# Introduction

Version 7 features significant noise reduction, due to correcting the mapping of MIDI note velocity to Tone amplitude. Plus it introduces a new option to model string sound in a more natural way by having non-integer based harmonics, using a novel algorithm and carefully calculated lookups.

The source code to Version 7 is available and has been significantly clarified and streamlined. If you have the ability to load version 7 onto the Tone Processors then either load the hex file or compile and load (but you’d need the MPLABX Pro licence to get the compiler optimisation required).

Although the Windows app has been altered to use the additional inhamonic option, you don’t have to load version 7.

# MIDI velocity to tone amplitude

The MIDI standard does not specify how the note velocity value (in the range 0 to 127) should relate to instrument 'loudness' in dB. Manufacturers of MIDI gear vary in their interpretation.

Pre-version 7 Tone Processor firmware (from the original Silicon Chip article June 2022) was my initial attempt and was a little naïve, because the lowest velocities were too soft – i.e the dynamic range was too high and didn’t feel quite right when playing.

The reasoning behind the new mapping is as follows:

It makes sense that there is a linear relationship between loudness and the MIDI velocity number. Although velocity 0 represents note off in the MIDI standard, we don't actually want 0 to be silent in the mapping curve of velocity value to dB. Instead we need to pick a sensible dynamic range between 'quietest'(0) and 'loudest' (127). In practice **doubling** the loudness between quiestest and loudest is a reasonable dynamic range. A change of 10 dB is accepted as the difference in level that is perceived by most listeners as “twice as loud” or “half as loud”.

So the new mapping does just this, and the dynamic range is 10dB.

The mapping is between the MIDI note velocity, 0 to 127, and an amplitude scaling factor between 0 and 65535. This sort of scaling is used throughout this firmware because it lends itself nicely to ***OutputAmplitude = InputAmplitude \* ScalingFactor/65536***, where the division is just a shift.

The graph shows that this new version boosts the amplitude of played notes, compared to the old version, apart from at the highest velocity. In fact the average tone amplitude value for version 1 was 24763 and for this version is 38954, which is a significant ~ 4dB (3.93) improvement of SNR. It also ‘feels’ more natural to play.

# Inharmonicity setting

To get the new option to display, there is a new setting (Tools>Settings) in the app where you set the Tone Processor firmware version. This needs to be set to 7.

A screenshot of a computer

Description automatically generated

The new option shows in the the Patch General settings as ‘Harmonic Algorithm’ and you can apply it to any patch:

A screenshot of a computer

Description automatically generated

Inharmonicity is a deep topic and I give an overview below, plus how it has been implemented :

Additive synthesisers commonly have only an integer-based relationship between harmonics. i.e the harmonic frequencies are exact multiples of the fundamental. However in the natural vibration of strings the relationship isn’t quite integer based (‘partials’ is really a better word than ‘harmonics’ in this case). This ‘inharmonicity’ is due to the stiffness of the string, which effectively shortens the string for higher frequencies (the end fixing points are less flexible at high frequencies), which sharpens the pitch.

So in a single string the upper harmonics are slightly ‘stretched’ and this inharmonicity results in a richer and far more complex sound when the string is plucked or hammered. This applies to all stringed instruments, and the same affect can also apply to non-stringed instruments for similar reasons.

The degree of stretching is a complex topic and depends on the thickness, length and material of the string. However research by Harvey Fletcher lead to this general equation:

A picture containing text, font, typography, white

Description automatically generated

Where:

* n is the harmonic number
* Fn is the harmonic (partial) frequency at harmonic number n
* F0 is the fundamental frequency
* B is a factor depending on the stiffness of the string

A consequence of this stretching comes into play when instruments are tuned, because it’s common to tune one string against another using ‘beat frequencies’, and minimising this beat frequency. If the harmonics are slightly stretched, this means that the tuning isn’t quite what you’d expect it to be if everything was integer-based.

This lead to research on pianos by Railsback, who discovered that piano-tuners, tuning by ear, tend to change the fundamental frequencies in a well-defined curve, now called the Railsback curve. This actually doesn’t have a precise mathematical formula, but is a curve based on observation. The acoustics of a piano are complex, because the string type varies, and also the length (especially in grand pianos), not to mention many keys have 3 associated strings. I’ve put an approximation of this curve into a lookup table, which is graphed here :

By applying this curve, plus the Harvey Fletcher equation, it’s possible to map the stretching of all the fundamentals and harmonics for every MIDI note played on a Piano. I’ve done just that and through a lot of careful calculation have developed a detailed lookup table within V7 firmware.

This graph summarises the inharmonicity for a range of ‘A’ notes :

However, implementing the stretching of harmonics raises issues :

The Spectral Sound Module is essentially wavetable-based, where the wavetable contains a single cycle of the fundamental plus related harmonics. The fact that the harmonics are usually integer-based makes things easy – because regardless of how many harmonics and their amplitudes, the summed total signal always starts and ends at 0 amplitude (provided all harmonics are in phase with the fundamental).

Stretching any harmonics results in the end value in the wavetable being non-zero and their being a nasty discontinuity, which will generate spurious frequency artifacts potentially above the nyquist frequency. i.e there is potential for significant harmonic distortion.

The ideal approach to achieving an inharmonic additive synthesiser would be to re-design and have a mass of oscillators for all harmonics, where their frequencies can be set independantly. However this required too much computation, and one of the major benefits of the current wavetable approach is that it’s computationally efficient – our hearing can’t distinguish changes of timbre less than 10ms, so having a wavetable recalculate at around this rate is ideal, with simple, fast cycling of the wavetable to generate the output signal in between.

So the novel approach taken has been to ‘smooth’ any discontinuity at the end of the wavetable, using a custom smoothing algorithm, that’s run at the end of recalculating the wavetable signal. This still introduces some harmonic distortion, and it would be very hard to actually calculate what this is, but in general the smoothing is likely to make it sub-nyquist.

Another point is that stretching of harmonics is relatively greater for higher harmonics, and at this point the proportion of any discontinuity is small compared to the overall length of the wavetable, and so the effect of distortion diminishes rapidly for higher harmonics.

So in summary, inharmonicity has been implemted, albeit in a novel way, and it should be noted that this approach isn’t ideal. Not only because of the small distortion mentioned, but also because there is a phase-shift of harmonics for every cycle of the fundamental. It’s completely unknown how our hearing might interpret this – if at all.

This needs deeper analysis really, applying this new setting in the Sound Module and abserving the effects.

# Other Signal-To-Noise investigations

Finally, apart from the benefit of the new velocity mapping descibed above, some investigation has also been done into the circuit noise. An immediate observation is that noise can also be greatly reduced by **disconnecting the USB cable when not needed, plus making sure the module volume control is set to max !**

It’s not clear if the USB noise is related to the connected laptop/pc or to the actual additional processing that occurs when USB is connected.

The noise that remains in the circuit is sadly largely due to the DAC. There are comments online relating to this baseline noise level, even when you force the DAC values to 0. If you disable the DAC then the overall circuit noise drops significantly and it’s a shame this isn’t the default noise floor !

One possibility is to add a noise gate circuit, and use the fact that the AGC in the Mixer chip has a 128 sample lookahead. I’ve already experimented with this, where I’ve made the Mixer ‘busy’ LED only go high when the peak absolute lookahead value is greater than 0. However my experimental noise circuit wasn’t effective as it was thrown together. It would have to be properly designed into a new version of the PCB to be effective, and at the moment I don’t feel it’s worth it as the noise level is already far improved.

# Other thoughts about the Tone Processor system design

A lot of effort has been put into thinking of ways to increase the polyphony on a single Tone Processor, with the view of potentially making a ‘lite’ version of the whole module, incorporating just a single tone processor.

One processing inefficiency at present is the fact that a Tone Processor is interrupted to supply a sample to the Mixer, every single sample. This is a lot of processing overhead from servicing these interrupts. A better way computationally would be to have a buffer and send blocks of samples in one go. This could massively reduce the interrupt overhead. However this would come at the cost of system latency and would also require the Mixer to have 6 receive buffers – one for each Tone Processor – which becomes a bit unweildy.

Another area of potential improvement is to use the dsp capabilities and built-in commands to the full, by calculating multiplication using vectors. This has tantalising potential but there isn’t enough space on the chip to fit everything in (the vectors have to be in specific places in memory).

So the overall conclusion is that the system is as good as it probably can be, for this circuit design and use of specific dsPICs !