|  |
| --- |
| #include <millisDelay.h> |

This is the header file we have imported. This is the library that uses the non-blocking method to replace delay() and millis() which has some issues so we avoid as much as we can; because

1. it is not non-blocking meaning code below it cannot run until it finishes execution

2. it is unsigned long ranging 0 to 4,294,967,295 meaning it resets to 0 in approx. 50 days and can mess up arithmetic conditions like unsigned subtraction, overflow

|  |
| --- |
| millisDelay buzzerDelay; |

It is the delay object; we have declared object buzzerDelay of type millisDelay

Ultrasonic sensor



* Ultrasonic sensor is a sensor that can measure distance emitting ultrasound at 40khz. Based on the time taken for this sound to travel back and forth, and the speed of the sound we calculate the distance.
* Ultrasonic sensor has 4 configuration pins; VCC, TRIG, ECHO, GND. The VCC is connected to +5V supply voltage. GND to GND and other two TRIG and ECHO to 8 and 9 Digital I/O in the Arduino Board.

First we need to define the Trigger Pin and Echo Pin that are connected to the Arduino board.

|  |
| --- |
| #define trigPin 8 //connect sensor trigPin to 8 #define echoPin 9 //connect echoPin to 9 |

In setup() function, we set *buzzer* and *trigPin* as OUTPUT and *echoPin* as INPUT

|  |
| --- |
| pinMode(buzzer, OUTPUT); //set buzzer as Output pinMode(echoPin, INPUT); //set echopin as input; sensor receives the signal pinMode(trigPin, OUTPUT); //set trigPin as output; sensor transmitter sends signal |

We have defined variables for the *halfDistance* and *timeTaken* as long and used boolean variable to set as flags to use as a condition.

|  |
| --- |
| long timeTaken, halfDistance; //32 bits number storage long thefirstDistance, currentDistance; boolean alarmBuzzDelay = false; //state of the buzzer delay boolean flagme = false; // State of the alarm boolean setAlarmTone = false; boolean alarmSet = false; boolean alarmActivate = false; |

*timeTaken* gives time taken for echo to receive ultrasonic sound after bouncing back

*halfDistance* gives the distance between sensor and the obstacle

|  |
| --- |
| long getDistance(){   digitalWrite(trigPin, LOW);  delayMicroseconds(2);  digitalWrite(trigPin, HIGH);  delayMicroseconds(10);   digitalWrite(trigPin, LOW); //now we turn off trigPin  timeTaken = pulseIn(echoPin, HIGH);   halfDistance = timeTaken\*0.034/2;   return halfDistance; //this returns the halfDistance value } |

The *getDistance()* function returns long data type variable.

In the *getDistance()* function, first you have to make sure that the trigPin is clear so we have to set that pin on a LOW State for just 2 µs.

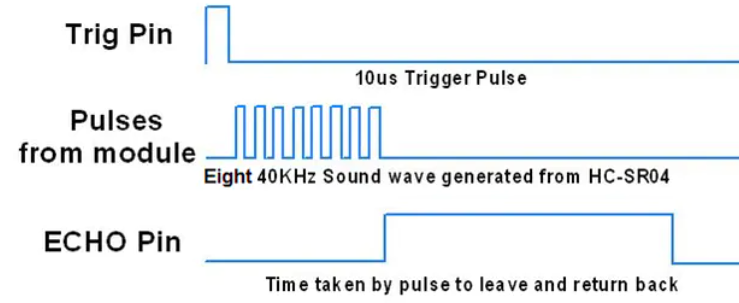
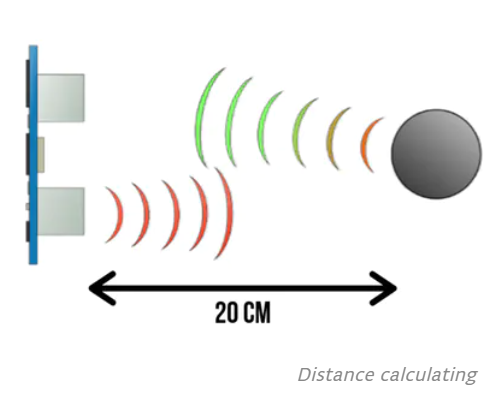
And to generate a 40KHz ultrasound wave we have to set the trigPin on HIGH State for 10 µs. That will send out an 8 cycle sonic burst which will travel at the speed sound and it will be received in the Echo Pin. Using the pulseIn()function you have to read the travel time and put that value into the variable “timeTaken”. This function has 2 parameters, name of the echo pin, ‘echoPin’ and for the second one you can write either HIGH or LOW. In this case, HIGH means that the pulseIn() function will wait for the pin to go HIGH caused by the bounced sound wave and it will start timing, then it will wait for the pin to go LOW when the sound wave will end which will stop the timing. This in the end, gives pulse duration in microseconds. As this duration is for the wave to leave the sensor, bounce on the obstacle and bounce back but we need the distance to the obstacle only so, we divide the distance by 2.

Speed of sound, v = 340m/s or 0.034 cm/µs

Time = distance/speed

t = s/v

s = t x v/2



|  |
| --- |
| void loop() { if (alarmSet){   alarmSet = false;  alarmActivate = true;   thefirstDistance = getDistance(); |

While loop() is running, when the user selects to activate alarm (user chooses A), boolean alarmSet flag is set true and callback getDistance function. The getDistance returns the halfDistance value and is stored in thefirstDistance which is the distance without any obstacle in front.

|  |
| --- |
| if (alarmActivate) { currentDistance = getDistance()+5;   if (currentDistance < thefirstDistance){   lcd.clear();  buzzerDelay.Start(10000);  alarmBuzzDelay = true; //set it to true  enterPassword(); //we callback this method. Should enter correct password to stop the alarm   } |

Then this runs as alarmActivate is set true. Here we constantly measure the distance for any obstacle. The value from getDistance callback is stored in currentDistance. We added 5 cm to getDistance to compensate upto 5cm extra distance for the accidental alarm trigger.

We use if (currentDistance < thefirstDistance) to constantly check if there is any invasion in our environment. If currentDistance is less than thefirstDistance, this means something has appeared in front of our sensor. Then it triggers alarm but we have used **non-blocking** method to hold 10 sec before *alarm sound* triggeringwhile the user can enter password to stop the alarm. This is done to lessen annoyance to the user and quickly prevent alarm buzzing in case of accidental trigger.

buzzerDelay.Start(10000); This uses Start() method from millisDelay.h header file.

The syntax is < Start(delay\_time\_in\_milliseconds) > It starts the timer for 10 seconds but this is a non-blocking method meaning code below it still runs while the timer is running.

|  |
| --- |
| void enterPassword() { //user inputs password and compares set password  flagme = true;  ..  ..  while(flagme) { //this does not run until the buzzerDelay completes 10 second  if (alarmBuzzDelay && buzzerDelay.justFinished()) { alarmBuzzDelay = false; tone(buzzer, alarmTone); //the alarm will continue buzzing while user has to enter correct password  }  pressedKey = myKeypad.getKey(); |

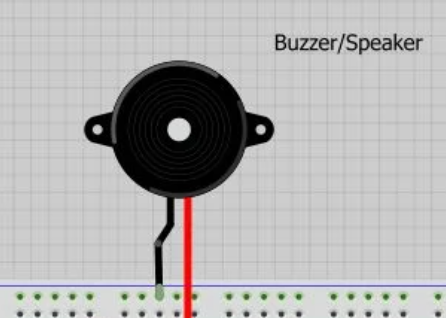
*flagme* is set true and while loop runs then check if buzzerDelay has reached 10sec. So until *alarmBuzzDelay* is true and *buzzerDelay* has completed 10 second only then the alarm starts buzzing. *justFinished()* is the method from millisDelay.h header that checks if the delay time which was 10sec has passed.

Syntax < tone(buzz pin name, frequency\_in\_Hz, duration) >

1. tone() generates square wave of the specified frequency on a pin.
2. Duration can be set
3. Until noTone() is called it will not stop producing waves
4. It can generate as low as 31Hz.
5. Only one tone can be played at a time. noTime(pin) should be called for multiple pins.
6. *duration* can be unsigned long only
7. *Frequency* parameter can only be unsigned int and in Hz.
8. tone() returns nothing

|  |
| --- |
| unsigned int alarmTone=900; |

*alarmTone* is declared as unsigned int as this is the only accepted data type by tone() method. This means it can hold 0 to 65535 value.



Buzzer

* It is a device that emits sound with a certain customized frequency.
* It can be controlled using the tone() built-in method.

Syntax < tone(buzzer\_name, frequency, time\_in\_millisecond) >

We have used tone() several times to give feedback to the user when pressing a key, when the process is completed, user errors, or set the alarm off.

tone(buzzer, 2000, 100) This produces 2000 Hz frequency of sound for 100ms. Thus, depending on the frequency and time we can set different sounds for better user experience (UX).

Here, we have connected the buzzer to pin 7 in arduino board and define buzzer pin.

|  |
| --- |
| #define buzzer 7 //connect buzzer to pin 7 in arduino board |

In line 200, when user has entered the alarm tone

|  |
| --- |
| if (pressedKey == '\*' && (tempAlarmTone.length() != 0)) { alarmTone = tempAlarmTone.toInt();  if (alarmTone > 20 && alarmTone <= 20000){  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Set Tone..done!"); tone(buzzer,alarmTone,3000); //this produces the sample alarm tone set by the user for 3 seconds  setAlarmTone = false; //this will help exit the loop  } |

Here if (pressedKey == '\*' && (tempAlarmTone.length() != 0)) checks if user has pressed ‘\*’ and also the string *tempAlarmTone* is not empty.

alarmTone = tempAlarmTone.toInt(); the toInt() is the built-in method that converts the tempAlarmTone String value to integer value and stores in alarmTone which we will use to set the alarm tone. But we will allow alarmTone to be 20-20kHz only because that is our normal hearing range. Otherwise the user has to enter the frequency again.

|  |
| --- |
| alarmActivate = false; noTone(buzzer); |

noTone() is used to silence the buzzer.

syntax <noTone (buzzer\_name)>