UE22CS342AA3: IoT
Mini Project

Title

Smart Ultrasonic Theremin

A musical device that produces sound without any contact with it, including cloud connectivity.

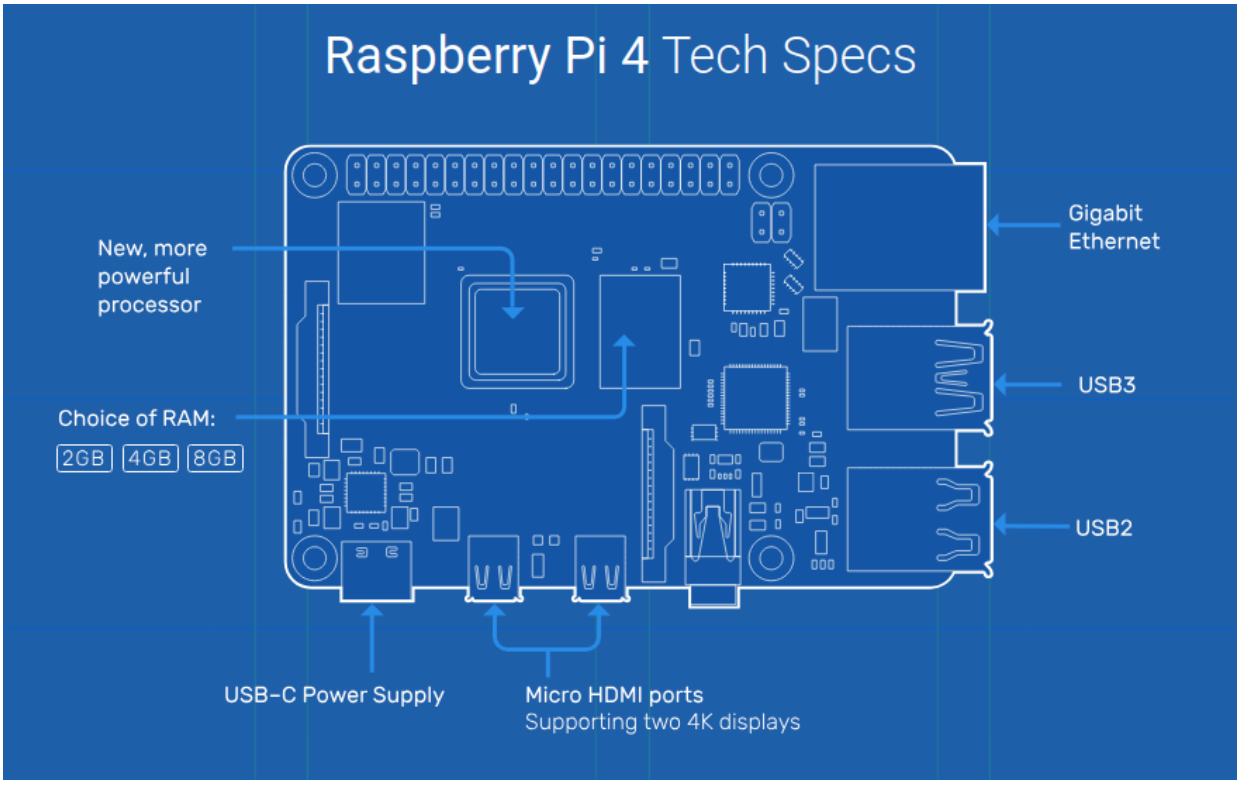
Joshua John **PES2UG22CS238**

Justin Carvalho **PES2UG22CS242**

Hardware Components:

- Raspberry Pi 4 Model B
- Ultrasonic Sound Sensors
- Breadboard
- Resistors 3 x 1k
- Jumpers
- Bluetooth Speaker

Development Board Specification:

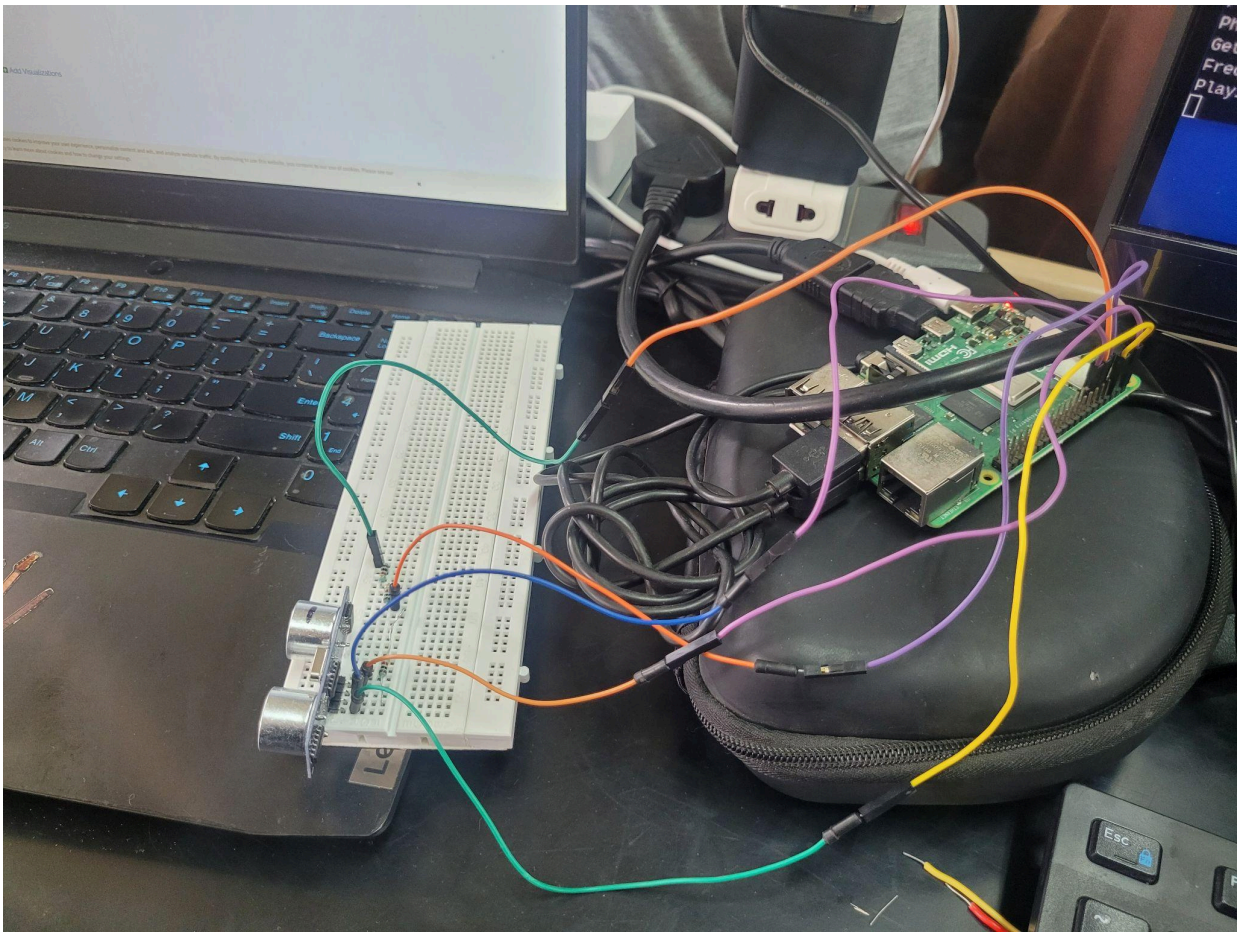


- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
- 8GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI® ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

Circuit Diagram:

Connections:

Raspberry Pi		**Ultrasonic Sound Sensor**
• 5V	->	VCC
• GPIO 17	->	TRIG
• GPIO 16	->	ECHO (connected via voltage divider)
• GND	->	GND



Code:

Source Files:

1) main.c

```
// main.c

#include        <stdio.h>
#include        "sensor.h"
#include        <wiringPi.h>
#include        "sound.h"
#include        <stdlib.h>
#include        <string.h>
#include        <curl/curl.h>

int main (void)
{

  // Initialize WiringPi and check for errors
  if (wiringPiSetup () == -1) {
            printf ("WiringPi setup failed!\n");
            return 1;           // Exit with error
            }

  printf ("WiringPI initialized!\n");

  setup_sensor ();
  printf ("Sensor initialized!\n");
```

```
float distance, distance_p;
float frequency;

// Main loop
while (1) {
    printf ("Getting distance...\n");
    distance = get_distance_p ();
    printf ("Pitch Distance: %.2f cm\n", distance);

    // Pre-processing to remove outliers:

    distance_p = remove_outliers(distance);
    printf("Pre-processed Pitch Distance: %.2f cm\n", distance_p);

    printf("Phase 2: sound.\n");

    printf("Get frequency.\n");
    frequency = get_frequency(distance_p);
    printf("Frequency: %.2f\n", frequency);

    printf("Playing sound.\n");
    play_sound(frequency, 0.2);

    CURL *curl;
    CURLcode res;

    // ThingSpeak Write API key
    const char *write_api_key = "5V0G0Q6PWNDSFSVD";

    // URL for ThingSpeak API
```

```
char url[200];
snprintf(url, sizeof(url), "https://api.thingspeak.com/update?api_key=%s", write_api_key);

// Data to send
float cloud_distance = distance_p;
float cloud_frequency = frequency;

// Prepare data to send
char data[200];
snprintf(data, sizeof(data), "&field1=%.2f&field2=%.2f", cloud_distance, cloud_frequency);

// Initialie CURL
curl_global_init(CURL_GLOBAL_DEFAULT);
curl = curl_easy_init();

if(curl) {
// set the url for post request
curl_easy_setopt(curl, CURLOPT_URL, url);

// set the data to be sent as part of the request
curl_easy_setopt(curl, CURLOPT_POSTFIELDS, data);

// send the HTTP post request
res = curl_easy_perform(curl);

if (res != CURLE_OK) {
fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
} else {
printf("Data sent to ThingSpeak successfully.\n");
}
```

```
        // Clean CURL
        curl_global_cleanup();
        }




        delay(50);
  }
}
```

2) sensor.c

```
// sensor.c - sensor setup and distance measurement code

#include        <wiringPi.h>
#include        "sensor.h"
#include        <stdio.h>

void setup_sensor()
{
        wiringPiSetupGpio();            // Use BCM GPIO numbering
        pinMode(TRIG_PIN, OUTPUT);      // Set Trigger pin as output
        pinMode(TRIG_PIN_2, OUTPUT);    // Set Trigger pin 2 as output

        pinMode(ECHO_PIN, INPUT);       // Set Echo pin as input
        pinMode(ECHO_PIN, INPUT);       // Set Echo pin 2 as input

        digitalWrite(TRIG_PIN, LOW); // Initialize Trigger pin to low
        digitalWrite(TRIG_PIN_2, LOW);  // Initialize Trigger pin 2 to low
```

```
        delay(30);
}

// Function to trigger the ultrasonic sensor and measure the distance

float get_distance_p ()
{
        // Send a pulse to trigger the ultrasonic sensor
        printf("Trigger pin 1 high.\n");
        digitalWrite(TRIG_PIN, HIGH);

        delayMicroseconds(10);          // Pulse width of 10 microseconds

        printf("Trigger pin 1 low.\n");
        digitalWrite(TRIG_PIN, LOW);

        // Wait for the echo pin to go HIGH and measure the pulse duration
        printf("digitalRead low.\n");
        while (digitalRead(ECHO_PIN)  == LOW) {
        // Waiting for the Echo pins to go HIGH
        }

        long start_time = micros();       // Record the start time

        printf("digitalRead high.\n");
        while (digitalRead(ECHO_PIN) == HIGH) {
        // Waiting for Echo pin to go low
        }

        long end_time = micros();         // Record the end time
```

```
        // Calculate the pulse duration
        long duration = end_time - start_time;

        // Calculate the distance in cm
        float distance = (duration / 2.0 ) * 0.0343;

        printf("Ending get_distance_p\n");

        return distance; // Return distance in cms
}

float remove_outliers(float distance)
{
        if (distance < 5) {          // To set the minimum distance possible
        return 5;
        } else if (distance > 400) {         // To set the max distance possible
        return 400;
        } else {
        return distance;
        }
}
```

3) sound.c

```
// sound.c

#include  "sound.h"

float get_frequency(float distance)
{
```

```
  const float min_distance = 3.0;
  const float max_distance = 50.0;
  const float min_frequency = 220;
  const float max_frequency = 880;

  // Linear mapping formula
  float frequency = min_frequency + (distance - min_distance) *
          (max_frequency - min_frequency) / (max_distance - min_distance);

  // Clamp frequency to the valid range
  if (frequency < min_frequency)
          frequency = min_frequency;
  if (frequency > max_frequency)
          frequency = max_frequency;

  return frequency;
}

void play_sound(float frequency, float duratio)
{
  snd_pcm_t *pcm_handle;
          int pcm;
          unsigned int rate = SAMPLE_RATE;
          int channels = 1; // Mono
          snd_pcm_uframes_t frames = 32;
          snd_pcm_hw_params_t *params;

          // Open the PCM device
          pcm = snd_pcm_open(&pcm_handle, "default", SND_PCM_STREAM_PLAYBACK, 0);
          if (pcm < 0) {
```

```
        fprintf(stderr, "ERROR: Cannot open PCM device: %s\n", snd_strerror(pcm));
        return;
        }

        // Set hardware parameters
        snd_pcm_hw_params_malloc(&params);
        snd_pcm_hw_params_any(pcm_handle, params);
        snd_pcm_hw_params_set_access(pcm_handle, params,
SND_PCM_ACCESS_RW_INTERLEAVED);
        snd_pcm_hw_params_set_format(pcm_handle, params, SND_PCM_FORMAT_FLOAT);
        snd_pcm_hw_params_set_channels(pcm_handle, params, channels);
        snd_pcm_hw_params_set_rate_near(pcm_handle, params, &rate, 0);
        snd_pcm_hw_params(pcm_handle, params);
        snd_pcm_hw_params_free(params);
        snd_pcm_prepare(pcm_handle);

        // Generate and play the sin wave
        int num_samples = SAMPLE_RATE * duratio;
        float buffer[frames * channels];
        for (int i = 0; i < num_samples / frames; i++) {
        for (int j = 0; j < frames; j++) {
        buffer[j] = AMPLITUDE * sinf(2.0 * M_PI * frequency * (i * frames + j) / SAMPLE_RATE);
        }
        snd_pcm_writei(pcm_handle, buffer, frames);
        }

        // Clean up
        snd_pcm_drain(pcm_handle);
        snd_pcm_close(pcm_handle);
}
```

Header files:

1) sensor.h

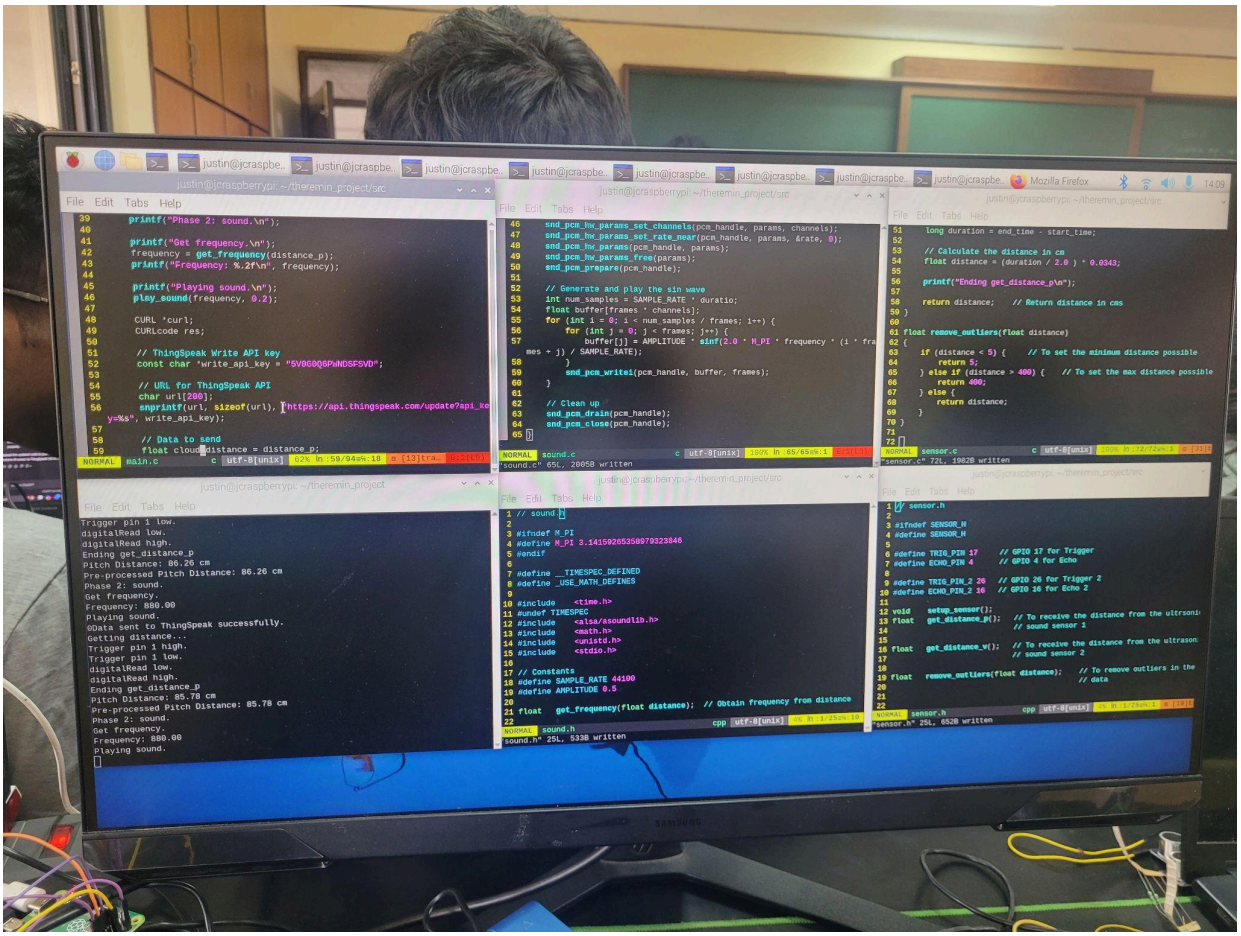
```
// sensor.h

#ifndef SENSOR_H
#define SENSOR_H

#define TRIG_PIN 17    // GPIO 17 for Trigger
#define ECHO_PIN 4     // GPIO 4 for Echo

#define TRIG_PIN_2 26   // GPIO 26 for Trigger 2
#define ECHO_PIN_2 16   // GPIO 16 for Echo 2

void      setup_sensor();
float   get_distance_p();   // To receive the distance from the ultrsonic
                    // sound sensor 1

float   get_distance_v();   // To receive the distance from the ultrasonic
                    // sound sensor 2

float   remove_outliers(float distance);    // To remove outliers in the
                                    // data




#endif  // SENSOR_H
```

2) sound.h

```
// sound.h

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define __TIMESPEC_DEFINED
#define _USE_MATH_DEFINES

#include          <time.h>
#undef TIMESPEC
#include          <alsa/asoundlib.h>
#include          <math.h>
#include          <unistd.h>
#include          <stdio.h>

// Constants
#define SAMPLE_RATE 44100
#define AMPLITUDE 0.5

float   get_frequency(float distance);  // Obtain frequency from distance

void      play_sound(float frequency, float duratio); // Generate and play
                                        // sin wave
```

Makefile:

```
CC = gcc

CFLAGS = -Iinclude

LIBS = -lwiringPi -lasound -lm -lcurl


all: theremin


theremin: src/main.o src/sensor.o src/sound.o

	$(CC) -o theremin src/main.o src/sensor.o src/sound.o $(LIBS)


src/main.o: src/main.c

	$(CC) -c src/main.c -o src/main.o $(CFLAGS) $(LIBS)


src/sensor.o: src/sensor.c

	$(CC) -c src/sensor.c -o src/sensor.o $(CFLAGS) $(LIBS)


src/sound.o: src/sound.c

	$(CC) -c src/sound.c -o src/sound.o $(CFLAGS) $(LIBS)


clean:

	rm -f src/*.o theremin
```


printf("Phase 2: sound.\n");
printf("Get frequency.\n");
frequency = get_frequency(distance_p);
printf("Frequency: %.2f\n", frequency);
printf("Playing sound.\n");
play_sound(frequency, 0.2);
// ThingSpeak Write API key
// URL for ThingSpeak API
// Data to send
snd_pcm_hw_params_set_channels(pcm_handle, params, channels);
snd_pcm_hw_params_set_rate_near(pcm_handle, params, &rate, 0);
snd_pcm_hw_params(pcm_handle, params);
snd_pcm_hw_params_free(params);
snd_pcm_prepare(pcm_handle);
// Generate and play the sin wave
// Clean up
long duration = end_time - start_time;
// Calculate the distance in cm
float distance = (duration / 2.0) * 0.0343;
printf("Ending get_distance_p\n");
return distance; // Return distance in cms
float remove_outliers(float distance)
if (distance < 5) { // To set the minimum distance possible
return 5;
} else if (distance > 400) { // To set the max distance possible
return 400;
return distance;
Trigger pin 1 low.
digitalRead low.
digitalRead high.
Ending get_distance_p
Pitch Distance: 86.26 cm
Pre-processed Pitch Distance: 86.26 cm
Phase 2: sound.
Get frequency.
Frequency: 880.00
Playing sound.
0Data sent to ThingSpeak successfully.
Getting distance..
Trigger pin 1 high.
Trigger pin 1 low.
digitalRead low.
digitalRead high.
Ending get_distance_p
Pitch Distance: 85.78 cm
Pre-processed Pitch Distance: 85.78 cm
Phase 2: sound.
Get frequency.
Frequency: 880.00
Playing sound.
// sound.H
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
#define __TIMESPEC_DEFINED
#define _USE_MATH_DEFINES
#include <time.h>
#undef TIMESPEC
#include <alsa/asoundlib.h>
#include <math.h>
#include <unistd.h>
#include <stdio.h>
// Constants
#define SAMPLE_RATE 44100
#define AMPLITUDE 0.5
float get_frequency(float distance); // Obtain frequency from distance
"sound.c" 65L, 2005B written
"sound.h" 25L, 533B written
// sensor.h
#ifndef SENSOR_H
#define SENSOR_H
#define TRIG_PIN 17 // GPIO 17 for Trigger
#define ECHO_PIN 4 // GPIO 4 for Echo
#define TRIG_PIN_2 26 // GPIO 26 for Trigger 2
#define ECHO_PIN_2 16 // GPIO 16 for Echo 2
void setup_sensor();
float get_distance_p(); // To receive the distance from the ultrsonic
// sound sensor 1
float get_distance_w(); // To receive the distance from the ultrason
// sound sensor 2
float remove_outliers(float distance); // To remove outliers in the
// data
"sensor.c" 72L, 1982B written
"sensor.h" 25L, 652B written

Cloud Platform:

Platform used: ThingSpeak - to transfer distance and frequency data.

Graph Implementation and Visualization: