Arduino Polyphonic Synthesizer

with OR Gate Distortion, Fuzz Circuit, and MIDI record

John Talbert - April 2019

Table of Contents

The Synth

This is a multi-voice programmable synthesizer built with the Arduino Pro Micro. Four voices come from Arduino output pins programmed to produce square waves and pulse waves.

One of these pins is used to subject an external audio input to unique distortions.

All the voices are combined with mixer and modulation circuitry creating a wide variety of crazy timbres.

USB MIDI Control over the Synthesizer can be set up using the USBMIDI.h Library.

Supplies:

Arduino Pro Micro from Sparkfun Electronics (5volt version).
Gutted Mix800 Mixer box from Ebay, has most of the needed pots and switches.
Two LM358 op amp chips
One Intersil CA3046 NPN transistor chip.
Sockets for the Pro Micro and the three chips.
Circuit board.
Two 5k or 10k pots.
Two 100k or 50k pots.
Two red LEDs.
Two pushbutton switches.
Misc. resistors and capacitors.
USB cable for the Arduino.
Optional 9 volt power wart.

Voice Frequency

Building a multi-voice synthesizer with the Arduino should be fairly easy given all its digital output pins that are capable of producing square wave and pulse wave audio signals. It even has a tone() function that can easily create a square wave signal at a specified frequency.

tone(pin#, frequency-in-hertz)

However, the tone function can only be used on a single output pin. Getting more than one voice will require some finagling. It can be accomplished by periodically toggling output pins at specific times. To toggle an output pin, you first read its present state and then write in the opposite state as follows :

digitalWrite(voice-pin#, !digitalRead(voice-pin#))

The exclamation point denotes a logical "not" operation, reading the pin state as its opposite state.

Toggling the pin state at equal time periods result in a specific frequency. To get this periodic change, first note that the Arduino loop() function will repeat at a fairly constant rate, and this rate will hopefully be pretty fast if the total loop program is short enough.

With an Arduino Uno clock rate of 16MHz, my program loop time turned out to be 44 useconds (44 millionth of a second, or a rate of about 22.7KHz). In order to produce a squarewave at the frequency of 440Hz (key of A) I would have to toggle the pin output every 26 times through the loop. This is easily done in the code by just decrementing a counter, initially set to 26, each time through the loop. When it reaches zero, toggle the voice pin and reset the counter to 26. The equation for calculating this counter value for different frequencies is given here.

loop-frequency/(2*frequency-desired)

The actual counter value for A-440 using the above equation is 25.83, which was rounded up to 26 for our program. This means that the equal-temperament tuning for A-440 will be slightly off. The higher the frequency, the smaller the counter value and the more "off" the tuning will be. For lower frequencies the counter values will be larger resulting in smaller overall errors when rounding off. So, in effect, the tuning for the lower notes will be spot-on, while the upper register tunings will get progressively worse.

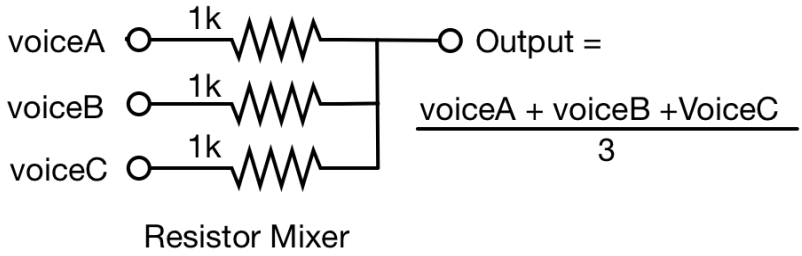
Here are the program lines for the voice counter:

```
// decrement Voice freq counter

--freq;

//toggle Voice pin when counter reaches zero

if (freq <= 0) {
digitalWrite(Voice_pin#, !digitalRead(Voice_pin#) );

//reload counter from a table or a slider value

freq = voiceCounter; }
```

This routine can be copied with other output pins and their respective counter values to create an Arduino Synth with any degree of polyphony, or number of voices.

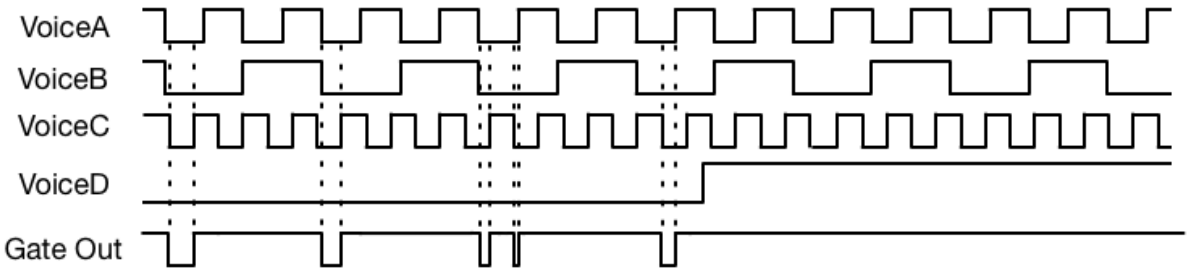
Simple Mixer/Gate

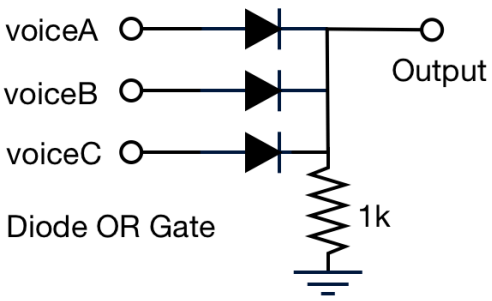
The Arduino voice outputs can be added together with a simple resistor mixer to produce a single output. Each pin output is connected to an equal value resistor. The other sides of the resistors are then connected together to form a single output. The output is then a simple mixed sum of all the voices. One disadvantage with this circuit is that you do lose volume as more voices are added. Three voices will have 1/3 their initial volume, four will have 1/4 the initial volume, and so on.



Resistor Mixer

The sound of squarewaves can get tedious. A diode OR gate circuit is an easy way to add variety in the form of voice modulation. The output of the diode OR gate is basically a pulse waveform that gets more raucous as you add more voices. The gate works by dropping the output low only when all the inputs happen to be low. This is illustrated in the waveform diagram below. VoiceD in the diagram is set to a low frequency to demonstrate how the sound output can be turned off and on, or gated, with a sub- audio frequency.



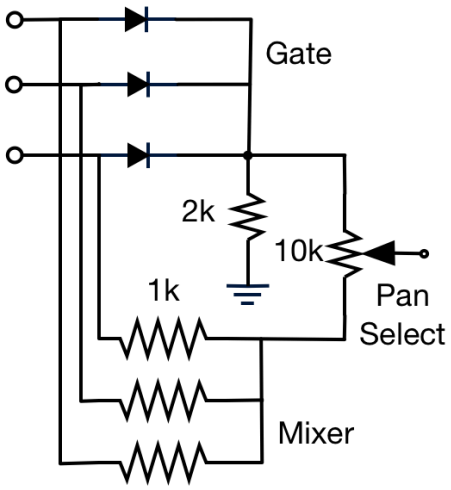


The circuit for a 3-input Diode gate is shown here. Any number of voices can be added by just attaching more Diodes.

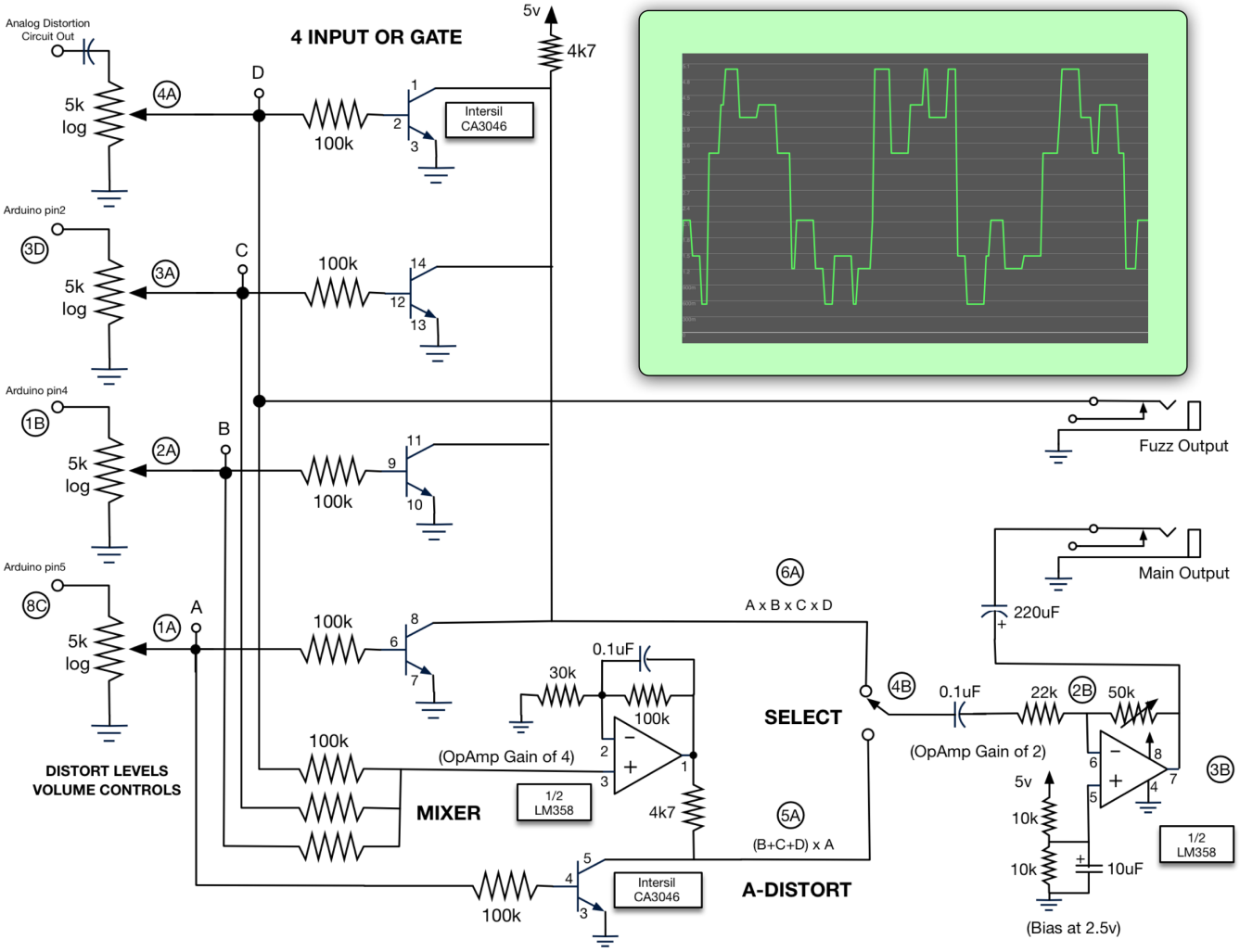
If a voice is not being used, it is important to set it LOW (digitalWrite(pin#, LOW)) to allow the other voices to sound. If any voice is set HIGH, it will effectively close the gate and turn off the other voices.

Both the Mixer and the Gate circuit can be combined into one circuit with a simple switch. However, a more interesting solution might be to use a waveform pan circuit as shown above. The Mixer and Gate outputs are connected to either end of a potentiometer. With the slider connection as the output, you can continuously pan between the two outputs to get any ratio you want of the two.

To make things even more interesting, consider adding volume controls to each of the inputs.



Enhanced Gate Circuit



The circuit shown above is an improved version of the simple Resistor Mixer and Diode Gate circuit. All four inputs now have volume controls which affect the modulation in somewhat complex ways. The 4-transistor gate circuit shown here is actually a NOR gate (negative OR). The output goes high to 5 volts only when all inputs are low; at all other times it remains low at zero volts.

For those interested I can give a simplified description here of how the transistor gate circuit works. Transistors operate in one of three possible states - cutoff, saturation, and active. What follows is a description of these three transistor states and how they affect the final output.

Cutoff State

When a voice input is low near zero volts, which can occur either when the squarewave input goes low or when its volume control is turned down, the transistor is put in the “cutoff state". This is a passive, or do-nothing state. The other transistors are allowed to massage the output in whatever way they want. This transistor will not interfere. When all the gate transistors are in this “cutoff” state, the output just looks like a 4k7 resistor hanging from 5 volts, which is, in effect, a “high” output, hence, the NOR gate definition - the output goes high only when all inputs are low with all the transistors in their cutoff state.

Saturation State

When a voice volume control is turned all the way up and the waveform goes high to 5 volts, the transistor goes into “saturation”. This is a very forceful state because the transistor now actively clamps the output line to zero volts, which is the ground the one transistor lead is connected to. This overrides what all the other transistors are doing. Any input going high will effectively take over the gate output and force it to zero volts.

When all the inputs are turned up in volume, the output waveform is a pulse waveform as illustrated in the Diode Gate waveform diagram, except that the output is now inverted with the pulses going high. The output has only two states, high and low. The transistors will alternate between cutoff (output going high when all of the transistors are in cutoff), and saturation (output going low when any of the transistors are in saturation).

Active State

Here is a condition that is not available with the previously described diode gate circuit and it offers some very interesting output waveforms.

When a voice volume control is not turned all the way up and the input waveform goes high, the transistor tries to go into "saturation" to pull the output line to zero but it doesn't have quite enough "juice" or power to get the job done. The transistor is then said to be in its "active" state. It will weakly pull down the output to some voltage between the 5 volt high and zero volt low.

The other transistors can add their effect to this in-between state, either clamping it to ground if in saturation, or pulling it closer to ground if in the active state.

Volume controls set in-between fully off or on, have a somewhat strange effect on the output. The waveform will no longer be a pulse wave, jumping between only two levels, zero and 5 volts. The output highs will still remain at 5 volts, but the lows will step to various levels between zero and 5 volts, like a wacky staircase.

For very low input volume control settings on all the inputs, the output lows are no longer way down at zero volts; they are closer to the 5 volt highs, creating a lower peak to peak output volume. For higher input volume settings on all the inputs, the lows steps flirt with getting closer to zero volts, creating a higher peak to peak volume.

See above for an example of the output when all the inputs are set at "in-between" volumes. Notice that the maximum highs are at 5 volts, the lowest lows are above the zero-volt floor, and there are lots of different steps in-between.

The enhanced transistor gate design offers the advantage of volume control even in its digitized output form and it also introduces interesting new timbres beyond the usual gated pulse output.

Enhanced Mixer Circuit

The enhanced mixer circuit, part of the circuit diagram in the previous section, starts with a simple three input resistor mixer adding together Voices B, C, and D. This is the same circuit shown in the Diode Mixer/Gate except that volume controls have been added to all three inputs. The mixer output will sound like three squarewave inputs mixed together at different levels. The loss in volume associated with any resistor mixer is corrected with an op-amp configuration that has a gain of 4 (obtained from the equation: 1 + 100k/30k).

Now for the fun part. The 3-voice mix from the output of the opamp is applied to a transistor distortion circuit controlled by Voice A. This is similar to the transistor gate circuits above except that the circuit output is connected through a 4k7 resistor to an actual signal instead of just 5 volts. How this distortion circuit works depends on conditions between the mix signal and the VoiceA control signal.

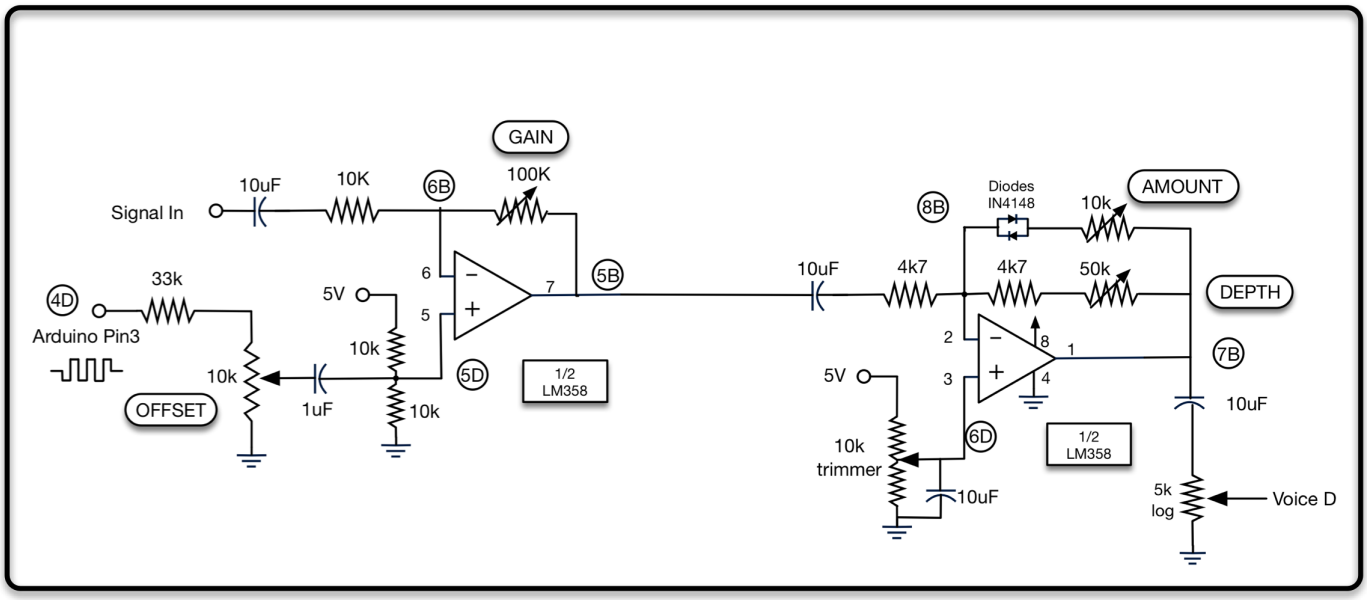
First of all, when the mix signal goes high the transistor circuit will act in exactly the same way as described above in the gate circuit, exhibiting all three transistor states of cutoff, saturation, and active state depending on the voltage of Voice A.

However, when the mix signal is low, or zero volts, the transistor has no voltage to work with and the output will be low no matter what Voice A is doing. A happy consequence of this feature is that any silence in the mix between notes will stay silent. You won't hear the VoiceA modulating signal between notes in the mix.

When the volume control of Voice A is turned down to zero, the transistor will go into cutoff allowing the output to just follow the mix signal. So if you want to just hear the straight mix of the other 3 voices without any added modulation, just turn down the Voice A control volume. Slowly turning up the control signal from zero will gradually add more modulation into the output mix. The modulation will have the pitch of Voice A.

A simple switch is used to select between the transistor gate circuit and the modulated mix circuit. A final opamp stage adds volume control to the final signal output.

Fuzz Circuit



The analog distortion circuit, shown above, is inserted in Voice D. It acts like a guitar fuzz box for whatever audio source is connected to the input jack. It can also act like another Arduino square wave voice when no external audio signal is present.

Gain Control

The first op-amp stage provides a signal gain adjustable from zero to 10. Ignoring the Offset control for now, the first stage also sits the input signal on 2.5 volts which is midway between the power supply ceiling of 5 volts and floor of zero volts.

The second stage adds distortion by chopping off the tops and bottoms of the signal in various ways. The IN4148 diodes are key to the signal distortion. They are always in one of two states, forward or reverse biased.

Depth Control

The Diodes are "reverse biased" in the middle voltages of the signal, when the signal travels between (2.5v + 0.6v) and (2.5v - 0.6v). In this state the diodes have infinite resistance which effectively cuts out the 10k pot leg of the opamp circuit, leaving only the 50k pot in series with a 4k7 resistor. The 50k 'Depth" pot then provides an adjustable gain from 1 to 10 for the middle of the signal. This basically affects how fast the voltages are rising or falling within the waveform. For example, a triangle wave with slowly rising and falling legs could be turned into a square wave with really fast rising and and falling legs.

Amount Control

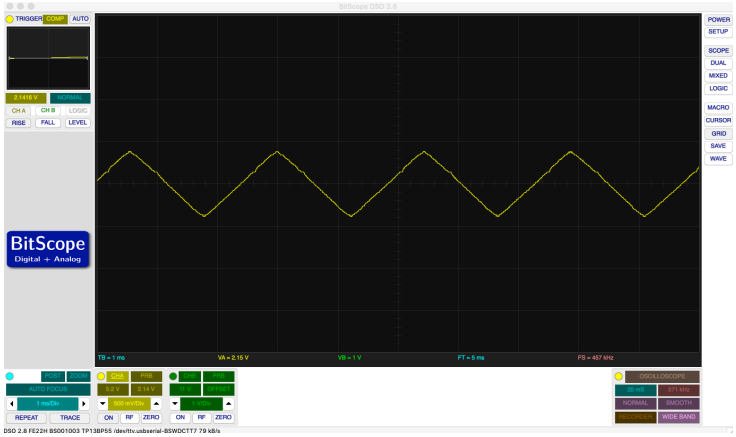
When the signal voltage rises above 3.1 volts or below 1.9 volts, one of the Diodes will "turn on" or become "forward biased". When this happens, the resistance of the Diode goes to zero which causes the 10k "Amount" pot to wake up and start affecting the gain of the opamp circuit. With the 10k pot turned low, the gain goes to zero which chops off the top and bottoms of the input signal turning it into more of a square wave heard as a "fuzz". With the 10k pot turned up, the gain is no longer zero. So instead of rudely chopping off the tops and bottoms of the signal, the opamp merely "squishes" them causing less "buzz" in the sound. This effectively creates a "soft" ceiling and floor for the signal tops and bottoms to crash against.

Knowing that the signal peak chopping and squishing happens around 1.9 volts and 3.1 volts, you can visualize that the 50k Depth control is used to push the peaks more or less into these two voltage areas where "nasty" things happen.

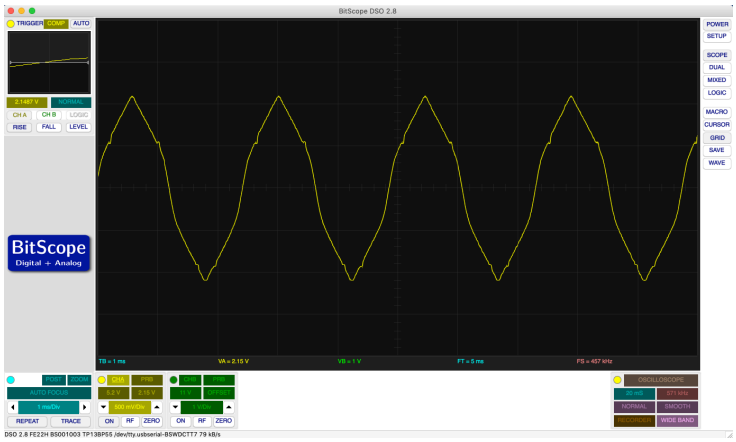
The figures on the next page show actual oscilloscope readings of the output for various settings of the distortion circuit.

Arduino Offset Control

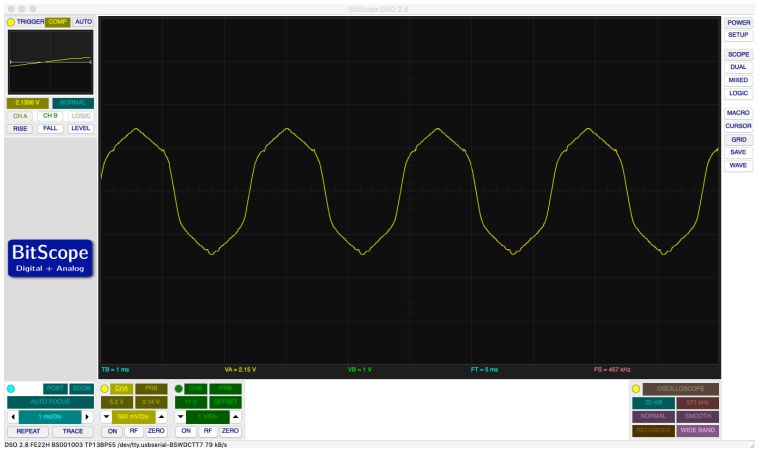
Added to this traditional “fuzz box” circuit is an Arduino output pin connected to an “Offset” control.



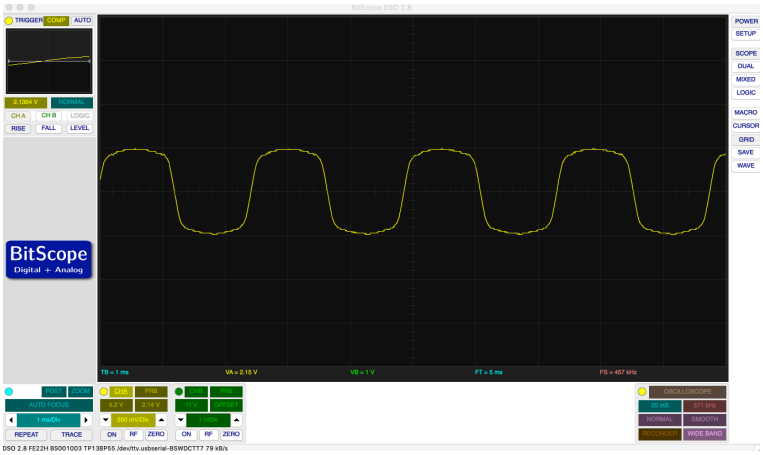
Triangle Wave Input



Depth turned up
Amount turned down
peaks accentuated



Depth turned up
Amount set in middle
peaks squished



Depth turned up
Amount turned up
peaks cut off

Normally we want the audio signal to sit right in the middle of the power supply between zero and 5 volts. For both op amps, a two resistor divider circuit creates a 2.5 voltage that is directly connected to the plus input of the op amp. This effectively sits the signal right in the middle of the power supply range. If the signal peaks try to go above 5 volts or below zero volts, they will get cut off causing distortion. Sitting the signal in the middle gives it the maximum amount of space to oscillate without hitting the ceiling or floor and distorting. But hey! We like distortion so much that we have created our own narrower soft ceiling and floor of 3.1v and 1.9v with the Diodes and have even added an offset circuit on the first stage to mess with the 2.5v level on which the signal normally sits. To get even more manic, we have connected an Arduino pin to the offset control. The Arduino can now be programmed to bounce the input signal up and down crashing against our Diode soft voltage limits.

So here is how the Arduino Offset Control works.

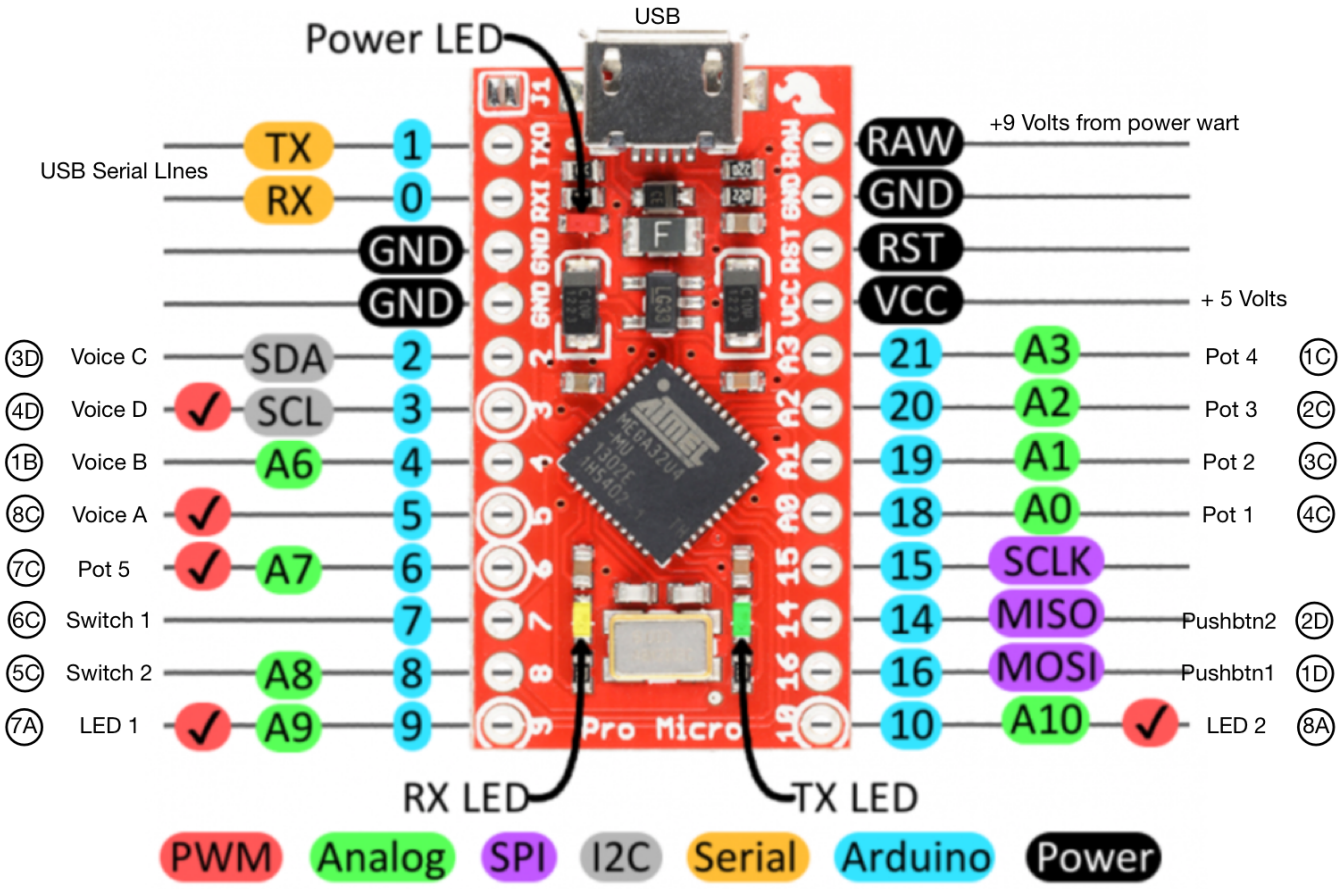
With no signal on either the input jack or the Arduino pin, the voice output of the second op amp should be a voltage of 2.5 volts. Use a voltmeter to adjust the trimmer to get exactly 2.5 volts. However, the signal at the voice D volume control will be sitting at zero volts due to the blocking capacitor.

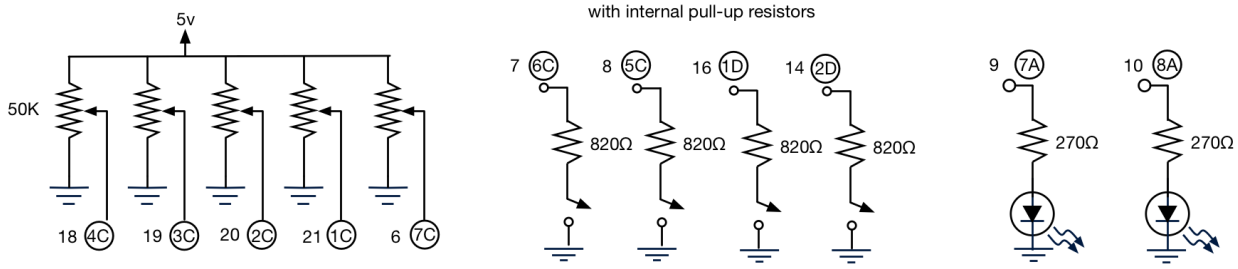
With no signal on the input jack, Voice D can be made to follow an oscillating Arduino pin voltage, acting like the other digital input voices. Turn up the Offset and Depth controls to make this happen.

With an input signal and the Arduino pin programmed to oscillate, the offset pot controls how much the input signal is bounced up and down at the frequency of the Arduino square wave.

The Arduino output used for the offset control is a square wave. The sharp transitions of the square wave result in audible clicks in the output. A capacitor across the 10k offset pot would help filter out some of the "clicking". As an afterthought though, a better solution would be to use an Arduino with a DAC output programmed to produce a sinewave type waveform for the offset signal.

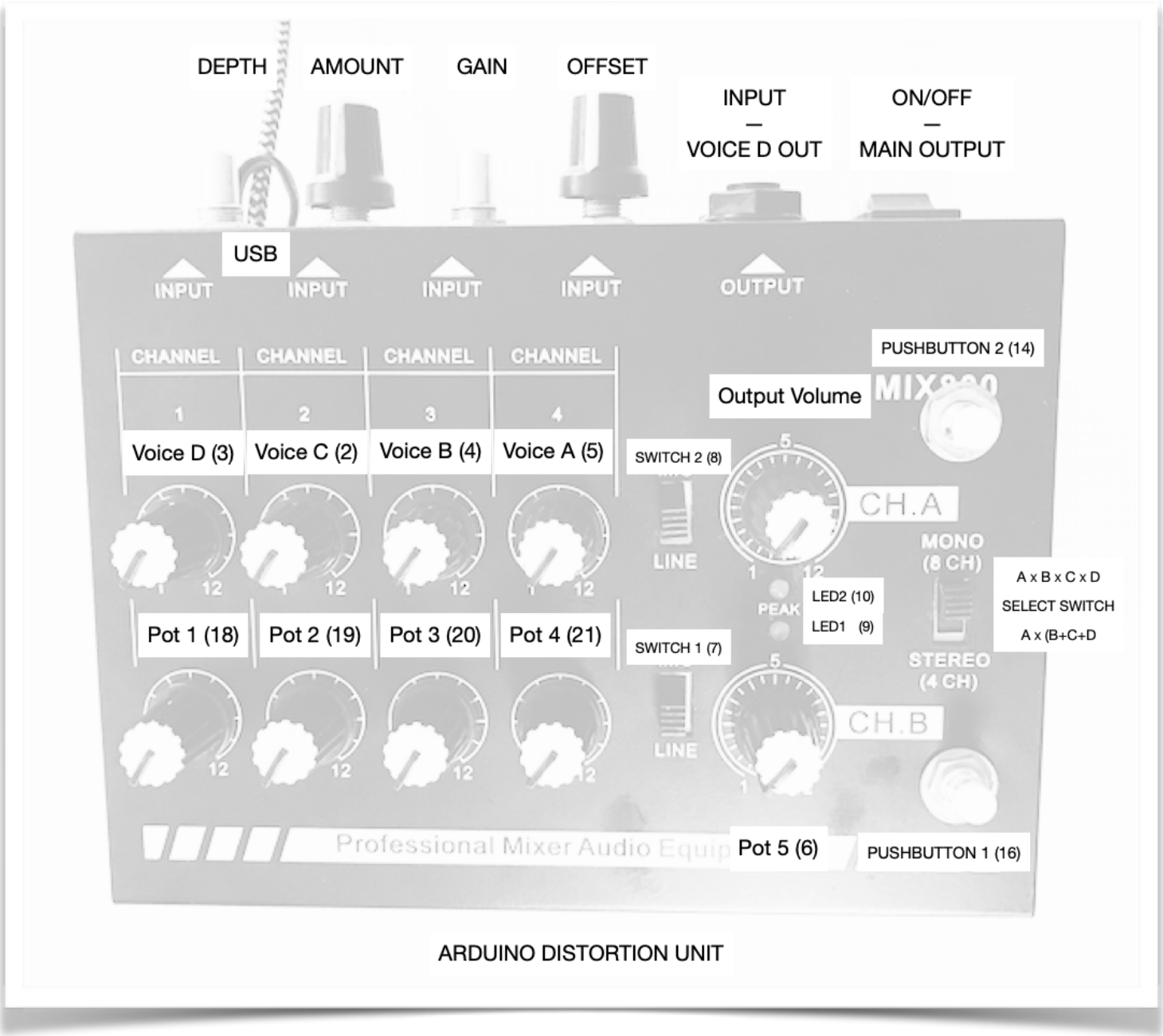
Arduino Pro Micro





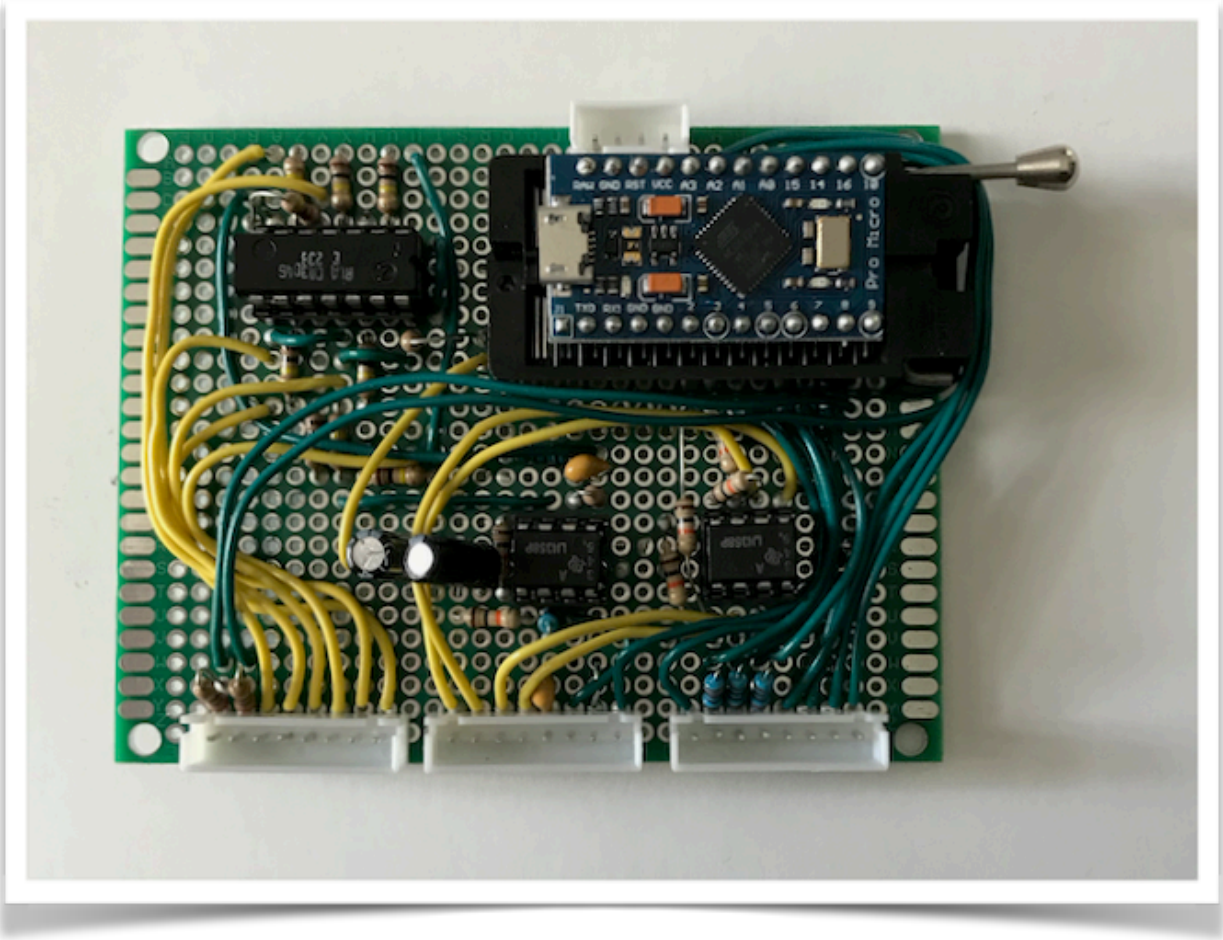
- An Arduino Pro Micro was used for this project. Be sure to get the 5 volt version of the board, not the 3.3 volt version.
- Four pins are connected to the Voice inputs of the Enhanced Mixer/Gate circuit. Note that Voice D goes through the Analog Fuzz circuit first so that it can directly affect an audio signal plugged into the input jack.
- These same output pins could also be sent to the simpler resistor mixer and diode gate circuit if you like. The Arduino programs shown later should also work with this simpler circuit.
- It is worth mentioning that another type of Arduino could be used with this project with one complication. Most of the newer Arduinos use 3.3 volt operation voltages. The switches and controller pots would use 3.3 volts also. Our circuit op amps, however, would not do very well with only 3.3 volt power. You would need to find a higher power source for the Mixer/Gate/Distortion circuitry.
- Five pins, to be programmed as ADC analog inputs, are connected to five controller potentiometers.
- Four pins, to be programmed as digital inputs, are connected to 2 slide switches and 2 pushbutton switches.
- Two pins are connected to LED indicator lights.
- The Arduino and Mixer/Gate/Distortion circuit can all be mounted on a small circuit board. Sockets are needed for the Arduino and the three chips. Edge connectors are used to help connect to all the chassis hardware. You will need to have some experience soldering a circuit together from circuit diagrams.
- A USB cable is needed for programming the Arduino. It can also supply the 5 volt power needed for the circuit. As an option, the Pro Micro has a RAW power input pin to connect a 9 volt power supply. The chassis has an on/off switch that can be used with an external supply.

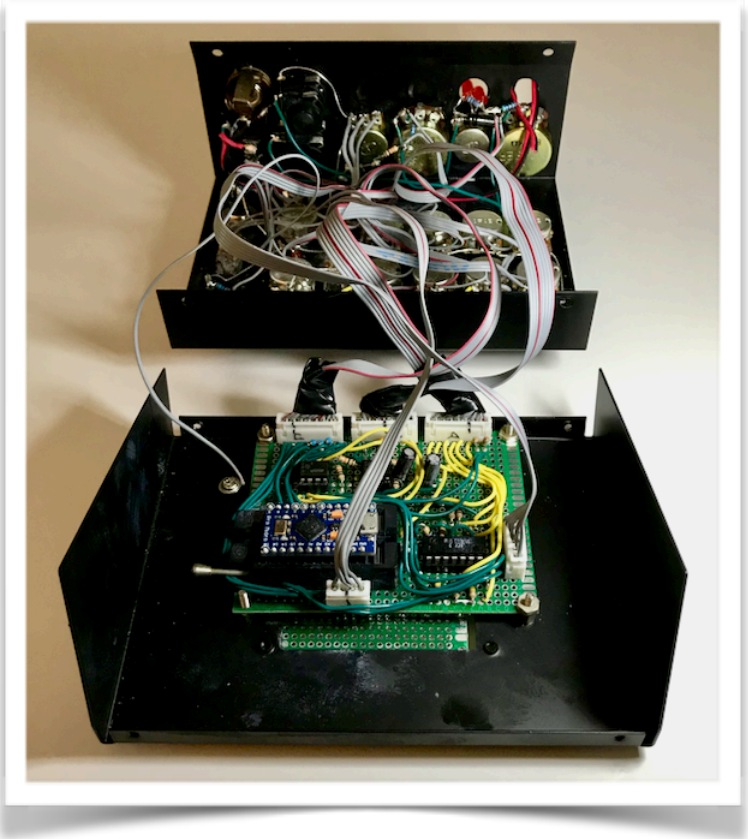
The Box



ARDUINO DISTORTION UNIT

The chassis used for this project was a small MIX800 mixer available from Ebay. Using a solder sucker, I removed the circuit board and kept all the pots, switches, and LED indicator lights, plus some of the jacks.





There were 33 connections between the circuit board and the chassis. To help manage this rat's nest, I used edge connectors on the circuit board and ribbon cable to make the connections.

Synth Test Program

This program continuously prints the values of the 5 pots and 4 switches. It also sets up tones in two voices. Use the Monitor in the Arduino app to see the control values displayed.

```
/*
    Arduino Distortion Unit
    Test Program
    Continuously print out the values from the 5 pots and 4 switches.

*/
//
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
//                  CONSTANTS and Variables
//
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
//
// For use on an Arduino Pro Micro.



//
// ANALOG INPUTS
//
const int POT1 = A0;  //Potentiometer pin numbers
const int POT2 = A1;
const int POT3 = A2;
const int POT4 = A3;
const int POT5 = A7;
```

```
int pot1 = 0;
int pot2 = 0;
int pot3 = 0;
int pot4 = 0;
int pot5 = 0;

//
//DIGITAL SWITCHES
//
const int S1 = 7;  //Switch pin numbers
const int S2 = 8;
const int S3 = 16;
const int S4 = 14;

boolean switch1 = 0;
boolean switch2 = 0;
boolean pushbutton1 = 0;
boolean pushbutton2 = 0;

//
// SYTHESIZER CONSTANTS
//

const int VOICEA = 5;  //Voice pin numbers
const int VOICEB = 4;
const int VOICEC = 2;
const int VOICED = 3;


const int LED1 = 9;  //Led pin numbers
const int LED2 = 10;
```

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

void setup() {

delay(1000);

pinMode(LED1, OUTPUT);    //turn on LED1 as power indicator
digitalWrite(LED1, HIGH);

pinMode(LED2, OUTPUT);
digitalWrite(LED2, LOW);


pinMode(VOICEA, OUTPUT);    //Lows enable distortion gates
digitalWrite(VOICEA, LOW);
pinMode(VOICEB, OUTPUT);
digitalWrite(VOICEB, LOW);
pinMode(VOICEC, OUTPUT);
digitalWrite(VOICEC, LOW);
pinMode(VOICED, OUTPUT);
digitalWrite(VOICED, HIGH);  //Set simple offset for Distortion Input
                    //Turn down when not using

pinMode(S1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(S1, HIGH);
pinMode(S2, INPUT);
digitalWrite(S2, HIGH);
pinMode(S3, INPUT);
digitalWrite(S3, HIGH);
pinMode(S4, INPUT);
digitalWrite(S4, HIGH);

 Serial.begin(9600);

} //End of Setup
```

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

void loop() {

 loadSensors();

  Serial.print("pot1 = ");
 Serial.print(pot1);
   Serial.print("  pot2 = ");
 Serial.print(pot2);
   Serial.print("   pot3 = ");
 Serial.print(pot3);
   Serial.print("   pot4 = ");
 Serial.print(pot4);
   Serial.print("   pot5 = ");
 Serial.print(pot5);

    Serial.print("     ");

   Serial.print("  switches ");
 Serial.print(switch1);
 Serial.print(switch2);
 Serial.print(pushbutton1);
 Serial.println(pushbutton2);

// ~~~~~~~~~~~~~~~~~~Voice Tests ~~~~~~~~~~~~~~~~~~~~~~~~~~

  tone(VOICEA, (100 + pot1));  // Voice A frequency put on pot 1

int v = map(pot2, 0, 255, 1, 40);  // Voice B frequency put on pot 2
  digitalWrite(VOICEB, HIGH);
  delay(v);
  digitalWrite(VOICEB, LOW);
  delay(v);

}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    END OF MAIN LOOP
```

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

void loadSensors(){      // load all current sensor values
    pot1 =   analogRead(POT1);
    pot2 =   analogRead(POT2);
    pot3 =   analogRead(POT3);
    pot4 =   analogRead(POT4);
    pot5 =   analogRead(POT5);
    switch1 = digitalRead(S1);
    switch2 = digitalRead(S2);
    pushbutton1 = digitalRead(S3);
    pushbutton2 = digitalRead(S4);
}
```

Modulated Voices Program

This program can be used with any of the circuits described above - a simple diode gate, a simple resistor mixer, a combination of both, the enhanced transistor gate, the enhanced mixer, or a combination of both. Both the program and and the circuit can easily be modified for any number of voices.

In this program a pot sweeps the frequency of each voice. The first voice uses the the Arduino tone() function reading the value of Slider1 for its frequency. The pot value can be mathematically massaged for any desired range of frequencies. The value is directly translated into Hertz by the tone() function.

The second and third voices use the counter and pin toggling approach to create a squarewave as described in the first section of this paper. Each voice counter is loaded with a value derived from its own slider. Again, the slider value can be mathematically massaged for any frequency range. If that range goes into sub-audio frequencies you can get some interesting pulsing on-off effects.

Switches were added to silence two of the voices. To silence a counter voice set its value LOW and bypass the counter decrement in the program. Don't make the mistake of setting the voice HIGH; in an OR gate a HIGH will turn off all the voices.

Sweeping frequencies against each other creates interesting modulation effects. However, if you want to get away from sweeping sounds, have the pots address an array of discrete counter values instead or have a switch step through an array of set values or trigger random values. The possibilities are endless. Enjoy exploring but be careful to keep the program short as the length of the program affects the speed of the counter decrement, limiting how high in frequency the voices can get.

What follows is the Arduino Synth program code.

```
/*
     3-VOICE ARDUINO SYNTHESIZER

    3 squarewave tones produced from Arduino pins

    Slider1 - Sets Frequency of Tone1 with arduino tone()
    Slider2 - Sets Frequency of Tone2
    Slider3 - Sets Frequency of Tone3
    Slider4 - Sets the decrement clock

    Switch1 - turns off or on Tone2
    Switch2 - turns off or on Tone3

Arduino's tone() function can only be used to set up a squarewave on one output.
Two more voices are created from a fast loop clock decrementing two freq values and toggling voice outputs
when they reach zero.  The freq values determine the frequency of the voices.


*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//
// For use on an Arduino Pro Micro.

//
// ANALOG INPUTS
//
const int POT1 = A0;  //Potentiometer pin numbers
const int POT2 = A1;
const int POT3 = A2;
const int POT4 = A3;
const int POT5 = A7;

int pot1 = 0;
int pot2 = 0;
int pot3 = 0;
int pot4 = 0;
int pot5 = 0;

//
//DIGIITAL SWITCHES
//
const int S1 = 7;  //Switch pin numbers
const int S2 = 8;
const int S3 = 16;
const int S4 = 14;

boolean switch1 = 0;
boolean switch2 = 0;
boolean pushbutton1 = 0;
boolean pushbutton2 = 0;
```

```
//
// SYTHESIZER CONSTANTS
//

const int VOICEA = 5;  //Voice pin numbers
const int VOICEB = 4;
const int VOICEC = 2;
const int VOICED = 3;


const int LED1 = 9;  //Led pin numbers
const int LED2 = 10;

int freq2 = 100;  //variables to set voice frequencies
int freq3 = 100;


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


void setup() {

delay(1000);

pinMode(LED1, OUTPUT);    //turn on LED1 as power indicator
digitalWrite(LED1, HIGH);

pinMode(LED2, OUTPUT);
digitalWrite(LED2, LOW);


pinMode(VOICEA, OUTPUT);    //Lows enable distortion gate
digitalWrite(VOICEA, LOW);
pinMode(VOICEB, OUTPUT);
digitalWrite(VOICEB, LOW);
pinMode(VOICEC, OUTPUT);
digitalWrite(VOICEC, LOW);
pinMode(VOICED, OUTPUT);
digitalWrite(VOICED, HIGH);  //Set simple offset for Distortion Input
                        //Turn down when not using

pinMode(S1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(S1, HIGH);
pinMode(S2, INPUT);
digitalWrite(S2, HIGH);
pinMode(S3, INPUT);
digitalWrite(S3, HIGH);
pinMode(S4, INPUT);
digitalWrite(S4, HIGH);

 Serial.begin(9600);

} //End of Setup
```

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

void loop() {

    tone(VOICEA, (analogRead(POT1) << 2));  // set frequency of Voice1 using tone( )

    delayMicroseconds(analogRead(POT4) << 4);  // wait before continuing, sets main loop clock

    freq2 = voiceRun(digitalRead(S1), VOICEC, freq2, POT2);  // Voice2

    freq3 = voiceRun(digitalRead(S2), VOICEB, freq3, POT3);  // Voice3

}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    END OF MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


 int voiceRun(bool off_on, int voicePin, int freq, int potPin){

   if (off_on) { digitalWrite(voicePin, LOW); }

           else{
             --freq;
             if (freq <= 0){
                 digitalWrite(voicePin, !digitalRead(voicePin));
                 freq = analogRead(potPin) >> 4;
             }
           }
  return freq;
  }

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

Random Note Program

```
/*
     3-VOICE ARDUINO SYNTHESIZER

     2 Voices with tones randomly produced and modulated by Voice A

     Pot1 - Sets base Frequency of Voice B
     Pot2 - Sets Frequency Range of Voice B
     Pot3 - Sets base Frequency of Voice C
     Pot4 - Sets Frequency Range of Voice C

     Switch1 - Selects between Pot5 setting the frequency of Voice A
               or setting overall note durations
     Switch2 - Stops everything at current sounding voices

Arduino's tone() function can only be used to set up a squarewave on one output.
Two more voices are created from the fast loop clock decrementing two freq values and toggling voice outputs
when they reach zero.  The freq values determine the frequency of the voices.


*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//
// For use on an Arduino Pro Micro.

//
// ANALOG INPUTS
//
const int POT1 = A0;  //Potentiometer pin numbers
const int POT2 = A1;
const int POT3 = A2;
const int POT4 = A3;
const int POT5 = A7;

int pot1 = 0;
int pot2 = 0;
int pot3 = 0;
int pot4 = 0;
int pot5 = 0;
```

```
//
//DIGIITAL SWITCHES
//
const int S1 = 7;  //Switch pin numbers
const int S2 = 8;
const int S3 = 16;
const int S4 = 14;

boolean switch1 = 0;
boolean switch2 = 0;
boolean pushbutton1 = 0;
boolean pushbutton2 = 0;


//
// SYTHESIZER CONSTANTS
//

const int VOICEA = 5;  //Voice pin numbers
const int VOICEB = 4;
const int VOICEC = 2;
const int VOICED = 3;


const int LED1 = 9;  //Led pin numbers
const int LED2 = 10;

int freqA = 100;  //variables to set voice frequencies
int freqB = 100;
int freqC = 100;
int freqD = 100;

int countA = 100;
int countB = 100;
int countC = 100;
int countD = 100;

long A_on =0;
long B_on =0;
long C_on =0;
long D_on =0;

long A_off = 100;
long B_off = 100;
long C_off = 100;
long D_off = 100;

long dur = 100;
bool stopV = 0;
```

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


void setup() {

delay(1000);

pinMode(LED1, OUTPUT);    //turn on LED1 as power indicator
digitalWrite(LED1, HIGH);

pinMode(LED2, OUTPUT);
digitalWrite(LED2, LOW);


pinMode(VOICEA, OUTPUT);    //Lows enable distortion gate
digitalWrite(VOICEA, LOW);
pinMode(VOICEB, OUTPUT);
digitalWrite(VOICEB, LOW);
pinMode(VOICEC, OUTPUT);
digitalWrite(VOICEC, LOW);
pinMode(VOICED, OUTPUT);
digitalWrite(VOICED, LOW);

pinMode(1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(S1, HIGH);
pinMode(S2, INPUT);
digitalWrite(S2, HIGH);
pinMode(S3, INPUT);
digitalWrite(S3, HIGH);
pinMode(S4, INPUT);
digitalWrite(S4, HIGH);

} //End of Setup

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

void loop() {

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    VOICE A
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

if (digitalRead(S1)){                       // divide up use of pot5 with Switch1
   dur = analogRead(POT5) >> 3;             //set envelope durations
}
        else{
        tone(VOICEA, (50 + analogRead(POT5)));  // set frequency of VoiceA using tone
        }
```

```
 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    VOICE B
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

if (B_on != 0){                          // if Voice is on, sounding
   if (stopV) {--B_on ;}

    --countB;                    // run the voice frequency toggling routine
   if (countB <= 0){
                digitalWrite(VOICEB, !digitalRead(VOICEB));
                countB = freqB;
   }

           if (B_on == 0){                // turn off voice if at end of on-duration
             digitalWrite(VOICEB, LOW);
             B_off = random(1, dur) << 4 ;  // get random 8 bit duration for B off time
             freqB = getFreqB();           // get new random frequency for voice
           }
}

// ------------

else if (B_off != 0){                     // if Voice is off, not sounding
    --B_off;
           if (B_off == 0){               // turn on voice if at end of off-duration
              B_on = random(1, dur) << 4 ;  // get random duration for on time
           }
}


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    VOICE C
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

if (C_on != 0){                          // if Voice is on, sounding
   if (stopV){--C_on ;}

   --countC;                    // run the voice frequency toggling routine
   if (countC <= 0){
                digitalWrite(VOICEC, !digitalRead(VOICEC));
                countC = freqC;
   }

           if (C_on == 0){                // turn off voice if at end of on-duration
             digitalWrite(VOICEC, LOW);
             C_off = random(1, dur) << 4 ;  // get random 8 bit duration for B off time
             freqC = getFreqC();           // get new random frequency for voice
           }
}

// ------------

else if (C_off != 0){                     // if Voice is off, not sounding
    --C_off;
           if (C_off == 0){               // turn on voice if at end of off-duration
              C_on = random(1, dur) << 4 ;  // get random duration for on time
           }
```

```
}


// ------------Switch2  stops everything to sounding notes----------------------

                        stopV = digitalRead(S3);

}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    END OF MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


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

   int basefreq = analogRead(POT3) >> 4;
   int result = basefreq + random(analogRead(POT4) >> 3); // range of frequencies around the base
   return result;
}

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

   int basefreq = analogRead(POT1) >> 4;
   int result = basefreq + random(analogRead(POT2) >> 3); // range of frequencies around the base
   return result;
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Arduino MIDI Synth Program

This program produces three voice polyphony played from a MIDI keyboard and modulated by Voice A.

The Arduino Pro Micro can be used as a USB type MIDI device using the Library USBMIDI.h. In fact, any Arduino board that uses the ATmega32U4 processor has HID capabilities and thus can be set up as a MIDI device over USB.

This program sets up MIDI control over Voices B, C, and D using MIDI NoteOn and NoteOff commands. Voice A is used to modulate the other voices. It uses the Arduino tone( ) function and a pot to control its frequency.

Alternatively, the tone( ) function could be used to generate equal temperament pitches by using the MIDI Note number to index an array of equal temperament frequencies. Note that the lowest notes are not accurate due to some limitations in the tone( ) function.

```
unsigned long toneFreq[128] = {8, 9, 9, 10, 10, 11, 12, 12, 13, 14, 15, 15, 16, 17, 18, 19, 21,
22, 23, 24, 26, 28, 29, 31, 33, 35, 37, 39, 41, 44, 46, 49, 52, 55, 58, 62, 65, 69, 73, 78,
82, 87, 92, 98, 104, 110, 117, 123, 131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233,
247, 262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494, 523, 554, 587, 622, 659,
698, 740, 784, 831, 880, 932, 988, 1047, 1109, 1175, 1245, 1319, 1397, 1480, 1568, 1661,
1760, 1865, 1976, 2093, 2217, 2349, 2489, 2637, 2794, 2960, 3136, 3322, 3520, 3729, 3951,
4186, 4435, 4699, 4978, 5274, 5588, 5920, 5920, 6645, 7040, 7459, 7902, 8372, 8870, 9397,
9956, 10548, 11175, 11840, 12544};
```

This array assigns a frequency to each of 127 MIDI note numbers or keys on a MIDI keyboard.

Voices B, C, and D produce squarewaves by counting down from a counter value at the speed of the main loop program. The output pin is toggled when the counter reaches zero and is reloaded. The values loaded into the counter come from another array. If the squarewave produced is to oscillate at the same equal temperament frequencies given in the array above, this array must be filled with values given by the formula:

Array Value = (loop frequency)/(2 * note-frequency)

The Note On routine includes a commented-out Serial.println( ) command that will print out the time period in microseconds that the main loop( ) program takes to complete one pass through the loop. The loop frequency is then just 1 divided by the loop time period. In the current program the loop period turned out to be 44 microseconds which results in the following array of values used to produce equal temperament frequencies for each of 128 MIDI Note values (calculated in a spreadsheet program).

```
unsigned long freqTable[128] = {1389, 1312, 1238, 1169, 1103, 1042, 983, 928, 875, 826, 780,
    736, 695, 656, 619, 584, 552, 521, 492, 464, 438, 413, 390, 368, 348, 328, 310, 292, 276,
    260, 246, 232, 219, 207, 195, 184, 174, 164, 155, 146, 138, 130, 123, 116, 109, 103, 98,
    92, 87, 82, 77, 73, 69, 65, 61, 58, 55, 52, 49, 46, 43, 41, 39, 37, 34, 33, 31, 29, 27,
    26, 24, 23, 22, 20, 19, 18, 17, 16, 15, 14, 14, 13, 12, 12, 11, 10, 10, 9, 9, 8, 8, 7, 7,
    6, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1};
```

Notice that there is no way the values at the top of the array can produce accurate, in tune, pitch frequencies. These smaller values have high tuning errors due to the somewhat long loop time of 44 microseconds. The tunings for Voices B and C start getting off around A 440 with the array value of 26. Though not ideal, this mistuning of pitch values for the higher notes may not be as noticeable if used with Gate distortion. The problem could be improved by somehow shortening the main loop program, or using a faster Arduino board.

With this program, your Synthesizer will show up as an Arduino MIDI device when its USB is connected to your computer. Connect any USB MIDI keyboard also to your computer to control the Synthesizer from the MIDI keyboard.

You will need some way of telling the computer to connect the output of the USB Keyboard to the USB Arduino MIDI device input. This may require a special app on your computer.

The free app MidiPipe can be used on a Mac to make USB connections. Create a Pipe with the Tools "MIDI In" at the top followed by "MIDI Out". Select your MIDI keyboard from "MIDI In" pull-down selections and the Arduino from "MIDI Out".

```
/*
    Arduino Distortion Synthesizer
    MIDI Program

    Uses MIDIUSB.h library to receive MIDI commands over the USB cable
    Works with all Arduinos that have the ATmega32U4 processor.
    These have HID capabilities.

    VoiceA is used to modulate the other three voices.  It uses the tone( ) function

    Voices B, C, and D are controlled from a USB MIDI keyboard.


*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//
// For use on an Arduino Pro Micro.

#include "MIDIUSB.h"

//
// ANALOG INPUTS
//
const int POT1 = A0;  //Potentiometer pin numbers
const int POT2 = A1;
const int POT3 = A2;
const int POT4 = A3;
const int POT5 = A7;

int pot1 = 0;
int pot2 = 0;
int pot3 = 0;
int pot4 = 0;
int pot5 = 0;

//
//DIGIITAL SWITCHES
//
const int S1 = 7;  //Switch pin numbers
const int S2 = 8;
const int S3 = 16;
const int S4 = 14;

boolean switch1 = 0;
boolean switch2 = 0;
```

```
boolean pushbutton1 = 0;
boolean pushbutton2 = 0;

//
// SYTHESIZER CONSTANTS
//

const int VOICEA = 5;  //Voice pin numbers
const int VOICEB = 4;
const int VOICEC = 2;
const int VOICED = 3;


const int LED1 = 9;  //Led pin numbers
const int LED2 = 10;

// Table for the countdown values used to set equal temperament frequecies for Voices B, C and D
// Values calculated in a spreadsheet:  (loop-Frequency)/(2 * MIDI-Note-Frequency)
// Recalculate for different loop-frequencies.  Tuning is off for the higher keys.

/*
 unsigned long MIDI_Note_Freq[128] = {8, 9, 9, 10, 10, 11, 12, 12, 13, 14, 15, 15, 16, 17, 18, 19, 21,
   22, 23, 24, 26, 28, 29, 31, 33, 35, 37, 39, 41, 44, 46, 49, 52, 55, 58, 62, 65, 69, 73, 78,
   82, 87, 92, 98, 104, 110, 117, 123, 131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233,
   247, 262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494, 523, 554, 587, 622, 659,
   698, 740, 784, 831, 880, 932, 988, 1047, 1109, 1175, 1245, 1319, 1397, 1480, 1568, 1661,
   1760, 1865, 1976, 2093, 2217, 2349, 2489, 2637, 2794, 2960, 3136, 3322, 3520, 3729, 3951,
   4186, 4435, 4699, 4978, 5274, 5588, 5920, 5920, 6645, 7040, 7459, 7902, 8372, 8870, 9397,
   9956, 10548, 11175, 11840, 12544};
*/

unsigned long freqTable[128] = {1389, 1312, 1238, 1169, 1103, 1042, 983, 928, 875, 826, 780,
   736, 695, 656, 619, 584, 552, 521, 492, 464, 438, 413, 390, 368, 348, 328, 310, 292, 276,
   260, 246, 232, 219, 207, 195, 184, 174, 164, 155, 146, 138, 130, 123, 116, 109, 103, 98,
   92, 87, 82, 77, 73, 69, 65, 61, 58, 55, 52, 49, 46, 43, 41, 39, 37, 34, 33, 31, 29, 27,
   26, 24, 23, 22, 20, 19, 18, 17, 16, 15, 14, 14, 13, 12, 12, 11, 10, 10, 9, 9, 8, 8, 7, 7,
   6, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1};

//current note on's for Arduino pin voices A, B, and C.  Use note value 0 for "not on".

byte CurrentNoteOn[3] = {0, 0, 0};

int freqA = 100;  //variables to set voice frequncies
int freqB = 100;
int freqC = 100;
int freqD = 100;
```

```
unsigned long deltamicro = 0;
unsigned long lastmicro = 0;

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


void setup() {

delay(1000);
tone(VOICEA, 64, 200);
tone(VOICEA, 164, 200);

pinMode(LED1, OUTPUT);    //turn on LED1 as power indicator
digitalWrite(LED1, HIGH);

pinMode(LED2, OUTPUT);
digitalWrite(LED2, LOW);


pinMode(VOICEA, OUTPUT);    //Lows enable distortion gates
digitalWrite(VOICEA, LOW);
pinMode(VOICEB, OUTPUT);
digitalWrite(VOICEB, LOW);
pinMode(VOICEC, OUTPUT);
digitalWrite(VOICEC, LOW);
pinMode(VOICED, OUTPUT);
digitalWrite(VOICED, LOW);

pinMode(S1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(S1, HIGH);
pinMode(S2, INPUT);
digitalWrite(S2, HIGH);
pinMode(S3, INPUT);
digitalWrite(S3, HIGH);
pinMode(S4, INPUT);
digitalWrite(S4, HIGH);

// Serial.begin(115200);   //Used to read loop time

} //End of Setup
```

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

void loop() {

//  deltamicro = micros() - lastmicro;   //Used to read loop time  (44usec with an arduino Mega board ??)
//  lastmicro = micros();               // Loop Frequency = 1/deltamicro  (deltamicro is in microseconds)

// Arduino's tone() function can only be used to set up a squarewave on one output - VoiceA.
// Three more voices are created from a fast loop time decrementing "freq" values and toggling voice
// outputs when they reach zero.  The "freq" values determine the frequency of the voices.

//VoiceA ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


     tone(VOICEA, (50 + analogRead(POT5)));  // set frequency of VoiceA using tone, used as modulator


  //VoiceB ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    if (CurrentNoteOn[0] != 0) {
      --freqB;                                // decrement Voice B freqB counter
      if (freqB <= 0){
         digitalWrite(VOICEB, !digitalRead(VOICEB));  //toggle Voice B pin when counter reaches zero
         freqB = freqTable[CurrentNoteOn[0]];         //reload counter from table

    }
      }

  //VoiceC ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    if (CurrentNoteOn[1] != 0) {                    // same as above for  Voice C
      --freqC;
      if (freqC <= 0){
         digitalWrite(VOICEC, !digitalRead(VOICEC));
        freqC = freqTable[CurrentNoteOn[1]];

    }
      }

 //VoiceD ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    if (CurrentNoteOn[2] != 0) {                    // same as above for  Voice D
      --freqD;
      if (freqD <= 0){
         digitalWrite(VOICED, !digitalRead(VOICED));
```

```
        digitalWrite(VOICED, !digitalRead(VOICED));
       freqD = freqTable[CurrentNoteOn[2]];

    }
     }

//--------------------------------------------------------------------------------


midiEventPacket_t rx = MidiUSB.read();

 switch (rx.header) {

  case 0:
          break; //No pending events

  case 0x9:  //NoteOn

  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  //   Serial.println(deltamicro);  //used to find the loop time in microseconds

  if (rx.byte3 == 0){    //  Note OFF, note velocity of zero.

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

            if (CurrentNoteOn[x] == rx.byte2){ // voice playing to turn off?

                    switch (x) {
                    case 0:
                      CurrentNoteOn[x] = 0;
                      digitalWrite(VOICEB, LOW);
                      break;
                    case 1:
                      CurrentNoteOn[x] = 0;
                      digitalWrite(VOICEC, LOW);
                      break;
                    case 2:
                      CurrentNoteOn[x] = 0;
                      digitalWrite(VOICED, LOW);
                      break;
                    }  // end of switch

            }  // end of Note check
            }  // end of for loop
```

```
}
else{    //Note ON,  note velocity not zero
        for (int x = 0; x < 3; x ++){    // search through the 3 Arduino pin voices

            if (CurrentNoteOn[x] == 0){      // this voice is available
                CurrentNoteOn[x] = rx.byte2;  //pitch
                break;  //  break out of voice search loop

            }  // end of Note On check
            }  // end of for loop
}
        break;
 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


 case 0x8: // Note Off

 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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

 if (CurrentNoteOn[x] == rx.byte2){    // which voice is playing the note to turn off?

           switch (x) {
           case 0:
             CurrentNoteOn[x] = 0;
             digitalWrite(VOICEB, LOW);
             break;
           case 1:
             CurrentNoteOn[x] = 0;
             digitalWrite(VOICEC, LOW);
             break;
           case 2:
             CurrentNoteOn[x] = 0;
             digitalWrite(VOICED, LOW);
             break;
           }  // end of switch

     }  // end of Note check


     }  // end of for loop
        break;
 //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 case 0xB:  // control Change
        break;
```

```
   default:
            break;
  }

}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  END OF MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

MIDI Record and Playback

This program builds on the previous MIDI Program by adding performance recording and playback capabilities. Throw the Record Switch to record up to 50 MIDI events and their times. Throw the Playback switch to continuously repeat the recorded material. Notes can be played live over the playback when voices are available.

Four 50 value arrays were created to hold event information - Event time, MIDI Note Number, Note Velocity, and Voice number (1, 2, or 3). The ability to record and playback up to 50 events works well for MIDI Note On and Off events. MIDI Control events, however, are usually very dense and would quickly overwhelm any size event array built in the limited memory space of the Arduino. For this reason, the program only covers MIDI Note On and Off events.

A timer function is needed to record event times. The Arduino millis( ) function is close to what is needed. Millis( ) returns the number of milliseconds passed since the board was powered on. Ideally, what is really needed is a millis( ) function that can be reset to zero time whenever a recording or playback is started. Sadly, this function can't be reset to zero, however, a work-around is possible. At start times a function called timestamp( ) reads the current millis( ) time value and stores it in a variable called startTime. timeStamp( ) is called at the start of recording, and at the start and repeat times of playback. Then, instead of using millis( ) directly, the function dur( ) is called which returns the value (millis( ) - startTime).

During recording, the dur( ) value is store in the eventTimes[ ] array whenever a MIDI Note ON or MIDI Note OFF event occurs, after which the array index is incremented to the next position in the array, ready for the next event.

During playback, dur( ) is compared with the next eventTime[ ] in the array. This happens once every time through the main loop( ) or, in our case, once every 44 microseconds. If the current time dur( ) is greater than or equal to the stored eventTime[ ] then the event is played, pulling data from same index position in the other arrays - eventNotes[ ], eventVelocity[ ], and eventVoice[ ]. The index value into the arrays is then incremented to prepare for picking up the next event values.

Two switches are watched in the main loop( ) program, one for record and one for playback. During playback, the note events in the arrays are repeated until the playback switch is turned off. At the start of each repetition the timeStamp( ) function is called to reset the dur( ) time to zero.

The Record/Playback feature of this program is useful but adds a lot of time to the loop( ) function which will degrade the higher pitch accuracy for those voices that use counters in the loop to build their waveforms.

What follows is the MIDI Synth program code.

```
/*
    Arduino Distortion Unit
    MIDI Program
    Uses MIDIUSB.h library to run MIDI commands over the USB cable
    Works with all Arduinos that have the ATmega32U4 processor.
    These have HID capabilities.

    VoiceA is used to modulate the other three voices.  It uses the tone( ) function

    Voices B, C, and D are controlled from a USB MIDI keyboard.

    Switch 1 (lower) can be turned on to record up to 50 note events and times.
    Switch 2 (upper) can be turned on to play back the recorded notes.

*/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  CONSTANTS and Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//
// For use on an Arduino Pro Micro.

#include "MIDIUSB.h"

//
// ANALOG INPUTS
//
const int POT1 = A0;  //Potentiometer pin numbers
const int POT2 = A1;
const int POT3 = A2;
const int POT4 = A3;
const int POT5 = A7;

int pot1 = 0;
int pot2 = 0;
int pot3 = 0;
int pot4 = 0;
int pot5 = 0;

//
//DIGIITAL SWITCHES
//
const int S1 = 7;  //Switch pin numbers
const int S2 = 8;
const int S3 = 16;
const int S4 = 14;

boolean switch1 = 0;
boolean switch2 = 0;
boolean pushbutton1 = 0;
boolean pushbutton2 = 0;

//
// SYTHESIZER CONSTANTS
//
```

```
const int VOICEA = 5;  //Voice pin numbers
const int VOICEB = 4;
const int VOICEC = 2;
const int VOICED = 3;


const int LED1 = 9;  //Led pin numbers
const int LED2 = 10;

//tone() frequency values for equal temperament A440 MIDI NoteON commands.  Used for VoiceA

// Note the limitations of tone() which at 16mhz specifies a minimum frequency of 31hz - in other words, notes
below
// B0 will play at the wrong frequency since the timer can't run that slowly!

unsigned long toneFreq[128] = {8, 9, 9, 10, 10, 11, 12, 12, 13, 14, 15, 15, 16, 17, 18, 19, 21,
   22, 23, 24, 26, 28, 29, 31, 33, 35, 37, 39, 41, 44, 46, 49, 52, 55, 58, 62, 65, 69, 73, 78,
   82, 87, 92, 98, 104, 110, 117, 123, 131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233,
   247, 262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494, 523, 554, 587, 622, 659,
   698, 740, 784, 831, 880, 932, 988, 1047, 1109, 1175, 1245, 1319, 1397, 1480, 1568, 1661,
   1760, 1865, 1976, 2093, 2217, 2349, 2489, 2637, 2794, 2960, 3136, 3322, 3520, 3729, 3951,
   4186, 4435, 4699, 4978, 5274, 5588, 5920, 5920, 6645, 7040, 7459, 7902, 8372, 8870, 9397,
   9956, 10548, 11175, 11840, 12544};

// Table for the countdown values used to set equal temperament frequecies for Voices B and C
// Values calculated in a spreadsheet:  (loop-Frequency)/(2 * MIDI-Note-Frequency)

unsigned long freqTable[128] = {1389, 1312, 1238, 1169, 1103, 1042, 983, 928, 875, 826, 780,
   736, 695, 656, 619, 584, 552, 521, 492, 464, 438, 413, 390, 368, 348, 328, 310, 292, 276,
   260, 246, 232, 219, 207, 195, 184, 174, 164, 155, 146, 138, 130, 123, 116, 109, 103, 98,
   92, 87, 82, 77, 73, 69, 65, 61, 58, 55, 52, 49, 46, 43, 41, 39, 37, 34, 33, 31, 29, 27,
   26, 24, 23, 22, 20, 19, 18, 17, 16, 15, 14, 14, 13, 12, 12, 11, 10, 10, 9, 9, 8, 8, 7, 7,
   6, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1};

byte eventNotes[50] = {0};          //holds recorded Midi Input NoteOn
unsigned long eventTimes[50] = {0};  //holds recorded times of Notes
byte eventVelocity[50] = {0};       // holds recorded note velocities
byte eventVoice[50] = {0};          // holds recorded voice assignment of notes

unsigned long eventTime = 0;
unsigned long startTime = 0; // created by timestamp()
int currentTime = 0;

int  eventIndex = 0;  // holds position within the 4 record arrays
bool record = 0;
bool playback = 0;

//current note on's for Arduino pin voices A, B, and C.  Use note value 0 for "not on".

byte CurrentNoteOn[3] = {0, 0, 0};

int freqA = 100;  //variables to set voice frequencies
int freqB = 100;
int freqC = 100;
int freqD = 100;
```

```
unsigned long deltamicro = 0;
unsigned long lastmicro = 0;

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


void setup() {

delay(1000);
tone(VOICEA, 64, 200);
delay(300);
tone(VOICEA, 164, 200);

pinMode(LED1, OUTPUT);    //turn on LED1 as power indicator
digitalWrite(LED1, HIGH);

pinMode(LED2, OUTPUT);
digitalWrite(LED2, LOW);


pinMode(VOICEA, OUTPUT);    //Lows enable distortion gates
digitalWrite(VOICEA, LOW);
pinMode(VOICEB, OUTPUT);
digitalWrite(VOICEB, LOW);
pinMode(VOICEC, OUTPUT);
digitalWrite(VOICEC, LOW);
pinMode(VOICED, OUTPUT);
digitalWrite(VOICED, LOW);

pinMode(S1, INPUT);  // Set up switch inputs with pullup resistors
digitalWrite(S1, HIGH);
pinMode(S2, INPUT);
digitalWrite(S2, HIGH);
pinMode(S3, INPUT);
digitalWrite(S3, HIGH);
pinMode(S4, INPUT);
digitalWrite(S4, HIGH);

// Serial.begin(115200);   //Used to read loop time

} //End of Setup

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

void loop() {

//  deltamicro = micros() - lastmicro;   //Used to read loop time  (44usec with an arduino Mega board ??)
//  lastmicro = micros();               // Loop Frequency = 1/deltamicro  (deltamicro is in microseconds)

// Arduino's tone() function can only be used to set up a squarewave on one output - VoiceA.
// Two more voices are created from a fast loop time decrementing two "freq" values and toggling voice outputs
// when they reach zero.  The "freq" values determine the frequency of the voices.
```

```
//VoiceA ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    tone(VOICEA, (50 + analogRead(POT5)));  // set frequency of VoiceA using tone, used as modulator


 //VoiceB ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   if (CurrentNoteOn[0] != 0) {
     --freqB;                                  // decrement Voice B freqB counter
     if (freqB <= 0){
        digitalWrite(VOICEB, !digitalRead(VOICEB));  //toggle Voice B pin when counter reaches zero
        freqB = freqTable[CurrentNoteOn[0]];         //reload counter from table

   }
     }
   else { digitalWrite(VOICEB, LOW); }

 //VoiceC ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   if (CurrentNoteOn[1] != 0) {                    // same as above for  Voice C
     --freqC;
     if (freqC <= 0){
        digitalWrite(VOICEC, !digitalRead(VOICEC));
       freqC = freqTable[CurrentNoteOn[1]];

   }
     }
   else { digitalWrite(VOICEC, LOW); }
//VoiceD ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   if (CurrentNoteOn[2] != 0) {                    // same as above for  Voice D
     --freqD;
     if (freqD <= 0){
        digitalWrite(VOICED, !digitalRead(VOICED));
       freqD = freqTable[CurrentNoteOn[2]];

   }
     }
   else { digitalWrite(VOICED, LOW); }

//-------------------------------------------------------------------------------
//
//---------------Playback of Notes[] at Times []-----------------------------
//

   while (playback) {

       currentTime = dur();
       while (eventTimes[eventIndex] <= currentTime){

       playEvent(eventNotes[eventIndex], eventVelocity[eventIndex], eventVoice[eventIndex]);

                eventIndex = eventIndex + 1;
                if (eventNotes[eventIndex] == 255){   // no more recorded notes
```

```
                    eventIndex = 0;             // rewind to beginning
                    timestamp();
                    break;
                }  //  end of rewind
        } // end of while
                    break;
    } //end of while playback


//-------------------------------------------------------------------------

//--------------switches used to start and stop record or playback ------------

//-------------------------------------------------------------------------


            if (digitalRead(S1) && !record) {  //start record
               playback = 0;
               record = 1;
               digitalWrite(LED2, HIGH);
               eventIndex = 0;
                     for (int i=0; i < 50; i++){
                        eventNotes[i] = 255;
                        eventTimes[i] = 0;
                     }
                timestamp();

            }

             else if (!digitalRead(S1) && record){  //stop record
               playback = 0;
               record = 0;
               digitalWrite(LED2, LOW);
            }

            else if (digitalRead(S2) && !playback){  //start playback
               record = 0;
               digitalWrite(LED2, HIGH);
               playback = 1;
               eventIndex = 0;
               timestamp();
            }

            else if (!digitalRead(S2) && playback){  //stop playback
               record = 0;
               digitalWrite(LED2, LOW);
               playback = 0;
               CurrentNoteOn[0] = 0;
               CurrentNoteOn[1] = 0;
               CurrentNoteOn[2] = 0;
            }
```

```
midiEventPacket_t rx = MidiUSB.read();

  switch (rx.header) {

    case 0:
          break; //No pending events

    case 0x9:  //NoteOn

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   //   Serial.println(deltamicro);  //used to find the loop time in microseconds

   if (rx.byte3 == 0){    //  Note OFF, note velocity of zero.

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

              if (CurrentNoteOn[x] == rx.byte2){ // voice playing to turn off?

                        switch (x) {
                        case 0:
                          CurrentNoteOn[x] = 0;
                          digitalWrite(VOICEB, LOW);
                          if (record){ loadEventArrays(rx.byte2, rx.byte3, x); }
                          break;
                        case 1:
                          CurrentNoteOn[x] = 0;
                          digitalWrite(VOICEC, LOW);
                          if (record){ loadEventArrays(rx.byte2, rx.byte3, x); }
                          break;
                        case 2:
                          CurrentNoteOn[x] = 0;
                          digitalWrite(VOICED, LOW);
                          if (record){ loadEventArrays(rx.byte2, rx.byte3, x); }
                          break;
                        }  // end of switch

              }  // end of Note check
              }  // end of for loop
   }
   else{    //Note ON,  note velocity not zero
          for (int x = 0; x < 3; x ++){    // search through the 3 Arduino pin voices

              if (CurrentNoteOn[x] == 0){      // this voice is available
                    CurrentNoteOn[x] = rx.byte2;  //pitch
                    if (record){ loadEventArrays(rx.byte2, rx.byte3, x); }
                    break;  //  break out of voice search loop

              }  // end of Note On check
              }  // end of for loop
   }
        break;

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

```
  case 0x8: // Note Off

  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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

  if (CurrentNoteOn[x] == rx.byte2){    // which voice is playing the note to turn off?

            switch (x) {
            case 0:
              CurrentNoteOn[x] = 0;
               digitalWrite(VOICEB, LOW);
              if (record){ loadEventArrays(rx.byte2, 0, 0); }
              break;
            case 1:
              CurrentNoteOn[x] = 0;
               digitalWrite(VOICEC, LOW);
              if (record){ loadEventArrays(rx.byte2, 0, 1); }
              break;
            case 2:
              CurrentNoteOn[x] = 0;
               digitalWrite(VOICED, LOW);
              if (record){ loadEventArrays(rx.byte2, 0, 2); }
              break;
            }  // end of switch

      }  // end of Note check


      }  // end of for loop
          break;
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  case 0xB:  // control Change
          break;

  default:
           break;

 }


}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  END OF MAIN LOOP
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                  Event Functions
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void loadEventArrays(int note, int velocity, int voice){
 eventTimes[eventIndex] = dur();
 eventNotes[eventIndex] = note;
 eventVelocity[eventIndex] = velocity;
 eventVoice[eventIndex] = voice;
 eventIndex = eventIndex + 1;
}

void playEvent(int n, int v, int voice){
      switch (voice) {
            case 0:
             if (v == 0) {CurrentNoteOn[0] = 0;}
             else {CurrentNoteOn[0] = n;}
             break;
           case 1:
             if (v == 0) {CurrentNoteOn[1] = 0;}
             else {CurrentNoteOn[1] = n;}
             break;
           case 2:
            if (v == 0) {CurrentNoteOn[2] = 0;}
             else {CurrentNoteOn[2] = n;}
             break;
            }  // end of switch
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//            TIMER  Loop Functions
//  ____________________________________________________________________________

void timestamp() {startTime = millis(); };
// store current time from the running clock millis()

unsigned long dur(){
return (millis() - startTime);
}
// returns the current time minus the last store timestamp

void waitTill(unsigned long msec) {
 while (dur() < msec) {};
}
//wait till the time duration from timestamp equals the given time in msec

//  ____________________________________________________________________________
```