TinyGrand® Piano Operator's Manual

Josh Winton

Fall 2019

Contents

0.1 Welcome!

This is the operator's manual for the TinyGrand® piano created by Josh Winton as his final project for the Embedded Systems and Microprocessors course at Hunter College during the Fall 2019 semester. It includes a general description of the project, a list of parts, steps to assemble the instrument, and the source code that controls it. This document also includes an explanation of the goals set at the conception of the project as well as whether they were achieved. Enjoy your new TinyGrand®!

1 Nuts and Bolts

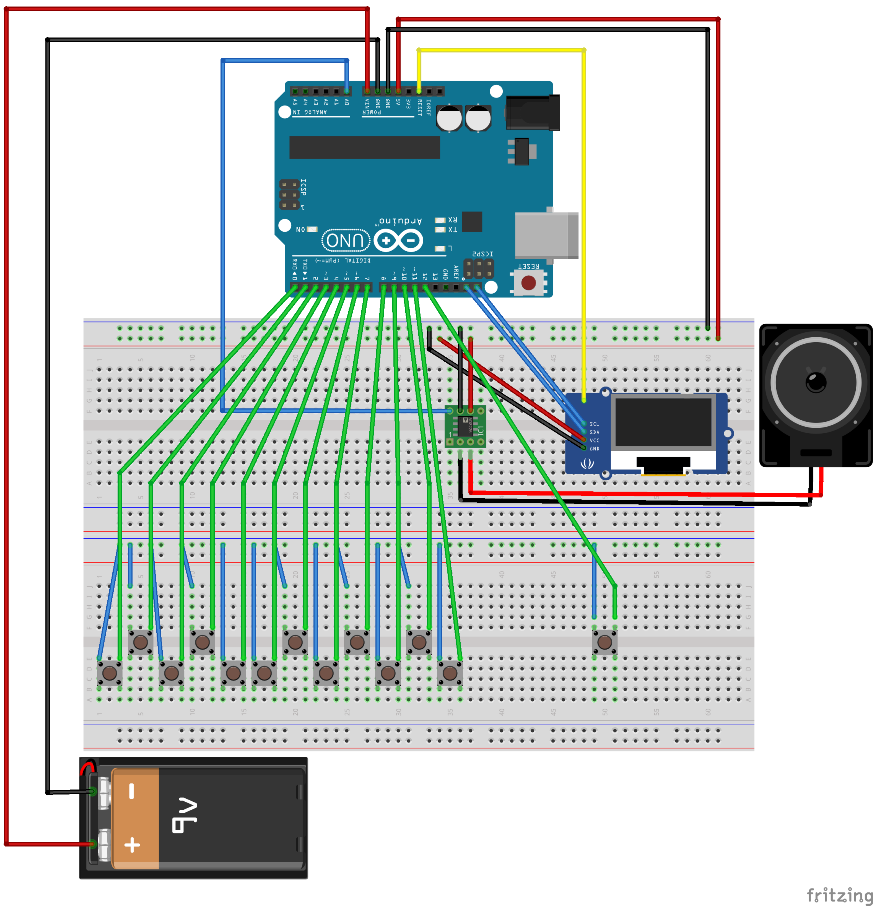
1.1 List of Parts

Here's a itemized list of the parts included with the TinyGrand$^{®}$ piano, including links to where to purchase them if you wanted to build your own:

- **Arduino Uno**
  `https://store.arduino.cc/usa/arduino-uno-rev3`
  This is the heart of the instrument. It takes care of everything from reading to the buttons, to generating the sounds, to controlling the display.
- **Buttons (13)**
  (`https://www.adafruit.com/product/1119`)
  These are the interface that you will use to control the Arduino: 12 are for playing notes, the 13th is for changing modes.
- **Adafruit Stereo Enclosed Speaker Set (3W 4 Ohm)**
  `https://www.adafruit.com/product/1669`
  This is how you'll get sound out of the TinyGrand.
- **Adafruit Stereo I2C Class D Audio Amplifier (2.8W)**
  `https://www.adafruit.com/product/1712`
  Used to power the speaker.
- **Adafruit Monochrome 1.3" 128x64 OLED graphic display I2C**
  `https://www.adafruit.com/product/938`
  Gives the artist feedback on what mode the instrument is in and what note is currently being played.
- **Adafruit 9V battery holder with switch**
  `https://www.adafruit.com/product/67`
  Allows you to take your TinyGrand on the go.
- **Breadboard (4)**
  These are the glue that hold the instrument together, the soundboard, if you will.

1.2 Schematic

Once you've waited weeks to receive all of the parts you need to construct your TinyGrand, you can begin to assemble it. This detailed schematic should facilitate this process. Having a soldering iron handy will help when it comes to putting together individual components.


Arduino
UNO
ANALOG IN
POWER
DIGITAL (PWM~)
ICSP
ICSP2
RESET
ON
TX
RX
L
SCL
SDA
VCC
GND
9V
−
+
fritzing

1.3 Operation

1.3.1 Turning on the TinyGrand

Once you've assembled all of the parts, it's time to test out your new instrument. To do this, simply connect the Arduino to power and the instrument will automatically begin its startup sequence. If you see the words "Tiny-Grand" on the display and hear a melody, it means everything has been initialized properly. To turn the instrument off, disconnect it from power. The battery in the device lasts for about 1.5 hours.

1.3.2 Playing a Tune

Once the initialization sequence is completed, you can begin to play compositions on the instrument. The instrument starts up in *Live* mode, which allows you play music as you would on a traditional keyboard. As you play, the current note is displayed.

1.3.3 Recording a Melody

To begin record a melody, switch the device to *Record* mode by pressing the **Mode** button. After you press the button, you'll see the word "Record" briefly appear on the screen. Operation in *Record* mode is essentially the same as in *Live* mode, except that your melody is being stored to be played back later.

1.3.4 Playing a Melody Back

After you've finished recording a melody, you can play if back to hear how it sounds. To do this, you must first record a melody in *Record* mode. Pressing the **Mode** button while in *Record* mode switches the device to *Play* mode, which plays back your song from memory. This mode will also be activated if your melody reaches the maximum size, 100 notes. Once a melody has been played back, it will be cleared from memory to make room for another and the device will reenter *Live* mode.

2 The Source

Here is the source code for the program that runs on the Arduino. I've also included links to the libraries that I used.

2.1 Primary Code

```
// Tone library for playing notes
#include <Tone.h>

// Serial Peripheral Interface - needed for I2C
#include <SPI.h>

// I2C - Used to communicate with OLED display
#include <Wire.h>

// Used to send graphics to display
#include <Adafruit_GFX.h>

// Display-specific library
#include <Adafruit_SSD1306.h>

// Amp-specific library
#include <Adafruit_TPA2016.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin, shared with Arduino reset pin
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Declaration for TPA2016 amp
Adafruit_TPA2016 audioamp = Adafruit_TPA2016();

// Initialize a tone
Tone tone1;

// Set input pins (digital 0-13)
const int buttonPinC =  0;
const int buttonPinCS = 1;
const int buttonPinD =  2;
const int buttonPinDS = 3;
const int buttonPinE =  4;
const int buttonPinF =  5;
const int buttonPinFS = 6;
const int buttonPinG =  7;
const int buttonPinGS = 8;
const int buttonPinA =  9;
const int buttonPinAS = 10;
const int buttonPinB =  11;
const int recordPin =   12;

// Initialize button states to be unpressed
int buttonStateC =      0;
int buttonStateCS =     0;
int buttonStateD =      0;
int buttonStateDS =     0;
int buttonStateE =      0;
int buttonStateF =      0;
int buttonStateFS =     0;
int buttonStateG =      0;
int buttonStateGS =     0;
int buttonStateA =      0;
int buttonStateAS =     0;
int buttonStateB =      0;
int recordState =       0;

// 0 is live, 1 is record, 2 is play
int MODE = 0;

// Keep track of current note while recording
int currentRecordNote = 0;

// Maximum number of notes stored for playback
const int NUM_NOTES = 100;

// Array for  storing recorded melodies
int melody[NUM_NOTES];

// Routine for clearing melody storage, empty notes are encoded as "-1"
void clearMelody(){
  for(int i = 0; i < NUM_NOTES; i++){
    melody[i] = -1;
  }
}

void setup() {
  Serial.begin(9600);

  // Use analog pin 0 for output to amp
  tone1.begin(A0);

  // Set display to horizontal rotation
  display.setRotation(2);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
```

```
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  display.setTextSize(2);      // Font size
  display.setTextColor(SSD1306_WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner

  // Set all buttons to pullup input mode
  pinMode(buttonPinC, INPUT_PULLUP);
  pinMode(buttonPinCS, INPUT_PULLUP);
  pinMode(buttonPinD, INPUT_PULLUP);
  pinMode(buttonPinDS, INPUT_PULLUP);
  pinMode(buttonPinE, INPUT_PULLUP);
  pinMode(buttonPinF, INPUT_PULLUP);
  pinMode(buttonPinFS, INPUT_PULLUP);
  pinMode(buttonPinG, INPUT_PULLUP);
  pinMode(buttonPinGS, INPUT_PULLUP);
  pinMode(buttonPinA, INPUT_PULLUP);
  pinMode(buttonPinAS, INPUT_PULLUP);
  pinMode(buttonPinB , INPUT_PULLUP);
  pinMode(recordPin, INPUT_PULLUP);

  // Initialize melody array
  clearMelody();

  // Show initial display buffer contents on the screen
  display.clearDisplay();
  display.display();
  display.println(F("TinyGrand"));
  display.display();
  display.setTextSize(3.5);      // Font size
  delay(2000); // Pause for 2 seconds
}

// Routine for reading all buttons
void readPins(){
  buttonStateC  = digitalRead(buttonPinC);
  buttonStateCS = digitalRead(buttonPinCS);
  buttonStateD  = digitalRead(buttonPinD);
  buttonStateDS = digitalRead(buttonPinDS);
  buttonStateE  = digitalRead(buttonPinE);
  buttonStateF  = digitalRead(buttonPinF);
  buttonStateFS = digitalRead(buttonPinFS);
  buttonStateG  = digitalRead(buttonPinG);
  buttonStateGS = digitalRead(buttonPinGS);
  buttonStateA  = digitalRead(buttonPinA);
  buttonStateAS = digitalRead(buttonPinAS);
  buttonStateB  = digitalRead(buttonPinB);
  recordState   = digitalRead(recordPin);
}

// Routine that controls LIVE mode
void live(){
  //  Indicate mode change on display
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println(F("LIVE"));
  display.display();

  delay(500);

  display.clearDisplay();
  display.display();

  // Loop runs while device is in LIVE mode
  while(true){
    // switch to RECORD if button has been pushed
    if(MODE == 1){
      record();
    }

    display.clearDisplay();
    display.setCursor(0, 0);
    readPins();

    /* Check if the pushbutton is pressed. If it is, the buttonState is LOW:
       play note and display it */
    if (buttonStateC == LOW) {
      tone1.play(NOTE_C3);
      display.println(F("C"));
      display.display();
    } else if(buttonStateCS == LOW){
      tone1.play(NOTE_CS3);
      display.println(F("C#"));
      display.display();
    } else if(buttonStateD == LOW){
      tone1.play(NOTE_D3);
      display.println(F("D"));
      display.display();
    } else if(buttonStateDS == LOW){
      tone1.play(NOTE_DS3);
      display.println(F("D#"));
      display.display();
    } else if(buttonStateE == LOW){
      tone1.play(NOTE_E3);
      display.println(F("E"));
      display.display();
    } else if(buttonStateF == LOW){
      tone1.play(NOTE_F3);
      display.println(F("F"));
      display.display();
    } else if(buttonStateFS == LOW){
      tone1.play(NOTE_FS3);
      display.println(F("F#"));
      display.display();
    } else if(buttonStateG == LOW){
      tone1.play(NOTE_G3);
      display.println(F("G"));
      display.display();
    } else if(buttonStateGS == LOW){
      tone1.play(NOTE_GS3);
      display.println(F("G#"));
      display.display();
    } else if(buttonStateA == LOW){
      tone1.play(NOTE_A3);
      display.println(F("A"));
      display.display();
    } else if(buttonStateAS == LOW){
      tone1.play(NOTE_AS3);
      display.println(F("A#"));
      display.display();
    } else if(buttonStateB == LOW){
      tone1.play(NOTE_B3);
      display.println(F("B"));
      display.display();
    } else if(recordState == LOW){
      MODE = 1;
    } else {
      // turn LED off:
      tone1.stop();
      display.println(F("OFF"));
      display.display();
    }
  }
}

void record(){
  clearMelody();
  currentRecordNote = 0;

  // Indicate mode change on display
  display.clearDisplay();
  display.println(F("RECORD"));
  display.display();
  delay(500);

  display.clearDisplay();
  display.display();

  while(true){
    display.clearDisplay();
    display.setCursor(0, 0);
    readPins();

    /* Check if the pushbutton is pressed. If it is, the buttonState is LOW:
       play note and display it */
    if (buttonStateC == LOW) {
      tone1.play(NOTE_C3);
      display.println(F("C"));
      display.display();
      while(true){
        readPins();
        if(buttonStateC == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_C3;
      currentRecordNote++;
    } else if(buttonStateCS == LOW){
      tone1.play(NOTE_CS3);
      display.println(F("C#"));
      display.display();
      while(true){
        readPins();
        if(buttonStateCS == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_CS3;
      currentRecordNote++;
    } else if(buttonStateD == LOW){
      tone1.play(NOTE_D3);
      display.println(F("D"));
      display.display();
      while(true){
        readPins();
        if(buttonStateD == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_D3;
      currentRecordNote++;
    } else if(buttonStateDS == LOW){
      tone1.play(NOTE_DS3);
      display.println(F("D#"));
      display.display();
      while(true){
        readPins();
        if(buttonStateDS == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_DS3;
      currentRecordNote++;
    } else if(buttonStateE == LOW){
      tone1.play(NOTE_E3);
      display.println(F("E"));
      display.display();
      while(true){
        readPins();
        if(buttonStateE == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_E3;
      currentRecordNote++;
    } else if(buttonStateF == LOW){
      tone1.play(NOTE_F3);
      display.println(F("F"));
      display.display();
      while(true){
        readPins();
        if(buttonStateF == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_F3;
      currentRecordNote++;
    } else if(buttonStateFS == LOW){
      tone1.play(NOTE_FS3);
      display.println(F("F#"));
      display.display();
      while(true){
        readPins();
        if(buttonStateFS == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_FS3;
      currentRecordNote++;
    } else if(buttonStateG == LOW){
      tone1.play(NOTE_G3);
      display.println(F("G"));
      display.display();
      while(true){
        readPins();
        if(buttonStateG == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_G3;
      currentRecordNote++;
    } else if(buttonStateGS == LOW){
      tone1.play(NOTE_GS3);
      display.println(F("G#"));
      display.display();
      while(true){
        readPins();
        if(buttonStateGS == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_GS3;
      currentRecordNote++;
    } else if(buttonStateA == LOW){
      tone1.play(NOTE_A3);
      display.println(F("A"));
      display.display();
      while(true){
        readPins();
        if(buttonStateA == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_A3;
      currentRecordNote++;
    } else if(buttonStateAS == LOW){
      tone1.play(NOTE_AS3);
      display.println(F("A#"));
      display.display();
      while(true){
        readPins();
        if(buttonStateAS == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_AS3;
      currentRecordNote++;
    } else if(buttonStateB == LOW){
      tone1.play(NOTE_B3);
      display.println(F("B"));
      display.display();
      while(true){
        readPins();
        if(buttonStateB == HIGH){
          break;
        }
      }
      melody[currentRecordNote] = NOTE_B3;
      currentRecordNote++;
    } else if(recordState == LOW){
      play();
      MODE = 0;
      return;
    } else {
      // turn LED off:
      tone1.stop();
      display.println(F("OFF"));
      display.display();
    }
  }
}

void play(){
  display.clearDisplay();
  display.println(F("PLAY"));
  display.display();

  delay(500);

  if(melody[0] == -1){
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println(F("EMPTY"));
    display.display();
    delay(500);
    MODE = 0;
    return;
  }

  for (int thisNote = 0; thisNote < NUM_NOTES; thisNote++) {
    Serial.println(melody[thisNote]);
    // Stop playing at end of melody
    if(melody[thisNote] == -1){
      display.clearDisplay();
      display.setCursor(0, 0);
      display.println(F("End"));
      display.display();
      delay(1000);
      MODE = 0;
      return;
    }

    // Start playing note
    tone1.play(melody[thisNote]);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int noteDuration = 1000 / 4; // All 1/4 notes
    int pauseBetweenNotes = noteDuration * 0.5;

    display.clearDisplay();
    display.setCursor(0, 0);
    display.println(F("PLAYING"));
    display.display();

    delay(noteDuration);

    display.clearDisplay();
    display.display();

    // Stop the tone playing:
    tone1.stop();
    delay(pauseBetweenNotes);
  }
  MODE = 0;
  return;
}

void loop() {
  live();
}
```

2.2 Libraries

- **Tone**
  `https://github.com/bhagman/Tone`
  Used for playing notes.
- **SPI**
  `https://www.arduino.cc/en/reference/SPI`
  Used in conjunction with Wire.
- **Wire**
  `https://www.arduino.cc/en/reference/wire`
  Used for controlling OLED display.

- **Adafruit_GFX**
  https://learn.adafruit.com/adafruit-gfx-graphics-library/overview
  Used to send text to OLED display.
- **Adafruit_SSD1306**
  https://github.com/adafruit/Adafruit_SSD1306
  Drivers for OLED display.
- **Adafruit_TPA2016**
  https://github.com/adafruit/Adafruit-TPA2016-Library/blob/master/Adafruit_TPA2016.h
  Driver for amp.

3 Goals

3.1 Achieved Goals

My primary goal in developing the TinyGrand piano was to create a keyboard that allowed users to play songs live or to record them and then play them back. This general goal was achieved.

I also wanted to have an on-board display attached to the instrument that provided information about the mode of the device and the current note being played. This goal was also achieved.

3.2 Unachieved Goals

3.2.1 3D Print

One of the features of my design at the outset was to design a 3D printed enclosure to house all of the components for the project in a rigid case. Although I was able to create a nice looking prototype, creating a design that fit everything properly and then printing it ended up taking too much time to be able to finish before the deadline. Instead of 3D printing, I ended up using a cardboard enclosurre.

3.2.2 Music Staff

One of my goals if I had some extra time was to set up a GUI that allowed the user to see their notes played on a music staff. This goal wasn't achieved because the display that I received was smaller than I imagined and because writing the code to display those custom images would have taken more time than I had.

3.2.3 Edit mode

In my mind, the edit mode went hand-in-hand with the GUI, so I think it made sense not to include the edit mode on this version. Furthermore, the extra buttons to allow scrolling would have been more than the Uno could fit, which would mean upgrading to the Mega. Since achieving the primary goals took longer than expected, I left this secondary feature out.

3.2.4 Chords

Another one of my secondary goals that I didn't get around to achieving because of a lack of time was the ability to play chords and record them. Although I'm sure this wouldn't take too long if I had included it from the outset, the current implementation of reading the keys only allows one key to be played or recorded at a time. Additionally, my data structure only allows the storage of one note per time unit.