Tic Tac Toe

Lance Yoshizaki, Guido De Santis

Overview of the Project

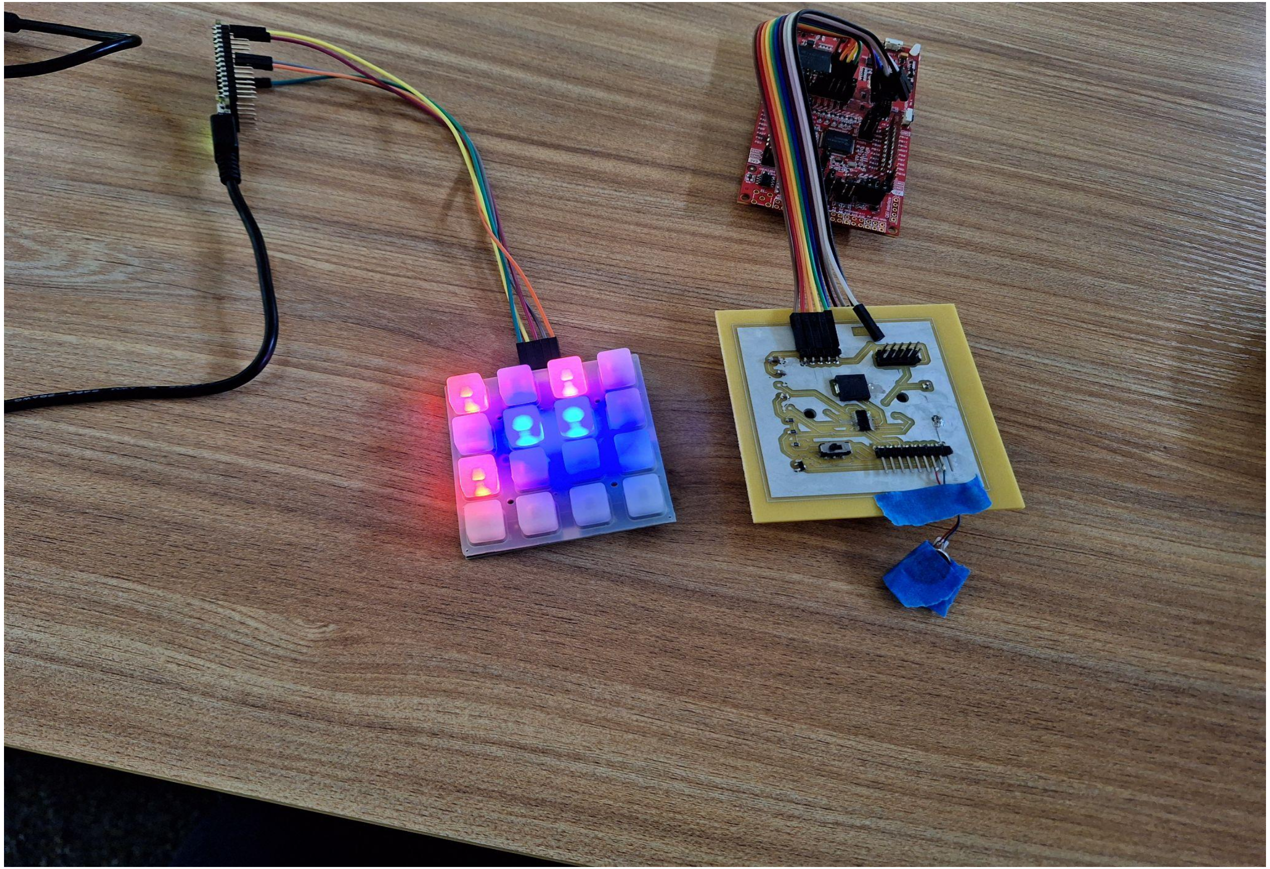
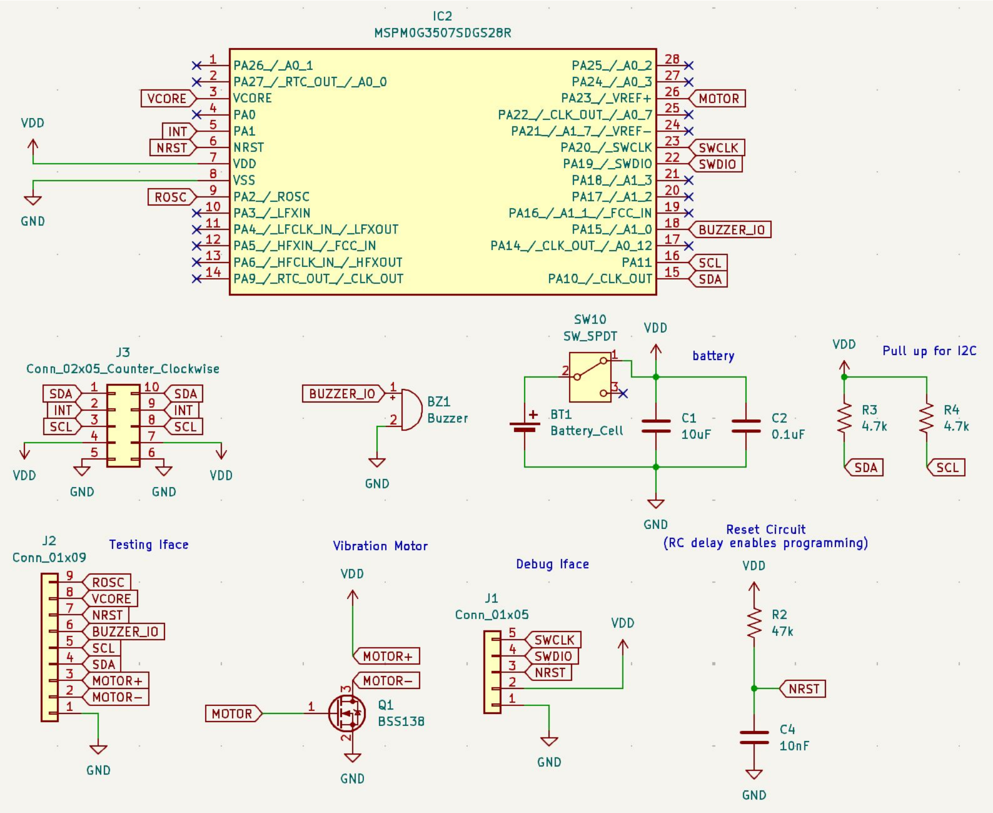


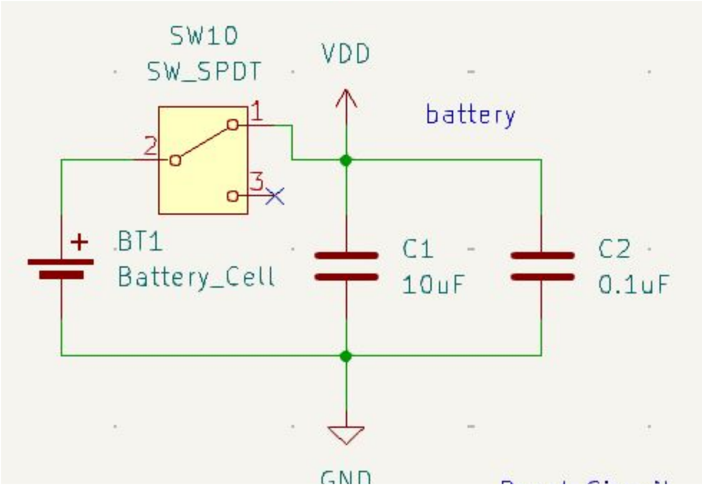
Table of Contents

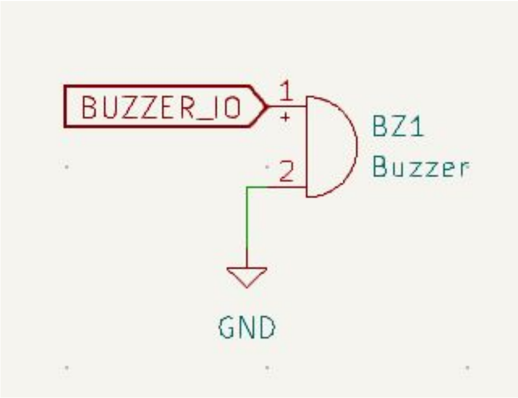
Project Design

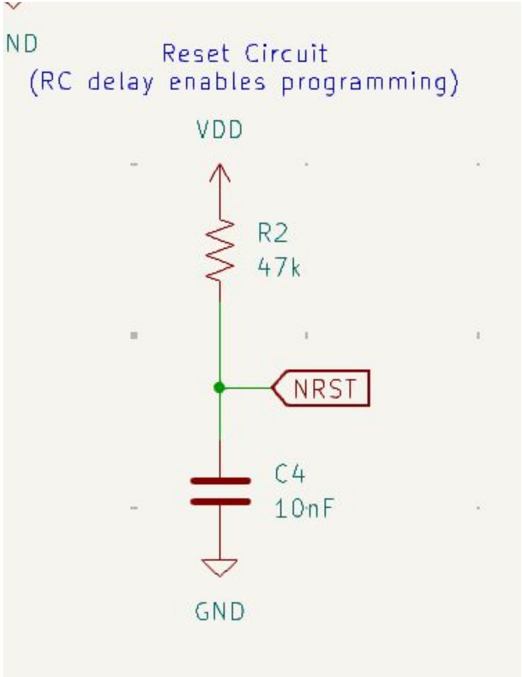
Schematic Design



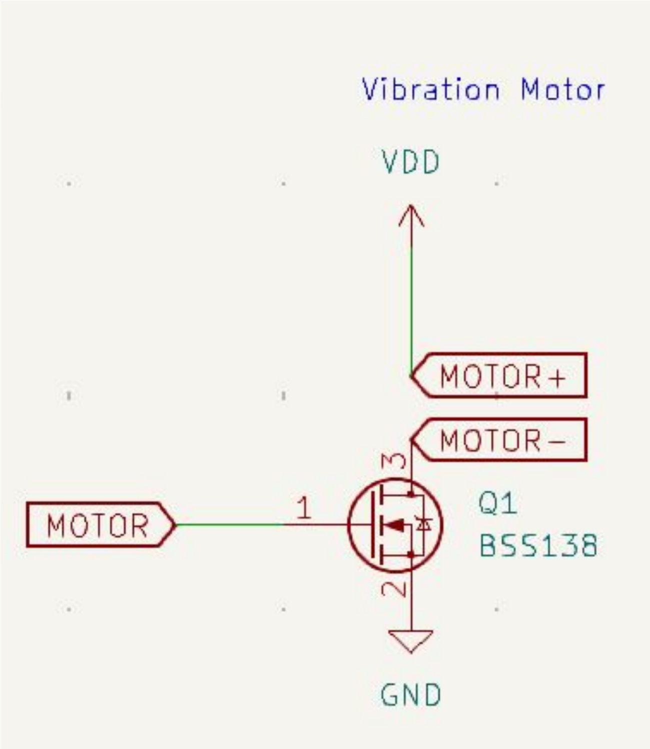
Familiar Design Choices

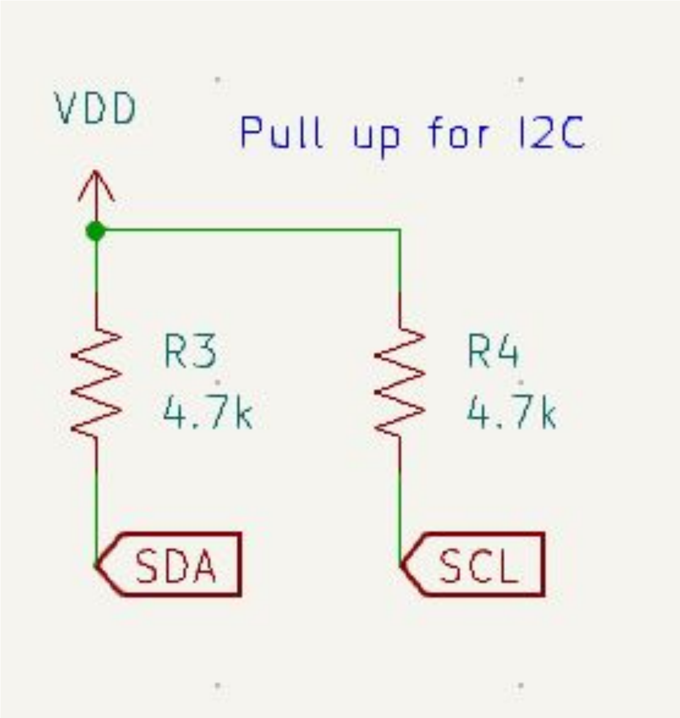






Design Choices



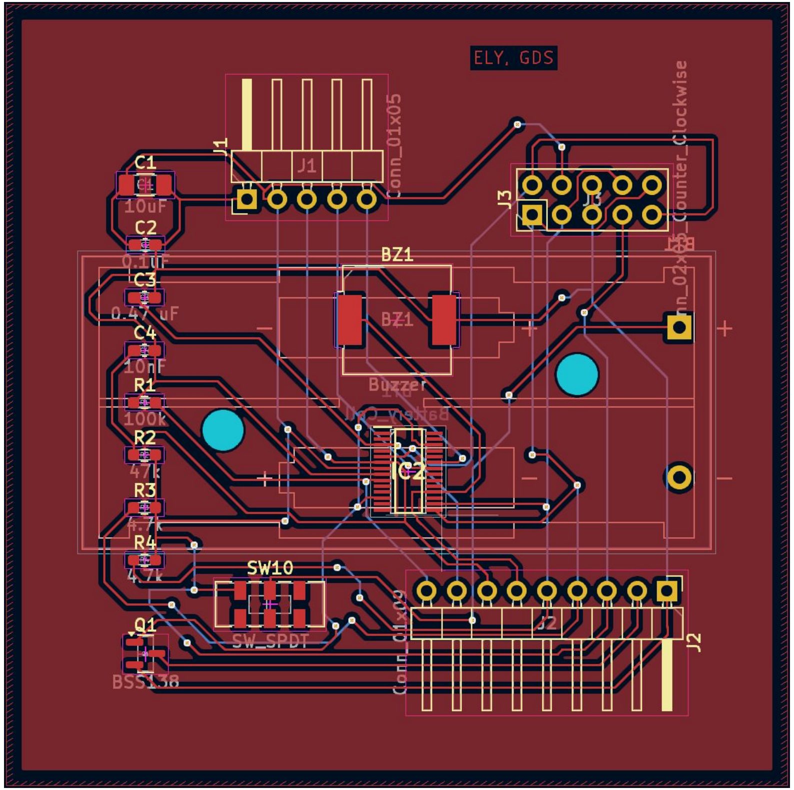


Design Choices

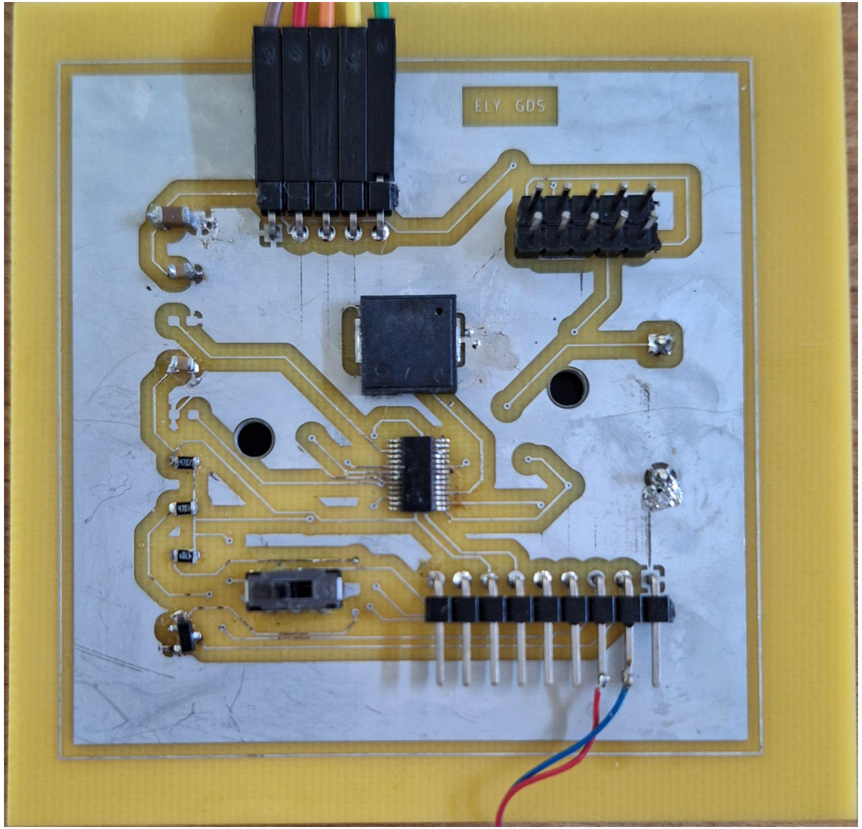
- We decided to use the MSPM0G3507SDGS28R because we didn't plan to use too many peripherals and one set of GPIO pins would suffice
    - it is also easier to solder than other packages with more pins
- NMOS to drive vibration motor
    - Typical GPIO pin current is much less than one that can drive a motor
- Pull up resistors for I2C Communication
    - 4.7 kOhms standard pull up resistance is used

PCB Design

KiCad Schematic →



Back and Front view of the PCB



PCB Design Considerations

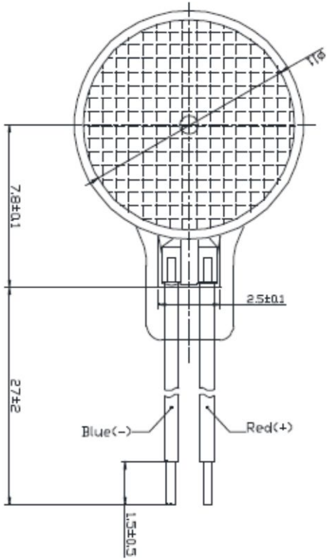
- Put the MCU in the center of the PCB for easier routing
- Imported design constraints from Lab 8 (8 mils of clearance)
- Make sure traces are not too close to each other to make soldering easier
- Angled connector for easier connection
- Made sure that no shorts occurred during soldering and verified connections with a multimeter

New Components

Vibration Motor

- Part: HD-EMC1103-LW27
- We are using 3.3V logic and this lies in operating voltage range
- Need to drive a higher current than what the GPIO pins can provide → use an NMOS to control switching

Parameters	**Values**	**Units**
Input Voltage	3.7	V_{DC}
Operating Voltage Range (DC)	3.0 ~ 4.5	V_{DC}
Starting Voltage	2.5	V_{DC}
Direction of Rotation	CW	-
Rated Speed Rated Voltage, Rated Load, Motor Fixed	8,000 ± 3,000	RPM
Rated Load Current Maximum at 3.7 V_{DC}, Rated Load, Motor Fixed	85	mA
Starting Current Maximum at 2.5 V_{DC}, Rotor Locked	170	mA
Mechanical Noise	≤ 50	dB



N-Channel MOSFET

- Part: BSS138
- 3.3V is enough to switch the transistor, also can provide sufficient current

I_S	Maximum Continuous Drain–Source Diode Forward Current		–	–	0.22	A
ON CHARACTERISTICS						
$V_{GS(th)}$	Gate Threshold Voltage	$V_{DS} = V_{GS}$, $I_D = 1$ mA	0.8	1.3	1.5	V
$\frac{\Delta V_{GS(th)}}{\Delta T_J}$	Gate Threshold Voltage Temperature Coefficient	$I_D = 1$ mA, Referenced to 25°C	–	–2	–	mV/°C
$R_{DS(on)}$	Static Drain–Source On–Resistance	$V_{GS} = 10$ V, $I_D = 0.22$ A	–	0.7	3.5	Ω
		$V_{GS} = 4.5$ V, $I_D = 0.22$ A	–	1.0	6.0	Ω
		$V_{GS} = 10$ V, $I_D = 0.22$ A, $T_J = 125$°C	–	1.1	5.8	
$I_{D(on)}$	On–State Drain Current	$V_{GS} = 10$ V, $V_{DS} = 5$ V	0.2	–	–	A
g_{FS}	Forward Transconductance	$V_{DS} = 10$ V, $I_D = 0.22$ A	0.12	0.5	–	S

Adafruit NeoTrellis 4x4 Board

-Has 5 pins (VIN,GND,INT,SDA,SCL)

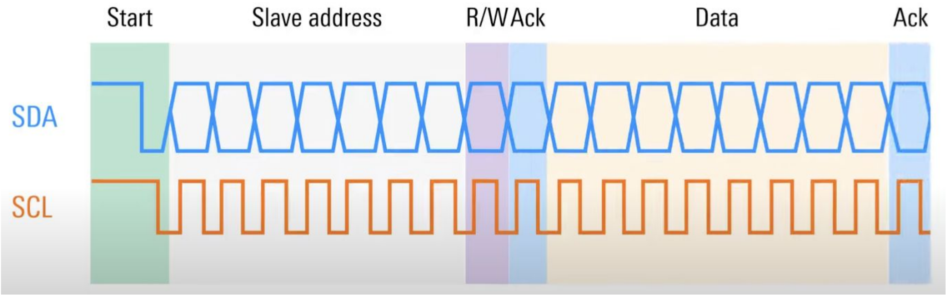
-Both button management and LED driving is handled over plain I2C.

-For proof of concept we used an Arduino Nano Every to run the game logic on the board in C++

-It is possible to tile up to 32 boards together with only one I2C connection

I2C

- Unlike SPI (which uses 4 lines for communication), I2C only requires 2 lines of communication



[2]

- Messages on the SDA line consist of address bits to identify the peripheral (0x2E for the NeoTrellis) and data bits that are written to the peripheral or read by the controller

Implementation

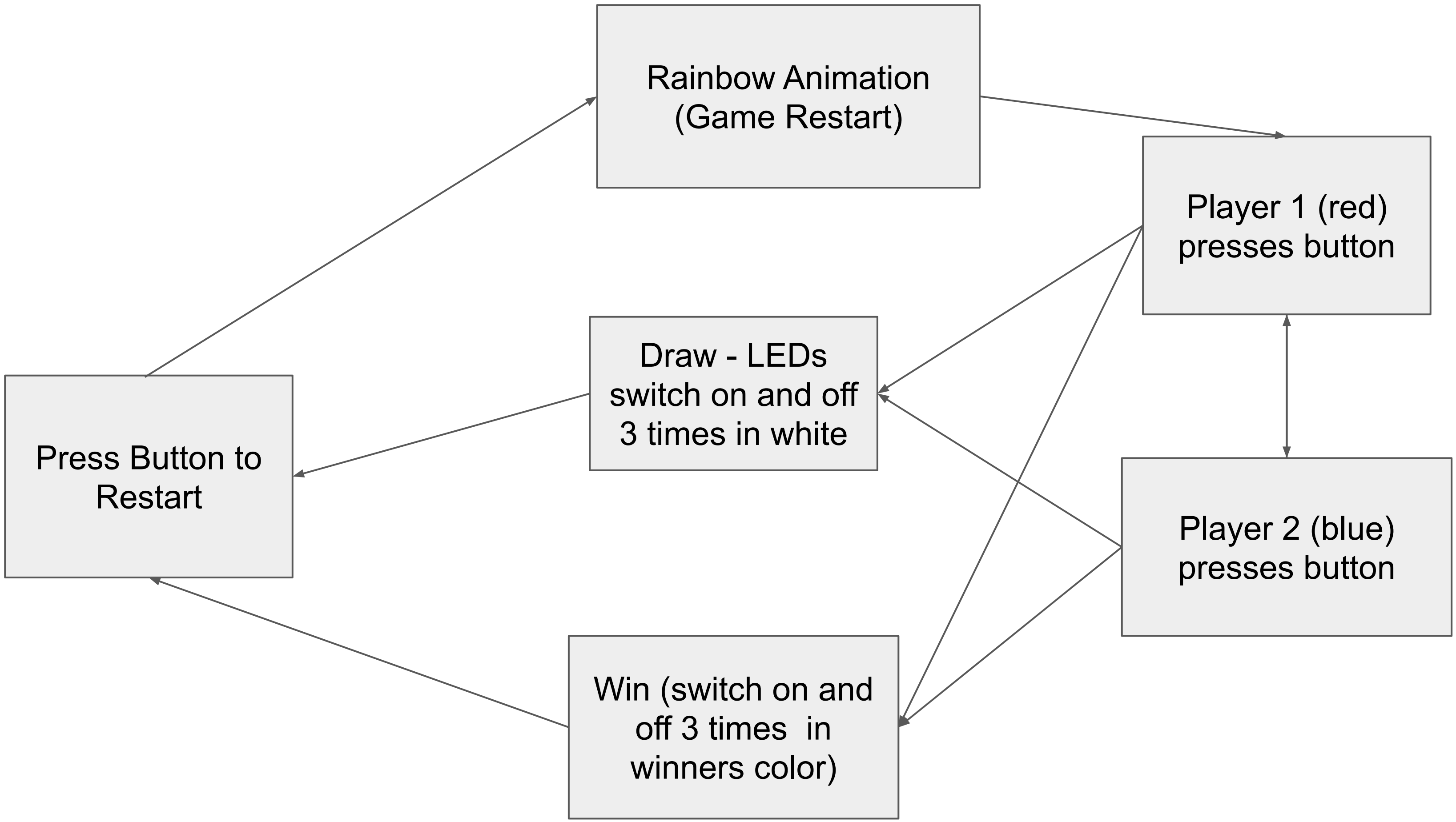
Implementation of Vibration Motor

- We apply a gate voltage to the NMOS using a GPIO pin to drive a current through the vibration motor
- Similar to playing sound on the buzzer
    - combine audio and haptic feedback

```c
void playToneAndVibrate(uint32_t freqHz, uint32_t duration_ms) {
    uint32_t period_us = 1000000 / freqHz;
    uint32_t half_period_us = period_us / 2;

    uint32_t elapsed_us = 0;
    uint32_t motor_toggle_interval_us = 20000; // motor toggles every 20ms
    uint32_t motor_timer_us = 0;
    uint8_t motor_state = 0;

    while (elapsed_us < duration_ms * 1000) {
        // Turn on buzzer
        BUZZER_IO_PORT->DOUTSET31_0 = BUZZER_IO;

        // Check if we need to toggle the motor NMOS' gate input
        if (motor_timer_us >= motor_toggle_interval_us) {
            motor_timer_us = 0;
            motor_state = !motor_state;
            if (motor_state) {
                MOTOR_PORT->DOUTSET31_0 = MOTOR;
            } else {
                MOTOR_PORT->DOUTCLR31_0 = MOTOR;
            }
        }

        delayMs(half_period_us / 1000);
        elapsed_us += half_period_us; motor_timer_us += half_period_us;

        // Buzzer OFF
        BUZZER_IO_PORT->DOUTCLR31_0 = BUZZER_IO;

        delayMs(half_period_us / 1000);
        elapsed_us += half_period_us;
        motor_timer_us += half_period_us;
    }

    // Ensure everything is OFF
    BUZZER_IO_PORT->DOUTCLR31_0 = BUZZER_IO;
    MOTOR_PORT->DOUTCLR31_0 = MOTOR;
}
```

Game Logic

1) When the game starts, a 3x3 Matrix (top left when wires stick out from the top) of the 4x4 grid lights up by calling the Wheel() function from the Seesaw library, which generates smooth color gradients across the RGB color wheel. The function works by taking a value from 0 to 255 and returns a corresponding RGB color. In this case it generates a rainbow effect on the NeoTrellis that goes from left to right.
1) The first player (represented by red leds) presses a button in the 3x3 matrix
2) The second player (represented by blue leds) presses another button in the matrix
3) If a player succeeds in placing three of their colored LEDs in a row — horizontally, vertically, or diagonally — they win, and all LEDs in the 3×3 matrix flash in that player's color three times. If all buttons in the matrix have been pressed and no player has won, the game ends in a draw and the 3×3 LEDs flash white three times.
4) Pressing any button in the matrix after a win or draw resets the game and restarts the rainbow animation.



Rainbow Animation (Game Restart)
Player 1 (red) presses button
Draw - LEDs switch on and off 3 times in white
Press Button to Restart
Player 2 (blue) presses button
Win (switch on and off 3 times in winners color)

Arduino Code

-In order to run this code, you need to install the Adarfruit seesaw library on Arduino IDE, select the right port connected to the Arduino and upload the code on the board.

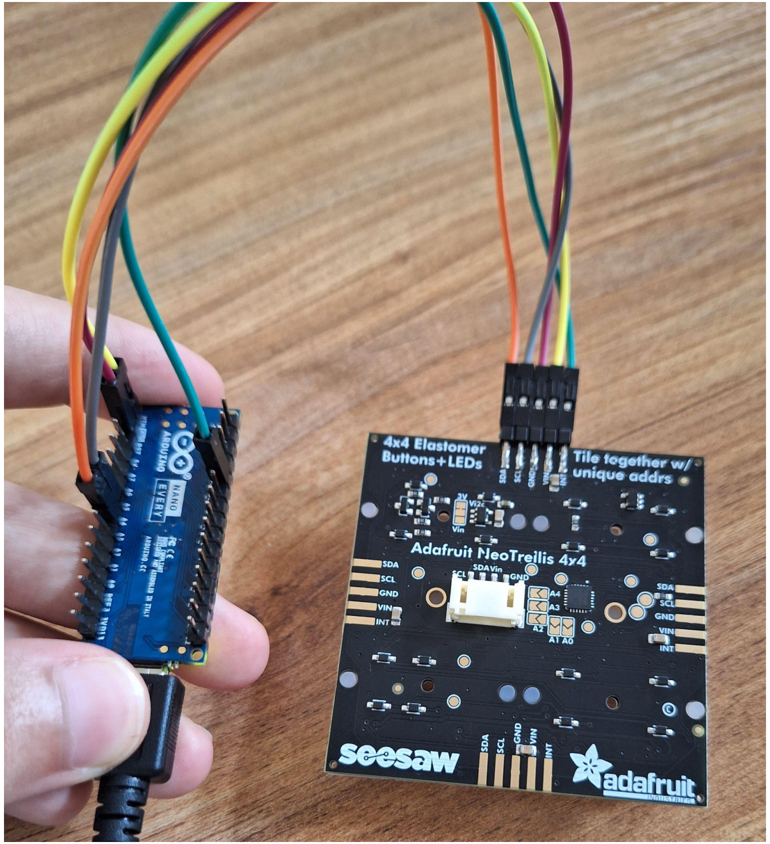
- To understand the functions of the seesaw library, we looked at the pattern_game sample code which can be found in Examples on your IDE.

```
bool checkWin(int player) {
  for (int i = 0; i < 3; i++) {
    if (board[i][0] == player && board[i][1] == player && board[i][2] == player) return true;
    if (board[0][i] == player && board[1][i] == player && board[2][i] == player) return true;
  }
  if (board[0][0] == player && board[1][1] == player && board[2][2] == player) return true;
  if (board[0][2] == player && board[1][1] == player && board[2][0] == player) return true;
  return false;
}

bool isDraw() {
  for (int r = 0; r < 3; r++)
    for (int c = 0; c < 3; c++)
      if (board[r][c] == 0) return false;
  return true;
}

void resetGame() {
  // Rainbow animation
  for (int r = 0; r < 3; r++) {
    for (int c = 0; c < 3; c++) {
      int idx = gameKeys[r][c];
      uint32_t color = Wheel(map(idx, 0, 10, 0, 255));
      trellis.pixels.setPixelColor(idx, color);
      trellis.pixels.show();
      delay(60);
```

Arduino Nano Every Pinout

-The datasheet can be found at this link https://content.arduino.cc/assets/Pinout-NANOevery_latest.pdf.

-We connected the VIN Pin on the Neotrellis to the VIN pin on the Arduino, Ground to Ground Pin, SDA to pin A4, SCL to pin A5 and Interrupt to pin D2



Implementation on MSPM0

- Adafruit has their own libraries written in C++ and Python, for use for an external Adafruit seesaw chip
- TI provides driverlib code in C to interface with I2C devices
- Little implementations of NeoTrellis could be found in C
    - Tried to port over the C++ libraries and adapt the driverlibs to the NeoTrellis but met with much difficulty
    - Debugging showed that the peripheral connection was responding to the MSPM0 but LEDs didn’t turn on

Sources Referenced

Sources
[1] https://learn.sparkfun.com/tutorials/i2c/all - “I2C”
[2] https://www.youtube.com/watch?v=CAvawEcxoPU&t=121s - “Understanding I2C”. Rohde and Schwarz.