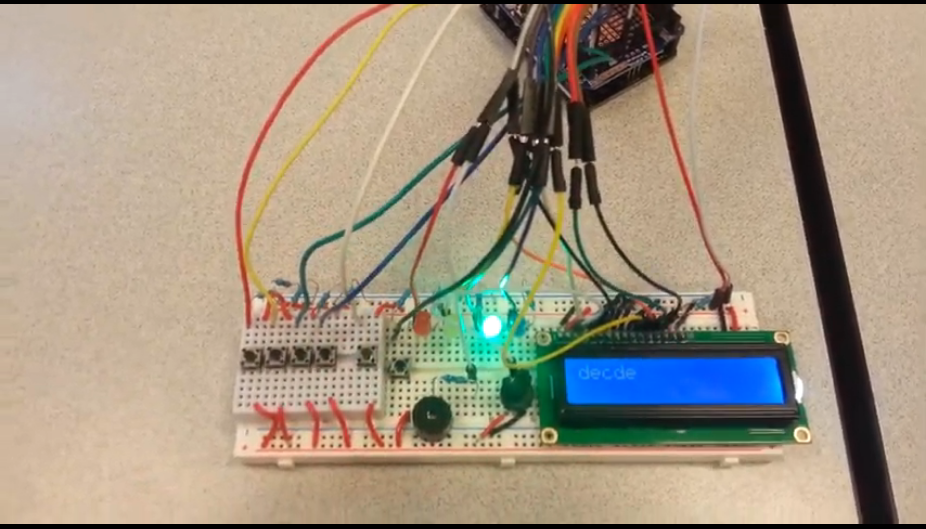
Sean Sands & Max Henderson | CS 351 | February 24, 2019

CS 351 Assignment 2

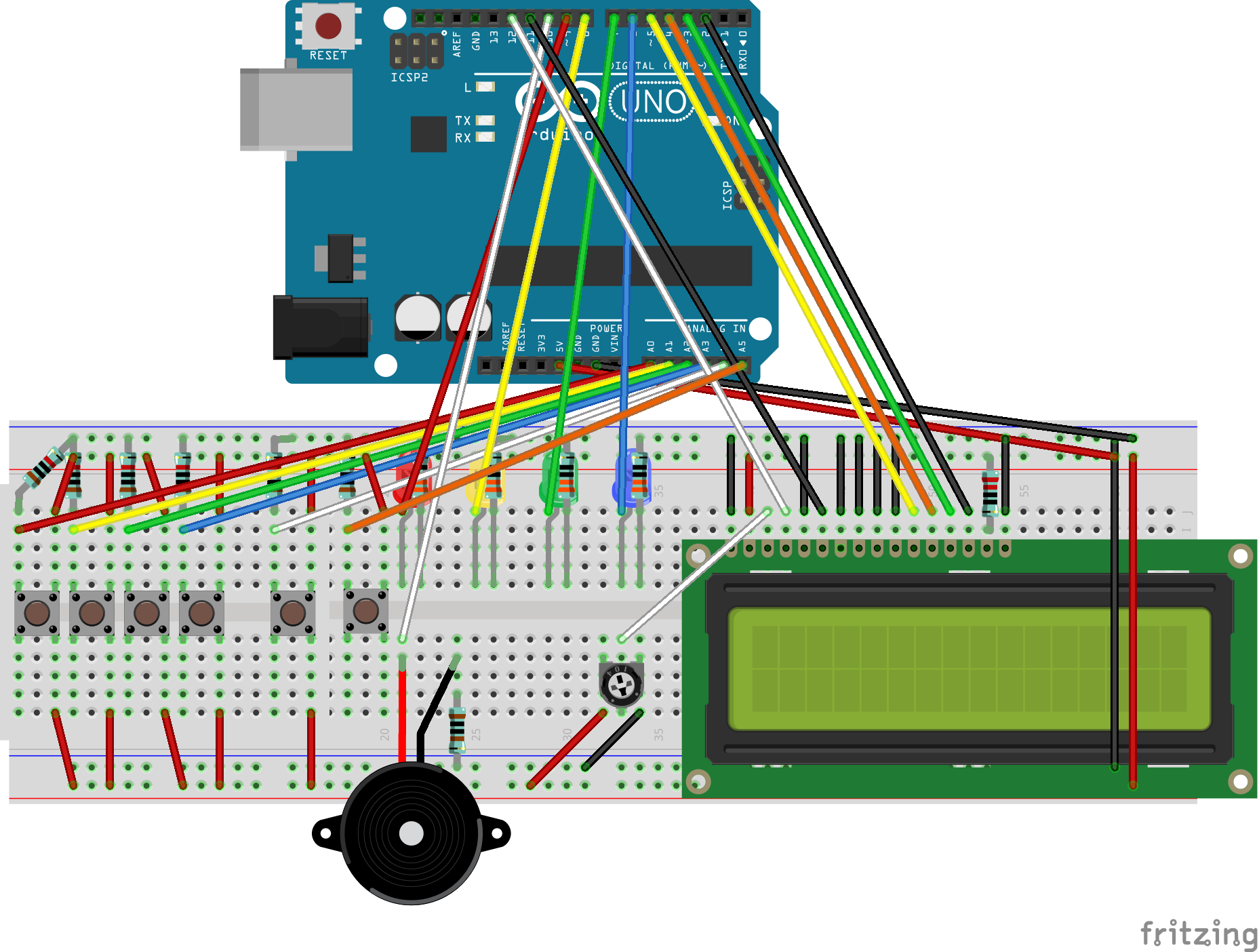
Keyboarduino



# Function

Description of interactive function here

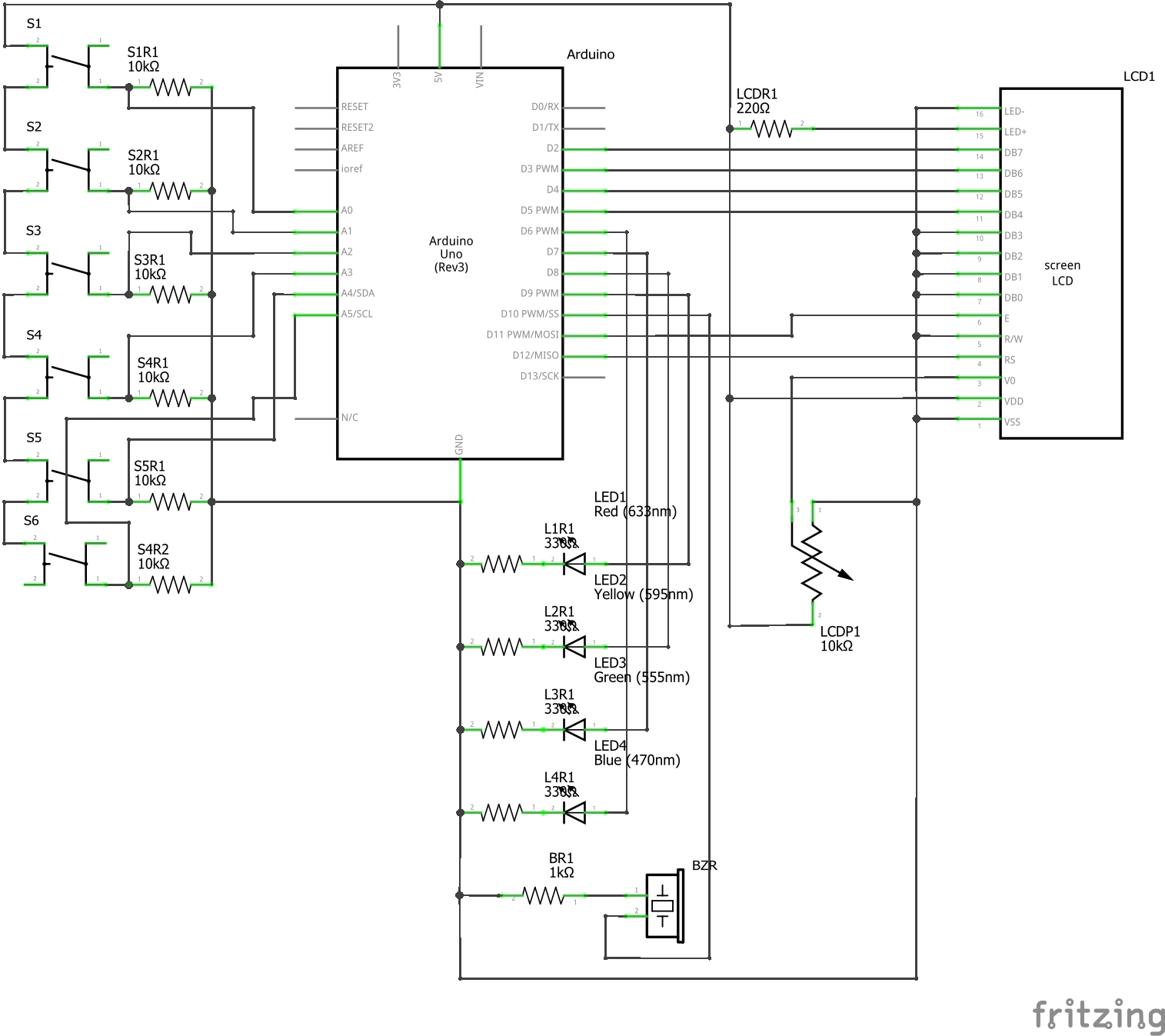
# Circuit



## BreadBoard

All connections to the Arduino are through a Prototyping Shield to allow the project wiring to remain stable even if using the Arduino for another project. 5 of 6 buttons are attached to a mini-breadboard because they fit better there. Each button bridges the gap, connecting top and bottom lines. The mini-breadboard is physically attached to the breadboard through a combination of double-sided tape and twice as many button-to-5V connections as electrically necessary. The rest is directly attached to a breadboard, and laid out very closely to the above image. The four LEDs are located in the top-middle of the board, each with an in-line resistor. The piezo buzzer is located in the bottom-middle of the board with an in-line resistor. The potentiometer is also located in the bottom-middle of the board to the right of the piezo buzzer. The LCD is on the right side of the board. Top and bottom power/ground lines are connected to each other by the wires on the right.

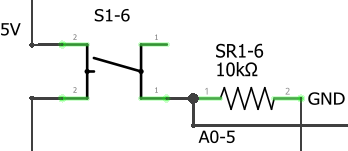
## SCHEMATIC



The sub circuits of this assignment implementation are as follows:

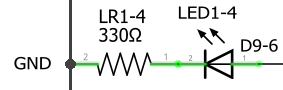
* 6 Button Circuits
* 4 LED Circuits
* 1 Buzzer Circuit
* 1 LCD Circuit

## Button Circuits



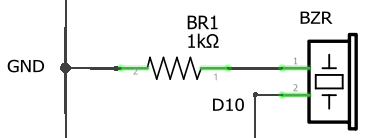
Each button is connected on one end to an input pin and to ground through a 10k resistor, and to 5V on the other end. When open, the input pin is pulled to ground. When closed, the input pin is pulled to 5V.

## LED Circuits



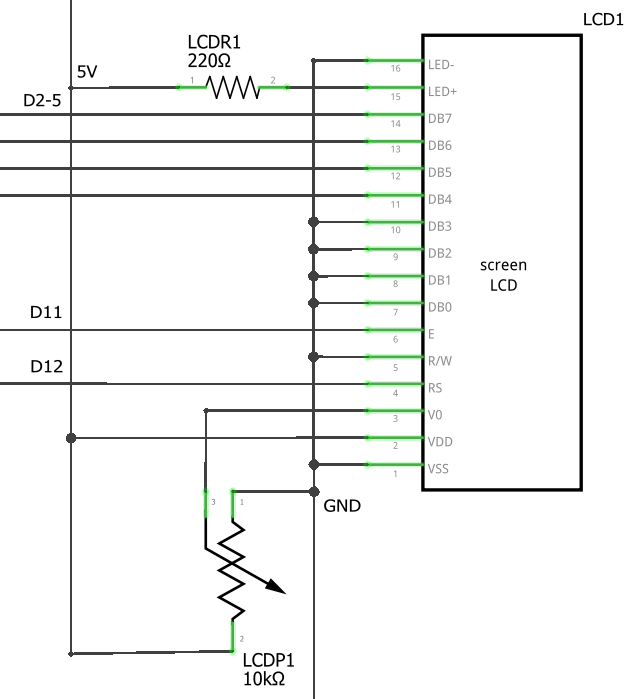
Each LED is connected on the positive end to an output pin and to ground through a 330 resistor on the negative end. When the output pin is on (5V), the LED will be on. When the output pin is off (GND), the LED will be off.

## Speaker Circuit



The buzzer is connected on one end to an output pin and to ground through a 1k resistor on the other end to ensure the pin is not damaged. When a waveform is passed by the output pin, the buzzer emits a tone with the frequency of that waveform.

## LCD Circuit



5V connects to VDD(2) directly, and to LED+(15) through a 220 resistor. Power is provided to the unit through VDD, and the backlight is powered through LED+. 5V also connects to one end of the 10k potentiometer.

GND connects to VSS(1), R/W(5), DB0-3(7-10), and LED-(16). GND also connects to the other end of the 10k potentiometer. VSS completes the power circuit with VDD. LED- completes the circuit with LED+. DB0-3 are unused in 4 bit mode, so are tied to ground to avoid spurious signals. The LCD unit is only used in write mode in this project, so R/W is tied to ground.

D2-5 connect to DB7-4(14-11). These 4 pins are used by the Arduino to communicate with the LCD unit.

D11 connects to E(6). It is a falling edge triggered clock.

D12 connects to RS(4). It determines whether the LCD unit will parse D2-5 as commands or data.

The middle connection of the 10k potentiometer is connected V0(3). As a potentiometer, the voltage can range between the voltages of the other two connections: in this case, 0V and 5V. The potentiometer can be adjusted to alter the contrast on the LCD between text and screen.

# Code

Intro to description of code here

## Preprocessor Commands

The frequencies of the four notes used by the assignment are defined during preprocessing.  
The LiquidCrystal library is included to drive the LCD Unit.

## Global Values

### Constants

int notes[]: array of note frequencies used for tones  
int noteDuration: duration of notes when played by replay(), in 2 ms ticks (125, so 250ms)  
int buttonPins[]: array of pins associated with the six buttons (A0-A5)  
int noteLedPins[]: array of pins associated with the four LEDs (9,8,7,6)  
int speakerPin: pin associated with the speaker (10)  
int numButtons: number of buttons, used for sanity checks by some functions  
int rs,en,d4,d5,d6,d7: pins used by the lcd (12,11,5,4,3,2)

### Variables

int buttonDebounceStates[]: array of integers used as rolling buffers by debounceBtn(), volatile  
bool buttonPressedStates[]: array of bools written by debounceBtn() and read by readBtn(), volatile  
bool lastButtonPressedStates[]: array of bools used by readBtn(), compared to the above array  
pressing: current actively pressed button, determined by press() when called by readBtn()  
playing: index of the next note to play when replaying, used by replay(), modified by press()  
playTime: time playing the current replayed note, incremented by the ISR, read by replay(), volatile  
played[]: array of characters representing recorded notes, allows easy translation to lcd (c is ‘c’, etc),  
 written by record(), read by replay(), cleared by reset()  
playedSize: number of elements in above array  
playedCap: max number of elements in above array

### Objects

lcd: a LiquidCrystal object that drives the LCD Unit

## Functions

rawBtnPressed(int btn): wrapper for digitalRead()s of buttonPins, called by debounceBtn()  
debounceBtn(int btn): updates a ButtonDebounceState, buttonPressedState, explained below  
noPlay(): stops tone() and turns off all LEDs  
play(char note): plays a note and lights the associated LED on a valid character, else noPlay()  
record (char n): adds a char to played(if there’s room), adds a char to the LCD (if there’s room)  
replay(): plays the next note in played if playTime>noteDuration,  
 called by loop() when pressing is 4, or press() (which also sets playTime>noteDuration)  
reset(): clears played, clears LCD  
press(): does the appropriate action based on a change in button state  
readBtn(int btn): compares a buttonPressedState to lastButtonPressedState,  
 assigns last and calls press() when they aren’t equal

## Setup

setup() starts by disabling interrupts. It then sets pressing to numButtons, a null state. It sets the mode of button pins to input and LED pins to output. It sets playTime to 0. It allocates 256 bytes for playing, and sets its capacity to 255 and size to 0. It initializes the lcd with lcd.begin(16, 2) (the width and height of the lcd). It then sets a number of internal values related to Timer1 in order for it to cycle every 2 ms(500 Hz) and enable compare interrupts. A more detailed explanation of the magic numbers is below in the Interrupt Service Routine section. Lastly, it reenables interrupts and completes.

## Loop

This function is called repeatedly. Each time it is called, it calls readBtn() for each button. Additionally, if pressing is equal to 4 (replay mode), it calls replay().

## read, press, Replay, and Record

readBtn() compares buttonPressedState, the debounced state of a button, to lastButtonPressedState, the last state of that button readBtn() has read. The comparison is indirect through a temporary value so an interruption changing buttonPressedState will not cause a mis-assignment. If they are not equal, the lastButtonPressedState is assigned, and press() is called for that button and that state.

press() takes an integer signifying a button and a bool signifying that button’s press state. If a tone button (0-3) is pressed down, it plays a tone, turns on a light, and sets pressing to that button. If the replay button is pressed down, it sets pressing to the replay button (which is used by loop to call replay), sets playTime to noteDuration+1 (so replay will immediately begin playing the first note), sets the playing index to 0, and calls replay(). If the reset button is pressed down, it calls reset(), which clears the note array. If a note button is unpressed and pressing is set to that button (meaning the button that was unpressed was the active tone), it stops playing the tone and turns off all LEDs. Nothing presently happens when replay or reset buttons are released.

replay() compares playTime (incremented by the ISR) to noteDuration. If playTime is greater, it compares the playing index and playedSize. If playing is lesser, it calls play() for the note at the index playing and increments playing, otherwise it stops playing by calling noPlay().

record() takes a character (‘c’ – ‘f’’). If played is already full, it does nothing. Otherwise, it appends the character to played. It also appends the character to the lcd if there is space on the screen.

## Interrupt Service Routine

Due to values set in setup(), ISR(TIMER1\_COMPA\_vect) will be called every two milliseconds (plus the time this function takes to execute). This occurs due to TCNT1 reaching the value of OCR1A (31999). TCNT1 is incremented according to the values of scaling flags. Since the scaling flags in TCCR1B are set as they are, the time is not multiplied (scaling options include 1(current), 8, 64, 256, 1024, as well as disabled and settings that instead increment it on rising or falling edges of pin 1 input). Because the compare interrupt flag in TIMSK1 is enables, it calls the ISR with the above vector. When the ISR is called, and is then reset back to 0 due to the CTC flag in TCCR1B. (16M/32k ~= .002) This function increments playTime and performs debounceBtn() for each button.

## Debouncing Algorithm

Because of imperfections in both user input and button build quality, the main loop does not directly read using digitalRead()s. It instead reads values set by a debouncing algorithm to avoid misreading momentary losses of connection as double presses.

Every time debounceBtn() is called, it left shifts that button’s buttonDebounceState and OR masks that with the inverse of that button’s digitalRead(). This uses buttonDebounceState as a 16-bit buffer of previous inputs.

When the buttonDebounceState is equal to 0x8000 (1 followed by 15 0s) it assigns that button’s buttonPressedState to true. This state occurs after the button has been held down for 30ms without interruption. By requiring the opposite value in the first bit, the function will not constantly assign a value to debounceButtonState during steady input. Conversely, when buttonDebounceState is equal to 0x7fff(0 followed by 15 1s), it assigns that button’s buttonPressedState to false. The above explanation with a reversal of terms (down to up) explains the reasoning for the choice of value. The above values can be changed and/or masked to make smaller buffers for shorter debouncing times. (0x0800 and 0x07ff, with & 0x0fff for an 11-tick buffer for example)