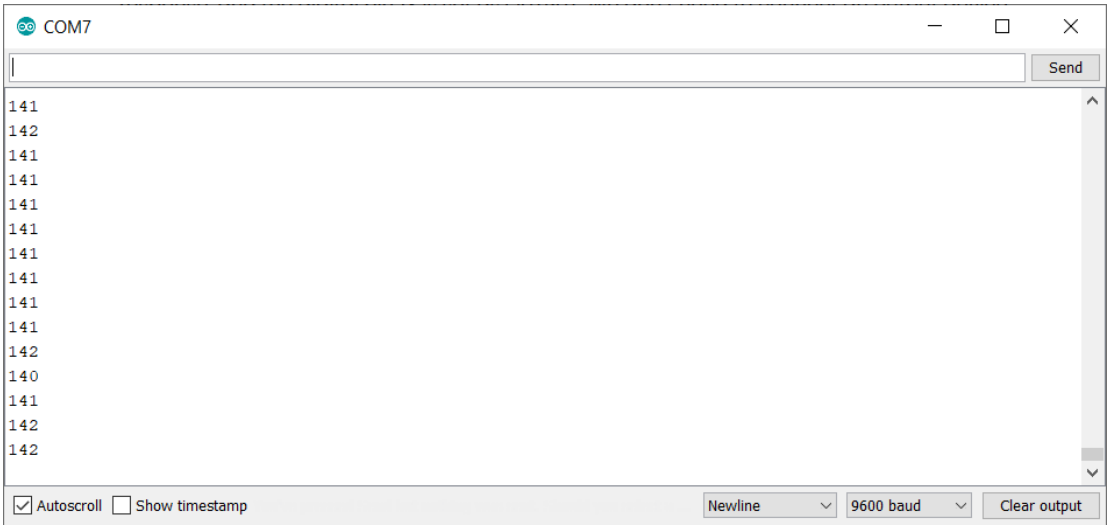

Fig. Serial Monitor Result (Without a Disturbance)

We can clearly see that the steady state value of the sensor's analog response is 141±1. Now that we know the steady value it would be easier for us to choose a threshold value

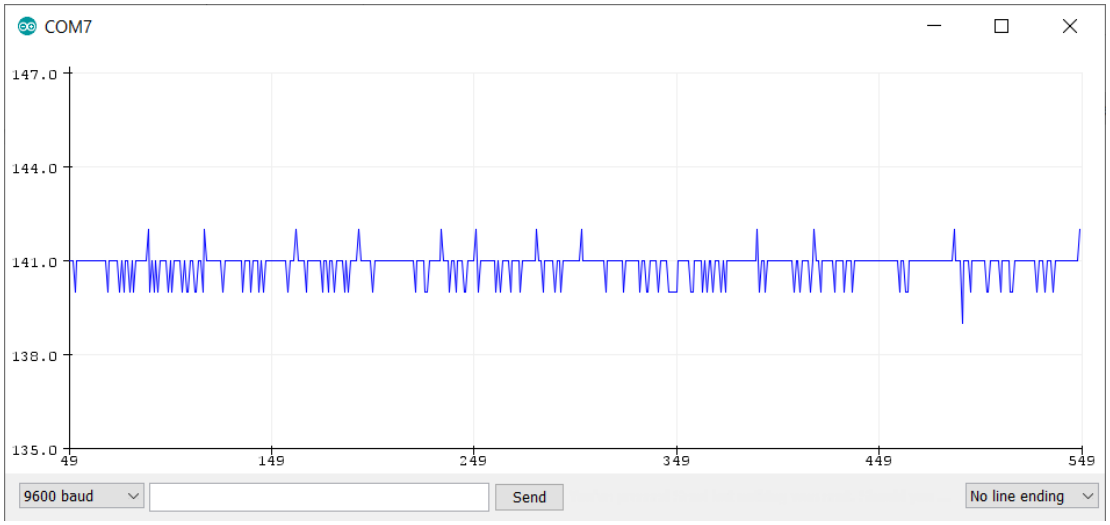

Fig. Serial Plotter Result (Without a Disturbance)

As we can see in the plot created by Serial Plotter, the analog response of the sound sensor lingers around 141. So, for the absolute sound detection system, we'll take the upper threshold as 155 and lower threshold as 125. We'll use an LED of red color to produce the output for this system.

Similarly, for the sequential sound detection system, we'll take four brackets of values and take a sequential approach to produce the output. We'll take four LEDs of different colors. The brackets of analog response which will be considered in the sequential sound detection system are given below along with their respective output combinations:

- 130 < Sensor's Response < 150: B
- 120 < Sensor's Response < 160: BG
- 110 < Sensor's Response < 170: BGW
- 100 < Sensor's Response < 180: BGWR

Note: Blue(B), White(W), Green(G) and Red(R) LEDs are used in this system. The output combinations stated above mention the color of LEDs that will be turned on at a particular interval of sensor's response, in sequence. For instance, "BGW" means that blue, green, and white LEDs will be ON in that exact order.

This combination will give a sequential output on the basis of intensity of the sound that is produced. This is why I have called it a Sequential Sound Detection System.

The Serial Plotter also shows the variation in response when a sound is created. Below is the graph of the response when a sound disturbance was created.

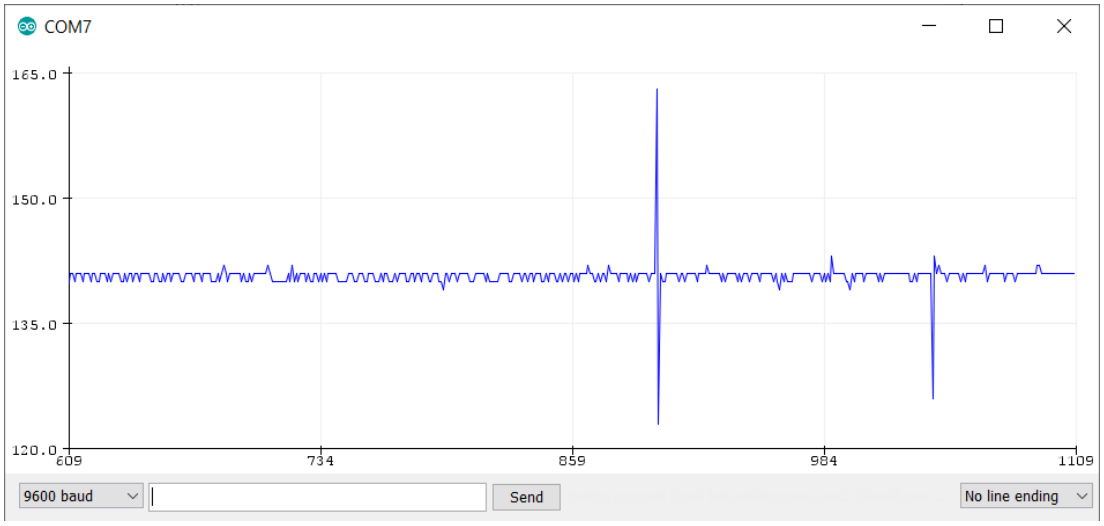

Fig. Serial Plotter Response (With a sound wave)

b) Absolute Sound Detection System

Now that we know the steady state value and have decided on the upper and lower threshold values, we can go ahead with implementing our sound detection system by making the circuit and uploading the code using the arduino programming interface.

Circuit:

For implementing the absolute sound detection system, we will make the following connections:

- Connect the 5V power pin of the Arduino board to the +ve power pin of the sound sensor.
- Connect the GND power pin of the Arduino board to the GND pin of the sound sensor.
- Connect the A0 analog pin of the Arduino board to the Analog Output (AO) pin of the sound sensor.
- Connect the anode of LED to the digital pin 13 and the cathode to the GND pin alongside the Digital pins.

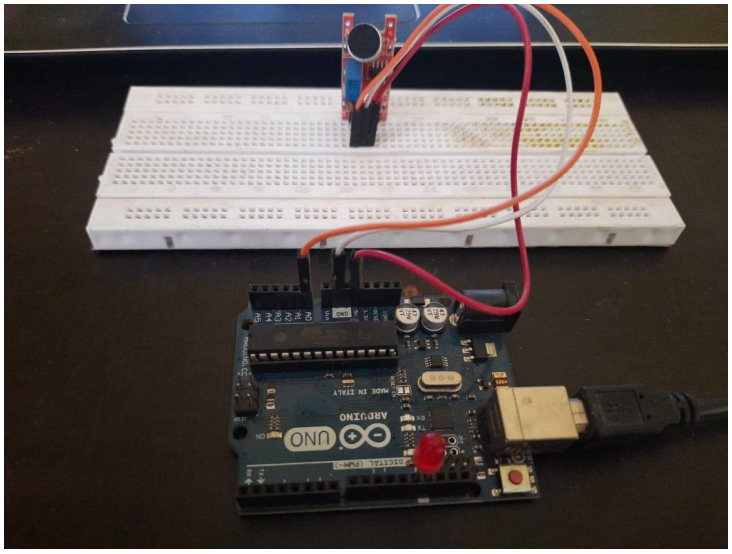

Fig. Circuit Setup

Code:

The code for the sequential sound detection model is given below.

```
const int ledpin=13;
const int soundpin=A0;

void setup() {
Serial.begin(9600);
pinMode(ledpin,OUTPUT);
```

```
pinMode(soundpin,INPUT);
}

void loop() {
Serial.println(analogRead(soundpin));
delay(0.1);
int soundsens=analogRead(soundpin);
if (soundsens>=155 or soundsens<=125 ) {
digitalWrite(ledpin,HIGH);
delay(1000);
}
else{
digitalWrite(ledpin,LOW);
}

}
```

In the code, the digital pin 13 which is connected to the LED is stored in the variable “ledpin”. Then, the response of analog pin A0 is stored in the variable “soundpin”, since the response of the sound sensing module is connected to analog pin A0.

In the setup block, the “soundpin” variable is set as INPUT and the “ledpin” variable is set as OUTPUT.

In the loop block of the code, first the response of the analog pin was read and then the response bracket conditions for the output were programmed using IF-ELSE statements.

Results:

The Output LED turned ON when the values of analog response from the sound sensor were greater than upper threshold or lower than the lower threshold.

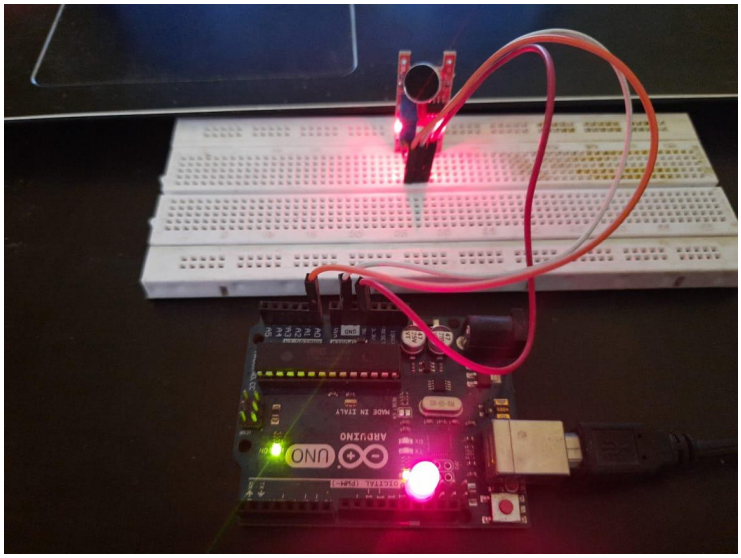

Fig. LED Output when Disturbance was Created

c) Sequential Sound Detection System

Circuit:

In the sequential sound detection model, the connections with the sound sensor remain the same as they were in the previously discussed model. But instead of using just the one digital pin (13) we used in the first model, we'll use four of them (13, 12, 11, 10). Each of the digital pins are connected to an LED with a different color. This will help us make a system where the sound created will trigger a different combination of LEDs depending on the response bracket that it is in.

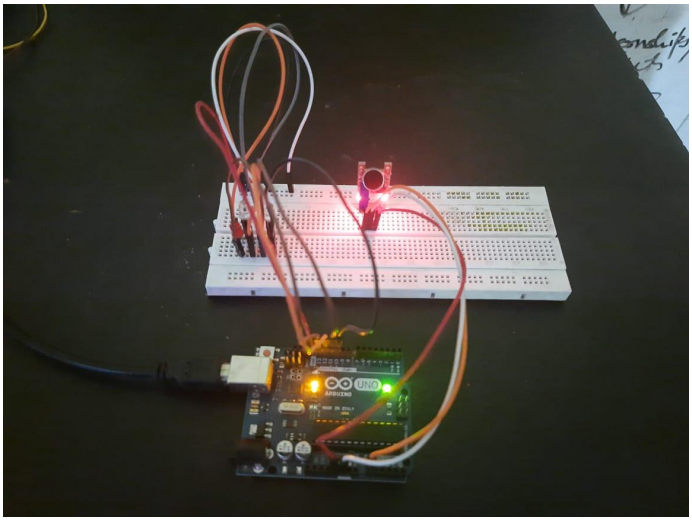

Fig. Fig. Circuit Setup

Code:

The code for the sequential sound detection model is given below.

```
const int ledpinr=13;
const int ledpinw=12;
const int ledping=11;
const int ledpinb=10;
const int soundpin=A0;

void setup() {
Serial.begin(9600);
pinMode(ledpinr,OUTPUT);
pinMode(ledpinw,OUTPUT);
pinMode(ledping,OUTPUT);
pinMode(ledpinb,OUTPUT);
pinMode(soundpin,INPUT);
}

void loop() {
Serial.println(analogRead(soundpin));
delay(10);
int soundsens=analogRead(soundpin);

if (soundsens>=150 or soundsens<=130 ) {
digitalWrite(ledpinb,HIGH);
delay(100);
}

if (soundsens>=160 or soundsens<=120 ) {
digitalWrite(ledping,HIGH);
delay(100);
}

if (soundsens>=170 or soundsens<=110 ) {
digitalWrite(ledpinw,HIGH);
delay(100);
}

if (soundsens>=180 or soundsens<=100 ) {
digitalWrite(ledpinr,HIGH);
delay(100);
}

else{
digitalWrite(ledpinb,LOW);
digitalWrite(ledping,LOW);
```

```
digitalWrite(ledpinw,LOW);
digitalWrite(ledpinr,LOW);
}

}
```

In the code, first each of the digital pins are being stored in a value depending on the color of LEDs they're connected with. For example, the red LED is connected to the digital pin 13. So, the variable that it is stored in is "ledpinr". Then, the response of analog pin A0 is stored in the variable "soundpin", since the analog response of the sound sensing module is connected to analog pin A0.

In the setup block, the "soundpin" variable is set as INPUT and the digital pin variables are set as OUTPUT. In the loop block of the code, first the response of the analog pin was read and then the output combinations according to the response brackets were programmed using conditional statements.

Results

In the sequential sound detection model, different combinations of LEDs were triggered depending upon the change in the response.

When the response after the sound disturbance was created was greater than 160 but less than 170, following response was created:

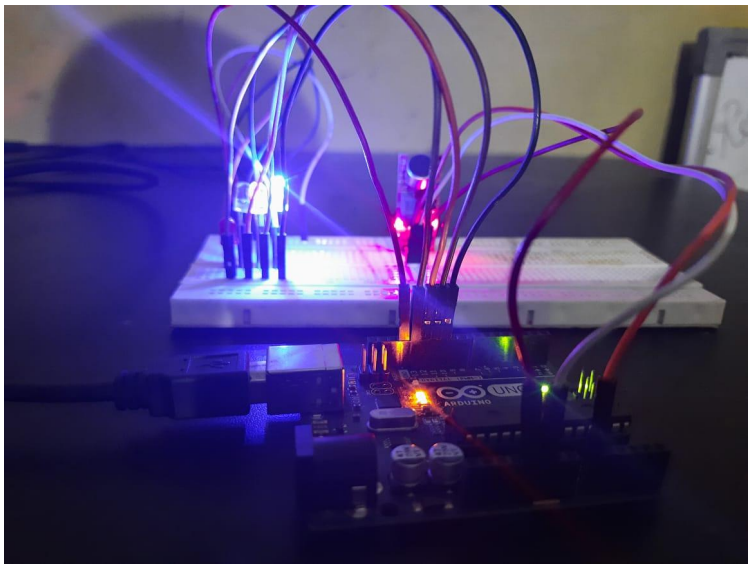

Fig. BG Combination because the response was between 160 and 170

Similarly, for other disturbances the combination of LEDs were triggered depending on the threshold bracket that the response was in.

Conclusion

In this project, the microcontroller board Arduino Uno, the microcontroller ATmega328, and the sound module KY-037 were studied for applications in sound detection systems.

The Arduino Uno proved to be a powerful microcontroller with multiple digital and analog pins to make rudimentary versions of both absolute and sequential sound detection systems. In both the systems, the arduino board was used to take input from the sound sensor and deliver output on the basis of the requirements of the systems.

The KY-037 proved to be a very flexible sound sensing module in terms of its sensitivity and response, having both digital and analog pins.

Both the systems gave the respectively desired outputs. Although rudimentary, both the systems can be used for security and monitoring applications. On one hand, the absolute sound detection system can be adjusted and used for applications like event-based sound detection like alerts for crash, loud noise, gunshots, etc. On the other hand, the sequential sound detection system can be adjusted according to the applications and trigger different sets of actions depending on the intensity of the sound detected.

References

[1]https://www-bsac.eecs.berkeley.edu/scripts/show_pdf_publication.php?pdfID=1365520205

[2]https://www.gartner.com/newsroom/id/3165317

[3]https://store.arduino.cc/usa/arduino-uno-rev3

[4]https://www.arduino.cc/en/Main/ArduinoBoardUnoSMD

[5]https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[6]https://create.arduino.cc/projecthub/electropeak/how-to-use-ky-037-sound-detection-sensor-with-arduino-a757a7

[7]https://www.seeedstudio.com/blog/2020/01/03/what-is-a-sound-sensor-uses-arduino-guide-projects/