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

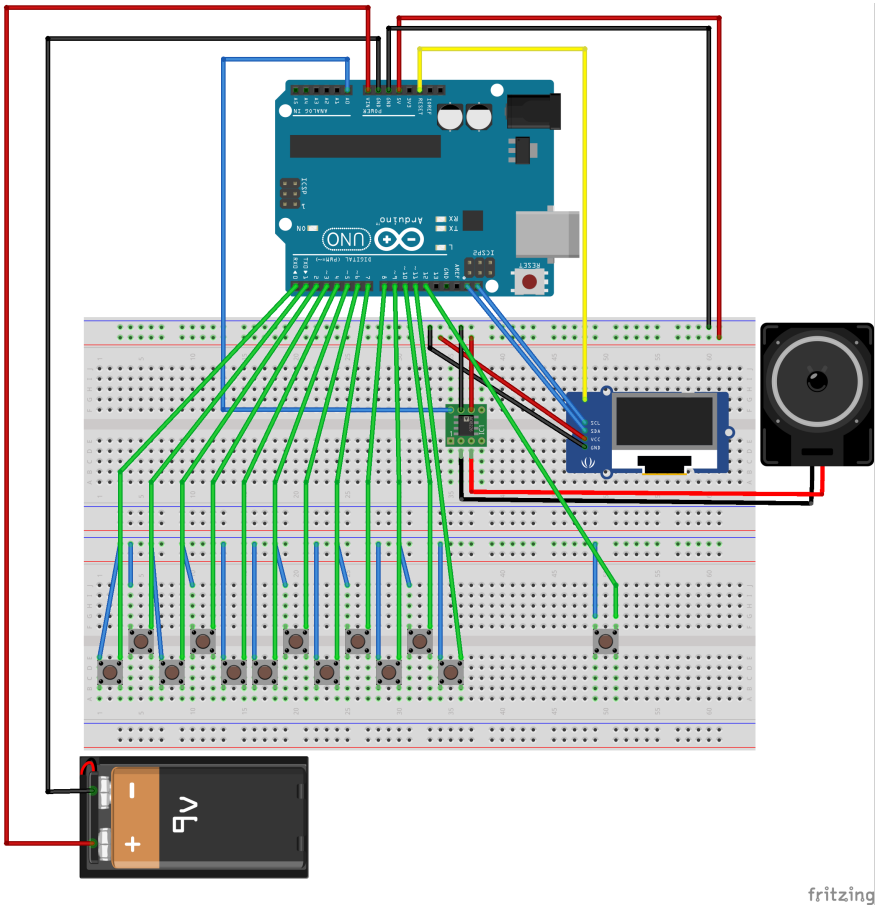
1.1 List of Parts

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.


UNO
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 "TinyGrand" 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

```
1  // Tone library for playing notes
2  #include <Tone.h>
3
4  // Serial Peripheral Interface - needed for I2C
5  #include <SPI.h>
6
7  // I2C - Used to communicate with OLED display
8  #include <Wire.h>
9
10 // Used to send graphics to display
11 #include <Adafruit_GFX.h>
12
13 // Display-specific library
14 #include <Adafruit_SSD1306.h>
15
16 // Amp-specific library
17 #include <Adafruit_TPA2016.h>
18
19 #define SCREEN_WIDTH 128 // OLED display width, in pixels
20 #define SCREEN_HEIGHT 32 // OLED display height, in
       pixels
21
22 // Declaration for an SSD1306 display connected to I2C (
       SDA, SCL pins)
23 #define OLED_RESET     -1 // Reset pin, shared with
       Arduino reset pin
24 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &
       Wire, OLED_RESET);
25
26 // Declaration for TPA2016 amp
27 Adafruit_TPA2016 audioamp = Adafruit_TPA2016();
28
29 // Initialize a tone
30 Tone tone1;
31
32 // Set input pins (digital 0-13)
33 const int buttonPinC =  0;
34 const int buttonPinCS = 1;
35 const int buttonPinD =  2;
36 const int buttonPinDS = 3;
37 const int buttonPinE =  4;
38 const int buttonPinF =  5;
```

```
39 const int buttonPinFS = 6;
40 const int buttonPinG =  7;
41 const int buttonPinGS = 8;
42 const int buttonPinA =  9;
43 const int buttonPinAS = 10;
44 const int buttonPinB =  11;
45 const int recordPin =   12;
46
47 // Initialize button states to be unpressed
48 int buttonStateC =      0;
49 int buttonStateCS =     0;
50 int buttonStateD =      0;
51 int buttonStateDS =     0;
52 int buttonStateE =      0;
53 int buttonStateF =      0;
54 int buttonStateFS =     0;
55 int buttonStateG =      0;
56 int buttonStateGS =     0;
57 int buttonStateA =      0;
58 int buttonStateAS =     0;
59 int buttonStateB =      0;
60 int recordState =       0;
61
62 // 0 is live, 1 is record, 2 is play
63 int MODE = 0;
64
65 // Keep track of current note while recording
66 int currentRecordNote = 0;
67
68 // Maximum number of notes stored for playback
69 const int NUM_NOTES = 100;
70
71 // Array for  storing recorded melodies
72 int melody[NUM_NOTES];
73
74 // Routine for clearing melody storage, empty notes are
       encoded as "-1"
75 void clearMelody(){
76   for(int i = 0; i < NUM_NOTES; i++){
77     melody[i] = -1;
78   }
79 }
80
81 void setup() {
82   Serial.begin(9600);
83
84   // Use analog pin 0 for output to amp
85   tone1.begin(A0);
86
87   // Set display to horizontal rotation
88   display.setRotation(2);
89
90   // SSD1306_SWITCHCAPVCC = generate display voltage from
         3.3V internally
91   if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { //
       Address 0x3C for 128x32
```

```
92      Serial.println(F("SSD1306 allocation failed"));
93      for(;;); // Don't proceed, loop forever
94    }
95
96    display.setTextSize(2);      // Font size
97    display.setTextColor(SSD1306_WHITE); // Draw white text
98    display.setCursor(0, 0);     // Start at top-left
          corner
99
100   // Set all buttons to pullup input mode
101   pinMode(buttonPinC, INPUT_PULLUP);
102   pinMode(buttonPinCS, INPUT_PULLUP);
103   pinMode(buttonPinD, INPUT_PULLUP);
104   pinMode(buttonPinDS, INPUT_PULLUP);
105   pinMode(buttonPinE, INPUT_PULLUP);
106   pinMode(buttonPinF, INPUT_PULLUP);
107   pinMode(buttonPinFS, INPUT_PULLUP);
108   pinMode(buttonPinG, INPUT_PULLUP);
109   pinMode(buttonPinGS, INPUT_PULLUP);
110   pinMode(buttonPinA, INPUT_PULLUP);
111   pinMode(buttonPinAS, INPUT_PULLUP);
112   pinMode(buttonPinB , INPUT_PULLUP);
113   pinMode(recordPin, INPUT_PULLUP);
114
115   // Initialize melody array
116   clearMelody();
117
118   // Show initial display buffer contents on the screen
119   display.clearDisplay();
120   display.display();
121   display.println(F("TinyGrand"));
122   display.display();
123   display.setTextSize(3.5);      // Font size
124   delay(2000); // Pause for 2 seconds
125 }
126
127 // Routine for reading all buttons
128 void readPins(){
129   buttonStateC  = digitalRead(buttonPinC);
130   buttonStateCS = digitalRead(buttonPinCS);
131   buttonStateD  = digitalRead(buttonPinD);
132   buttonStateDS = digitalRead(buttonPinDS);
133   buttonStateE  = digitalRead(buttonPinE);
134   buttonStateF  = digitalRead(buttonPinF);
135   buttonStateFS = digitalRead(buttonPinFS);
136   buttonStateG  = digitalRead(buttonPinG);
137   buttonStateGS = digitalRead(buttonPinGS);
138   buttonStateA  = digitalRead(buttonPinA);
139   buttonStateAS = digitalRead(buttonPinAS);
140   buttonStateB  = digitalRead(buttonPinB);
141   recordState   = digitalRead(recordPin);
142 }
143
144 // Routine that controls LIVE mode
145 void live(){
146   //  Indicate mode change on display
```

```
147   display.clearDisplay();
148   display.setCursor(0, 0);
149   display.println(F("LIVE"));
150   display.display();
151
152   delay(500);
153
154   display.clearDisplay();
155   display.display();
156
157   // Loop runs while device is in LIVE mode
158   while(true){
159     // switch to RECORD if button has been pushed
160     if(MODE == 1){
161       record();
162     }
163
164     display.clearDisplay();
165     display.setCursor(0, 0);
166     readPins();
167
168     /* Check if the pushbutton is pressed. If it is, the
          buttonState is LOW:
169        play note and display it */
170     if (buttonStateC == LOW) {
171       tone1.play(NOTE_C3);
172       display.println(F("C"));
173       display.display();
174     } else if(buttonStateCS == LOW){
175       tone1.play(NOTE_CS3);
176       display.println(F("C#"));
177       display.display();
178     } else if(buttonStateD == LOW){
179       tone1.play(NOTE_D3);
180       display.println(F("D"));
181       display.display();
182     } else if(buttonStateDS == LOW){
183       tone1.play(NOTE_DS3);
184       display.println(F("D#"));
185       display.display();
186     } else if(buttonStateE == LOW){
187       tone1.play(NOTE_E3);
188       display.println(F("E"));
189       display.display();
190     } else if(buttonStateF == LOW){
191       tone1.play(NOTE_F3);
192       display.println(F("F"));
193       display.display();
194     } else if(buttonStateFS == LOW){
195       tone1.play(NOTE_FS3);
196       display.println(F("F#"));
197       display.display();
198     } else if(buttonStateG == LOW){
199       tone1.play(NOTE_G3);
200       display.println(F("G"));
201       display.display();
```

```
202     } else if(buttonStateGS == LOW){
203       tone1.play(NOTE_GS3);
204       display.println(F("G#"));
205       display.display();
206     } else if(buttonStateA == LOW){
207       tone1.play(NOTE_A3);
208       display.println(F("A"));
209       display.display();
210     } else if(buttonStateAS == LOW){
211       tone1.play(NOTE_AS3);
212       display.println(F("A#"));
213       display.display();
214     } else if(buttonStateB == LOW){
215       tone1.play(NOTE_B3);
216       display.println(F("B"));
217       display.display();
218     } else if(recordState == LOW){
219       MODE = 1;
220     } else {
221       // turn LED off:
222       tone1.stop();
223       display.println(F("OFF"));
224       display.display();
225     }
226   }
227 }
228 
229 void record(){
230   clearMelody();
231   currentRecordNote = 0;
232 
233   // Indicate mode change on display
234   display.clearDisplay();
235   display.println(F("RECORD"));
236   display.display();
237   delay(500);
238 
239   display.clearDisplay();
240   display.display();
241 
242   while(true){
243     display.clearDisplay();
244     display.setCursor(0, 0);
245     readPins();
246 
247     /* Check if the pushbutton is pressed. If it is, the
            buttonState is LOW:
248        play note and display it */
249     if (buttonStateC == LOW) {
250       tone1.play(NOTE_C3);
251       display.println(F("C"));
252       display.display();
253       while(true){
254         readPins();
255         if(buttonStateC == HIGH){
256           break;
```

```
257         }
258       }
259       melody[currentRecordNote] = NOTE_C3;
260       currentRecordNote++;
261     } else if(buttonStateCS == LOW){
262       tone1.play(NOTE_CS3);
263       display.println(F("C#"));
264       display.display();
265       while(true){
266         readPins();
267         if(buttonStateCS == HIGH){
268           break;
269         }
270       }
271       melody[currentRecordNote] = NOTE_CS3;
272       currentRecordNote++;
273     } else if(buttonStateD == LOW){
274       tone1.play(NOTE_D3);
275       display.println(F("D"));
276       display.display();
277       while(true){
278         readPins();
279         if(buttonStateD == HIGH){
280           break;
281         }
282       }
283       melody[currentRecordNote] = NOTE_D3;
284       currentRecordNote++;
285     } else if(buttonStateDS == LOW){
286       tone1.play(NOTE_DS3);
287       display.println(F("D#"));
288       display.display();
289       while(true){
290         readPins();
291         if(buttonStateDS == HIGH){
292           break;
293         }
294       }
295       melody[currentRecordNote] = NOTE_DS3;
296       currentRecordNote++;
297     } else if(buttonStateE == LOW){
298       tone1.play(NOTE_E3);
299       display.println(F("E"));
300       display.display();
301       while(true){
302         readPins();
303         if(buttonStateE == HIGH){
304           break;
305         }
306       }
307       melody[currentRecordNote] = NOTE_E3;
308       currentRecordNote++;
309     } else if(buttonStateF == LOW){
310       tone1.play(NOTE_F3);
311       display.println(F("F"));
312       display.display();
```

```
313       while(true){
314         readPins();
315         if(buttonStateF == HIGH){
316           break;
317         }
318       }
319       melody[currentRecordNote] = NOTE_F3;
320       currentRecordNote++;
321     } else if(buttonStateFS == LOW){
322       tone1.play(NOTE_FS3);
323       display.println(F("F#"));
324       display.display();
325       while(true){
326         readPins();
327         if(buttonStateFS == HIGH){
328           break;
329         }
330       }
331       melody[currentRecordNote] = NOTE_FS3;
332       currentRecordNote++;
333     } else if(buttonStateG == LOW){
334       tone1.play(NOTE_G3);
335       display.println(F("G"));
336       display.display();
337       while(true){
338         readPins();
339         if(buttonStateG == HIGH){
340           break;
341         }
342       }
343       melody[currentRecordNote] = NOTE_G3;
344       currentRecordNote++;
345     } else if(buttonStateGS == LOW){
346       tone1.play(NOTE_GS3);
347       display.println(F("G#"));
348       display.display();
349       while(true){
350         readPins();
351         if(buttonStateGS == HIGH){
352           break;
353         }
354       }
355       melody[currentRecordNote] = NOTE_GS3;
356       currentRecordNote++;
357     } else if(buttonStateA == LOW){
358       tone1.play(NOTE_A3);
359       display.println(F("A"));
360       display.display();
361       while(true){
362         readPins();
363         if(buttonStateA == HIGH){
364           break;
365         }
366       }
367       melody[currentRecordNote] = NOTE_A3;
368       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
      ++) {
```

```
424     Serial.println(melody[thisNote]);
425     // Stop playing at end of melody
426     if(melody[thisNote] == -1){
427       display.clearDisplay();
428       display.setCursor(0, 0);
429       display.println(F("End"));
430       display.display();
431       delay(1000);
432       MODE = 0;
433       return;
434     }
435
436     // Start playing note
437     tone1.play(melody[thisNote]);
438
439     // to distinguish the notes, set a minimum time
          between them.
440     // the note's duration + 30% seems to work well:
441     int noteDuration = 1000 / 4; // All 1/4 notes
442     int pauseBetweenNotes = noteDuration * 0.5;
443
444     display.clearDisplay();
445     display.setCursor(0, 0);
446     display.println(F("PLAYING"));
447     display.display();
448
449     delay(noteDuration);
450
451     display.clearDisplay();
452     display.display();
453
454     // Stop the tone playing:
455     tone1.stop();
456     delay(pauseBetweenNotes);
457   }
458   MODE = 0;
459   return;
460 }
461
462 void loop() {
463   live();
464 }
```

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.