Team Name:

The C-Team

Team Members:

Dow Cox, Bernie Friesel, Austin DuCrest, Nolan Magee, Cole Wilson

Class Number and Name:

ECE 3130-001 Microcomputer Systems

Submission Date:
4-30-2025

Table of Contents:

Introduction

We implemented a soundboard, i.e. a synthesizer. This is primarily accomplished by using the on-board buzzer.

We assigned the buttons of the keypad and the switch buttons to an associated frequency. Each frequency has an associated musical note it makes the sound of, and we use those sounds to make songs.

The user can play octaves 1 through 6 for musical notes C, D, E, F, G, A, B. The user switches through the octaves by using the push button SW2. We also included external LEDs that will light up indicating which octave is currently being played.

The LCD displays the current octaves (3&4, 1&2, or 5&6). Whenever you press a button, the LCD also displays the current note being played in addition to the frequency of that note, e.g. C4 266 Hz.

The board has external buzzers to allow for multiple buttons to be played at once. This allows the user to make more unique notes like an actual keyboard / piano.

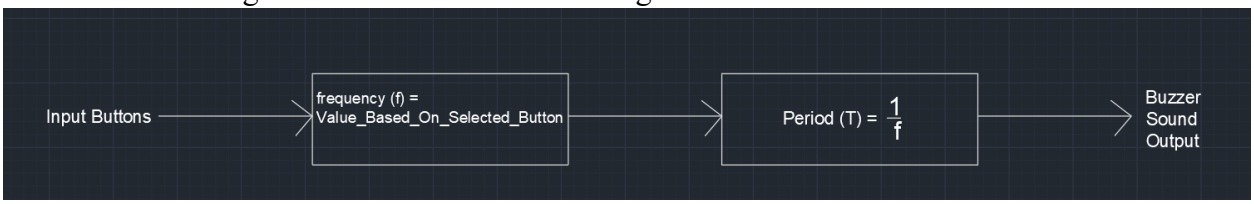
Our LCD was configured in such a way that we had a button (SW5) specifically for the use of toggling modes.

We had 2 modes: Mode 0 and Mode 1. Each mode change would update the LCD with new options for each "Page". Mode 0 is our main mode where most of our functionality is accomplished.

- **Mode 0:** SW2 allows us to switch between octaves for user note playing. SW3 allows us to toggle recording. SW3 is programmed to start or stop recording. SW4 programmed to handle our toggle repeated playback option allowing you to play the sound back to you. So, you would press SW3, play the sounds you desire, then press SW3 again, then press SW4 to play the sounds until the recording is over or until you press SW4 again.
- **Mode 1:** This mode simply toggles the playback mode, with 2 options toggled using SW2.
    - Play from enable (PFE): playbacks are based on whenever the user enables it, timing it right can be more complicated
    - Play from Loop (PFL): playback is based on loops so user can start playbacks during the middle of another one, timing is simpler

Specifications and Description

General Block Diagram of what the board is doing:



Whenever we press a certain input button using a push button, it tells the program to deliver a certain frequency to the buzzer, which is accomplished by converting the frequency to period and inputting that value to the code controlling the buzzer.

We got the general values for the frequencies from an online chart and an audio engineering textbook, and we adjusted these frequencies as needed to get the right sounds from our buzzer. We went up to octave 6 (C7 is part of octave 6).

[1] From muted.io

	0	1	2	3	4	5	6	7	8
C	16.35	32.7	65.41	130.81	261.63	523.25	1046.5	2093	4186
C#	17.32	34.65	69.3	138.59	277.18	554.37	1108.73	2217.46	4434.92
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.63
D#	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489	4978
E	20.6	41.2	82.41	164.81	329.63	659.25	1318.51	2637	5274
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F#	23.12	46.25	92.5	185	369.99	739.99	1479.98	2959.96	5919.91
G	24.5	49	98	196	392	783.99	1567.98	3135.96	6271.93
G#	25.96	51.91	103.83	207.65	415.3	830.61	1661.22	3322.44	6644.88
A	27.5	55	110	220	440	880	1760	3520	7040
A#	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951	7902.13

the values are in Hertz (Hz), the top row represents the octave (from 0 to 8)

Detailed Implementation

Interface Design

The LCD provides a human machine interface (HMI) to tell the user what the keypad and switches are doing.

The keypad is a bunch of pushbuttons, and these are primarily used by the user to generate musical notes/sounds as if they were playing an instrument.

The switches are used to change the boards operating modes.

The external LEDs implemented on the breadboard provide visual feedback to the user on what octave is being played.

The external buzzers implemented on the breadboard allow the user to play multiple notes at once.

Microcontroller Resource Utilization

We ran into some polling issues while programming the buzzer. To fix this, we utilized the microcontroller's pulse width modulation (PWM). The PWM allows us to asynchronously switch the buzzers output sounds without having to use delays. The PWM is essentially one. Delays were not preferable since our buzzer and music relied on exact timings.

For example, in our buzzer.c:

```
void SetFrequency(double freq, uint8_t timer)
{
    int freqInt = (int)freq;
    if (currentFrequency[timer] == freqInt)
        return;
    currentFrequency[timer] = freqInt;
    if (freq <= 0)
    {
        // Stop the timer if frequency is 0
        HAL_TIM_PWM_Stop(&htimEXT[timer], timers[timer].TIM_CHANNEL);
        return;
    }
```

Software

We implemented the code using Keil uVision and VS Code coding environments and programmed primarily in C-language. We used Keil whenever downloading the program to the board. We utilized GitHub to share our main program and other files online with each other. The code was quite lengthy with lots of initializations for the keypad and such, and you can find the code in our GitHub:

GitHub: https://github.com/dwc42/ece3130_project

Our code contains a mixture of .c files and .h files. The header files are primarily used for function prototypes, while C files are used for function definitions. Our main file with our int main() is in run.c. Buzzer.c, date.c, events.c, lcd.c, list.c are where additional function definitions are located.

Our software correlated to our desired musical outputs detailed here:

Excel Spreadsheet of specifically what musical note each input button generates the Buzzer Sound Outputs:

SW2 Adjusts the Octaves Used By cycling through a matrix in our code		(Note: C3 and C5 repeat because we are using octaves)						Note: keypad button A (K3) remains unused with dummy value "0" in matrix								
Row 1	Input Buttons (Keypad)	*	7	4	1	0	8	5	2	#	9	6	3	D	C	B
	Frequency	131	147	165	175	196	220	247	262	292	330	349	392	440	494	523
	Buzzer Sound Output (Musical Notes	C3	D3	E3	F3	G3	A3	B3	C4	D4	E4	F4	G4	A4	B4	C5
Row 2	Input Buttons (Keypad)	*	7	4	1	0	8	5	2	#	9	6	3	D	C	B
	Frequency	33	37	41	44	49	55	62	65	73	82	87	98	110	123	131
	Buzzer Sound Output (Musical Notes	C1	D1	E1	F1	G1	A1	B1	C2	D2	E2	F2	G2	A2	B2	C3
Row 3	Input Buttons (Keypad)	*	7	4	1	0	8	5	2	#	9	6	3	D	C	B
	Frequency	523	587	659	698	784	880	988	1047	1175	1319	1397	1568	1760	1975	2093
	Buzzer Sound Output (Musical Notes	C5	D5	E5	F5	G5	A5	B5	C6	D6	E6	F6	G6	A6	B6	C7

Note: these are color coded to the LED colors we used for the hardware
Note: an octave is 8 notes, the underlined notes indicate they are used for another octave

These interface directly with frequency matrix in run.c:

```
int Frequencies[3][16] =
    {
        {175, 262, 392, 0, 165, 247, 349, 523, 147, 220, 330, 494, 131, 196, 292, 440},
        {44, 65, 98, 0, 41, 62, 87, 131, 37, 55, 82, 123, 33, 49, 73, 110},
        {698, 1047, 1568, 0, 659, 988, 1397, 2093, 587, 880, 1319, 1975, 523, 784, 1175, 1760}};
```

The frequency matrix goes in order of K0 to K15 for each row, with the row being selected by SW2. SW2 accomplishes this by changing which part of the matrix the keypad starts reading:

```
void switchPressCallback(enum SWITCHS key)
{

    switch (key)
    {

    case BUTTON_SWITCH2:
    {

        presetIndex = (presetIndex + 1) % 3;
    }
    break;
```

K3, that is: button A, holds a dummy value of 0 since it should not be used or pressed by the user. This is because we can play 2 octaves on the keypad at once, with each octave having 8 notes, and our 2 octaves share a note. E.g. Octaves 1 and 2 share C2 as the highest and lowest notes respectively. So we only use 15 out of the 16 keys of the keypad.

Implementing these frequencies to be displayed on the LCD was fairly simple. Using the Excel spreadsheet and the Frequency matrix we created a struct to hold the frequency, octave, and note.

Sample of struct:

```
struct NoteProperties
    {
        uint16_t frequencies;
        char octaves;
        char note;

    };
 struct NoteProperties newFrequencies[3][16] =
    {
        { {175,'3','F',},{262,'4','C'},{392,'4','G'},{165,'3','E'},
        { {44,'1','F'},{65,'2','C'},{98,'2','G'},{41,'1','E'},{62,'
        { {698,'5','F'},{1047,'6','C'},{1568,'6','G'},{659,'5','E'}
    };
```

:

After we had this, displaying these to the keypad was straightforward using Write_Char_LCD and Write_String_LCD. Displaying the frequency value was bit more complicated since it was an integer instead of a char or string. The Display Number function converts the integer to a string that can be written on the LCD.

Run.c also contains how the program cycles through different modes of operation which adjust the playback mode.

Buzzer.c contains timers and initialization code for our board's 4 buzzers:

```
struct timer
{
    GPIO_TypeDef *GPIO;
    uint16_t GPIO_PIN;
    uint32_t GPIO_AF_TIM;
    uint64_t TIM_OCMODE_PWM;
    uint32_t TIM_CHANNEL;
    TIM_TypeDef *TIM;
};
struct timer timers[4] = {
    {GPIOC, GPIO_PIN_9, GPIO_AF2_TIM3, TIM_OCMODE_PWM1, TIM_CHANNEL_4, TIM3},
    {GPIOA, GPIO_PIN_0, GPIO_AF1_TIM2, TIM_OCMODE_PWM1, TIM_CHANNEL_1, TIM2},
    {GPIOB, GPIO_PIN_6, GPIO_AF2_TIM4, TIM_OCMODE_PWM1, TIM_CHANNEL_1, TIM4},
    {GPIOA, GPIO_PIN_1, GPIO_AF2_TIM5, TIM_OCMODE_PWM1, TIM_CHANNEL_2, TIM5}};
```

Note how the timer struct array corresponds to the pins utilized for each of the 4 buzzers. Basically, how the system knows which buzzer to play a sound at a specified frequency we call AddFrequency() with an end date of 0 which pushes to the end of an array of plays which are structs that contain a frequency and an end date. Then through polling the system takes the first 4 elements and plays those sounds on the respective buzzers.

We decided to use timers for the buzzers and adjusted the calculation of the period accordingly:

```
uint32_t timerClock = 72000000;
uint32_t prescaler = 72 - 1;
uint32_t period = timerClock / ((freq + 1) * (prescaler + 1));

// Update the timer configuration
htimEXT[timer].Init.Prescaler = prescaler;
htimEXT[timer].Init.Period = period;
```

Date.c keeps track of the number of milliseconds the board has been on. This is important for timing and is what makes the whole board work.

Events.c makes the buttons work through an event handler system. This handles how the buttons are pressed and handles what happens if multiple buttons are pressed.

Lcd.c contains the code to make the LCD work. This is primarily in the form of some of the given code from the lecture slides like Write_SR_LCD and LCD_nibble_write.

List.c handles the arrays keeping track of how long buttons have been pressed. This is useful for recording playback.

Hardware

We used a STM32-NUCLEO-L476RG microcontroller connected to an EDUBASE-L452 1173-H2. This board was provided to us for the purposes of the class/lab.

We utilized 6 external LEDs on the breadboard. The specific pin connections we utilized are detailed here:

External LEDs							
	Breadboard Position	1	8	13	19	24	29
	Selected Octave	1	2	3	4	5	6
	Pin Block	CN7	CN7	CN7	CN7	CN7	CN7
	Pin Connection	PC_3	PC_2	PC_11	PC_10	PA_14	PC_0

Note: these octaves match those in the specifications and description (p. 4).

We utilized 3 external buzzers in addition to the one on the breadboard (the default one on the board is connected to PC_9). The specific pin connections are as follows:

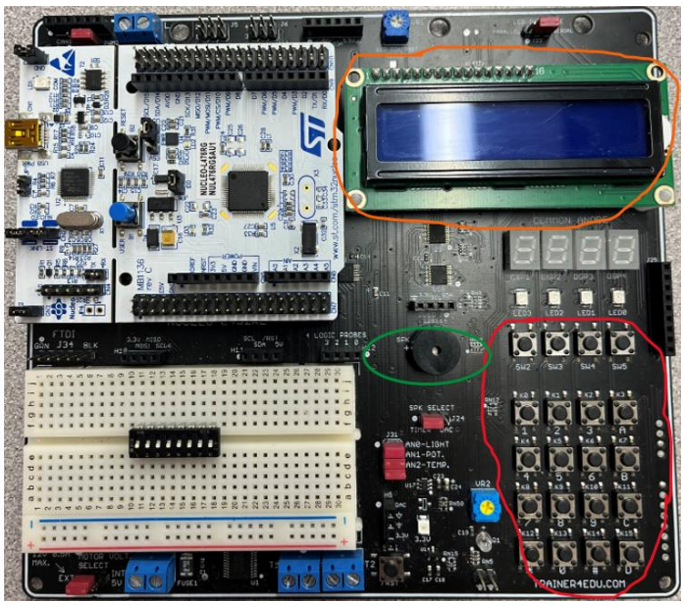
External Buzzers					
	Buzzer Number	1	2	3	4
	Breadboard Number	4	10	16	23
	Pin Block	CN10	CN8	CN9	CN8
	Pin Connection	PC_9	PA_1	PB_6	PA_0

Full view of board without external hardware:

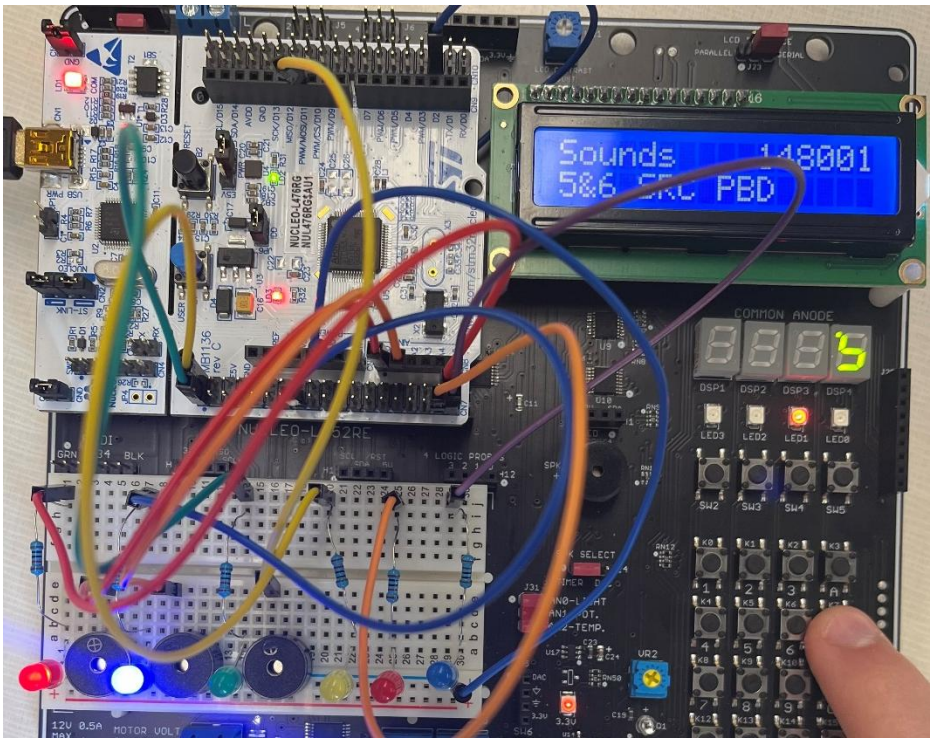
LCD – Circled in Orange

Buzzer – Circled in Green

Keypad and Switches (Pushbuttons) – Circled in Red



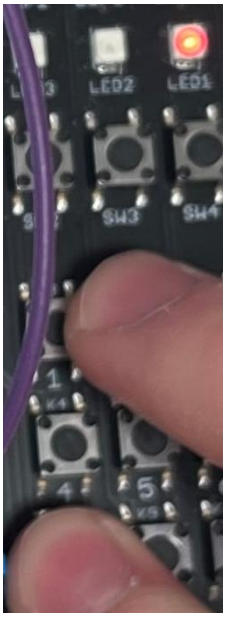
Full view of board with external hardware:



The external LEDs light up anytime the associated octave is being played, as demonstrated in this picture. These LEDs are paired with 220 Ω resistors to avoid blowing up the diode. These are connected to the microcontroller using wires. We utilized the manufacturer's website [2] to know how to interface these connections with our code, as detailed above.

The on-board LED1 triggers whenever you press multiple buttons at once on different columns.

This gives the user visual feedback.



Analysis

Testing

We tested the functionality of the input buttons (keypad) by pressing each button and checking if the buzzer made a sound. This was easy to test, since if we hit the button and it made a noise that was a good thing.

Then our musical experts Bernie and Austin tested the actual tones / musical notes being played. They made sure the buzzer was making the right noise and if we needed to adjust anything for a better output.

Bernie did a lot of troubleshooting with the recording functionality on the board to identify any issues with recording overlaps, issues with playing multiple buttons in the same row, and so on.

We ran into an error whenever we pressed multiple buttons at once on the same column the buzzer got confused. We decided to rotate the frequency matrix so that we could play multiple sounds on the column instead of the same row. This is not a perfect fix, but it allows us to play musical major intervals.

Testing the recording functionality was about as straightforward as testing the other musical notes. We just had to listen to the board and make sure it played the right sounds in the right order. Our initial and now default mode of recording PFE can be a bit finicky whenever trying to create songs from recorded playbacks, so we implemented the PFL mode as well.

Public Safety

This system utilizes a low voltage board without much physical danger to the user in terms of being electrocuted or shocked.

However, frequencies under 20 Hz and above 20 kHz are beyond the human range of hearing and can be dangerous to listen to. Accordingly, we avoided inputting any frequencies that exceed the human range of hearing. E.g. the musical note C0 has a frequency of 16.35 Hz so we avoided programming that note. However, we did have to implement a dummy value of 0 Hz for input button A (K3). This key should not be audible and should not be played.

It should also be noted that this board contains solder, which often contains lead which is carcinogenic. Anytime you handle this board you should wash your hands afterward.

Global/Cultural/Economic Factors

Music remains one of the great innovations of mankind, and every culture around the world has some form of music. Music has become synonymous with Tennessee through Nashville and great country music stars, and people have made great economic profit in the music industry.

This is made possible by innovative technologies like electric guitars and keyboards. Our board is a relatively simple implementation of a soundboard but serves as a good demonstration of what can be accomplished by engineers in a short amount of time.

The educational board we used cost about $125 [4] for educational use, and the microcontroller we used cost about $25 [3]. The external LEDs and buzzers we implemented only cost about $10 each for packs of these, so the overall cost was about $170.

This overall cost is relatively inexpensive, and you may be able to implement it cheaper if you use only the necessary components (buttons, buzzers, LEDs, resistors, LCD, microcontroller). But for our purposes the trainer board and microcontroller were provided for free, so they were the best options for us.

Teamwork Experience

Collaboration and Inclusivity

We maintained a co-learning environment where if one of us did not understand something the others would explain why or how we needed to do something. This allowed us to work efficiently with each other while everyone still learned something.

Dow has the most experience with coding and he took the lead on the project with writing most of the software, and took on a managerial role with the other team to help them implement some of the other features.

Austin and Bernie have the most music experience. Austin knows how to play the piano and Bernie is an audio engineer, so they contributed a lot to knowing the different music notes and how to accomplish the makings of the songs. Bernie did a lot of troubleshooting with getting the right frequencies in the matrix, and Austin helped troubleshoot the LCD displaying those frequencies.

Nolan worked primarily on implementing external LEDs on the breadboard to give the user feedback on what octave is currently being played. He implemented the necessary resistors to keep the LEDs from burning and interfaced the LEDs with the associated code.

Cole primarily worked on writing the project report to communicate the work done by the team and helped with coding as needed. Cole made the structure for the report and made most of the rough draft. Nolan edited the report and implemented the PowerPoint presentation that showcases the best features of our project and includes videos of our project working.

Leadership, Goals, and Planning

Our team shares many classes together, so we brainstormed ideas among ourselves for about a week until having a meeting to finalize our plan and get an idea of where we needed to start and what we wanted to work on.

We implemented weekly group meetings that lasted about an hour to keep the project on track and clear up any confusion we had from working asynchronously.

Dow quickly took the lead while working on the project. As the group figured out what we wanted to accomplish, Dow often knew the best way to implement that. He used a hands-on leadership style and directed the other group members on what we needed to accomplish for specific tasks.

We knew the buzzer and the buttons would be the most important parts of the project, so figuring out how to use those was the main priority. We made small goals to build momentum like just making the buzzer make noise or making the pushbuttons blink an LED. Then once we accomplished that we moved on to new goals like pressing multiple buttons at once and adding a button to adjust the frequencies being played.

This pattern continued until the project deadline approached. We finalized what we needed to finish up on the project and got the project ready to present in its final state.

Appendix

Musical Notes and their frequencies:

[1] “Note Frequency Chart (Pitch to Note),” *muted.io*. https://muted.io/note-frequencies/

Pinouts for board:

[2] “NUCLEO-L476RG | Mbed,” *os.mbed.com*. https://os.mbed.com/platforms/ST-Nucleo-L476RG/

Cost Estimate for Microcontroller:

[3] “Amazon.com: STM32 Nucleo-64 Development Board with STM32L476RG MCU NUCLEO-L476RG: Electronics,” *Amazon.com*, 2025. https://www.amazon.com/STM32-Nucleo-64-Development-STM32L476RG-NUCLEO-L476RG/dp/B01IO3N646 (accessed Apr. 07, 2025).

Cost Estimate for Board:

[4] “EduBase-V2,” *Trainer4edu.com*, 2025. https://trainer4edu.com/edubase_v2/index.html (accessed Apr. 07, 2025).