

SNES Chiptune Guide has moved!!

The guide has been completely overhauled & updated as a static website, located at <https://samplemance.rs/snesguide>. This Google Doc (version 1) is no longer maintained, nor does it contain any of the revisions put into version 2. **Thank you everyone** who's used my guide these last few years! I hope you like the new site! **-HVB** (9/20/2025)

I. Introduction	3
How to Listen to SNES Chiptune Files	4
II. C700 vs SNESMOD	5
III. How to use C700	8
A. MIDI Setup	9
Advanced Setup: MIDI CCs (Control Changes)	11
Super Advanced Setup: MIDI NRPNs & RPNS	12
NRPN: DSP Register Write	13
RPN: Pitch Bend Sensitivity	15
B. Global Behavior Settings	16
Poly & Engine	17
VoiceAlloc	17
Bend Range	18
Velocity Curve	18
Vib. Depth & Vib. Rate	19
Multi Bank A/B/C/D	19
C. Volume & Echo Delay Settings	20
Volume Sliders	20
Echo Delay	21
Delay Time	21
Feedback	22
Filter	23
D. The C700 Sampler	24
Loading and Browsing Samples	24
Sampler Features Pt. 1	26
Bank, High Key, Low Key	26
Root Key	26
Loop Point	26
Sample Rate	27
Priority: NoteOn, Release	27
Sampler Features Pt. 2	28
Sample Volume	28

Effect Toggle: Echo	28
Effect Toggle: PMOD (Pitch Modulation)	29
Effect Toggle: Noise	30
Effect Toggle: Mono	30
Effect Toggle: Glide	31
Envelope	32
AR (Attack)	32
DR (Decay), SL (Sustain)	32
SR2 (Release)	33
SR1 (The Other Decay)	34
E. Outputting to .spc/.smc	35
The "Set Recorder..." Pane	35
Tips & Tricks for .spc	37
Regarding Storage Space	37
Pitch Effects	38
Volume/Panning Fades	39
IV. How to use SNESMOD	41
A. Using the Converter	42
B. Setting Up Your .it Module	44
General Requirements	44
Echo Delay	44
Effects Column support	46
Volume Column support	48
C. SNESMOD Tracking Tips	48
Tip #1: Note Cuts	48
Tip #2: Rapid Volume/Panning Changes	49
Tip #3: Space-Saving	50
Tip #4: Envelopes	52
Tip #5: Controlling SPC700 Features	53
V. Dealing with SNES Samples	55
A. Ripping Samples from .spc Files	55
Additional Tips and Quirks of .spc Ripping	57
Condensed Step-by-Step Guide	58
B. Preparing & Optimizing Samples	59
Sample Rate	59
Loop Points	59
.brr Format	61
Pre-Emphasis	61
C. Repository of SNES Sample Rips	62
Credits & Thanks	64

I. Introduction

HELLO AND WELCOME to **INTENSNES TECH... with HVB**, the poorly-named spinoff of the "[Intense Tech with Defense Mech](#)" series aimed at **creating chiptunes for the Super Nintendo Entertainment System**, also known as the Super Famicom in Japan. This guide aims to familiarize any reader with what the SNES can do musically, and how to do it, with as little jargon as possible. Whether you're new to making SNES music or just want to dig deeper into its features, this will be the guide for you!

The SNES's soundchip, **SPC700**, supports 8 channels of sampled sound and has a built-in echo delay system. A "Gaussian" filter is also applied to the entire output, dulling some of the higher frequencies and covering up some of the noise created by its compressed **sample format, .brr (Bit Rate Reduction)**. These features together give the SNES its distinct characteristic sound.

We will be taking a deep dive into **two possible methods** of creating **hardware authentic SNES chiptunes** and going over the SPC700 chip's features along the way:

1. **C700 VST**: A VST plugin through which you can input MIDI data and record playback to **.spc (SNES chiptune file)** or **.smc (SNES ROM)**. Notably, this means you can create hardware compatible chiptunes right in your DAW of choice!
 - » Download: <http://picopicose.com/software.html>
 - » Download option 2: <https://github.com/osoumen/C700>
 - » Experimental fork "C700 Microtunable" which adds support for Scala tuning files: https://drive.google.com/file/d/1iRs3ZIZ3e_tuu5cE6BWjfYx71TDWX6Bz/view
2. **SNESMOD**: A converter tool that takes an 8-channel **.it tracker module file** and converts it to **.spc (SNES chiptune file)**.
 - » Download (newer fork of the original): <https://drive.google.com/file/d/17P4thxLaZPnT7VsFkXMbOsqeIPwA6TXV/view?usp=sharing> (documentation included)
 - » Download (older original version, not recommended): <http://snes.mukunda.com/>

Each of these options has its own perks and downsides, which we will cover. However, they are currently the most **fully-featured** methods of working with the SNES's SPC700 soundchip in music-making software.

Additional options to consider:

- If you're comfortable with creating music through MML (essentially [programming the music directly in text form](#)), a third feature-rich method is [the compiler "AddMusicKFF"](#). To learn more, check out pedipanol's [MML guide](#).
- If you are only after "authentic SNES sound" to use in a DAW for fuller productions, consider the [\(paid\) VST "chipsynth SFC"](#) as well, which has accurate and full-featured SPC700 sound usable like a synth. However, it has **no file export**, making it outside the purview of this guide.
- The multi-chip tracker [Furnace](#) offers SPC700 as an option with **no file export**.
- If you own a SNES and a MIDI keyboard, [Super MIDI Pak](#) both lets you play your system like a synth *and* can produce .spc files; this of course is a costly option.

After learning about both C700 and SNESMOD, we will cover in detail how to handle **SNES samples** – both how to **rip samples from .spc files** (including all your favorite soundtracks) and how to **prepare non-SNES samples** for use on the SNES with as few hiccups as possible. There is also a living repository of ripped samples from games!

How to Listen to SNES Chiptune Files

Crucial to our mission is being able to tell that our chiptuning work is done correctly. How do we **play back an .spc or .smc file**?

The standard tool used for playing back **.spc** files is named **SPCPlay**, and you can [download it here](#). It is newly updated sometimes, so check back every now and then for new versions. Several other options for .spc playback are listed [on SNESMusic.org](#), including foobar2000 and Winamp plugins as well as multiplatform options.

For **.smc** files (or .sfc, which is usable interchangeably), just about any SNES emulator will do, since they are just Super Nintendo ROMs that play back a song. A popular multiplatform option is [bsnes](#), and a popular Windows option is [Snes9x](#).

II. C700 vs SNESMOD

So, which option should I choose?

After extensive use of both and seeing others use both, my verdict is that **C700 should be your go-to method** unless you are a die-hard .it tracker person or creating music for use in an actual SNES game. C700 can be used with any music-making software that supports VSTs and MIDI, produces higher quality sound, and has fewer audio glitches to deal with. Its ability to export to ROM (.smc) allows for more sample and song space than would be possible for .spc. However, if you're determined to work with the .it format, SNESMOD can get the job done, and a more recent fork adds an array of fine controls over SPC700 chip features which are comparable to C700's.

What do you mean C700 produces higher quality sound?

.spc (SNES chiptune) files contain three things:

- 1) **Song data** (all the notes, their velocities, panning, volume, etc.)
- 2) **Samples** used in the song
- 3) **Sound driver** code

That 3rd thing is very relevant to how your song will sound, as the sound driver determines how all that song data interfaces with the SPC700 soundchip. In this respect, not all tools are created equal. Despite improvements to SNESMOD, its sound is slightly less crisp, and it is very frequently plagued by unwanted "pop" or "click" sounds between notes. More on that in the SNESMOD section.

C700 also "dynamically allocates" every note to one the 8 channels of the soundchip, meaning it tries its best to avoid the SNES's polyphony restrictions by separating the notes out across different channels to let them ring. Depending on the arrangement of your song, this may result in a fuller sound as well.

Additionally, C700 comes with an option to "**preemphasize**" **samples** (on by default) to counteract the global Gaussian filter applied to the SNES's audio. This, too, allows for crisper sound in C700 by comparison, though you yourself could perform preemphasis on your samples – a tool for this is included with the modern SNESMOD fork.

As an example, here are four versions of the same* song produced by these tools, two apiece:

(*To be fair, these are two different project files with slightly different ADSR envelopes, vibratos, and echo settings; the SNESMOD example is tracked in OpenMPT whereas the C700 example is sequenced in FL Studio. The samples and notes are the same, though.)

EXAMPLE 1: SNESMOD (properly tracked)

[EX1 \(Google Drive link\)](#)

This one is about as good as you can get SNESMOD to sound (although the samples were not preemphasized). Though it's a little dry, it's mostly free from unwanted pops and crackles. You can hear a bit of that between 0:12 and 0:19, though, as well as at 0:26.

EXAMPLE 2: SNESMOD (improperly tracked)

[EX2 \(Google Drive link\)](#)

This example showcases SNESMOD's glaring "pop noise" issue, which most frequently occurs when new notes trigger in a channel where a previous note was playing. This can be carefully avoided by cutting notes 1 tick before new notes play using the **SCx** command (Note Cut after **x** Ticks), where **x** will equal *your speed setting minus 1*. In this example, my song is at speed 3, and I removed all SC2 commands from the song. This is the result. You can hear popping noises all throughout the melody. Echo delay even applies to them.

EXAMPLE 3: C700 (no echo or sample preemphasis)

[EX3 \(Google Drive link\)](#)

In this example, echo delay was disabled and preemphasis was turned off, to showcase the raw sound for comparison. Even without preemphasis the sound is substantially cleaner and brighter.

EXAMPLE 4: C700 (full)

[EX4 \(Google Drive link\)](#)

The final version of the song, with preemphasis enabled and echo delay on (notably a higher echo setting than the SNESMOD examples used).

So wait, you said C700 has disadvantages too.

Yep. Data efficiency. Let's talk about data, storage, and filesize for a second! **The SNES's soundchip has 64KB of RAM** for audio purposes, which in practice refers to the aforementioned song data, samples used in the song, *and* a variable amount of space used for the SNES's built-in echo delay system, depending on how long of a delay value you choose. All of that has to add up to **64KB or less**, else you will meet an overflow error.

This is a crucial point in which C700 and SNESMOD differ: C700 is a *recorder*, while SNESMOD is a *converter*. C700 records MIDI data played through it, and MIDI data adds up *fast*. Every step of every pitch bend, every vibrato and portamento, every panning and volume change – all of that is data that C700 will be recording, and in practice, this means you have to be very careful with automation and event data. **It is an order of magnitude less efficient than SNESMOD**, whose simple tracker data will instead just say "I want a vibrato of speed X and depth Y, please."

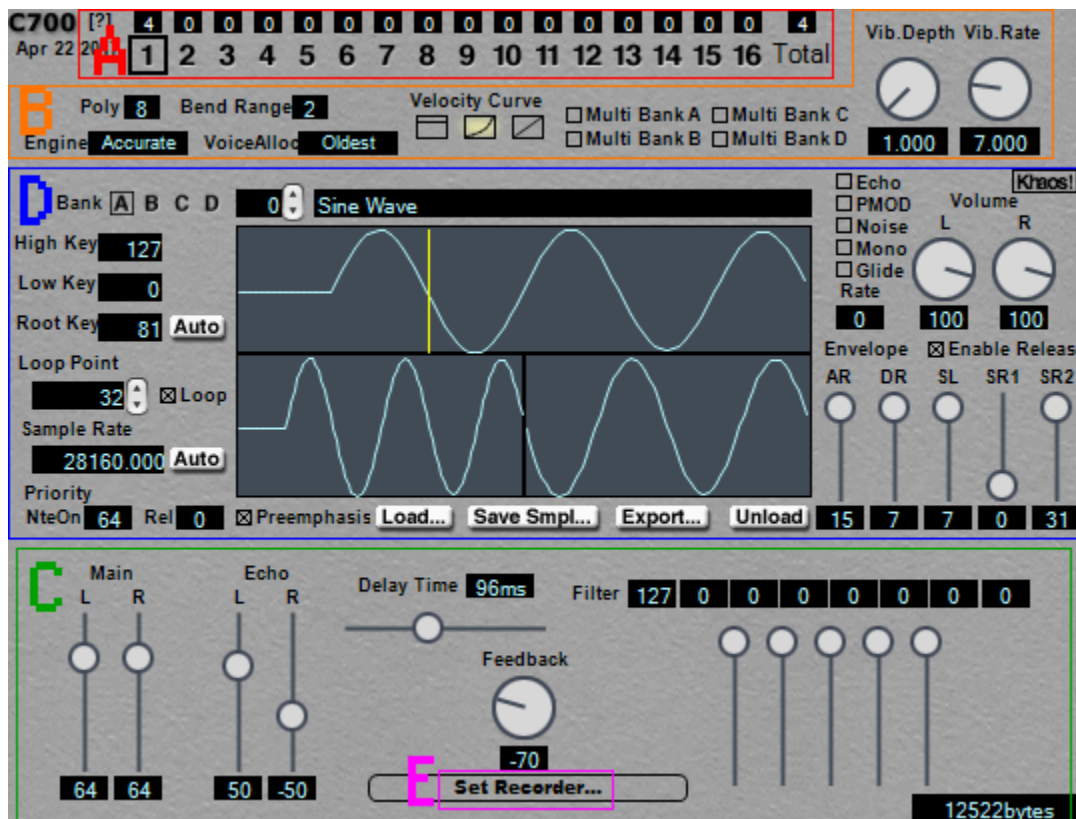
However, **C700 has one more ace up its sleeve: SNES ROM (.smc) output!** If .smc is an acceptable filetype for your chiptune, i.e. you don't need to be making .spc specifically, the need to be careful with MIDI event data is heavily diminished. **With .smc output, the song data is offloaded to the ROM**, leaving you with the ability to nearly fill your allotted 64KB of "ARAM" with just samples + the variable amount required for using the echo delay feature. **I highly recommend this method** for that reason alone. Not only does C700 make up for its inefficiency this way, but you have way more room for samples than you would in any other method. And who doesn't want to push boundaries?

One thing to note: Due to C700's inefficiency with data, it is not a recommended choice if you intend to create music for use in an actual SNES game.

III. How to use C700

The many fun features of the SPC700 chip will also be covered throughout this section. Even if your goal isn't to use C700 specifically, this section may prove helpful if you wish to learn about what the soundchip can do.

In [section A](#), we'll get set up with MIDI. In [section B](#), we'll cover the global settings of C700. In [section C](#), we'll talk about volume and the SNES's echo delay feature. In [section D](#), we'll get our samples set up and ready to play. Lastly, in [section E](#), we'll set up output for .spc/.smc files and cover optimization tips.



Here's what we're working with. A sampler section (D) in the middle, and lots of global settings above and below.

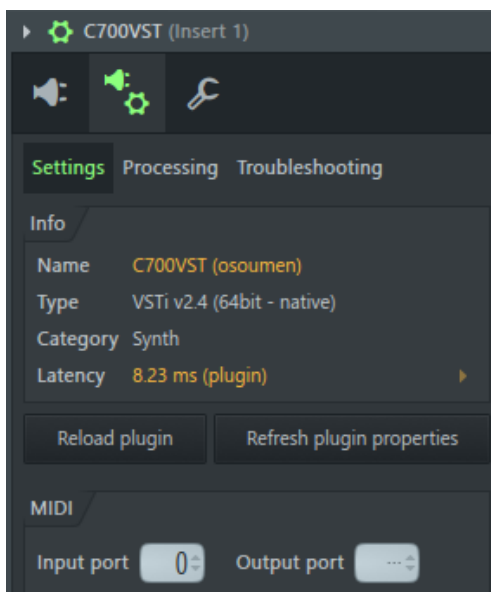
A. MIDI Setup

First things first: Load up **one (1) instance of the C700 VST** in your music-making program. Consult with its manual if you are unsure how to load a VST plugin. You may need to specify its location in your DAW's settings in order for the plugin to appear. In order to use C700 to record chiptune files, we must use *only one* instance of the plugin.

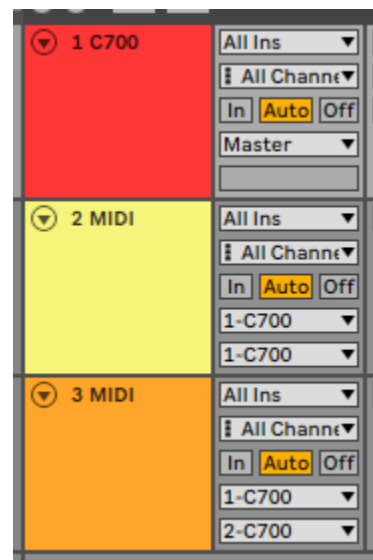
In section A, at the top of the plugin, the **numbers 1-16** correspond to **MIDI channels**. Clicking on the numbers will reveal the samples assigned to each channel (we'll cover the sampler in section D). Above each channel number is an indication of the **maximum polyphony reached by each channel**, which you can reset by hitting "stop" in your DAW's playback controls. In our example above, channel 1 has *at some point* had 4 notes playing at once, but no other channels have produced sound. The channel numbers also light up when notes play in that channel.

These 16 channels are how you'll be sending C700 data, but first we need to make sure C700 can *receive* data. In your DAW's settings for the VST, **assign C700 a "MIDI input port"** of your choosing. This is the port you'll be speaking to the VST with.

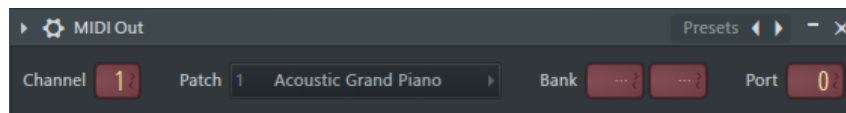
Example from FL Studio: Click the gear icon at the top left of the C700 plugin window to access the settings pane. In this example the assigned "MIDI input port" is 0.



Example from Ableton Live: On each track, there are four dropdown menus. The top two control where the signal comes from, while the bottom two control where it's going. Set the bottom two to output the MIDI tracks to one of C700's 16 inputs.

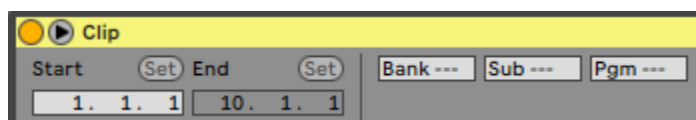


Next, create [up to] 16 instances of your DAW's MIDI plugin (in FL Studio it is named "MIDI Out"). Assign each instance a unique channel number, 1 through 16, and **also** assign each one the **port number you just chose**. This means that the instance set to channel 1 will be sending data to channel 1 in the VST, and so on.



Example from FL Studio: Channel is set at top left, Port at top right.

You can also use these MIDI channels to select **which sample** you want to be played. The **"Patch" number** you choose will correspond to the **sample's assigned number in C700**. Some software, such as our example FL Studio above, begin numbering at 1, while others begin at 0. (As you can see on the right, C700 begins at 0.) Regardless, the very first patch number will correspond to the very first sample, and so on. Multiple MIDI channels can be assigned the same patch (i.e. sample), and the patch can be automated to change within your song (called a "Program Change" in MIDI terms).



Example from Ableton Live: Ableton's equivalent to the Patch setting is in the Clip settings. Click the arrow and use the program (Pgm) box to choose the C700 sample.

Note that you can assign samples to the MIDI channels within the VST too (and this is a recommended practice for organization), but using this method exclusively does not allow for Program Changes within your song. If you do choose to set your samples up this way, **set a patch of "none"** instead of any number. Assigning samples to MIDI channels within C700 will have no effect unless *no patch* is set in the MIDI channel.

C700 comes with a few basic waveform samples already loaded up for you, so try clicking through the channels in the VST and changing the patch numbers on your MIDI channels to get a sense of how it behaves, and consider creating a template for this portion of setup. [Here](#) is a template for FL Studio 20, though I recommend learning this setup method yourself too, since it is widely applicable to other VSTs that can receive information from multiple MIDI channels.

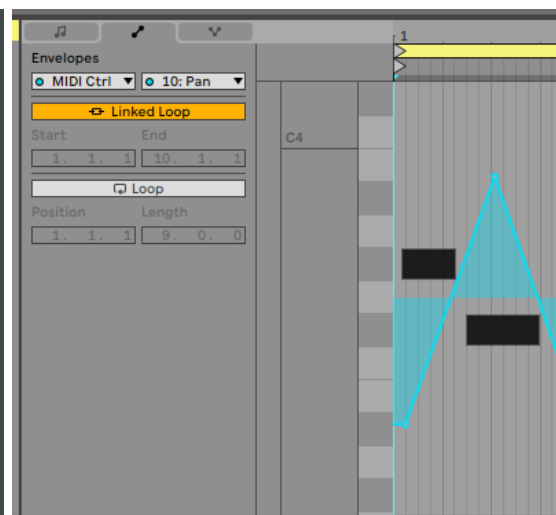
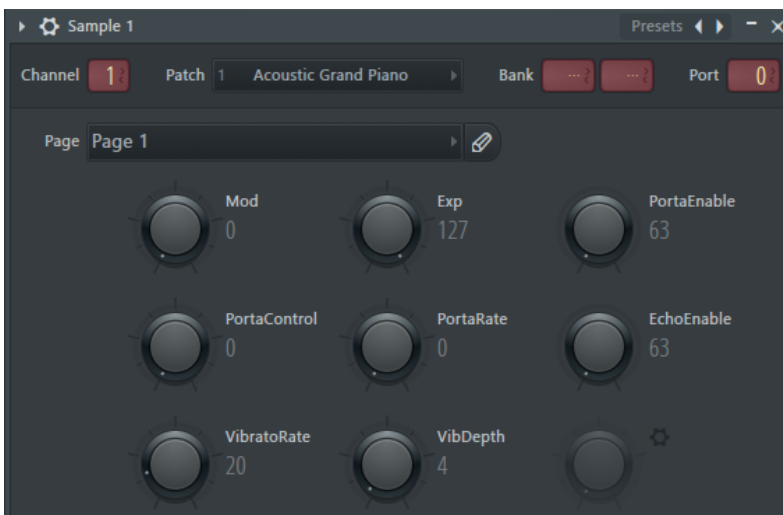
Advanced Setup: MIDI CCs (Control Changes)

Clicking on the "C700 [?]" text at the top left of the plugin will reveal the following **list of numbers and associated effects** to the right, most of which will be discussed or referenced individually later.

Each of these numbers is a "**MIDI Control Change**" or CC for short – put simply, CCs are **parameters that you can automate or assign values to** using MIDI. Your 16 MIDI channels should have assignable knobs or values for these controllers; check your DAW's manual if you can't find this. The listed number (such as **1** for **Modulation Depth**) is referred to as the "CC number", and it can be assigned a value between 0 and 127. For effects that are simply off/on, values 0-63 represent "off" and 64-127 represent "on". Other effects will map their minimum and maximum values **proportionately** to the 0-127 scale.

MIDI CCs are assigned and controlled on individual MIDI channels. If you're making a template file for your future C700 exploits, it may be useful to set these up to be used if desired. Below are examples for FL Studio & Ableton Live. [Here](#) is a Reaper CC map with custom names set for each CC.

- CC 1: Modulation Depth
- CC 5: Portamento Rate
- CC 6: Data Entry (LSB)
- CC 7: Volume
- CC 10: Pan
- CC 11: Expression
- CC 38: Data Entry (MSB)
- CC 55: Channel Limit
- CC 56: Channel Priority
- CC 57: Release Priority
- CC 64: Damper
- CC 65: Portamento Enable
- CC 72: SR2 (5bit)
- CC 73: AR (4bit)
- CC 75: DR (3bit)
- CC 76: Vibrato Rate
- CC 77: Vibrato Depth
- CC 80: SL (3bit)
- CC 82: SR1 (5bit)
- CC 84: Portamento Control
- CC 91: Echo Enable
- CC 92: Pitch Modulation On
- CC 93: Noise On
- CC 120: All Sound Off
- CC 121: Reset All Controller
- CC 126: Mono Mode On
- CC 127: Poly Mode On



In most cases*, using CC numbers is only necessary if you want to automate or change that particular setting somewhere within your song. As we'll be covering in the rest of this guide, most of these settings are available to enable within the C700 interface – just not automatable (i.e. changeable within the song) through any other means. **Note** also that any automation of these parameters is *not reflected visually* in C700's interface.

*CC#1 (Modulation), which controls automatic vibrato, is an exception to this. Its effect is only available as a MIDI CC, though vibrato can be programmed manually with channel pitch too.

If you want automation control for more than volume/panning/pitch, you will have to get comfortable with MIDI CCs. They will offer you finer control over your work.

Super Advanced Setup: MIDI NRPNs & RPNs

Note: I would recommend skipping this section for now unless you are very comfortable with MIDI CCs and/or the SNES soundchip already. This section covers using multiple MIDI CCs in conjunction to write specific values to the soundchip's "DSP Register", which is both powerful and dangerous.

If you checked out the CC list in the VST as described above (by clicking "C700 [?]" in the top left), you might have noticed a portion not yet covered:

NRPN
rrH(LSB)
7eH(MSB): DSP Register Write

RPN
00H(LSB)/
00H(MSB): Pitch Bend Sensitivity

Oh god, right? Well, once we make some sense of what all this is telling us, it's manageably complicated. "NRPN" and "RPN" are MIDI terms that refer to **two CCs each** – **#98-99** for NRPN, and **#100-101** for RPN. They both receive additional parameter data from **MIDI CC#6 (Data Entry LSB)** and **MIDI CC#38 (Data Entry MSB)**. This means that in total, **four CCs are used** to send a single parameter change. There's a lot of MIDI terminology nested in there; if you (optionally) want to know more about what the terms mean, I recommend [this brief guide](#).

NRPN: DSP Register Write

⚠ **Warning!** ⚠ Before we dive into this advanced feature, *please* use caution with the volume of your system. You may cause glitches and unexpected sounds by writing to the DSP Register, and in particular, **modifying echo delay settings has the potential to cause a feedback loop**, so be prepared to mute or otherwise disable sound, especially if using headphones. Hearing damage is no joke. Be extremely careful.

The arcane text that C700 gives us for NRPN is actually explaining the proper values to set for **MIDI CC#98 (NRPN LSB)** as well as **MIDI CC#99 (NRPN MSB)**:

CC#98: The indicated value of "rrH" means a variable in hexadecimal; rr will be the **hexadecimal value of the DSP Register you're writing to**. [A list of those DSP Register hex values can be found here](#). Convert the hexadecimal value to decimal to get the appropriate CC value. (Visit <https://hextodecimal.com/> if you need help with that.) For example, 0C is the value for "Main Volume (left output)", which is 12 in decimal. You would then set this CC to 12 to indicate you'd like to control the left Main volume.

CC#99: The indicated value "7eH" means "7E in hexadecimal", which is **126**. This CC will be set to 126. Easy.



Once you've set both the NRPN CCs, the two Data Entry CCs are used to actually **choose the value you're sending** to your chosen DSP Register. **CC#6** is your main controller for the value. In our example above, setting it to 0 will mute the left Main volume. In theory, **CC#38** can be used for additional fine control in the neighborhood of the CC#6 value, but this may not be useful in practice, since these values don't have a wide enough range to take advantage of this. (It likely exists because of standard MIDI implementations, though notably 6 and 38 are swapped from their usual MIDI usage.)

Since these CCs are assigned to MIDI channels, if you wish to control **more than one DSP Register parameter simultaneously** in your song, you will need to set up a new group of these CCs on a different channel. It doesn't matter which channel(s) you use in this case. A channel isn't locked into the one parameter forever though; changing CC#98 mid-song will allow you to control a different parameter *instead* (be careful to not immediately write the same CC#6 value again to your new parameter!).

The NPRN DSP Register Write is the **only way to automate certain parameters** of C700, such as the **volume sliders** and the **echo delay time, feedback, and filter**. One limitation of this feature, however, is that *the values can only be positive*, despite the soundchip possessing a range of -128 to +127 for many settings*.

* If you are determined, [another fork of the software](#) adds a negative/positive switcher to CC#38.)

Take extra care when handling the echo settings; it is very easy to create a **feedback loop** or **horrible distortion** by tweaking these settings, and you do not want hearing damage! Careful use of this feature can lead to some pretty amazing sounds, though.

Potentially useful **NRPN (CC#98)** values include:

0C (12) - Main Volume (left output) - set CC#98 to 12, then use CC#6 to control

1C (28) - Main Volume (right output) - set CC#98 to 28, then use CC#6 to control; etc

2C (44) - Echo Volume (left output)

3C (60) - Echo Volume (right output)

0D (13) - Echo Feedback ⚠ BE CAREFUL

7D (125) - Echo Delay Time (make sure you have enough memory for your maximum setting by checking with the slider in the C700 interface) - **use values 0-15** (0 = 0ms delay, 15 = 240ms delay). Higher values are technically unsupported by the system (& will fail in .spc export specifically), and they wrap around anyway (i.e. 16-31 is another cycle of 0ms to 240ms, so is 32-47, etc.).

xF (15, 31, 47, 63, 79, 95, 111, 127) - Echo Delay Filter coefficients ⚠ ⚠ ⚠ BE EXTRA CAREFUL! It's very easy to create feedback loops with these, but they control the prevalence of frequency ranges within the echo delay – the 8 values above the 5 sliders at the bottom-right of C700 (covered in [section C: Echo Delay](#)).

The other DSP Register options are either controllable via other built-in MIDI CCs or are likely not useful unless you're involved in SNES game or tool development. If you are, check the list linked above.

RPN: Pitch Bend Sensitivity

Closely related to "Pitch Bend Range" (covered in [section B](#)), this advanced function is considerably more benign than handling the SNES's DSP Registers yourself, but it does provide a unique option.

C700 already has a "Bend Range" global behavior setting (covered in section B below), but it can't be automated. It is limited to a full span of +/- 2 octaves, which is quite substantial already.

Pitch Bend Sensitivity, however, proportionally changes the **sensitivity of pitch data in general**. This means that 1 cent of pitch-change will no longer represent 1 cent, *but* it also means that you can **pitch bend farther than two octaves** – in fact you can pitch bend to the whole playable range of the sample, if you want.

The setup is as follows:

CC#100 - 0

CC#101 - 0

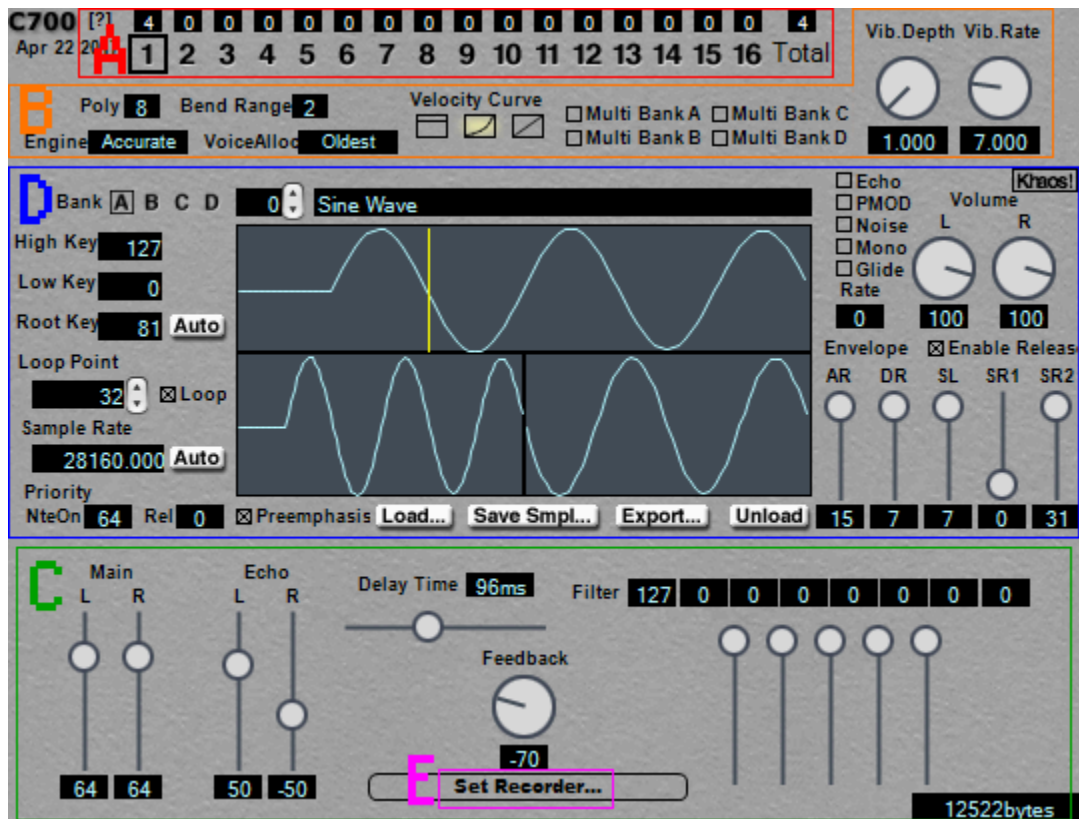
CC#38 - 0 (changing this will affect how the start of your sample sounds, so play around with it if you're curious)

CC#6 - the main control for the setting, ranged 0-127, but most values above 40-50 exceed the entire span of the sound's playable range. 0 is no pitch bend at all. 12 is approximately 1 octave (though not exactly).

It is not easy to control this feature, but if enormous pitch bends are desired, perhaps as an effect, RPN Pitch Bend Sensitivity can be used. Note that this **can't be used on a channel currently controlling NRPNs**, since they share CCs #6 and #38, so if you wish to use multiple parameters, place this one on a different MIDI channel.

It may be useful to use **CC#121 (Reset All Controllers)** to "deactivate" your Pitch Bend Sensitivity changes if the desired effect is only temporary. This will indeed reset all other CCs as well, though.

B. Global Behavior Settings



This is the same image as before, but relocated for ease of viewability!

In this section we'll be covering the global behavior settings available in C700, located below the MIDI channel and polyphony info in **section B**.

All of the areas with **black backgrounds and blue text** can be modified either by **clicking and dragging** up/down, or by **clicking and typing** in a value. Some only respond to one or the other. If one doesn't work, try the other! (This is true for the entire interface, not just this section!)

Poly & Engine

Poly 8

Engine Accurate

Let's talk about **polyphony on the SNES** (and in C700). This soundchip has a maximum polyphony of 8 - that is, at most, **eight different notes can be playing at once**. This is what **Poly 8** represents, and if you're trying to make hardware-authentic chiptune files, **Poly** and **Engine** will both never be changed.

If you're *not* trying to create a hardware-playable chiptune, changing the **Engine** setting from "**Accurate**" to "**Relaxed**" will allow you to increase the **Poly** setting up to a maximum of **16**, as well as load larger samples above the 64KB limit. However, in "**Accurate**" mode, **Poly** will be locked to 8 no matter what. Don't mess with these two settings if you're here to make chiptune files, but this is an exciting option for larger productions!

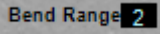
VoiceAlloc

VoiceAlloc Oldest

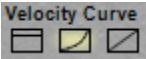
The **VoiceAlloc** (voice allocation) setting determines in what way C700 makes cuts to obey the 8-polyphony restriction. There are two options, with the default being "**Oldest**". This means that **in the event of a new note surpassing 8 polyphony, the oldest currently-playing note will be cut to make room for the new note**. The other option, "**SameCh**" (same channel), alters this rule slightly: if a new note would surpass 8 polyphony, C700 will cut the oldest currently-playing note *in the same MIDI channel as the new note*. If no such note exists (i.e. the new note is the only active note in the MIDI channel), the "**Oldest**" behavior will still apply.

Additional, optional information about voice allocation in C700: **Voice allocation to the soundchip's 8 channels is dynamic**, meaning that the VST does its best to separate the notes out across channels so as many things as possible can be heard without anything being cut. This cannot be controlled and has no adverse effect on the sound (aside from channel muting in your .spc player / emulator being made impractical), but it is an interesting feature of the software. Depending on the arrangement, a fuller sound may come about than if each musical line were sequenced in its own SPC700 channel. Notably, if every sample is set to Mono mode, voice allocation is static.

Bend Range

The **Bend Range**  setting controls how many semitones you can bend the pitch of your channels. By default it is set to **2**, which means you can bend the pitch as far as two notes away in either direction. Increase this value to be able to bend further! Even though the VST lets you increase this number to 24 (which is the maximum in the MIDI spec), some programs may restrict you to **12** (one octave up or down). In some programs you may need to set this to 12 to achieve two semitones of bend range. A related, more advanced feature that expands Bend Range proportionally is the MIDI RPN "**Pitch Bend Sensitivity**", covered in the **MIDI Setup** section above.

Velocity Curve

The **Velocity Curve**  setting handles how you want C700 to interpret the velocity parameter of your notes (i.e. their loudness). You have three options. On the **left** is a straight horizontal line: velocity will be *ignored* and all notes will be played with the same velocity no matter what. On the **right** is a linear slope upwards: velocity will be *linear* with all velocity values interpreted literally. In the **middle** is actually our default setting, a curved shape that makes it so the uppermost velocities have greater volume difference than the lower/middle ones. Choosing between this and linear is up to personal preference.

Vib. Depth & Vib. Rate

Vib. Depth and **Vib. Rate** (vibrato depth and rate) are *global settings* that define **how vibrato will sound if enabled** on a channel. **Depth** controls how far up and down the pitch will waver, and **Rate** determines how quickly the vibrato waveform will cycle. The default Depth of **1.000** means that the full pitch range of the vibrato is 1 semitone – that is, the lowest and highest point are 1 semitone apart. These settings apply globally to all channels, although they can be automated (MIDI CCs #77 and #76 respectively), and vibratos of your choosing can also be programmed by using MIDI channel pitch instead. Specifically, these two settings affect the automatic vibrato heard by using **MIDI CC#1 (Modulation)**, which you might also know as "the mod wheel" if you have a MIDI keyboard.



Multi Bank A/B/C/D

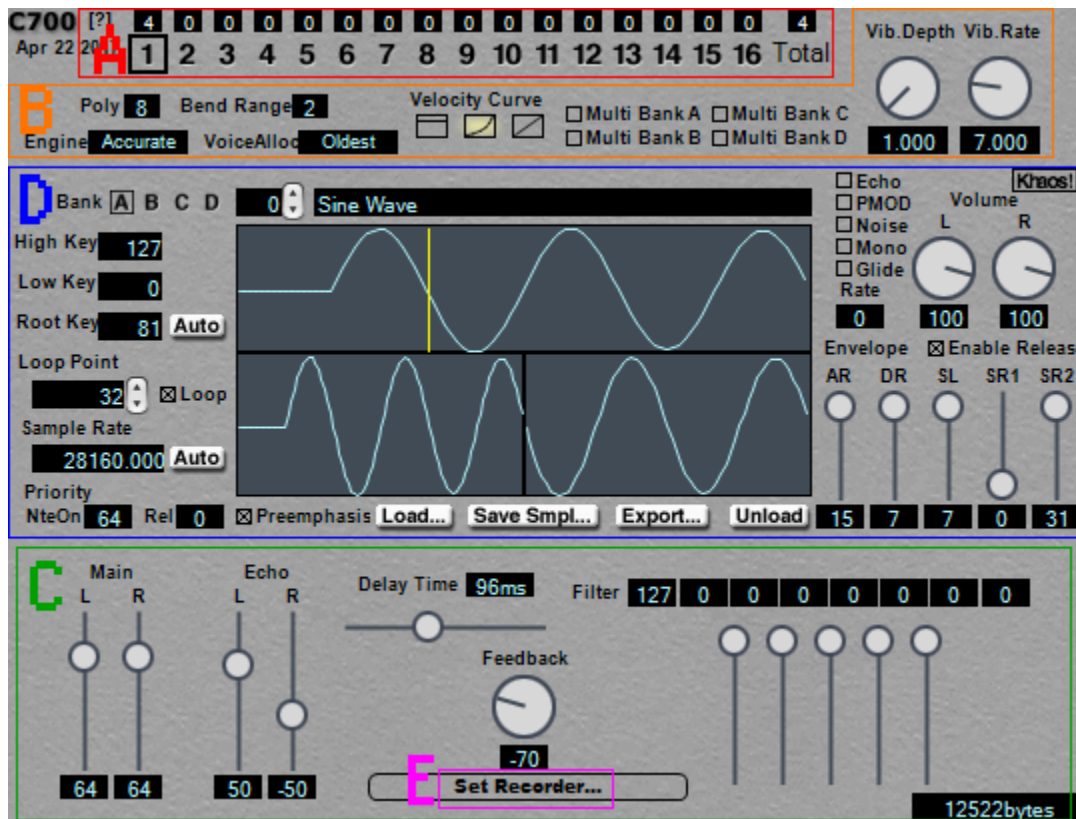
☐ Multi Bank A ☐ Multi Bank C
☐ Multi Bank B ☐ Multi Bank D

C700 can store samples in four different banks (A through D). Turning on **Multi Bank** mode for a bank will allow you to play multiple different samples in that bank without the need to change channels or use a Program Change event to swap samples – most commonly this is used for mapping a drumkit to one channel.

Multi Bank has to be used in conjunction with the sampler's "**High Key**" and "**Low Key**" settings to have any effect. The numbers in those settings correspond to notes (**0** being the lowest, **127** highest). For example, if you wanted to map a simple drumkit, you could set a kick drum to a range of **66-66**, snare drum to **67-67**, and tom to **68-74**. Only one note would be able to trigger the kick and snare, but a small range would be available to the tom.

If this method suits your workflow, making use of the banks will be very useful; however it is entirely possible to ignore this feature and sequence drums in their own individual MIDI channels, too. **Samples all default to bank A**, so it's recommended to select bank B, C or D for all the samples you want to group together, then enable Multi Bank for whichever one you chose.

C. Volume & Echo Delay Settings

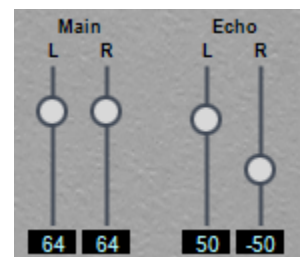


Wow, it's this image again! That means it's time for a new section!

Now we will cover the section at the bottom, **section C**, which mainly concerns **global volume** and the SNES's built-in **echo delay** feature. Although the echo delay settings are global, each sample has a toggle to disable/enable echo delay *for that sample*, and the toggle can be automated on a per-channel basis as well (using **MIDI CC#91**).

Volume Sliders

The sliders at the bottom left, "**Main**" and "**Echo**", affect the **global volume output** of the plugin – Main governs the "dry" volume of your song while Echo governs the echo delay's volume. The left (L) and right (R) stereo can be controlled separately in both cases. C700 defaults to a Main volume of **64**, which can comfortably be raised to the maximum value of **127** in most cases!



You will notice that the range of values also includes negative numbers; **negative numbers invert the phase of the waveform**. If both numbers are negative or positive, this has no practical meaning; however, a positive and negative number (as in our example for **Echo**, L=**50** while R=**-50**) will cause **sound to be canceled out** if played back in a **mono** setup. C700 **defaults** to these values for the Echo volume sliders, so if you want your music to be mono-compatible, make sure to change these values to match. Note that there is a timbral difference in this echo setting though – in stereo, it will have a wider, "surround-like" effect.

Here's some Echo slider examples to illustrate the difference in sound:

EXAMPLE 1: C700 DEFAULT ECHO SLIDER VALUES (50 and -50)

[EX1 \(Google Drive link\)](#)

EXAMPLE 2: EQUAL ECHO SLIDER VALUES (both 50)

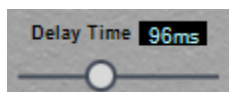
[EX2 \(Google Drive link\)](#)

EXAMPLE 3: EXAMPLE 1 BUT MIXED DOWN TO MONO (i.e. no delay is heard)

[EX3 \(Google Drive link\)](#)

Echo Delay

The whole rest of this pane is devoted to **echo delay settings**, so let's learn everything there is to learn about this key aspect of the SNES sound. A few parameters can be controlled: **Delay Time**, or the length of time between delay notes triggering; **Feedback**, or the volume that gets fed back into the delay to keep the effect going; and **Filter**, which lets you control the prevalence of a few frequency ranges within the echo delay.



Delay Time

The **Delay Time** setting, which affects the **amount of time until delays occur**, can be controlled in increments of **16ms** (milliseconds), ranging from **0ms** to **240ms**. For example, if set to maximum, the following would occur: [note] - [240 milliseconds] - [first delay note] - [240 milliseconds] - [second delay note], and so on. At a value of only **16ms**, the space between delays will be quite low.

Interestingly, the **0ms** setting does produce additional sound and is *not* the same as disabling echo delay entirely. It can be used in conjunction with the **Filter** setting (described below) to provide a sort of EQ boost, though you lose the ability to utilize longer echo delay at the same time by doing this. [Here is an isolated example of 0ms Delay Time. Main volume is set to 0, so only Echo is heard, and exactly one slider from the Filter setting is being raised & lowered at a time.](#)

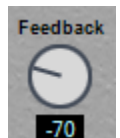
Notably, **each increment of 16ms takes up 2KB of your allotted 64KB of space**, meaning that the higher you set your Delay Time, the less space you will have for samples (and potentially song data, if you are making .spc rather than .smc). At the maximum setting of **240ms**, this effect takes up a whopping 30KB of space!

At the bottom-right of C700, you will see an indication of how much space is being taken up by echo delay, samples, and sound driver **12522bytes**. Watch the amount change as you move the Delay Time slider. If you exceed 64KB, this indicator will turn **red** until you've cut back enough size. You may have to restart your DAW as well if your playback sounds glitchy after experiencing this.

Feedback

The Feedback setting is a single knob that determines the volume of sound to "feed back" into the echo delay to keep the effect going. In practice this means **the higher you turn this knob, the longer the delay will keep occurring**. Turning it to maximum will make the delay perpetual, while turning it to **0** will make it so there is exactly one delay and then it stops. Most likely, you will want to choose a space in between.

Note that the available range is **-128** to **127**, much like the Echo (and Main) sliders. A negative value inverts the phase and slightly alters the sound. Experiment with what sounds best to you on any given song! Be careful of causing a feedback loop with large values; be ready to mute volume.



Filter

The Filter setting (also called the "FIR filter") lets you control **how prominent certain frequency ranges are within the echo delay effect**. Five sliders are available to control, and by default, all are raised to maximum, meaning **the default echo delay will apply fully across all frequencies**. On the left is the lowest frequency range; on the right is the highest. Turning one down all the way will remove that frequency range from the echo delay.

Loosely the five sliders correspond to (in Hz) **0k, 4k, 8k, 12k, and 16k**. Eight boxes above are also available for value input (-128 to 127), but only use these if you know what you're doing; they control the actual values the SNES uses to determine this stuff, while the sliders offer an easier, more intuitive approach. (You will notice that the sliders affect multiple values when modified.) You may be able to achieve different sounds with value input, but **be careful of feedback loops** resulting from wonky settings, especially when the sum of values is very negative or very positive.



Generally speaking, you can get away with louder, more intense echo delay if you turn down the highest-frequency sliders, but depending on the sounds you're applying the echo to, it may instead be helpful to turn down the lowest one, clearing away mud in lower frequencies. It is fruitful to experiment with these values on individual songs.

As mentioned above in the "Delay Time" section, if Delay Time is set to 0ms, these sliders can be used as a makeshift EQ setting. Check out the example linked above to hear what each frequency range sounds like.

D. The C700 Sampler



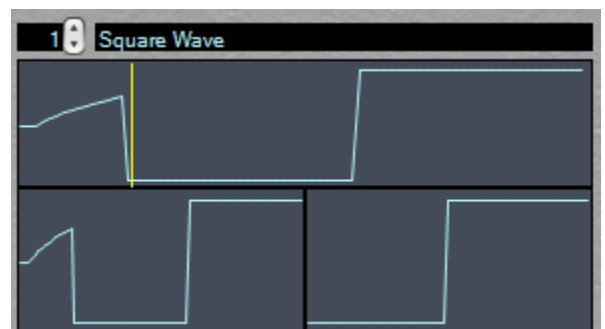
It's time for the fun stuff.

Loading and Browsing Samples

Let's get sampling! There are two ways to load samples in C700: The first is to simply **drag-and-drop** a sample from your filesystem onto the waveform display in the center of the VST. The other way is to click "Load..." at the bottom, and from there you will find that there are **three acceptable filetypes** for loading samples: **.wav / .brr / .spc**.

Importing .wav and .brr (the SNES's compressed sample format) is straightforward, but .spc import is interesting. **Importing an .spc** into C700 will add **every sample** from that .spc file into the VST at once – be warned that it also **overwrites existing samples**. This is an easy way to load in several sounds at once from a song on a soundtrack, as it doesn't even require a sample rip. Note that ADSR envelope information is not preserved. It simply loads samples from an .spc in a batch.

So you've loaded a sample into C700 now – how do you load more? The **Sample Number** located next to the Sample Name **0 Sine Wave** is where you **scroll through samples**. In our example above, sample #0 is Sine Wave; seen below, sample #1 is Square Wave. **Clicking the up arrow** (or scroll-wheeling) will take you to Sample #1, #2, etc. You can also **type a number** into the box and hit Enter to be taken to that number sample.



Samples can be assigned to the **16 MIDI channels** at the top, although **if you're using Patch numbers** on your MIDI channels as described above in section A (MIDI Setup), **this will only be cosmetic**. (Setting your Patch to "none" *will* make the assigned sample play on that channel, but it can't be changed mid-song if you do this.) Regardless of overall method, it is a useful organizational tool to be able to quickly look through your chosen samples for each channel. I recommend this practice!

Note/warning: **A minor cosmetic glitch** sometimes occurs with the sample display in C700, and can be fixed by scrolling through any sample on any channel. The glitch causes the same sample to display for every channel, though this does not seem to affect playback.

Next to **Load...** are three more buttons. **Save Smpl...** allows you to save the selected sample in .brr format. **Export...** inexplicably allows you to save the sample and its envelope settings as an **.xi instrument**, a tracker instrument format loadable in OpenMPT, among other things. **Unload** removes the selected sample from C700's memory.

You can generate a random waveform by clicking the **Khaos!** button at the top right of the Sampler section. Keep in mind it will overwrite whatever is loaded for the selected sample.

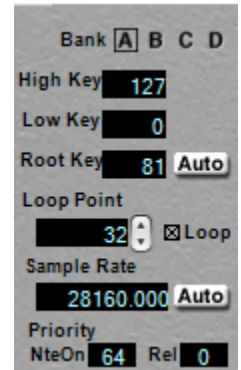
If you exceed the available memory of 64KB, the filesize indicator at the bottom-right will turn **angry red** and you will likely hear a glitchy popping sound. Once you reduce the size it will change its color back, but you may also have to restart your DAW for samples to play back correctly.

Sampler Features Pt. 1

Now that we know how to load samples and scroll through them, it's time to dig into the **settings and features available in the Sampler!** We'll start on the left side.

Bank, High Key, Low Key

These are only used in conjunction with the "Multi Bank" feature covered in **section B**; otherwise they will have no effect. Samples may be assigned to one of four different banks, A through D (all samples are in bank A by default).



When Multi Bank is enabled for a bank, **the values for Low Key and High Key determine the range of notes** where the sample will play. To get a kick drum to play on just one note, for example, you could set both values to **66**. If you then set a snare to **67**, the next note above the kick would trigger the snare instead. Continuing this pattern, you could set a small range of **68-74** for a tom sample and hear it at a few different pitches. This is a useful tool for sequencing drumkits or SFX, but can also be ignored entirely.

Root Key

Root Key is C700's root pitch setting; changing the value offsets the pitch in semitones. If you press the "Auto" button, C700 will attempt to automatically tune your sample to the correct note. (It doesn't tune to the nearest octave, so you may end up with something much higher or lower, depending on the sample.)

Loop Point

The **Loop Point** value, if "Loop" is ticked, determines where the beginning of the loop is (in *samples*, the tiny unit of audio). **This number must be a multiple of 16**, as you will see if you try scrolling. **Loop start points will be imported** if they were stored in the sample. However, C700 will do some resampling to ensure this value is a multiple of 16 if you import a sample with different settings. Data after the loop end point will be truncated (if there is any). See "Loop Points" in section V for more information.

Sample Rate

Lowering the **Sample Rate** will lower the pitch of your sample. This does not seem to actually reduce the quality or sample size, as one might expect, but it can be useful to **lower the Sample Rate if its value is very high** when imported (above 60,000 or so), because C700 has an upper limit of 120,000. Wherever that limit is reached, that's the highest note you can play using that sample.

Similarly, **if the Sample Rate value is quite low** (below around 8,000 or so), it may need to be raised. **Doubling the value will raise the pitch by one octave.**

If you press the "Auto" button, C700 attempts to tune your sample and place it in an optimal range. This behavior is always the same regardless of where the Sample Rate was previously set, i.e. it doesn't tune to the nearest octave (much like Root Key).

Despite the presence of these settings, I recommend preparing your samples' tuning and loop points outside of C700 in a more thorough sampler such as OpenMPT's.

Priority: NoteOn, Release

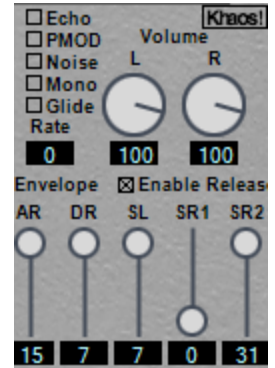
This setting allows you to choose the **priority of a sample** in the event that the sequenced notes would **break the 8-sound polyphony limit**. By default, all samples are set to a **NoteOn** priority of **64**. Lowering this value will make the selected sample the first to be cut in the event you exceed maximum polyphony. Conversely, increasing this value will ensure the sample isn't considered for polyphony cuts [until all lower-priority samples have been considered]. This setting is helpful if you find yourself brushing up against the limit and want to ensure your most important elements are never cut, or if there is a part of lesser importance you'd rather be cut first.

There is a separate value for **Release** priority - by default it is set to **0**, meaning samples that are **only still sounding due to a release envelope** (SR2) will be the **first** thing to be cut for polyphony reasons. Raising this value above the NoteOn priority values will give the release envelope a higher priority, ensuring it won't be cut so quickly if polyphony is exceeded.

Use these settings in conjunction with the **VoiceAlloc** (voice allocation) global setting to further tweak how C700 handles priority.

Sampler Features Pt. 2

Now let's look at the right side of the sampler, where we get to control several **effects that shape the sound of the selected sample**. We have volume control, several effect toggles, and a slightly unusual but intuitive ADSR envelope. **These effects apply to the sample itself**, meaning that if you assign this sample to multiple channels, the same settings will be recalled, and if they are changed, the change will occur in all cases.



(**Protip:** You can avoid this behavior by using MIDI CCs to control the parameters instead, but in most cases this interface will be sufficient and easier to use. Use MIDI CCs if you wish to automate any of these parameters.)

Sample Volume

The **Volume** knobs are a "**master volume**" setting for your selected sample. You can control the left (L) and right (R) stereo individually. Keep in mind though that **only mono samples are supported**, so this effect is somewhat limited. Most of the time, this value can be comfortably raised to the maximum of **127**. This feature can't be automated; instead use note velocity and channel volume to control how loud your sounds are within the song.

Effect Toggle: Echo

The **Echo** toggle enables the SNES's echo delay for this sample. If it is ticked, echo delay will apply to the sample. This effect can also be automated using **MIDI CC#91 (Echo Enable)**, where it will apply to the MIDI channel you assign the CC instead. Using the CC method, a sample could potentially be heard both with and without echo delay at the same time on different channels.

Effect Toggle: PMOD (Pitch Modulation)

The **Pitch Modulation (PMOD)** toggle enables a cool SNES feature for your selected sample: Pitch Modulation! This is sort of like a chaotic FM synthesis; if it is ticked, the sample will be used to modulate the frequency of another sample currently playing.

The result is usually a very distorted tone, but the sound is very distinct and can be used as a wonderful effect. With simple waveforms, a limited 2-op FM sound is possible!

On the sample (or MIDI channel) you've enabled Pitch Modulation for, **it is best to place notes slightly after the note you're trying to pitch modulate**, rather than at the same time. The offsetting will more reliably trigger the effect. The implementation of pitch modulation on the SPC700 chip is supposed to apply to *the channel before the note being used to modulate* (e.g. channel 7 pitch-modulates channel 6), so I suspect this is probably because of C700's dynamic voice allocation – new notes frequently appear in the next channel of the soundchip. This is likely also why samples with PMOD toggled on will often trigger the modulation on themselves if playing multiple notes simultaneously.

(Troubleshooting note: One reason this effect might not be triggering in specific cases is because SPC700 channel 1 can't modulate anything. But there's no way to force C700 to put a specific note on a specific SPC700 channel. So, try putting a "dummy note" before a note that should be triggering PMOD but isn't, to offset the channel count.)

Pitch Modulation can be toggled on your MIDI channels instead by using **MIDI CC#92 (Pitch Modulation)**.

Here are a few examples of this effect in action:

EXAMPLE 1: a sine wave with a long release envelope (SR2) & pitchmod enabled

[EX1 \(Google Drive link\)](#)

EXAMPLE 2: pitchmod used in a solo (along with some wide vibratos)

[EX2 \(Google Drive link\)](#)

EXAMPLE 3: pitchmod and echo delay combine to form a guitar amp-like sound

[EX3 \(Google Drive link\)](#)

Effect Toggle: Noise

The **Noise** toggle will enable Noise mode for the selected sample, a similar hissing effect to the dedicated noise channels of many, many soundchips. The SNES's noise output is not particularly robust, but **32** total frequencies are available. Note that enabling this effect here will lock you out of using the sample instead of noise (unless you use the MIDI CC listed below), although you can load a very small "placeholder" sample to avoid taking up much unnecessary space.

To avoid this behavior, **MIDI CC#93 (Noise On)** can be used on a MIDI channel to enable Noise mode for only that channel. The sample is otherwise unaffected.

Effect Toggle: Mono

The **Mono** toggle is actually one of the most useful tools for keeping polyphony under control. Enabling this mode will **restrict the selected sample to a polyphony of 1**, meaning if an additional note triggers while a note is sounding, the pitch simply shifts to the new note's pitch, and the channel never hits a higher polyphony. **The new note does not trigger a fresh ADSR envelope**, so this also has a unique sound characteristic reminiscent of many synths.

Turning this on is **highly recommended** for any sample that won't require **more than one note at a time**, such as a lead, bass, or drum. It is also helpful if you are using release envelopes (SR2), which can easily trigger extra polyphony otherwise. New notes will trigger fresh ADSR envelopes if the previous note has already been released, and the old envelope will cut off.

Mono mode is controllable with **MIDI CC#126 (Mono Mode On)** or **MIDI CC#127 (Poly Mode On)**. I am not certain why two separate CCs exist for this, since they effectively function identically (but in opposite directions). Functionally there does not seem to be a difference. The most recently-set one takes priority.

Note 1: Enabling Mono mode for every sample in your project will disable C700's dynamic voice allocating behavior (where new notes get spread across all 8 channels depending on what's available).

Note 2: Avoid using the same Mono-enabled sample on MIDI channel pairs **1+9, 2+10, ... 8+16**; if you do, each channel's note will cut the other off. There is some sort of "mirroring" split between MIDI channels 1-8 and 9-16 that causes this behavior.

Effect Toggle: Glide

Glide is the portamento setting, and also comes with a "**Rate**" box where a value of **0** (immediate) to **127** (agonizingly slow) can be input. The terms *glide* and *portamento* both refer to a **pitch slide between notes** – when a new note occurs, rather than the new pitch immediately sounding, it is approached with a pitch bend *from the previous pitch*. It's like scooping into a note when you're singing, or that big clarinet solo in the intro of *Rhapsody in Blue*. Instead of _|_ it looks more like _/|_.

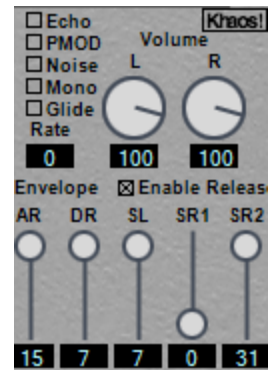
The **Rate** value controls the **speed of the portamento**. At low values, the portamento will occur quite quickly, and at high values, it takes an eternity to land on the new note.

If you want to use portamento only selectively (which is likely), use **MIDI CC#65 (Portamento Enable)** to enable this feature for a MIDI channel, along with **MIDI CC#5 (Portamento Rate)** to automate the speed. If you want to choose a specific pitch for porta to start from, use **MIDI CC#84 (Portamento Control)** to select a note – the lower the value, the lower the note.

The Glide effect can be used in conjunction with Mono mode for a very natural "synth solo lead" sound, but it also functions in regular polyphonic mode too. The previous note will keep playing while the new one begins bending upward starting from the last note.

Envelope

Now, at last, the **ADSR volume envelope settings**! If you have used envelopes before, you will feel relatively comfortable in this section already, but as you can see, a fifth slider is available ("SR1" is the additional one, which we'll cover last), making this feature a bit quirky in comparison. Careful use of the features, though, can lead to some very expressive sounds.



AR (Attack)

The Attack value determines **how quickly notes will reach their full volume** – by default (**15**), this is immediate. Lowering this slider will (counterintuitively) increase the attack time, meaning the volume will swell up from zero at a certain rate. At the lowest value of **0**, the swell will take the longest amount of time to complete. At high values like **13** or **14**, full volume will be reached quite quickly.

Controllable via **MIDI CC#73 (AR)**.

[EXAMPLE](#) - AR value at minimum (longest attack time), gradually raising to maximum

DR (Decay), SL (Sustain)

After the Attack envelope has completed its task of reaching full volume, the Decay value determines **how quickly notes will fade their volume** – and the Sustain value determines **what volume to sustain after the fade**. By default, with both these sliders at their maximum value of **7**, Decay occurs immediately and Sustain remains at full volume.

Lowering the DR slider increases the amount of **time it takes to fade the volume down** to the level chosen in the SL slider. Even the minimum value of **0** is fairly quick at less than a second long. **Note** that the DR slider will have no effect if the SL slider is set to maximum (**7**, which is default), since there is no lower volume to fade to.

Lowering the SL slider lowers the sustain volume, or the **volume level at which you want the note to sustain** after the Decay envelope has completed its fade-out task. The

lower the value set for SL, the lower the sustain volume. **Note** however that the lowest value, **0**, doesn't reduce the sound to a volume of zero. It still sustains at a low volume.

DR is controllable via **MIDI CC#75 (DR)**, and **SL** is controllable via **MIDI CC#80 (SL)**.

[EXAMPLE](#) - DR slider being raised from minimum (longest decay time) to maximum, with SL set to minimum

[EXAMPLE](#) - SL slider being raised from minimum (lowest sustain volume) to maximum, with DR set to minimum

SR2 (Release)

Going out-of-order for a moment, let's cover the rightmost slider first. This is a traditional Release envelope, which controls **the fade-out time after a note finishes** (i.e. releases). Lowering the SR2 slider will increase Release time, meaning that **the lower you set SR2, the longer "tail" you will hear after your notes**. By default, the maximum value of **31** means that the note simply cuts off immediately – a Release time of zero. Conversely, lowering this slider all the way causes notes to keep sounding perpetually!

SR2 is a potentially hazardous effect towards the 8-polyphony limit, as release tails do count towards polyphony. A quick sequence of short notes can easily cause polyphony cuts if SR2 is utilized. Enabling **Mono** mode is one way to avoid this, but may affect the sound too harshly if the reverb-like effect of long Release envelopes is desired. Keep an eye on the MIDI channel polyphony indicators at the top of the VST. Also consider changing the **VoiceAlloc** global setting to **SameCh**, meaning notes in the same channel will be cut for polyphony reasons before any other notes.

There is also a tickbox for "**Enable Release**", which is on by default. Unchecking this box disables the Release envelope. Surprisingly, there is no MIDI CC which controls this specifically; however, SR2 is controllable via **MIDI CC#72 (SR2)**, and could effectively be disabled by setting the maximum value (127) for the CC.

[EXAMPLE](#) - SR2 being slowly lowered, increasing release time

SR1 (The Other Decay)

If you thought the Decay and Sustain portions of the envelope were rather underwhelming, you're correct, and that's because the system is able to make up for it with this extremely useful slider. **SR1 controls how quickly notes fade to zero volume**, and if *any* value besides the default (0) is set, a held note will eventually fade out.

You can think of this feature as a "**master note length**" slider. Near its maximum value of 31, notes will become short little plucks. At medium and low values, notes will still be several seconds long, but they will slowly taper off to zero volume. Notes can still be cut before this time if desired - only SR2 (Release) affects what happens after notes conclude.

Use of SR1 in conjunction with DR/SL allows for a healthy amount of nuance in SNES volume envelopes. I recommend exploring this feature!

Controllable via **MIDI CC#82 (SR1)**.

[EXAMPLE](#) - SR1 being raised, decreasing the amount of time it takes to fade to zero

E. Outputting to .spc/.smc

First things first – you're going to need one additional file from [the C700 download page](#), and that is "**playercode.bin**". Here is a [direct link](#) to that file, and a [GDrive mirror](#) just in case. *This is necessary to record your .spc or .smc.*

The "Set Recorder..." Pane

By clicking **Set Recorder...** at the bottom of the VST, an additional pane of **options related to file output** can be accessed. These settings tell C700 **how to create your chiptune file** (or ROM).

The screenshot shows the "Set Recorder..." dialog box. At the top, there's a "Save Path" field with a "Choose" button. Below it are two checkboxes: "Save as *.spc" and "Save as *.smc". To the right, there's a "smc Format" dropdown menu currently set to "NTSC", and a "PlayerCode" field containing "Valid [20170319]" with a "Load" button. Further down, there are three "Set" buttons for "Record Start Pos [ppq]", "Loop Start Pos [ppq]", and "Record End Pos [ppq]". Below these are several text input fields: "Game Title", "Song Title for spc", "Name of dumper f", "Artist of Song for spc", "Comments for spc", "Repeat num for spc", and "Fade milliseconds for spc". At the bottom right is an "OK" button.

The "Set Recorder..." screen, where you set up the VST to record your chiptune file or SNES ROM.

Again, first up – if this is your first time, go ahead and locate **playercode.bin** after clicking "Load" to the right of **PlayerCode**. The VST should remember this from now on.

Select a **Save Path** at the top, which is the directory your .spc/.smc will be created in.

Tick the boxes for "**Save as *.spc**" and/or "**Save as *.smc**" as desired. As discussed earlier, you may cause a **memory overload** with **.spc** (the SNES song format), so **.smc** (the SNES ROM format) is **recommended** instead. You will have ample space to work with if using .smc. Note that you can also switch between generating an NTSC or PAL ROM via a setting at the top right.

In the newest version of C700, if you do cause a memory overload in recording **.spc**, a second file with extension **.700** will be created. Similarly to ROM export, this extra file is where song data is offloaded to. This feature is not widely supported, although [SPCPlay](#) will play them. In this case, the actual **.spc** file will contain no song data, only samples and other necessary components. Prior versions of C700 simply created **.spc** files that eventually hit their limit and stop suddenly (which was only testable through playback). **If you see a .700 file, your .spc overflowed, and you will have to reduce the amount of data in your song** in order to cleanly receive a fully working, independent **.spc** file.

The **Record Start Position**, **Loop Start Position** and **Record End Position** are set by moving your DAW's playhead (cursor) to a specific point in the song and then clicking "Set" on the right. (Values may also be typed in.) The "Record End Position" *must be set* in order for the VST to record anything at all. In our example, it's set to **16.000** which equates to 16 beats, or 4 bars of 4/4.

By default, **your song will start to loop** after the Record End Position is reached. If you don't want it to loop from the beginning, set "Loop Start Position" to wherever you want the loop to begin. **If you don't want it to loop at all**, add a bit of blank space after your song ends, then set the Loop Start Position *and* Record End Position somewhere within the blank space after your song.

There is an interesting behavioral difference in loops between **.spc** and **.smc**. The two settings at the bottom, **Repeat number for spc** and **Fade milliseconds for spc**, control how many loops will occur and the length of the fadeout after the final loop concludes, respectively. However, as the names might imply, **these settings only apply to .spc**; a **.smc** ROM file will continue looping and never fade out.

The remaining five options fill in metadata for **.spc** files. Most of these do not apply to **.smc** either, although Game Title will be preserved. SPC metadata can also be added after-the-fact with tools like the foobar2000 or Winamp plugins that enable **.spc** playback (foo_gep and SNESamp respectively).

Let's get recording, then! As long as you have **Save as *.spc/*.smc checked**, the C700 recorder is listening. One option to record is simply to **play back your song in real time**. It will start recording at your chosen Record Start Position, and once it reaches the Record End Position, your .spc/.smc will be output to the File Path you selected. (Note that there is no indication within the plugin that this is happening.) You should be able to achieve the same result by exporting an audio or MIDI file from your DAW, too, if you (understandably) don't want to record in real time every time.

Tips & Tricks for .spc

Creating an **.spc** in C700 **without overflowing on memory** is a fair bit trickier than using the .smc output, but it is absolutely doable if you know what to avoid, as well as where you can cut corners. You may have to get more hands-on with MIDI information than you're used to, but hey, isn't that good practice?

Regarding Storage Space

Since your **.spc** has to contain samples, song data, sound driver info, and echo delay, all neatly compacted into **64KB**, storage space is going to be tight.

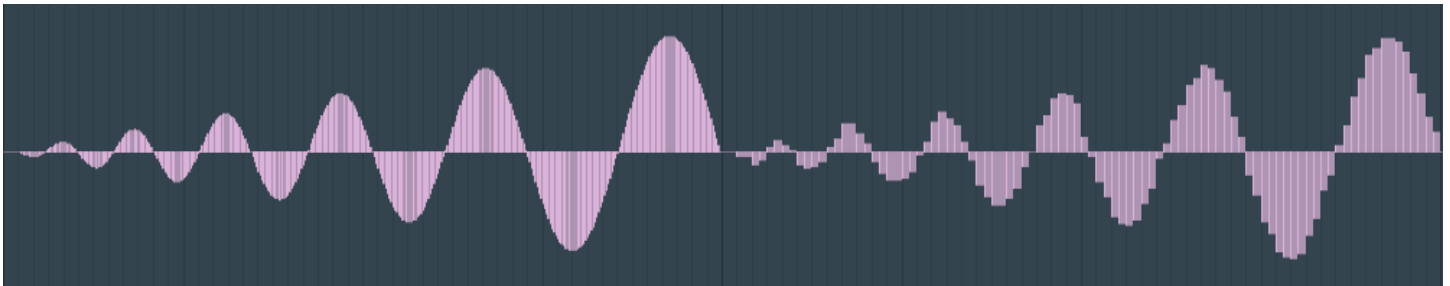
At the very bottom-right of C700, an indicator **12522bytes** is shown of how much memory is being consumed by **everything but song data**. Unfortunately there is no way for C700 to know how much data your song will consume until it tries. **Adding samples or increasing Delay Time** will increase memory consumption. The indicator turns **red** if you exceed the allotted space this way (and you may have to restart your DAW after resolving that issue; playback can be glitchy afterwards).

For making **.spc** specifically, it's best to ensure this value is under about **40000 bytes** (40KB), although depending on how much MIDI data is contained in your song, your mileage may vary. If using ROM export instead, you may get quite close to 64KB (it will turn red if you exceed maximum size).

Pitch Effects

Perhaps the single **easiest way** to bloat the memory of your song is with a **pitch effect** – things like vibrato, portamento, or pitch bends. This is because **every small change in pitch** is a piece of data that C700 is recording. That data adds up far quicker than you'd expect.

This makes **MIDI CC#1 (Modulation)** completely unusable on an **.spc** – pitch changes are written constantly, even during silence! You will instead have to program your vibratos using your MIDI channels' "channel pitch" control, as shown in the example below (FL Studio).



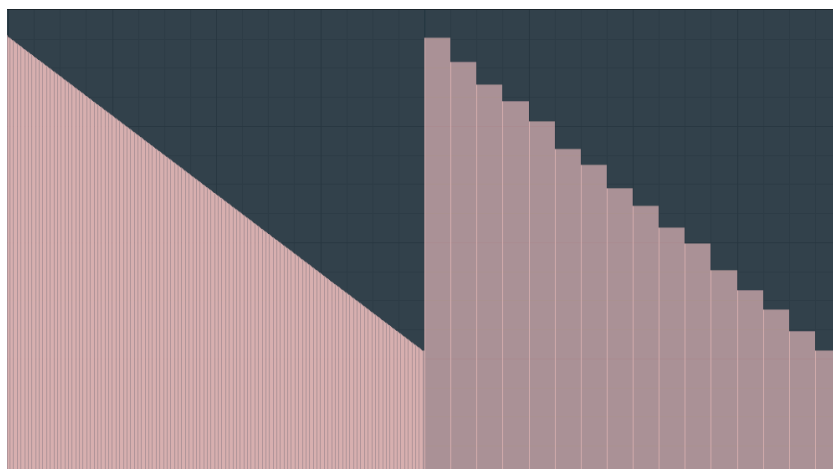
In fact, if you want to be using much vibrato at all, you will have to get very comfortable with MIDI channel pitch. Using **very fine pitch data** like the example on the left will quickly overload your song. You can **save significant space** by drawing over it **at a higher grid size**, such as in the example on the right, where it is snapped to $1/6$ step. There is virtually no perceptible difference at this level, but it is still relatively fine information (24 pieces of information per beat), and consumes a lot of data. Should you choose to go further and save more space, a vibrato fineness of $1/4$ beat is about as low as you can go while still getting a convincing vibrato sound.

Glide/Portamento is another dicey feature to utilize in **.spc**, but it can be done! You will have to use the **MIDI CCs** for Portamento: **CC#65 (Portamento Enable)**, **CC#5 (Portamento Rate)**, and optionally **CC#84 (Portamento Control)** which sets the note to glide from. If enabled on a sample globally, it will probably consume too much data over time, so using CC#65 to toggle the feature off and on selectively is recommended. The longer your selected Portamento Rate, the worse the additional data consumption will be.

If using **automation** to control any of these parameters, be sure there are **no fine curves** or slopes, and avoid unnecessary changes in parameters. This reduces the number of times C700 has to record new data.

Volume/Panning Fades

Similarly to pitch effects, gradual changes in volume or panning will quickly consume valuable song data, and are best implemented using **less fine event data or automation**, such as reducing a volume fadeout from a smooth slope to a slope with new points every 1/4 beat, as in the example below. (You can go finer if you want, but naturally more data will be consumed.)



Note that reducing this value much further may brush up against an **SPC700 chip limitation** which causes a small "crackle" or "pop" sound to occur when volume or panning shifts a large amount very suddenly. In this example, if a **note was playing** during the **immediate shift** halfway through, a "pop" would be heard. (This is true of **.smc** and other tools' exports as well.) In fact, a small "crackle" is always being produced during volume or panning fades, but it is only particularly noticeable during an immediate faraway shift.

This is useful for both **.spc** warriors and **.smc** mages:

[EXAMPLE 1](#) (sine wave) and [EXAMPLE 2](#) (square wave) of **a single held note across the above two bars of volume event data**. Note the "pop" sound that occurs halfway, and note the "crackle" sound behind the sine wave in particular (which has no overtones to mask the crackling). Both audio examples are normalized, and thus raised

to a level much greater than would be perceptible within your song, but this **quirk of the soundchip** is still significant. **Fades** of volume or panning will create slight crackling noise, and this noise becomes much more perceptible in the event of a huge, **immediate shift**.

Careful use of **envelopes** is preferred to using volume event data, for the most part. If only a few fadeouts are desired, you could automate **SR1** (MIDI CC#82) to occasionally switch to a different value instead. This would both save space and avoid the slight noise created by volume event data, if its presence bothers you.

Note that **no popping/crackling will occur** if the channel volume (or panning) shifts while **no note is playing**, so if you desire different volumes between sections, change the setting during a rest. Data will still be consumed even during silence, so don't use more event data than you have to. Much like with pitch effects, **don't leave any fine curves** or slopes in **automation** data. They will quickly consume a ton of space.

IV. How to use SNESMOD

First things first – if you need to install an .it tracker, I strongly recommend [OpenMPT](#). If you didn't already download SNESMOD, here are a couple places you can do that:

Recommended: [Newer fork of the original](#) which contains multiple options with varying feature sets. Includes the original, too.

Original SNESMOD (fewer features): <http://snes.mukunda.com/>

Whether you're using new or old, ample **documentation is included** in .txt files within your download. In the new fork, a dedicated folder named **/doc/** contains several text files covering not only a "manual" but also helpful tips and optimization info. Examples of some features in action are also provided in the **/examples/** folder. Given the presence of this fairly comprehensive documentation, this guide will mostly focus on general use and learning what to expect and what pitfalls to avoid. Even though you're converting an .it module, the full suite of .it commands is not utilized, and you will have to track in a somewhat specific way to get the best results.

[If you want to reference the full .it effect list, check out [this section of the OpenMPT manual](#).]

One disadvantage of SNESMOD over C700 is that **the resulting sound will be slightly different from what you hear in the tracker**. For example, you won't be able to hear the SNES's echo delay while you work, and ADSR envelopes may slightly differ from your settings. Other quirks, such as unwanted pops, may occur in the .spc and need ironing out, too. **Checking your work often is recommended** to ensure everything works as intended.

One option to get closer-to-correct sound while working is the free ["SnesDelay" VST](#). You can assign SnesDelay to **individual channels** or to the **Master** at the bottom of the **"General" tab in OpenMPT**. (For channels, use the dropdown "Effect:" menu below the volume/panning sliders; for the Master, checkmark "Master" in the "Mix Settings" at bottom-right, then select SnesDelay in the "Output to" dropdown.) This option is nowhere near perfect, but is potentially better than hearing no echo at all. **Be warned, the SnesDelay VST does not save your settings between sessions**, so if you tweak its default, you will have to do so again next time too. Saving a screenshot of your settings may be useful.

A. Using the Converter

The short answer: inside the SNESMOD .zip you downloaded are a series of **.bat files** that activate one of four different converter options. Opening any of these will prompt

you to select an **.it file** (it's easiest to copy your file into the same folder). Once you do, you hit Enter and an **.spc** will be created. Done!

Let's go over the options for conversion, which use different sound drivers:

- **Lite:** This is the original implementation! It is suitable for most needs, but lacks the robust feature suite needed to control many SPC700 features mid-song, such as echo delay parameters, ADSR envelopes, and several others. Notably it uses less pattern data than its brethren.
- **Pitchmod:** The next step up in terms of available features. This SNESMOD driver also allows the ability to toggle pitch modulation and noise.
- **SuperNoFX:** "Super SNESMOD" offers mid-song control of echo delay time, feedback and filter, along with a few other interesting features. It sacrifices support for sound effects, which may be relevant if you are creating music for a SNES game, but otherwise won't matter. Basically all its unique functions are accessed through Sxx and Zxx commands.
- **Celes:** The most fully-featured of all the SNESMOD drivers. In addition to the features of the others, Celes lets you alter ADSR envelopes mid-song and adds some dedicated effect commands for echo feedback control and faux filter sweeps. Similarly to SuperNoFX, it sacrifices support for sound effects, which will only matter if you're making music for a SNES game.

Detailed information on each of the SNESMOD drivers can be found in the **/doc/** folder. Each effect command's function and usage are listed in their respective **.txt files**. You can read more about the SPC700 features themselves above in the C700 VST guide.

One thing to note about the feature-rich converter options is that **you won't hear any of their effects** while writing your song in the tracker; you will have to run the converter and listen to the result. In the case of SuperNoFX and Celes, **many .it effect commands have been repurposed**, which may also make playback in the tracker more difficult to judge as you work. (For example, lxx controls echo feedback in Celes but makes a sound-off-and-on tremor effect in .it.)

Once you know which of the converters you want to use, run the **.bat** file for that converter. A window will pop up prompting you to enter the name of your file. Move or copy **your .it file into the same folder** as the .bat, type its filename into the prompt window, then hit Enter. Bam, your shiny new **.spc** will be created!

For original SNESMOD: Using the original converter is quite a bit simpler since there is only one option. Simply **drag your .it file onto "smconv.exe"** in the folder. Then you'll get your **.spc** file in the same folder as the **.it**!

Note: If you're comfortable with the **command line** interface, all the SNESMOD variants can be controlled that way, too. Running `smconv --help` (for SuperNoFX/Celes), `smconv-lite --help` (for Lite), or `pmconv --help` (for Pitchmod) will show you the additional options available during conversion; primarily these are intended for game development and the like. In the documentation, each of the tools is referred to as "smconv" so be sure to change references to that to "smconv-lite" or "pmconv" to actually point to Lite or Pitchmod.

The converter **.bat** files are easiest to use if you move or copy your **.it** into the same folder. You can simply type in the name of the file, e.g. *mysong.it*, and the converter will work its magic. You can also input an entire filepath instead, but make sure to put it in quotation marks, e.g. *"C:\CoolMusic\mysong2.it"*. Note that if you do this, the output .spc will be located in the *CoolMusic* folder rather than the SNESMOD folder.

If you run into errors, some common problems are not using Instrument mode (which is required) and incorrectly input echo delay info. Both of these will be covered in the guide so read on!

B. Setting Up Your .it Module

General Requirements

There are a few requirements for your .it module:

- No more than 8 channels
- Instrument mode **must** be used. No blank instruments (without samples)
- Mono samples only

- Song Properties: "Old Effects" set to off, "Linear Frequency Slides" set to on
- **OpenMPT 1.31 or newer:** In Setup -> Advanced tab, set `ITCompressionMono` to 0 (v1.31 changed the default value to 7, adding compression that breaks SNESMOD)

Despite the requirement to use Instruments, many features of .it Instruments are not supported. With SNESMOD, the primary function of Instruments is to set the ADSR volume envelope. Here are the restrictions:

- No **filter** or filter envelopes
- No **panning envelopes** or **pitch envelopes**
- **Envelope sustain** is possible, but must be confined to **one node** only (that is, the sustain is one static point only)
 - No envelope **carry**
 - Envelope **nodes** must be no more than **256 ticks apart** (this is quite a lot though)
 - **NO:** New Note Actions, sample map with multiple samples, random pitch/panning variation, auto-vibrato (found in the Sample tab), pitch/pan separation
- Echo Delay settings are set via Song Comments

Echo Delay

Wait, what did you just say about Song Comments?

One of the quirkiest parts of SNESMOD is that you place a bunch of values into the **comments section of the .it module** ("Song Message") in order to **set up the echo delay**. In OpenMPT, this is located in the rightmost tab "Comments", after Samples and Instruments. Since you will have to run the converter to hear how your delay sounds in the .spc song, setting this up may take some trial and error.

⚠ Warning! ⚠ Modifying the echo delay settings, particularly the feedback (EFB) and filter (EFIR), **may create a feedback loop** which can cause hearing damage. **Be extremely careful** with the volume, and be prepared to mute or stop the sound in the event you hear a feedback loop starting (identifiable by the echo getting louder over time).

The following is an example of a SNESMOD delay setup comment, which you may copy and paste into your own module's Song Message section:

```
[[SNESMOD]]
edl 6
efb 50
evol 31 -31
efir 127 0 0 0 0 0 0 0
eon 1 2 3
```

So what do those five commands mean? Let's dig in – if you already read about echo delay in C700, you will be familiar with these concepts.

EDL: Delay Time, 0 through 15. This number **multiplied by 16** will tell you the **number of milliseconds until the delay is heard**. In our example, Delay Time is **96ms**; the maximum value of 15 will produce a Delay Time of 240ms. Note, however, that **additional memory is consumed** at a rate of 2KB per +1 increase, meaning that the maximum value takes up 30KB of the available 64KB.

EFB: Delay Feedback, -128 through 127. **Be very careful with large values here**, as you do not want to cause a feedback loop. This value determines the volume of sound to "feed back" into the delay to keep it going. In practice, **the higher the value, the longer the delay will keep occurring**. Negative values invert the phase of the waveform, slightly altering the sound.

EVOL: Echo Volume, -128 through 127 (2x). The left and right stereo may be controlled individually here, with a popular choice being *one positive, one negative* (as with the C700 default). This creates a pseudo-surround sound effect in stereo, though the echo wouldn't be heard at all if played back in mono. Regardless, this is a fairly straightforward effect controlling **the overall volume of the echo delay**. Higher values increase the echo volume, and negative values invert the phase of the waveform. If only one value is given, it is applied to both left and right stereo. In our example, the echo volume is set rather low (31 -31), around 25%.

EFIR: Echo Filter Coefficients, -128 through 127 (8x). ⚠ **Be careful! It is very easy to create a feedback loop with this setting! Don't damage your hearing!** ⚠ This setting controls the **filter** applied to the echo delay, and the **prominence of various frequencies within the delay**. Up to 8 values can be input here (if fewer, the remainder will be assigned a value of 0). The collective functions of the values are rather arcane; generally the later values will be more likely to blow out the sound because each value is fed into the next via some sort of multiplication math. **It is best not to use very many large values to avoid feedback loops.** If the sum of all these values is very positive or very negative (above ~150 in either direction) it is more likely to cause a feedback loop. The given example (127 0 0 0 0 0 0 0) is a very open sound without reduced influence of any frequency range – it is the same as the C700 default.

EON: Enable echo for these channels, 1-8 (up to 8x). The **channels** you list here will **begin the song with echo delay enabled**. In our example above, only channels 1-3 start with echo delay enabled. This can be modified within your song using the SNESMOD-specific effect commands **S01** (echo on for channel) and **S02** (echo off for channel). If you wanted to start your song with echo enabled in every channel, you would put "eon 1 2 3 4 5 6 7 8". Additionally, echo delay can be enabled/disabled globally using **S03/S04** respectively.

Effects Column support

Many of the .it effect column commands are supported, and depending on which converter variation you are using, some of the unsupported commands are repurposed for SNES-specific features. We'll go over which .it commands work and which don't.

Here are the things you **can** do:

- The main pitch effects are fine. **Exy/Fxy** (pitch slide down/up), **Gxy** (portamento), and **Hxy** (vibrato) all work as intended, and unlike with C700, they don't take up excessive space. Use them to your heart's content in the effects column.
- Most pattern-playback effects are fine. **Bxx** and **Cxx** (pattern jump and pattern break) work just fine (*though the Celes converter option repurposes Cxx*). **Axx**

(song speed) and **Txy** (tempo) are fine. However, there are a few unusable pattern-related commands as well (**SBx**, **SEx**, **S6x**).

- Most volume effects are supported too. **Dxy** (volume slide) as well as **Kxy** (volume slide + vibrato) work fine. The channel volume commands **Mxx** (set) and **Nxx** (slide) work (*although the Celes driver repurposes them instead*), as do the global volume commands **Vxy** (set) and **Wxy** (slide).
- Panning (**Xxx** or the less-fine **S8x**) and panning slide (**Pxy**) are supported.
- The arpeggio command (**Jxy**) is supported as well.
- Note retrigger (**Qxy**) is supported! Thank goodness!
- Crucially, **SCx** (note cut after x ticks) and **SDx** (note delay by x ticks) both work as intended. **SCx** is unfortunately an essential command for clean tracking with SNESMOD.

Now for the things you **can't** do:

- Despite **Kxy** being functional, **Lxy** (volume slide + portamento) does not work. Neither does **Uxy** (fine vibrato).
- **SBx** (pattern loop), **SEx** (pattern row delay), and **S6x** (pattern tick delay) all do not work, though their effects can be achieved manually.
- Most of the commands beginning with **S** do not work, with the exception of **S8x** as an option to control panning, varying implementations of vibrato waveform control, and the essential **SCx** and **SDx**. The rest are not only unsupported, but many of them have been repurposed by SNESMOD for a SNES-specific feature.
- Tremolo (**Rxy**) and tremor (**Ixy**) both do not work. (*Tremor is repurposed in Celes.*)
- **Oxx** (sample start offset) is unfortunately not supported. (*Celes repurposes it.*)
- "Panbrello" (**Yxy**, the oscillator-controlled panning slide) is not supported either. (*Likewise, it's repurposed in the Celes driver.*)

Volume Column support

In the **volume column**, the only unsupported effects are the pitch commands: **Ex**, **Fx**, **Gx** and **Hx** (which would correspond to their effect column counterparts).



The panning command **Px** works fine, as do the volume command and volume slides: **Vx** (volume), **Cx/Dx** (volume slide up/down), **Ax/Bx** (fine volume slide up/down, which applies only on the first tick of each row).

C. SNESMOD Tracking Tips

If you'd like to read someone else's words on this subject, check out a .txt file in your **/doc/** folder named "snesmod optimisation&tips.txt". This person has also compiled a lot of very useful information on SNESMOD and handling samples, some of which I will cover here and below in the Samples section.

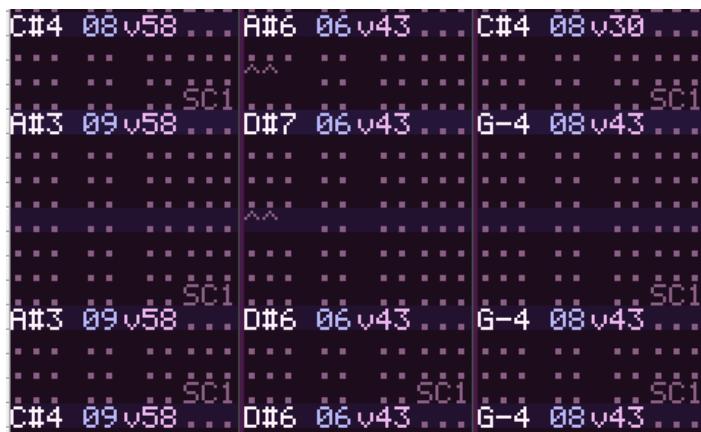
Tip #1: Note Cuts

You are destined to experience clicking noises between notes in SNESMOD. Really. You will be surprised. *Usually*, this occurs when a new note is triggered without the previous note being cut off. Think of your basic melody situation where each note flows into the next. In very many situations, a click will be produced in between those notes.

You could cut off the note the row before, but that might not be the sound you want. Enter our new best friend, **SCx** (Note Cut after **x** Ticks). This also might not be the sound you want, but unfortunately that's as good as it's going to get (sorry!). The idea here is, instead of cutting off the note the row before using  (leaving you with a whole row of silence), **an SCx command is placed the row before the new note**, with **x = song speed - 1**. For example, if your song is at speed 6 (that is, 6 ticks per row), you'll be using . In the end, one tick of silence is created before the new note. Hopefully this silence is negligible to you; SCx should be your first line of fire if you experience clicks between notes in your converted .spc.

You may prefer to track your song first – the wait and see method – without any SCx and identify where you hear clicks. Probably some instruments will have them, while others won't. You could also preventatively add them as you track; it can be a bit of a pain to comb back through an entire module to add these commands. On the other hand, you may want to compromise the sound as little as possible. It's up to you, but

this is bound to be an important part of your SNESMOD journey if you want to avoid the clicking noises.



Example: This song is at speed 2, so `SC1` commands are placed before new notes to avoid clicking sounds.

One interesting quirk about the note-cut clicks is that echo delay applies to them, too. This may make their presence more noticeable, depending on how much echo delay you're using.

Tip #2: Rapid Volume/Panning Changes

Another likely culprit for popping noises comes from **quick changes in volume or panning** – especially hard shifts from faraway values, but "fades" can cause this too, particularly abrupt ones. These sound more like "**crackling**" than the note-cut clicks, and are not quite as loud either. However, you may notice them from time to time.

Note: You are **far more likely** to encounter this problem if you're using a **fast song speed**, like speed 1 or 2.

Though there is no cure-all for this problem, a few steps can be taken:

- If possible, keep volume fades confined to your samples' **ADSR envelopes**, rather than using the volume command (Vx). Keep in mind you can create **multiple Instruments** with different ADSR envelopes for any of your samples.
- Relatedly, using the **Note Off ==** command will **release** your ADSR envelope if you include a **sustain point**, so this can be used to taper notes' volume as well.

- If you must use several volume or panning commands in a row, set their values as **close together** as possible. Avoid sharp, sudden changes if you can.
- If going for rapid panning changes, make the shifts occur **during silence** if possible ("prepare" for the next note).
- If at fast speeds, the sound may be less crackly if you only put volume/panning commands every other line. This may not be feasible for quicker fades; however it may be worth testing if you are having trouble. Differences of ~4 or greater seem to be problematic at high speeds.

Notably, this is not a SNESMOD-exclusive problem. This is just how the SPC700 soundchip handles this info – imperfectly. This issue exists with the C700 VST too, and again is why I recommend using envelopes as much as you can. It may even save space! Speaking of which...

Tip #3: Space-Saving

You may want to push SNESMOD to the limits and fit as much into 64KB as you can! A noble goal – in that case, you should familiarize yourself with a few space-saving tips for .it modules. These will also save space if you ever need to trim the size of any other .it module!

- **Effect Continues:** Instead of using the 00 "continue" command for effects, just repeat the effect; each instance of this saves a tiny bit of space. For example, a vibrato over the course of several rows would read "H55 H55 H55 H55" instead of "H55 H00 H00 H00". This is, surprisingly, more efficient.
- **Volume commands:** Volume commands do take up space, which can add up over time. Consider setting the "Global Volume" of your Instrument to something other than 64 (maximum) if you're frequently using it at a lower volume level.
- **Repeated effects:** Repeated uses of the same command in a channel are compressed further in the module. In addition to saving space on effects, this is another way you can potentially save space with volume commands. If you are alternating between two volumes several times, it is more space-efficient to separate

them into two channels – for example, one channel at volume 64 and the other at volume 32.

- **Blank rows:** You may be able to halve the number of rows and double the speed value of your song to save space, if every other row is blank. If it's close, you may be able to use the SDx command (Note Delay by X Ticks) to move a few things and attempt this.
- **Blank rows part 2:** Blank rows can also be trimmed to save space if you skipped over them with Bxx or Cxx.
- **Duplicate pattern sections:** If multiple patterns contain sections of duplicate data, the patterns can be split and the duplicate data included in the pattern order multiple times. For example: if patterns 0 and 1 both began with the same 16 rows, they could be split up, with pattern 2 containing the duplicate data, and the pattern order reading "2 0 2 1" instead.
- **Channel muting and unmuting:** If you have a section of music that is repeated verbatim with new material on top (such as a new melodic line or bass), you can use the **Mxx** command (channel volume) to "mute" and later "unmute" the channel(s) containing the new material, effectively cutting your pattern data consumption **in half**. For example: Set `M00` before this section so it's muted on the first passthrough, then at the end of the pattern, use another Mxx command (`M40` is maximum) to set the new volume for the second passthrough. (Note: Mxx is repurposed for something else in the Celes driver specifically, so this will not be helpful if you use that converter option.)

One feature of the converter is that it tells you the details of how much space is taken up by the various parts of your song, as well as how much free space is left, after creating your .spc. If you're looking to trim the file down, or if you've already passed the 64KB limit, these metrics will no doubt be helpful to you.

Tip #4: Envelopes

As mentioned before, it is helpful to use the ADSR volume envelopes of your Instruments to control volume fades, rather than using volume commands in the patterns. They will most likely sound cleaner overall. If you have never used this feature in OpenMPT, head to the Instruments tab and click the word "**Vol**" above the graph to

enable the volume envelope – it's next to Pan, Pitch and Filter, the rest of which are not supported by SNESMOD. Immediately to the left of Vol, make sure you've selected the leftmost option "Show Volume Envelope" (this is default).

To the right of Vol and friends are three envelope features: **loop**, **sustain**, and carry. Carry is unsupported for SNESMOD, but clicking on the other two will enable a full envelope loop and a sustain point, respectively. With SNESMOD, **the sustain point must be on only one node**. This is the point that the volume envelope holds until the note is released.

Double-click in the graph area to add a new node point. The lower the point, the lower the volume. Add points and drag them around until you've gotten the desired sound; the sample can be tested here with your keyboard or a MIDI controller.

With SNESMOD, nodes must be within 256 ticks of one another – tick numbers are displayed in the bottom-right of the screen.

Additional Instruments using the same sample can be created with different volume envelopes; this can be worthwhile even if you only use them once.

SNESMOD is going to be **approximating** your envelope to the best of its ability using the values you may remember from C700's volume envelope settings, so it's recommended to run the converter and check how everything sounds, too. (Attack only has 16 different values, Decay and Sustain have 8 each, Release has 32 and the special "note length" release function also has 32 values.)

Tip #5: Controlling SPC700 Features

It's best to consult with the included documentation to see the full array of SNESMOD-specific effects, particularly since several versions of the converter exist that do different things:

- **/doc/snesmod_and_pitchmod_doc.txt** covers Lite and Pitchmod. All these features are available in Pitchmod, but only some are available in Lite, as

indicated by whether or not the word "SNESMod" is written in the header above the effect. If it's unsupported in Lite, that space will be blank instead.

- **/doc/supernofx_doc.txt** covers the SuperNoFX driver. Pretty much everything from "Echo Write Flag" onward is a new feature compared to Lite and Pitchmod. Bear in mind some features utilize existing .it effect commands that are otherwise unsupported by SNESMOD, but they are relegated to Sxx and Zxx.
- **/doc/celes_doc.txt** covers the Celes driver. The layout is slightly different so its new features (compared to Lite and Pitchmod) are scattered throughout. Many more .it commands are repurposed in Celes. **Both Celes and SuperNoFX** have a multipurpose use of the **Zxx** command that is defined by a "ZMODE". Which mode you're using Zxx for in a channel is set via **S9x** commands.

SuperNoFX and **Celes** offer the widest suite of features to control, but **Pitchmod** also includes several features, including on/off toggles for echo delay (S01-04, also available in Lite), pitch modulation (S05-08; S92-S93), noise generation (S09-S0C; S1x-S2x), some ADSR control (M7F-MFF), and vibrato waveform choice (S30-S36). Read more about their implementation in the documentation mentioned above.

The most significant additional offering of **SuperNoFX** is the ability to **control echo delay parameters within the song**, i.e. changing delay time, feedback level, and the values that define the "FIR filter" (which emphasizes different frequency ranges), including the ability to sweep the values with specific commands. **Celes** offers a somewhat different, even more thorough implementation of echo delay control as well as the ability to set ADSR envelopes for a given channel... and a bunch of other stuff you may find useful!

One interesting feature of SuperNoFX and Celes which I have not personally used is the ability to enable "**wavetable**" synthesis on a channel, cycling through up to 8 samples without re-attacking the note. This was a little-used SNES feature that allowed for things like crude PWM (pulse-width modulation) or faux filter sweeps. Celes offers a similar FM synthesis option as well, though I have also not utilized it, and unlike the wavetable feature, no example is given in **/examples/**.

Notably, if you're making music for a SNES game, the SuperNoFX and Celes drivers remove support for sound effects in exchange for their wealth of control options. This won't matter unless you're making music for a game, in which case it may matter quite a lot.

V. Dealing with SNES Samples

All things considered, the Super Nintendo's sampler is pretty quirky. Both C700 and SNESMOD try their best to do the proper conversion work for you, but this isn't always perfect, and you may find yourself wanting to learn the specifics of the sampler and fix up sounds that aren't working right, or that may be taking up unnecessary space. Or maybe you want to get started with some existing samples; look no further!

Here we'll cover three things:

- A) How to **rip samples from .spc** files (including game soundtracks!)
- B) How to **prepare other, non-SNES samples** for use on the SNES
- C) A **repository of ripped samples** from games

A. Ripping Samples from .spc Files

In order to rip samples from existing .spc files, we'll need to use a tool called **split700n**, which can be downloaded here: [here \(Discord link\)](#) or [here \(GDrive link\)](#).

We'll also need some .spc files to start with. Two amazing resources, [Zophar's Domain](#) and [SNESMusic](#), exist online to download just about any SNES game's music in **either .spc or .mp3 format**. [This webpage](#) also contains a ton of .spc soundtrack rips. Grab some .spc files from your favorite game, perhaps Super Mario Kart?, and let's get started!

The tool itself is fairly straightforward to use. Simply place any number of .spc files in the **/bin/** folder, then run **"extract.bat"** in the same folder. When you do, a command prompt window will appear and ask you to press any key to continue. **Every sample from every .spc** will be ripped and placed into this folder.


If you're ripping an entire soundtrack, you will wind up with very, very many **.brr** files – several hundred or more, all numbered and named after the .spc they were ripped from. **Most of these are duplicates**; in many SNES games, every .spc file will contain the same samples. *However*, there are also quite a few games that dynamically allocated their samples, meaning **some samples are only located in specific .spc songs**. If you want to be thorough, you will want to include every .spc from a soundtrack to rip all samples from that game. On the other hand, if you know you want a sample from a specific tune, you can rip it easily.

If you're ripping many files in a soundtrack, you will want to use a program to identify and clean out duplicates quickly. Two free options are [Fast Duplicate File Finder](#)

(recommended) and the duplicate finder within [CCleaner](#) (may require multiple scans). Simply scan the **/bin/** folder for duplicates and delete them.

For dealing with .brr files, you have a few options – the standalone tool [BRRPlay](#) can be used to open and listen to .brr samples. C700 can also load them, as can OpenMPT (choose "16-bit signed" settings when importing; see below if you end up with noise). It is possible to get right to work on your music from here.

Additional processing may be desired though – you may want to make **modifications** to the samples, such as **tuning them**, for which **OpenMPT is highly recommended**. Not all games have their samples all tuned to the same note, and even fewer are tuned to the correct one. The rightmost icon in OpenMPT's sampler (a tuning fork) will automatically tune the sample to a chosen note. You can also set a keyboard shortcut for this by going to View > Setup > Keyboard > "Tune Sample to Given Note". Tuning has to be done sample-by-sample, but since the process is otherwise automatic it is still a fairly quick process to tune all samples.

You also may want to **convert the .brr files to .wav** – two options exist here as well. OpenMPT can "**Save All**" samples at once in the Samples tab; click the arrow next to the Save  icon to access this function. Set the filetype to .wav (or .flac if desired, but .wav is more widely supported by samplers), then format the filenames as you like. *It's recommended to use %sample_name%.wav instead of %sample_filename%.wav because the "Name" field can be longer. If desired, you can set useful Names for your samples in OpenMPT like "flute" or "sinewave".*

Also **included with split700n** is a tool named **brr2wav.exe**; this can convert your new .brr files to .wav as well. Drag any number of .brr files onto the .exe file and it'll do its magic!

Both tools may be useful from time to time. Occasionally brr2wav gets sample loop points wrong, but also occasionally, OpenMPT fails to properly import some soundtracks' .brr rips, instead creating lots of garbled noisy samples (examples: *Plok*, *Ken Griffey Jr.'s Winning Run*). At times you will need one or the other.

Additional Tips and Quirks of .spc Ripping

- If you want to give **readable names** to your samples, try using the **Name field in OpenMPT** as described above. When you hit Save All (via the Save icon's dropdown arrow), use `%sample_name%.wav` instead of `%sample_filename%.wav` and your files will be saved using the Names you set.
- In addition to exact duplicates of samples, you will frequently end up with **near-duplicates**, or samples that are **truncated versions of other samples**. For example, you may have a snare sample as *well as* another sample that contains only the tail end of that same snare sound. There may also be duplicates where the loop points of these samples will be obviously incorrect. **These are not useful and are best discarded**. Because of this, if you're sorting through a bunch of samples, it may be useful to sort them by *largest filesize* – that way you run across the "correct" versions first by looking at the largest samples.
- Most games will have a handful of small, jagged-looking "samples" that produce noisy, buzzy synth waveforms if looped (these don't have embedded loop points most of the time). These are not actually samples from the game; in fact I'm not sure what they are, but possibly they are from envelopes of some kind. Most likely, **these can be discarded too**, but if you like how they sound there's no harm in using them either.

Condensed Step-by-Step Guide

Here is a condensed step-by-step guide to use as a checklist for ripping an entire soundtrack to **.wav** samples:

- 1) Place all .spc files into the **/bin/** folder where split700.exe and extract.bat are.
- 2) Open **extract.bat**, which rips the .brr files.
- 3) Use a duplicate file finder (such as [Fast Duplicate File Finder](#)) to **delete copies**.

- 4) (Optional) Select all .brrs and drag them onto brr2wav.exe to convert to .wav.
- 5) **Sort by filesize**, largest first.
- 6) **Load all samples into OpenMPT** (whether .brr or .wav) by selecting them all and dragging them onto the huge waveform area in the "Samples" tab. Choose 16-bit signed import settings.
- 7) **Change the "Name" field** to something more useful for each sample.
- 8) Weed out any partial duplicates and other **useless samples** by hand.
- 9) **Tune all samples** you wish (likely all but drums & SFX). Either click the tuning fork icon (the rightmost one in the row) or set a shortcut in OpenMPT (View > Setup > Keyboard > Tune Sample to Given Note) to automatically tune a sample.
- 10) Click the arrow next to the Save icon in the Samples tab and choose **"Save All..."**. Recommended file name format: %sample_name%.wav. This pulls from the "Name" field instead (which you may have filled in step 7).

B. Preparing & Optimizing Samples

You may wish to use samples that aren't ripped from SNES games/songs already! You can totally do this; however there are a few quirks you must adhere to, as well as some things you can do to save space.

Sample Rate

The SNES's sampler caps at a rate of **32000 Hz**, while most samples use the CD-quality format of 44100 Hz or similarly-sized 48000 Hz. You can comfortably downsample your sound to 32000 Hz because it'll never get higher quality than that anyway.

(Downsampling can be performed easily in OpenMPT *[Ctrl+R in Samples tab]* or Audacity *["Project Rate" in lower-left corner].*)

You can downsample as much as you want, but quality will be lost unusually quick thanks to the SNES's global "Gaussian filter", which blunts a lot of the highest frequencies already in an attempt to cover up the noise created by its compressed sample format (.brr, which reduces each 32 bytes to only 9).

C700 will do this for you, but you may want to perform this yourself with SNESMOD to get a more accurate sound while you work.

Tip: Rather than continually downsampling to get what you want, first undo any existing downsampling to avoid extra loss in sound quality when you execute the downsampling.

Loop Points

One very significant quirk of the SNES sampler is its handling of loop points:

- 1) **The loop must start *and* end on a multiple of 16 samples** (the tiny unit that pieces of audio are composed of). That means a loop can be 16 samples long at shortest, followed by 32, 48, and so on.
- 2) **No data after the loop point ends.** (This would be a waste of space anyway!) In C700, any existing data after the loop point will be automatically truncated.

The first point is the one that may prove challenging. **Both C700 and SNESMOD will automatically attempt to resample** your sound in order to fit the loop point within a multiple of 16. With C700 you'll be able to hear the result right away, but with SNESMOD you will need to actually run the converter.

Slightly **resampling your sound yourself** (reducing its size proportionally, so thus also reducing the loop point's size proportionally) is one option at your disposal for getting the loop point sized as a multiple of 16. OpenMPT's sampler is a good tool for this.

Unless you do some math, this may be a **trial-and-error** process: resample, check loop length, undo resampling if not divisible by 16, repeat.

However if you feel like doing some math, you could also check how many samples fewer the loop has to be to follow the "16 rule" and then proportionally reduce the size of the sample by just enough so that the loop is reduced to the proper size too. For example: Your sound has 200 samples in it and loops from 100-200. The loop size, 100, isn't divisible by 16; the next functional size down would be 96. That's a difference of 4 samples, and the loop comprises half the sound, so if your sound is resampled to have 8 samples less, the loop will be the correct size.

If your sample's **loop is fairly simple** (like a cycle of a waveform), you may be able to **include additional cycles within the boundary of the loop** to achieve a number divisible by 16. For example: You have a loop that's 20 samples long; one cycle of the wave passes in 20 samples. If you incorporate 4 cycles instead, that's 80 samples, which is divisible by 16.

Once your loop point is divisible by 16, you can offset it by a few samples if necessary (adding silence to the start) so that the loop start & loop end are also multiples of 16.

That's it! Again, this may take a good bit of trial and error unless you are keen on math. Both programs attempt to do this for you, but it's possible that you are unhappy with the resampling, or that you end up wasting some storage space by your loop being "unrolled" (another option – simply adding more instances of the loop until its length is divisible by 16).

.brr Format

Although **neither C700 nor SNESMOD require** that you use the actual .brr format, you may for some reason want to convert your .wav samples yourself. For that, you can download [wav2brr](#), one of a few tools in the linked program package. It includes its own tutorial and notably requires [Python](#) (coding language) installed.

C700 can also **save samples in .brr format** if you click "**Save Smpl...**" below the waveform indicator in the middle of the plugin.

.BRR is notable for reducing the size of a sample by over two-thirds – a ratio of 9 to 32 bytes! The compressed samples are a little noisier, which the SNES attempts to accommodate for using its "global Gaussian filter". The result is a big part of what gives the SNES its unique and easily identifiable sound quality.

Pre-Emphasis

In an attempt to counteract the "flatness" of the upper frequencies caused by the aforementioned global filter, samples can be "pre-emphasized" – essentially EQing the high frequencies to have greater presence. **C700 does this automatically**, and SNESMOD comes with **preemp.py**, a Python script that can pre-emphasize samples for you, though it too requires [Python](#) (coding language) installed. If you use this, lower your samples' volumes (amplitudes) before pre-emphasizing in order to keep them from clipping in the end. (Try reducing to ~60%, but your mileage may vary.)

By default C700 has pre-emphasis enabled, and it is a global setting, meaning it will either be On or Off for *all* loaded samples.

C. Repository of SNES Sample Rips

Here is a collection of **labeled .wav sample rips from video games**. With a few exceptions, most of these have been ripped by yours truly over the last several years; sounds were labeled to the best of my ability. These packs should also have the pitched instruments **tuned** to C. If you spot a batch or a sample that isn't tuned, let me know and I'll replace it.

This list will be updated approximately once per week with a new game, because I host a weekly one-hour compo on [Battle of the Bits](#) named "YEA VIDEOGAME" that uses a different SNES game's sample pack each time.

ActRasier	Equinox
ActRaiser 2	E.V.O.: Search for Eden
Arkanoid: Doh It Again	F-Zero
Asameshimae Nyanko	Faceball 2000
Axelay	Fatal Fury
Bahamut Lagoon	Final Fantasy IV
Barkley Shut Up and Jam!	Final Fantasy V
Batman Forever	Final Fantasy VI [v2, tuning fix]
Battletoads & Double Dragon	Final Fantasy Mystic Quest
Bebe's Kids	Fire Emblem: Genealogy of the Holy War
Bishoujo Janshi Suchie-Pai	Fire Emblem: Monshou no Nazo
Brain Lord	Fire Emblem: Thracia 776
Breath of Fire	Frantic Flea
Breath of Fire 2	Front Mission
Brett Hull Hockey '95	Front Mission: Gun Hazard
Bubsy in Claws Encounters of the Furred Kind	Gradius III
Bubsy II	Gundam Wing: Endless Duel
Bust-A-Move	Harvest Moon
Castlevania: Dracula X	HyperZone
Chrono Trigger	Illusion of Gaia
Claymates	International Superstar Soccer
Contra III: The Alien Wars	J.R.R. Tolkien's The Lord of the Rings, Vol. I
Cool Spot	Jeopardy!
Cybernator	Jikkyou Powerful Pro Yakyuu 3
Darius Twin	Joe & Mac: Caveman Ninja
Daze Before Christmas	Joe & Mac 2: Lost in the Tropics
Disney's Aladdin	Jojo's Bizarre Adventure
Dolucky no Puzzle Tour '94	Jurassic Park
Donald Duck: Mahou no Boushi	Jurassic Park Part 2: The Chaos Continues
Donkey Kong Country	Ken Griffey Jr. Presents: Major League Baseball
Donkey Kong Country 2	Ken Griffey Jr.'s Winning Run
Donkey Kong Country 3	Killer Instinct
Doom [v2, missing samples fix]	Kirby no Kirakira Kids (Kirby's Star Stacker)
Dragon Ball Z: Hyper Dimension	Kirby Super Star
Dragon Quest III	Kirby's Avalanche
Dragon Quest V	Kirby's Dream Course
Dragon Quest VI	Kirby's Dream Land 3
Earth Defense Force	Legend of the Mystical Ninja
Earthbound [v2, tuning fix]	Lemmings
Earthworm Jim	Live A Live
Earthworm Jim 2	Lufia & The Fortress of Doom

[Lufia II: Rise of the Sinistrals](#)
[Majin Tensei II: Spiral Nemesis](#)
[Mario Is Missing](#)
[Mario Paint](#)
[Mario's Super Picross](#)
[Mario's Time Machine](#)
[Mega Man 7](#) (not my rip)
[Mega Man Soccer](#)
[Mega Man X](#)
[Mega Man X2](#)
[Mega Man X3](#)
[Megaman & Bass](#)
[Monopoly](#)
[Mortal Kombat](#)
[Mortal Kombat II](#)
[Mortal Kombat 3](#)
[Mr. Nutz](#)
[Ms. Pac-Man](#)
[Nobunaga's Ambition](#)
[Ogre Battle: The March of the Black Queen](#)
[Operation Logic Bomb](#)
[Panel de Pon \(Tetris Attack\)](#)
[Pilotwings](#)
[Pitfall: The Mayan Adventure](#)
[Plok](#) [v2, tuning fix]
[Pop'n TwinBee](#) [v2, missing samples fix]
[Psycho Dream](#)
[Q*bert 3](#)
[R-Type III](#)
[Race Drivin'](#)
[Ranma ½: Hard Battle](#)
[Romancing SaGa](#)
[Romancing SaGa 2](#)
[Romancing SaGa 3](#)
[Rudra no Hihou \(Rudra's Treasure\)](#)
[Run Saber](#)
[Secret of Evermore](#)
[Secret of Mana](#)
[Seiken Densetsu 3 \(Trials of Mana\)](#)
[Shadowrun](#)
[Shaq Fu](#)
[Shin Megami Tensei](#)
[Sid Meier's Civilization](#)
[Side Pocket](#)
[SimCity](#)
[Space Invaders](#)
[Spiderman and the X-Men in Arcade's Revenge](#)

[Spindizzy Worlds](#)
[Star Fox](#)
[Star Ocean](#)
[Street Fighter Alpha 2](#)
[Street Fighter II](#)
[Stunt Race FX](#)
[Sunset Riders](#)
[Super Adventure Island](#)
[Super Adventure Island II](#)
[Super Back to the Future II](#)
[Super Bomberman](#)
[Super Bomberman 2](#)
[Super Bomberman 3](#)
[Super Bomberman 4](#)
[Super Bomberman 5](#)
[Super Castlevania IV](#)
[Super Drift Out](#)
[Super F-1 Hero](#)
[Super Mario Kart](#)
[Super Mario RPG: Legend of the Seven Stars](#)
[Super Mario World](#)
[Super Mario World 2: Yoshi's Island](#)
[Super Metroid](#) (not my rip)
[Super Punch-Out!!](#)
[Super Puyo Puyo 2](#)
[Super Scope 6](#)
[Syvalion](#)
[Tactics Ogre: Let Us Cling Together](#)
[Tales of Phantasia](#)
[Taz-Mania](#)
[Teenage Mutant Ninja Turtles IV: Turtles in Time](#)
[Terranigma](#)
[Tetris & Dr. Mario](#)
[Tetris 2](#)
[The Lion King](#)
[The Shinri Game 2: Magical Trip](#)
[Time Trax](#)
[Tiny Toon Adventures: Buster Busts Loose!](#)
[Top Gear](#)
[Top Gear 2](#)
[Treasure Hunter G](#)
[Uniracers](#)
[Vegas Stakes](#)
[Wagyan Paradise](#)
[Waterworld](#)
[Wild Guns](#)
[Wizardry Gaiden IV: Throb of the Demon's Heart](#)

[Wizardry I-II-III: The Story of Llylgamyn](#)
[Wizardry V: Heart of the Maelstrom](#)
[Wizardry VI: Bane of the Cosmic Forge](#)
[Wonder Project J](#)
[Yoshi's Cookie](#)

[Yoshi's Safari](#)
[Ys III: Wanderers from Ys](#)
[Ys IV: Mask of the Sun](#)
[Ys V: Lost Kefin, Kingdom of Sand](#)
[Zelda III: A Link to the Past](#)
[Zombies Ate My Neighbors](#)

Credits & Thanks

- C700 VST was created by **osoumen**: [GitHub](#) | [SoundCloud](#) | [Twitter](#)
- C700 Microtunable was forked by **doctorn0gloff**: [SoundCloud](#) | [Twitter](#)
- SNESMOD was created by **mukunda**: [Website](#)
- Work on the newer SNESMOD fork was done by **Augustus Blackheart** ([Website](#)) and **KungFuFurby** ([GitHub](#) | [Twitter](#))
- SPCPlay was created by **dgrfactory**: [GitHub](#) | [Twitter](#) | [Website](#)
- Split700 was created by **gocha**: [GitHub](#) | [Twitter](#)
- Thank you to **DEFENSE MECHANISM** for compiling a working executable for the newer SNESMOD drivers, and to the unknown person who compiled the others.
- Special thanks to **those who proofread and/or beta-tested** this guide and offered advice and information, both pre- and post-release.
- This guide was written by **damifortune (HVB)**: [SoundCloud](#) | [Bandcamp](#) | [Twitter](#)