Jacobs University Bremen

Introduction to Intelligent Mobile Systems Lab I
Course: CH09-320113

Musical Keyboard with simple commands

Albrit Bendo
Betelhem Nebebe
Lorenzo Rota
Sara Azeddine

Instructor
Dr. Fangning Hu

November 9th, 2018

1 Introduction

In this project, we created a musical keyboard which works as follows. First, we created the circuit which consists of 7 tone buttons and 4 functional buttons. Each of the 7 buttons has its corresponding sound that we determined (the "piano" notes), while the other 4 buttons are to *record*, *reset*, *play/pause* and *restart* the melody when requested. We used an Active Buzzer , which is also connected to a specific pin, to transmit the sound and a LED light which lights up when we record or play the tones.

2 Usage description

What we created is a simple piano. The user just presses one of the 7 buttons of the left side of the breadboard to play a melody (the notes are C, D, E, F, G, A and B from left to right). Sounds are specified in the code and each button will play a different sound. If the user wants to record the sound he/she presses the first button starting from the 4 buttons (from left to right) and then presses the keys. If the user wants to reset (delete and record other notes) he/she presses the 2nd button. If he/she wants to play or pause the sound that he/she recorded he/she presses the 3rd one and if he/she wants to restart playing from scratch he/she presses the 4th button.

3 Theory

The Arduino has a handy `tone()` function which can be used to generate varying frequency signals in order to produce different sounds using a buzzer. It is nothing but a crystal which converts mechanical vibrations into electricity or vice versa. Here we apply a variable current (frequency) for which the crystal vibrates and therefore produces sound. Hence, in order to let the buzzer make some noise we have to make the Buzzer's electric crystal vibrate: the pitch and tone of the noise depend on how fast the crystal vibrates. Hence, the tone and pitch can be controlled by varying the frequency of the current. The tone function can generate a particular frequency on a specific pin. Our project has only 7 push buttons for the notes, so each button can play only one particular musical note and thus we can play only 7 notes in total. The notes selected in this project are the notes C, D, E, F, G , A and B, which can be played using the buttons 1 to 7 respectively. Moreover, we also have 4 push buttons dedicated to the commands listed above in the introduction.

4 Materials

In this project, we created a musical keyboard using the following components

- One Arduino Uno
- A Breadboard
- An Active Buzzer
- USB Cable
- Jumper wires
- 11 Push Button switches
- LED

5 Circuit design

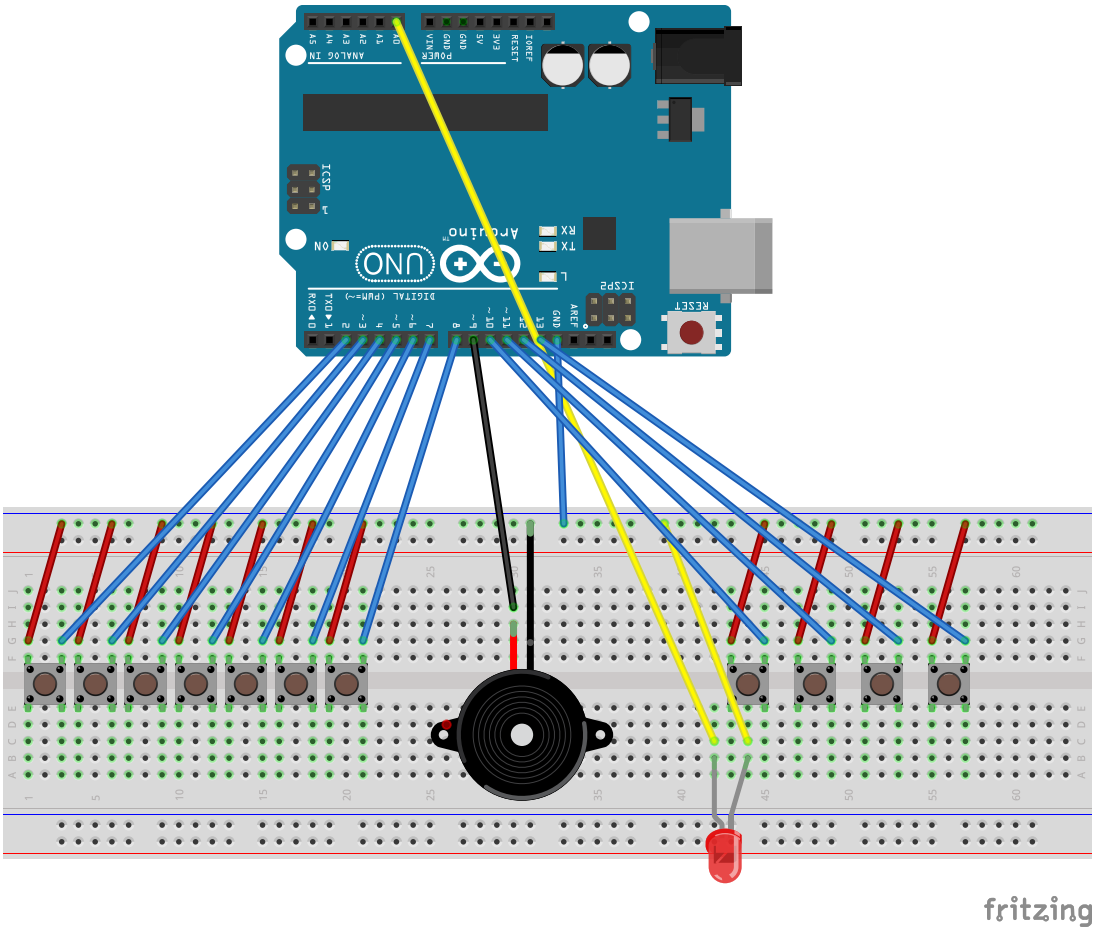

Figure 1: Circuit diagram

6 Arduino source code

```
#include "pitches.h"

// Buzzer pin
const int BUZZER_PIN  = 9;

// Key button pins
const int C = 2; // Button 1
const int D = 3; // Button 2
const int E = 4; // Button 3
const int F = 5; // Button 4
const int G = 6; // Button 5
const int A = 7; // Button 6
const int B = 8; // Button 7

// Modulation pins
const int RECORD_PIN = 10;
const int RESET_PIN = 11;
const int LOAD_PIN = 12;
const int STOP_PIN = 13;

// Indicator pin
const int LED_PIN = A0;

// Modulation memory
int counter = 0;
bool is_recording = false;
bool is_playing = false;
bool is_reset = false;

// Melody size and note duration
const int melody_size = 200;
int duration(50); // Every note lasts 50 milliseconds.

```

```
// Melody and individual notes arrays
int melody[melody_size] = {N_REST}; // Initialize empty melody
int c[] = {N_C5}; // Plays C Note
int d[] = {N_D5}; // Plays D Note
int e[] = {N_E5}; // Plays E Note
int f[] = {N_F5}; // Plays F Note
int g[] = {N_G5}; // Plays G Note
int a[] = {N_A5}; // Plays A Note
int b[] = {N_B5}; // Plays B Note

void setup() {
  Serial.begin (9600);
  //Set every button as an INPUT_PULLUP
  pinMode(C, INPUT_PULLUP);
  pinMode(D, INPUT_PULLUP);
  pinMode(E, INPUT_PULLUP);
  pinMode(F, INPUT_PULLUP);
  pinMode(G, INPUT_PULLUP);
  pinMode(A, INPUT_PULLUP);
  pinMode(B, INPUT_PULLUP);
  pinMode(RECORD_PIN, INPUT_PULLUP);
  pinMode(RESET_PIN, INPUT_PULLUP);
  pinMode(LOAD_PIN, INPUT_PULLUP);
  pinMode(STOP_PIN, INPUT_PULLUP);
}

void loop() {
  // Modulation signals
  // Recording button pressed
  if (digitalRead(RECORD_PIN) == LOW) {
    Serial.println("Recording started");
    counter = 0;
    is_recording = true;
    is_playing = false;
    delay(200);
    analogWrite(LED_PIN, 1023); // Turn on LED
  }
  // Reset button pressed
  else if (digitalRead(RESET_PIN) == LOW) {
    Serial.println("Reset started");
    counter = 0;
    is_playing = false;
    is_recording = false;
    is_reset = true;
    delay(200);
    analogWrite(LED_PIN, 1023); // Turn on LED
  }
  // Play button pressed while not playing
  else if (digitalRead(LOAD_PIN) == LOW && is_playing == false) {
    Serial.println("Playing started");
    is_playing = true;
    is_recording = false;
    delay(200);
    analogWrite(LED_PIN, 1023); // Turn on LED
  }
  // Play button pressed while playing
  else if (digitalRead(LOAD_PIN) == LOW && is_playing == true) {
    Serial.println("Playing pauzed");
    is_playing = false;
    is_recording = false;
    delay(200);
    analogWrite(LED_PIN, 0); // Turn off LED
  }
  // Stop button pressed
  else if (digitalRead(STOP_PIN) == LOW) {
    Serial.println("Playing stopped");
    counter = 0;
    is_playing = false;
    delay(200);
    analogWrite(LED_PIN, 0); // Turn off LED
  }
  event_handler();
}


void event_handler() {
  // Execute when program is in the reset state
  if (is_reset == true) {
    melody[counter] = N_REST;
    counter++;
    if (counter == melody_size - 1) {
      Serial.println("Reset finished");
      is_reset = false;
```

```
      counter = 0;
      analogWrite(LED_PIN, 0); // Turn off LED
    }
  }
  // Execute when the program is in the recording state
  if (is_recording == true) {
    if (digitalRead(C) == LOW) {
      tone(BUZZER_PIN, c[0], duration);
      melody[counter] = N_C5;
      Serial.println("Note: C");
    }
    else if (digitalRead(D) == LOW) {
      tone(BUZZER_PIN, d[0], duration);
      melody[counter] = N_D5;
      Serial.println("Note: D");
    }
    else if (digitalRead(E) == LOW) {
      tone(BUZZER_PIN, e[0], duration);
      melody[counter] = N_E5;
      Serial.println("Note: E");
    }
    else if (digitalRead(F) == LOW) {
      tone(BUZZER_PIN, f[0], duration);
      melody[counter] = N_F5;
      Serial.println("Note: F");
    }
    else if (digitalRead(G) == LOW) {
      tone(BUZZER_PIN, g[0], duration);
      melody[counter] = N_G5;
      Serial.println("Note: G");
    }
    else if (digitalRead(A) == LOW) {
      tone(BUZZER_PIN, a[0], duration);
      melody[counter] = N_A5;
      Serial.println("Note: A");
    }
    else if (digitalRead(B) == LOW) {
      tone(BUZZER_PIN, b[0], duration);
      melody[counter] = N_B5;
      Serial.println("Note: B");
    } else {
      melody[counter] = N_REST;
    }
    delay(duration);
    if (counter == melody_size - 1) {
      Serial.println("Recording stopped");
      is_recording = false;
      counter = 0;
      analogWrite(LED_PIN, 0); // Turn off LED
    }
    counter++;
  }
  // Execute when the program is in the play state
  if (is_playing == true) {
    tone(BUZZER_PIN, melody[counter], duration);
    switch (melody[counter]) {
      case N_C5:
        Serial.println("Note: C");
        break;
      case N_D5:
        Serial.println("Note: D");
        break;
      case N_E5:
        Serial.println("Note: E");
        break;
      case N_F5:
        Serial.println("Note: F");
        break;
      case N_G5:
        Serial.println("Note: G");
        break;
      case N_A5:
        Serial.println("Note: A");
        break;
      case N_B5:
        Serial.println("Note: B");
        break;
    }
    delay(50);
    if (counter == melody_size - 1) {
      // uncomment this to have no loop
      //is_playing = false;
      Serial.println("Playing repeat");
```

```
200         counter = 0;
201       }
202       counter++;
203     }
204 }
```

Listing 1: keyboard/keyboard.ino

The code consists of four main components: The variable initialisations, which consists of the pin locations for the keyboard keys, modulation buttons (the record, reset, play/pause, stop actions) and the LED indicator, boolean variables to store the state for the event handler, and arrays for storing the melody and individual keys. The second component is the `void setup()` function, which is used to instantiate the Serial connection between the Arduino and the computer, as well as initialise the pinModes for the different button switches. The reason that `INPUT_PULLUP` was used to define the `pinMode` as opposed to `INPUT`, is because the internal pull-up resistors ensures a defined voltage. The third component is the `void loop()` function, where the modulation states are stored, based on whether LOW voltage is read from the respective button switches. A delay of 200ms is included after each reading to ensure that a single button press is only read once. The `event_handler()` function is called at the end of each loop iteration, which makes a procedure call to the last function called `void event_handler()`. In this section, the respective modulations are executed when the program is in the respective state. In the reset state, the melody array gets overridden with `N_REST` frequency values, to produce zero sound, whereas in the recording state, the keyboard key values are read into the array. In the playing state, the melody tones are read from the array, and produced through the `tone()` function. For each of the modulations, and input readings, a short output is written to the Serial monitor, so that the user can debug or write program extensions for the keyboard.

7 Contribution of group members

Albrit Bendo: Built the circuit.
Betelhem Nebebe: Lab report.
Lorenzo Rota: Code writing.
Sara Azeddine: Lab report.

8 Appendix

```
/*********************************************
 * Public Constants
 *********************************************/
#define N_REST  0
#define N_B0  31
#define N_C1  33
#define N_CS1 35
#define N_D1  37
#define N_DS1 39
#define N_E1  41
#define N_F1  44
#define N_FS1 46
#define N_G1  49
#define N_GS1 52
#define N_A1  55
#define N_AS1 58
#define N_B1  62
#define N_C2  65
#define N_CS2 69
#define N_D2  73
#define N_DS2 78
#define N_E2  82
#define N_F2  87
#define N_FS2 93
#define N_G2  98
#define N_GS2 104
#define N_A2  110
#define N_AS2 117
#define N_B2  123
#define N_C3  131
#define N_CS3 139
#define N_D3  147
#define N_DS3 156
#define N_E3  165
#define N_F3  175
#define N_FS3 185
#define N_G3  196
#define N_GS3 208
#define N_A3  220
#define N_AS3 233
#define N_B3  247
#define N_C4  262
#define N_CS4 277
#define N_D4  294
#define N_DS4 311
#define N_E4  330
#define N_F4  349
#define N_FS4 370
#define N_G4  392
#define N_GS4 415
#define N_A4  440
#define N_AS4 466
#define N_B4  494
#define N_C5  523
#define N_CS5 554
#define N_D5  587
#define N_DS5 622
#define N_E5  659
#define N_F5  698
#define N_FS5 740
#define N_G5  784
#define N_GS5 831
#define N_A5  880
#define N_AS5 932
#define N_B5  988
#define N_C6  1047
#define N_CS6 1109
#define N_D6  1175
#define N_DS6 1245
#define N_E6  1319
#define N_F6  1397
#define N_FS6 1480
#define N_G6  1568
#define N_GS6 1661
#define N_A6  1760
#define N_AS6 1865
#define N_B6  1976
#define N_C7  2093
#define N_CS7 2217
#define N_D7  2349
```

```
#define N_DS7  2489
#define N_E7   2637
#define N_F7   2794
#define N_FS7  2960
#define N_G7   3136
#define N_GS7  3322
#define N_A7   3520
#define N_AS7  3729
#define N_B7   3951
#define N_C8   4186
#define N_CS8  4435
#define N_D8   4699
#define N_DS8  4978
```

Listing 2: keyboard/pithces.h