PianoBit

Group: 331CA
Student: Mara Fichioș

Project Summary: The PianoBit project is an innovative and educational digital mini-piano that utilizes an Arduino Uno as the core control unit. Designed to mimic the functionality of a basic piano, the system includes a 4×4 button matrix, a set of 16 LEDs, an active buzzer, and an LCD screen. The aim of the project is to build a simple, interactive musical instrument while exploring various hardware techniques such as button matrix scanning, LED multiplexing, and sound generation.

General Description

- PianoBit is a digital mini-piano constructed using an Arduino Uno, a 4×4 matrix of buttons, 16 corresponding LEDs, and an active buzzer.
- The core objective of the project is to emulate the behavior of a basic electronic piano with 16 keys. It is designed to explore efficient hardware management using direct multiplexing, thus eliminating the need for dedicated shift registers.
- Originally envisioned as an 8-key prototype, the design was expanded to 16 keys in order to mirror the octave structure of a real instrument more closely and to meet the increased complexity requirements specified in the course. The new version also has an LCD which showcases the user the current note that they are playing.
- The system provides a practical and educational platform for understanding key matrix scanning, LED multiplexing, and sound generation, while also working with registers and Arduino.

The block diagram illustrates how the core modules of the PianoBit project are functionally interconnected to achieve real-time user interaction. At the center of the system lies the Arduino Uno, which acts as the main control unit. It manages data flow between input and output modules, orchestrating the behavior of the piano. The button matrix connects directly to the Arduino's digital pins, and the Arduino detects exactly which key has been pressed. This key is translated into a musical note. Once a key is identified, the Arduino triggers two parallel outputs: it sends a corresponding frequency signal to the buzzer, generating an audible tone and it lights up the matching LED by transmitting data serially to the shift registers. These convert the serial input into parallel output, illuminating the specific LED assigned to the pressed key.

Simultaneously, the Arduino communicates with the I2C LCD display to visually show the name of the note being played. This communication happens via the I2C protocol using only two data lines (SCL

and SDA), allowing the LCD to operate without consuming multiple digital I/O pins.All modules draw power from a shared 5V supply. This architecture ensures that pressing a single key results in synchronized audio, visual, and textual feedback, making the PianoBit both interactive and educational.

Hardware Design

Component List:

Component	Quantity	Description
Arduino Uno	1	Microcontroller for project control
Buttons	16	16 buttons arranged in a matrix configuration
74HC595 Shift Register	2	Serial-to-parallel shift registers for controlling LEDs
LEDs	16	LED indicators to correspond to each key
220Ω Resistors	16	Current limiting resistors for LEDs
Buzzer	1	Passive Buzzer for sound output
Breadboard	4	For connecting components without soldering
Jumper Wires	Multiple	For making connections between components
I2C LCD	1	Displays message for user

Overview on hardware:

- **Button Matrix**: the 4×4 button matrix uses multiplexing to reduce the number of pins needed on the Arduino. Each row pin is connected to a digital output pin, and each column pin is connected to a digital input pin. When a row is activated, the Arduino checks if any button in the corresponding column is pressed. The button matrix is multiplexed, meaning we scan one row at a time while the others are inactive.
- **Shift registers**: the 74HC595 shift registers are used to control the 16 LEDs. These shift registers use the Serial Data Input (DS) to receive data, the Shift Register Clock Pin (SH_CP) to clock the data in, and the Latch Pin (ST_CP) to latch the data into the shift register. The Arduino sends 8-bit data to the shift registers, which control the state of the LEDs (on or off). The LEDs are arranged in two groups of 8, each controlled by a separate shift register. By connecting the shift registers in daisy chain, the second shift register is controlled by the first one, effectively expanding the number of outputs.
- **LEDs**: 16 LEDs are divided into two groups of 8, each controlled by a separate 74HC595 shift register. The shift registers are connected in **daisy chain mode**, allowing the Arduino to control both registers with just 3 pins (Data, Clock, Latch).
- **Buzzer**: the buzzer is used to generate sound when a key is pressed. The Arduino outputs a PWM signal to the buzzer pin (Pin 10), which generates different frequencies based on the key pressed.
- **Power supply**: the project runs on 5V provided by the Arduino, and the components have minimal power consumption. The shift registers, LEDs, and buzzer are powered by the 5V supply.
- **I2C LCD Display:**

The LCD uses the I2C protocol with two dedicated lines:

1. **SDA (Serial Data Line)** connected to Arduino analog pin A4
2. **SCL (Serial Clock Line)** connected to Arduino analog pin A5

This communication interface allows sending commands and data over just two wires, greatly reducing wiring complexity. The LCD module typically includes a PCF8574 I/O expander chip, which converts the serial I2C data to parallel signals needed to drive the LCD. Pull-up resistors (usually onboard) maintain stable HIGH levels on SDA and SCL lines. The I2C address used for the LCD in this project is 0x27 (7-bit), shifted left by one bit in code to account for the read/write flag.

Optimizations:

In this project, I've utilized multiplexing for the button matrix to reduce the number of pins required for detecting 16 buttons. By multiplexing, one row at a time is activated while scanning for button presses in the corresponding columns. This allows the usage of only 8 I/O pins to control 16 buttons instead of needing 16 individual pins.

For the LEDs, I used 74HC595 shift registers to control 16 LEDs with only 3 I/O pins (Data, Clock, and Latch). The shift registers are connected in daisy-chain mode, meaning the output of the first shift register is connected to the input of the second. This allows the Arduino to control a total of 16 outputs (8 from each shift register) while only using 3 pins, significantly reducing the number of I/O pins required for controlling the LEDs.

Below there are some pictures that showcase the whole circuit:

Pin Usage

Button Matrix:

- **Rows**: Pins 2, 3, 4, 5 (configured as output).
  1. These pins are used as outputs to drive the rows of the button matrix. Since the matrix requires only one row to be active at a time, using output pins allows us to control which row is being scanned.
- **Columns**: Pins 6, 7, 8, 9 (configured as input with pull-up resistors).

1. These pins are configured as inputs with internal pull-up resistors, as the columns in a button matrix need to be read to detect key presses. The pull-up resistors ensure that the default state of the columns is HIGH, making it easier to detect when a button is pressed (which will pull the corresponding column LOW).

Shift Registers:

- **Pin 11**: Data pin (DS) – connected to the first shift register.
  1. Pin 11 is used to send serial data (on or off) to the shift register. This allows the Arduino to control multiple LEDs with just a few pins by sending 8-bit data at a time to the shift registers, which then control the corresponding LEDs.
- **Pin 13**: Clock pin (SH_CP) – connected to both shift registers.
  1. Pin 13 is used to send the clock signal, which synchronizes the data being transferred to the

shift registers. Each pulse of the clock pin shifts the data one bit further into the shift register.

- **Pin 12**: Latch pin (ST_CP) – connected to both shift registers.
  1. Pin 12 controls the latch pin of the shift register. When the latch pin is triggered, the shift register outputs the data stored and updates the state of the LEDs. This pin is essential for making sure that the LED states are properly updated when the data has been shifted in.

Pin 10: Buzzer – connected to the passive buzzer for generating sound.

1. Pin 10 is a PWM-capable pin used to generate the different frequencies required for the buzzer. By varying the frequency with the `tone()` function, the Arduino can produce musical notes corresponding to the key presses.

LEDs: The shift registers control the LEDs through the Q0-Q7 pins on each shift register. The first shift register controls LEDs 0-7, and the second shift register controls LEDs 8-15.

1. By using the Q0-Q7 pins of each shift register, we can control up to 16 LEDs using only 3 pins on the Arduino (Data, Clock, and Latch). This greatly reduces the number of I/O pins required for controlling the LEDs, allowing us to build a more efficient and scalable system.

I2C LCD Display Pins:

- **Pin A4 (SDA - Serial Data Line)** - Bidirectional data line for I2C communication. Transmits commands and data between Arduino and LCD module. Requires pull-up resistors to ensure stable logic levels.
- **Pin A5 (SCL - Serial Clock Line)** - Clock line generated by the master (Arduino) to synchronize data transfer on SDA. Also requires pull-up resistors.

Software Design

Development Environment:

- Arduino IDE

Libraries Used:

No external Arduino libraries were used in this project. All communication protocols and peripheral controls were implemented manually by direct register manipulation and custom functions.

Explanation:

- The project uses low-level control of the I2C bus through direct manipulation of the AVR TWI (Two Wire Interface) registers (`TWBR`, `TWCR`, `TWDR`, `TWSR`), instead of relying on the Arduino Wire library. - LCD communication is implemented manually by sending commands and data over I2C using bit-banged sequences with precise timing. - Shift registers and keypad scanning are handled using direct port manipulation and interrupts without external libraries.

Global Constants and Defines for LCD:

- `define LCD_ADDR (0x27 « 1)`

```
Defines the 7-bit I2C address of the LCD module shifted left by one bit to
form the 8-bit address used by the TWI hardware.
```

- `define LCD_BACKLIGHT 0x08`

```
Controls the LCD backlight bit on the PCF8574 I/O expander. Setting this bit
turns the backlight on.
```

- `define ENABLE 0x04`

```
The Enable (EN) signal bit for the LCD controller, used to latch commands or
data.
```

- `define READ_WRITE 0x02`

```
Read/Write bit; set low for write operations (not used for reading in this
code).
```

- `define REGISTER_SELECT 0x01`

```
Selects whether data sent to the LCD is a command (`0`) or data (`1`).
```

Software Structure:

* **Initialization Functions:**

1. `setup()` initializes serial communication, I2C registers, LCD, pin modes for rows, columns, buzzer, and shift registers.
2. Configures Timer1 for row multiplexing and enables pin change interrupts on keypad columns.

* **Interrupt Service Routines (ISRs):**

1. `ISR(TIMER1_COMPA_vect)`: Cycles active row in the keypad matrix by toggling row pins to enable scanning one row at a time.
2. `ISR(PCINT2_vect)` and `ISR(PCINT0_vect)`: Detect any change on keypad columns and set a flag (`keypadChanged`) to trigger keypad scanning.

* **Keypad Scanning:**

1. `scanKeypad()` scans the 4×4 matrix by activating rows sequentially and reading columns from port registers.
2. Implements debounce logic by timing stable button presses before confirming input.
3. Updates buzzer tone, LEDs, and LCD message when a valid key press or release is detected.

* **LCD Control:**

1. Custom low-level I2C communication functions (`i2c_init()`, `i2c_start()`, `i2c_write()`, etc.) handle communication with the LCD via the I2C bus.
2. `lcd_send()` splits data/commands into 4-bit nibbles, sending each half to the LCD with control signals.
3. High-level LCD functions (`lcd_clear_custom()`, `lcd_setCursor_custom()`, `lcd_print_custom()`)

control display content.

* **LED Control:**

1. `updateLED()` uses two chained 74HC595 shift registers controlled by `shiftOut()` to light LEDs corresponding to pressed keys.
2. Interrupts are temporarily disabled during shifting to prevent timing issues.
3. `clearAllLEDs()` turns off all LEDs by shifting zeroes to both registers.

* **Main Loop:**

1. Continuously checks if a keypad change flag is set by ISRs to scan keypad immediately.
2. If no immediate change, performs periodic keypad scanning to catch any missed input.

* **Sound Generation:**

1. Uses Arduino `tone()` function to play frequencies mapped to pressed keys via the buzzer.
2. Stops tone when no button is pressed.

Functions:

Function	Role / Description
`setup()`	Initializes pins, timers, interrupts, LCD, and hardware peripherals. Prepares the system for operation.
`loop()`	Main loop that checks for keypad changes via interrupts or periodic scanning, then processes key states.
`scanKeypad()`	Scans the 4×4 button matrix to detect which button is pressed, implements debounce logic, updates state.
`updateLED(int)`	Controls two 74HC595 shift registers to turn on the LED corresponding to the pressed key, clears others.
`clearAllLEDs()`	Turns off all LEDs by sending zeros to the shift registers.
`lcd_init_custom()`	Sends initialization commands to the LCD over I2C to configure 4-bit mode, display on, cursor off, etc.
`lcd_command(uint8_t)`	Sends a command byte to the LCD to control cursor, clear display, or other instructions.
`lcd_data(uint8_t)`	Sends a data byte (character) to the LCD to be displayed at the current cursor position.
`lcd_send(uint8_t, uint8_t)`	Sends one byte to the LCD split into two 4-bit nibbles with control bits (command or data).
`lcd_write4bits(uint8_t)`	Sends 4 bits to the LCD via I2C, including backlight and enable signal sequencing for proper timing.
`lcd_clear_custom()`	Clears the LCD display and delays for command execution.
`lcd_setCursor_custom(int, int)`	Sets the cursor position on the LCD based on row and column arguments.
`lcd_print_custom(const char*)`	Prints a null-terminated string character by character on the LCD.
`i2c_init()`	Initializes I2C hardware registers for communication at approximately 100 kHz clock speed.
`i2c_start(uint8_t)`	Generates an I2C start condition and sends the device address, waits for acknowledgment.
`i2c_write(uint8_t)`	Writes one byte of data on the I2C bus and waits for acknowledgment from the slave device.

`i2c_stop()`	Sends an I2C stop condition to end communication.
`ISR(TIMER1_COMPA_vect)`	Timer1 Compare Interrupt Service Routine that multiplexes rows of the button matrix by cycling active row pins.
`ISR(PCINT2_vect)`	Pin Change Interrupt Service Routine for Port D columns that signals when keypad state has changed.
`ISR(PCINT0_vect)`	Pin Change Interrupt Service Routine for Port B columns that signals when keypad state has changed.

Program Flow:

* **Setup Phase:**

- Initialize serial communication for debugging.
- Initialize I2C communication registers manually.
- Initialize the LCD display by sending low-level commands over I2C.
- Set initial LCD message to prompt user (“Apasa o nota:”).
- Configure timer to run in CTC mode, generating interrupts to multiplex button matrix rows.
- Enable pin change interrupts on column inputs to detect button presses immediately.
- Configure pins controlling the shift registers (for LEDs) and buzzer.
- Clear all LEDs and set initial states.

* **Interrupt Service Routines (ISRs):**

- **Timer1 Compare Match ISR:**
  1. Cycles through the rows of the button matrix by setting the previous row HIGH (inactive) and the next row LOW (active).
  2. This multiplexing allows scanning one row at a time.
- **Pin Change Interrupts (PCINT0 and PCINT2):**
  1. Triggered on any change on the column input pins.
  2. Sets a flag (`keypadChanged`) to indicate that a keypad scan should be performed.

* **Main Loop (`loop`):**

- If the `keypadChanged` flag is set by the ISR:
  1. Perform a keypad scan to detect button presses/releases.
  2. Clear the `keypadChanged` flag.
  3. Record the time of this scan.
- Otherwise, if a certain time (`scanInterval`) has passed since the last scan:
  1. Perform a periodic keypad scan to ensure no button press is missed.

* **Keypad Scanning (`scanKeypad`):**

- For each row:
  1. The active row is LOW (due to Timer1 ISR multiplexing).
  2. Read the column input pins directly from hardware registers.
  3. Detect if any button in the active row is pressed by checking which column reads LOW.
- Calculate the index of the pressed button based on the row and column.
- Implement debouncing by checking if the detected button remains stable for a debounce period (`buttonStableDelay`).
- If a stable button press is detected:
  1. Update the LCD to display the pressed note.
  2. Start playing the corresponding tone on the buzzer.

3. Light up the corresponding LED by updating the shift registers.
4. Log the pressed button and frequency on the serial monitor.

- If no button is pressed (release detected):
  1. Stop the buzzer.
  2. Turn off all LEDs.
  3. Reset the LCD message to the prompt.

* **LED Update (`updateLED`):**

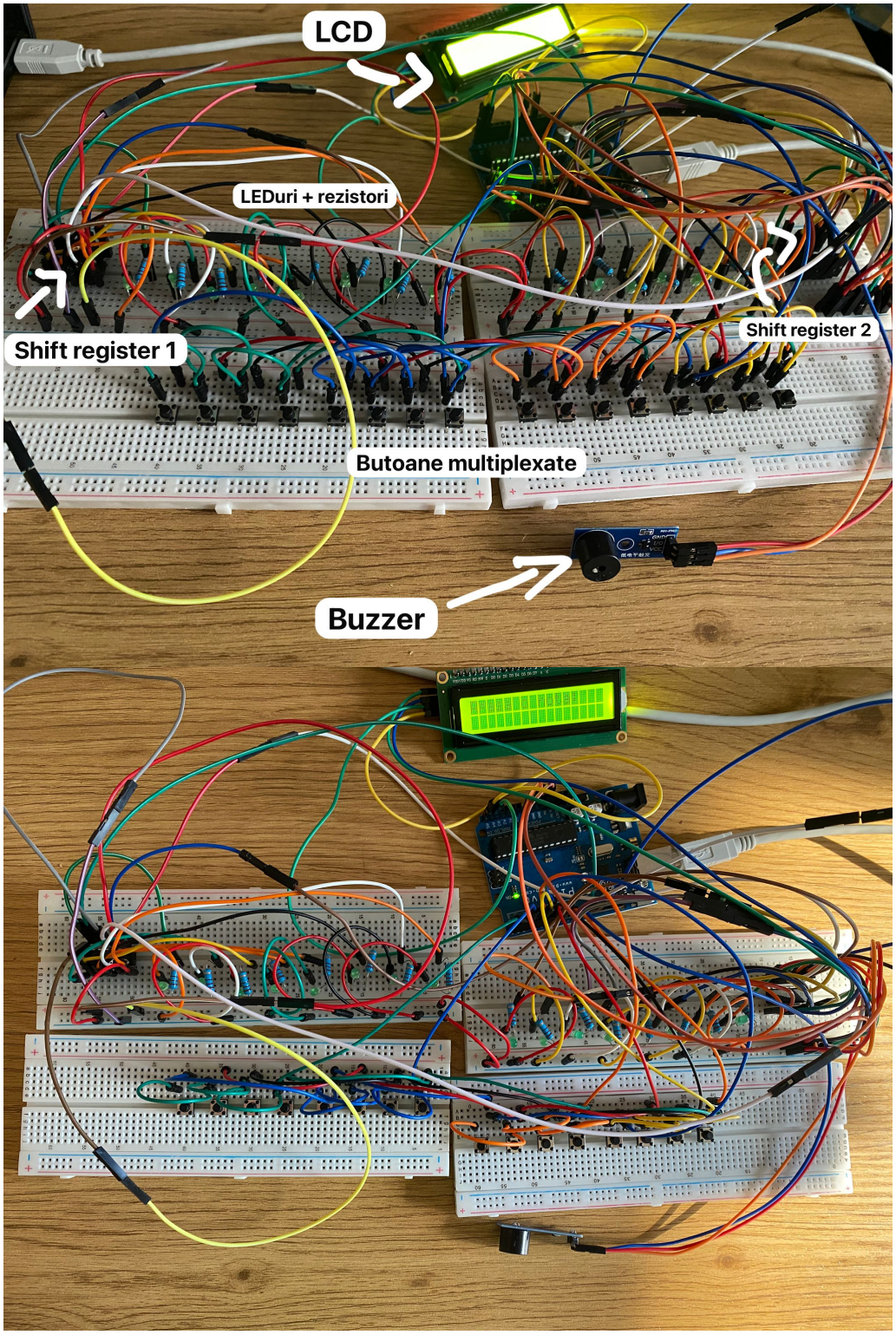
- Generate a 16-bit pattern with only the bit corresponding to the pressed button set.
- Temporarily disable interrupts to avoid glitches during data shifting.
- Shift out the pattern into two chained 74HC595 shift registers (high byte first, then low byte).
- Re-enable interrupts.

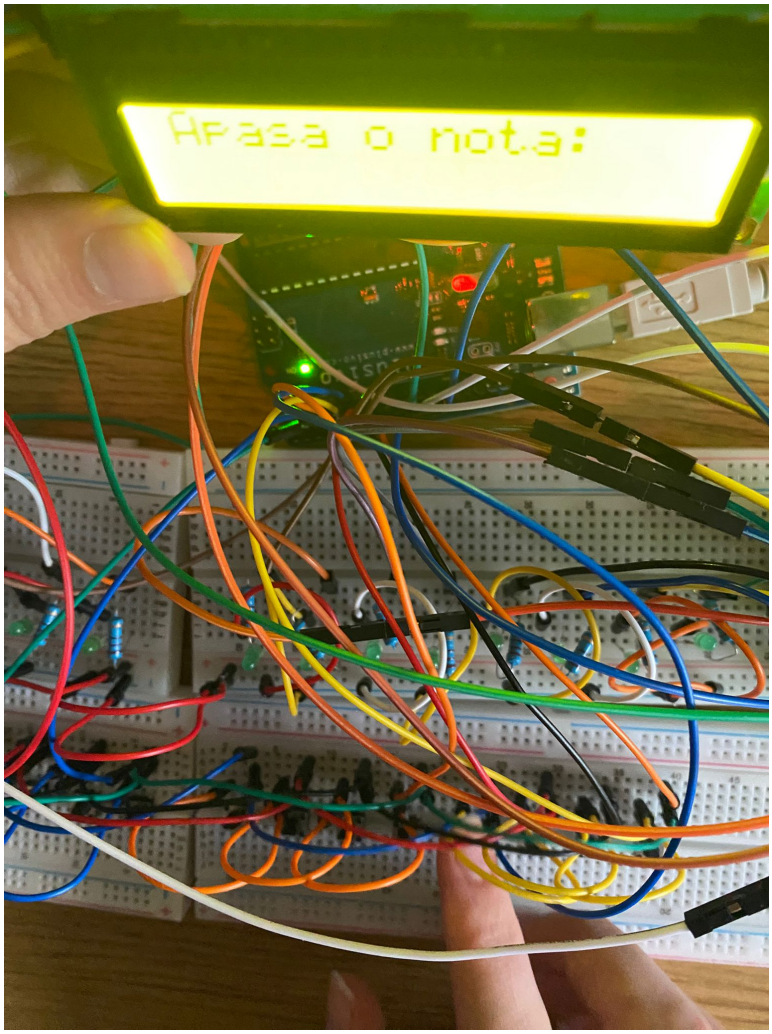
* **Debounce Mechanism:**

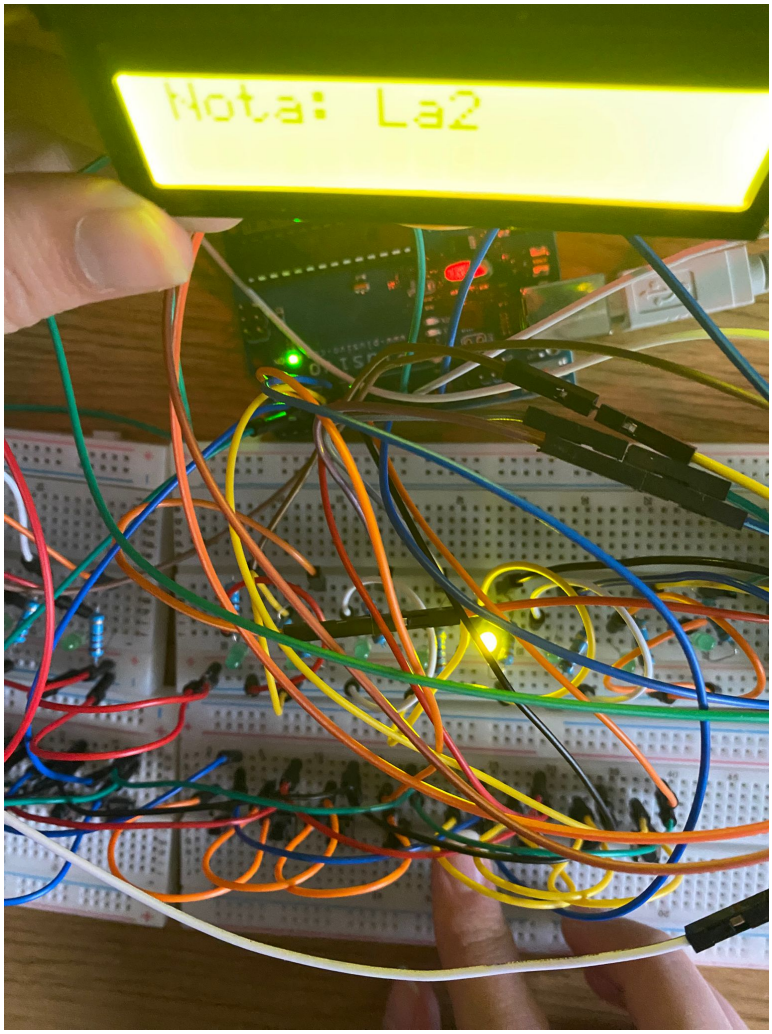
- Detects changes in button state.
- Resets a timer whenever a change is detected.
- Only confirms a button state change if stable for at least 50 milliseconds.
- Prevents false triggering caused by mechanical switch bounce.

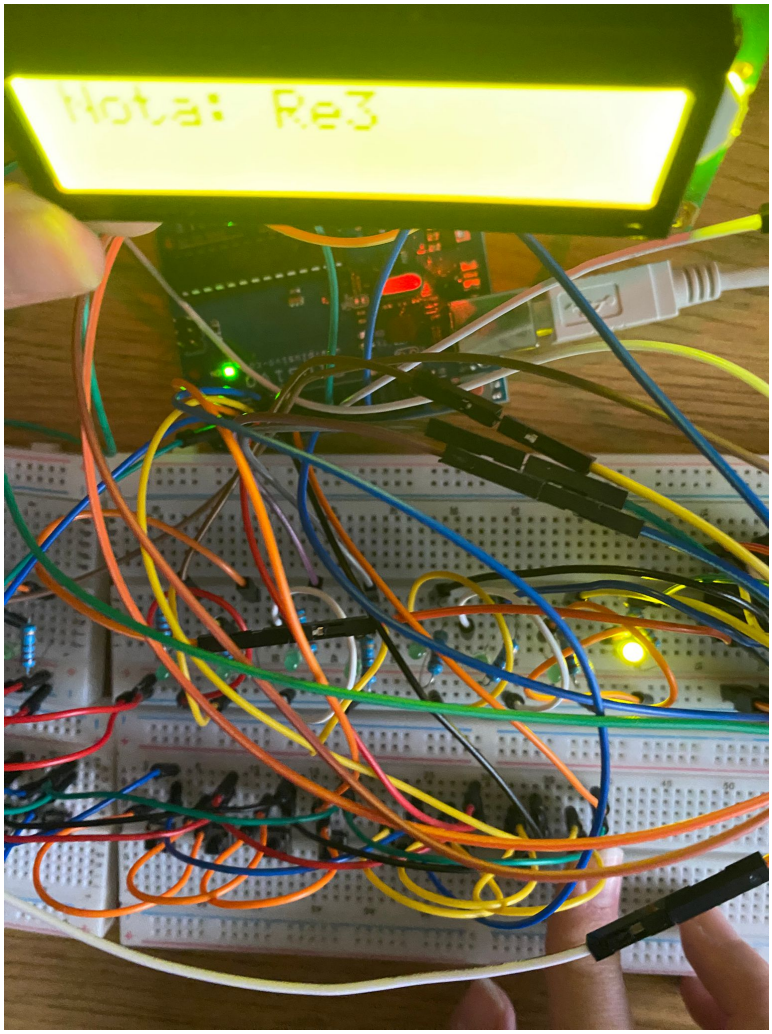
Results

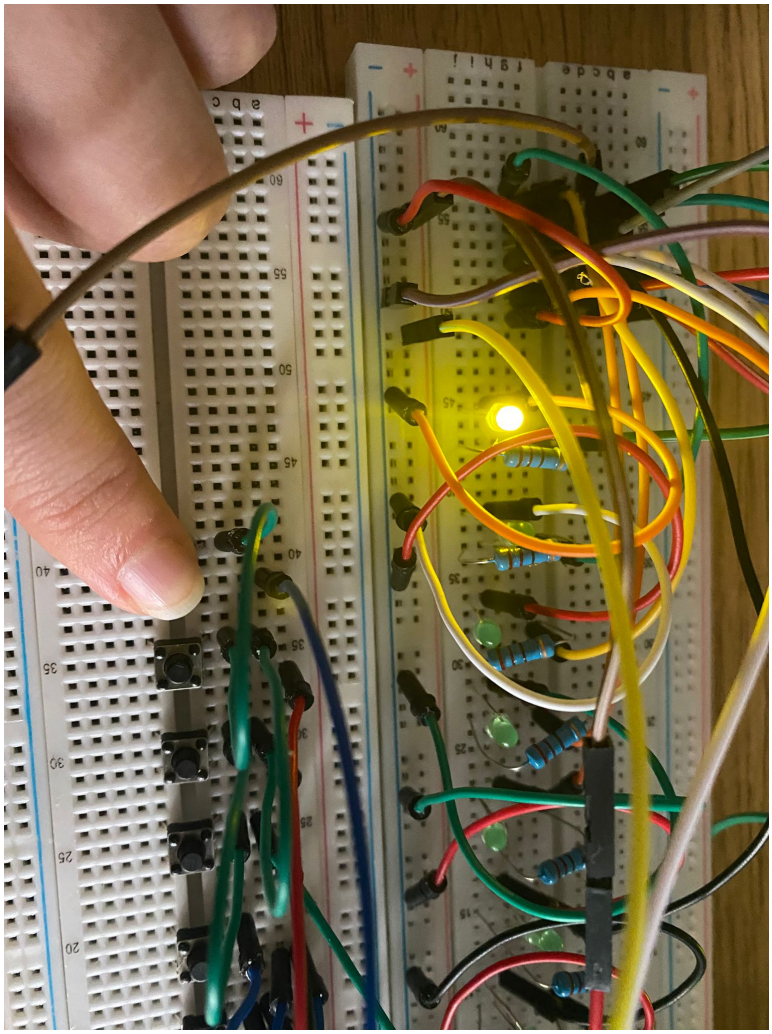
- All 16 keys are correctly detected using matrix scanning.
- The buzzer successfully plays a unique tone for each key.
- Corresponding LEDs light up with no visible flickering due to multiplexing.
- The LCD outputs the correct note that is played.
- The Arduino handles real-time input/output tasks with stable performance.


LCD
LEDuri + rezistori
Shift register 1
Shift register 2
Butoane multiplexate
Buzzer


Apasa o nota:


Nota: La2


Nota: Re3



Conclusions

This project taught me a lot about working efficiently with limited Arduino pins using multiplexing and shift registers. Implementing I2C manually for the LCD really helped me understand how communication protocols work at a low level. I also learned why software debouncing is crucial for reliable button inputs. Overall, building this from scratch boosted my skills in embedded systems and gave me more confidence in handling hardware and software together without relying on external libraries.

Logbook

- Week 1: Chose project idea and defined functionality
- Week 2: Hardware design
- Week 3: Software design

Bibliography/ Resources

Project Repo:

- https://github.com/marafichios/PianoBit/tree/main

Websites used for buying the components:

- https://www.optimusdigital.ro/ro/?gad_source=1&gad_campaignid=20864846564&gbraid=0AAAADv-p3AcpG7M_HbgPQ0KQ4uBq-Ehv&gclid=CjwKCAjw24vBBhABEiwANFG7y3y28oexWP6opxU_ynoO9b2NToMi9NKPKU70vFu9IwIw7i5ApsXHoxoCl2QQAvD_BwE
- https://www.emag.ro/

Bibliography:

- https://docs.onion.io/omega2-maker-kit/starter-kit-using-shift-register.html
- https://docs.arduino.cc/tutorials/communication/guide-to-shift-out/

Export to PDF