**Accelerometer Based Bike Bell**

**General Description:**

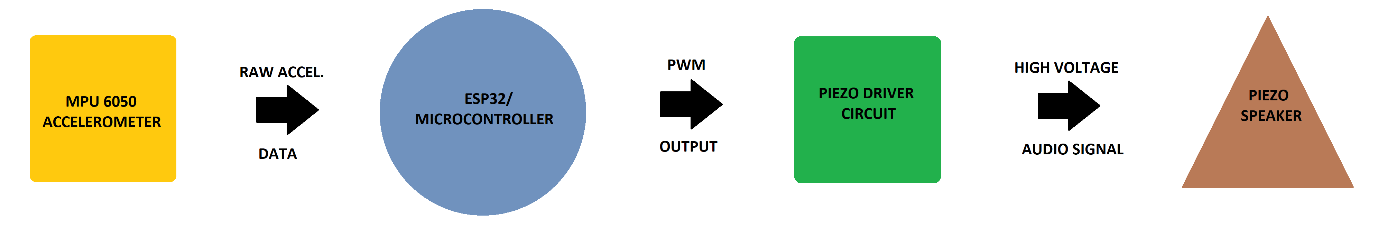
Unlike traditional electronic bells which ring on push of button this bell makes use of a microcontroller (ESP32) and an accelerometer sensor. The bell is nothing but a piezoelectric speaker actuated by the microcontroller according to the signals received from the accelerometer. The accelerometer provides the live acceleration value to the microcontroller and these values are then processed by the microcontroller to detect sudden changes in the motion of the bicycle. The intensity of motion is mapped with the volume of the piezoelectric speaker, in such a way that the volume of piezoelectric speaker is modulated by the changing acceleration values.

This device provides selectable audio files in wav format which can be updated using the WIFI or Bluetooth of the microcontroller (ESP32).

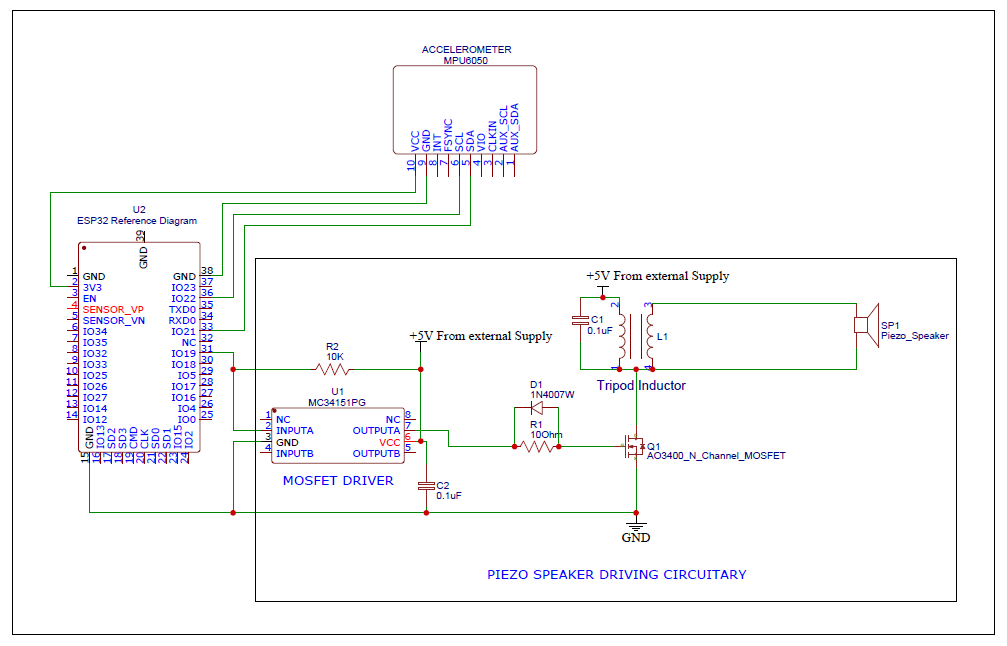
**Components Used:**

|  |  |  |  |
| --- | --- | --- | --- |
| Serial | Components Name | Reference Image | Quantity |
| 1 | ESP32 |  | 1 |
| 2 | Piezoelectric Speaker |  | 1 |
| 3 | 10 Ohm, 10KOhm |  | 1,1 |
| 4 | AO3400 N-Channel MOSFET |  | 1 |
| 5 | 0.1uF Capacitor |  | 2 |
| 6 | MC34151PF MOSFET Driver |  | 1 |
| 7 | MPU6050 accelerometer |  | 1 |
| 8 | 3 pin Inductor |  | 1 |
| 9 | 1N4007 Diode |  | 1 |

**Block Diagram:**



**Schematics:**



**Working:**

The device consists of an Inertial Measurement Unit (MPU6050), ESP32 (microcontroller), MC34151 MOSFET driver IC, Tripod Inductor (3 pin inductor/transformer), Piezoelectric Speaker, AO3400 N-Channel MOSFET, and a few other passive components. The ESP32 communicates with the MPU6050 using I2C protocol, the ESP32 requests the data from MPU6050 and the MPU6050 responds by raw 16-bit long acceleration data in X, Y, and Z direction. Using the acceleration values obtained from the MPU6050 net acceleration is calculated and then this acceleration value is stored in a circular buffer. The circular buffer is used in order to store the previous few values of the acceleration. Using the previous values of the net acceleration, average acceleration is calculated. This average acceleration is then subtracted from the current value of net acceleration which gives us the instantaneous variation in acceleration (approximately). This variation is directly liked with the motion of the vehicle on which the bell is attached.

After getting the variation, the value is checked against the least possible value to trigger the volume. Doing so removes the very slow and random low amplitude noise. After checking for the least value, if the variation is strong enough then volume is calculated. The volume is directly proportional to the variation.

Now coming to the actuation part, to actuate the speaker we need constant supply of audio signal. This is obtained from the audio file stored in the internal memory of the ESP32. The supported format for audio file is **.wav**. In the very beginning at the time of ESP32 start-up a 100,000 long 8-bit wide buffer is made to store the audio stream. File is read from the internal storage, depending upon the bits per sample, no of channels present and the sampling frequency, different operations are done on the data to convert it into 8 bit per sample, single channel, and same sampling frequency. After doing all this, a PWM channel is made to produce the output PWM on one of the pins of ESP32. Using the delay in microseconds, precise delay is introduced between the write of each PMW value in order to obtain different sampling frequency output.

Below is the table show the max length of the audio which can be played in loop on ESP32 with different sampling frequency, given the 8-bit buffer size of 100,000:

|  |  |  |
| --- | --- | --- |
| Serial No. | Sampling Frequency | Playback Time |
| 1 | 8000 Hz | 12.5 sec |
| 2 | 11025 Hz | 9.07 sec |
| 3 | 16000 Hz | 6.25 sec |
| 4 | 22050 Hz | 4.53 sec |
| 5 | 32000 Hz | 3.125 sec |
| 6 | 44100 Hz | 2.267 sec |
| 7 | 48000 Hz | 2.083 sec. |

As of now only the above written sampling frequencies are supported by the device. The performance of the device is optimum for the frequencies 16000, 22050, and 32000Hz. These frequencies strike a good balance between the playback length and the audio quality. The frequencies below 16000 does not produce nice audio notes, and the frequencies above 32000Hz does not provide long playback time.

Note: The playback time given in the table above is the **max** playback time for a single file. There is option to store multiple audio file in the internal storage using **SPIFFS**.

**Important information:**

* In this device a three-pin inductor/ transformer is used. This device is used to boos the low voltage 3.7V signal to high voltage signal. I had used the transformer from the electric bell device I ordered online. Depending on your device operation, the turn ration and inductance can be different which can ultimately affect the loudness of the sound produced by the inductor.
* The primary side of the transformer is connected between the 5V (external supply) and MOSFET. The secondary side of the transformer is connected to the piezoelectric speaker. As this transformer is a 3-pin device, it becomes difficult to know which pins are for primary and which one is for secondary.
* To find the primary and secondary terminals of the inductor, use a multimeter to find the resistance between the three terminals taken two at a time. You will find three different values of the resistance. Suppose the R1, R2, and R3 are the values of resistance obtained such that R1 > R2 > R3. Then R3 is the resistance of the primary side and R1 is the resistance of the secondary side. So, connect accordingly.
* If you are not getting loud output then you can try different values of the C1 cap in the range 0.01uF to 3uF. The capacitors are required to be ceramic. Polar capacitors won’t work.
* Piezo electric driver circuit requires its own power supply so don’t try to power the driver circuit using the 3.3V output from the ESP32.
* The power to MPU6050 is necessarily supplied from ESP32 3.3V output only.