California State University, Northridge
Department of Electrical & Computer Engineering

Final Project Report: LaunchTone

Zachary Martin
ECE 595R - Robotics & Embedded Systems
Shahnam Mirzaei, Ph.D.
May 16, 2025

LaunchTone: A Musical Instrument Made Using The Texas Instruments MSP432 LaunchPad Microcontroller

Zachary Martin
College of Engineering and Computer Science, Electrical and Computer Engineering Department
California State University, Northridge, California, United States of America
zachary.martin.144@my.csun.edu

Abstract—This paper outlines the design and implementation of a touchless musical instrument as the final project in a Robotics and Embedded Systems university course. The instrument uses the TI LaunchPad MSP432-based microcontroller board, ultrasonic sensors, and speakers to play sounds with frequencies modulated by the user's hands. Code Composer Studio is used to write software using multiple LaunchPad modules that enable various functions. The instrument was not successfully completed as described but did generate musical notes with pitch proportional to the distance of a user's hand to an ultrasonic sensor.

***Index Terms*—3D printing, embedded systems programming, frequency modulation, microcontrollers, musical instruments, sensor interfacing, ultrasonic sensors**

I. Introduction

A. Background

Music is present in all known societies and has been an integral part of the human experience for millenia. A flute made of mammoth ivory was discovered in Geissenklösterle cave in southern Germany; the flute, which played a five-note scale, is estimated to be approximately 35,000 years old [1]. Modern musical instruments of all kinds are rich with embedded electronics. Some use electronics to enhance their sound, in the case of electric guitars or drums, and others produce sound exclusively from electronics, such as synthesizers. One lesser-known musical instrument which uses electronics is the theremin.

In the early 1900s, when working with an amplifier circuit, Léon Theremin observed that he could modulate the frequency by moving his hand nearer to or farther from the antenna [2]. Shortly after, he had invented the eponymous instrument, originally called the ætherphone [3]. The theremin is played by moving one hand around a rod controlling pitch and the other around a rod controlling volume. The rods each form a capacitor with the thereminist's hands, and the capacitive sensing controls internal oscillators to produce sound [4]. The theremin is the only musical instrument played by *not* touching it; touching the antennae produces no sound.

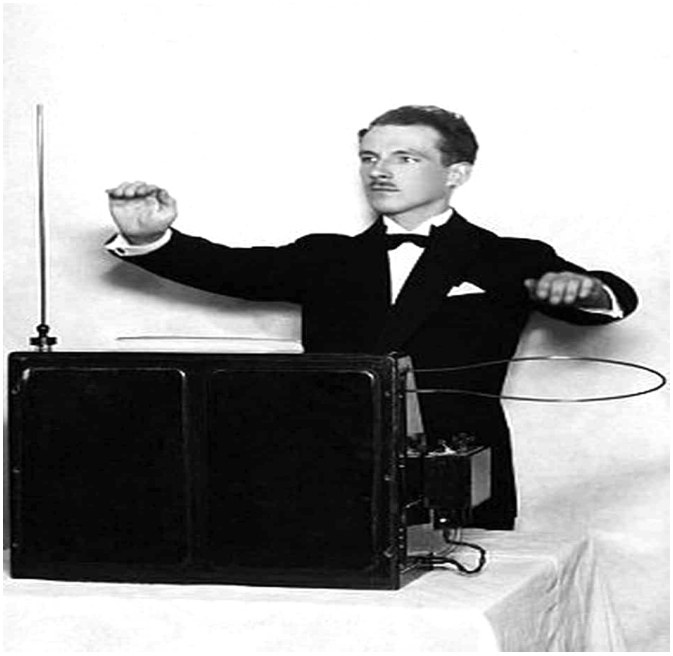

Fig. 1. Léon Theremin playing his newly invented instrument. Adapted from [2]

B. Objective

The aim of this project is to design another musical instrument played without physical contact: the LaunchTone. Instead of capacitive sensing, hand position will be detected using ultrasonic sensor data and mapped to a range of frequencies via software. The heart of this device is the Texas Instruments MSP432P401R LaunchPad microcontroller board, which will be used to drive the sensors, speakers, status LEDs, and other peripherals. The LaunchTone should be capable of the following:

- Playing musical notes which have pitch and volume proportional to the distance between ultrasonic sensors and the user's hands.
- Updating the pitch of the notes frequently enough to produce a recognizable melody.
- Providing information to the user about the ultrasonic sensor's status and the musical notes being played.
- Receiving user input that switches sound modes, both by changing the musical scales of the mapped range of notes and by switching sound profiles entirely.

II. Design

A. Plan

The musical instrument's design is relatively simple. Ultrasonic sensors measure the distance to any obstruction above them and report this distance to the LaunchPad. The LaunchPad maps the distance of one to a musical tone with a specific frequency and period and the other to a volume level, as long as the distance measurements are valid, i.e. not too close or too far. That information is displayed on an LCD screen and through the activation of status LEDs. The tones are played over the speakers as long as the audio killswitch is in the active position, otherwise no sound is played. Pushbuttons are used to cycle between modes and musical scales. A high-level block diagram can be seen in fig. 2.

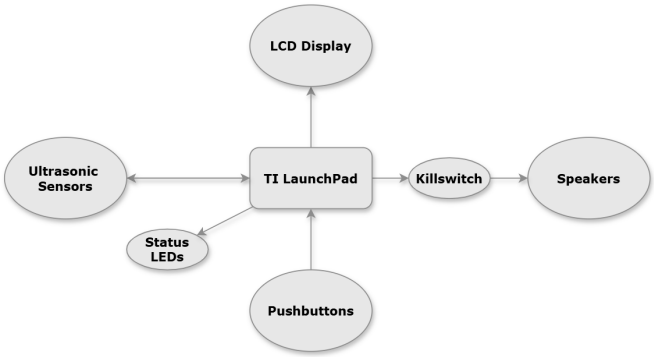

Fig. 2. High-level block diagram of the LaunchTone.

Unfortunately, due to time constraints and defective hardware, the project was not able to be fully implemented as designed. One of the ultrasonic sensors arrived defective and could not be replaced in time, and the LCD screen was not able to be configured before the project deadline. Additionally, the pushbutton array suffered an unknown problem and currently does not function.

B. LaunchPad Utilization

Several LaunchPad modules will be used to realize the LaunchTone. In order to improve performance and ensure that the musical tones can be played for as specific a duration as possible, periodic interrupts are used wherever possible. One TimerA module coordinates 10 microsecond ultrasonic sensor trigger pin pulses and echo pin timeouts, while another uses frequency modulation to generate tones on the speakers. An eUSCI module is used in UART mode to enable printing of data to the Code Composer Studio serial console.

The LaunchPad's GPIO ports are used extensively throughout the project. LEDs are connected to GPIO pins and controlled via software. The ultrasonic sensor trigger pins are held high for 10 microseconds every scanning cycle, and an interrupt is triggered on the negative edge of the signal to mark the time at which the ultrasonic sound wave emission has ended. Interrupts are enabled on the positive edge of the echo pin signals to indicate the time at which the sound wave has been received after reflecting off of the nearest object. These two times are used to compute the ultrasonic sensor distance readings, which is then mapped to a musical note.

Each musical note is expressed in software as a square wave with 50% duty cycle and a period in microseconds, which is set to the rollover time of the second TimerA module. The interrupts generated by that timer module toggle the signal on another GPIO pin, which is connected to the gate of an N-type MOSFET. The MOSFET's drain and source are connected to an external power source such as a battery, which allows for enough power to be delivered to the speakers while still providing the switching characteristics needed for high frequency notes.

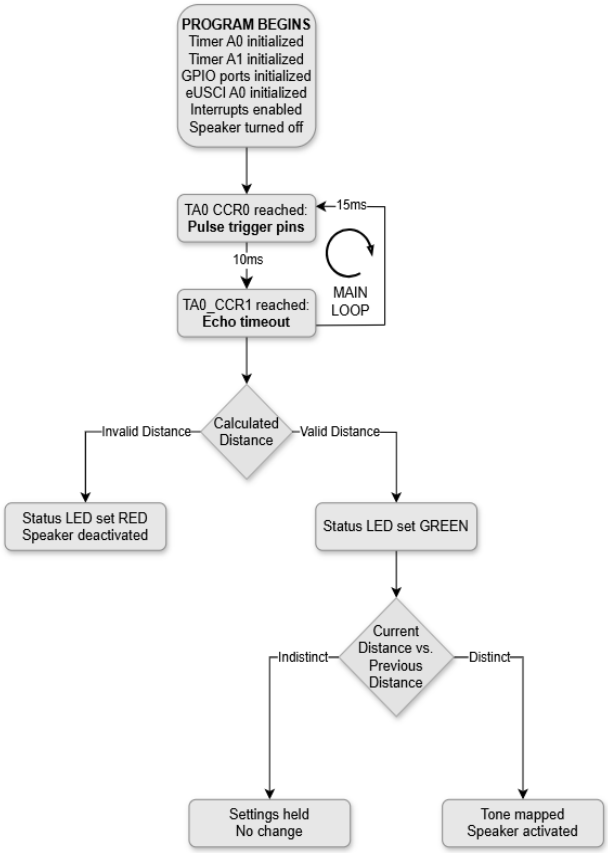

Fig. 3. Flowchart showing cyclic program execution.

III. Components

Components used in the project were sourced from the CSUN Robotics lab when possible and purchased online otherwise. Table 1 lists all components currently available for purchase. Additionally, the black faceplate used to interface with the switch, buttons, and sensors and to hold the speakers,

LEDs, and LCD display was fabricated using a Bambu Lab P1S 3D printer available through the CSUN ECE department.

TABLE I
TOUCHTONE COMPONENTS

Part Name	Quantity
TI MSP432P401R LaunchPad Development Kit	1
Diffused Red-Green LED - 18mm Round	2
US-100 Ultrasonic Sensor	2
3W 8Ω Speaker	2
Illuminated Toggle Switch	1
Normally Open Pushbutton - Red	4
IRL540N Power N-Type MOSFET	1
330 Ω Resistor	2
Breadboard	1

IV. Implementation

Several obstacles were encountered in the implementation of the LaunchTone, both with software and hardware. At first, an attempt was made to coordinate the timing of the ultrasonic sensor trigger and echo pins using the SysTick timer available on the LaunchPad. After designing and testing it, however, it was determined that the SysTick timer did not offer sufficiently comprehensive interrupt generation or resolution to fit the specifications of the US-100 sensors as they were used in the project. Instead, the TimerA module was used, which allowed for several capture-compare (CCR) registers each with their own interrupt handling, and a resolution of up to 8.3 nanoseconds. This was ultimately a much better choice.

The microcontroller was originally connected to a TI-RSLK robot chassis, as it was used as an educational development kit for students in the lab portion of the course. This chassis is remarkably feature-rich, including six bumper sensors, an 8-channel QTRX infrared sensor array, two motors with encoders and wheels, among other useful modules all connected via GPIO to the LaunchPad. However, as none of these were relevant to the musical instrument's design, the fact that so many GPIO pins were already in use grew very problematic. After using all available pins with the necessary interrupt generation capability, other pins were used regardless of their additional connections to the RSLK. For example, certain bumper sensors, when pressed, could erroneously activate the Red-Green LED lights.

Within a short time, though, those pins too were doubly in use and the only other free pins had significantly detrimental effects when overwritten. The motor pins were the most problematic; during one particularly late night of testing various GPIO pin combinations the musical instrument drove away when played. After this, the RSLK chassis was disconnected altogether and the GPIO connections were redone completely (and much more simply). Even after this redesign, the button inputs failed to register in software. At present it is unclear why the buttons did not work correctly, or even whether the problem is caused by hardware or software. A list of all pin connections can be seen in table 2.

TABLE II
COMPONENT-LAUNCHPAD PIN CONNECTIONS

Component Name	Component Pin	LaunchPad Pin
Ultrasonic Sensor 1	TRIG	P4.0
	ECHO	P4.1
	VDD	+5V
	GND	GND
	GND2	GND
Ultrasonic Sensor 2	TRIG	P4.2
	ECHO	P4.3
	VDD	+5V
	GND	GND
	GND2	GND
Killswitch	VDD	P4.5
	GND	GND
	TRAVELER[a]	RED WIRE[a]
Speakers	RED WIRE[a]	TRAVELER[a]
	BLK WIRE	GND
Red-Green LED 1	1	P10.0
	2	GND
	3	P10.1
Red-Green LED 2	GND	P10.2
	GND2	GND
	TRIG	P10.3
Pushbutton 1	1	P2.4
	2	+5V
Pushbutton 2	1	P2.5
	2	+5V
Pushbutton 3	1	P2.6
	2	+5V
Pushbutton 4	1	P2.7
	2	+5V

[a]Connections formed between killswitch and speakers directly.

Fig. 4. 3D-printed faceplate of the LaunchTone.

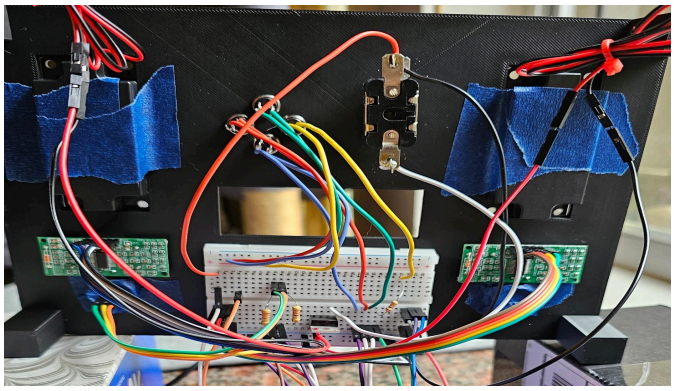

Fig. 5. Underside of faceplate showing wiring connections.

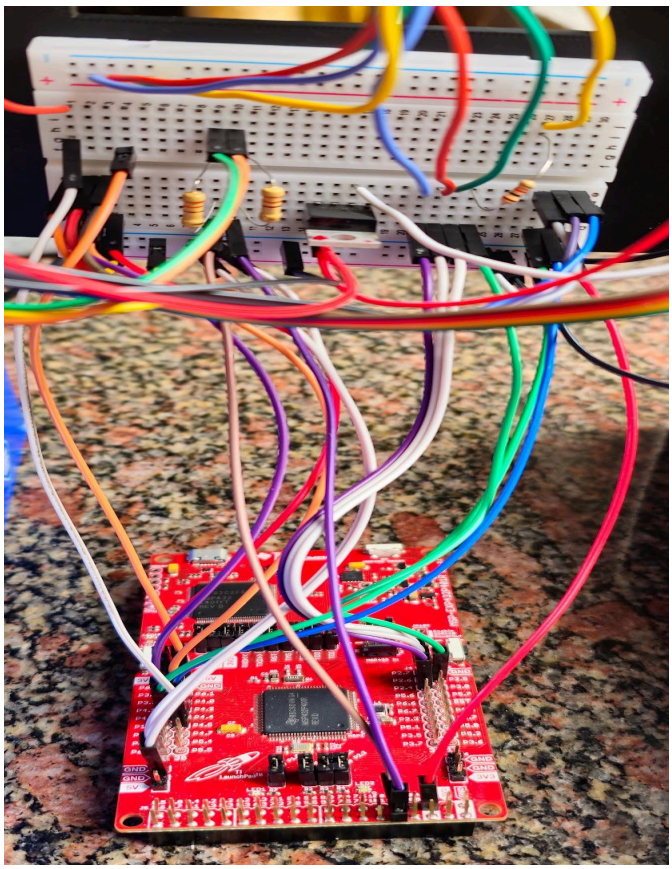

Fig. 6. Wiring connections between the faceplate breadboard and LaunchPad microcontroller.

V. Conclusion

A. Results

While the design requirements as outlined in section 1.B. of this report were not met, several key features were realized. The LaunchTone was able to play musical notes which had pitch proportional to the distance between an ultrasonic sensor and the user's hand. The sensor data was presented to the user via CCS terminal using UART protocol. The Red-Green status LED correctly activated when the ultrasonic sensor was detecting an object within the mapped range. A talented musician would certainly be able to explore various hand movements and positions to produce unique melodies and engage in creative expression using the LaunchTone instrument, even in its current unfinished state.

B. Further Development

There are many aspects of this project well suited to future development. Firstly, the full realization of the original design involves introducing an LCD screen, fixing the faulty ultrasonic sensor and buttons, and introducing multiple sound profiles via software. In the process of constructing the LaunchTone, though, many additional features that would enhance usability and improve musicality became clear. A filter algorithm should be applied to the ultrasonic sensor data to smooth transitions between sounds. The ultrasonic waves themselves have a wide beam angle, which can cause a single hand to appear in both sensor fields at once. It is worth exploring mechanical solutions to this problem, such as extending the metal barrel around the sensor emitter slightly. Additionally, the entire device should not just have a faceplate but an entire enclosure that contains the wiring and LaunchPad microcontroller. This could also be 3D-printed to minimize cost. There are nearly infinite possibilities to explore alternative sounds through well-written software. Drums, piano, guitar, and electronic dance music (EDM) sound profiles could all be emulated very accurately using the current instrument.

C. Student Learning Outcomes

Many student learning goals were achieved in the course of this project. Hands-on experience was gained in the design, application, and testing of embedded systems. Peripheral interfacing, both on the hardware and software side, is critical to understand and was used heavily in this project. C programming was used to perform a plethora of tasks using information gleaned from real-world datasheets and reference manuals. Overall, this project provided invaluable practice with many meaningful engineering principles and should be included in more engineering courses.

References

[1] Smithsonian National Museum of Natural History, "Musical Instruments," *Human Origins Program*, Oct. 22, 2021. [Online]. Available: https://humanorigins.si.edu/evidence/behavior/art-music/musical-instruments.

[2] The Linda Hall Library, "Leon Theremin," *The Linda Hall Library*, Aug. 2, 2022. [Online]. Available: www.lindahall.org/about/news/scientist-of-the-day/leon-theremin/. Accessed: May 12, 2025.

[3] R. Baumann, "The Theremin Turns 100," *Bibliolore*, Oct. 5, 2020. [Online]. Available: https://bibliolore.org/2020/10/05/the-theremin-turns-100/. Accessed: May 13, 2025.

[4] Pavek Museum, "The Theremin - How it Works," *Pavek Museum*. [Online]. Available: https://web.archive.org/web/20220417112911/http://www.pavekmuseum.org/theremin/theop.html. Accessed: May 13, 2025.

[5] Texas Instruments, “MSP432P4xx Technical Reference Manual,” Archive. Available: web.archive.org/web/20200402132841/http:/www.ti.com/lit/ug/slau356i/slau356i.pdf. Accessed: May 13, 2025.

[6] draw.io, “Flowchart Maker & Online Diagram Software”. https://app.diagrams.net/

main.c

```c
/**
 * Main program for ECE595R Final Project
 * Spring 2025, Professor Shahnam Mirzaei
 * Digital Theramin / Air Drums
 *
 * WIRING:
 *
 *      Ultrasonic Sensors:
 *          - VDD to 5V/3.3V, GND to GND, GND2 to GND
 *          - TRIG1 to P4.0, ECHO1 to P4.1
 *          - TRIG2 to P4.2, ECHO2 to P4.3=
 *
 *      Speakers:
 *          - Currently mono audio (both speakers driven by the same signal)
 *          - P4.5 output to killswitch to gate of power NMOS
 *
 *      LEDs:
 *          - Common Anode RG LED 1:
 *              - Red+ to P10.2, Green+ to P10.3, GND to GND
 *          - Common Anode RG LED 2:
 *              - Red+ to P10.0, Green+ to P10.1, GND to GND
 */

// IMPORTED FILES -------------------------------------------------------------

#include <stdint.h>                 // uintN_t types
#include <stdio.h>                  // Printf
#include "msp.h"                    // MSP432P401R library
#include "inc/Clock.h"              // Clock_Delay1us(), Clock_Delay1ms(),
#include "inc/CortexM.h"            // Enable_Interrupts()
#include "inc/EUSCI_A0_UART.h"      // Directs printf to serial console

// HANDWRITTEN FILES ----------------------------------------------------------

#include "inc/Timer_A0.h"           // Ultrasonic sensor timing
#include "inc/Timer_A1.h"           // Speaker signal timing/PWM
#include "inc/SysTick.h"            // CURRENTLY UNUSED
#include "inc/GPIO.h"               // All GPIO pins (sensors, speakers, LEDs)
#include "inc/Ultrasonic_Sensor.h"  // Ultrasonic sensor functionality
#include "inc/Speaker.h"            // Speaker functionality

// MAIN FUNCTION --------------------------------------------------------------

void main(void)
{
    Clock_Init48MHz();              // System clock
    EUSCI_A0_UART_Init_Printf();    // Serial console for printf statements
    Timer_A0_Init();                // Ultrasonic sensor timing
    Timer_A1_Init();                // Timer A1 speaker tone timing
    P2_Init();                      // Pushbuttons
    P10_Init();                     // RG LEDs
    P4_Init();                      // Ultrasonic sensor pins on P4.0 - P4.3
    P4_5_Init();                    // Speaker pin on P4.5

    EnableInterrupts();             // Enable ALL system interrupts

    Turn_Off_Speaker();


    while(1) {

        if (US_sensors_trig_ready)
        {
            Trigger_US_Sensors();
            US_sensors_trig_ready = 0;
        }
        if (US_sensors_echo_ready) {
            Calculate_US_Distances();
            US_sensors_echo_ready = 0;

            // DEBUG PRINTS
            //printf("T1: %6d, E1: %6d, T2: %6d, E2: %6d\n", US1_trig_end_us,
            //        US1_echo_start_us, US2_trig_end_us, US2_echo_start_us);
            printf("MAIN: US1 Distance: %5.1fcm, US2 Distance: %5.1fcm\n",
                   US1_distance_cm, US2_distance_cm);

            Update_Speakers();
        }
    }
}
```

Ultrasonic_Sensor.c

```c

#include "inc/Ultrasonic_Sensor.h"

// This function pulses the trigger pins of the ultrasonic sensors for 10us
// and resets all sensor timing measurements. The falling edge of the trigger
// pins triggers a GPIO interrupt which captures the end of the trigger signal
void Trigger_US_Sensors()
{
    // Reset all sensor timing measurements
    US1_trig_end_us   = 0;
    US1_echo_start_us = 0;
    US2_trig_end_us   = 0;
    US2_echo_start_us = 0;

    // Pulse both trigger signals (P4.0, P4.2)
    P4->OUT |= 0x05;
    Clock_Delay1us(10); // Minimum 10us from datasheet
    P4->OUT &= ~0x01;
    Clock_Delay1us(5);  // Ensure that falling edges occur at different times
    P4->OUT &= ~0x04;
}

// This function converts sensor timing measurements to distance measurements
// in accordance with the formula: Distance = V_sound * (t / 2)
void Calculate_US_Distances()
{
    // Calculate the distance measurements
    US1_distance_cm = SPEED_OF_SOUND *
            (float)(US1_echo_start_us - US1_trig_end_us) / 2 / 1000000 * 100;
    US2_distance_cm = SPEED_OF_SOUND *
            (float)(US2_echo_start_us - US2_trig_end_us) / 2 / 1000000 * 100;

    // Replace out-of-bounds values with sentinel value (-1) and set LEDs
    if ((US1_distance_cm > SPEAKER_UPPER_BOUND_CM) || (US1_distance_cm < 1))
    {
        US1_distance_cm = -1;
        Set_RG_LED1(RG_LED1_RED);
    }
    else { Set_RG_LED1(RG_LED1_GREEN); }

    if ((US2_distance_cm > SPEAKER_UPPER_BOUND_CM) || (US2_distance_cm < 1))
    {
        US2_distance_cm = -1;
        Set_RG_LED2(RG_LED2_RED);
    }
    else { Set_RG_LED2(RG_LED2_GREEN); }
}
```

Ultrasonic_Sensor.h

```c

#ifndef ULTRASONIC_SENSOR_H_
#define ULTRASONIC_SENSOR_H_

#include <stdint.h>
#include "msp.h"
#include "inc/Clock.h"
#include "inc/GPIO.h"
#include "inc/EUSCI_A0_UART.h"

#define SPEED_OF_SOUND 344
#define SPEAKER_UPPER_BOUND_CM 100
#define SPEAKER_LOWER_BOUND_CM 10

volatile uint8_t US_sensors_trig_ready; // "Can pulse triggers" flag
volatile uint8_t US_sensors_echo_ready; // "Can calculate distance" flag

volatile uint8_t  US1_active;           // "Is waiting for response" flag
volatile uint32_t US1_trig_end_us;      // Falling edge of US1 trigger time
volatile uint32_t US1_echo_start_us;    // Rising edge of US1 echo time
volatile float    US1_distance_cm;      // Calculated US1 distance reading

volatile uint8_t  US2_active;           // "Is waiting for response" flag
volatile uint32_t US2_trig_end_us;      // Falling edge of US2 trigger time
volatile uint32_t US2_echo_start_us;    // Rising edge of US2 echo time
volatile float    US2_distance_cm;      // Calculated US2 distance reading

void Trigger_US_Sensors(void);
void Calculate_US_Distances(void);

#endif /* ULTRASONIC_SENSOR_H_ */
```

Speaker.c

```c

#include "inc/Speaker.h"

uint16_t current_tone = 0;
uint16_t previous_tone = 1;

// GPT-GENERATED PERIODS
uint16_t note_periods_us[18] = {
    3822, // C4:   261.63 Hz
    3608, // C#4:  277.18 Hz
    3405, // D4:   293.66 Hz
    3214, // D#4:  311.13 Hz
    3034, // E4:   329.63 Hz
    2863, // F4:   349.23 Hz
    2703, // F#4:  369.99 Hz
    2551, // G4:   392.00 Hz
    2408, // G#4:  415.30 Hz
    2273, // A4:   440.00 Hz
    2145, // A#4:  466.16 Hz
    2025, // B4:   493.88 Hz
    1911, // C5:   523.25 Hz
    1804, // C#5:  554.37 Hz
    1703, // D5:   587.33 Hz
    1607, // D#5:  622.25 Hz
    1517, // E5:   659.25 Hz
    1432  // F5:   698.46 Hz
};

void Update_Speakers()
{
    // Sets tone of speakers once per recieved distance measurement
    if (US2_distance_cm == -1 && speaker_is_on) { Turn_Off_Speaker(); }

    else if (US2_distance_cm > SPEAKER_LOWER_BOUND_CM &&
             US2_distance_cm < SPEAKER_UPPER_BOUND_CM)
    {
        current_tone = Map_Distance_To_Tone(US2_distance_cm);

        if (current_tone != previous_tone || !speaker_is_on)
        {
            Play_Tone(current_tone);
            previous_tone = current_tone;
        }
    }
}

uint16_t Map_Distance_To_Tone(float distance)
{
    int bin = (int)distance;
    bin = bin / 10;
    return note_periods_us[bin * 2];
}

void Play_Tone(uint16_t note_period_us)
{
    Update_Timer_A1_Period(note_period_us);
    speaker_is_on = 1;
}

void Turn_Off_Speaker()
{
    TIMER_A1->CTL &= ~0x0030;
    speaker_is_on = 0;
}
```

Speaker.h

```h

#ifndef SPEAKER_H_
#define SPEAKER_H_

#include <stdint.h>
#include "msp.h"            // MSP432P401R library
#include "inc/Clock.h"      // Clock_Delay1ms()
#include "inc/Timer_A1.h"   // Update_Timer_A1_Period()
#include "inc/Ultrasonic_Sensor.h"

uint8_t speaker_is_on;

uint16_t Map_Distance_To_Tone(float distance);
void Update_Speakers(void);
void Play_Tone(uint16_t note_period_us);
void Turn_Off_Speaker(void);

#endif
```

GPIO.c

```c

#include "inc/GPIO.h"

// PUSHBUTTONS ----------------------------------------------------------------

// Initializes pushbuttons on P2.4 through P2.7
void P2_Init()
{
    P2->DIR  &= ~0xF0;  // Set pushbutton pins to input
    P2->SEL1 &= ~0xF0;  // Default function select
    P2->SEL0 &= ~0xF0;  // ^^^
    P2->IES  &= ~0xF0;  // P4.0, P4.2 flags set with low-to-high transition
    P2->IE   |=  0xF0;  // Interrupts enabled on P2.4 through P2.7

    NVIC->ISER[1] |= 0x00000010;  // Enable interrupts (#36 == 32 + 4 == bit 4)
}

// Pushbutton interrupt handler (*** TEST PHASE ***)
void PORT2_IRQHandler()
{
    uint16_t pb_status = P2->IV;
    switch (pb_status) {
        case 0x02: // P2.0 IFG
            printf("PB0 pressed!\n");
            break;
        case 0x04: // P2.1 IFG
            printf("PB1 pressed!\n");
            break;
        case 0x06: // P2.2 IFG
            printf("PB2 pressed!\n");
            break;
        case 0x08: // P2.3 IFG
            printf("PB3 pressed!\n");
            break;
        default: break;
    }
}

// ULTRASONIC SENSORS ---------------------------------------------------------

// Initializes ultrasonic sensors 1 and 2 on P4.0/P4.1 and P4.2/P4.3
void P4_Init()
{
    P4->DIR  |=  0x05;  // Set trigger pins (P4.0, P4.2) to output
    P4->DIR  &= ~0x0A;  // Set echo pins (P4.1, P4.3) to input
    P4->SEL1 &= ~0x0F;  // Default function select
    P4->SEL0 &= ~0x0F;  // ^^^
    P4->IES  |=  0x05;  // Set trigger IFGs on high-to-low transition
    P4->IES  &= ~0x0A;  // Set echo IFGs on low-to-high transition
    P4->IE   |=  0x0F;  // Interrupts enabled on P4.0 through P4.3

    NVIC->ISER[1] |= 0x00000040;  // Enable interrupts (#38 == 32 + 6 == bit 6)
}

// Ultrasonic trigger/echo pin interrupt handler
void PORT4_IRQHandler()
{
    uint16_t iv = P4->IV; // Find highest pending interrupt and clear its flag
    switch (iv) {
        case 0x02: // P4.0 IFG (US1 TRIG)
            US1_trig_end_us = TIMER_A0->R;
            US1_active = 1;
            break;
        case 0x04: // P4.1 IFG (US1 ECHO)
            if (US1_active) {
                US1_echo_start_us = TIMER_A0->R;
                US1_active = 0;
            }
            break;
        case 0x06: // P4.2 IFG (US2 TRIG)
            US2_trig_end_us = TIMER_A0->R;
            US2_active = 1;
            break;
        case 0x08: // P4.3 IFG (US2 ECHO)
            if (US2_active) {
                US2_echo_start_us = TIMER_A0->R;
                US2_active = 0;
            }
            break;
        default: break;
    }
}
```

GPIO.c (continued)

```c
// SPEAKERS ------------------------------------------------------------------

// Initializes speaker output on P4.5
void P4_5_Init()
{
    P4->DIR  |=  0x20;  // Set speaker pin (P4.5) to output
    P4->SEL1 &= ~0x20;  // Default function select
    P4->SEL0 &= ~0x20;  // ^^^
    P4->OUT  &= ~0x20;  // Clear output of speaker pin
}

// RG STATUS LEDS ------------------------------------------------------------

// Initializes RG LED pins on P10.0 through P10.3
void P10_Init()
{
    P10->SEL0 &= ~0x0F;
    P10->SEL1 &= ~0x0F;
    P10->DIR  |=  0x0F;
    P10->OUT  &= ~0x0F;
}

// Sets the color displayed by the Red-Green LED above US1
void Set_RG_LED1(uint8_t color)
{
    switch (color)
    {
    case RG_LED1_OFF:
        P10->OUT &= ~(RG_LED1_RED + RG_LED1_GREEN);
        break;
    case RG_LED1_GREEN:
        P10->OUT &= ~RG_LED1_RED;
        P10->OUT |= RG_LED1_GREEN;
        break;
    case RG_LED1_RED:
        P10->OUT &= ~RG_LED1_GREEN;
        P10->OUT |= RG_LED1_RED;
        break;
    default: break;
    }
}

// Sets the color displayed by the Red-Green LED above US2
void Set_RG_LED2(uint8_t color)
{
    switch (color)
    {
    case RG_LED2_OFF:
        P10->OUT &= ~(RG_LED2_RED + RG_LED2_GREEN);
        break;
    case RG_LED2_GREEN:
        P10->OUT &= ~RG_LED2_RED;
        P10->OUT |= RG_LED2_GREEN;
        break;
    case RG_LED2_RED:
        P10->OUT &= ~RG_LED2_GREEN;
        P10->OUT |= RG_LED2_RED;
        break;
    default: break;
    }
}
```

GPIO.h

```c

#ifndef GPIO_H_
#define GPIO_H_

#include <stdint.h>
#include <stdio.h>
#include "msp.h"        // MSP432P401R library
#include "Clock.h"

#define RG_LED1_RED      0x02
#define RG_LED1_GREEN    0x01
#define RG_LED1_OFF     ~0x03
#define RG_LED2_RED      0x08
#define RG_LED2_GREEN    0x04
#define RG_LED2_OFF     ~0x0C

extern volatile uint8_t US1_active;
extern volatile uint8_t US2_active;
extern volatile uint32_t US1_echo_start_us; // Falling edge of US1 trigger time
extern volatile uint32_t US1_trig_end_us;   // Rising edge of US2 echo time
extern volatile uint32_t US2_trig_end_us;   // Falling edge of US2 trigger time
extern volatile uint32_t US2_echo_start_us; // Rising edge of US2 echo time

void P2_Init(void);
void P10_Init(void);
void P4_Init(void);
void P4_5_Init(void);
void Port4_IRQHandler(void);
void Set_RG_LED1(uint8_t color);
void Set_RG_LED2(uint8_t color);

#endif /* GPIO_H_ */
```

Timer_A0.c

```c

#include "inc/TIMER_A0.h"

void Timer_A0_Init()
{
    TIMER_A0->CTL     &= ~0x0030;   // Stop Timer A0
    TIMER_A0->CTL     |=  0x02C0;   // SMCLK (f = 24MHz), prescale by 8
    TIMER_A0->EX0     |=  0x0002;   // Divide input by 3, f = 1MHz (T = 1us)
    TIMER_A0->CCTL[0] |=  0x0010;   // Enable interrupts on timer rollover
    TIMER_A0->CCTL[1] |=  0x0010;   // Enable interrupts on cutoff register

    TIMER_A0->CCR[0] = SAMPLE_PERIOD_MS * 1000 - 1; // Rollover period (in us)
    TIMER_A0->CCR[1] = CUTOFF_TIME_MS   * 1000 - 1; // ECHO timeout cutoff time

    NVIC->ISER[0] |= 0x00000100;    // Enable TA0 CCR0 interrupt (interrupt 8)
    NVIC->ISER[0] |= 0x00000200;    // Enable TA0 CCRn interrupt (interrupt 9)

    TIMER_A0->CTL |= 0x0014;        // Clear and enable Timer A0 in up mode
}

// Indicates timer rollover
void TA0_0_IRQHandler()
{
    TIMER_A0->CCTL[0] &= ~0x0001;   // Acknowledge interrupt (clear IFG bit)
    US_sensors_trig_ready = 1; // Set flag to trigger US sensors in main loop
}

// Indicates Ultrasonic ECHO timeout
void TA0_N_IRQHandler()
{
    // Determine the source of the interrupt
    if (TIMER_A0->CCTL[1] & 0x0001)
    {
        TIMER_A0->CCTL[1] &= ~0x0001; // Acknowledge interrupt (clear IFG bit)

        US_sensors_echo_ready = 1;  // Set flag to read US sensors in main loop

        // Ensure that the sensors are not awaiting response
        if (US1_active) { US1_active = 0; }
        if (US2_active) { US2_active = 0; }
    }
}
```

Timer_A0.h

```c

#ifndef TIMER_A0_H_
#define TIMER_A0_H_

#include <stdint.h>
#include "msp.h"        // MSP432P401R library

#define SAMPLE_PERIOD_MS 25
#define CUTOFF_TIME_MS 10

extern volatile uint8_t US_sensors_trig_ready;
extern volatile uint8_t US_sensors_echo_ready;
extern volatile uint8_t US1_active;
extern volatile uint8_t US2_active;

void Timer_A0_Init(void);
void TA0_0_IRQHandler(void);
void TA0_N_IRQHandler(void);

#endif /* TIMER_A0_H_ */
```

Timer_A1.c

```c

#include "inc/Timer_A1.h"

void Timer_A1_Init()
{
    TIMER_A1->CTL     &= ~0x0030; // Stop Timer A1
    TIMER_A1->CTL     |=  0x02C0; // Source to SMCLK (f = 24MHz), prescale by 8
    TIMER_A1->EX0     |=  0x0002; // Divide input by 3 again, f = 1MHz (T=1us)
    TIMER_A1->CCTL[0] |=  0x0010; // Enable interrupts on timer rollover
    TIMER_A1->CCTL[1] |=  0x0010; // Enable interrupts on cutoff register
    TIMER_A1->CCR[0]  = 3822 - 1; // Initialize to C4 note (261.63Hz or so)
    TIMER_A1->CCR[1]  = 1911 - 1; // 50% Duty Cycle

    NVIC->ISER[0] |= 0x00000400;  // Enable TA1 CCR0 interrupt (interrupt 10)
    NVIC->ISER[0] |= 0x00000800;  // Enable TA1 CCRn interrupt (interrupt 11)

    TIMER_A1->CTL |= 0x0014;      // Clear and enable Timer A1 in up mode
}

// Toggles the speaker pin on timer rollover (CCR0 value)
void TA1_0_IRQHandler()
{
    TIMER_A1->CCTL[0] &= ~0x0001;   // Acknowledge interrupt and clear IFG flag
    P4->OUT ^= 0x20;                // Toggle P4.5
}

// Toggles the speaker pin on CCR1 value
void TA1_N_IRQHandler()
{
    if (TIMER_A1->CCTL[1] & 0x0001)
    {
        TIMER_A1->CCTL[1] &= ~0x0001;   // Acknowledge interrupt and clear IFG
        P4->OUT ^= 0x20;                // Toggle P4.5
    }
}

// Shuts down Timer_A1, modifies the period using the period_us argument, sets
// pulse width to half the new period, and restarts Timer_A1
void Update_Timer_A1_Period(uint16_t period_us)
{
    TIMER_A1->CTL &= ~0x0030;               // Shut down Timer A1
    TIMER_A1->CCR[0] = period_us - 1;       // Set new period
    TIMER_A1->CCR[1] = (period_us / 2) - 1; // Set 50% duty cycle
    TIMER_A1->CTL |= 0x0014;                // Clear and restart timer
}

// Shuts down Timer_A1, modifies the pulse width using the pulse_width_us
// argument, and restarts Timer_A1
void Update_Timer_A1_Pulse_Width(uint16_t pulse_width_us)
{
    TIMER_A1->CTL &= ~0x0030;               // Shut down Timer A1
    TIMER_A1->CCR[1] = pulse_width_us - 1;  // Set new pulse width
    TIMER_A1->CTL |= 0x0014;                // Clear and restart timer
}
```

Timer_A1.h

```h

#ifndef INC_TIMER_A1_H_
#define INC_TIMER_A1_H_

#include "msp.h"
#include <stdio.h>
#include <stdint.h>

uint16_t period_us;
uint16_t pulse_width_us;

void Timer_A1_Init(void);
void TA1_0_IRQHandler(void);
void TA1_N_IRQHandler(void);
void Update_Timer_A1_Period(uint16_t period_us);
void Update_Timer_A1_Pulse_Width(uint16_t pulse_width_us);

#endif /* INC_TIMER_A1_H_ */
```