



The Majestic SID Radio

The Majestic SID Radio is basically a classic game synthesizer chip, the SID 6581, controlled by an Arduino Micro microprocessor combined with the "Radio Music" Euro module.

SID History (from Wikipedia)

The SID was devised by engineer Robert "Bob" Yannes, who later co-founded the Ensoniq digital synthesizer company. The chip, like the first product using it, the Commodore 64, was finished in time for the Consumer Electronics Show in the first weekend of January 1982. Even though Yannes was partly displeased with the result, his colleague Charles Winterble said: "This thing is already 10 times better than anything out there and 20 times better than it needs to be."

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 Machinehome computers. It was one of the first sound chips of its kind to be included in a home computer prior to the digital sound revolution.

SID synth Basics

The 6581 has three synthesizer "voices" which can be used independently or in conjunction with each other (or external audio sources) to create complex sounds. Each voice consists of a tone oscillator/waveform generator, an envelope generator and an amplitude modulator. The tone oscillator produces one of four waveforms at a selected frequency. The volume of each oscillator is controlled by an ADSR envelope generator. When triggered by a Gate signal, the ADSR envelope generator creates programmable rates of increasing and decreasing volume for the voice. All three voices plus an external audio input can be routed through a programmable filter.

SID CHIP Specifications

Three separately programmable independent audio oscillators (8 octave range, approximately 16 - 4000 Hz)

Four different waveforms per audio oscillator (sawtooth, triangle, pulse, noise)

One programmable filter featuring low-pass, high-pass and band-pass outputs. Bandpass with 6 dB/oct. Lowpass/Highpass with 12 dB/octave. The different filter modes are sometimes combined to produce additional timbres, a notch-reject filter, for instance. Variable Resonance.

Three attack/decay/sustain/release (ADSR) volume controls (48dB exponential response range), one for each audio oscillator. Attack Rate: 2ms-8sec, Decay Rate: 6ms-24sec, Sustain Level: 0-peak volume, Release Rate: 6ms-24sec.

Ring Modulation of one voice with another.

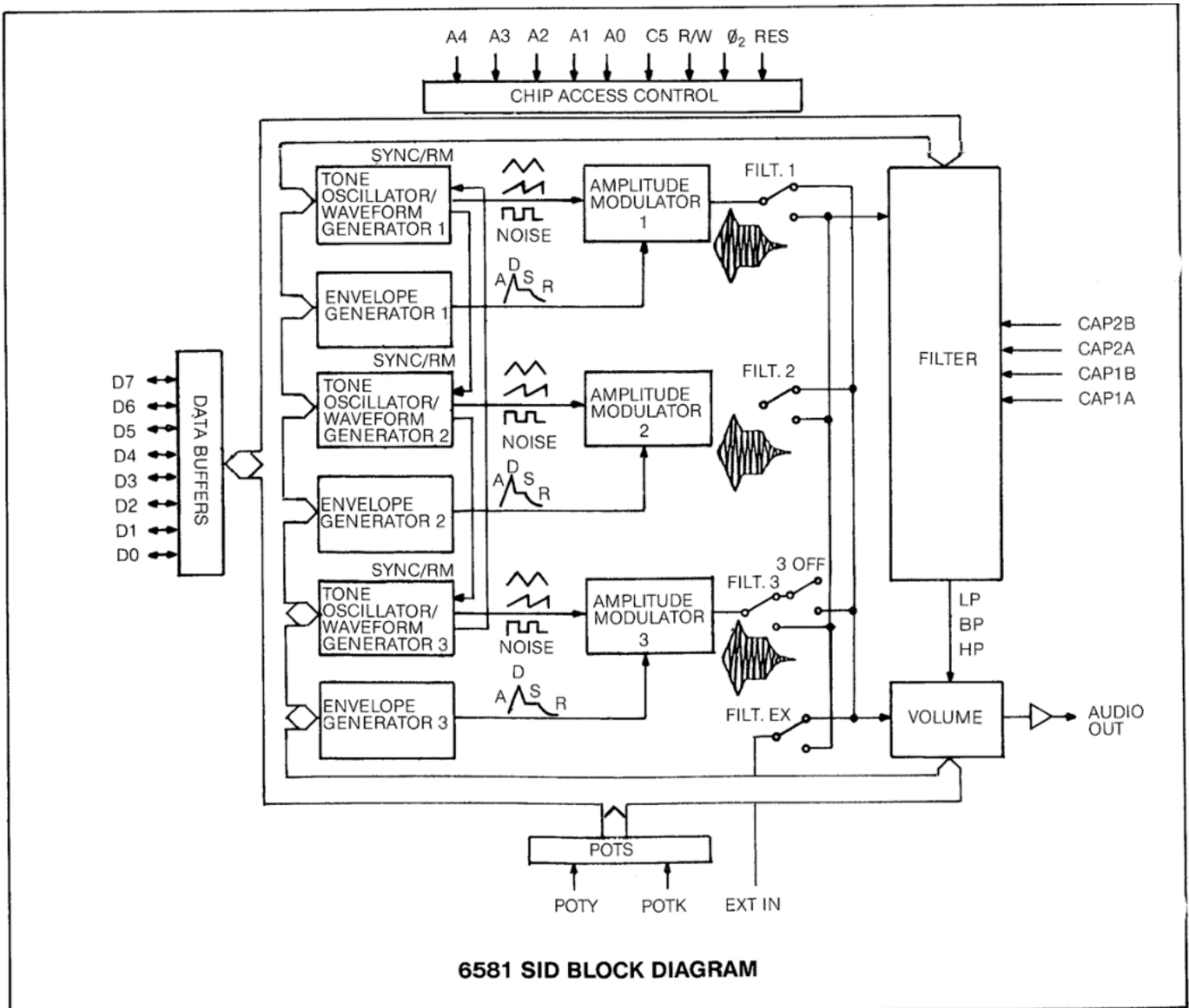
Oscillator sync with another oscillator.

Two 8-bit A/D converters (typically used for game control paddles) for external pot controls

External audio input (for sound mixing with external signal sources). Can be routed through the filter.

Random number/modulation generator

Master Volume Control



6581 SID BLOCK DIAGRAM

SID Chip Programmable Registers

There are 29 8-bit registers in the SID chip to control the generation of sound.

Each of the three Voices have 7 Write-only Registers:

- Low Frequency (**FreqLo**) 8-bit fine tune
- High Frequency (**FreqHi**) 8-bit course tune
- Pulse Width Low (**PulseWLo**) 8-bit fine set of Pulse Waveform duty cycle
- Pulse Width High (**PulseWHi**) 4-bit course set of Pulse Waveform duty cycle
- Control Register (**Cont**) Waveform select, RingMod, Sync, and Envelope Gate
- Envelope Attack/Decay (**EnvAD**) 4-bit Attack time set, 4-bit Decay time set
- Envelope Sustain/Release (**EnvSR**) 4-bit Sustain Level, 4-bit Release time set

The Programmable Filter is controlled by 4 Write-only Registers:

- Center Frequency Low (**FCLo**) 3-bit fine tune of filter's cutoff frequency
- Center Frequency High (**FCHi**) 8-bit course tune of the cutoff frequency
- Resonance/Bypass (**FiltRes**) 4-bit Resonance set, Filter Bypass Ext/3/2/1
- Mode/Volume (**ModVol**) 4-bit filter select (3Off/HP/BP/LP), 4-bit Output Volume

Finally, there are 4 Read-Only registers that the microprocessor can use to modulate voice or filter values:

- POTX (**potX**) 8-bit A/D value from an external PotX controller
- POTY (**potY**) 8-bit A/D value from an external PotY controller
- OSC3/Random (**osc3_rand**) 8-bit value from Voice 3 output, (modulate other voices?)
- ENV3 (**env3**) 8-bit value from Voice 3's Envelope, (modulate the filter frequency?)

More detailed descriptions can be found in the SID Datasheet.

SID CONTROL REGISTERS

There are 29 eight-bit registers in SID which control the generation of sound. These registers are either WRITE-only or READ-only and are listed below in Table 1.









Address		Reg #	Data								Reg Name	Reg Type				
A4	A3	A2	A1	A0	(Hex)	D7	D6	D5	D4	D3	D2	D1	D0			
VOICE 1																
0	0	0	0	0	0	00	F7	F6	F5	F4	F3	F2	F1	F0	Freq Lo	Write-only
1	0	0	0	0	1	01	F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
2	0	0	0	1	0	02	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
3	0	0	0	1	1	03	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
4	0	0	1	0	0	04	NOISE				TEST		SYNC	GATE	Control Reg	Write-only
5	0	0	1	0	1	05	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	Attack/Decay	Write-only
6	0	0	1	1	0	06	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0	Sustain/Release	Write-only
VOICE 2																
7	0	0	1	1	1	07	F7	F6	F5	F4	F3	F2	F1	F0	Freq LO	Write-only
8	0	1	0	0	0	08	F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
9	0	1	0	0	1	09	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
10	0	1	0	1	0	0A	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
11	0	1	0	1	1	0B	NOISE				TEST		SYNC	GATE	Control Reg	Write-only
12	0	1	1	0	0	0C	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	Attack/Decay	Write-only
13	0	1	1	0	1	0D	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0	Sustain/Release	Write-only
VOICE 3																
14	0	1	1	1	0	0E	F7	F6	F5	F4	F3	F2	F1	F0	Freq Lo	Write-only
15	0	1	1	1	1	0F	F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
16	1	0	0	0	0	10	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
17	1	0	0	0	1	11	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
18	1	0	0	1	0	12	NOISE				TEST		SYNC	GATE	Control Reg	Write-only
19	1	0	0	1	1	13	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	Attack/Decay	Write-only
20	1	0	1	0	0	14	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0	Sustain/Release	Write-only
Filter																
21	1	0	1	0	1	15	—	—	—	—	—	FC2	FC1	FC0	FC LO	Write-only
22	1	0	1	1	0	16	FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3	FC HI	Write-only
23	1	0	1	1	1	17	RES3	RES2	RES1	RES0	Filt EX	Filt 3	Filt 2	Filt 1	RES/Filt	Write-only
24	1	1	0	0	0	18	3 OFF	HP	BP	LP	VOL3	VOL2	VOL1	VOL0	Mode/Vol	Write-only
Misc																
25	1	1	0	0	1	19	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	POTX	Read-only
26	1	1	0	1	0	1A	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	POTY	Read-only
27	1	1	0	1	1	1B	07	06	05	04	03	02	01	00	OSC3/Random	Read-only
28	1	1	1	0	0	1C	E7	E6	E5	E4	E3	E2	E1	E0	ENV3	Read-only

TABLE 1 — SID REGISTER MAP

The Arduino Microprocessor

An Arduino Pro Micro is used to control the SID chip and provide analog and digital controllers that can manipulate the SID registers and/or your interactions with them. On the back of the Majestic SID Radio is a USB jack used to connect the Arduino to a USB port on any desktop computer. Download the Arduino IDE (Integrated Development Environment) software to your desktop computer and use it to program your own methods of interacting with the SID synthesizer.

In addition to providing complete control over the SID synthesizer, the Arduino can also read the outputs of several controller devices. These include two toggle switches, two softpot/ribbon controllers, two slide pots, a multi-turn pot, and a regular rotary pot. The SID chip itself has the ability to read two external pots. That makes a total of two digital inputs and 8 analog inputs that can be programmed to manually manipulate the SID synthesizer parameters or your interaction with them.

One Arduino output pin, D1, has been assigned to modulate and distort the final output signal of the SID chip. D1 is a digital output pin, either high or low. When low, it does nothing to the output signal, allowing it to pass unaffected. When high, it can go between completely cutting off the signal to mildly distorting the signal, as set by the “Volume” knob on the front panel. For the simplest application, program the Arduino to set D1 high and then use the “Volume” control to adjust the amount of signal distortion and amplitude. For more drastic effects, use the Arduino TONE command to set up an audio or sub audio frequency on D1, and adjust the amount of modulation with the front panel “Volume” control.

Programming the SID Synthesizer

Programming the SID Synthesizer through the Arduino requires a detailed study of the SID Datasheet along with a look at the circuit diagrams showing specifically how the SID chip is connected to the Arduino, all of which is provided in this documentation. Most of this detail work, however, is not necessary if the SID program template is used. This template sets up all the

variables and functions needed to program the SID synthesizer. Here is an overview of the variables and functions built into the SID Program Template.

SID Voice Variables

Matrix variables for the SID voice registers. The range of values are shown in parenthesis. The Voice number 1, 2, or 3 is specified within the matrix brackets.

Attack[voice] (0 to 15)

Decay[voice] (0 to 15)

Sustain[voice] (0 to 15)

Release[voice] (0 to 15)

FreqLo[voice] (0 to 255)

FreqHi[voice] (0 to 255)

PulseWLo[voice] (0 to 255)

PulseWHi[voice] (0 to 15)

Waveshape[voice] (Load by adding the Waveshape Constants shown below)

SID Constants

**Constants for the Waveshape variable. Add the constants together for multiple selections, for example, to make voice 2 ring modulate with a Sawtooth run:

```
Waveshape[2] = SAWTOOTH + RINGMOD;
```

NOISE

PULSE

SAWTOOTH

TRIANGLE

TEST

RINGMOD

SYNC

Constants for the **ldResFilt() function. This function sets the resonance of the filter, but it also determines which voices are sent through the filter and which

will bypass the filter. For example, to send only Voices 1 and 2 through the filter and set the resonance to half, run the function:

```
ldResFilt(FILT1 + FILT2, 7);
```

FILTEX (send the External Input signal through the filter, else bypass it)

FILT3 (send Voice 3 through the filter, else bypass it)

FILT2 (send Voice 2 through the filter, else bypass it)

FILT1 (send Voice 1 through the filter, else bypass it)

****Constants for the `ldModeVol()` function.** This function sets the filter type along with the SID output volume. For example, to make the filter a lowpass and set the output volume to full, run this function:

```
ldModeVol( LP, 15);
```

OFF3 (no Voice 3 on the SID output. Useful when using Voice 3 for Ring Mod)

HP (high pass filter)

BP (band pass filter)

LP (lowpass filter)

BP + LP (band reject filter)

SID Controller Variables

Here are containers for the four SID Read Registers. Use the function `readRegisters()` to fill these variables with current values.

potX

potY

osc3_rand

env3

Here are containers for the controllers on the Majestic Radio box along with their usage, including Arduino pin numbers. Use the function `loadSensors()` to load all of them with current values.

ribbon1 = analogRead(A0);

ribbon2 = analogRead(A1);

```
slider1 = analogRead(A2);
slider2 = analogRead(A3);
middlePot = analogRead(A4);
rightPot = analogRead(A5);
switch1 = digitalRead(0);
switch2 = digitalRead(13);
```

TONEpin

A convenient constant for the modulation/distortion pin. To set it high run: **digitalWrite(TONEpin, HIGH);** To turn it into a modulating squarewave run: **tone(TONEpin, frequency);** or **tone(TONEpin, frequency, duration);** where frequency is in Hertz and duration (optional) is in milliseconds.

Functions

loadAddress(address); low level selection of a SID register

loadData(data); low level loading of data into the selected SID register

resetSID(); load zeros into all the SID registers

readRegisters(); fill potX, potY, osc3_rand, and env3 with current values

loadSensors(); read all 8 controllers and fill their variables with current data

ldFreqLo(voice); load fine tune frequency from FreqLo[voice]

ldFreqHi(voice); load course tune frequency from FreqHi[voice]

ldPulseWLo(voice); load fine pulse width from PulseWLo[voice]

ldPulseWHi(voice); load course pulse width from PulseWHi[voice]

ldEnvAD(voice); load Attack[voice] and Decay[voice] data

ldEnvSR(voice); load Sustain[voice] and Release[voice] data

ldFCLo(data); load fine tune Filter Cutoff frequency data (0 - 7)

ldFCHi(data); load course tune Filter Cutoff frequency data (0 - 255)

ldFiltRes(filt, res); FILTEX/FILT3/FILT2/FILT1, and filter resonance (0-15)

ldModeVol(mode, vol); 3OFF/HP/BP/LP, and output volume (0-15)

ldGate(voice, gate);

start the voice ADSR envelope (Attack/Decay/Sustain sections) with gate=1, or end the voice ADSR envelope (Release section) with gate=0. Also loads the Waveshape[voice] matrix data.

Useful Event Timer Functions

```
unsigned long timestamp = 0;  
void timeStamp() {timestamp = millis(); }  
//store the current time from the running clock millis()
```

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

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

The SID External Input

The SID chip incorporates an external audio input, which is mixed with the 3 oscillator voices. The external input can also be routed through the SID filter. The choice of sending each of the four signals through the filter or bypassing it is set up in the **FiltRes** register. The final SID output is then a mix of three filtered or unfiltered SID voices plus the filtered or unfiltered external audio input. The volume of this final mix is controlled by the master volume control in the **ModVol** register. In the Majestic SID Radio box this final mix is also affected by the previously described Modulation/Distortion D1 signal controlled by the front panel "Volume" pot.

A top panel Input Jack paired with an Input Volume control allows any audio signal to be connected to the SID external input. (The signal should be no more than 3 volts peak to peak). If nothing is inserted into the Input jack, the jack is normaled to one of two built-in sources for the External Input. A top panel red

pushbutton chooses between the output of the RadioMusic Euro module or an Electret Microphone positioned under the top panel fabric.

The Radio Music Euro Module

The RadioMusic Module is a looping sample playback device. Like a radio, this module works on a series of banks and stations. Each of the 16 banks can contain many different stations. Each station is .raw audio file stored in a bank directory on the SD card. Choose a bank by pressing and holding the RESET switch. Choose a station by turning the STATION knob or plugging a voltage into TUNE.

The SD Card can be Fat16 or Fat32 formatted. It can contain up to 16 folders, which become the “banks” in the module, and are labeled “0”, “1”, “2”, etc (without the quotes). Each folder can have up to 75 soundfiles, which become the “stations” in the module, though fewer files make them easier to select when using the STATION knob. The soundfiles must be mono, 16bit, 44.1kHz rate. They must also be headerless Wav files with the .raw suffix. Both Amadeus Pro and Audacity apps can create these types of files.

More details of the RadioMusic operation can be found in the included manual.

The Majestic SID Radio Panels

The cabinet used to hold the SID chip, Arduino, RadioMusic and Controllers is an antique radio with a brass front plate. What follows are descriptions and illustrations of Radio Panels.

Music
Thing
Modular



STATION



START



RESET
Hold for bank

STATION START



RESET OUT



RADIO
MUSIC

Front Panel

There are three rotary pots on the brass front panel. The left pot, labeled "VOLUME", is hardwired to control the amount of modulation/distortion originating from the Arduino D1 output pin. The middle control, labeled "PHONO RADIO", is a multi-turn pot connected to the Arduino Analog Input A4. The right control, labeled "TUNING", is connected to the Arduino Analog Input A5.

Two slide pots have been placed in the radio's tuning window. The upper slide control is connected to the Arduino Analog Input A2 and the lower one is connected to A3.

Top of Radio

The top of the Radio has a fabric insert under which a speaker is centered. To the left of the speaker, under the fabric, is a small Electret Microphone. The close positioning of the speaker and microphone allows for feedback distortion controlled by the Input Volume control.

Also on top of the cabinet are two 8-inch Soft Pots, or ribbon controllers. These are connected to the Arduino Analog inputs A0 (back) and A1 (front). The rotary pot to the left of each ribbon and the switch to the right require some explanation.

When the ribbon controller is not pressed, there is no connection to the Arduino analog input. Usually the Arduino input is merely tied to GND (zero voltage) through a resistor to provide a zero reading when the controller is not being used. However, it was decided to do something more interesting here – make the grounded resistor variable and powered or not through a switch.

When the switch is engaged and the ribbon is not being used, the pot will act as a full range controller. When the ribbon is pressed, the pot acts as an offset adjustment to the ribbon readings.

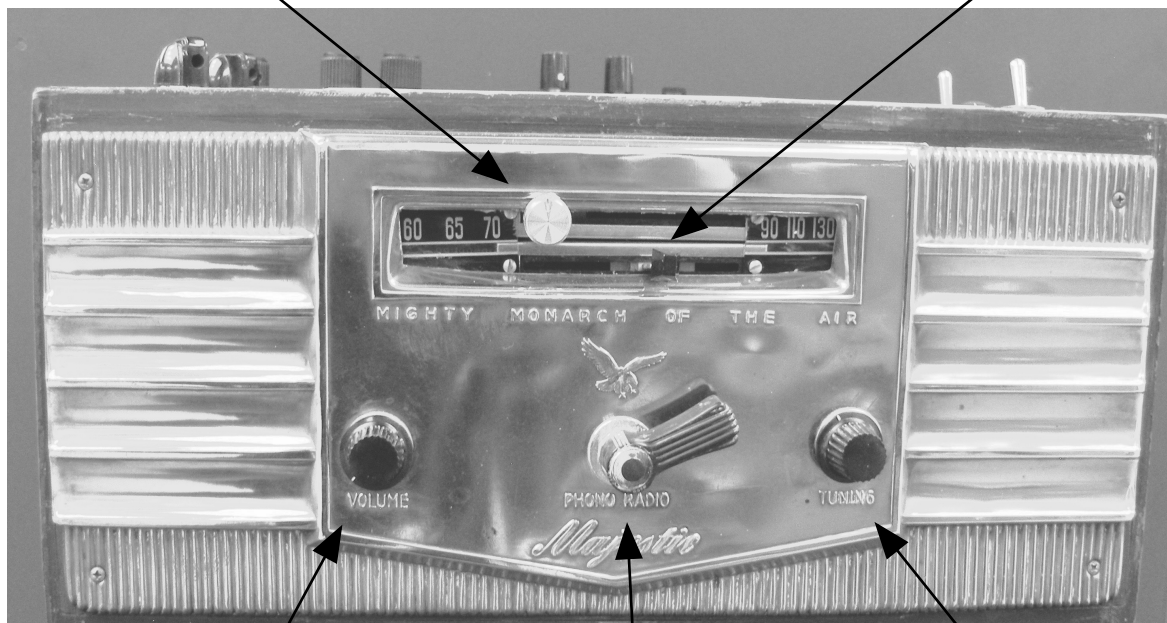
When the switch is disengaged and the ribbon is not being used, the pot is just a connection to Ground producing a zero output no matter what its setting. When the ribbon is pressed, the pot act as a sort of output range adjustment for the ribbon output.

Top Slide Pot

Arduino A2

Lower Slide Pot

Arduino A3



“Volume”
Amount of Modulation/Distortion
from
Arduino D1 Output

“Phono Radio”
MultiTurn Pot
Arduino A4

“Tuning”
Right Pot
Arduino A5

Also note the toggle switch between the two ribbon switches; this is the On/Off Power switch for the whole radio unit.

Top back panel

At the left edge of the top pack panel is an Output jack to allow connection of the output signal to any sound system. When no connection is made to this jack, the output signal is fed to the “less than hi-fidelity” internal speaker.

As a side note, a small circuit board kit called the “Snooper” is part of the Radio circuitry. It provides the LM386 audio amp to power the internal speaker, and an op-amp preamp circuit to boost the internal microphone signal.

Next in line on the left is a volume control and input jack for the External Audio Input to the SID chip. A red pushbutton selects between the RadioMusic Module and the Electret mic for the Input Jack’s normaLED connection.

Two sets of 8 LEDs are connected to various internal SID/Arduino pins as labeled in the circuit diagrams.

In front of the LEDs, the RadioMusic euro module is mounted. It can function on its own using its own output and input jacks, or it can be used in conjunction with the SID chip since its output is also hardwired to the External Input selector.

Finally, two toggle switches are connected to the Arduino digital input pins D0 and IO13 to act as digital (high or low) controllers. Please note that D0 also function as the USB signal input when programming the Arduino. It would be helpful to leave this switch in the Off position (0) whenever the USB is connected for programming.

External Input to SID
Jack and Volume Control
Normaled to Radio Music Euro or Internal Mic

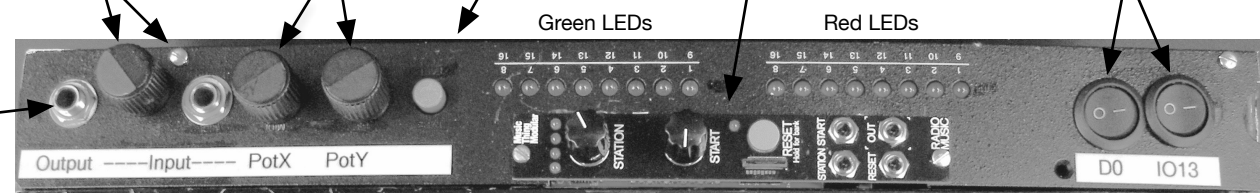
Red Selector Button
Radio Music or Internal Mic

Two SID Control Pots
POTX, POTY

Radio Music Euro Module

Toggle Switches
Arduino D0, IO13

Output Jack
Normaled to Internal Speaker



Adjust Pots for
Ribbons

Switch for
Ribbon

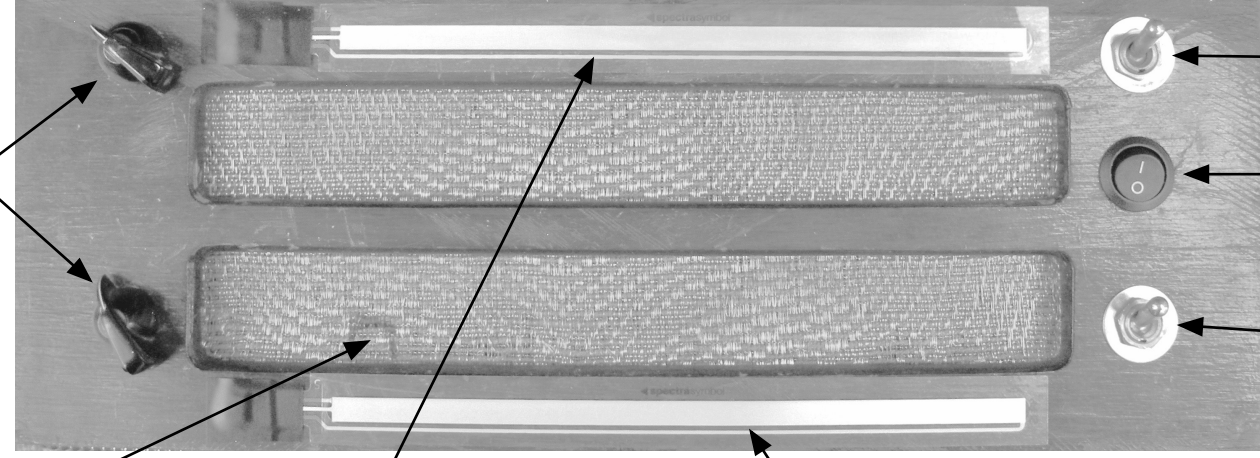
POWER SWITCH

Switch for
Ribbon

Internal Mic

Back SoftPot/Ribbon Controller
Arduino A0

Front SoftPot/Ribbon Controller
Arduino A1



Commodore SID 6581 DataSheet

<http://www.waitingforfriday.com/?p=661>

Commodore SID 6581 Datasheet - WFFwiki

Introduction

This article is a reproduction of the original Commodore 6581 Sound Interface Device (SID) datasheet. I made this by taking a photocopy of an original document and using OCR to capture the content, then the document was hand-edited, formatted for mediawiki and reassembled here with (cleaned-up and straightened) diagrams and tables.

The reason for this was that most sources on the web are low quality PDFs and, since the documents are graphical copies of the original, they cannot be searched or indexed. The SID chip is a complex device, so I hope anyone developing projects around this device will find this mediawiki format datasheet useful.

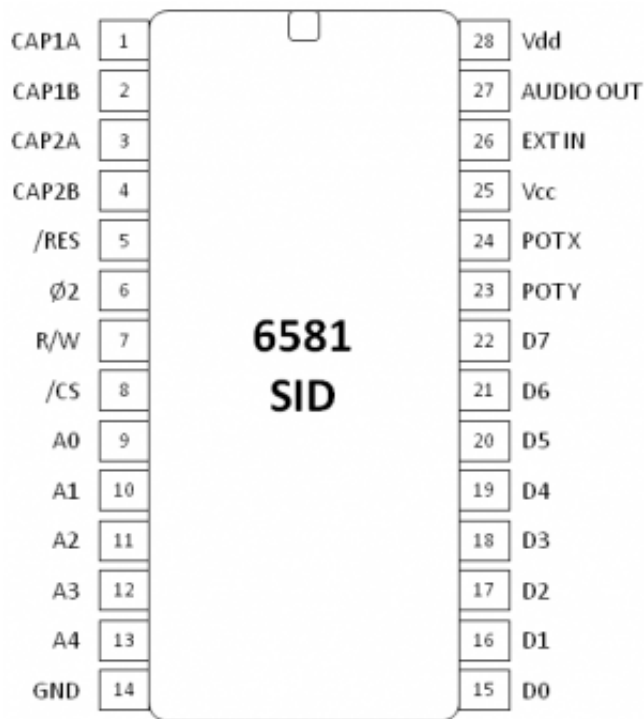
Since it was converted mainly by hand (OCR is not terribly accurate!) I would appreciate it if you could notify me of any errors or omissions you find so I can make this as accurate as possible. If you would like to make a copy of this document please note the Creative Commons licence referenced at the bottom of the page.

Concept

The 6581 Sound Interface Device (SID) is a single-chip, 3-voice electronic music synthesizer/sound effects generator compatible with the 65XX and similar microprocessor families. SID provides wide-range, high-resolution control of pitch (frequency), tone color (harmonic content) and dynamics (volume). Specialized control circuitry minimizes software overhead, facilitating use in arcade/home video games and low-cost musical instruments.

- - Cutoff range: 30 Hz-12 kHz
 - 12 dB/octave Rolloff
 - Low pass, Band pass, High pass, Notch outputs
 - Variable Resonance

6581 Pin Configuration

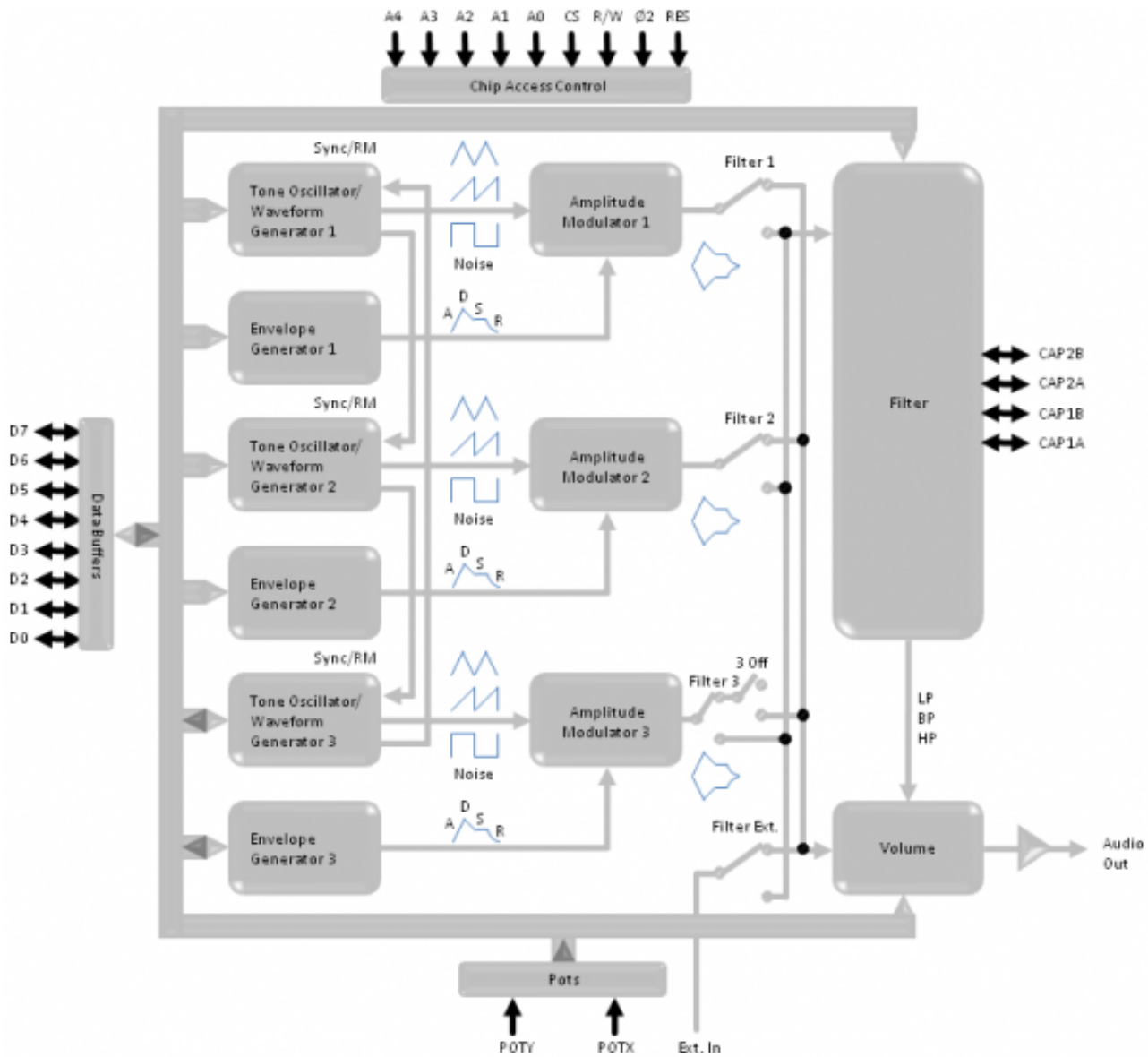


Description

The 6581 consists of three synthesizer “voices” which can be used independently or in conjunction with each other (or external audio sources) to create complex sounds. Each voice consists of a Tone Oscillator/Waveform Generator, an Envelope Generator and an Amplitude Modulator. The Tone Oscillator controls the pitch of the voice over a wide range. The Oscillator produces four waveforms at the selected frequency, with the unique harmonic content of each waveform providing simple control of tone color. The volume dynamics of the oscillator are controlled by the Amplitude Modulator under the direction of the Envelope Generator. When triggered, the Envelope Generator creates an amplitude envelope with programmable rates of increasing and decreasing volume. In addition to the three voices, a programmable Filter is provided for generating complex, dynamic tone colors via subtractive synthesis.

SID allows the microprocessor to read the changing output of the third Oscillator and third Envelope Generator. These outputs can be used as a source of modulation information for creating vibrato, frequency/filter sweeps and similar effects. The third oscillator can also act as a random number generator for games. Two A/D converters are provided for interfacing SID with potentiometers. These can be used for “paddles” in a game environment or as front panel controls in a music synthesizer. SID can process external audio signals, allowing multiple SID chips to be daisy-chained or mixed in complex polyphonic systems.

6581 SID Block Diagram



SID Control Registers

There are 29 eight-bit registers in SID which control the generation of sound. These registers are either WRITE-only or READ-only and are listed below in Table 1.

Address					Reg #	Data								Reg Name	Reg Type
A4	A3	A2	A1	A0	(Hex)	D7	D6	D5	D4	D3	D2	D1	D0		
VOICE 1															
0	0	0	0	0	00	F7	F6	F5	F4	F3	F2	F1	F0	Freq Lo	Write-only
1	0	0	0	0	01	F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
2	0	0	0	1	02	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
3	0	0	0	1	03	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
4	0	0	1	0	04	NOISE				TEST		SYNC	GATE	Control Reg	Write-only
5	0	0	1	0	05	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	Attack/Decay	Write-only
6	0	0	1	0	06	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0	Sustain/Release	Write-only
VOICE 2															
7	0	0	1	1	07	F7	F6	F5	F4	F3	F2	F1	F0	Freq LO	Write-only
8	0	1	0	0	08	F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
9	0	1	0	0	09	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
10	0	1	0	1	0A	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
11	0	1	0	1	0B	NOISE				TEST		SYNC	GATE	Control Reg	Write-only
12	0	1	1	0	0C	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	Attack/Decay	Write-only
13	0	1	1	0	0D	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0	Sustain/Release	Write-only
VOICE 3															
14	0	1	1	1	0E	F7	F6	F5	F4	F3	F2	F1	F0	Freq Lo	Write-only
15	0	1	1	1	0F	F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
16	1	0	0	0	10	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
17	1	0	0	0	11	—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
18	1	0	0	1	12	NOISE				TEST		SYNC	GATE	Control Reg	Write-only
19	1	0	0	1	13	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	Attack/Decay	Write-only
20	1	0	1	0	14	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0	Sustain/Release	Write-only
Filter															
21	1	0	1	0	15	—	—	—	—	—	FC2	FC1	FC0	FC LO	Write-only
22	1	0	1	1	16	FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3	FC HI	Write-only
23	1	0	1	1	17	RES3	RES2	RES1	RES0	Filt EX	Filt 3	Filt 2	Filt 1	RES/Filt	Write-only
24	1	1	0	0	18	3 OFF	HP	BP	LP	VOL3	VOL2	VOL1	VOL0	Mode/Vol	Write-only
Misc															
25	1	1	0	0	19	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	POTX	Read-only
26	1	1	0	1	1A	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	POTY	Read-only
27	1	1	0	1	1B	07	06	05	04	03	02	01	00	OSC3/Random	Read-only
28	1	1	1	0	1C	E7	E6	E5	E4	E3	E2	E1	E0	ENV3	Read-only

Table 1

SID Register Description

Voice 1

Freq Lo/Freq Hi (Registers 00-01)

Together these registers form a 16-bit number which linearly controls the Frequency of Oscillator 1. The frequency is determined by the following equation:

$$F_{out} = (F_n * F_{clk}/16777216) \text{ Hz}$$

Where F_n is the 16-bit number in the Frequency registers and F_{clk} is the system clock applied to the $\emptyset 2$ input (pin 6). For a standard 1.0 Mhz clock, the frequency is given by:

$$F_{out} = (F_n * 0.0596) \text{ Hz}$$

A complete table of values for generating 8 octaves of the equally-tempered musical scale with concert A

(440 Hz) tuning is provided in [Appendix A](#). It should be noted that the frequency resolution of SID is sufficient for any tuning scale and allows sweeping from note to note (portamento) with no discernible frequency steps.

PW Lo/PW Hi (Registers 02-03)

Together these registers form a 12-bit number (bits 4-7 of PW Hi are not used) which linearly controls the Pulse Width (duty cycle) of the Pulse waveform on Oscillator 1. The pulse width is determined by the following equation:

$$PW_{out} = (PW_n / 40.95) \%$$

Where PW_n is the 12-bit number in the Pulse Width registers.

The pulse width resolution allows the width to be smoothly swept with no discernible stepping. Note that the Pulse waveform on Oscillator 1 must be selected in order for the Pulse Width registers to have any audible effect. A value of 0 or 4095 (\$FFF) in the Pulse Width registers will produce a constant DC output, while a value of 2048 (\$800) will produce a square wave.

Control Register (Register 04)

This register contains eight control bits which select various options on Oscillator 1.

Gate (Bit 0)

The GATE bit controls the Envelope Generator for Voice 1. When this bit is set to a one, the Envelope Generator is Gated (triggered) and the ATTACK/DECAY/SUSTAIN cycle is initiated. When the bit is reset to a zero, the RELEASE cycle begins. The Envelope Generator controls the amplitude of Oscillator 1 appearing at the audio output, therefore, the GATE bit must be set (along with suitable envelope parameters) for the selected output of Oscillator 1 to be audible. A detailed discussion of the Envelope Generator can be found in [Appendix B](#).

Sync (Bit 1)

The SYNC bit, when set to a one, Synchronizes the fundamental frequency of Oscillator 1 with the fundamental frequency of Oscillator 3, producing “Hard Sync” effects. Varying the frequency of Oscillator 1 with respect to Oscillator 3 produces a wide range of complex harmonic structures from Voice 1 at the frequency of Oscillator 3. In order for sync to occur Oscillator 3 must be set to some frequency other than zero but preferably lower than the frequency of Oscillator 1. No other parameters of Voice 3 have any effect on sync.

Ring Mod (Bit 2)

The RING MOD bit, when set to a one, replaces the Triangle waveform output of Oscillator 1 with a “Ring

Modulated” combination of Oscillators 1 and 3. Varying the frequency of Oscillator 1 with respect to Oscillator 3 produces a wide range of non-harmonic overtone structures for creating bell or gong sounds and for special effects. In order for ring modulation to be audible, the Triangle waveform of Oscillator 1 must be selected and Oscillator 3 must be set to some frequency other than zero. No other parameters of Voice 3 have any effect on ring modulation.

Test (Bit 3)

The TEST bit, when set to a one, resets and locks Oscillator 1 at zero until the TEST bit is cleared. The Noise waveform output of Oscillator 1 is also reset and the Pulse waveform output is held at a DC level. Normally this bit is used for testing purposes, however, it can be used to synchronize Oscillator 1 to external events, allowing the generation of highly complex waveforms under real-time software control.

Triangle Wave (Bit 4)

When set to a one, the Triangle waveform output of Oscillator 1 is selected. The Triangle waveform is low in harmonics and has a mellow, flute-like quality.

Sawtooth Wave (Bit 5)

When set to a one, the Sawtooth waveform of Oscillator 1 is selected. The sawtooth waveform is rich in even and odd harmonics and has a bright, brassy quality.

Square Wave (Bit 6)

When set to a one, the Pulse waveform output of Oscillator 1 is selected. The harmonic content of this waveform can be adjusted by the Pulse Width registers, producing tone Qualities ranging from a bright, hollow square wave to a nasal, reedy pulse. Sweeping the pulse width in real-time produces a dynamic “phasing” effect which adds a sense of motion to the sound. Rapidly jumping between different pulse widths can produce interesting harmonic sequences.

Noise (Bit 7)

When set to a one, the Noise output waveform of Oscillator 1 is selected. This output is a random signal which changes at the frequency of Oscillator 1. The sound quality can be varied from a low rumbling to hissing white noise via the Oscillator 1 Frequency registers. Noise is useful in creating explosions, gunshots, jet engines, wind, surf and other un-pitched sounds, as well as snare drums and cymbals. Sweeping the Oscillator frequency with Noise selected produces a dramatic rushing effect. One of the output waveforms must be selected for Oscillator 1 to be audible, however it is NOT necessary to deselect waveforms to silence the output of Voice 1. The amplitude of Voice 1 at the final output is a function of the Envelope Generator only.

NOTE: The oscillator output waveforms are NOT additive. If more than one output waveform is selected

simultaneously, the result will be a logical ANDing of the waveforms. Although this technique can be used to generate additional waveforms beyond the four listed above, it must be used with care. If any other waveform is selected while Noise is on, the Noise output can “lock up”. If this occurs, the Noise output will remain silent until reset by the TEST bit or by bringing /RES (pin 5) low.

Attack/Decay (Register 05)

Bits 4-7 of this register (ATK0-ATK3) select 1 of 16 ATTACK rates for the Voice 1 Envelope Generator. The ATTACK rate determines how rapidly the output of Voice 1 rises from zero to peak amplitude when the Envelope Generator is Gated. The 16 ATTACK rates are listed below in Table 2.

Bits 0-3 (DCY0-DCY3) select 1 of 16 DECAY rates for the Envelope Generator. The DECAY cycle follows the ATTACK cycle and the DECAY rate determines how rapidly the output falls from the peak amplitude to the selected SUSTAIN level. The 16 DECAY rates are listed in Table 2.

		Attack Rate	Release Rate
DEC	HEX	(Time/Cycle)	(Time/Cycle)
0	(0)	2 mS	6 mS
1	(1)	8 mS	24 mS
2	(2)	16 mS	48 mS
3	(3)	24 mS	72 mS
4	(4)	38 mS	114 mS
5	(5)	56 mS	168 mS
6	(6)	68 mS	204 mS
7	(7)	80 mS	240 mS
8	(8)	100 mS	300 mS
9	(9)	250 mS	750 mS
10	(A)	500 mS	1.5 S
11	(B)	800 mS	2.4 S
12	(C)	1 S	3 S
13	(D)	3 S	9 S

14	(E)	5 S	15 S
15	(F)	8 S	24 S

Table 2

NOTE: Envelope rates are based on a 1.0 Mhz $\emptyset 2$ clock. For other $\emptyset 2$ frequencies, multiply the given rate by 1 Mhz / $\emptyset 2$. The rates refer to the amount of time per cycle. For example, given an ATTACK value of 2, the ATTACK cycle would take 16 mS to rise from zero to peak amplitude. The DECAY/RELEASE rates refer to the amount of time these cycles would take to fall from peak amplitude to zero.

Sustain/Release (Register 06)

Bits 4-7 of this register (STN0-STN3) select 1 of 16 SUSTAIN levels for the Envelope Generator. The SUSTAIN cycle follows the DECAY cycle and the output of Voice 1 will remain at the selected SUSTAIN amplitude as long as the Gate bit remains set. The SUSTAIN levels range from zero to peak amplitude in 16 linear steps, with a SUSTAIN value of 0 selecting zero amplitude and a SUSTAIN value of 15 (#F) selecting the peak amplitude.

A SUSTAIN value of 8 would cause Voice 1 to SUSTAIN at an amplitude one-half the peak amplitude reached by the ATTACK cycle.

Bits 0-3 (RLS0-RLS3) select 1 of 16 RELEASE rates for the Envelope Generator. The RELEASE cycle follows the SUSTAIN cycle when the Gate bit is reset to zero. At this time, the output of Voice 1 will fall from the SUSTAIN amplitude to zero amplitude at the selected RELEASE rate. The 16 RELEASE rates are identical to the DECAY rates.

NOTE: The cycling of the Envelope Generator can be altered at any point via the Gate bit. The Envelope Generator can be Gated and Released without restriction. For example, if the Gate bit is reset before the envelope has finished the ATTACK cycle, the RELEASE cycle will immediately begin, starting from whatever amplitude had been reached. If the envelope is then Gated again (before the RELEASE cycle has reached zero amplitude), another ATTACK cycle will begin, starting from whatever amplitude had been reached. This technique can be used to generate complex amplitude envelopes via real-time software control.

Voice 2

Registers 07-\$0D control Voice 2 and are functionally identical to registers 00-06 with these exceptions:

When selected, SYNC synchronizes Oscillator 2 with Oscillator 1.

When selected, RING MOD replaces the Triangle output of Oscillator 2 with the ring modulated combination of Oscillators 2 and 1.

Voice 3

Registers \$0E-\$14 control Voice 3 and are functionally identical to registers 00-06 with these exceptions:

When selected, SYNC synchronizes Oscillator 3 with Oscillator 2.

When selected, RING MOD replaces the Triangle output of Oscillator 3 with the ring modulated combination of Oscillators 3 and 2.

Typical operation of a voice consists of selecting the desired parameters: frequency, waveform effects (SYNC, RING MOD) and envelope rates, then gating the voice whenever the sound is desired. The sound can be sustained for any length of time and terminated by clearing the Gate bit. Each voice can be used separately, with independent parameters and gating, or in unison to create a single, powerful voice. When used in unison, a slight detuning of each oscillator or tuning to musical intervals creates a rich, animated sound.

Filter

FC Lo/FC Hi (Registers \$15, \$16)

Together these registers form an 11-bit number (bits 3-7 of FC LO are not used) which linearly controls the Cutoff (or Center) Frequency of the programmable Filter. The approximate Cutoff Frequency ranges between 30Hz and 10KHz with the recommended capacitor values of 2200pF for CAP1 and CAP2. The frequency range of the Filter can be altered to suit specific applications. Refer to the Pin Description section for more information.

RES/Filt (Register \$17)

Bits 4-7 of this register (RES0-RES3) control the Resonance of the Filter, resonance of a peaking effect which emphasizes frequency components at the Cutoff Frequency of the Filter, causing a sharper sound. There are 16 Resonance settings ranging linearly from no resonance (0) to maximum resonance (15 or #F).

Bits 0-3 determine which signals will be routed through the Filter:

Filt 1 (Bit 0)

When set to a zero, Voice 1 appears directly at the audio output and the Filter has no effect on it. When set to a one, Voice 1 will be processed through the Filter and the harmonic content of Voice 1 will be altered according to the selected Filter parameters.

Filt 2 (Bit 1)

Same as bit 0 for Voice 2.

Filt 3 (Bit 2)

Same as bit 0 for voice 3.

Filtex (Bit 3)

Same as bit 0 for External audio input (pin 26).

Mode/Vol (Register \$18)

Bits 4-7 of this register select various Filter mode and output options:

LP (Bit 4)

When set to a one, the low Pass output of the Filter is selected and sent to the audio output. For a given Filter input Signal, all frequency components below the Filter Cutoff Frequency are passed unaltered, while all frequency components above the Cutoff are attenuated at a rate of 12 dB/Octave. The low Pass mode produces full-bodied sounds.

BP (Bit 5)

Same as bit 4 for the Band Pass output. All frequency components above and below the Cutoff are attenuated at a rate of 6 dB/Octave. The Band Pass mode produces thin, open sounds.

HP (Bit 6)

Same as bit 4 for the High Pass output. All frequency components above the Cutoff are passed unaltered, while all frequency components below the Cutoff are attenuated at a rate of 12 dB/Octave. The High Pass mode produces tinny, buzzy sounds.

3 OFF (Bit 7)

When set to a one, the output of Voice 3 is disconnected from the direct audio path. Setting Voice 3 to bypass the Filter (FILT 3 = 0) and setting 3 OFF to a one prevents Voice 3 from reaching the audio output. This allows Voice 3 to be used for modulation purposes without any undesirable output.

NOTE: The Filter output modes ARE additive and multiple Filter modes may be selected simultaneously.

For example, both LP and HP modes can be selected to produce a Notch (or Band Reject) Filter response. In order for the Filter to have any audible effect, at least one Filter output must be selected and at least one Voice must be routed through the Filter. The Filter is, perhaps, the most important element in SID as it allows the generation of complex tone colors via subtractive synthesis. The Filter is used to eliminate specific frequency components from a harmonically-rich input signal). The best results are achieved by varying the Cutoff Frequency in real-time.

Bits 0-3 (VOL0-VOL3) select 1 of 16 overall Volume levels for the final composite audio output. The output volume levels range from no output (0) to maximum volume (15 or #F) in 16 linear steps. This control can be used as a static volume control for balancing levels in multi-chip systems or for creating dynamic volume effects, such as Tremolo. Some Volume level other than zero must be selected in order for SID to produce any sound.

Misc

POTX (Register \$19)

This register allows the microprocessor to read the position of the potentiometer tied to POTX (pin 24), with values ranging from 0 at minimum resistance, to 255 (#FF) at maximum resistance. The value is always valid and is updated every 512 \emptyset 2 clock cycles. See the Pin Description section for information on post and capacitor values.

POTY (Register \$1A)

Same as POTX for the pot tied to POTY (pin 23).

OSC 3/RANDOM (Register \$1B)

This register allows the microprocessor to read the upper 8 output bits of Oscillator 3. The character of the numbers generated is directly related to the waveform selected. If the Sawtooth waveform of Oscillator 3 is selected, this register will present a series of numbers incrementing from 0 to 255 (\$FF) at a rate determined by the frequency of Oscillator 3. If the Triangle waveform is selected, the output will increment from 0 up to 255, then decrement down to 0. If the Pulse waveform is selected, the output will jump between 0 and 255. Selecting the Noise waveform will produce a series of random numbers, therefore, this register can be used as a random number generator for games. There are numerous timing and sequencing applications for the OSC 3 register, however, the chief function is probably that of a modulation generator. The numbers generated by this register can be added, via software, to the Oscillator or Filter Frequency registers or the Pulse Width registers in real-time. Many dynamic effects can be generated in this manner. Siren-like sounds can be created by adding the OSC 3 Sawtooth output to the frequency control of another oscillator. Synthesizer "Sample and Hold" effects can be produced by adding the OSC 3 Noise output to the Filter Frequency control registers. Vibrato can be produced by setting Oscillator 3 to a frequency around 7 Hz and adding the OSC 3 Triangle output (with proper scaling) to the Frequency control of another oscillator. An unlimited range of effects are available by altering the frequency of Oscillator 3 and scaling the OSC 3 output. Normally, when Oscillator 3 is used for modulation, the audio output of Voice 3 should be eliminated (3 OFF = 1).

ENV 3 (Register \$1C)

Same as OSC 3, but this register allows the microprocessor to read the output of the Voice 3 Envelope Generator. This output can be added to the Filter Frequency to produce harmonic envelopes, WAH WAH,

and similar effects. "Phaser" sounds can be created by adding this output to the frequency control registers of an oscillator. The Voice 3 Envelope Generator must be gated in order to produce any output from this register. The OSC 3 register, however, always reflects the changing output of the oscillator and is not affected in any way by the Envelope Generator.

SID Pin Description

CAP1A, CAP1B (Pins 1,2)/CAP2A, CAP2B Pins 3,4)

These pins are used to connect the two integrating capacitors required by the programmable Filter. C1 connects between pins 1 and 2, C2 between pins 3 and 4. Both capacitors should be the same value. Normal operation of the Filter over the audio range (approximately - 30 Hz-12 KHz) is accomplished with a value of 2200 pF for C1 and C2. Polystyrene capacitors are preferred. In complex polyphonic systems, where many SID chips must track each other, matched capacitors are recommended. The frequency range of the Filter can be tailored to specific applications by the choice of capacitor values. For example, a low-cost game may not require full high-frequency response, In this case, larger values for C1 and C2 could be chosen to provide more control over the bass frequencies of the Filter. The approximate maximum Cutoff Frequency of the Filter is given by:

$$FC_{max} = 2.6E-5/C$$

Where C is the capacitor value. The range of the Filter extends approximately 9 octaves below the maximum Cutoff Frequency.

/RES (Pin 5) -This TTL-level input is the reset control for SID. When brought low for at least ten $\emptyset 2$ cycles, all internal registers are reset to zero and the audio output is silenced. This pin is normally connected to the reset line of the microprocessor or a power-on-clear circuit.

$\emptyset 2$ (Pin 6) -This TTL-level input is the master clock for SID. All oscillator frequencies and envelope rates are referenced to this clock. $\emptyset 2$ also controls data transfers between SID and the microprocessor. Data can only be transferred when $\emptyset 2$ is high. Essentially, $\emptyset 2$ acts as a high-active chip select as far as data transfers are concerned. This pin is normally connected to the system clock, with a nominal operating frequency of 1.0 MHz.

R/W (Pin 7) -This TTL-level input controls the direction of data transfers between SID and the microprocessor. If the chip select conditions have been met, a high on this line allows the microprocessor to Read data from the selected SID register and a low allows the microprocessor to Write data into the selected SID register. This pin is normally connected to the system Read/Write line.

/CS (Pin 8) -This TTL-level input is a low active Chip select which controls data transfers between SID and the microprocessor. /CS must be low for any transfer. A Read from the selected SID register can only occur if /CS is low, $\emptyset 2$ is high and R/W is high. A Write to the selected SID register can only occur if /CS is low,

Ø2 is high and R/W is low. This pin is normally connected to address decoding circuitry, allowing SID to reside in the memory map of a system.

A0-A4 (Pins 9-13) -These TTL-level inputs are used to select one of the 29 SID registers. Although enough addresses are provided to select 1 of 32 registers, the remaining three register locations are not used. A Write to any of these three locations is ignored and a Read returns invalid data. These pins are normally connected to the corresponding address lines of the microprocessor so that SID may be addressed in the same manner as memory.

GND (Pin 14) -For best results, the ground line between SID and the power supply should be separate from ground lines to other digital circuitry. This will minimize digital noise at the audio output.

D0-D7 (Pins 15-22) -These bidirectional lines are used to transfer data between SID and the microprocessor. They are TTL compatible in the output mode and capable of driving 2 TTL loads in the output mode. The data buffers are usually in the high-impedance off state. During a Write operation, the data buffers remain in the off (input) state and the microprocessor supplies data to SID over these lines. During a Read operation, the data buffers turn on and SID supplies data to the microprocessor over these lines. The pins are normally connected to the corresponding data lines of the microprocessor.

POTX, POTY (Pins 24, 23) -These pins are inputs to the A/D converters used to digitize the position of potentiometers. The conversion process is based on the time constant of a capacitor tied from the POT pin to ground, charged by a potentiometer tied from the POT pin to +5 volts. The component values are determined by

$$RC = 4.7E-4$$

Where R is the maximum resistance of the pot and C is the capacitor.

The larger the capacitor, the smaller the POT value jitter. The recommended values for R and C are 470 KOhms and 1000 pF.

Note that a separate pot and cap are required for each POT pin.

Vcc (Pin 25) - As with the GND line, a separate +5 VDC line should be run between SID Vcc and the power supply in order to minimize noise. A bypass capacitor should be located close to the pin.

Ext In (Pin 26) -This analog input allows external audio signals to be mixed with the audio output of SID or processed through the Filter. Typical sources include voice, guitar and organ. The input impedance of this pin is in the order of 100 KOhms. Any signal applied directly to the pin should ride at DC level of 6 volts and should not exceed 3 volts p-p. In order to prevent any interference caused by DC level differences, external signals should be AC-coupled to EXT IN by an electrolytic capacitor in the 1-10uF range. As the direct audio path (FILTEX = 0) has unity gain, EXT IN can be used to mix outputs of many SID chips by daisy-chaining. The number of chips that can be chained in this manner is determined by the amount of noise

and distortion allowable at the final output. Note that the output Volume control will affect not only the three SID voices, but also any external inputs.

Audio Out (Pin 27) -This open-source buffer is the final audio output of SID, comprised of the three SID voices, the Filter and any external input. The output level is set by the output Volume control and reaches a maximum of approximately 3 volts p-p at a 6 volt DC level. A source resistor from AUDIO OUT to ground is required for proper operation. The recommended resistance is 1 KOhm for a standard output impedance. As the output of SID rides at a 6 volt DC level, it should be AC-coupled to any audio amplifier with an electrolytic capacitor in the 1-10uF range.

Vdd (Pin 28) - As with Vcc, a separate + 12 VDC line should be run to SID Vdd and a bypass capacitor should be used.

See [Appendix C](#) for typical SID application.

6581 SID Characteristics

Absolute Maximum Ratings

- Supply Voltage, Vdd -0.3 VDC to +17 VDC
- Supply Voltage, Vcc -0.3 VDC to +7 VDC
- Input Voltage (analog), Vina -0.3 VDC to +17 VDC
- Input Voltage (digital), Vind -0.3 VDC to +7 VDC
- Operating Temperature, Ta 0° C to +70° C
- Storage Temperature, Tstg -55° C to +150° C

All inputs contain protection circuitry to prevent damage due to high static discharges. Care should be exercised to prevent unnecessary application of voltages in excess of the allowable limits.

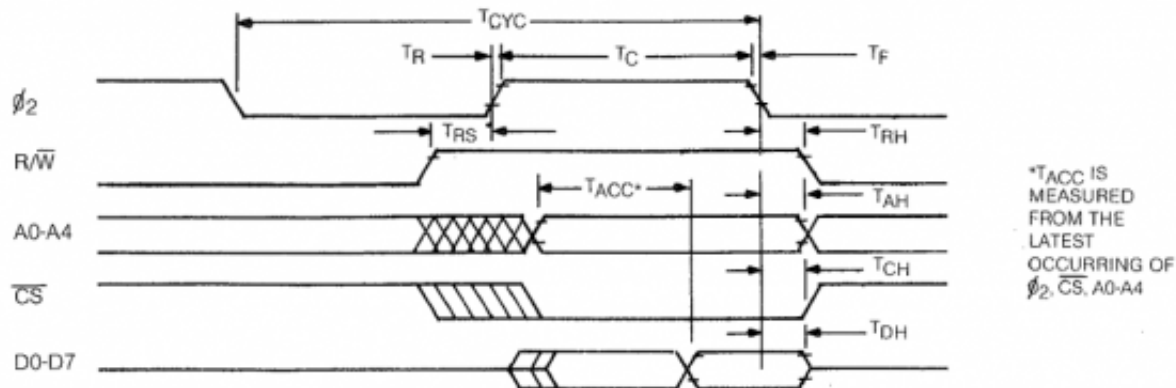
Comment

Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. These are stress ratings only. Functional operation of this device at these or any other conditions above those indicated in the operational sections of this specification is not implied and exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS (V _{dd} =12±5% VDC / V _{cc} =5±5% VDC. T _a =0 to 70°C)						
Characteristic		Symbol	Min	Typ	Max	Units
Input High Voltage	(RES, ϕ_2 , R/W, \overline{CS} , A0-A4, D0-D7)	V _{Ih}	2	—	V _{cc}	VDC
Input Low Voltage		V _{Il}	-0.3	—	0.8	VDC
Input Leakage Current	(RES, ϕ_2 , R/W, \overline{CS} , A0-A4; V _{in} =0-5 VDC)	I _{in}	—	—	2.5	μ A
Three-State (Off) Input Leakage Current	(D0-D7; V _{cc} =max, V _{in} =0.4-2.4 VDC)	I _{tsi}	—	—	10	μ A
Output High Voltage	(D0-D7; V _{cc} =min, I _{load} =200 μ A)	V _{oh}	2.4	—	V _{cc} -0.7	VDC
Output Low Voltage	(D0-D7; V _{cc} =max, I _{load} =3.2 mA)	V _{ol}	GND	—	0.4	VDC
Output High Current	(D0-D7; Sourcing, V _{oh} =2.4 VDC)	I _{oh}	200	—	—	μ A
Output Low Current	(D0-D7; Sinking, V _{ol} =0.4 VDC)	I _{ol}	3.2	—	—	mA
Input Capacitance	(RES, ϕ_2 , R/W, \overline{CS} , A0-A4, D0-D7)	C _{in}	—	—	10	pF
Pot Trigger Voltage	(POTX, POTY)	V _{pot}	—	V _{cc} /2	—	VDC
Pot Sink Current	(POTX, POTY)	I _{pot}	500	—	—	μ A
Input Impedance	(EXT IN)	R _{in}	100	150	—	KOhms
Audio Input Voltage	(EXT IN)	V _{in}	5.0 —	6.0 0.3	7.5 3	VDC VAC
Audio Output Voltage	(AUDIO OUT; 1 KOhm load, volume=max) One Voice on: All Voices on:	5.0 V _{out}	6.5 5.0 25 .75	8.0 6.5 8 3	8.0 — 1.2 .4	VDC VACp-p VACp-p
Power Supply Current	(V _{dd})	I _{dd}	—	25	40	mA
Power Supply Current	(V _{cc})	I _{cc}	—	70	100	mA
Power Dissipation	(Total)	P _d	—	600	1000	mW

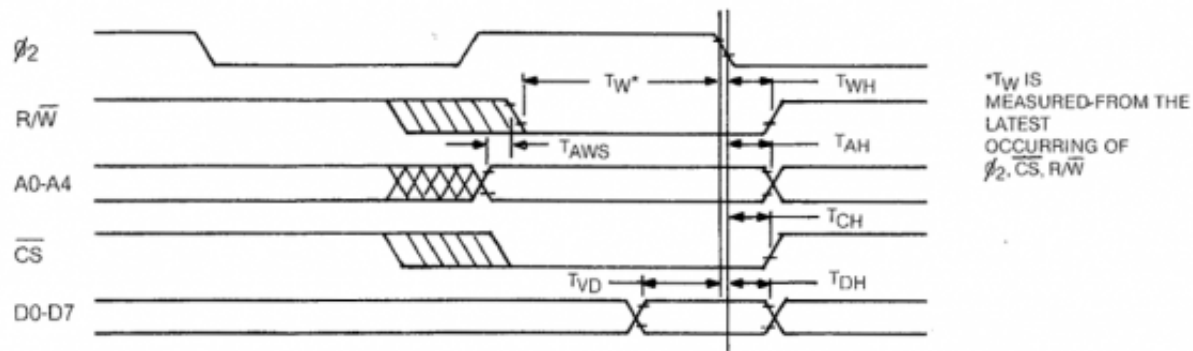
6581 (SID) Timing

READ CYCLE



Symbol	Name	Min	Typ	Max	Units
T _{CYC}	Clock Cycle Time	1	—	20	μ S
T _C	Clock High Pulse Width	450	500	10,000	nS
T _R , T _F	Clock Rise/Fall Time	—	—	25	nS
T _{RS}	Read Set-up Time	0	—	—	nS
T _{RH}	Read Hold Time	0	—	—	nS
T _{ACC}	Access Time	—	—	350	nS
T _{AH}	Address Hold Time	10	—	—	nS
T _{CH}	Chip Select Hold Time	0	—	—	nS
T _{DH}	Data Hold Time	20	—	—	nS

WRITE CYCLE



Symbol	Name	Min	Typ	Max	Units
TW	Write Pulse Width	350	—	—	nS
TWH	Write Hold Time	0	—	—	nS
$TAWS$	Address Set-up Time	0	—	—	nS
TAH	Address Hold Time	10	—	—	nS
TCH	Chip Select Hold Time	0	—	—	nS
TVd	Valid Data	80	—	—	nS
TDH	Data Hold Time	10	—	—	nS

Appendix A - Equal-Tempered Musical Scale Values

The following table lists the numerical values which must be stored in the SID Oscillator frequency control registers to produce the notes of the equal-tempered musical scale. The equal-tempered scale consists of an octave containing 12 semitones (notes): C, D, E, F, G, A, B and C#, D#, F#, G#, A#. The frequency of each semitone is exactly the 12th root of 2 times the frequency of the previous semitone. The table is based on a ϕ_2 = clock of 1.0 Mhz. Refer to the equation given in the Register Description for use of other master clock frequencies. The scale selected is concert pitch, in which A4 = 440 Hz. Transpositions of this scale and scales other than the equal-tempered scale are also possible.

	Musical	Freq	Osc Fn	Osc Fn		Musical	Freq	Osc Fn	Osc Fn
	Note	(Hz)	(Decimal)	(Hex)		Note	(Hz)	(Decimal)	(Hex)
1	C0\$	17.32	291	0123	49	C4\$	277.18	4650	122A
2	D0	18.35	308	0134	50	D4	293.66	4927	133F
3	D0\$	19.44	326	0146	51	D4\$	311.13	5220	1464
4	E0	20.60	346	015A	52	E4	329.63	5530	159A
5	F0	21.83	366	016E	53	F4	349.23	5859	16E3
6	F0\$	23.12	388	0184	54	F4\$	370.00	6207	183F
7	G0	24.50	411	018B	55	G4	392.00	6577	1981

8	G0\$	25.96	435	01B3	56	G4\$	415.30	6968	1B38
9	A0	27.50	461	01CD	57	A4	440.00	7382	1CD6
10	A0\$	29.14	489	01E9	58	A4\$	466.16	7821	1E80
11	B0	30.87	518	0206	59	B4	493.88	8286	205E
12	C1	32.70	549	0225	60	C5	523.25	8779	224B
13	C1\$	34.65	581	0245	61	C5\$	554.37	9301	2455
14	D1	36.71	616	0268	62	D5	587.33	9854	267E
15	D1\$	38.89	652	028C	63	D5\$	622.25	10440	28C8
16	E1	41.20	691	02B3	64	E5	659.25	11060	2B34
17	F1	43.65	732	02DC	65	F5	698.46	11718	2DC6
18	F1\$	46.25	776	0308	66	F5\$	740.00	12415	307F
19	G1	49.00	822	0336	67	G5	783.99	13153	3361
20	G1\$	51.91	871	0367	68	G5\$	830.61	13935	366F
21	A1	55.00	923	039B	69	A5	880.00	14764	39AC
22	A1\$	58.27	978	03D2	70	A5\$	932.33	15642	3D1A
23	B1	61.74	1036	040C	71	B5	987.77	16572	40BC
24	C2	65.41	1097	0449	72	C6	1046.50	17557	4495
25	C2\$	69.30	1163	048B	73	C6\$	1108.73	18601	48A9
26	D2	73.42	1232	04D0	74	D6	1174.66	19709	4CFC
27	D2\$	77.78	1305	0519	75	D6\$	1244.51	20897	518F
28	E2	82.41	1383	0567	76	E6	1318.51	22121	5669
29	F2	87.31	1465	05B9	77	F6	1396.91	23436	5B8C
30	F2\$	92.50	1552	0610	78	F6\$	1479.98	24830	60FE
31	G2	98.00	1644	066C	79	G6	1567.98	26306	6602
32	G2\$	103.83	1742	06CE	80	G6\$	1661.22	27871	6CDF

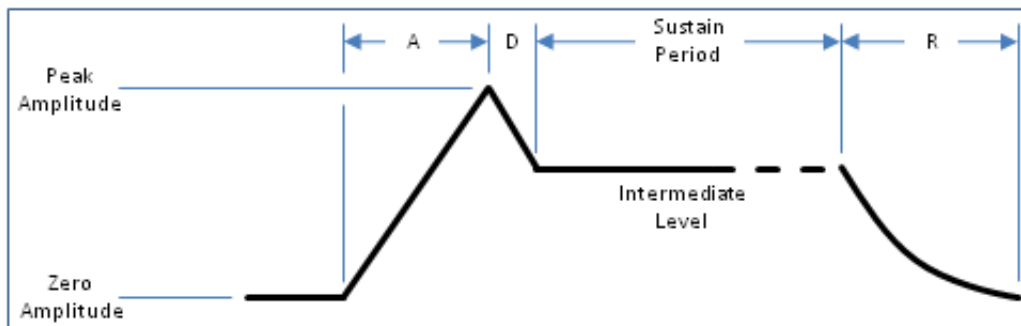
33	A2	110.00	1845	0735	81	A6	1760.00	29528	7358
34	A2\$	116.54	1955	07A3	82	A6\$	1864.65	31234	7A34
35	B2	123.47	2071	0817	83	B6	1975.53	33144	8178
36	C3	130.81	2195	0893	84	C7	2093.00	35115	892B
37	C3\$	138.59	2325	0915	85	C7\$	2217.46	37203	9153
38	D3	146.83	2463	099F	86	D7	2349.32	39415	99F7
39	D3\$	155.56	2610	0A32	87	D7\$	2489.01	41759	A31F
40	E3	164.81	2765	0ACD	88	E7	2637.02	44242	ACD2
41	F3	174.61	2930	0B72	89	F7	2793.83	46873	B719
42	F3\$	185.00	3104	0C20	90	F7\$	2959.95	49660	C1FC
43	G3	196.00	3288	0C08	91	G7	3135.96	52613	C085
44	G3\$	207.65	3484	0D9C	92	G7\$	3322.44	55741	0980
45	A3	220.00	3691	0E6B	93	A7	3520.00	59056	E6B0
46	A3\$	233.08	3910	0F46	94	A7\$	3729.31	62567	F467
47	B3	246.94	4143	102F	95	B7	3951.06	*66288	*1F2F0

Although the table above provides a simple and quick method for generating the equal-tempered scale, it is very memory inefficient as it requires 192 bytes for the table alone. Memory efficiency can be improved by determining the note value algorithmically. Using the fact that each note in an octave is exactly half the frequency of that note in the next octave, the note look-up table can be reduced from 96 entries to 12 entries, as there are 12 notes per octave. If the 12 entries (24 bytes) consist of the 16-bit values for the eighth octave (C7 through B7), then notes in lower octaves can be derived by choosing the appropriate note in the eighth octave and dividing the 16-bit value by two for each octave of difference. As division by two is nothing more than a right-shift of the value, the calculation can easily be accomplished by a simple software routine. Although note B7 is beyond the range of the Oscillators this value should still be included in the table for calculation purposes (the MSB of B7 would require a special software case, such as generating this bit in the CARRY before shifting). Each note must be specified in a form which indicates which of the 12 semitones is desired, and which of the eight octaves the semitone is in. Since four bits are necessary to select 1 of 12 semitones and three bits are necessary to select 1 of 8 octaves, the information can fit in one byte, with the lower nybble selecting the semitone (by addressing the look-up table) and the

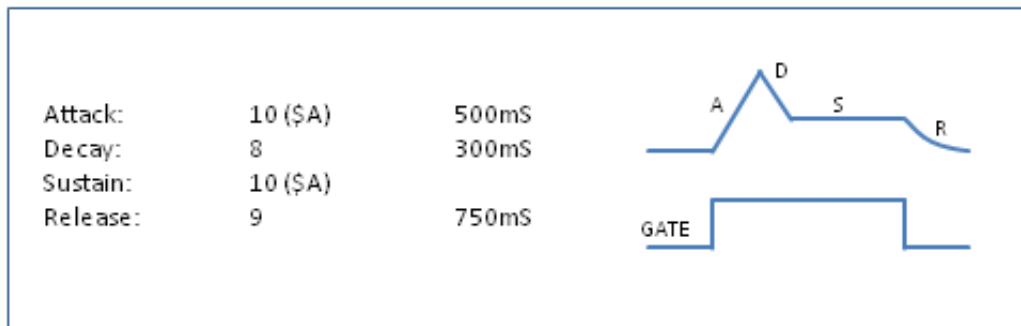
upper nybble being used by the division routine to determine how many times the table value must be right-shifted.

Appendix B - SID Envelope Generators

The four-part ADSR (ATTACK, DECAY, SUSTAIN, RELEASE) envelope generator has been proven in electronic music to provide the optimum trade-off between flexibility and ease of amplitude control. Appropriate selection of envelope parameters allows the simulation of a wide range of percussion and sustained instruments. The violin is a good example of a sustained instrument. The violinist controls the volume by bowing the instrument. Typically, the volume builds slowly, reaches a peak, then drops to an intermediate level. The violinist can maintain this level for as long as desired, then the volume is allowed to slowly die away. A "snapshot" of this envelope is shown below:

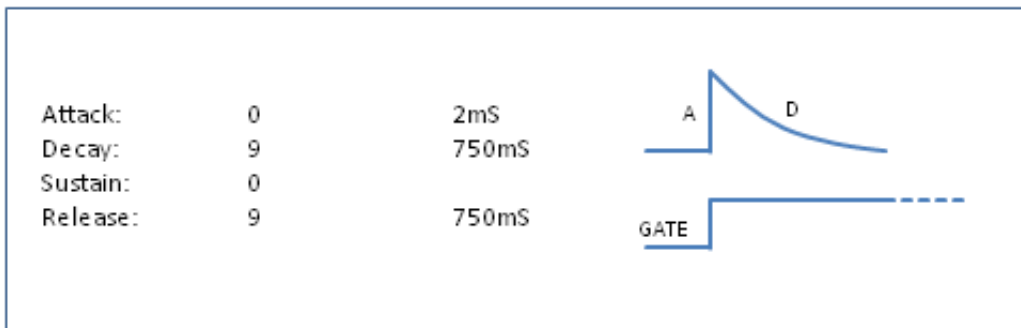


This volume envelope can be easily reproduced by the ADSR as shown below, with typical envelope rates:



Note that the tone can be held at the intermediate SUSTAIN level for as long as desired. The tone will not begin to die away until GATE is cleared. With minor alterations, this basic envelope can be used for brass and woodwinds as well as strings.

An entirely different form of envelope is produced by percussion instruments such as drums, cymbals and gongs, as well as certain keyboards such as pianos and harpsichords. The percussion envelope is characterized by a nearly instantaneous attack, immediately followed by a decay to zero volume. Percussion instruments cannot be sustained at a constant amplitude. For example, the instant a drum is struck, the sound reaches full volume and decays rapidly regardless of how it was struck. A typical cymbal envelope is shown below:



Note that the tone immediately begins to decay to zero amplitude after the peak is reached, regardless of when GATE is cleared. The amplitude envelope of pianos and harpsichords is somewhat more complicated, but can be generated quite easily with the ADSR. These instruments reach full volume when a key is first struck. The amplitude immediately begins to die away slowly as long as the key remains depressed. If the key is released before the sound has fully died away, the amplitude will immediately drop to zero. This envelope is shown below:

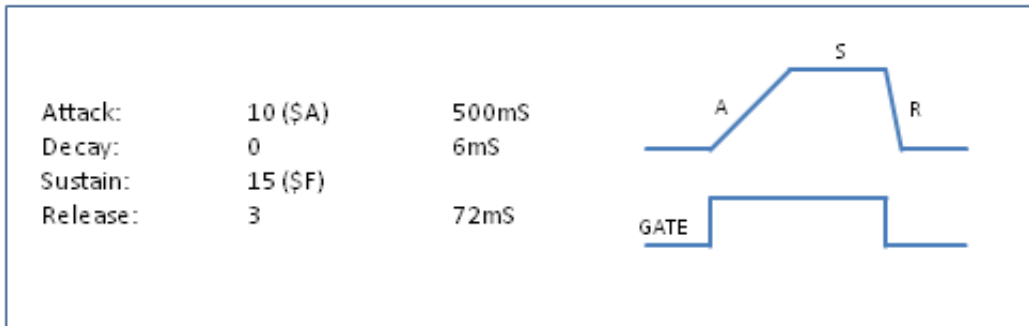


Note that the tone decays slowly until GATE is cleared, at which point the amplitude drops rapidly to zero.

The most simple envelope is that of the organ. When a key is pressed, the tone immediately reaches full volume and remains there. When the key is released, the tone drops immediately to zero volume. This envelope is shown below:

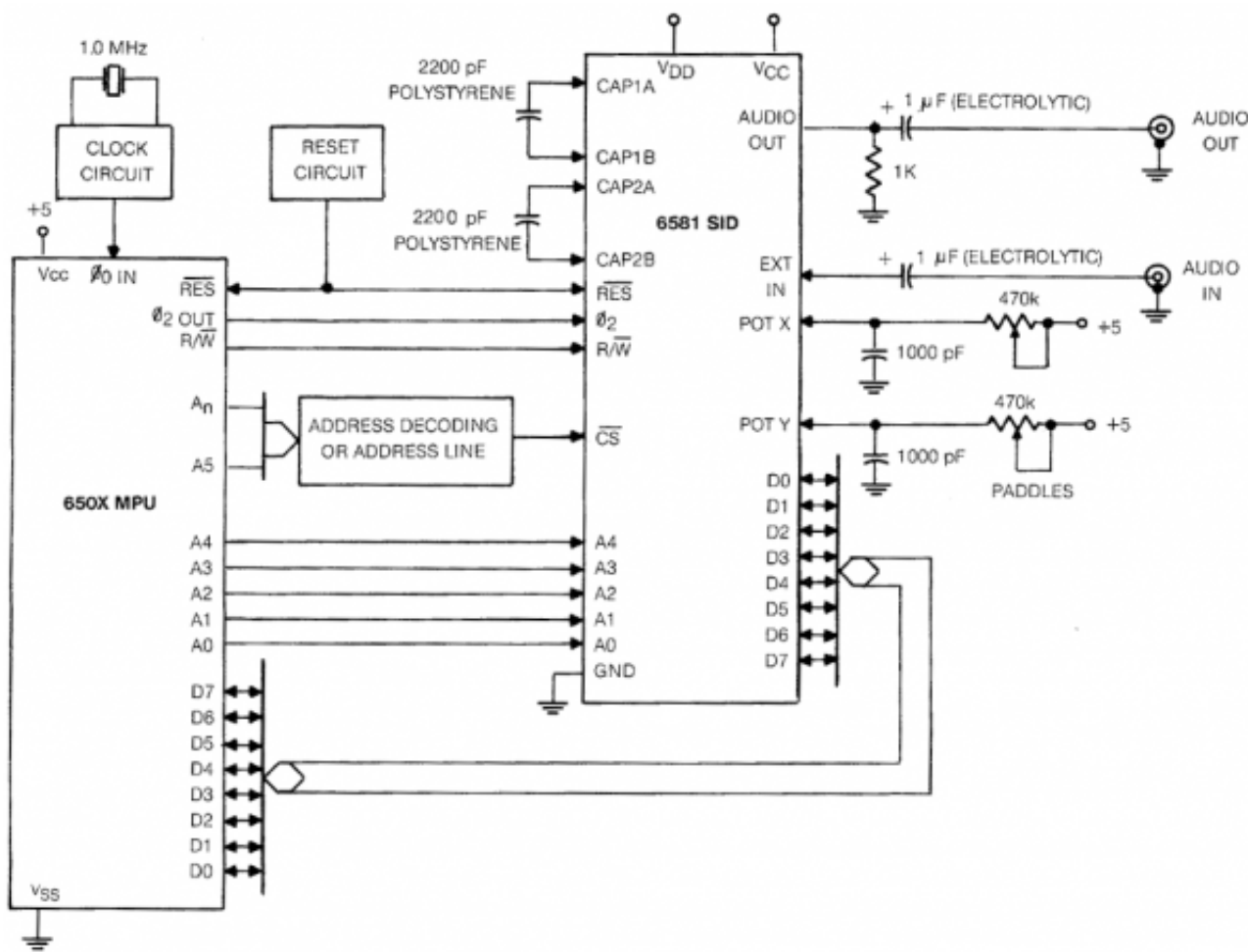


The real power of SID lies in the ability to create original sounds rather than simulations of acoustic instruments. The ADSR is capable of creating envelopes which do not correspond to any “real” instruments. A good example would be the “backwards” envelope. This envelope is characterized by a slow attack and rapid decay which sounds very much like an instrument that has been recorded on tape then played backwards. This envelope is shown below:



Many unique sounds can be created by applying the amplitude envelope of one instrument to the harmonic structure of another. This produces sounds similar to familiar acoustic instruments, yet notably different. In general, sound is quite subjective and experimentation with various envelope rates and harmonic contents will be necessary in order to achieve the desired sound.

Appendix C - Typical 6581 SID Application



Donate to waitingforfriday.com:

If you like this site and want to help support future projects, or you just want to show appreciation for a project you built, used or enjoyed, please consider leaving a PayPal donation. It's quick, secure and helps

RadioMusic Euro Module

Music
Thing
Modular



STATION



START



RESET
Hold for bank

STATION START



RESET OUT



RADIO
MUSIC



How to use the Radio Music module

Tom Whitwell edited this page on Mar 29 · 8 revisions

What is this module?

Radio Music is a virtual radio module, so it behaves a bit like a radio. It is designed to be a source of unexpected audio, not a drum loop player or a sample mangler.

Like a radio, this module works on a series of banks and stations. Each of the 16 banks can contain many different stations. Each station is .raw audio file stored in a bank directory on the SD card. Choose a bank by pressing and holding the RESET switch. Choose a station by turning the STATION knob or plugging a voltage into TUNE.

Detailed controls and displays

Adding samples to the SD Card

- Full instructions and tools are in the [Setting up the SD Card](#) section.

Bank and meter LEDs

The LEDs at the top do two jobs. When audio is playing, they act as a simple VU meter. While choosing banks, they show the bank number in binary

- 0 ○○○○
- 1 ●○○○
- 2 ○●○○
- 3 ●●○○
- 4 ○○●○
- 5 ●○●○
- 6 ○●●○
- 7 ●●●○
- 8 ○○○●
- 9 ●○○●
- 10 ○●○●
- 11 ●●○●
- 12 ○○●●
- 13 ●○●●
- 14 ○●●●
- 15 ●●●●

Station knob

This is how you choose which file to play from the current folder. It works exactly like a radio tuning knob.

- You can put up to 75 files in each folder, but with that many files it might be hard to accurately tune to a station.



Building the module

1. Parts List
2. Preparing your Teensy 3.1
3. Formatting and setting up the SD Card
4. Building the module
5. Testing and calibration
6. Troubleshooting and FAQ
7. PCB Versions
8. Schematics

Using the module

1. How to use the Radio Music module
2. Suggested Audio for the SD Card
3. Audio packs to download
4. Patch Recipes
5. How to tweak your module by editing the settings.txt file

Hacking and expanding the module

1. Reprogramming the module
2. Alternative firmware for Radio Music

Background

1. John Cage, Stockhausen and music from radio

- If you're between stations, you might get a rapid juddering sound as the module slips between stations. Just retune the knob. If you find it a big problem, put fewer files in each directory so the channels are more spaced out.
- Like real radio stations, the files continue to play in the background - they don't re-trigger each time you select a new station.

Start knob

This knob sets where the file will play from if you press the RESET button. It does nothing until you press the 'Reset' button! (This behaviour can be changed by [editing settings.txt](#))

- It's quite a coarse control, with a resolution of 512 steps. So, if you have a 30 minute radio show, the smallest shift you can make will be around 3.5 seconds. If you're playing a 4 second loop, the control will be finer.

Reset button & LED

TAP the Reset button restarts the current track at the point determined by the Start knob or the Start CV input.

- If you're playing a large file (a 30 minute .raw file is 150mb) it will take a few milliseconds to skip to the end of the file, so the button feels less responsive than it does with smaller files, or the earlier parts of bigger files.

HOLD the Reset button for more than 200ms to change banks. Keep the button held down to skip through all 16 banks.

- Bank position is saved on the module after power-down.

SD Card Slot

Full details of the SD file structure are here: [SD Card Details](#)

- It's possible to hot-swap SD cards. If the module detects the SD card has been removed, it will reboot. When the module boots up, it looks for an SD card for a while - the bank LEDs flash when no card is present. After a while it will give up, and you'll need to turn the power on and off.

Tune CV Input

This is the CV equivalent of the Station Pot. If the CV changes, the module will immediately re-tune to the relevant station.

- The module 'understands' 0v to +5v signals, and is protected against higher or negative voltages.
- The CV input is added to the knob position; if the knob is at 12 o'clock and 2.5v is applied to the CV, it's the equivalent of the knob being fully clockwise. Unfortunately you can't 'pull the pot down' by applying a negative voltage.

Start CV input

This is the CV equivalent of the Start pot. Like the start pot, it does nothing until a 'Reset' CV is received. This CV input behaves like the Tune CV - it's zero to +5v, added to the pot position.

Reset Trigger input

A positive clock here triggers the reset button. It should not trigger the bank change.

Output

This is a normal modular-level audio output.

- The output level is set by the trimmer on the back of the module
- The output is AC-coupled so cannot output control voltages





SD Card: Format & File Structure

Tom Whitwell edited this page on Oct 25 · 22 revisions

Setting up files on the Micro SD Card

Pages 18

Learn more about setting up the Micro SD card for Radio Music in this [great video from Voltage Control Lab](#)

- Use decent branded SD Cards from a reputable supplier. For a few quid, cheapo cards aren't worth the trouble. I've had good results with Sandisk Ultra and Kingston cards, both from Amazon. At the time of writing, a 32gb Sandisk Ultra is £20, a 16gb Kingston is £5.50.
- SD Card can be Fat16 or Fat32 formatted. Most SD cards come Fat32 formatted, so you don't need to do anything
- They can be up to 32gb
- 32gb in the format described below = approximately 100 hours of recording time, 200 x 30 minute files or 16 banks x 12 x 30 minute files.
- Files should be:
 - Mono
 - 16 Bit
 - 44.1 kHz
 - Headerless Wav files
 - using the .raw suffix
- The module can read up to 16 folders in the top level directory of the SD card
- Subdirectories will be ignored, folders beyond 16 may cause a crash
- Each folder becomes a bank on the module
- Banks are filled in ?? order [I still haven't worked out how this works]
- Each folder can contain up to 75 files. More files make it harder to accurately select the files with the knob or control voltage

SD Card Folder Structure

Music Thing Modular

Building the module

1. Parts List
2. Preparing your Teensy 3.1
3. Formatting and setting up the SD Card
4. Building the module
5. Testing and calibration
6. Troubleshooting and FAQ
7. PCB Versions
8. Schematics

Using the module

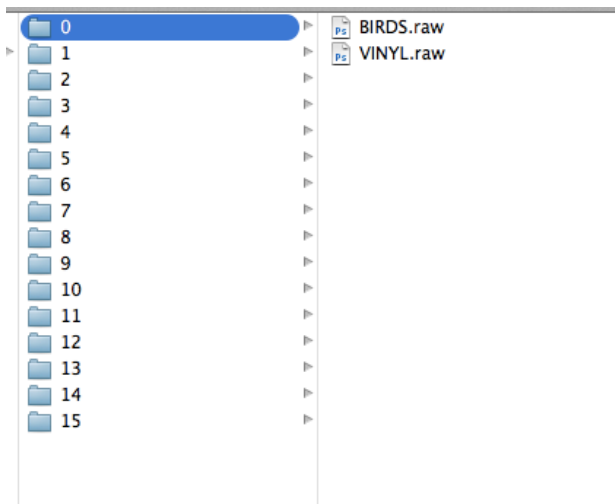
1. How to use the Radio Music module
2. Suggested Audio for the SD Card
3. Audio packs to download
4. Patch Recipes
5. How to tweak your module by editing the settings.txt file

Hacking and expanding the module

1. Reprogramming the module
2. Alternative firmware for Radio Music

Background

1. John Cage, Stockhausen and music from radio



<https://github.com/TomWhitw>

Clone in Desktop

- The card should be structured as 16 folders, named "0", "1", "2" etc (without the quote marks)
- Inside the folders, files can be named however you like, but must be in 8:3 format: NOISE.RAW, GOAT.RAW, HPSCHD.RAW
- Rather than explaining it, or messing about with the 20gb+ standard library, the easiest thing is to download this: [Empty SD Card File Format](#)
- It's not actually empty; each bank contains birdsong & vinyl sounds, and the final bank contains test tones (so it's 19mb download)
- Copy it onto a high quality SD card into the root, so when you open the card on your computer, you just see the folders.
- Don't put any other files onto the SD card; the module won't load. The module should ignore the cruft that OSX creates on SD cards; trashes, spotlight files etc.
- Once you put the SD card into the module, it will create a settings.txt file, which can be edited.
- MAXIMUM FILE LIMIT: You can add 75 files into each of the 16 folders. However, the module cannot handle more than about 330 files in total. This limit may be lifted with a future firmware upgrade.

Free software for headerless .raw wav file conversion

- Audacity - (Windows, Mac, Linux) - <http://audacity.sourceforge.net/>
- Sbooth Max - Quick batch conversion (OSX) - <http://sbooth.org/Max/>

A command-line tool made for converting files to Radio Music format

- [FormatRadio](#) is a well documented tool by Apolakipso that downloads free samples and converts them into Radio Music formatted folders.

Creating headerless .raw wav files in Audacity

- Open your source file - i.e. a stereo MP3
- Ensure Project Rate is set to 44100 (Bottom left hand corner of the screen)
- Mix to Mono (Tracks / Stereo Track to Mono)
- You may want to apply some audio compression to even out levels (Effect / Compressor). Radio recordings are often aggressively compressed, as you'll see if you import FM radio recordings.
- You can remove any silence from the sample using Effect / Truncate Silence

- Normalise (Effect / Normalise - I normally choose -0.1dB)
- Once the file is ready, choose:
 - File / Export
 - Format: "Other Uncompressed Files"
 - Options
 - Header: "RAW (header-less)"
 - Encoding: "Signed 16-bit PCM"
 - The resulting file will have a .raw suffix. You can't double click to open it again in Audacity, you have to use File / Import / Import Raw Data

Copying files in OSX Terminal

- You may find it easier to copy huge files using Terminal, rather than dragging and dropping in Finder.
- rsync is a useful command, if you have a local set of folders that you want to copy onto an SD Card and keep it sync.

```
rsync -va --delete ~/Folder1/ ~/Folder2/
```

- Read [How to use rsync](#) to make sure you understand it before overwriting files!

Backlog

- Load normal Wav files
- Automatically handle different bit depths / sample rates
- Load compressed mp3/flac files
 - This from Paul Stoffregen "Just wanted to let you know Frank B is working on porting the Real Networks Helix MP3 code. Already a couple people have managed to get their Teensy to play MP3 files. The code is still pretty rough, but at least it is possible. However, seeking in MP3 is tricky. The data is stored in 26 millisecond overlapping blocks, which isn't nearly as nice to work with as uncompressed audio. It also tends to eat up chunks of CPU time every 26 ms, which might add some noticeable latency."





Customise your module: Editing settings.txt

Tom Whitwell edited this page on Mar 5, 2015 · 7 revisions

How to personalise your Radio Music module

Pages 18

- When the module starts up, it checks the SD card for a file called settings.txt
- If no file is found, it will create one on the card, with the default settings
- By editing settings.txt you can change how the module responds and behaves in quite significant ways
- The file is a simple text file you can open in any text editor. Change the numbers to change the behaviour of the module.
- If you have any trouble, just delete the settings.txt file. The module will create a new one according to the defaults, which are:

```
MUTE=0
DECLICK=15
ShowMeter=1
meterHIDE=2000
ChanPotImmediate=1
ChanCVImmediate=1
StartPotImmediate=0
StartCVImmediate=0
StartCVDivider=2
Looping=1
```

Mute

- If MUTE=1 (1 = true, 0 = false) then the module will fade the sound out while making changes - changing channel or resetting. This reduces clicks, making the audio smoother, but slows down the responsiveness of the module considerably. If you trigger a change, it will start the fade (lasting as long as DECLICK below), start playing again when the volume is zero, then fade back in.

DECLICK

- If MUTE is true, then the audio fades for the number of milliseconds in DECLICK. If you set this to 200, the fades are very audible.
- If DECLICK is high and you repeatedly trigger changes, the audio may appear to be constantly faded out - it will never get loud enough to hear

ShowMeter

- If ShowMeter is true (1), the 4 LEDs at the top of the module act as a VU meter when audio is playing
- If ShowMeter is false (0), the 4 LEDs display the current bank number in binary



Building the module

1. Parts List
2. Preparing your Teensy 3.1
3. Formatting and setting up the SD Card
4. Building the module
5. Testing and calibration
6. Troubleshooting and FAQ
7. PCB Versions
8. Schematics

Using the module

1. How to use the Radio Music module
2. Suggested Audio for the SD Card
3. Audio packs to download
4. Patch Recipes
5. How to tweak your module by editing the settings.txt file

Hacking and expanding the module

1. Reprogramming the module
2. Alternative firmware for Radio Music

Background

1. John Cage, Stockhausen and music from radio

- If ShowMeter is true, meterHIDE sets how long the number is shown after changing banks, in milliseconds

ChanPotImmediate and variations

- These settings determine how the module's knobs and CV inputs behave
- If the setting is true (1), then any change in the knob position or CV input will immediately change the channel or the playback position
- If the setting is false (0), then knob or CV changes only make a difference when RESET is pressed or a trigger hits the reset input
- This is quite confusing, so I'll try to explain
 - The default setting is: Station selection is immediate, start selection is on reset.
 - The Station knob operates exactly like a radio tuning knob.
 - But the Start knob does nothing until RESET is pressed.
 - That's because the Start knob is more sensitive than the Station knob. Station selects from one of maybe 20 or 30 positions, while Start might be one of 512 positions. It's easier to nudge Start, causing a glitch in the audio, so I left it 'locked' until RESET is pressed.
 - However, there are a lot of fun musical opportunities when Start is set to Immediate, so experiment. These audio samples [Random Drums](#) and [Random Voices](#) are easier to achieve when StartCVImmediate=0, as explained in the [Random Beatbox](#) patch recipe.

StartCVDivider

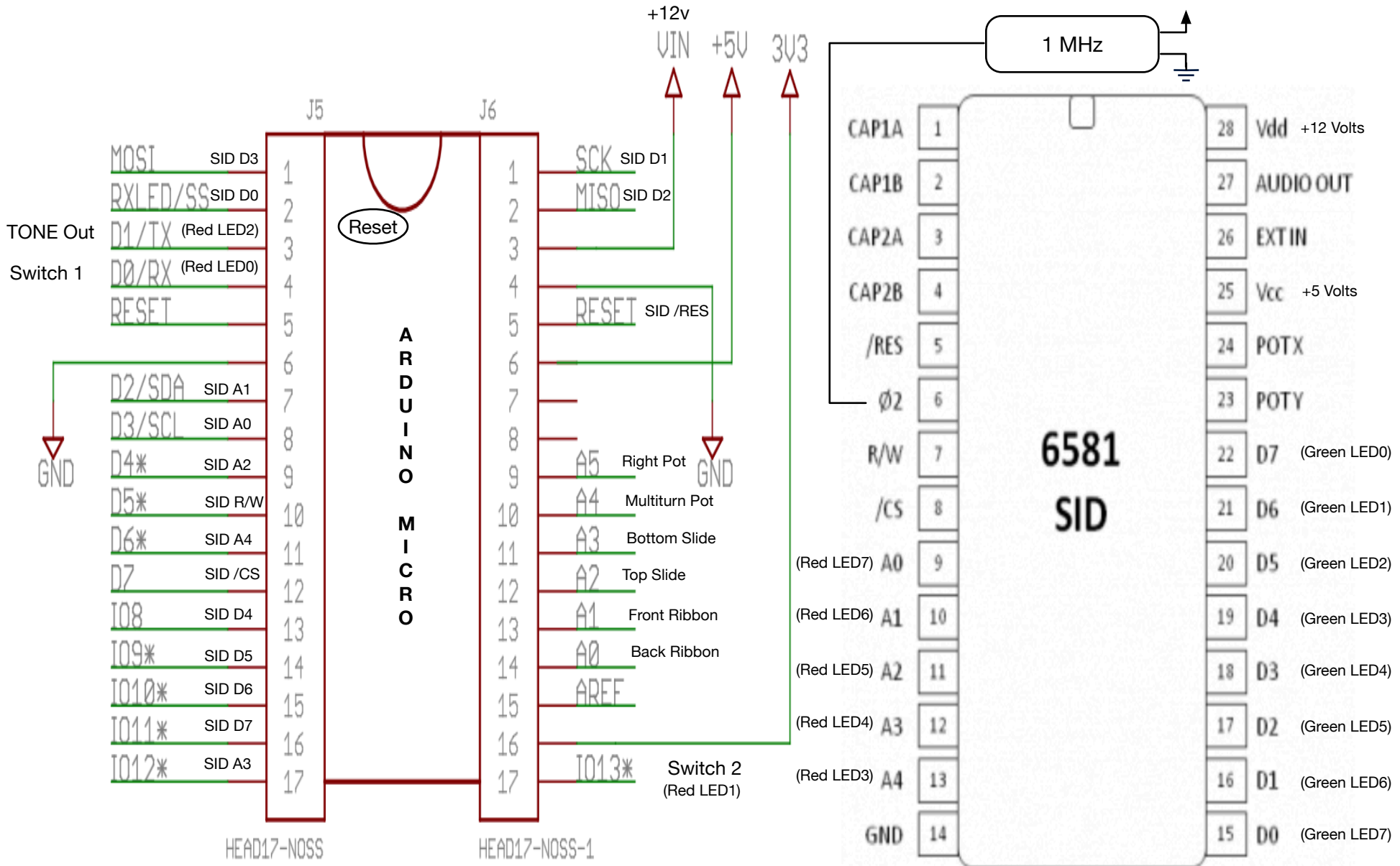
- This number sets the quantisation of the Start CV (and Pot) inputs, to make them less sensitive to noise and movement. Higher numbers = coarser resolution.
 - If the setting is 1, there are 1023 possible Start CV positions. In a 10 minute sample, you can select to a resolution of half a second.
 - If the setting is 8, there are 128 possible start positions, a resolution of 5 seconds.

Looping

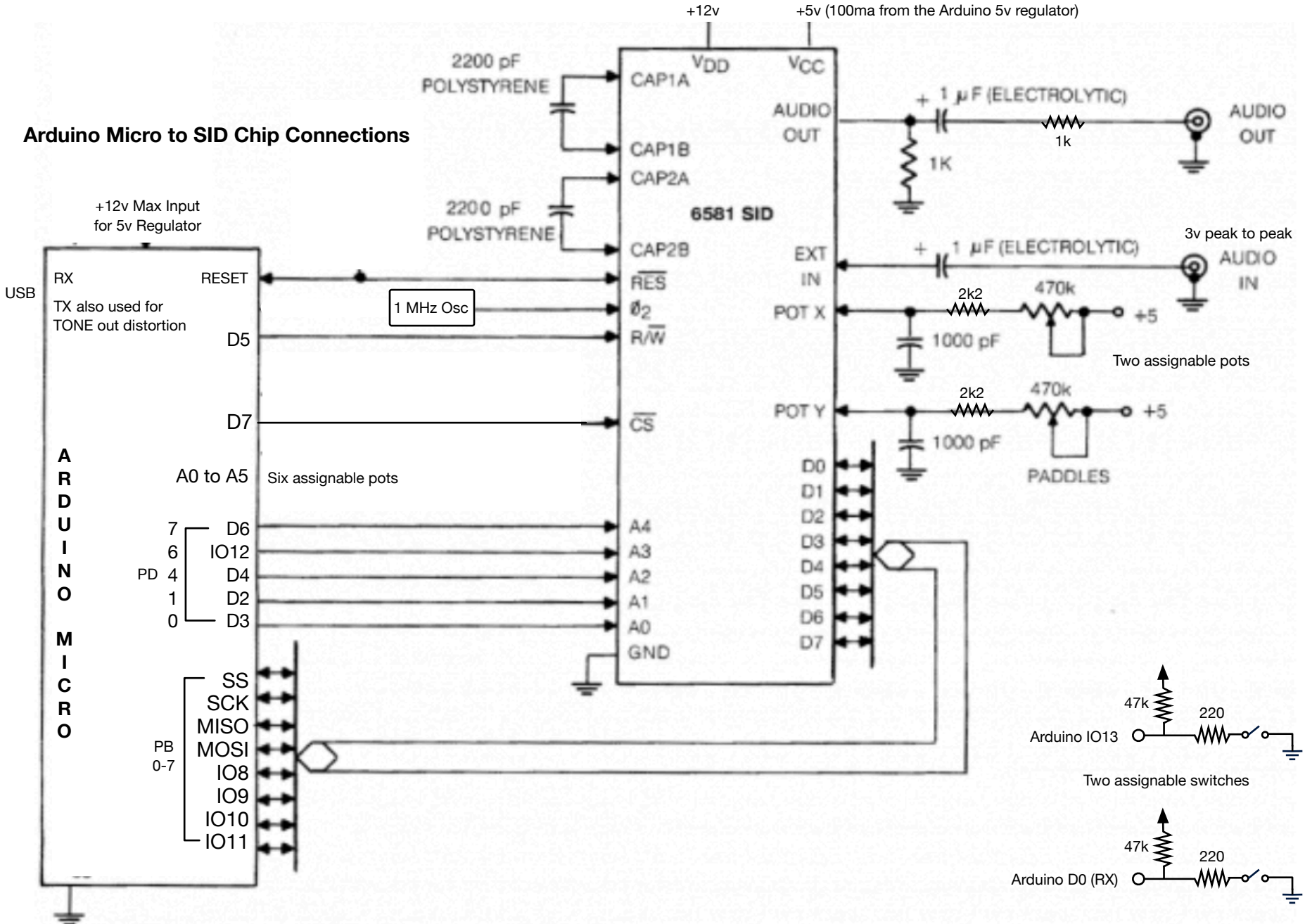
- If Looping is false (0) then files will only play once after they've been triggered.
- v1.2 and above: Looping=0 also disables the radio system. In normal use, the files appear to 'play in the background' as you skip between channels. With looping=0, this is disabled. Every time you move to a channel, it starts from the position set by the start control. This is more useful for drum-style sample triggering.
- This may have an impact on the hot-swapping system, please report any issues.

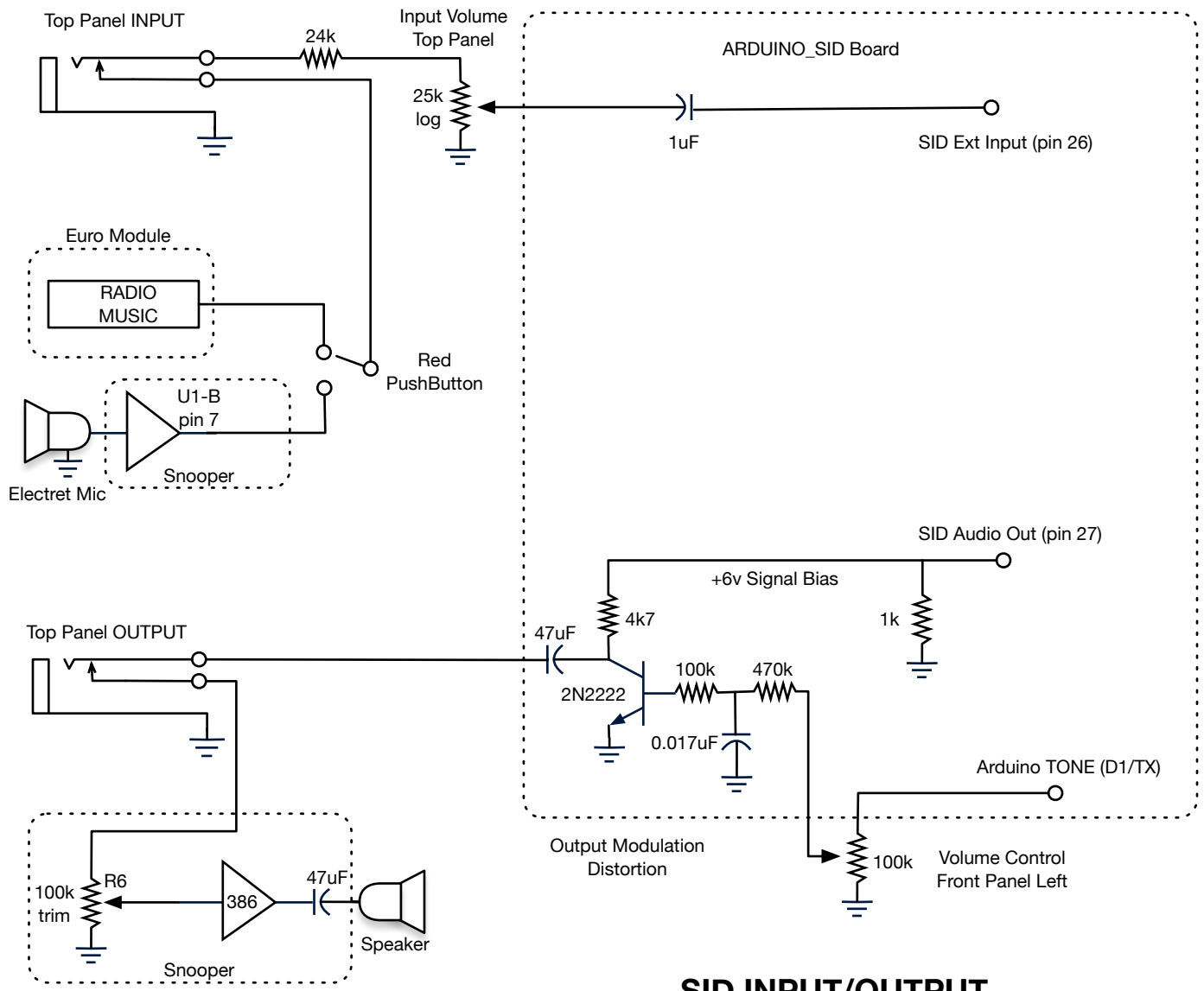
Majestic Radio

Circuit Diagrams



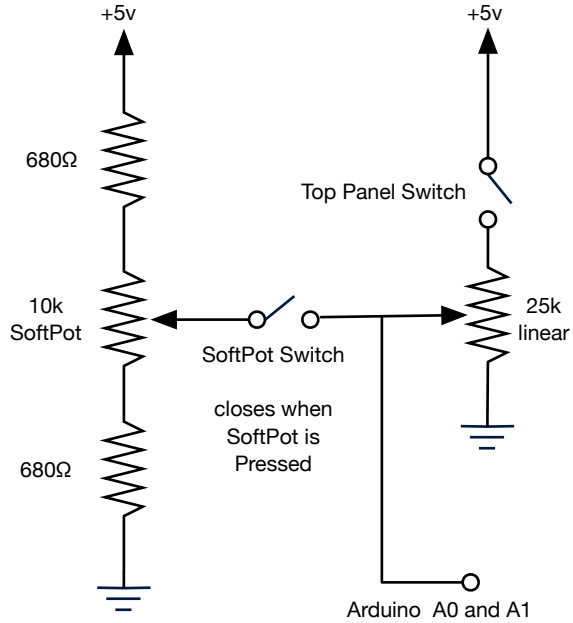
Arduino Micro to SID Chip Connections





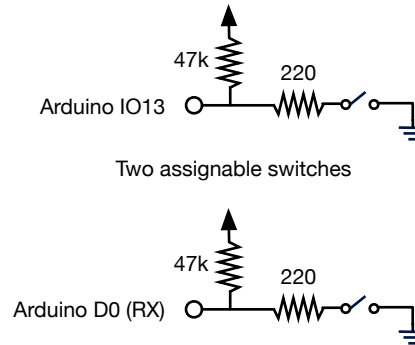
SID INPUT/OUTPUT

Circuit for SoftPot/Ribbon Controller



SoftPot Pressed	25k pot sets range of Soft Pot	25k pot sets offset of SoftPot
SoftPot Open Switch	25k pot Grounds the Output to Zero	25k pot alone sets output value
	Panel Switch Open	Panel Switch Closed

Circuit for Switches

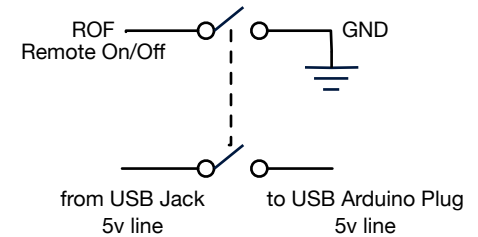


Two assignable switches

Cisco Power Module

- BLK GND (RTN)
- BLK GND (RTN)
- WHT ROF
- GRN -12v
- ORG +12v
- RED +5v

Power Switch

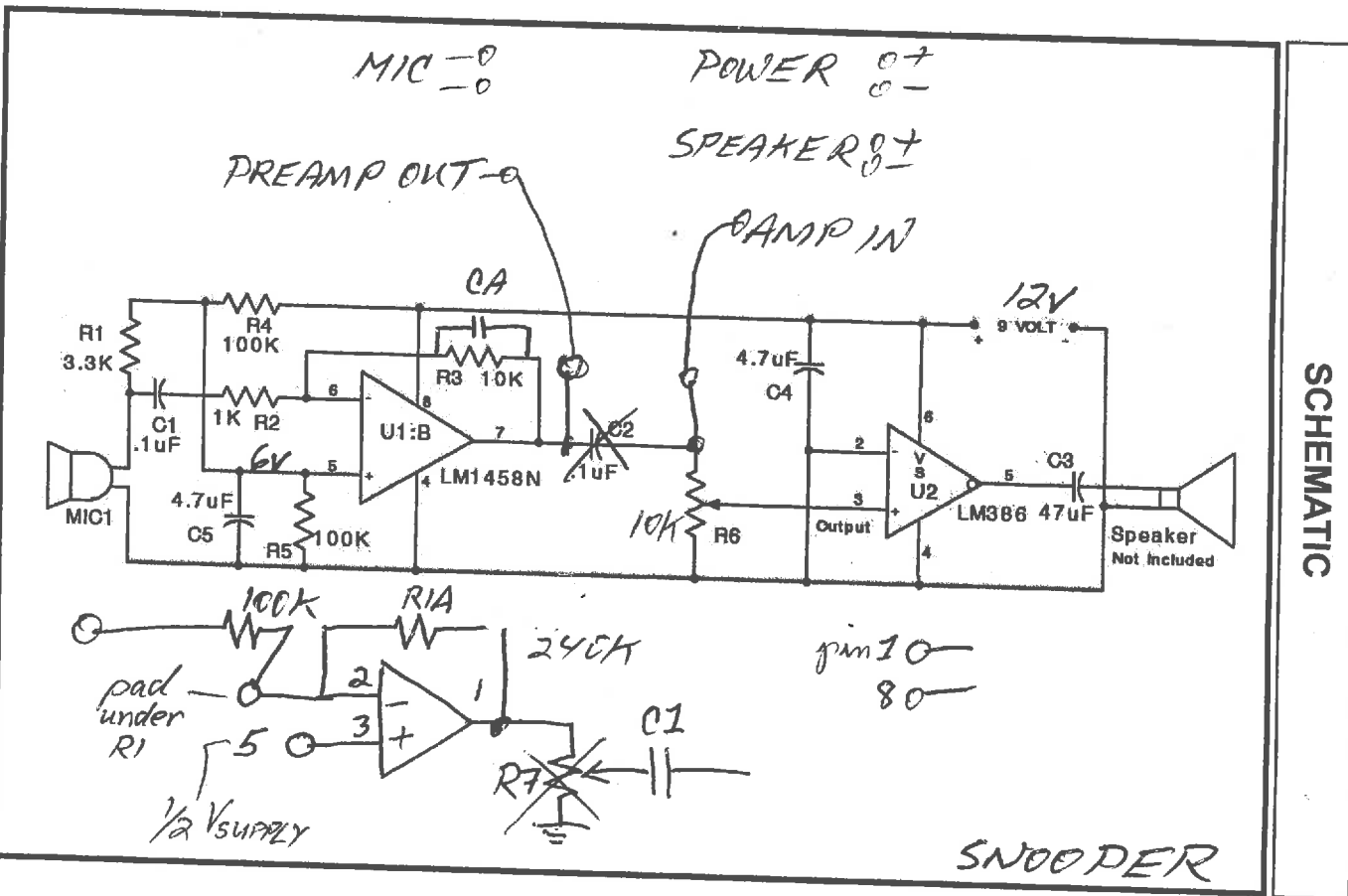


To avoid LatchUp in the SID CMOS chip, +5v must not be present at any SID pins when the +12v power is not present.

So +5v from the USB cable is prevented from reaching the circuit when the Power Switch is not on.

SID
CONTROLLER
CIRCUITS

PCM-2738D



SCHEMATIC

Majestic Radio

Arduino Program
Template


```

/*
//~~~~~Commodore SID Chip Controlled by an Arduino Micro~~~~~
//~~~~~

3 Voice Synthesizer controlled through 29 8-bit registers and 3 Control lines

~~~~~Arduino Pin Assignments:~~~~~

8 bit Data on IO11, IO10, IO9, IO8, MOSI, MISO, SCK, SS (PB0-7)
5 Address Lines on D3, D2, D4, IO12, D6
Chip Select (active low) on D7
R/W (write low) on D5
Clock from a 1MHz Oscillator chip
Arduino and SID Reset lines tied together

Two Switches on D0 (RX pin), and IO13
Six Continuous controllers on A0 through A5
Two more Continuous Controller read from the SID chip PotX and PotY
TONE function used on D1 (TX pin) to modulate/distort the SID output

~~~~~ Programming Variables/Constants~~~~~

// 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 Sustain[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 to High Pass
const int BP = 32; // set Filter to BandPass
const int LP = 16; // set Filter to LowPass

~~~~~Programming Functions~~~~~

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

ldFreqLo(int voice){ //8-bit fine tune frequency -- FreqLo
ldFreqHi(int voice){ //8-bit course tune frequency -- FreqHi
ldPulseWLo(int voice){ //8-bit fine tune Pulse Width -- PulseWLo
ldPulseWHi(int voice){ //4-bit course tune Pulse Width -- PulseWHi
ldGate(int voice, bool gate){ // Gates the ADSR Envelope (also loads Waveshape)
ldEnvAD(int voice){ //4-bit Attack Time, 4-bit Decay time -- Env Attack/Decay
ldEnvSR(int voice){ //4-bit Sustain Level, 4-bit Release time -- Env Sustain/Release

ldFCLo(int data){ //3-bit fine tune Filter Cutoff Frequency
ldFCHi(int data){ //8-bit course tune Filter Cutoff Frequency
ldResFilt(int filt, int res){ //FILTEX/FILT3/FILT2/FILT1, 4-bit Filter Resonance
ldModeVol(int mode, int vol){ //Filter Type 30FF/HP/BP/LP, 4-bit Output Volume

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

```

```

const int RW = 5; // set up names for some Arduino pins
const int CS = 7;
const int TONEpin = 1;
const int ADDR0 = 3;
const int ADDR1 = 2;
const int ADDR2 = 4;
const int ADDR3 = 12;
const int ADDR4 = 6;

// Controllers

int ribbon1 = 0; //Use ribbon1 = analogRead(A0);
int ribbon2 = 0; //Use ribbon2 = analogRead(A1);
int slider1 = 0; //Use slider1 = analogRead(A2);
int slider2 = 0; //Use slider2 = analogRead(A3);
int middlePot = 0; //Use middlePot = analogRead(A4);
int rightPot = 0; //Use rightPot = analogRead(A5);
int switch1 = 0; //Use switch1 = digitalRead(0);
int switch2 = 0; //Use switch2 = digitalRead(13);

//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 Sustain[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 to High Pass
const int BP = 32; // set Filter to BandPass
const int LP = 16; // set Filter to LowPass

unsigned long timestamp;
unsigned long duration;

//~~~~~
//                               SETUP
//~~~~~

void setup() {

delay(5000); // Allow time for entering new code when programming since TX/RX are being used

pinMode(RW, OUTPUT);
digitalWrite(RW, LOW); // High for Read, Low for Write
pinMode(CS, OUTPUT);
digitalWrite(CS, HIGH); // Chip Select active low, to perform Read or Write to register
pinMode(TONEpin, OUTPUT); // Arduino output for TONE (also used as USB TX during programming)
digitalWrite(TONEpin, HIGH);
DDRB = B11111111; // Setup Data Lines as Outputs

pinMode(ADDR0, OUTPUT); // Address Select lines for SID Registers
pinMode(ADDR1, OUTPUT);
pinMode(ADDR2, OUTPUT);

```

```

pinMode(ADDR3, OUTPUT);
pinMode(ADDR4, OUTPUT);

resetSID();

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

} //End of Setup

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

void loop() {

} // End of Main Loop

// _____
//                               Basic Address and Data Functions
// _____

void loadAddress(int address){
  if ((address & 1) > 0){digitalWrite(ADDR0, HIGH);}
  else {digitalWrite(ADDR0, LOW);};
  if ((address & 2) > 0){digitalWrite(ADDR1, HIGH);}
  else {digitalWrite(ADDR1, LOW);};
  if ((address & 4) > 0){digitalWrite(ADDR2, HIGH);}
  else {digitalWrite(ADDR2, LOW);};
  if ((address & 8) > 0){digitalWrite(ADDR3, HIGH);}
  else {digitalWrite(ADDR3, LOW);};
  if ((address & 16) > 0){digitalWrite(ADDR4, HIGH);}
  else {digitalWrite(ADDR4, LOW);};
}

void loadData(int data){
  PORTB = data;
  digitalWrite(CS, LOW);
  delayMicroseconds(3);
  digitalWrite(CS, 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
  DDRB = B00000000; //Setup Data Lines as Inputs
  digitalWrite(RW, HIGH); // Setup Read/Write for a Data Read

  loadAddress(25);
  digitalWrite(CS, LOW);
  delayMicroseconds(3);
  potX = PINB;
  delayMicroseconds(3);
  digitalWrite(CS, HIGH);

  loadAddress(28);
  digitalWrite(CS, LOW);
  delayMicroseconds(3);
  env3 = PINB;
  delayMicroseconds(3);
  digitalWrite(CS, HIGH);

  loadAddress(27);
  digitalWrite(CS, LOW);
  delayMicroseconds(3);
  osc3_rand = PINB;
  delayMicroseconds(3);
  digitalWrite(CS, HIGH);

  loadAddress(26);
  digitalWrite(CS, LOW);
  delayMicroseconds(3);
  potY = PINB;
  delayMicroseconds(3);
  digitalWrite(CS, HIGH);

  digitalWrite(RW, LOW); // Reset Read/Write to Data Write

```

```

    DDRB = B11111111;    // Reset Data Lines as Outputs
}

void loadSensors(){    // load all current sensor values
    ribbon1 = analogRead(A0) >> 2;
    switch1 = digitalRead(0);
    ribbon2 = analogRead(A1) >> 2;
    readRegisters();
    slider1 = analogRead(A2) >> 2;
    slider2 = analogRead(A3) >> 2;
    switch2 = digitalRead(13);
    middlePot = analogRead(A4) >> 2;
    rightPot = analogRead(A5) >> 2;
}
// -----
//                               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, bool 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 = ((Sustain[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 30FF/HP/BP/LP, 4-bit Output Volume
    loadAddress(24);
    loadData((vol & 15) + (mode & B11110000));
}

//~~~~~

```

```
//          Time Functions
// _____
void timeStamp() {timestamp = millis(); };
// store current time from the running clock millis()

unsigned long dur(){
  return (millis() - timestamp);
}
// 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
```

Majestic Radio

Arduino Program
Sample

Majestic SID Radio

Arduino Example Programs

Instructions:

Combine these "Setup Synth", "Main Loop", and "Loop Function" sections with the "SID Arduino Template" program, replacing those same sections in the Template with the ones here. Look under the Tool Menu of the Arduino IDE programming app, and set the "Board" type to "Arduino/Genuino Micro".

Example 1:

```
//Triangle waveform on voice 2 distorted by TONEpin.
//Triangle frequency on back ribbon.
//LoPass filter frequency on top Slider, half Resonance.
//Modulation frequency on MultiTurn Pot.

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

ldFiltRes(15, 7); // Send all through Filter, Resonance at half
ldModeVol(LP, 15); // LowPass Filter, Output Volume at max

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

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

Waveshape[2] = TRIANGLE;
ldGate(2, 1); //Turn on voice 2 and leave on

} //End of Setup

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

void loop() {

  int v = 2;

  loadSensors();
  ldFCHi(slider1);
  tone(TONEpin, middlePot); //Distortion frequency

  FreqHi[v] = ribbon1;
  ldFreqHi(v);

} // End of Main Loop
//~~~~~
```

Example 2:

```
// Three voices with waveforms set by the 2 switches and frequency set by the 2 ribbons and lower slide.
// ADSR envelopes set once in Setup. Durations of envelopes set by random values from the Right Pot.
// Distortion/4th voice on the Volume and Multiturn pot. LP Filter frequency on upper Slide pot.
// All 9 controllers are in play at all times.
```

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

ldFiltRes(15, 7); // Send all through Filter, Resonance at half
ldModeVol(LP, 15); // LowPass Filter, Output Volume at max

//envelopes will be mainly Attack/Decay/Sustain and then
//retriggered almost immediately after Release
```

```

for (int i=1; i<4; i++){

    Attack[i] = 10;
    Decay[i] = 10;
    ldEnvAD(i);

    Sustain[i] = 1;
    Release[i] = 1;
    ldEnvSR(i);

} //End of ADSR envelope load loop

} //End of Setup

//-----
//          MAIN LOOP
//-----
boolean state1=0; //State of voice Gate
boolean state2=0;
boolean state3=0;

int dur1 = 0; //Voice duration counts
int dur2 = 0;
int dur3 = 0;

void loop() {

loadSensors(); //read all 8 pot and switch values
//readRegisters(); //make 4 SID read registers available

ldFCHi(slider1); //Filter cutoff frequency - TopSlider
tone(TONEpin, middlePot); //Distortion frequency - MultiTurn Pot

//Two Ribbons and Bottom Slider have 10 bit values to set Voice Frequencies
//Shift right 2 for high byte, shift left 6 for low byte

FreqHi[1] = ribbon1>>2;
ldFreqHi(1);
FreqHi[2] = ribbon2>>2;
ldFreqHi(2);
FreqHi[3] = slider2>>2;
ldFreqHi(3);

FreqLo[1] = ribbon1<<6;
ldFreqLo(1);
FreqLo[2] = ribbon2<<6;
ldFreqLo(2);
FreqLo[3] = slider2<<6;
ldFreqLo(3);

// Set voice waveshapes with 2 Switches, loaded inside ldGate() function

if (switch1 && switch2)
{Waveshape[1]=SAWTOOTH; Waveshape[2]=TRIANGLE; Waveshape[3]=NOISE;}
if (switch1 && !switch2)
{Waveshape[1]=SAWTOOTH; Waveshape[2]=TRIANGLE; Waveshape[3]=SAWTOOTH;}
if (!switch1 && switch2)
{Waveshape[1]=SYNC; Waveshape[2]=TRIANGLE; Waveshape[3]=TRIANGLE;}
if (!switch1 && !switch2)
{Waveshape[1]=RINGMOD; Waveshape[2]=TRIANGLE; Waveshape[3]=TRIANGLE;}

// Trigger voice ADSR envelopes on & off. Decrement envelope durations.
// Durations set with random values between 1 and Right Pot value.

if (dur1 == 0){
    dur1 = random(1, rightPot);
    if (state1) { ldGate(1, 0); state1=0; }
    else { ldGate(1, 1); state1=1; }
} else { dur1 -- 1; }

if (dur2 == 0){
    dur2 = random(1, rightPot);
    if (state1) { ldGate(2, 0); state1=0; }
    else { ldGate(2, 1); state1=1; }
} else { dur2 -- 1; }

if (dur3 == 0){

```



```

dur3 = random(1, rightPot);
  if (state1) { ldGate(3, 0); state1=0; }
  else      { ldGate(3, 1); state1=1; }
} else { dur3 -= 1; }

delay(10); // Sets envelope duration base count value

} //End of Main Loop

```

Example 3

// This example loops through a short tune. Change the tune frequency with ribbon1 (also detunes it).
// Randomize the frequencies with Right Pot. Change the tempo with ribbon2 (only at the tune starts).

```

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

ldFiltRes(15, 7); // Send all through Filter, Resonance at half
ldModeVol(LP, 15); // LowPass Filter, Output Volume at max

for (int i=1; i<4; i++){

  Attack[i] = 1;
  Decay[i] = 2;
  ldEnvAD(i);

  Sustain[i] = 5;
  Release[i] = 2;
  ldEnvSR(i);

} //End of ADSR envelope load loop

loadSensors();

} //End of Setup

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

int basebeat=10;

void loop() {

basebeat = ribbon2 + 10;

ldFCHi(slider1); //Filter cutoff frequency - TopSlider
tone(TONEpin, middlePot); //Distortion frequency - MultiTurn Pot

// Set voice waveshapes with 2 Switches, loaded inside ldGate() function

if (switch1 && switch2)
{Waveshape[1]=NOISE; Waveshape[2]=NOISE; Waveshape[3]=NOISE;}
if (switch1 && !switch2)
{Waveshape[1]=SAWTOOTH; Waveshape[2]=SAWTOOTH; Waveshape[3]=SAWTOOTH;}
if (!switch1 && switch2)
{Waveshape[1]=TRIANGLE; Waveshape[2]=TRIANGLE; Waveshape[3]=TRIANGLE;}
if (!switch1 && !switch2)
{Waveshape[1]=TRIANGLE; Waveshape[2]=TRIANGLE; Waveshape[3]=TRIANGLE;}

//Play notes with vOn(beat #, voice, frequency), vOff(beat #, voice)

timeStamp();
vOn(1, 1, 4387);
vOn(2, 2, 5530);
vOn(3, 3, 6577); vOff(4, 3);
vOn(4, 3, 8779); vOff(5, 3);
vOn(5, 3, 11060); vOff(6, 3);
vOn(6, 3, 6577); vOff(7, 3);
vOn(7, 3, 8779); vOff(8, 3);
vOn(8, 3, 11060);

vOff(8, 1); vOff(8, 2); vOff(9, 3);

vOn(9, 1, 4387);
vOn(10, 2, 4927);
vOn(11, 3, 7382); vOff(12, 3);
vOn(12, 3, 9854); vOff(13, 3);

```

```

vOn(13, 3, 11718); vOff(14, 3);
vOn(14, 3, 7382); vOff(15, 3);
vOn(15, 3, 9854); vOff(16, 3);
vOn(16, 3, 11718);

vOff(17, 2); vOff(20, 3); vOff(20, 1);

} //End of Main Loop

//~~~~~
//          Loop Functions
//~~~~~

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

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

void waitTill(int beat) {
    beat = beat * basebeat;
    while (dur() < beat) {};
}
//wait till the time duration from timestamp equals the given beat time

void vOn(int beat, int voice, int pitch){
    loadSensors();
    pitch = pitch + (ribbon1 << 6) + random(rightPot << 4);
    FreqHi[voice] = pitch>>8;
    IdFreqHi(voice);
    FreqLo[voice] = pitch & 255;
    IdFreqLo(voice);

    waitTill(beat);
    IdGate(voice, 1);
}

void vOff(int beat, int voice){
    waitTill(beat);
    IdGate(voice, 0);
}
//~~~~~

```