AY & SID
Vintage Synthesizer

controlled by an Arduino Mega

John Talbert - April 20, 2018

Table of Contents

Introduction

This project incorporates two vintage synthesizer sound chips:

The **AY-3-8910** is a 3-voice programmable sound generator designed by General Instrument. The AY-3-8910 and its variants became popular chips in many 1970 and 80s arcade games, and were used on, among others, the Intellivision and Vectrex video game consoles, Amstrad CPC, Oric 1, Colour Genie, Elektor TV Games Computer and Sinclair ZX Spectrum 128/+2/+3 home computers as well as the Mockingboard and Cricket sound cards for the Apple II family.

The MOS Technology 6581/8580 **SID** (**Sound Interface Device**) is the built-in Programmable Sound Generator chip of Commodore's CBM-II, Commodore 64, Commodore 128 and Commodore MAX Machine home computers of the 1980s. It was one of the first sound chips of its kind to be included in a home computer prior to the digital sound revolution.

The two sound chips are hardwired to an Arduino Mega microcomputer. Several controllers are also connected to the Arduino to facilitate realtime performance of the synthesizer sound outputs. These include 4 switches, 6 sliders, 6 light sensors, one soft pot, 9 other pots and volume controls, a MIDI input and a MIDI output. All the controllers can be programmed to affect whatever synthesizer parameter you desire using the Arduino's programming IDE app and a simple USB connection between a host computer and the project box.

AY/SID Project Box

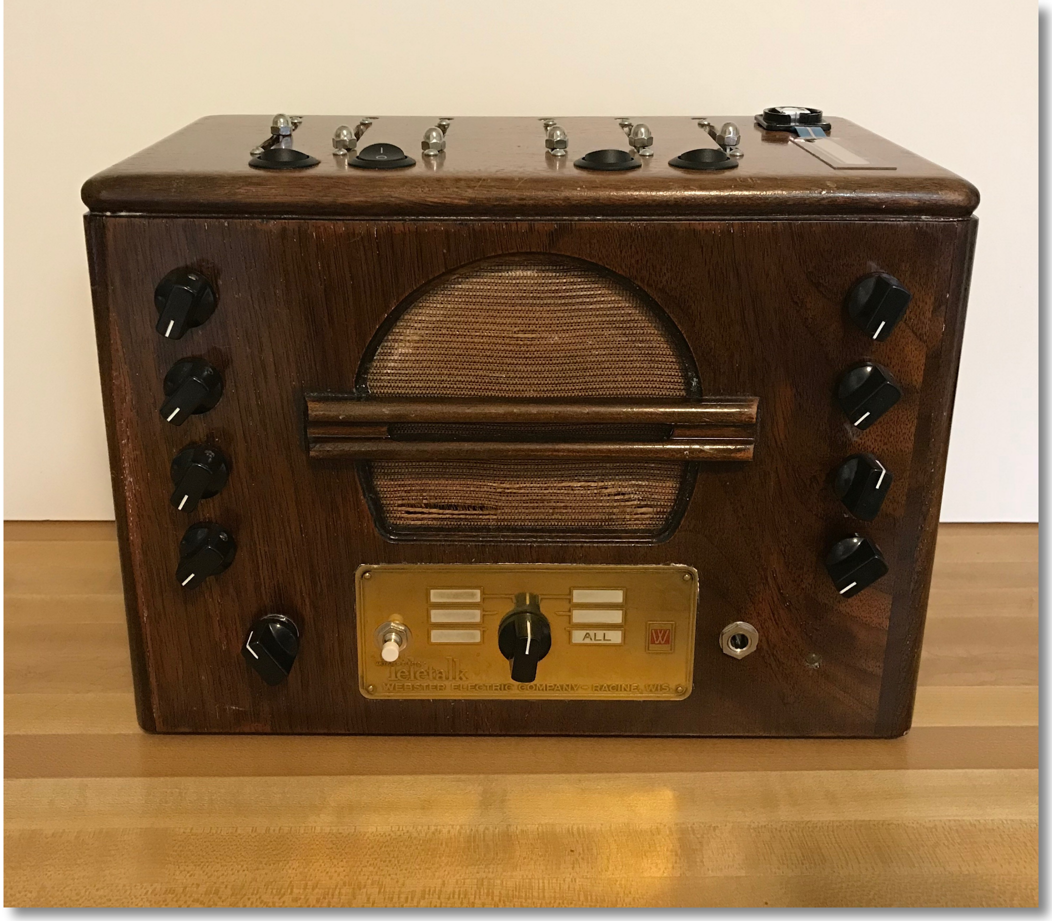
Note: the box used is an old Teletalk Office Intercom

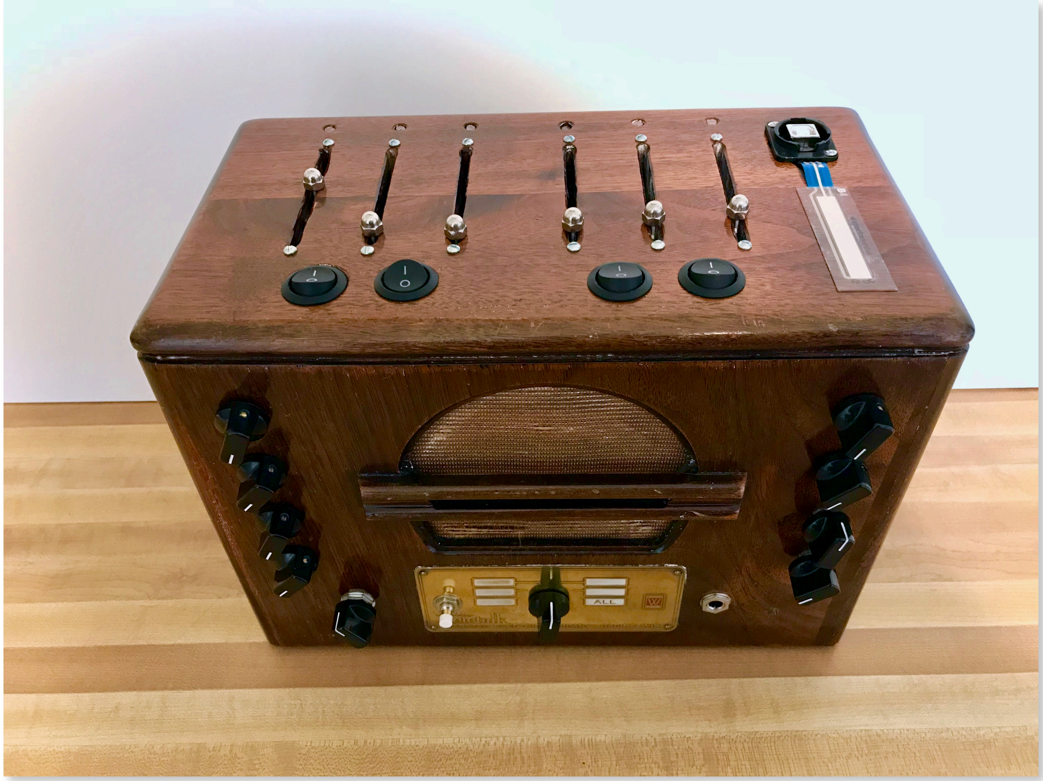
A third sound generator, Voice D, has been added to the 3 squarewave voices of the AY synthesizer. Voice D is generated by combining 3 digital outputs from the Arduino with a diode gate circuit. The sonic possibilities of the AY synth's 3 simple squarewaves have been expanded to include gate modulation of Voice D with the three AY voices added together ((VoiceA + VoiceB + VoiceC) * Voice D), or gate modulation of all 4 voices together (VoiceA * VoiceB * VoiceC * VoiceD), set by a front panel switch. Voice D can be disabled by simply setting low all three of the Arduino pins used to create it, in which case, the 3 AY voices are either mixed together or modulated together. Each of the 4 voices has a front panel volume control.

The SID chip is a more advanced synthesizer compared to the AY chip. Its three voices can be pulse, triangle, sawtooth or noise waveforms. Each voice has an ADSR (Attack, Decay, Sustain, Release) Envelope Generator. Two voices can be paired to create Ring Modulation or SYNC Modulation. The three voices plus an added external signal can be sent through a Filter with programmable cutoff Frequency and Resonance. The AY output is prepatched to serve as this SID external input. HighPass, LowPass, or BandPass can be selected as the Filter response.

Read the Tech Sheets for the AY and SID chips for detailed descriptions of the synth chip parameters. I have created an Arduino Code Template with all the functions and constants needed to access the synth parameters for both chips and manipulate them with the box controller devices.

The following pages provide Circuit Diagrams, Photos and Arduino Code examples.

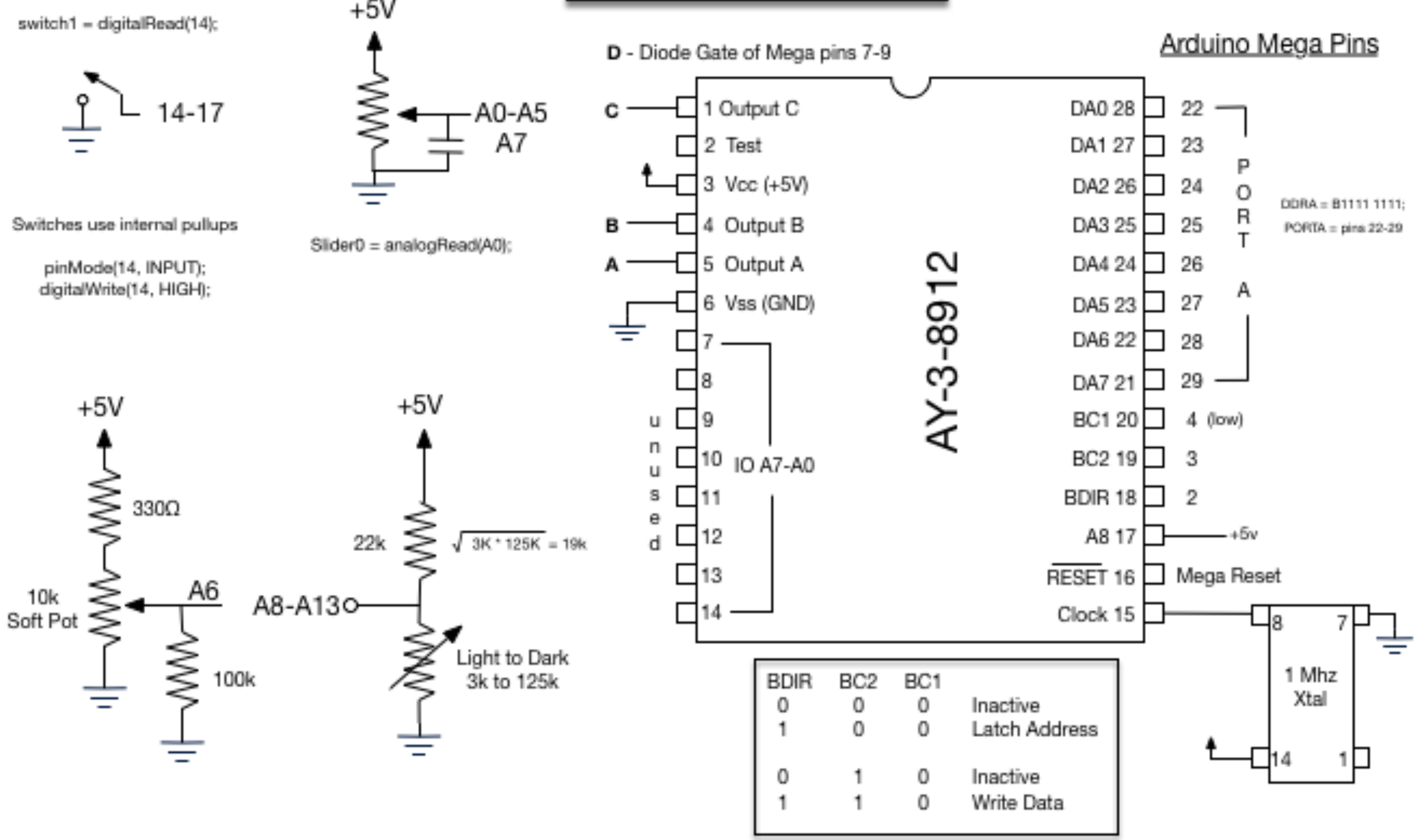

ALL
W
Teletalk
WEBSTER ELECTRIC COMPANY - RACINE, WIS.


ALL
W

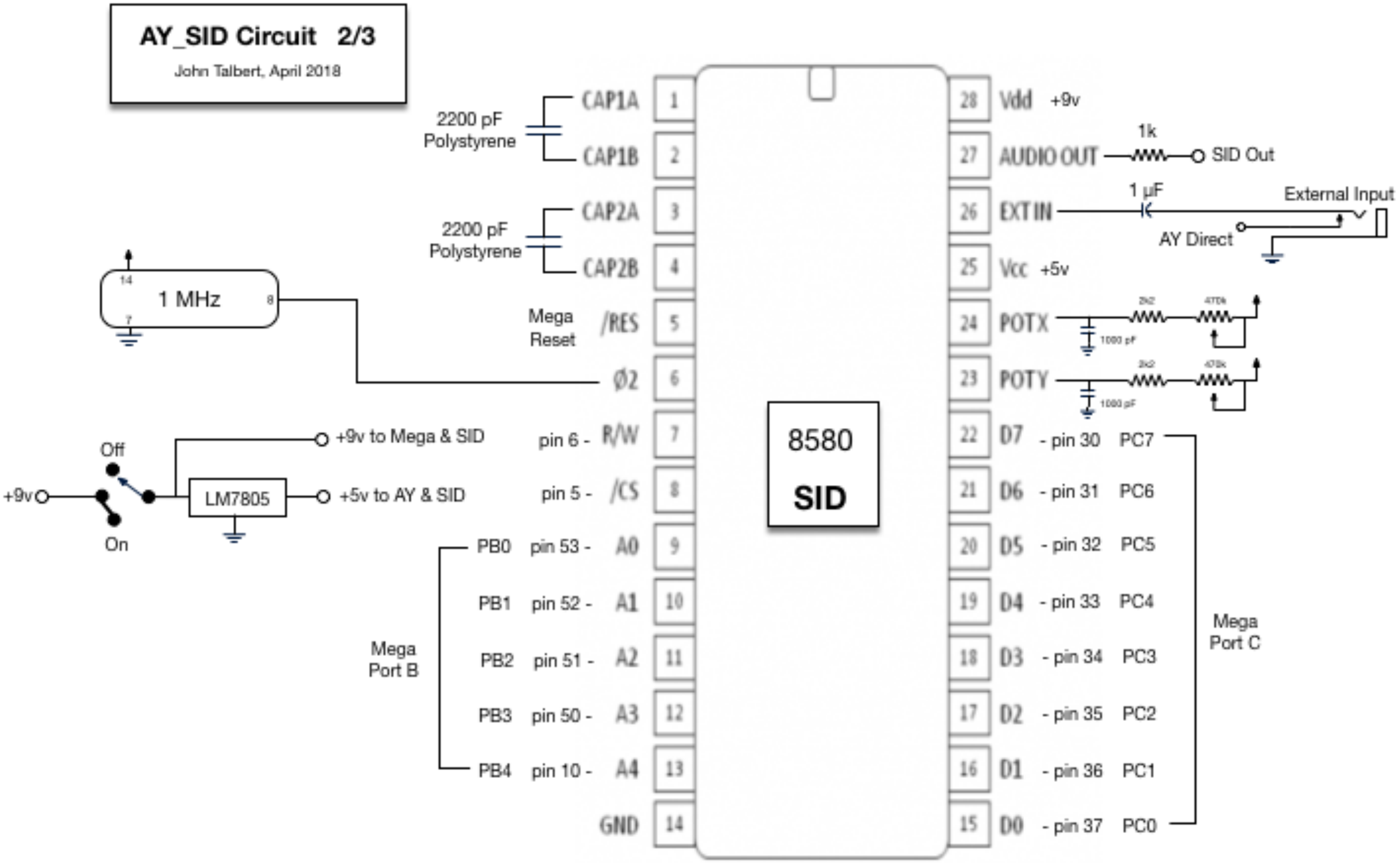
Circuit Diagrams with Photos

AY_SID 1/3

John Talbert, April 2018

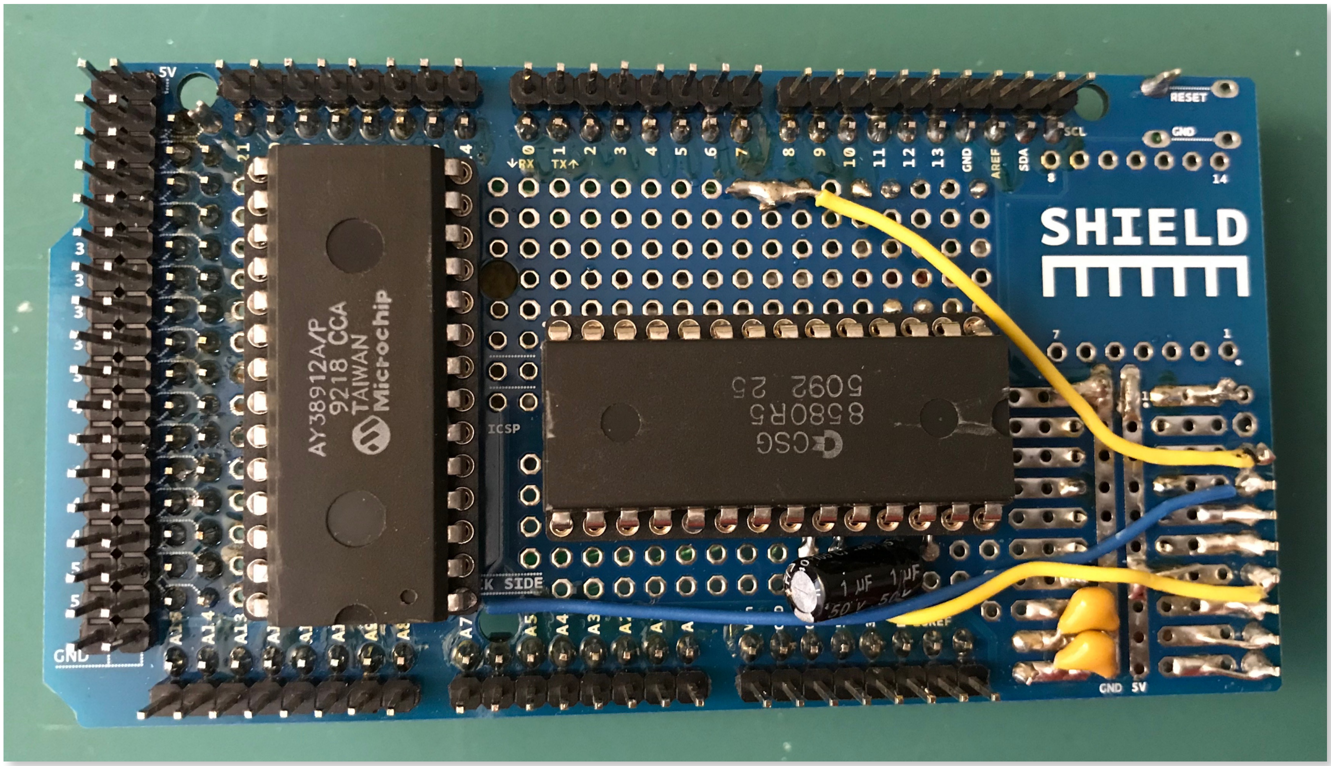


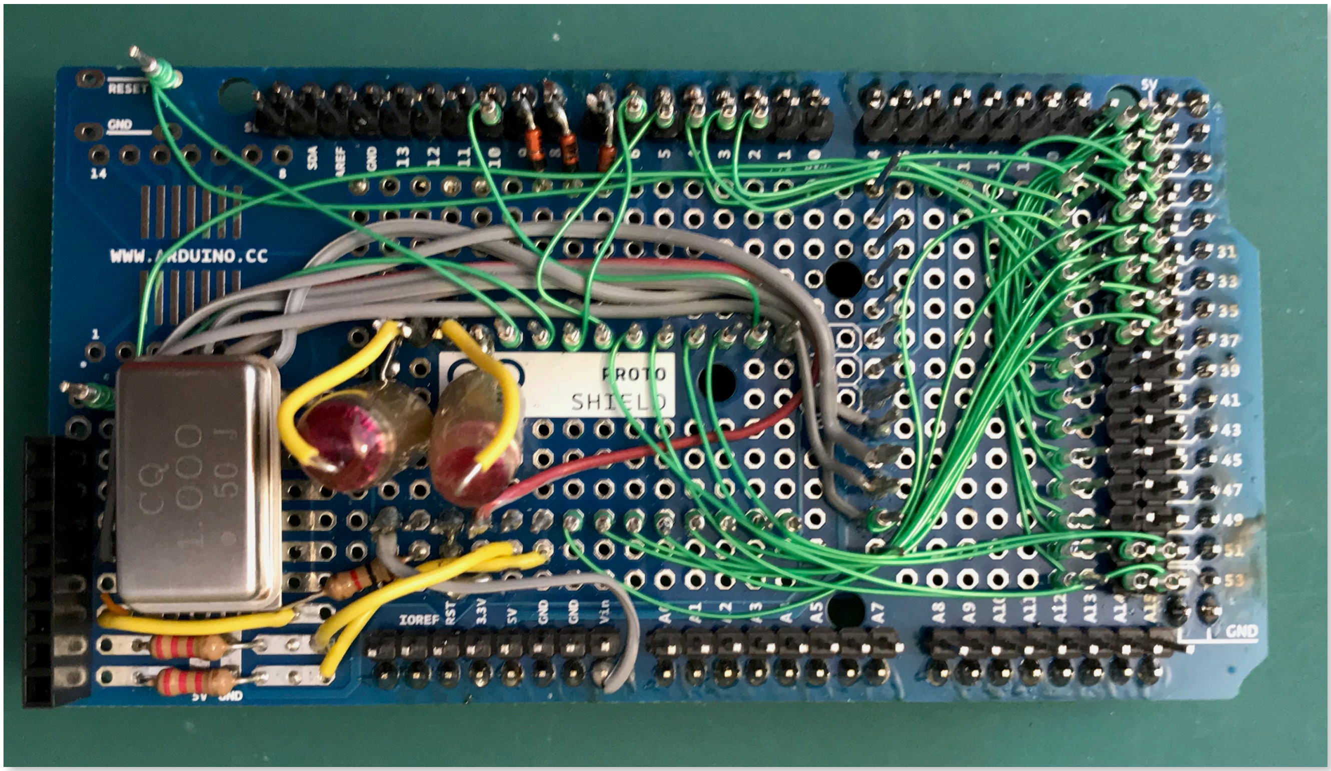
BDIR	BC2	BC1	
0	0	0	Inactive
1	0	0	Latch Address
0	1	0	Inactive
1	1	0	Write Data



AY_SID Circuit 2/3

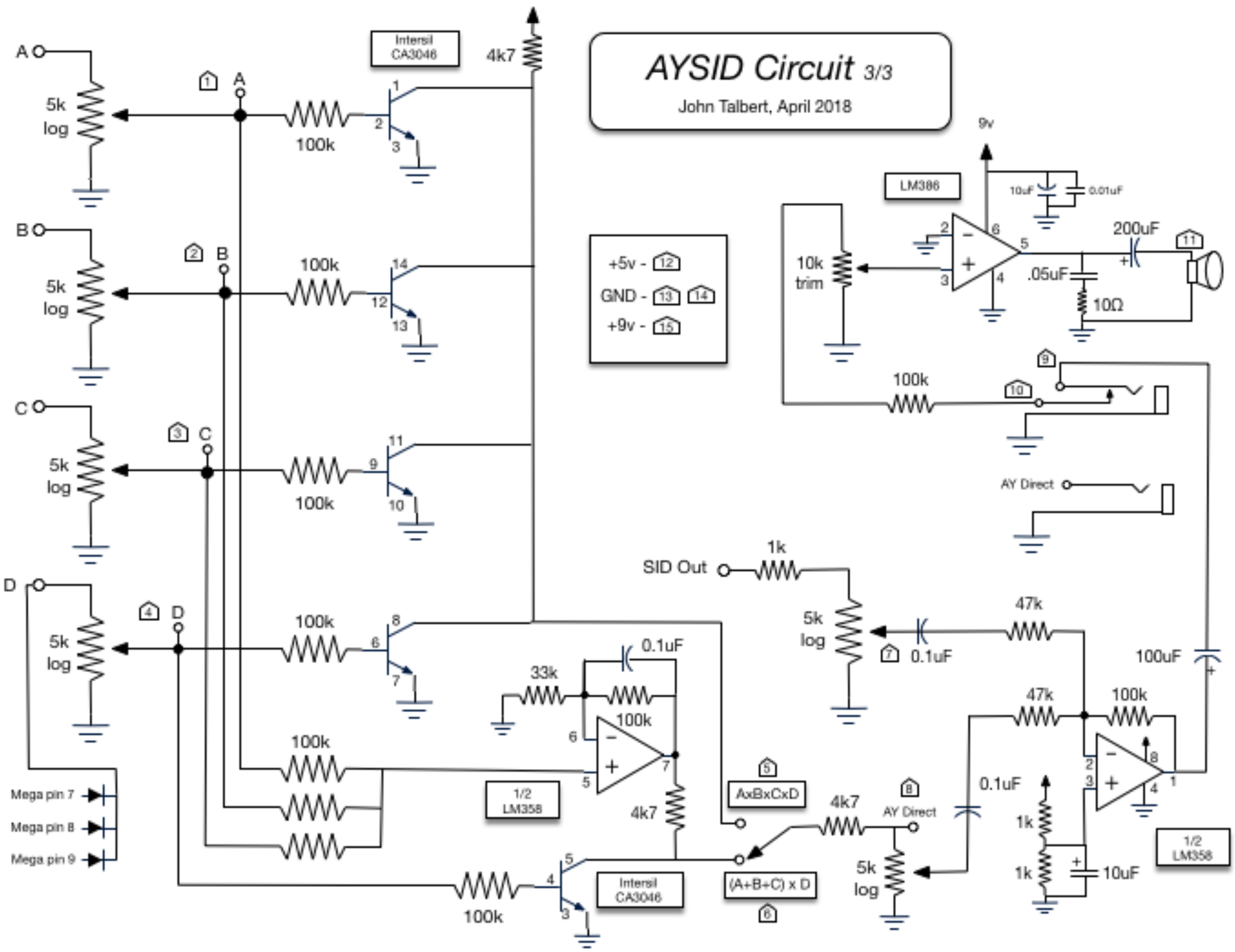
John Talbert, April 2018

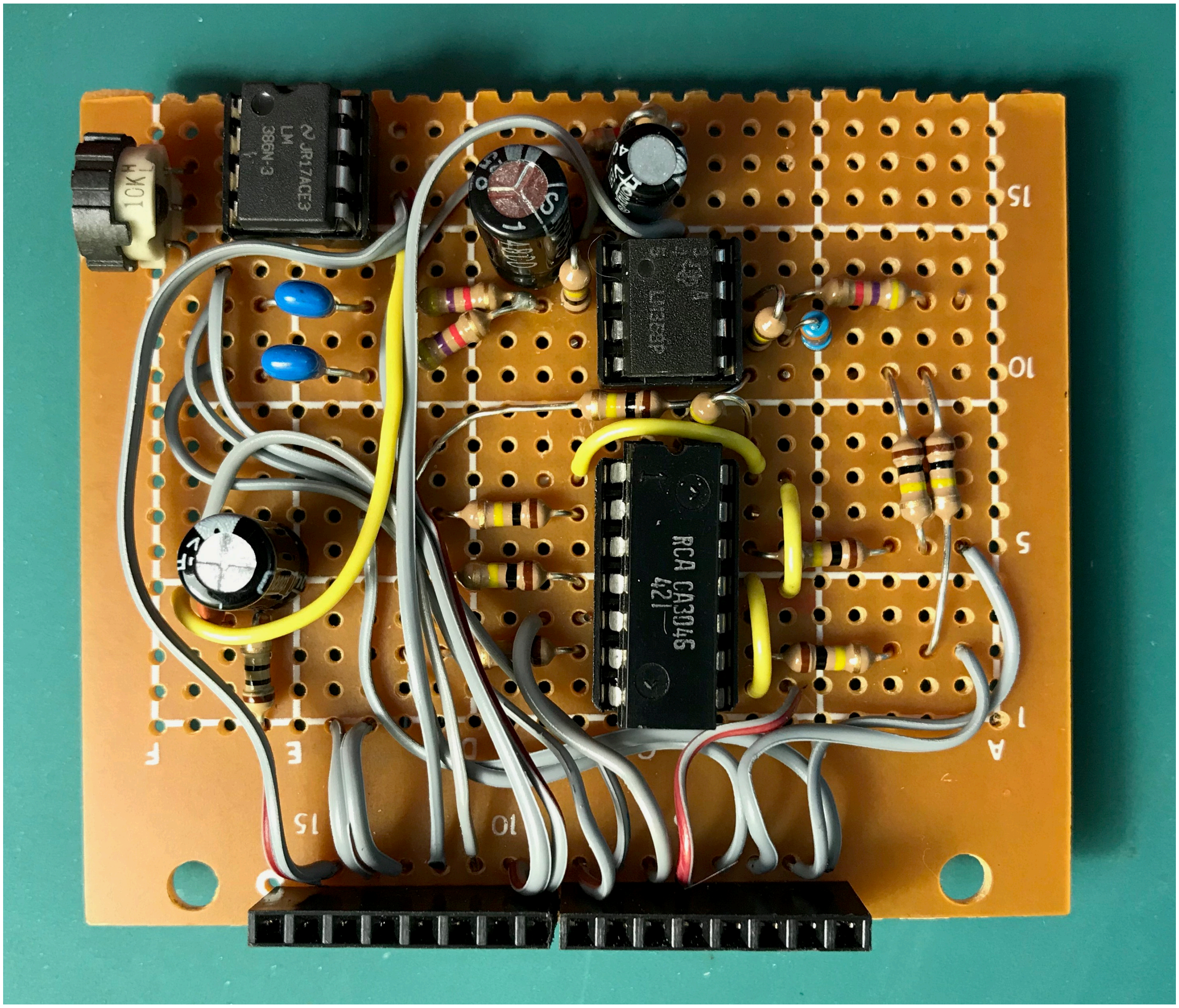

AY38912A/P
9218 CCA
TAIWAN
Microchip
CSG
8580R5
5092 25
SHIELD
ICSP
RESET
GND


WWW.ARDUINO.CC
PROTO
SHIELD
CQ
1.000
50 J
RESET
GND
IOREF

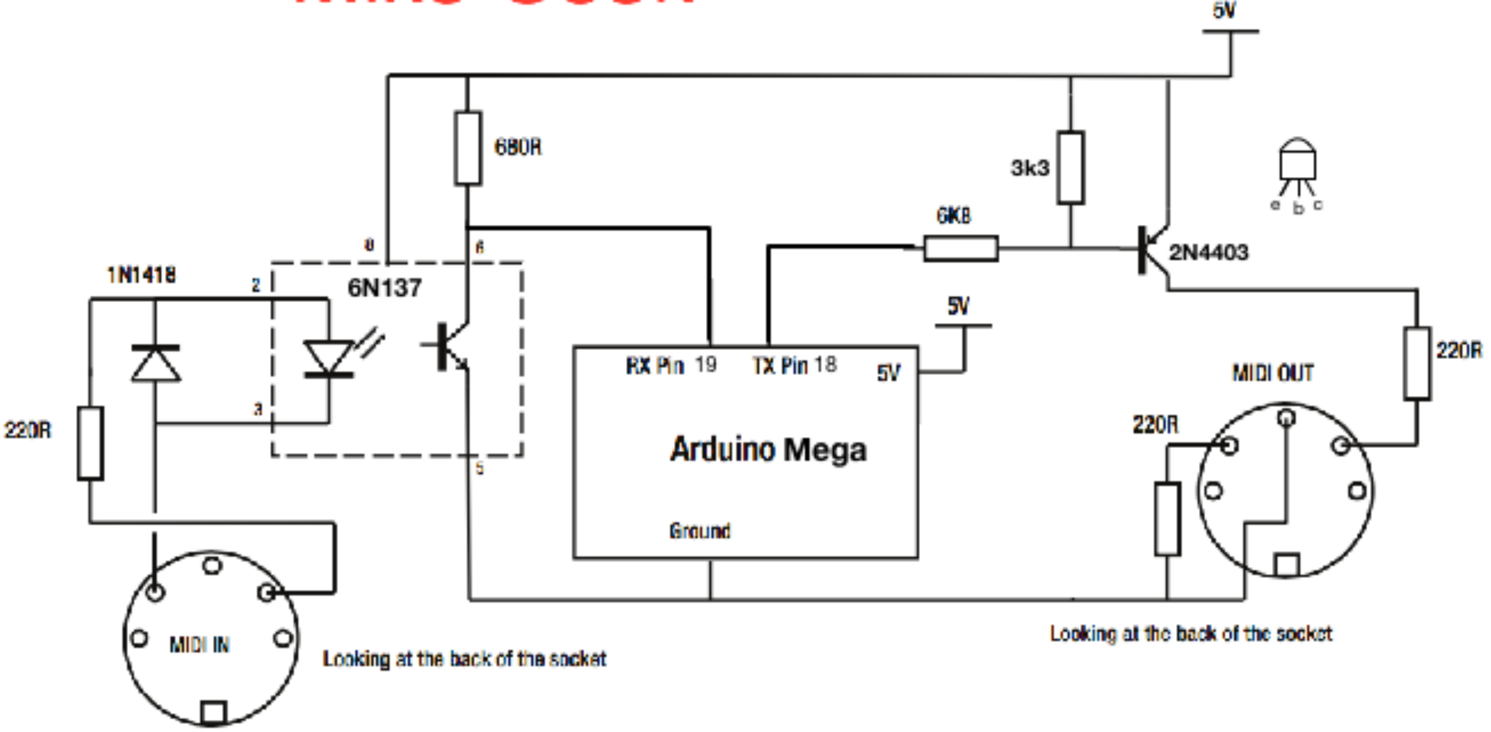
AYSID Circuit 3/3

John Talbert, April 2018




10K
JR17ACE3
LM
386N-3
RCA CA3046
421
15
10
5
1
A
E
F
15
10

Arduino Music and Audio Projects
Mike Cook



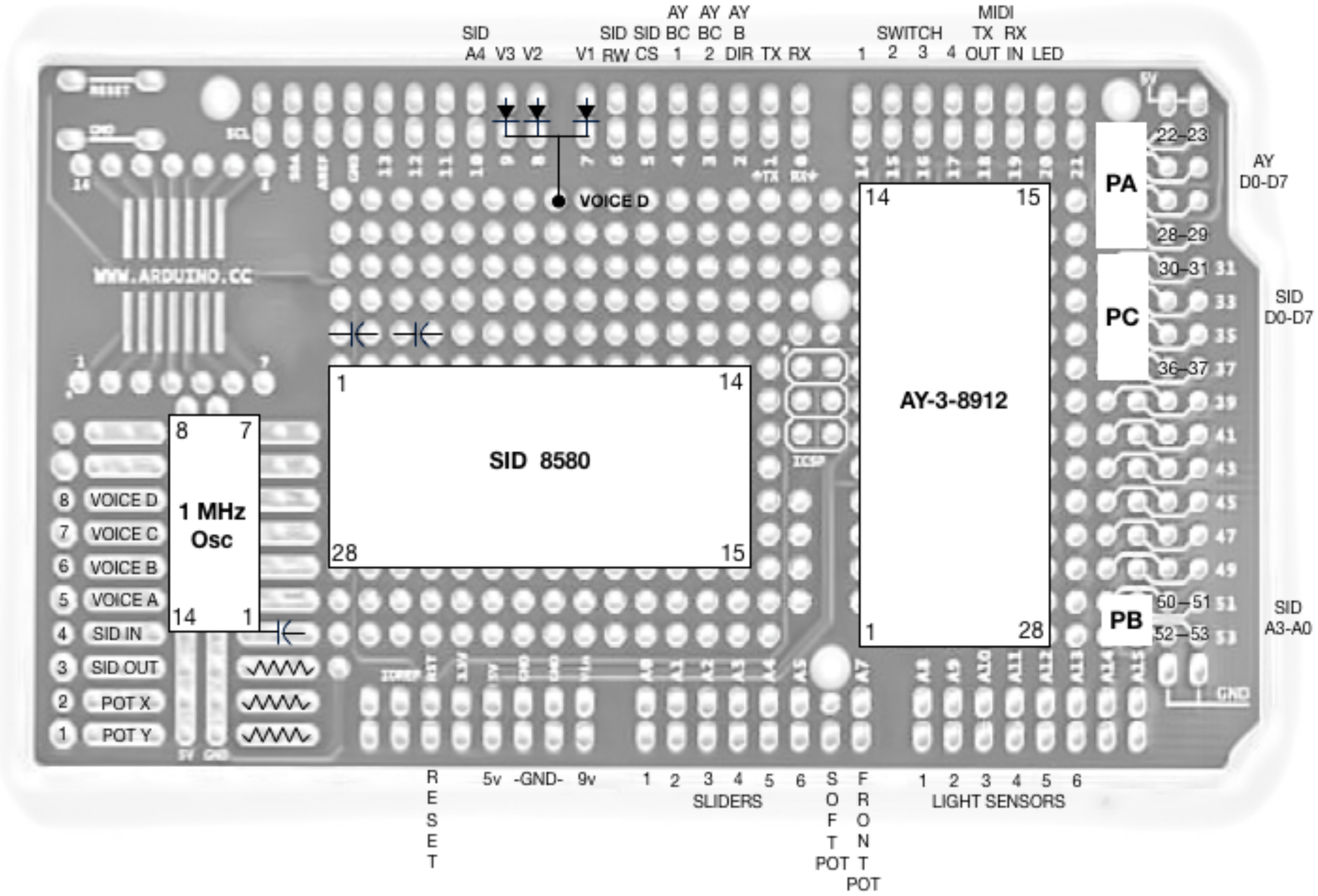
Circuit on small long board next to Arduino

Arduino Pin Assignment

	AY_SID Mega Arduino Pin Connections JT 8/2017
0	Rx0 -- USB programming
1	Tx0 -- USB programming
2	AY BDIR
3	AY BC2
4	AY BC1 (low)
5	SID CS (active low)
6	SID R/W (low write)
7	Voice 1(pwm) Voice 1-3 input to a Diode Gate
8	Voice 2 (pwm) Diode Gate output is AY Voice D
9	Voice 3 (pwm)
10	PB4, SID A4
11-13	PB5-PB7 unused
14	Switch 1
15	Switch 2
16	Switch 3
17	Switch 4

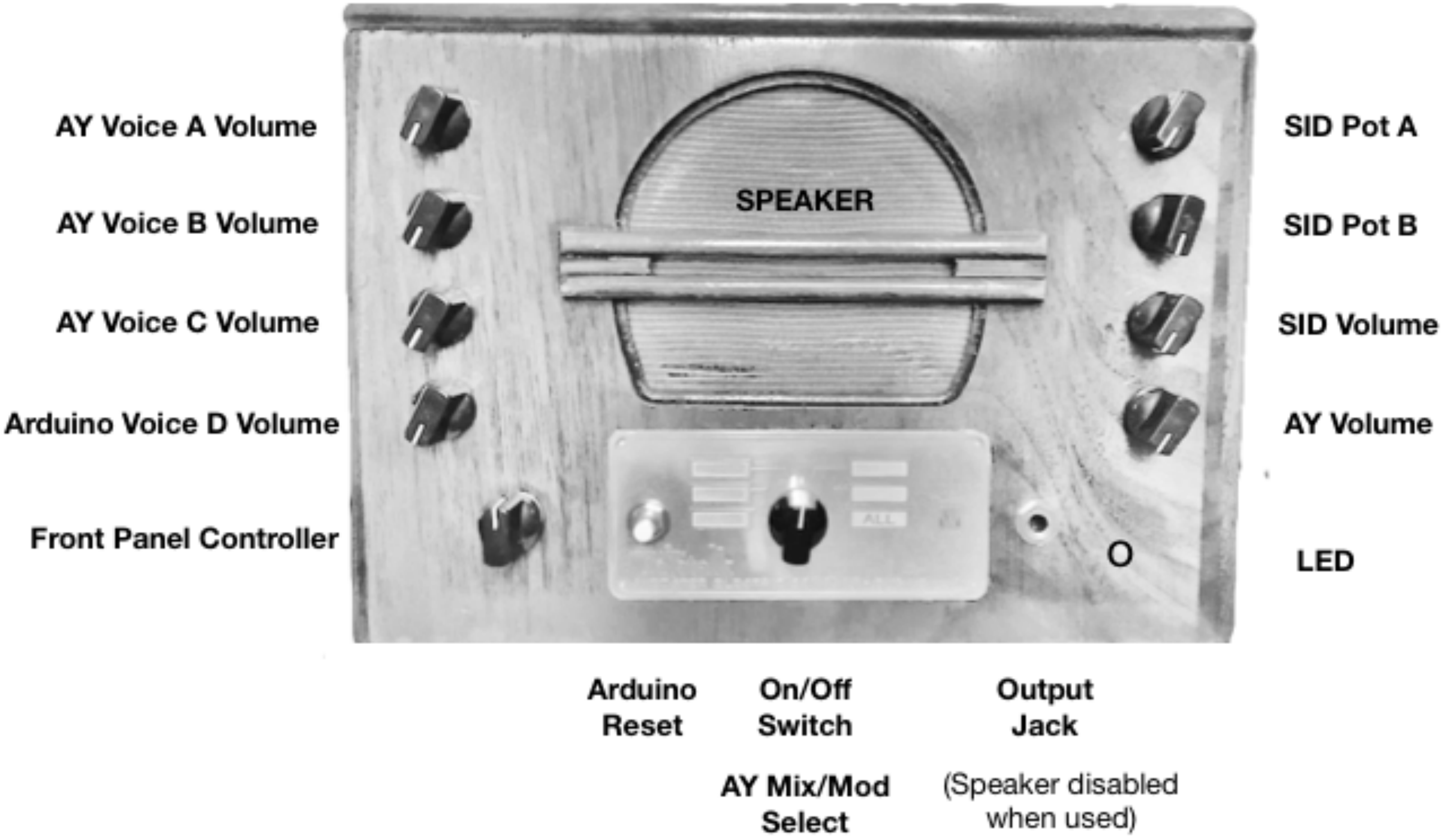
18	MIDI Out Tx, Serial1
19	MIDI In Rx, Serial1
20	LED Front Panel
21	
22-29	**PA, AY DA0-DA7**
37-30	**PC SID D0-D7**
53-50	PB0-PB3, SID A0-A3 (reverse pin numbering)
10	PB4, SID A4
A0	Slider1
A1	Slider2
A2	Slider3
A3	Slider4
A4	Slider5
A5	Slider6
A6	Soft Pot
A7	Front Panel Pot
A8	Light Sensor 1
A9	Light Sensor 2
A10	Light Sensor 3
A11	Light Sensor 4
A12	Light Sensor 5
A13	Light Sensor 6
A14	
A15	

Mega Protoboard (pin side)

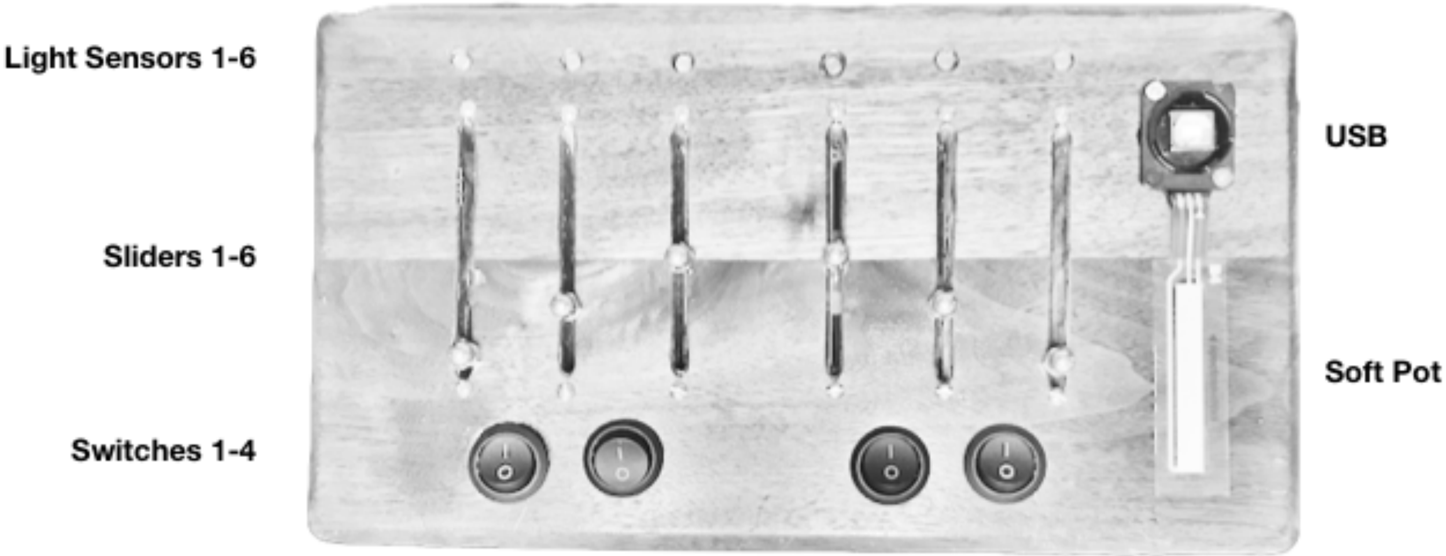


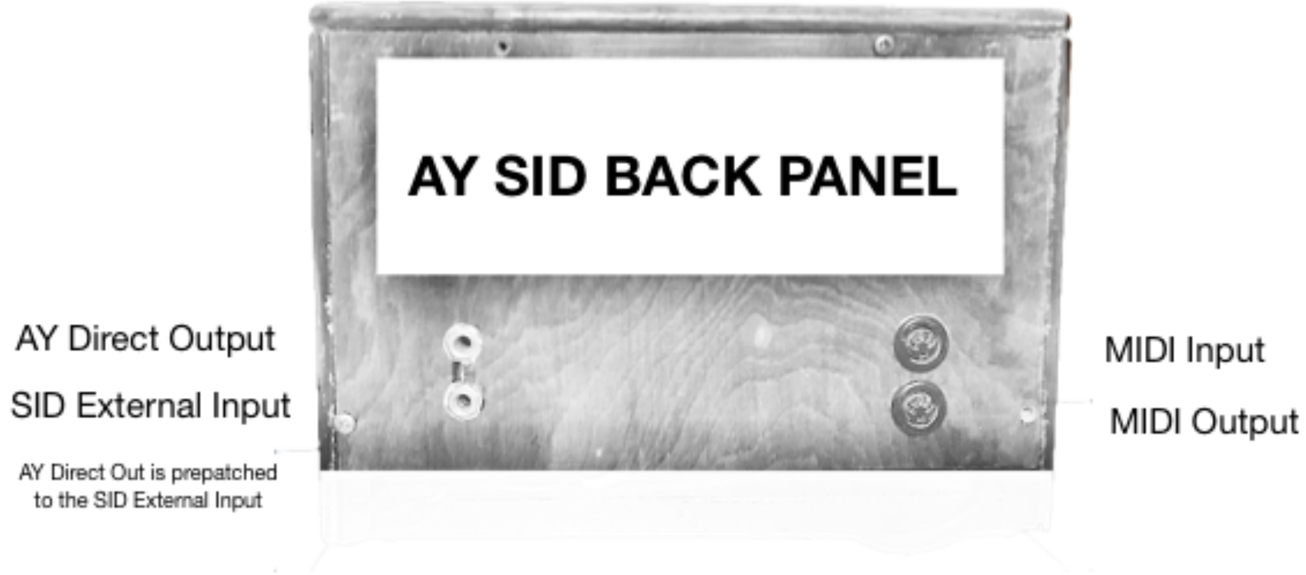
Panel Controls

Front Panel Controls



Top Panel Controls





AY SID BACK PANEL

AY Direct Out is prepatched to the SID External Input

Arduino Code Examples

AYSID_SensorTest

```
/*
          Print Sensor Values.  Test Arduino Voice D
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
          CONSTANTS and Variables
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/

//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;
```

```
//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;


const int VoiceD1 = 7;
const int VoiceD2 = 8;
const int VoiceD3 = 9;



const int LED = 20;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    SETUP()
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


void setup() {

  delay(1000);

Serial.begin(9600);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);

pinMode(VoiceD1, OUTPUT);
digitalWrite(VoiceD1, LOW);
pinMode(VoiceD2, OUTPUT);
digitalWrite(VoiceD2, LOW);
pinMode(VoiceD3, OUTPUT);
digitalWrite(VoiceD3, LOW);
```

```
pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;

}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    Main LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loop() {

 digitalWrite(LED, HIGH);
 /*
 delay(100);
 digitalWrite(LED, LOW);
 delay(50);
 */

 loadSensors();

  Serial.print("s1 = ");
 Serial.print(slider1);
   Serial.print("  s2 = ");
 Serial.print(slider2);
   Serial.print("   s3 = ");
 Serial.print(slider3);
   Serial.print("   s4 = ");
 Serial.print(slider4);
   Serial.print("   s5 = ");
 Serial.print(slider5);
  Serial.print("   s6 = ");
 Serial.print(slider6);

    Serial.print("     ");
```

```
   Serial.print("   soft = ");
  Serial.print(softPot);
   Serial.print("   front = ");
  Serial.print(frontPot);

   Serial.print("     ");

    Serial.print("l1 = ");
  Serial.print(lightSensor1);
    Serial.print("  l2 = ");
  Serial.print(lightSensor2);
    Serial.print("   l3 = ");
  Serial.print(lightSensor3);
    Serial.print("   l4 = ");
  Serial.print(lightSensor4);
    Serial.print("   l5 = ");
  Serial.print(lightSensor5);
   Serial.print("   l6 = ");
  Serial.print(lightSensor6);

 Serial.print("     ");

   Serial.print("  switches ");
  Serial.print(switch1);
  Serial.print(switch2);
  Serial.print(switch3);
  Serial.println(switch4);


tone(VoiceD1, (100 + frontPot));

v = map(slider1, 0, 255, 1, 40);

  digitalWrite(VoiceD2, HIGH);
  delay(v);
  digitalWrite(VoiceD2, LOW);
  delay(v);

} //End of Loop
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loadSensors(){      // load all current sensor values
    slider1 =   analogRead(Slider1) >> 2;
    slider2 =   analogRead(Slider2) >> 2;
    slider3 =   analogRead(Slider3) >> 2;
    slider4 =   analogRead(Slider4) >> 2;
    slider5 =   analogRead(Slider5) >> 2;
    slider6 =   analogRead(Slider6) >> 2;
    softPot =   analogRead(SoftPot) >> 2;
    frontPot =  analogRead(FrontPot) >> 2;
    lightSensor1 =   analogRead(LightSensor1) >> 2;
    lightSensor2 =   analogRead(LightSensor2) >> 2;
    lightSensor3 =   analogRead(LightSensor3) >> 2;
    lightSensor4 =   analogRead(LightSensor4) >> 2;
    lightSensor5 =   analogRead(LightSensor5) >> 2;
    lightSensor6 =   analogRead(LightSensor6) >> 2;
    switch1 = digitalRead(Switch1);
    switch2 = digitalRead(Switch2);
    switch3 = digitalRead(Switch3);
    switch4 = digitalRead(Switch4);
}
```

AYSID_MIDIoutTest

```
/*
                MIDI OUT tested
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#include <MIDI.h>  // version 4.x.x
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// for use on an Arduino Mega.  Midi set up on Serial1, pins 18 and 19.

 const int MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
```

```
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;



const int LED = 20;
int switchx = 0;


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    SETUP()
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


void setup() {

delay(1000);

  MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);


pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
```

```
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;


} //End of Setup

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loop() {

 // before setup() use:   #include <MIDI.h>
//  and MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);
//  in setup() use: MIDI.begin(MIDI_CHANNEL_OMNI);


 for (int i=30; i<60; i++){  //play notes going up

  loadSensors();
  int dur = slider1 + 20;  // note duration
  int velocity = slider2;  // note velocity
  velocity = map(velocity, 0, 255, 0, 127);

  MIDI.sendNoteOn(i, velocity, 1);  //main note_on, note_off
  delay(dur);
  MIDI.sendNoteOff(i, velocity, 1);
  delay(dur);

    if (!switch1){  //All notes off button
    for (int j=0; j<127; j++){
      MIDI.sendNoteOff(i, 0, 1);
    }
  }

  if (!switch2){  //Random Program Change button
    int prog = random(0, 127);
    MIDI.sendProgramChange(prog, 1);
  }
```

```
stopped:
 if(!switch3){  //stop button
   switch3 = digitalRead(Switch3);
   goto stopped;
 }


 }  //End of ramping notes
} //End of Main Loop



void loadSensors(){      // load all current sensor values
     slider1 =   analogRead(Slider1) >> 2;
     slider2 =   analogRead(Slider2) >> 2;
     slider3 =   analogRead(Slider3) >> 2;
     slider4 =   analogRead(Slider4) >> 2;
     slider5 =   analogRead(Slider5) >> 2;
     slider6 =   analogRead(Slider6) >> 2;
     softPot =   analogRead(SoftPot) >> 2;
     frontPot =  analogRead(FrontPot) >> 2;
     lightSensor1 =   analogRead(LightSensor1) >> 2;
     lightSensor2 =   analogRead(LightSensor2) >> 2;
     lightSensor3 =   analogRead(LightSensor3) >> 2;
     lightSensor4 =   analogRead(LightSensor4) >> 2;
     lightSensor5 =   analogRead(LightSensor5) >> 2;
     lightSensor6 =   analogRead(LightSensor6) >> 2;
     switch1 = digitalRead(Switch1);
     switch2 = digitalRead(Switch2);
     switch3 = digitalRead(Switch3);
     switch4 = digitalRead(Switch4);
}
```

AYSID_MIDIinputTest

```
/*
		MIDI INPUT and OUTPUT tested with a MIDI input CallBack Function

		On each NOTE On message received 3 notes will be played.
		Slider 1 sets the playback speed.
		Slider 2 sets the note spread

*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//			CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//
// For use on an Arduino Mega.  MIDI set up on Serial1, pins 18 and 19.

#include <MIDI.h>  // version 4.x.x
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// User created MIDI Callback functions here

 const int MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;
```

```
int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;



const int LED = 20;
int switchx = 0;
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  Callback MIDI_In Test function
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// user created Callback Function for MIDI Input Test

 void myHandleNoteOn(byte channel, byte note, byte velocity){

  velocity = 64;

  int x = slider1; //CV_IN1 pot sets arpeggio speed
  int y = slider2; //CV_IN2 pot sets arpeggio pitch range
  y = map(y, 0, 900, 0, 20);

  MIDI.sendNoteOn(note, velocity, 1);
  delay(x);
  MIDI.sendNoteOn(note + y, velocity, 1);
  delay(x);
  MIDI.sendNoteOn(note + y + y, velocity, 1);
  delay(x);

  MIDI.sendNoteOff(note + y, velocity, 1);
  MIDI.sendNoteOff(note + y + y, velocity, 1);
  MIDI.sendNoteOff(note, velocity, 1);
 }

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  SETUP()
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


void setup() {

delay(1000);

//MIDI Callback Handle references here
//MIDI.begin(MIDI_CHANNEL_OMNI);

  MIDI.setHandleNoteOn(myHandleNoteOn); //for Callback MIDI In Test
  MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);
```

```
pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;


} //End of Setup

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loop() {

loadSensors();

// On MIDI.read() MIDI class will call Callback functions.
// User created callback function myHandleNoteOn() in section before setup()
// and MIDI.setHandleNoteOn(myHandleNoteOn) in setup() section

MIDI.read();

} //End of Main Loop
```

```
void loadSensors(){      // load all current sensor values
    slider1 =   analogRead(Slider1) >> 2;
    slider2 =   analogRead(Slider2) >> 2;
    slider3 =   analogRead(Slider3) >> 2;
    slider4 =   analogRead(Slider4) >> 2;
    slider5 =   analogRead(Slider5) >> 2;
    slider6 =   analogRead(Slider6) >> 2;
    softPot =   analogRead(SoftPot) >> 2;
    frontPot =  analogRead(FrontPot) >> 2;
    lightSensor1 =   analogRead(LightSensor1) >> 2;
    lightSensor2 =   analogRead(LightSensor2) >> 2;
    lightSensor3 =   analogRead(LightSensor3) >> 2;
    lightSensor4 =   analogRead(LightSensor4) >> 2;
    lightSensor5 =   analogRead(LightSensor5) >> 2;
    lightSensor6 =   analogRead(LightSensor6) >> 2;
    switch1 = digitalRead(Switch1);
    switch2 = digitalRead(Switch2);
    switch3 = digitalRead(Switch3);
    switch4 = digitalRead(Switch4);
}
```

AYSID_AYTest1

```
/*
                AY-3-8912 SYNTHESIZER CHIP
    3 Voice Synthesizer controlled through 16 8-bit registers and 2 Control lines

                Control lines are BC1 (pin D4),BC2 (pin D3), BDIR (pin D2)
                Data lines are PortA
                Clock from a 1MHz Oscillator chip
                AY Reset tied to Arduino Reset

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

            Switch1 - Added Noise or ToneOnly
            Switch2 - High/Low Tune

            Slider1 - Noise Frequency
            Slider2 - Tune
            Slider3 - Repeat Time
            Slider4 - Envelope Time
            Slider5 - Envelope Type
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#include <MIDI.h>  // version 4.x.x
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// User created MIDI Callback functions here

 const int  MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
const int Slider3 = 2;
```

```
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;

//
// AY SYTHESIZER CONSTANTS
//
const int AYBDIR = 2;  // set up names for some Arduino pins
const int AYBC1 = 4;
const int AYBC2 = 3;
const int AYFineTuneA = 0;  // AY Synth Control Registers
const int AYFineTuneB = 2;
const int AYFineTuneC = 4;
const int AYCourseTuneA = 1;
const int AYCourseTuneB = 3;
const int AYCourseTuneC = 5;
const int AYNoisePeriod = 6;
const int AYEnable = 7;
const int AYAmpA = 8;
const int AYAmpB = 9;
const int AYAmpC = 10;
const int AYEnvFineTune = 11;
const int AYEnvCourseTune = 12;
const int AYEnvShape = 13;

const int VoiceD1 = 7;
const int VoiceD2 = 8;
const int VoiceD3 = 9;

//
//SID SYNTHESIZER CONSTANTS
//
const int SIDCS = 5;
const int SIDRW = 6;

//SID read register values

int potX = 0;
int potY = 0;
int osc3_rand = 0;
int env3 = 0;

int addr[4] = {0, 0, 7, 14};  //voice register address offsets 1, 2, 3
int v = 0;

// voice register values (Use voice = 1, 2, or 3.  Don't use zero)

int Attack[4] = {0, 0, 0, 0};      // (0 to 15)
int Decay[4] = {0, 0, 0, 0};       // (0 to 15)
int Sustainx[4] = {0, 15, 15, 15};  // (0 to 15)
int Release[4] = {0, 0, 0, 0};     // (0 to 15)
int FreqLo[4] = {0, 0, 0, 0};      // (0 to 255)
int FreqHi[4] = {0, 16, 16, 16};   // (0 to 255)
int PulseWLo[4] = {0, 0, 0, 0};    // (0 to 255)
int PulseWHi[4] = {0, 8, 8, 8};    // (0 to 15)
int Waveshape[4] = {0, 0, 0, 0};   // Load with the bit values below

// bit values for the voice Waveshape (Control) register

const int NOISE = 128;
const int PULSE = 64;
const int SAWTOOTH = 32;
const int TRIANGLE = 16;
const int TEST = 8;
const int RINGMOD = 4;
const int SYNC = 2;

// bit values for filt of ldResFilt, add the ones you want, zero for none

const int FILTEX = 8;  // send external signal through the Filter
const int FILT3 = 4;   // send Voice 3 through the Filter
const int FILT2 = 2;   // send Voice 2 through the Filter
const int FILT1 = 1;   // send Voice 3 through the Filter

// bit values for mode of ldModeVol, add the ones you want, zero for none

const int OFF3 = 128;  // no Voice3 in the output (when used in ring modulation)
const int HP = 64;     // set Filter tos High Pass
const int BP = 32;     // set Filter to BandPass
const int LP = 16;     // set Filter to LowPass

unsigned long timestamp;
unsigned long duration;

const int LED = 20;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                SETUP()
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


void setup() {

delay(1000);

//MIDI Callback Handle references here
//MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);
```

```
pinMode(AYBDIR, OUTPUT);
digitalWrite(AYBDIR, LOW);
pinMode(AYBC1, OUTPUT);
digitalWrite(AYBC1, LOW);
pinMode(AYBC2, OUTPUT);
digitalWrite(AYBC2, LOW);

pinMode(SIDRW, OUTPUT);
digitalWrite(SIDRW, LOW);    // High for Register Read, Low for Write
pinMode(SIDCS, OUTPUT);
digitalWrite(SIDCS, HIGH);   // Chip Select active low, Read/Write to register

pinMode(VoiceD1, OUTPUT);
digitalWrite(VoiceD1, LOW);
pinMode(VoiceD2, OUTPUT);
digitalWrite(VoiceD2, LOW);
pinMode(VoiceD3, OUTPUT);
digitalWrite(VoiceD3, LOW);

pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;

resetSID();

}  //end of Setup

//-------------------------------------------------------
//                MAIN LOOP
//-------------------------------------------------------

void loop() {

 loadSensors();

tone(VoiceD1, (100 + frontPot));
```

```
//--------- Switch2/Slider2 changes AY 3-voice pitches --------------

          AYldFineTuneA(slider2);
          AYldFineTuneB(slider2);
          AYldFineTuneC(slider2);

          if (switch2){
          AYldCourseTuneA(5);
          AYldCourseTuneB(4);
          AYldCourseTuneC(3);
          }
          else{
          AYldCourseTuneA(2);
          AYldCourseTuneB(1);
          AYldCourseTuneC(0);
          }
//---------------  Slider 1 adjusts the Noise Frequency -------------

        AYldNoisePeriod(slider1 >> 3); //Noise Period to Max

//--------------------- Switch1 Noise adds Noise -----------

        if (switch1){
        AYldEnable(B011000);  //Enable noise on VoiceC (low enable)
        }
        else {
        AYldEnable(B111000);  //Enable only tones (low enable)
        }

//------  Slider4 and 5 controls the Envelope on the 3 AY voices ---------

        AYldAmpA(B10000);  // Amplitude controlled by Envelope
        AYldAmpB(B10000);
        AYldAmpC(B10000);

        AYldEnvCourseTune(slider4 >> 2);
        AYldEnvFineTune(slider4 >> 2); //env period

        AYldEnvShape(slider5 >> 4);  //16 Envelope Types

//--------------------  Slider3 controls the tempo ---------

         delay(300 + (slider3 << 3));
 //--------------------------------------------------------

}  //end of loop
```

```
//
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//              AY Address and Data Load Functions
//
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void AYloadAddress(int address){
  PORTA = (address & B1111);
  delayMicroseconds(2);

  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);

  digitalWrite(AYBDIR, LOW);
}

void AYloadData(int data){

  PORTA = data;

  delayMicroseconds(2);
  digitalWrite(AYBC2, HIGH);
  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);
  digitalWrite(AYBDIR, LOW);
  digitalWrite(AYBC2, LOW);
}

void AYloadDataShort(int data){
  PORTA = (data & B111111);

  delayMicroseconds(2);
  digitalWrite(AYBC2, HIGH);
  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);
  digitalWrite(AYBDIR, LOW);
  digitalWrite(AYBC2, LOW);
}
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//            AY General Control Register Load
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void AYldSynth(int address, int data){  // Load 4-bit Control Register Address then 8-bit data
  AYloadAddress(address);
  AYloadData(data);
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//            AY Individual Control Register Load
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void AYldFineTuneA(int data){  //8-bit fine tune A
  AYloadAddress(AYFineTuneA);
  AYloadData(data);
}

void AYldFineTuneB(int data){  //8-bit fine tune B
  AYloadAddress(AYFineTuneB);
  AYloadData(data);
}

void AYldFineTuneC(int data){  //8-bit fine tune C
  AYloadAddress(AYFineTuneC);
  AYloadData(data);
}

void AYldCourseTuneA(int data){  //4-bit course tune A
  AYloadAddress(AYCourseTuneA);
  AYloadDataShort(data);
}

void AYldCourseTuneB(int data){  //4-bit course tune B
  AYloadAddress(AYCourseTuneB);
  AYloadDataShort(data);
}

void AYldCourseTuneC(int data){  //4-bit course tune C
  AYloadAddress(AYCourseTuneC);
  AYloadDataShort(data);
}

void AYldNoisePeriod(int data){  //5-bit noise period tune
  AYloadAddress(AYNoisePeriod);
```

```
 AYloadDataShort(data);
}

void AYldEnable(int data){  //Low 1-bit enable, Noise C/B/A Tone C/B/A
 AYloadAddress(AYEnable);
 AYloadDataShort(data);
}

void AYldAmpA(int amp){  //4-bit amplitude of A if 0-15, else use Env if 16 (mode bit high)
 AYloadAddress(AYAmpA);
 AYloadDataShort(amp);
}

void AYldAmpB(int amp){  //4-bit amplitude of B if 0-15, else use Env if 16 (mode bit high)
 AYloadAddress(AYAmpB);
 AYloadDataShort(amp);
}

void AYldAmpC(int amp){  //4-bit amplitude of C if 0-15, else use Env if 16 (mode bit high)
 AYloadAddress(AYAmpC);
 AYloadDataShort(amp);
}

void AYldEnvFineTune(int data){  //8-bit Envelope Period fine tune
 AYloadAddress(AYEnvFineTune);
 AYloadData(data);
}

void AYldEnvCourseTune(int data){  //8-bit Envelope Period course tune
 AYloadAddress(AYEnvCourseTune);
 AYloadData(data);
}

void AYldEnvShape(int data){  //4-bit Envelope Shape. Continue/Attack/Alternate/Hold
 AYloadAddress(AYEnvShape);
 AYloadDataShort(data);
}


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//              SID Basic Address and Data Functions
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
void loadAddress(int address){
 PORTB = address;
}
```

```
void loadData(int data){
 PORTC = data;
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);
}

void resetSID(){
 for(int i=0; i<25; i++){
  loadAddress(i);
  loadData(0);
 }
 ldModeVol(LP, 255);
 }

void readRegisters(){      //Collect values of all 4 SID readable registers
 DDRC = B00000000;       //Setup Data Lines as Inputs
 digitalWrite(SIDRW, HIGH);  // Setup Read/Write for a Data Read

 loadAddress(25);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 potX = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);

 loadAddress(28);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 env3 = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);

 loadAddress(27);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 osc3_rand = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);

 loadAddress(26);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 potY = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);
 digitalWrite(SIDRW, LOW);  // Reset Read/Write to Data Write
 DDRC = B11111111;       // Reset Data Lines as Outputs
}


void loadSensors(){      // load all current sensor values
    slider1 =   analogRead(Slider1) >> 2;
    slider2 =   analogRead(Slider2) >> 2;
    slider3 =   analogRead(Slider3) >> 2;
    slider4 =   analogRead(Slider4) >> 2;
    slider5 =   analogRead(Slider5) >> 2;
    slider6 =   analogRead(Slider6) >> 2;
    softPot =   analogRead(SoftPot) >> 2;
    frontPot =  analogRead(FrontPot) >> 2;
    lightSensor1 =   analogRead(LightSensor1) >> 2;
    lightSensor2 =   analogRead(LightSensor2) >> 2;
    lightSensor3 =   analogRead(LightSensor3) >> 2;
    lightSensor4 =   analogRead(LightSensor4) >> 2;
    lightSensor5 =   analogRead(LightSensor5) >> 2;
    lightSensor6 =   analogRead(LightSensor6) >> 2;
    switch1 = digitalRead(Switch1);
    switch2 = digitalRead(Switch2);
    switch3 = digitalRead(Switch3);
    switch4 = digitalRead(Switch4);
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//            SID Individual Control Register Load
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 /*      ldFreqLo, ldFreqHi,        --voices 1, 2, 3   Frequency
       ldPulseWLo, ldPulseWHi,    --voices 1, 2, 3   Pulse Width
       ldGate,                  --voices 1, 2, 3   Gate the Envelope (+ Waveshape)
       ldEnvAD, ldEnvSR,         --voices 1, 2, 3   Envelope ADSR
       ldFCLo, ldFCHi, ldResFilt  --Filter Cutoff Frequency/Resonance
       ldModeVol                 --FilterType/OutputVolume
*/

void ldFreqLo(int voice){  //8-bit fine tune frequency -- FreqLo
 loadAddress(addr[voice]);
 loadData(FreqLo[voice] & 255);
}
```

```
void ldFreqHi(int voice){  //8-bit course tune frequency -- FreqHi
  loadAddress(addr[voice]+1);
  loadData(FreqHi[voice] & 255);
}

void ldPulseWLo(int voice){  //8-bit fine tune Pulse Width -- PulseWLo
  loadAddress(addr[voice]+2);
  loadData(PulseWLo[voice] & 255);
}

void ldPulseWHi(int voice){  //4-bit course tune Pulse Width -- PulseWHi
  loadAddress(addr[voice]+3);
  loadData(PulseWHi[voice] & 15);
}

void ldGate(int voice, int gate){  // Gates the ADSR Envelope (also loads Waveshape)

  loadAddress(addr[voice]+4);
  loadData(Waveshape[voice] + gate);
}

void ldEnvAD(int voice){  //4-bit Attack Time, 4-bit Decay time -- Env Attack/Decay
  loadAddress(addr[voice]+5);
  int x = ((Attack[voice] & 15) << 4) + (Decay[voice] & 15);
  loadData(x);
}


void ldEnvSR(int voice){  //4-bit Sustain Level, 4-bit Release time -- Env Sustain/Release
  loadAddress(addr[voice]+6);
  int x = ((Sustainx[voice] & 15) << 4) + (Release[voice] & 15);
  loadData(x);
}

void ldFCLo(int data){  //3-bit fine tune Filter Cutoff Frequency
  loadAddress(21);
  loadData(data & 7);
}

void ldFCHi(int data){  //8-bit course tune Filter Cutoff Frequency
  loadAddress(22);
  loadData(data & 255);
}
```

```
void ldFiltRes(int filt, int res){   //FILTEX/FILT3/FILT2/FILT1, 4-bit Filter Resonance,
  loadAddress(23);
  int x = ((res & 15) << 4) + (filt & 15);
  loadData(x);
}

void ldModeVol(int mode, int vol){   //Filter Type 3OFF/HP/BP/LP, 4-bit Output Volume
  loadAddress(24);
  loadData((vol & 15) + (mode & B11110000));
```

AYSID_SidTest1

```
/*
        SID Chip SYNC and RING MODULATION tested

        switch1 -> Switch between Sync and Ring Modulation
        switch2 -> Voice3 modulator output turned On or OFF
        Slider1 -> Voice3 Modulator Fine Tune
        PotY -> Voice3 Modulator Course Tune
        Slider4 -> Voice1 Modulated Fine Tune
        PotX -> Voice1 Modulated Course Tune
        Slider6 -> Overall Volume

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

         Commodore SID Chip Controlled by an Arduino Micro
    3 Voice Synthesizer controlled through 29 8-bit registers and 3 Control lines

            8 bit Data on PortC
            5 Address Lines on Port B
            Chip Select (active low) on D5
            R/W (write low) on D6
            Clock from a 1MHz Oscillator chip
            Arduino and SID Reset lines tied together
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/

//              CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#include <MIDI.h>  // version 4.x.x
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// User created MIDI Callback functions here

 const int  MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
```

```
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;

//
// AY SYTHESIZER CONSTANTS
//
const int AYBDIR = 2;  // set up names for some Arduino pins
const int AYBC1 = 4;
const int AYBC2 = 3;
const int AYFineTuneA = 0;  // AY Synth Control Registers
const int AYFineTuneB = 2;
const int AYFineTuneC = 4;
const int AYCourseTuneA = 1;
const int AYCourseTuneB = 3;
const int AYCourseTuneC = 5;
const int AYNoisePeriod = 6;
const int AYEnable = 7;
const int AYAmpA = 8;
const int AYAmpB = 9;
const int AYAmpC = 10;
const int AYEnvFineTune = 11;
const int AYEnvCourseTune = 12;
const int AYEnvShape = 13;

const int VoiceD1 = 7;
const int VoiceD2 = 8;
const int VoiceD3 = 9;

//
//SID SYNTHESIZER CONSTANTS
//
const int SIDCS = 5;
const int SIDRW = 6;

//SID read register values

int potX = 0;
int potY = 0;
int osc3_rand = 0;
int env3 = 0;

int addr[4] = {0, 0, 7, 14};  //voice register address offsets 1, 2, 3
int v = 0;

// voice register values (Use voice = 1, 2, or 3.  Don't use zero)

int Attack[4] = {0, 0, 0, 0};      // (0 to 15)
int Decay[4] = {0, 0, 0, 0};       // (0 to 15)
int Sustainx[4] = {0, 15, 15, 15};  // (0 to 15)
int Release[4] = {0, 0, 0, 0};     // (0 to 15)
int FreqLo[4] = {0, 0, 0, 0};      // (0 to 255)
int FreqHi[4] = {0, 16, 16, 16};   // (0 to 255)
int PulseWLo[4] = {0, 0, 0, 0};    // (0 to 255)
int PulseWHi[4] = {0, 8, 8, 8};    // (0 to 15)
int Waveshape[4] = {0, 0, 0, 0};   // Load with the bit values below
```

```
// bit values for the voice Waveshape (Control) register

const int NOISE = 128;
const int PULSE = 64;
const int SAWTOOTH = 32;
const int TRIANGLE = 16;
const int TEST = 8;
const int RINGMOD = 4;
const int SYNC = 2;

// bit values for filt of ldResFilt, add the ones you want, zero for none

const int FILTEX = 8;  // send external signal through the Filter
const int FILT3 = 4;   // send Voice 3 through the Filter
const int FILT2 = 2;   // send Voice 2 through the Filter
const int FILT1 = 1;   // send Voice 3 through the Filter

// bit values for mode of ldModeVol, add the ones you want, zero for none

const int OFF3 = 128;  // no Voice3 in the output (when used in ring modulation)
const int HP = 64;     // set Filter tos High Pass
const int BP = 32;     // set Filter to BandPass
const int LP = 16;     // set Filter to LowPass

unsigned long timestamp;
unsigned long duration;

const int LED = 20;
int switchx = 0;
int gate = 1;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  SETUP()
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


void setup() {

delay(1000);

//MIDI Callback Handle references here
//MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);
```

```
pinMode(AYBDIR, OUTPUT);
digitalWrite(AYBDIR, LOW);
pinMode(AYBC1, OUTPUT);
digitalWrite(AYBC1, LOW);
pinMode(AYBC2, OUTPUT);
digitalWrite(AYBC2, LOW);

pinMode(SIDRW, OUTPUT);
digitalWrite(SIDRW, LOW);    // High for Read, Low for Write
pinMode(SIDCS, OUTPUT);
digitalWrite(SIDCS, HIGH);   // Chip Select active low, to perform Read or Write to register

pinMode(VoiceD1, OUTPUT);
pinMode(VoiceD2, OUTPUT);
pinMode(VoiceD3, OUTPUT);

pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;

resetSID();

// ~~~~~~~~~~~~~Setup Synth values~~~~~~~~~~~~~~~~~~~~~~~

Attack[1] = 0;
Decay[1] = 0;
ldEnvAD(1);

Sustainx[1] = 15;
Release[1] = 0;
ldEnvSR(1);

Waveshape[1] = TRIANGLE;
digitalWrite(LED, HIGH);


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
} //End of Setup

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loop() {

 readRegisters();
 loadSensors();

 ldModeVol(switch2 * OFF3, slider6 >>4);   //No Filtering, Amplitude on slider6

 if (switch1 != switchx){
 if (switch1){gate = 1 + RINGMOD;} else {gate = 1 + SYNC;}
 ldGate(1, gate);
 switchx = switch1;
}

 FreqLo[3] = slider1;
 FreqHi[3] = potY;
 FreqLo[1] = slider4;
 FreqHi[1] = potX;

 ldFreqLo(3); //Modulation Voice
 ldFreqHi(3);
 ldFreqLo(1); //Modulated Voice
 ldFreqHi(1);

} //End of Main Loop




// ______________________________________________________________
//             AY Address and Data Load Functions
// ______________________________________________________________

void AYloadAddress(int address){
 PORTA = (address & B1111);
 delayMicroseconds(2);

 digitalWrite(AYBDIR, HIGH);
 delayMicroseconds(2);
```

```
 digitalWrite(AYBDIR, LOW);
}

void AYloadData(int data){

 PORTA = data;

 delayMicroseconds(2);
 digitalWrite(AYBC2, HIGH);
 digitalWrite(AYBDIR, HIGH);
 delayMicroseconds(2);
 digitalWrite(AYBDIR, LOW);
 digitalWrite(AYBC2, LOW);
}

void AYloadDataShort(int data){
 PORTA = (data & B111111);

 delayMicroseconds(2);
 digitalWrite(AYBC2, HIGH);
 digitalWrite(AYBDIR, HIGH);
 delayMicroseconds(2);
 digitalWrite(AYBDIR, LOW);
 digitalWrite(AYBC2, LOW);
}


// ____________________________________________________________
//          AY General Control Register Load
// ____________________________________________________________

void AYldSynth(int address, int data){  // Load 4-bit Control Register Address then 8-bit data
 AYloadAddress(address);
 AYloadData(data);
}

// ____________________________________________________________
//          AY Individual Control Register Load
// ____________________________________________________________


void AYldFineTuneA(int data){  //8-bit fine tune A
 AYloadAddress(AYFineTuneA);
 AYloadData(data);
}
```

```
void AYldFineTuneB(int data){  //8-bit fine tune B
  AYloadAddress(AYFineTuneB);
  AYloadData(data);
}

void AYldFineTuneC(int data){  //8-bit fine tune C
  AYloadAddress(AYFineTuneC);
  AYloadData(data);
}

void AYldCourseTuneA(int data){  //4-bit course tune A
  AYloadAddress(AYCourseTuneA);
  AYloadDataShort(data);
}

void AYldCourseTuneB(int data){  //4-bit course tune B
  AYloadAddress(AYCourseTuneB);
  AYloadDataShort(data);
}

void AYldCourseTuneC(int data){  //4-bit course tune C
  AYloadAddress(AYCourseTuneC);
  AYloadDataShort(data);
}

void AYldNoisePeriod(int data){  //5-bit noise period tune
  AYloadAddress(AYNoisePeriod);
  AYloadDataShort(data);
}

void AYldEnable(int data){  //Low 1-bit enable, Noise C/B/A Tone C/B/A
  AYloadAddress(AYEnable);
  AYloadDataShort(data);
}

void AYldAmpA(int amp){  //4-bit amplitude of A if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpA);
  AYloadDataShort(amp);
}

void AYldAmpB(int amp){  //4-bit amplitude of B if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpB);
  AYloadDataShort(amp);
}
```

```
void AYldAmpC(int amp){  //4-bit amplitude of C if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpC);
  AYloadDataShort(amp);
}

void AYldEnvFineTune(int data){  //8-bit Envelope Period fine tune
  AYloadAddress(AYEnvFineTune);
  AYloadData(data);
}

void AYldEnvCourseTune(int data){  //8-bit Envelope Period course tune
  AYloadAddress(AYEnvCourseTune);
  AYloadData(data);
}

void AYldEnvShape(int data){  //4-bit Envelope Shape. Continue/Attack/Alternate/Hold
  AYloadAddress(AYEnvShape);
  AYloadDataShort(data);
}


// ____________________________________________________________________
//              SID Basic Address and Data Functions
// ____________________________________________________________________

void loadAddress(int address){
 PORTB = address;
}

void loadData(int data){
  PORTC = data;
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);
}

void resetSID(){
  for(int i=0; i<25; i++){
    loadAddress(i);
    loadData(0);
  }
  ldModeVol(LP, 255);
  }
```

```
void readRegisters(){     //Collect values of all 4 SID readable registers
 DDRC = B00000000;       //Setup Data Lines as Inputs
 digitalWrite(SIDRW, HIGH);  // Setup Read/Write for a Data Read

 loadAddress(25);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 potX = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);

 loadAddress(28);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 env3 = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);

 loadAddress(27);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 osc3_rand = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);

 loadAddress(26);
 digitalWrite(SIDCS, LOW);
 delayMicroseconds(3);
 potY = PINC;
 delayMicroseconds(3);
 digitalWrite(SIDCS, HIGH);

 digitalWrite(SIDRW, LOW);  // Reset Read/Write to Data Write
 DDRC = B11111111;       // Reset Data Lines as Outputs
}


void loadSensors(){     // load all current sensor values
    slider1 =   analogRead(Slider1) >> 2;
    slider2 =   analogRead(Slider2) >> 2;
    slider3 =   analogRead(Slider3) >> 2;
    slider4 =   analogRead(Slider4) >> 2;
    slider5 =   analogRead(Slider5) >> 2;
    slider6 =   analogRead(Slider6) >> 2;
    softPot =   analogRead(SoftPot) >> 2;
    frontPot =  analogRead(FrontPot) >> 2;
    lightSensor1 =   analogRead(LightSensor1) >> 2;
    lightSensor2 =   analogRead(LightSensor2) >> 2;
    lightSensor3 =   analogRead(LightSensor3) >> 2;
    lightSensor4 =   analogRead(LightSensor4) >> 2;
    lightSensor5 =   analogRead(LightSensor5) >> 2;
    lightSensor6 =   analogRead(LightSensor6) >> 2;
    switch1 = digitalRead(Switch1);
    switch2 = digitalRead(Switch2);
    switch3 = digitalRead(Switch3);
    switch4 = digitalRead(Switch4);
}
// ______________________________________________________________
//            SID Individual Control Register Load
// ______________________________________________________________

 /*      ldFreqLo, ldFreqHi,        --voices 1, 2, 3   Frequency
        ldPulseWLo, ldPulseWHi,    --voices 1, 2, 3   Pulse Width
        ldGate,                  --voices 1, 2, 3   Gate the Envelope (+ Waveshape)
        ldEnvAD, ldEnvSR,          --voices 1, 2, 3   Envelope ADSR
        ldFCLo, ldFCHi, ldResFilt  --Filter Cutoff Frequency/Resonance
        ldModeVol                  --FilterType/OutputVolume
*/

void ldFreqLo(int voice){  //8-bit fine tune frequency -- FreqLo
 loadAddress(addr[voice]);
 loadData(FreqLo[voice] & 255);
}

void ldFreqHi(int voice){  //8-bit course tune frequency -- FreqHi
 loadAddress(addr[voice]+1);
 loadData(FreqHi[voice] & 255);
}

void ldPulseWLo(int voice){  //8-bit fine tune Pulse Width -- PulseWLo
 loadAddress(addr[voice]+2);
 loadData(PulseWLo[voice] & 255);
}

void ldPulseWHi(int voice){  //4-bit course tune Pulse Width -- PulseWHi
 loadAddress(addr[voice]+3);
 loadData(PulseWHi[voice] & 15);
}

void ldGate(int voice, int x){  // Gates the ADSR Envelope (also loads Waveshape)

 loadAddress(addr[voice]+4);
 loadData(Waveshape[voice] + x);
}
```

```
void ldEnvAD(int voice){  //4-bit Attack Time, 4-bit Decay time -- Env Attack/Decay
 loadAddress(addr[voice]+5);
 int x = ((Attack[voice] & 15) << 4) + (Decay[voice] & 15);
 loadData(x);
}


void ldEnvSR(int voice){  //4-bit Sustain Level, 4-bit Release time -- Env Sustain/Release
 loadAddress(addr[voice]+6);
 int x = ((Sustainx[voice] & 15) << 4) + (Release[voice] & 15);
 loadData(x);
}

void ldFCLo(int data){  //3-bit fine tune Filter Cutoff Frequency
 loadAddress(21);
 loadData(data & 7);
}

void ldFCHi(int data){  //8-bit course tune Filter Cutoff Frequency
 loadAddress(22);
 loadData(data & 255);
}

void ldFiltRes(int filt, int res){   //FILTEX/FILT3/FILT2/FILT1, 4-bit Filter Resonance,
 loadAddress(23);
 int x = ((res & 15) << 4) + (filt & 15);
 loadData(x);
}

void ldModeVol(int mode, int vol){   //Filter Type 3OFF/HP/BP/LP, 4-bit Output Volume
 loadAddress(24);
 loadData((vol & 15) + (mode & B11110000));
}
```

AYSID_AYRandom

```
/*
              AY-3-8912 SYNTHESIZER CHIP
   3 Voice Synthesizer controlled through 16 8-bit registers and 2 Control lines

             Control lines are BC1 (pin D4),BC2 (pin D3), BDIR (pin D2)
             Data lines are PortA
             Clock from a 1MHz Oscillator chip
             AY Reset tied to Arduino Reset

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

          Switch1 - Added Noise to VoiceA
          Switch2 - Slow down

          Sliders 1-3 ->  Base Frequency, Voices A-C
          Sliders 4-6 ->  Frequency Ranger, Voices A-C
          FrontPanelPot -> Note Duration/Speed
          SoftPot ->  Noise Frequency
          LightSensor1 -> VoiceD Frequency
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
//-------------------------------------------------------
//               CONSTANTS and Variables
//-------------------------------------------------------

#include <MIDI.h>  // version 4.x.x
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// User created MIDI Callback functions here

 const int  MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
```

```
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;

//
// AY SYTHESIZER CONSTANTS
//
const int AYBDIR = 2;  // set up names for some Arduino pins
const int AYBC1 = 4;
const int AYBC2 = 3;
const int AYFineTuneA = 0;  // AY Synth Control Registers
const int AYFineTuneB = 2;
const int AYFineTuneC = 4;
const int AYCourseTuneA = 1;
const int AYCourseTuneB = 3;
const int AYCourseTuneC = 5;
const int AYNoisePeriod = 6;
const int AYEnable = 7;
const int AYAmpA = 8;
const int AYAmpB = 9;
const int AYAmpC = 10;
const int AYEnvFineTune = 11;
const int AYEnvCourseTune = 12;
const int AYEnvShape = 13;

const int VoiceD1 = 7;
const int VoiceD2 = 8;
const int VoiceD3 = 9;

//
//SID SYNTHESIZER CONSTANTS
//
const int SIDCS = 5;
const int SIDRW = 6;

//SID read register values

int potX = 0;
int potY = 0;
int osc3_rand = 0;
int env3 = 0;

int addr[4] = {0, 0, 7, 14};  //voice register address offsets 1, 2, 3
int v = 0;

// voice register values (Use voice = 1, 2, or 3.  Don't use zero)

int Attack[4] = {0, 0, 0, 0};      // (0 to 15)
int Decay[4] = {0, 0, 0, 0};       // (0 to 15)
int Sustainx[4] = {0, 15, 15, 15};  // (0 to 15)
int Release[4] = {0, 0, 0, 0};     // (0 to 15)
int FreqLo[4] = {0, 0, 0, 0};      // (0 to 255)
int FreqHi[4] = {0, 16, 16, 16};   // (0 to 255)
int PulseWLo[4] = {0, 0, 0, 0};    // (0 to 255)
int PulseWHi[4] = {0, 8, 8, 8};    // (0 to 15)
int Waveshape[4] = {0, 0, 0, 0};   // Load with the bit values below
```

```
// bit values for the voice Waveshape (Control) register

const int NOISE = 128;
const int PULSE = 64;
const int SAWTOOTH = 32;
const int TRIANGLE = 16;
const int TEST = 8;
const int RINGMOD = 4;
const int SYNC = 2;

// bit values for filt of ldResFilt, add the ones you want, zero for none

const int FILTEX = 8;  // send external signal through the Filter
const int FILT3 = 4;   // send Voice 3 through the Filter
const int FILT2 = 2;   // send Voice 2 through the Filter
const int FILT1 = 1;   // send Voice 3 through the Filter

// bit values for mode of ldModeVol, add the ones you want, zero for none

const int OFF3 = 128;  // no Voice3 in the output (when used in ring modulation)
const int HP = 64;     // set Filter tos High Pass
const int BP = 32;     // set Filter to BandPass
const int LP = 16;     // set Filter to LowPass

unsigned long timestamp;
unsigned long duration;

const int LED = 20;

int freq=0;
int durA=0;
int durA_count=0;
int durB=0;
int durB_count=0;
int durC=0;
int durC_count=0;
int envA=0;
int envB=0;
int envC=0;
int dur=0;
```

```
//----------------------------------------------------
//                    SETUP()
//----------------------------------------------------


void setup() {

delay(1000);

//MIDI Callback Handle references here
//MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);

pinMode(AYBDIR, OUTPUT);
digitalWrite(AYBDIR, LOW);
pinMode(AYBC1, OUTPUT);
digitalWrite(AYBC1, LOW);
pinMode(AYBC2, OUTPUT);
digitalWrite(AYBC2, LOW);

pinMode(SIDRW, OUTPUT);
digitalWrite(SIDRW, LOW);    // High for Register Read, Low for Write
pinMode(SIDCS, OUTPUT);
digitalWrite(SIDCS, HIGH);   // Chip Select active low, Read/Write to register

pinMode(VoiceD1, OUTPUT);
digitalWrite(VoiceD1, LOW);
pinMode(VoiceD2, OUTPUT);
digitalWrite(VoiceD2, LOW);
pinMode(VoiceD3, OUTPUT);
digitalWrite(VoiceD3, LOW);

pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;
```

```
resetSID();

}  //end of Setup

//-------------------------------------------------------
//                MAIN LOOP
//-------------------------------------------------------

void loop() {

 loadSensors();

// -----------Voice D from Arduino D6 ------------------


    dur = frontPot >> 3; //set envelope durations

    tone(VoiceD1, (lightSensor1 << 1));  // set frequencey of Voice D



// -----------Switch1 turns on noise in one voice ------------------

        AYldNoisePeriod(softPot >> 3); //Noise Frequency

        if (switch1){
        AYldEnable(B110000);  //Enable noise and tones in one voice (low enable)
        }
        else {
        AYldEnable(B111000);  //Enable only tones (low enable)
        }


// ------------Voice A from AY Chip---------------------

if (envA != 0){  //ramping down voice A envelope, 15 to 0

      if (durA_count != 0){  //wait for a count of durA
          durA_count -= 1;
      }
       else {  // when the count reaches zero decrement voice A envelope, reset count
          durA_count = durA;
          envA -= 1;
          AYldAmpA(envA);
       }
```

```
  }
  else{  // when envelope reaches zero, reset voice A with new frequency and envelope

          freq = getFreqA();  //get new random pitch for voice A
          AYldFineTuneA(freq & B11111111);
          AYldCourseTuneA(freq >>8);

          durA = random(1, dur) ;  // get random 8 bit duration for envelope A

          durA_count = durA;
          envA = 15;
          AYldAmpA(envA); //set voice A full on
   }



 // ------------Voice B from AY Chip---------------------

if (envB != 0){  //ramping down voice B envelope, 15 to 0

          if (durB_count != 0){  //wait for a count of durB
               durB_count -= 1;
          }
            else {  // when the count reaches zero decrement voice B envelope, reset count
               durB_count = durB;
               envB -= 1;
               AYldAmpB(envB);
            }
  }
  else{  // when envelope reaches zero, reset voice B with new frequency and envelope

          freq = getFreqB();  //get new random pitch for voice B
          AYldFineTuneB(freq & B11111111);
          AYldCourseTuneB(freq >>8);

          durB = random(1, dur) ;  // get random 8 bit duration for envelope B

          durB_count = durB;
          envB = 15;
          AYldAmpB(envB); //set voice B full on
   }


// ------------Voice C from AY Chip---------------------

if (envC != 0){  //ramping down voice C envelope, 15 to 0
```

```
        if (durC_count != 0){  //wait for a count of durC
            durC_count -= 1;
        }
          else {  // when the count reaches zero decrement voice C envelope, reset count
            durC_count = durC;
            envC -= 1;
            AYldAmpC(envC);
          }
 }
 else{  // when envelope reaches zero, reset voice C with new frequency and envelope

        freq = getFreqC();  //get new random pitch for voice C
        AYldFineTuneC(freq & B11111111);
        AYldCourseTuneC(freq >>8);

        durC = random(1, dur) ;  // get random 8 bit duration for envelope C

        durC_count = durC;
        envC = 15;
        AYldAmpC(envC); //set voice C full on
  }

 // ------------Switch2  Slows everything to almost a standstill---------------------

                    if (switch2 == 0){
                    delay(250);
  }
} // End of Loop

int getFreqA() {   // getting random frequency for Voices

   int basefreq = (slider1) + 10;
   int result = basefreq + random(slider4); // range of frequencies around the base
   return result;
}

int getFreqB() {   // getting random frequency for Voices

   int basefreq = (slider2) + 10;
   int result = basefreq + random(slider5); // range of frequencies around the base
   return result;
}
```

```
int getFreqC() {   // getting random frequency for Voices

   int basefreq = (slider3) + 10;
   int result = basefreq + random(slider6); // range of frequencies around the base
   return result;
}


// _______________________________________________________________
//            AY Address and Data Load Functions
// _______________________________________________________________

void AYloadAddress(int address){
 PORTA = (address & B1111);
 delayMicroseconds(2);

 digitalWrite(AYBDIR, HIGH);
 delayMicroseconds(2);

 digitalWrite(AYBDIR, LOW);
}

void AYloadData(int data){

 PORTA = data;

 delayMicroseconds(2);
 digitalWrite(AYBC2, HIGH);
 digitalWrite(AYBDIR, HIGH);
 delayMicroseconds(2);
 digitalWrite(AYBDIR, LOW);
 digitalWrite(AYBC2, LOW);
}

void AYloadDataShort(int data){
 PORTA = (data & B111111);

 delayMicroseconds(2);
 digitalWrite(AYBC2, HIGH);
 digitalWrite(AYBDIR, HIGH);
 delayMicroseconds(2);
 digitalWrite(AYBDIR, LOW);
 digitalWrite(AYBC2, LOW);
}
```

```
// ______________________________________________________________
//            AY General Control Register Load
// ______________________________________________________________

void AYldSynth(int address, int data){  // Load 4-bit Control Register Address then 8-bit data
 AYloadAddress(address);
 AYloadData(data);
}

// ________________________________________________________________
//            AY Individual Control Register Load
// _________________________________________________________________


void AYldFineTuneA(int data){  //8-bit fine tune A
 AYloadAddress(AYFineTuneA);
 AYloadData(data);
}

void AYldFineTuneB(int data){  //8-bit fine tune B
 AYloadAddress(AYFineTuneB);
 AYloadData(data);
}

void AYldFineTuneC(int data){  //8-bit fine tune C
 AYloadAddress(AYFineTuneC);
 AYloadData(data);
}

void AYldCourseTuneA(int data){  //4-bit course tune A
 AYloadAddress(AYCourseTuneA);
 AYloadDataShort(data);
}

void AYldCourseTuneB(int data){  //4-bit course tune B
 AYloadAddress(AYCourseTuneB);
 AYloadDataShort(data);
}

void AYldCourseTuneC(int data){  //4-bit course tune C
 AYloadAddress(AYCourseTuneC);
 AYloadDataShort(data);
}
```

```
void AYldNoisePeriod(int data){  //5-bit noise period tune
 AYloadAddress(AYNoisePeriod);
 AYloadDataShort(data);
}

void AYldEnable(int data){  //Low 1-bit enable, Noise C/B/A Tone C/B/A
 AYloadAddress(AYEnable);
 AYloadDataShort(data);
}

void AYldAmpA(int amp){  //4-bit amplitude of A if 0-15, else use Env if 16 (mode bit high)
 AYloadAddress(AYAmpA);
 AYloadDataShort(amp);
}

void AYldAmpB(int amp){  //4-bit amplitude of B if 0-15, else use Env if 16 (mode bit high)
 AYloadAddress(AYAmpB);
 AYloadDataShort(amp);
}

void AYldAmpC(int amp){  //4-bit amplitude of C if 0-15, else use Env if 16 (mode bit high)
 AYloadAddress(AYAmpC);
 AYloadDataShort(amp);
}

void AYldEnvFineTune(int data){  //8-bit Envelope Period fine tune
 AYloadAddress(AYEnvFineTune);
 AYloadData(data);
}

void AYldEnvCourseTune(int data){  //8-bit Envelope Period course tune
 AYloadAddress(AYEnvCourseTune);
 AYloadData(data);
}

void AYldEnvShape(int data){  //4-bit Envelope Shape. Continue/Attack/Alternate/Hold
 AYloadAddress(AYEnvShape);
 AYloadDataShort(data);
}
```

```
// ____________________________________________________
//              SID Basic Address and Data Functions
// ____________________________________________________

void loadAddress(int address){
 PORTB = address;
}

void loadData(int data){
  PORTC = data;
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);
}

void resetSID(){
  for(int i=0; i<25; i++){
    loadAddress(i);
    loadData(0);
  }
  ldModeVol(LP, 255);
  }

void readRegisters(){      //Collect values of all 4 SID readable registers
  DDRC = B00000000;       //Setup Data Lines as Inputs
  digitalWrite(SIDRW, HIGH);  // Setup Read/Write for a Data Read

  loadAddress(25);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  potX = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(28);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  env3 = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(27);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  osc3_rand = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);
  loadAddress(26);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  potY = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  digitalWrite(SIDRW, LOW);  // Reset Read/Write to Data Write
  DDRC = B11111111;       // Reset Data Lines as Outputs
}


void loadSensors(){      // load all current sensor values
    slider1 =   analogRead(Slider1) >> 2;
    slider2 =   analogRead(Slider2) >> 2;
    slider3 =   analogRead(Slider3) >> 2;
    slider4 =   analogRead(Slider4) >> 2;
    slider5 =   analogRead(Slider5) >> 2;
    slider6 =   analogRead(Slider6) >> 2;
    softPot =   analogRead(SoftPot) >> 2;
    frontPot =  analogRead(FrontPot) >> 2;
    lightSensor1 =   analogRead(LightSensor1) >> 2;
    lightSensor2 =   analogRead(LightSensor2) >> 2;
    lightSensor3 =   analogRead(LightSensor3) >> 2;
    lightSensor4 =   analogRead(LightSensor4) >> 2;
    lightSensor5 =   analogRead(LightSensor5) >> 2;
    lightSensor6 =   analogRead(LightSensor6) >> 2;
    switch1 = digitalRead(Switch1);
    switch2 = digitalRead(Switch2);
    switch3 = digitalRead(Switch3);
    switch4 = digitalRead(Switch4);
}
// ____________________________________________________________
//            SID Individual Control Register Load
// ____________________________________________________________

  /*      ldFreqLo, ldFreqHi,        --voices 1, 2, 3   Frequency
         ldPulseWLo, ldPulseWHi,    --voices 1, 2, 3   Pulse Width
         ldGate,                  --voices 1, 2, 3   Gate the Envelope (+ Waveshape)
         ldEnvAD, ldEnvSR,          --voices 1, 2, 3   Envelope ADSR
         ldFCLo, ldFCHi, ldResFilt  --Filter Cutoff Frequency/Resonance
         ldModeVol                --FilterType/OutputVolume
*/
```

```
void ldFreqLo(int voice){  //8-bit fine tune frequency -- FreqLo
 loadAddress(addr[voice]);
 loadData(FreqLo[voice] & 255);
}

void ldFreqHi(int voice){  //8-bit course tune frequency -- FreqHi
 loadAddress(addr[voice]+1);
 loadData(FreqHi[voice] & 255);
}

void ldPulseWLo(int voice){  //8-bit fine tune Pulse Width -- PulseWLo
 loadAddress(addr[voice]+2);
 loadData(PulseWLo[voice] & 255);
}

void ldPulseWHi(int voice){  //4-bit course tune Pulse Width -- PulseWHi
 loadAddress(addr[voice]+3);
 loadData(PulseWHi[voice] & 15);
}

void ldGate(int voice, int gate){  // Gates the ADSR Envelope (also loads Waveshape)

 loadAddress(addr[voice]+4);
 loadData(Waveshape[voice] + gate);
}

void ldEnvAD(int voice){  //4-bit Attack Time, 4-bit Decay time -- Env Attack/Decay
 loadAddress(addr[voice]+5);
 int x = ((Attack[voice] & 15) << 4) + (Decay[voice] & 15);
 loadData(x);
}


void ldEnvSR(int voice){  //4-bit Sustain Level, 4-bit Release time -- Env Sustain/Release
 loadAddress(addr[voice]+6);
 int x = ((Sustainx[voice] & 15) << 4) + (Release[voice] & 15);
 loadData(x);
}

void ldFCLo(int data){  //3-bit fine tune Filter Cutoff Frequency
 loadAddress(21);
 loadData(data & 7);
}

void ldFCHi(int data){  //8-bit course tune Filter Cutoff Frequency
 loadAddress(22);
 loadData(data & 255);
}
```

```
void ldFiltRes(int filt, int res){   //FILTEX/FILT3/FILT2/FILT1, 4-bit Filter Resonance,
  loadAddress(23);
  int x = ((res & 15) << 4) + (filt & 15);
  loadData(x);
}

void ldModeVol(int mode, int vol){   //Filter Type 3OFF/HP/BP/LP, 4-bit Output Volume
  loadAddress(24);
  loadData((vol & 15) + (mode & B11110000));
}
//-------------------------------------------------------
```

AYSID_SidTest2

```
/*
                SID Chip tested.
                Voice 2 Gated continuously On and tested with the Filter.

                slider1 -> Filter Cutoff Frequency
                slider2 -> Filter Resonance/Q
                slider3 -> Waveform Frequency
                slider4 -> Waveform Amplitude
                switch1 -> LowPass Filter on/off
                switch2 -> BandPass Filter on/off
                switch3 -> HighPass Filter on/off
                switch4 -> Noise/Sawtooth select
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

            Commodore SID Chip Controlled by an Arduino Micro
     3 Voice Synthesizer controlled through 29 8-bit registers and 3 Control lines

               8 bit Data on PortC
               5 Address Lines on Port B
               Chip Select (active low) on D5
               R/W (write low) on D6
               Clock from a 1MHz Oscillator chip
               Arduino and SID Reset lines tied together
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
//________________________________________________
//                  CONSTANTS and Variables
//________________________________________________

#include <MIDI.h>  // version 4.x.x
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// User created MIDI Callback functions here

 const int  MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
```

```
const int Slider2 = 1;
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;

//
// AY SYTHESIZER CONSTANTS
//
const int AYBDIR = 2;  // set up names for some Arduino pins
const int AYBC1 = 4;
const int AYBC2 = 3;

const int AYFineTuneA = 0;  // AY Synth Control Registers
const int AYFineTuneB = 2;
const int AYFineTuneC = 4;
const int AYCourseTuneA = 1;
const int AYCourseTuneB = 3;
const int AYCourseTuneC = 5;
const int AYNoisePeriod = 6;
const int AYEnable = 7;
const int AYAmpA = 8;
const int AYAmpB = 9;
const int AYAmpC = 10;
const int AYEnvFineTune = 11;
const int AYEnvCourseTune = 12;
const int AYEnvShape = 13;

const int VoiceD1 = 7;
const int VoiceD2 = 8;
const int VoiceD3 = 9;

//
//SID SYNTHESIZER CONSTANTS
//
const int SIDCS = 5;
const int SIDRW = 6;

//SID read register values

int potX = 0;
int potY = 0;
int osc3_rand = 0;
int env3 = 0;

int addr[4] = {0, 0, 7, 14};  //voice register address offsets 1, 2, 3
int v = 0;

// voice register values (Use voice = 1, 2, or 3.  Don't use zero)

int Attack[4] = {0, 0, 0, 0};      // (0 to 15)
int Decay[4] = {0, 0, 0, 0};       // (0 to 15)
int Sustainx[4] = {0, 15, 15, 15};  // (0 to 15)
int Release[4] = {0, 0, 0, 0};     // (0 to 15)
int FreqLo[4] = {0, 0, 0, 0};      // (0 to 255)
int FreqHi[4] = {0, 16, 16, 16};   // (0 to 255)
int PulseWLo[4] = {0, 0, 0, 0};    // (0 to 255)
int PulseWHi[4] = {0, 8, 8, 8};    // (0 to 15)
int Waveshape[4] = {0, 0, 0, 0};   // Load with the bit values below

// bit values for the voice Waveshape (Control) register

const int NOISE = 128;
const int PULSE = 64;
const int SAWTOOTH = 32;
const int TRIANGLE = 16;
const int TEST = 8;
const int RINGMOD = 4;
const int SYNC = 2;

// bit values for filt of ldResFilt, add the ones you want, zero for none

const int FILTEX = 8;  // send external signal through the Filter
const int FILT3 = 4;   // send Voice 3 through the Filter
const int FILT2 = 2;   // send Voice 2 through the Filter
const int FILT1 = 1;   // send Voice 3 through the Filter

// bit values for mode of ldModeVol, add the ones you want, zero for none

const int OFF3 = 128;  // no Voice3 in the output (when used in ring modulation)
const int HP = 64;     // set Filter tos High Pass
const int BP = 32;     // set Filter to BandPass
const int LP = 16;     // set Filter to LowPass

unsigned long timestamp;
unsigned long duration;

const int LED = 20;
int switchx = 0;

//______________________________________________
//                 SETUP()
//______________________________________________


void setup() {

delay(1000);

//MIDI Callback Handle references here
//MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);
```

```
pinMode(AYBDIR, OUTPUT);
digitalWrite(AYBDIR, LOW);
pinMode(AYBC1, OUTPUT);
digitalWrite(AYBC1, LOW);
pinMode(AYBC2, OUTPUT);
digitalWrite(AYBC2, LOW);

pinMode(SIDRW, OUTPUT);
digitalWrite(SIDRW, LOW);    // High for Read, Low for Write
pinMode(SIDCS, OUTPUT);
digitalWrite(SIDCS, HIGH); //Chip Select active low,Read or Write to register

pinMode(VoiceD1, OUTPUT);
pinMode(VoiceD2, OUTPUT);
pinMode(VoiceD3, OUTPUT);

pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;

resetSID();

// ~~~~~~~~~~~~~~~Setup Synth values~~~~~~~~~~~~~~~~~~~~~


ldModeVol(LP, 15);  // LowPass Filter, Output Volume at max

Attack[2] = 0;
Decay[2] = 0;
ldEnvAD(2);

Sustainx[2] = 15;
Release[2] = 0;
ldEnvSR(2);

Waveshape[2]=SAWTOOTH;
ldGate(2, 1);
```

```
//____________________________________________

} //End of Setup

//____________________________________________
//                    MAIN LOOP
//____________________________________________

void loop() {

 digitalWrite(LED, HIGH);
 int v = 2;
 int filter = 0;

 loadSensors();

if (switch4 != switchx){
 if (switch4){Waveshape[2]=NOISE;} else {Waveshape[2]=SAWTOOTH;}
 ldGate(2, 1);  //load Waveshape and Gate and leave it on
 switchx = switch4;
}

 ldFiltRes(FILT2, slider2 >>4); //Voice2 to filter, resonance on slider2
 filter = (switch1 * LP) + (switch2 * BP) + (switch3 * HP);
 ldModeVol(filter, slider4 >>4); //Filter type on switches, Amplitude on slider4

 ldFCHi(slider1 >>2); //Filter Cutoff Frequency
 FreqHi[v] = slider3; //Fill Voice Freq Matrix with value from slider
 ldFreqHi(v); //Course Tune Frequency

} //End of Main Loop



//____________________________________________
//               AY Address and Data Load Functions
//____________________________________________

void AYloadAddress(int address){
 PORTA = (address & B1111);
 delayMicroseconds(2);

 digitalWrite(AYBDIR, HIGH);
 delayMicroseconds(2);

 digitalWrite(AYBDIR, LOW);
}
```

```
void AYloadData(int data){

  PORTA = data;

  delayMicroseconds(2);
  digitalWrite(AYBC2, HIGH);
  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);
  digitalWrite(AYBDIR, LOW);
  digitalWrite(AYBC2, LOW);
}

void AYloadDataShort(int data){
  PORTA = (data & B111111);

  delayMicroseconds(2);
  digitalWrite(AYBC2, HIGH);
  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);
  digitalWrite(AYBDIR, LOW);
  digitalWrite(AYBC2, LOW);
}


//______________________________________________________
//              AY General Control Register Load
//______________________________________________________

void AYldSynth(int address, int data){  // Load 4-bit Control Register Address then 8-bit data
  AYloadAddress(address);
  AYloadData(data);
}
//______________________________________________________
//              AY Individual Control Register Load
//______________________________________________________


void AYldFineTuneA(int data){  //8-bit fine tune A
  AYloadAddress(AYFineTuneA);
  AYloadData(data);
}

void AYldFineTuneB(int data){  //8-bit fine tune B
  AYloadAddress(AYFineTuneB);
  AYloadData(data);
}
```

```
void AYldFineTuneC(int data){  //8-bit fine tune C
  AYloadAddress(AYFineTuneC);
  AYloadData(data);
}

void AYldCourseTuneA(int data){  //4-bit course tune A
  AYloadAddress(AYCourseTuneA);
  AYloadDataShort(data);
}

void AYldCourseTuneB(int data){  //4-bit course tune B
  AYloadAddress(AYCourseTuneB);
  AYloadDataShort(data);
}

void AYldCourseTuneC(int data){  //4-bit course tune C
  AYloadAddress(AYCourseTuneC);
  AYloadDataShort(data);
}

void AYldNoisePeriod(int data){  //5-bit noise period tune
  AYloadAddress(AYNoisePeriod);
  AYloadDataShort(data);
}

void AYldEnable(int data){  //Low 1-bit enable, Noise C/B/A Tone C/B/A
  AYloadAddress(AYEnable);
  AYloadDataShort(data);
}

void AYldAmpA(int amp){  //4-bit amplitude of A if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpA);
  AYloadDataShort(amp);
}

void AYldAmpB(int amp){  //4-bit amplitude of B if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpB);
  AYloadDataShort(amp);
}

void AYldAmpC(int amp){  //4-bit amplitude of C if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpC);
  AYloadDataShort(amp);
}
```

```
void AYldEnvFineTune(int data){  //8-bit Envelope Period fine tune
  AYloadAddress(AYEnvFineTune);
  AYloadData(data);
}

void AYldEnvCourseTune(int data){  //8-bit Envelope Period course tune
  AYloadAddress(AYEnvCourseTune);
  AYloadData(data);
}

void AYldEnvShape(int data){  //4-bit Envelope Shape. Continue/Attack/Alternate/Hold
  AYloadAddress(AYEnvShape);
  AYloadDataShort(data);
}


//____________________________________________________
//              SID Basic Address and Data Functions
//____________________________________________________

void loadAddress(int address){
 PORTB = address;
}

void loadData(int data){
  PORTC = data;
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);
}

void resetSID(){
  for(int i=0; i<25; i++){
    loadAddress(i);
    loadData(0);
  }
  ldModeVol(LP, 255);
  }

void readRegisters(){      //Collect values of all 4 SID readable registers
  DDRC = B00000000;       //Setup Data Lines as Inputs
  digitalWrite(SIDRW, HIGH);  // Setup Read/Write for a Data Read

  loadAddress(25);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  potX = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(28);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  env3 = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(27);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  osc3_rand = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(26);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  potY = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  digitalWrite(SIDRW, LOW);  // Reset Read/Write to Data Write
  DDRC = B11111111;       // Reset Data Lines as Outputs
}


void loadSensors(){      // load all current sensor values
    slider1 =   analogRead(Slider1) >> 2;
    slider2 =   analogRead(Slider2) >> 2;
    slider3 =   analogRead(Slider3) >> 2;
    slider4 =   analogRead(Slider4) >> 2;
    slider5 =   analogRead(Slider5) >> 2;
    slider6 =   analogRead(Slider6) >> 2;
    softPot =   analogRead(SoftPot) >> 2;
    frontPot =  analogRead(FrontPot) >> 2;
    lightSensor1 =   analogRead(LightSensor1) >> 2;
    lightSensor2 =   analogRead(LightSensor2) >> 2;
    lightSensor3 =   analogRead(LightSensor3) >> 2;
    lightSensor4 =   analogRead(LightSensor4) >> 2;
    lightSensor5 =   analogRead(LightSensor5) >> 2;
    lightSensor6 =   analogRead(LightSensor6) >> 2;
    switch1 = digitalRead(Switch1);
    switch2 = digitalRead(Switch2);
    switch3 = digitalRead(Switch3);
    switch4 = digitalRead(Switch4);
```

```
}
//______________________________________________
//          SID Individual Control Register Load
//______________________________________________

 /*    ldFreqLo, ldFreqHi,       --voices 1, 2, 3   Frequency
       ldPulseWLo, ldPulseWHi,   --voices 1, 2, 3   Pulse Width
       ldGate,                   --voices 1, 2, 3   Gate the Envelope (+ Waveshape)
       ldEnvAD, ldEnvSR,         --voices 1, 2, 3   Envelope ADSR
       ldFCLo, ldFCHi, ldResFilt  --Filter Cutoff Frequency/Resonance
       ldModeVol                 --FilterType/OutputVolume
*/

void ldFreqLo(int voice){  //8-bit fine tune frequency -- FreqLo
 loadAddress(addr[voice]);
 loadData(FreqLo[voice] & 255);
}

void ldFreqHi(int voice){  //8-bit course tune frequency -- FreqHi
 loadAddress(addr[voice]+1);
 loadData(FreqHi[voice] & 255);
}

void ldPulseWLo(int voice){  //8-bit fine tune Pulse Width -- PulseWLo
 loadAddress(addr[voice]+2);
 loadData(PulseWLo[voice] & 255);
}

void ldPulseWHi(int voice){  //4-bit course tune Pulse Width -- PulseWHi
 loadAddress(addr[voice]+3);
 loadData(PulseWHi[voice] & 15);
}

void ldGate(int voice, int x){  // Gates the ADSR Envelope (also loads Waveshape)

 loadAddress(addr[voice]+4);
 loadData(Waveshape[voice] + x);
}

void ldEnvAD(int voice){  //4-bit Attack Time, 4-bit Decay time -- Env Attack/Decay
 loadAddress(addr[voice]+5);
 int x = ((Attack[voice] & 15) << 4) + (Decay[voice] & 15);
 loadData(x);
}
```

```
void ldEnvSR(int voice){  //4-bit Sustain Level, 4-bit Release time -- Env Sustain/Release
 loadAddress(addr[voice]+6);
 int x = ((Sustainx[voice] & 15) << 4) + (Release[voice] & 15);
 loadData(x);
}

void ldFCLo(int data){  //3-bit fine tune Filter Cutoff Frequency
 loadAddress(21);
 loadData(data & 7);
}

void ldFCHi(int data){  //8-bit course tune Filter Cutoff Frequency
 loadAddress(22);
 loadData(data & 255);
}

void ldFiltRes(int filt, int res){   //FILTEX/FILT3/FILT2/FILT1, 4-bit Filter Resonance,
 loadAddress(23);
 int x = ((res & 15) << 4) + (filt & 15);
 loadData(x);
}

void ldModeVol(int mode, int vol){   //Filter Type 3OFF/HP/BP/LP, 4-bit Output Volume
 loadAddress(24);
 loadData((vol & 15) + (mode & B11110000));
}
```

```
/*
      Full functioning AY Synth Chip with MIDI IN

      ** MIDI Input Note ON and Note OFF handled by Callback function
         MIDI Note Velocity sets AY note Amplitude. 3 voice polyphony with AY Voices
      ** Slider1 controls Noise frequency, Noise turns on with slider1 above zero
      ** Sliders 4, 5, and 6 act as pitch wheels and detuning for AY voices A, B, and C
      ** Tap bottom of SoftPot - All 3 AY voices get levels from their AMP Controls
         Tap Middle of SoftPot - VoiceA is on the AY Envelope, Voices B and C on AMP Control
         Tap Top of SoftPot - All 3 AY voices on the AY Envelope
      ** Slider2 and Slider3 control Envelope Period, Course and Fine.
      ** Switches 1-4 set the AY Envelope type.
               Switch 1 - set to zero for single attack on MIDI Note ON
               Switch 2 - 0 is instant attack then decay, 1 is slow attack
               Switch 3 - 0 for Sawtooth, 1 for Triangle
               Switch 4 - 0 for repeating, 1 for single
*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                      CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//
// For use on an Arduino Mega.  MIDI set up on Serial1, pins 18 and 19.

#include <MIDI.h>  // version 4.x.x
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// User created MIDI Callback functions here or at end

 const int MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
```

```
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;
```

```
boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;

//
// AY SYTHESIZER CONSTANTS
//
const int AYBDIR = 2;  // set up names for some Arduino pins
const int AYBC1 = 4;
const int AYBC2 = 3;

const int AYFineTuneA = 0;  // AY Synth Control Registers
const int AYFineTuneB = 2;
const int AYFineTuneC = 4;
const int AYCourseTuneA = 1;
const int AYCourseTuneB = 3;
const int AYCourseTuneC = 5;
const int AYNoisePeriod = 6;
const int AYEnable = 7;
const int AYAmpA = 8;
const int AYAmpB = 9;
const int AYAmpC = 10;
const int AYEnvFineTune = 11;
const int AYEnvCourseTune = 12;
const int AYEnvShape = 13;

const int VoiceD1 = 7;
const int VoiceD2 = 8;
const int VoiceD3 = 9;

const int LED = 20;
int switchx = 0;

//AY frequency values for equal temperment A440 MIDI NoteON commands
```

```
byte AYMidiNoteHi[128] = {
  15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
  14, 14, 13, 12, 11, 11, 10, 9, 9, 8, 8, 7,
  7, 7, 6, 6, 5, 5, 5, 4, 4, 4, 4, 3,
  3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0
};

byte AYMidiNoteLow[128] = {
  211, 211, 211, 211, 211, 211, 211, 211, 211, 211, 211, 211,
  239, 25, 78, 141, 218, 47, 143, 247, 104, 225, 97, 233,
  119, 12, 167, 71, 237, 152, 71, 252, 180, 112, 49, 244,
  188, 134, 83, 36, 246, 204, 164, 126, 90, 56, 24, 250,
  222, 195, 170, 146, 123, 102, 82, 63, 45, 28, 12, 253,
  239, 225, 213, 201, 190, 179, 169, 159, 150, 142, 134, 127,
  119, 113, 106, 100, 95, 89, 84, 80, 75, 71, 67, 63,
  60, 56, 53, 50, 47, 45, 42, 40, 38, 36, 34, 32,
  30, 28, 27, 25, 24, 22, 21, 20, 19, 18, 17, 16,
  15, 14, 13, 13, 12, 11, 11, 10, 9, 9, 8, 8,
  7, 7, 7, 6, 6, 6, 5, 5
};

//current note on's for voices A, B, and C.  Use note value 0 for "not on".
byte CurrentNoteOn[3] = {0, 0, 0};

byte Mode[3] = { 0, 0, 0 };  // sets up volume by 4-bit Amp, or Envelopes

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                      SETUP()
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
void setup() {

delay(1000);

//MIDI Callback Handle references here
//MIDI.begin(MIDI_CHANNEL_OMNI);

  MIDI.setHandleNoteOn(myHandleNoteOn); //for Callback MIDI In
  MIDI.setHandleNoteOff(myHandleNoteOff); //for Callback MIDI In
  MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);

pinMode(AYBDIR, OUTPUT);
digitalWrite(AYBDIR, LOW);
pinMode(AYBC1, OUTPUT);
digitalWrite(AYBC1, LOW);
pinMode(AYBC2, OUTPUT);
digitalWrite(AYBC2, LOW);

pinMode(VoiceD1, OUTPUT);
digitalWrite(VoiceD1, LOW);
pinMode(VoiceD2, OUTPUT);
digitalWrite(VoiceD2, LOW);
pinMode(VoiceD3, OUTPUT);
digitalWrite(VoiceD3, LOW);


pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
```

```
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;

                    AYldEnable(B111000);  //Enable only tones (low enable)

 Serial.begin(9600);

} //End of Setup

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                      MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loop() {

loadSensors();
tone(VoiceD1, (frontPot >> 2));  // set frequency of Voice D


// -----------slider1 turns on noise, noise freq on slider1  ------------------

                if (slider1 > 10){ AYldEnable(B000000);  // add noise to all 3 voices
                                   AYldNoisePeriod(slider1 >> 5); //Noise Frequency
                }
                else {AYldEnable(B111000);  } //Enable only tones (low enable)

// ----------------add note glide/detune on each voice--------------------------

                 AYldFineTuneA(AYMidiNoteLow[CurrentNoteOn[0]] - (slider4 >> 3) );
                 AYldFineTuneB(AYMidiNoteLow[CurrentNoteOn[1]] - (slider5 >> 3) );
                 AYldFineTuneC(AYMidiNoteLow[CurrentNoteOn[2]] - (slider6 >> 3) );
```

```
// -----------------Envelopes ---------------------------------------------------

       if (softPot > 15 && softPot < 200) { //Serial.print(softPot); Serial.println(" bottom");
                                          Mode[0]=0; Mode[1]=0; Mode[2]=0; delay(1000); }
       else if (softPot > 400 && softPot < 600){ //Serial.print(softPot); Serial.println( "
middle");
                                               Mode[0] = 16; Mode[1]=0; Mode[2]=0; delay(1000); }
       else if (softPot > 800) { //Serial.print(softPot); Serial.println(" top");
                                 Mode[0]=16; Mode[1]=16; Mode[2]=16; delay(1000); }

            AYldEnvFineTune(slider3 >> 2);
            AYldEnvCourseTune(slider2 >> 2);
// -------------------------------------------------------------------------------

// On MIDI.read() MIDI class will call Callback functions.
// User created callback function myHandleNoteOn() and myHandleNoteOff() at end
// MIDI.setHandleNoteOn(myHandleNoteOn), MIDI.setHandleNoteOff(myHandleNoteOff) in setup()

MIDI.read();

} //End of Main Loop


void loadSensors(){      // load all current sensor values
      slider1 =   analogRead(Slider1);
      slider2 =   analogRead(Slider2);
      slider3 =   analogRead(Slider3);
      slider4 =   analogRead(Slider4);
      slider5 =   analogRead(Slider5);
      slider6 =   analogRead(Slider6);
      softPot =   analogRead(SoftPot);
      frontPot =  analogRead(FrontPot);
      lightSensor1 =   analogRead(LightSensor1) >> 2;
      lightSensor2 =   analogRead(LightSensor2) >> 2;
      lightSensor3 =   analogRead(LightSensor3) >> 2;
      lightSensor4 =   analogRead(LightSensor4) >> 2;
      lightSensor5 =   analogRead(LightSensor5) >> 2;
      lightSensor6 =   analogRead(LightSensor6) >> 2;
      switch1 = digitalRead(Switch1);
      switch2 = digitalRead(Switch2);
      switch3 = digitalRead(Switch3);
      switch4 = digitalRead(Switch4);
}

//  _________________________________________________________________
//                AY Address and Data Load Functions
//  _________________________________________________________________

void AYloadAddress(int address){
  PORTA = (address & B1111);
  delayMicroseconds(2);

  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);

  digitalWrite(AYBDIR, LOW);
}

void AYloadData(int data){

  PORTA = data;

  delayMicroseconds(2);
  digitalWrite(AYBC2, HIGH);
  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);
  digitalWrite(AYBDIR, LOW);
  digitalWrite(AYBC2, LOW);
}

void AYloadDataShort(int data){
  PORTA = (data & B111111);

  delayMicroseconds(2);
  digitalWrite(AYBC2, HIGH);
  digitalWrite(AYBDIR, HIGH);
  delayMicroseconds(2);
  digitalWrite(AYBDIR, LOW);
  digitalWrite(AYBC2, LOW);
}


//  ____________________________________________________________________
//                AY General Control Register Load
//  ____________________________________________________________________

void AYldSynth(int address, int data){  // Load 4-bit Control Register Address then 8-bit data
  AYloadAddress(address);
  AYloadData(data);
}

//  ______________________________________________________________________
//                AY Individual Control Register Load
//  _______________________________________________________________________


void AYldFineTuneA(int data){  //8-bit fine tune A
  AYloadAddress(AYFineTuneA);
  AYloadData(data);
}

void AYldFineTuneB(int data){  //8-bit fine tune B
  AYloadAddress(AYFineTuneB);
  AYloadData(data);
}

void AYldFineTuneC(int data){  //8-bit fine tune C
```

```
  AYloadAddress(AYFineTuneC);
  AYloadData(data);
}

void AYldCourseTuneA(int data){  //4-bit course tune A
  AYloadAddress(AYCourseTuneA);
  AYloadDataShort(data);
}

void AYldCourseTuneB(int data){  //4-bit course tune B
  AYloadAddress(AYCourseTuneB);
  AYloadDataShort(data);
}

void AYldCourseTuneC(int data){  //4-bit course tune C
  AYloadAddress(AYCourseTuneC);
  AYloadDataShort(data);
}

void AYldNoisePeriod(int data){  //5-bit noise period tune
  AYloadAddress(AYNoisePeriod);
  AYloadDataShort(data);
}

void AYldEnable(int data){  //Low 1-bit enable, Noise C/B/A Tone C/B/A
  AYloadAddress(AYEnable);
  AYloadDataShort(data);
}

void AYldAmpA(int amp){  //4-bit amplitude of A if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpA);
  AYloadDataShort(amp);
}

void AYldAmpB(int amp){  //4-bit amplitude of B if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpB);
  AYloadDataShort(amp);
}

void AYldAmpC(int amp){  //4-bit amplitude of C if 0-15, else use Env if 16 (mode bit high)
  AYloadAddress(AYAmpC);
  AYloadDataShort(amp);
}

void AYldEnvFineTune(int data){  //8-bit Envelope Period fine tune
  AYloadAddress(AYEnvFineTune);
  AYloadData(data);
}

void AYldEnvCourseTune(int data){  //8-bit Envelope Period course tune
  AYloadAddress(AYEnvCourseTune);
  AYloadData(data);
}

void AYldEnvShape(int data){  //4-bit Envelope Shape. Continue/Attack/Alternate/Hold
  AYloadAddress(AYEnvShape);
  AYloadDataShort(data);
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                      Callback MIDI_In Handles
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// user created Callback Function for MIDI Input Test

 void myHandleNoteOn(byte channel, byte note, byte velocity){

    for (int x = 0; x < 3; x ++){    // search through the 3 AY voices

    if (CurrentNoteOn[x] == 0){      // this AY voice is available

                 switch (x) {
```

```
                 case 0:
                   AYldCourseTuneA(AYMidiNoteHi[note]);
                   AYldFineTuneA(AYMidiNoteLow[note]);
                   velocity = ((velocity >> 4) + 6) & 15;
                   AYldAmpA(velocity + Mode[0]);
                   AYldEnvShape(!switch4 + (!switch3 * 2) + (!switch2 * 4) + (!switch1 * 8));
                   if (velocity == 0) {CurrentNoteOn[x] = 0; }
                   else {CurrentNoteOn[x] = note; }
                   break;
                case 1:
                   AYldCourseTuneB(AYMidiNoteHi[note]);
                   AYldFineTuneB(AYMidiNoteLow[note]);
                   velocity = ((velocity >> 4) + 6) & 15;
                   AYldAmpB(velocity + Mode[1]);
                   AYldEnvShape(!switch4 + (!switch3 * 2) + (!switch2 * 4) + (!switch1 * 8));
                   if (velocity == 0) {CurrentNoteOn[x] = 0; }
                   else {CurrentNoteOn[x] = note; }
                   break;
                case 2:
                   AYldCourseTuneC(AYMidiNoteHi[note]);
                   AYldFineTuneC(AYMidiNoteLow[note]);
                   velocity = ((velocity >> 4) + 6) & 15;
                   AYldAmpC(velocity + Mode[2]);
                   AYldEnvShape(!switch4 + (!switch3 * 2) + (!switch2 * 4) + (!switch1 * 8));
                   if (velocity == 0) {CurrentNoteOn[x] = 0; }
                   else {CurrentNoteOn[x] = note; }
                   break;
                 }  // end of switch

                 break;  //  break out of loop

     }  // end of Note On check


}  // end of for loop
}  // end of Handle
```

```
 void myHandleNoteOff(byte channel, byte note, byte velocity){
 for (int x = 0; x < 3; x ++){          // search through the 3 AY voices

    if (CurrentNoteOn[x] == note){      // which voice is playing the note to turn off?

                 switch (x) {
                 case 0:
                   AYldAmpA(0);
                   CurrentNoteOn[x] = 0;
                   break;
                case 1:
                   AYldAmpB(0);
                   CurrentNoteOn[x] = 0;
                   break;
                case 2:
                  AYldAmpC(0);
                   CurrentNoteOn[x] = 0;
                   break;
                 }  // end of switch

     }  // end of Note check


}  // end of for loop
}  //end of Handle
```

```
/*
                       Full functioning SID Synth Chip with MIDI IN

      ** MIDI Input Note ON and Note OFF handled by Callback function
         2 voice polyphony with Voices 1 and 2.
      ** Slider1 - frequency offset for Voices 1
      ** Slider2 - frequency offset for Voices 2
      ** Slider3 - frequency offset for Modulating Voice 3 (course tune)
      ** Slider4 - Pulse width Voice 1,
      ** Slider5 - Pulse width Voice 2, Voice3 frequency Fine
      ** Slider6 - not used

      ** Tap bottom of SoftPot - Voice 1 gets waveform from switches and sliders 5 & 6
                                 ADSR envelope from Sliders 1,2,3,4
         Tap Middle of SoftPot - Voice 2 gets waveform from switches and sliders 5 & 6
                                 ADSR envelope from Sliders 1,2,3,4
         Tap Top of SoftPot - Get from switches - Voice1 Sync, Ring Mod and Voice 3 Sync, Ring 
Mod
                              Get from Sliders 1,2,3 - Filter type HP, BP, LP  (on or off)
                              Get from Sliders 4,5,6 - Filter on Voice 1,2,3, External (on or 
off)

      ** Switches 1-4 set the waveform type (noise, pulse, saw, triangle)
         RINGMOD on slider5, SYNC on slider6, for voices 1 and 2.
         Values entered with softpot.

      ** PotX sets filter cutoff frequency, PotY sets filter resonance
      ** 3OFF = 1, no modulation voice 3 in output
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
//____________________________________________________
//                      CONSTANTS and Variables
//____________________________________________________

#include <MIDI.h>  // version 4.x.x
```

```
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

// User created MIDI Callback functions here

 const int  MIDI_TX = 18;  //on SERIAL1
 const int MIDI_RX = 19;  //on SERIAL1
//
// ANALOG INPUTS
//
const int Slider1 = 0;
const int Slider2 = 1;
const int Slider3 = 2;
const int Slider4 = 3;
const int Slider5 = 4;
const int Slider6 = 5;
const int SoftPot = 6;
const int FrontPot = 7;
const int LightSensor1 = 8;
const int LightSensor2 = 9;
const int LightSensor3 = 10;
const int LightSensor4 = 11;
const int LightSensor5 = 12;
const int LightSensor6 = 13;

int slider1 = 0;
int slider2 = 0;
int slider3 = 0;
int slider4 = 0;
int slider5 = 0;
int slider6 = 0;
int softPot = 0;
int frontPot = 0;
int lightSensor1 = 0;
int lightSensor2 = 0;
int lightSensor3 = 0;
int lightSensor4 = 0;
int lightSensor5 = 0;
int lightSensor6 = 0;

//
//DIGIITAL SWITCHES
//
const int Switch1 = 14;
const int Switch2 = 15;
const int Switch3 = 16;
const int Switch4 = 17;

boolean switch1 = 0;
boolean switch2 = 0;
boolean switch3 = 0;
boolean switch4 = 0;



const int VoiceD1 = 7;
const int VoiceD2 = 8;
const int VoiceD3 = 9;

//
//SID SYNTHESIZER CONSTANTS
//
const int SIDCS = 5;
const int SIDRW = 6;

//SID read register values

int potX = 0;
int potY = 0;
int osc3_rand = 0;
int env3 = 0;

int addr[4] = {0, 0, 7, 14};  //voice register address offsets 1, 2, 3
int v = 0;

// voice register values (Use voice = 1, 2, or 3.  Don't use zero)

int Attack[4] = {0, 0, 0, 0};      // (0 to 15)
int Decay[4] = {0, 0, 0, 0};       // (0 to 15)
int Sustainx[4] = {0, 15, 15, 15};  // (0 to 15)
int Release[4] = {0, 0, 0, 0};     // (0 to 15)
int FreqLo[4] = {0, 0, 0, 0};      // (0 to 255)
int FreqHi[4] = {0, 16, 16, 16};   // (0 to 255)
int PulseWLo[4] = {0, 0, 0, 0};    // (0 to 255)
int PulseWHi[4] = {0, 8, 8, 8};    // (0 to 15)
int Waveshape[4] = {0, 0, 0, 0};   // Load with the bit values below

// bit values for the voice Waveshape (Control) register

const int NOISE = 128;
const int PULSE = 64;
const int SAWTOOTH = 32;
const int TRIANGLE = 16;
const int TEST = 8;
const int RINGMOD = 4;
const int SYNC = 2;

// bit values for filt of ldResFilt, add the ones you want, zero for none

const int FILTEX = 8;  // send external signal through the Filter
const int FILT3 = 4;   // send Voice 3 through the Filter
const int FILT2 = 2;   // send Voice 2 through the Filter
const int FILT1 = 1;   // send Voice 3 through the Filter

int filt = 0;

// bit values for mode of ldModeVol, add the ones you want, zero for none

const int OFF3 = 128;  // no Voice3 in the output (when used in ring modulation)
const int HP = 64;     // set Filter tos High Pass
const int BP = 32;     // set Filter to BandPass
const int LP = 16;     // set Filter to LowPass
bool hp = 0;
bool bp = 0;
bool lp = 0;
int res = 0;


const int LED = 20;

//SID frequency values for equal temperment A440 MIDI NoteON commands

byte SIDMidiNoteHi[128] = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
  2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4,
  4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 8,
  8, 9, 9, 10, 10, 11, 12, 12, 13, 14, 15, 16,
  17, 18, 19, 20, 21, 22, 24, 25, 27, 28, 30, 32,
  34, 36, 38, 40, 43, 45, 48, 51, 54, 57, 61, 64,
  68, 72, 76, 81, 86, 91, 96, 102, 108, 115, 122, 129,
  137, 145, 153, 163, 172, 183, 193, 205, 217, 230, 244, 244,
  244, 244, 244, 244, 244, 244, 244, 244, 244, 244, 244, 244,
  244, 244, 244, 244, 244, 244, 244, 244
};

byte SIDMidiNoteLow[128] = {
  137, 145, 154, 163, 173, 183, 194, 206, 218, 231, 244, 3,
  18, 35, 52, 70, 90, 110, 132, 155, 180, 205, 233, 6,
  37, 69, 104, 141, 179, 220, 8, 54, 103, 155, 210, 12,
  73, 139, 208, 25, 103, 185, 16, 108, 206, 54, 163, 24,
  147, 21, 160, 50, 205, 114, 32, 217, 156, 107, 71, 47,
  38, 43, 63, 100, 155, 228, 64, 177, 56, 215, 141, 95,
  75, 86, 127, 200, 53, 199, 128, 98, 112, 173, 27, 189,
  151, 171, 253, 145, 107, 142, 255, 196, 225, 90, 54, 122,
  45, 86, 250, 34, 213, 28, 255, 137, 194, 180, 108, 108,
  108, 108, 108, 108, 108, 108, 108, 108, 108, 108, 108, 108,
  108, 108, 108, 108, 108, 108, 108, 108
};
 //current note on's for voices A, B, and C.  Use note value 0 for "not on".
byte CurrentNoteOn[3] = {0, 0, 0}; //holds voice note values even during Release
byte VoiceOn[3] = {0, 0, 0};  // reflects voice off and on from MIDI NoteON/Off

byte Mode[3] = { 0, 0, 0 };  // sets up volume by 4-bit Amp, or Envelopes

//____________________________________________________
//                      SETUP()
//____________________________________________________


void setup() {

delay(1000);

//MIDI Callback Handle references here
//MIDI.begin(MIDI_CHANNEL_OMNI);

  MIDI.setHandleNoteOn(myHandleNoteOn); //for Callback MIDI In
  MIDI.setHandleNoteOff(myHandleNoteOff); //for Callback MIDI In
  MIDI.begin(MIDI_CHANNEL_OMNI);

pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);


pinMode(SIDRW, OUTPUT);
digitalWrite(SIDRW, LOW);    // High for Read, Low for Write
pinMode(SIDCS, OUTPUT);
digitalWrite(SIDCS, HIGH); //Chip Select active low,Read or Write to register

pinMode(VoiceD1, OUTPUT);
pinMode(VoiceD2, OUTPUT);
pinMode(VoiceD3, OUTPUT);

pinMode(Switch1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(Switch1, HIGH);
pinMode(Switch2, INPUT);
digitalWrite(Switch2, HIGH);
pinMode(Switch3, INPUT);
digitalWrite(Switch3, HIGH);
pinMode(Switch4, INPUT);
digitalWrite(Switch4, HIGH);

DDRA = B11111111;  // outputs
DDRB = B11111111;
DDRC = B11111111;

resetSID();

// ~~~~~~~~~~~~~~Setup Synth values~~~~~~~~~~~~~~~~~~~~


ldModeVol(LP, 15);  // LowPass Filter, Output Volume at max

Attack[1] = 0;
Decay[1] = 0;
ldEnvAD(1);
Sustainx[1] = 15;
Release[1] = 0;
ldEnvSR(1);
Waveshape[1] = TRIANGLE;

Attack[2] = 0;
Decay[2] = 0;
ldEnvAD(2);
Sustainx[2] = 15;
Release[2] = 0;
```

```
ldEnvSR(2);
Waveshape[2] = TRIANGLE;

Attack[3] = 0;
Decay[3] = 0;
ldEnvAD(3);
Sustainx[3] = 15;
Release[3] = 0;
ldEnvSR(3);
Waveshape[3] = TRIANGLE;

ldFiltRes(8, 0);
ldModeVol(OFF3, 15);

Serial.begin(9600);

digitalWrite(LED, HIGH);

//__________________________

} //End of Setup

//__________________________
//          MAIN LOOP
//__________________________

void loop() {

  digitalWrite(LED, HIGH);

  loadSensors();
  readRegisters();
  tone(VoiceD1, (frontPot << 1));  // set frequency of Voice D

// Frequency offsets for Voices 1 and 2 from Sliders 1 and 2
```

```
  FreqHi[1] = SIDMidiNoteHi[CurrentNoteOn[0]] + (slider1 >> 3) ;
  ldFreqHi(1);
  FreqHi[2] = SIDMidiNoteHi[CurrentNoteOn[1]] + (slider2 >> 3) ;
  ldFreqHi(2);

// Course and Fine tune of Modulating Voice 3 from Sliders 3 and 6

  FreqLo[3]=slider5 >> 2;
  ldFreqLo(3);
  FreqHi[3] = (slider3 >> 2) + (slider5 >> 8);
  ldFreqHi(3);

// Pulse width of Voices 1 and 2 (12 bits) from Sliders 4 and 5 (10 bits)

  PulseWHi[1] = slider4 >> 6;
  ldPulseWHi(1);
  PulseWLo[1] = slider4 << 2;
  ldPulseWLo(1);

  PulseWHi[2] = slider5 >> 6;
  ldPulseWHi(2);
  PulseWLo[2] = slider5 << 2;
  ldPulseWLo(2);

  // Filter cutoff Frequency (11 bits, ignore lower 3) from potX (8 bits)
  ldFCHi(potX);

 //SoftPot used for loading Voice Waveforms, Envelopes, Sync/RingMod modes, Filter Modes

   if (softPot > 15 && softPot < 200) {  //tap bottom of pot
   // Serial.print(softPot);
   // Serial.println(" bottom");
     Waveshape[1] = 0;
     if (switch1){
     Waveshape[1]=(PULSE * !switch2)+(SAWTOOTH * !switch3)+(TRIANGLE * !switch4);}
     else { Waveshape[1] = NOISE; }
     if (slider5 > 20) {Waveshape[1] = TRIANGLE + RINGMOD;}
     if (slider6 > 20) {Waveshape[1] = Waveshape[1] + SYNC;}

     Attack[1] = slider1 >> 6;
     Decay[1] = slider2 >> 6;
     Sustainx[1] = slider3 >> 6;
     Release[1] = slider4 >> 6;
     ldEnvAD(1);
     ldEnvSR(1);
     delay(1000);
     }

  else if ((softPot > 400) && (softPot < 600)){  //tap middle of pot
   //Serial.print(softPot);
  // Serial.println( " middle");
     Waveshape[2] = 0;
     if (switch1){
Waveshape[2]=(PULSE * !switch2)+(SAWTOOTH * !switch3)+(TRIANGLE * !switch4);}
else { Waveshape[2] = NOISE; }
     if (slider5 > 20) {Waveshape[2] = TRIANGLE + RINGMOD;}
     if (slider6 > 20) {Waveshape[2] = Waveshape[2] + SYNC;}

     Attack[2] = slider1 >> 6;
     Decay[2] = slider2 >> 6;
     Sustainx[2] = slider3 >> 6;
     Release[2] = slider4 >> 6;
     ldEnvAD(2);
     ldEnvSR(2);
     delay(1000);
     }

  else if (softPot > 800) {  //tap top of pot
   //Serial.print(softPot);
   //Serial.println(" top");

     if (slider1 > 10) {hp=1;} else {hp=0;}
```

```
          if (slider2 > 10) {bp=1;} else {bp=0;}
          if (slider3 > 10) {lp=1;} else {lp=0;}

          filt = (!switch1 * FILT1) + (!switch2 * FILT2) + (!switch3 * FILT3) + (!switch4 *
FILTEX);
          ldFiltRes(filt, potY >> 4);
          ldModeVol( (hp * HP) + (lp * LP) + (bp * BP) + OFF3, 15);
          delay(1000);
          }


  MIDI.read();

} //End of Main Loop

//____________________________________________________
//                SID Basic Address and Data Functions
//____________________________________________________

void loadAddress(int address){
 PORTB = address;
}

void loadData(int data){
  PORTC = data;
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);
}

void resetSID(){
  for(int i=0; i<25; i++){
    loadAddress(i);
    loadData(0);
  }
  ldModeVol(LP, 255);
  }

void readRegisters(){      //Collect values of all 4 SID readable registers
  DDRC = B00000000;        //Setup Data Lines as Inputs
  digitalWrite(SIDRW, HIGH);  // Setup Read/Write for a Data Read

  loadAddress(25);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  potX = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(28);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  env3 = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(27);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  osc3_rand = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  loadAddress(26);
  digitalWrite(SIDCS, LOW);
  delayMicroseconds(3);
  potY = PINC;
  delayMicroseconds(3);
  digitalWrite(SIDCS, HIGH);

  digitalWrite(SIDRW, LOW);  // Reset Read/Write to Data Write
  DDRC = B11111111;       // Reset Data Lines as Outputs
```

```
}


void loadSensors(){      // load all current sensor values
      slider1 =   analogRead(Slider1);
      slider2 =   analogRead(Slider2);
      slider3 =   analogRead(Slider3);
      slider4 =   analogRead(Slider4);
      slider5 =   analogRead(Slider5);
      slider6 =   analogRead(Slider6);
      softPot =   analogRead(SoftPot);
      frontPot =  analogRead(FrontPot);
      lightSensor1 =   analogRead(LightSensor1) >> 2;
      lightSensor2 =   analogRead(LightSensor2) >> 2;
      lightSensor3 =   analogRead(LightSensor3) >> 2;
      lightSensor4 =   analogRead(LightSensor4) >> 2;
      lightSensor5 =   analogRead(LightSensor5) >> 2;
      lightSensor6 =   analogRead(LightSensor6) >> 2;
      switch1 = digitalRead(Switch1);
      switch2 = digitalRead(Switch2);
      switch3 = digitalRead(Switch3);
      switch4 = digitalRead(Switch4);
}
//____________________________________________________
//               SID Individual Control Register Load
//____________________________________________________

  /*       ldFreqLo, ldFreqHi,        --voices 1, 2, 3   Frequency
           ldPulseWLo, ldPulseWHi,    --voices 1, 2, 3   Pulse Width
           ldGate,                    --voices 1, 2, 3   Gate the Envelope (+ Waveshape)
           ldEnvAD, ldEnvSR,          --voices 1, 2, 3   Envelope ADSR
           ldFCLo, ldFCHi, ldResFilt  --Filter Cutoff Frequency/Resonance
           ldModeVol                  --FilterType/OutputVolume
*/

void ldFreqLo(int voice){  //8-bit fine tune frequency -- FreqLo
```

```
  loadAddress(addr[voice]);
  loadData(FreqLo[voice] & 255);
}

void ldFreqHi(int voice){  //8-bit course tune frequency -- FreqHi
  loadAddress(addr[voice]+1);
  loadData(FreqHi[voice] & 255);
}

void ldPulseWLo(int voice){  //8-bit fine tune Pulse Width -- PulseWLo
  loadAddress(addr[voice]+2);
  loadData(PulseWLo[voice] & 255);
}

void ldPulseWHi(int voice){  //4-bit course tune Pulse Width -- PulseWHi
  loadAddress(addr[voice]+3);
  loadData(PulseWHi[voice] & 15);
}

void ldGate(int voice, int x){  // Gates the ADSR Envelope (also loads Waveshape)

  loadAddress(addr[voice]+4);
  loadData(Waveshape[voice] + x);
}

void ldEnvAD(int voice){  //4-bit Attack Time, 4-bit Decay time -- Env Attack/Decay
  loadAddress(addr[voice]+5);
  int x = ((Attack[voice] & 15) << 4) + (Decay[voice] & 15);
  loadData(x);
}


void ldEnvSR(int voice){  //4-bit Sustain Level, 4-bit Release time -- Env Sustain/Release
  loadAddress(addr[voice]+6);
  int x = ((Sustainx[voice] & 15) << 4) + (Release[voice] & 15);
  loadData(x);
```

```
}

void ldFCLo(int data){  //3-bit fine tune Filter Cutoff Frequency
  loadAddress(21);
  loadData(data & 7);
}

void ldFCHi(int data){  //8-bit course tune Filter Cutoff Frequency
  loadAddress(22);
  loadData(data & 255);
}

void ldFiltRes(int filt, int res){   //FILTEX/FILT3/FILT2/FILT1, 4-bit Filter Resonance,
  loadAddress(23);
  int x = ((res & 15) << 4) + (filt & 15);
  loadData(x);
}

void ldModeVol(int mode, int vol){   //Filter Type 3OFF/HP/BP/LP, 4-bit Output Volume
  loadAddress(24);
  loadData((vol & 15) + (mode & B11110000));
}


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                      Callback MIDI_In Handles
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// user created Callback Function for MIDI Input Test

 void myHandleNoteOn(byte channel, byte note, byte velocity){

    for (int x = 0; x < 2; x ++){    // search through the 2 SID voices

    if (VoiceOn[x] == 0){      // this SID voice is available
          if (velocity > 0){
```

```
       switch (x) {
       case 0:
         FreqHi[1]=SIDMidiNoteHi[note];
         ldFreqHi(1);
         FreqLo[1]=SIDMidiNoteLow[note];
         ldFreqLo(1);
         ldGate(1, 1);
         CurrentNoteOn[x] = note;
         VoiceOn[x] = 1;
         break;
      case 1:
         FreqHi[2]=SIDMidiNoteHi[note];
         ldFreqHi(2);
         FreqLo[2]=SIDMidiNoteLow[note];
         ldFreqLo(2);
         ldGate(2, 1);
         CurrentNoteOn[x] = note;
         VoiceOn[x] = 1;
         break;
       }  // end of switch

}  // end of if (velocity > 0)

else if (velocity == 0){

       switch (x) {
       case 0:
         ldGate(1, 0);
         VoiceOn[x] = 0;
         break;
      case 1:
         ldGate(2, 0);
         VoiceOn[x] = 0;
         break;

  }  // end of switch for velocity 0
```

```
          }  // end of if (velocity == 0)

                 break;  //  break out of loop

     }  // end of Note On check


}  // end of for loop
}  // end of Handle



 void myHandleNoteOff(byte channel, byte note, byte velocity){
 for (int x = 0; x < 3; x ++){          // search through the 3 AY voices

    if (CurrentNoteOn[x] == note){      // which voice is playing the note to turn off?

                 switch (x) {
                 case 0:
                   ldGate(1, 0);
                   VoiceOn[x] = 0;
                   break;
                case 1:
                   ldGate(2, 0);
                   VoiceOn[x] = 0;
                   break;

                 }  // end of switch

     }  // end of Note check


}  // end of for loop
}  //end of Handle
```

Table 1-1

MIDI note number	Organ	Piano	Note	Frequency	AY =1MHz/16freq	AY Hi Byte	AY Low Byte	SID=freq/0.0596	SID Hi Byte	SID Low Byte
0				8.18	7640.59	29.00	217.00	137.25	0.00	137.00
1				8.66	7217.09	28.00	49.00	145.30	0.00	145.00
2				9.18	6808.28	26.00	152.00	154.03	0.00	154.00
3				9.72	6430.04	25.00	30.00	163.09	0.00	163.00
4				10.3	6067.96	23.00	180.00	172.82	0.00	173.00
5				10.91	5728.69	22.00	97.00	183.05	0.00	183.00
6				11.56	5406.57	21.00	31.00	193.96	0.00	194.00
7				12.25	5102.04	19.00	238.00	205.54	0.00	206.00
8				12.98	4815.10	18.00	207.00	217.79	0.00	218.00
9				13.75	4545.45	17.00	193.00	230.70	0.00	231.00
10				14.57	4289.64	16.00	194.00	244.46	0.00	244.00
11				15.43	4050.55	15.00	211.00	258.89	1.00	3.00
12				16.35	3822.63	14.00	239.00	274.33	1.00	18.00
13				17.32	3608.55	14.00	25.00	290.60	1.00	35.00
14				18.35	3405.99	13.00	78.00	307.89	1.00	52.00
15				19.45	3213.37	12.00	141.00	326.34	1.00	70.00
16				20.6	3033.98	11.00	218.00	345.64	1.00	90.00
17				21.83	2863.03	11.00	47.00	366.28	1.00	110.00
18				23.12	2703.29	10.00	143.00	387.92	1.00	132.00
19				24.5	2551.02	9.00	247.00	411.07	1.00	155.00
20				25.96	2407.55	9.00	104.00	435.57	1.00	180.00
21		1	A0	27.5	2272.73	8.00	225.00	461.41	1.00	205.00
22		2	A#0/Bb0	29.14	2144.82	8.00	97.00	488.93	1.00	233.00
23		3	B0	30.87	2024.62	7.00	233.00	517.95	2.00	6.00
24		4	C1	32.7	1911.31	7.00	119.00	548.66	2.00	37.00
25		5	C#1/Db1	34.65	1803.75	7.00	12.00	581.38	2.00	69.00
26		6	D1	36.71	1702.53	6.00	167.00	615.94	2.00	104.00
27		7	D#1/Eb1	38.89	1607.10	6.00	71.00	652.52	2.00	141.00
28		8	E1	41.2	1516.99	5.00	237.00	691.28	2.00	179.00
29		9	F1	43.65	1431.84	5.00	152.00	732.38	2.00	220.00
30		10	F#1/Gb1	46.25	1351.35	5.00	71.00	776.01	3.00	8.00
31		11	G1	49	1275.51	4.00	252.00	822.15	3.00	54.00
32		12	G#1/Ab1	51.91	1204.01	4.00	180.00	870.97	3.00	103.00
33		13	A1	55	1136.36	4.00	112.00	922.82	3.00	155.00
34		14	A#1/Bb1	58.27	1072.59	4.00	49.00	977.68	3.00	210.00
35		15	B1	61.74	1012.31	3.00	244.00	1035.91	4.00	12.00
36	1	16	C2	65.41	955.51	3.00	188.00	1097.48	4.00	73.00
37	2	17	C#2/Db2	69.3	901.88	3.00	134.00	1162.75	4.00	139.00
38	3	18	D2	73.42	851.27	3.00	83.00	1231.88	4.00	208.00
39	4	19	D#2/Eb2	77.78	803.55	3.00	36.00	1305.03	5.00	25.00
40	5	20	E2	82.41	758.40	2.00	246.00	1382.72	5.00	103.00
41	6	21	F2	87.31	715.84	2.00	204.00	1464.93	5.00	185.00
42	7	22	F#2/Gb2	92.5	675.68	2.00	164.00	1552.01	6.00	16.00
43	8	23	G2	98	637.76	2.00	126.00	1644.30	6.00	108.00
44	9	24	G#2/Ab2	103.83	601.95	2.00	90.00	1742.11	6.00	206.00
45	10	25	A2	110	568.18	2.00	56.00	1845.64	7.00	54.00
46	11	26	A#2/Bb2	116.54	536.30	2.00	24.00	1955.37	7.00	163.00
47	12	27	B2	123.47	506.20	1.00	250.00	2071.64	8.00	24.00
48	13	28	C3	130.81	477.79	1.00	222.00	2194.80	8.00	147.00
49	14	29	C#3/Db3	138.59	450.97	1.00	195.00	2325.34	9.00	21.00
50	15	30	D3	146.83	425.66	1.00	170.00	2463.59	9.00	160.00
51	16	31	D#3/Eb3	155.56	401.77	1.00	146.00	2610.07	10.00	50.00
52	17	32	E3	164.81	379.22	1.00	123.00	2765.27	10.00	205.00
53	18	33	F3	174.61	357.94	1.00	102.00	2929.70	11.00	114.00
54	19	34	F#3/Gb3	185	337.84	1.00	82.00	3104.03	12.00	32.00
55	20	35	G3	196	318.88	1.00	63.00	3288.59	12.00	217.00
56	21	36	G#3/Ab3	207.65	300.99	1.00	45.00	3484.06	13.00	156.00
57	22	37	A3	220	284.09	1.00	28.00	3691.28	14.00	107.00
58	23	38	A#3/Bb3	233.08	268.15	1.00	12.00	3910.74	15.00	71.00
59	24	39	B3	246.94	253.10	0.00	253.00	4143.29	16.00	47.00
60	25	40	C4 (middle C)	261.63	238.89	0.00	239.00	4389.77	17.00	38.00
61	26	41	C#4/Db4	277.18	225.49	0.00	225.00	4650.67	18.00	43.00
62	27	42	D4	293.66	212.83	0.00	213.00	4927.18	19.00	63.00
63	28	43	D#4/Eb4	311.13	200.88	0.00	201.00	5220.30	20.00	100.00
64	29	44	E4	329.63	189.61	0.00	190.00	5530.70	21.00	155.00
65	30	45	F4	349.23	178.97	0.00	179.00	5859.56	22.00	228.00
66	31	46	F#4/Gb4	369.99	168.92	0.00	169.00	6207.89	24.00	64.00
67	32	47	G4	392	159.44	0.00	159.00	6577.18	25.00	177.00
68	33	48	G#4/Ab4	415.3	150.49	0.00	150.00	6968.12	27.00	56.00
69	34	49	A4 concert pitch	440	142.05	0.00	142.00	7382.55	28.00	215.00
70	35	50	A#4/Bb4	466.16	134.07	0.00	134.00	7821.48	30.00	141.00
71	36	51	B4	493.88	126.55	0.00	127.00	8286.58	32.00	95.00
72	37	52	C5	523.25	119.45	0.00	119.00	8779.36	34.00	75.00
73	38	53	C#5/Db5	554.37	112.74	0.00	113.00	9301.51	36.00	86.00
74	39	54	D5	587.33	106.41	0.00	106.00	9854.53	38.00	127.00
75	40	55	D#5/Eb5	622.25	100.44	0.00	100.00	10440.44	40.00	200.00
76	41	56	E5	659.26	94.80	0.00	95.00	11061.41	43.00	53.00
77	42	57	F5	698.46	89.48	0.00	89.00	11719.13	45.00	199.00
78	43	58	F#5/Gb5	739.99	84.46	0.00	84.00	12415.94	48.00	128.00
79	44	59	G5	783.99	79.72	0.00	80.00	13154.19	51.00	98.00
80	45	60	G#5/Ab5	830.61	75.25	0.00	75.00	13936.41	54.00	112.00
81	46	61	A5	880	71.02	0.00	71.00	14765.10	57.00	173.00
82	47	62	A#5/Bb5	932.33	67.04	0.00	67.00	15643.12	61.00	27.00
83	48	63	B5	987.77	63.27	0.00	63.00	16573.32	64.00	189.00
84	49	64	C6	1046.5	59.72	0.00	60.00	17558.72	68.00	151.00
85	50	65	C#6/Db6	1108.73	56.37	0.00	56.00	18602.85	72.00	171.00
86	51	66	D6	1174.66	53.21	0.00	53.00	19709.06	76.00	253.00
87	52	67	D#6/Eb6	1244.51	50.22	0.00	50.00	20881.04	81.00	145.00
88	53	68	E6	1318.51	47.40	0.00	47.00	22122.65	86.00	107.00
89	54	69	F6	1396.91	44.74	0.00	45.00	23438.09	91.00	142.00
90	55	70	F#6/Gb6	1479.98	42.23	0.00	42.00	24831.88	96.00	256.00
91	56	71	G6	1567.98	39.86	0.00	40.00	26308.39	102.00	196.00
92	57	72	G#6/Ab6	1661.22	37.62	0.00	38.00	27872.82	108.00	225.00
93	58	73	A6	1760	35.51	0.00	36.00	29530.20	115.00	90.00
94	59	74	A#6/Bb6	1864.66	33.52	0.00	34.00	31286.24	122.00	54.00
95	60	75	B6	1975.53	31.64	0.00	32.00	33146.48	129.00	122.00
96	61	76	C7	2093	29.86	0.00	30.00	35117.45	137.00	45.00
97		77	C#7/Db7	2217.46	28.19	0.00	28.00	37205.70	145.00	86.00
98		78	D7	2349.32	26.60	0.00	27.00	39418.12	153.00	250.00
99		79	D#7/Eb7	2489.02	25.11	0.00	25.00	41762.08	163.00	34.00
100		80	E7	2637.02	23.70	0.00	24.00	44245.30	172.00	213.00
101		81	F7	2793.83	22.37	0.00	22.00	46876.34	183.00	28.00
102		82	F#7/Gb7	2959.96	21.12	0.00	21.00	49663.76	193.00	255.00
103		83	G7	3135.96	19.93	0.00	20.00	52616.78	205.00	137.00
104		84	G#7/Ab7	3322.44	18.81	0.00	19.00	55745.64	217.00	194.00
105		85	A7	3520	17.76	0.00	18.00	59060.40	230.00	180.00
106		86	A#7/Bb7	3729.31	16.76	0.00	17.00	62572.32	244.00	108.00
107		87	B7	3951.07	15.82	0.00	16.00	62572.00	244.00	108.00
108		88	C8	4186.01	14.93	0.00	15.00	62572.00	244.00	108.00
109			C#8/Db8	4434.92	14.09	0.00	14.00	62572.00	244.00	108.00
110			D8	4698.64	13.30	0.00	13.00	62572.00	244.00	108.00
111			D#8/Eb8	4978.03	12.56	0.00	13.00	62572.00	244.00	108.00
112			E8	5274.04	11.85	0.00	12.00	62572.00	244.00	108.00
113			F8	5587.65	11.19	0.00	11.00	62572.00	244.00	108.00
114			F#8/Gb8	5919.91	10.56	0.00	11.00	62572.00	244.00	108.00
115			G8	6271.93	9.97	0.00	10.00	62572.00	244.00	108.00
116			G#8/Ab8	6644.88	9.41	0.00	9.00	62572.00	244.00	108.00
117			A8	7040	8.88	0.00	9.00	62572.00	244.00	108.00
118			A#8/Bb8	7458.62	8.38	0.00	8.00	62572.00	244.00	108.00
119			B8	7902.13	7.91	0.00	8.00	62572.00	244.00	108.00
120			C9	8372.02	7.47	0.00	7.00	62572.00	244.00	108.00
121			C#9/Db9	8869.84	7.05	0.00	7.00	62572.00	244.00	108.00
122			D9	9397.27	6.65	0.00	7.00	62572.00	244.00	108.00
123			D#9/Eb9	9956.06	6.28	0.00	6.00	62572.00	244.00	108.00
124			E9	10548.08	5.93	0.00	6.00	62572.00	244.00	108.00
125			F9	11175.3	5.59	0.00	6.00	62572.00	244.00	108.00
126			F#9/Gb9	11839.82	5.28	0.00	5.00	62572.00	244.00	108.00
127			G9	12543.85	4.98	0.00	5.00	62572.00	244.00	108.00
top of range			G#9/Ab9	13289.75	4.70	0.00	5.00	62572.00	244.00	108.00