Distortion Reduction Techniques for 8-bit Magic Sinewaves

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2003 as GuruGram #19
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

A new class of **math functions** called **Magic Sinewaves** lets you efficiently produce power sinewaves that can have **any** chosen number of low harmonics forced very near **zero**. And do so using the fewest possible switching events for the highest possible energy efficiency. A new Intro appears **here**, along with a development proposal **here**, a tutorial **here**, and calculators **here**, and seminars and workshops **here**.

Magic Sinewaves only work properly when their pulse positions and delays are **exactly** specified. The various **Magic Sinewave Calculators** typically do show what may be excessive distortion if you try to use 8-bit quantized data for your table lookup values. Obviously, 8-bit data is desirable for lookup table simplicity and compactness.

It turns out that...

Some sneaky "smoke and mirror" trickery can very dramatically reduce 8-bit magic sinewave distortion.

These tricks include **bigger chips**, **delta for all**, **offsetting**, **delay scaling**, **linear differencing**, **quadratic interpolation**, **plot viewing**, and **progressive builds**. All of which seem to be fairly easily implemented. And, taken together, should be able to dramatically reduce 8-bit **Magic Sinewave** distortion.

Ploy #1: Bigger chips

It is fastest and simplest to just go ahead and use 16-bit delay values instead of 8-bit ones. While this doubles the needed storage, overall microchip sizes would often remain fairly small and economical.

Ploy #2: Delta for all

Delta magic sinewaves only need **one-half** the storage of best efficeicy ones, so you could do deltas with 16-bit resolution in the same storage as a best efficiency with 8-bit resolution. The often optimum delta-28 would thus require 1400 8-bit bytes of storage and confortably fit in a 2K address space. Note that deltas reject only **(3n/4-1)** harmonics.

Ploy #3: Offsetting

By storing a few extra **offset** values and summing certain calculations, some significant distortion reduction is possible. Consider **p1s** as it ranges from 10.097 degrees at full unity amplitude up to 11.987 degrees at one percent amplitude. We'll note in passing that **p1s** "backs up" as the pulses get fatter.

We can accurately generate a 10.097 degree **offset** and then only generate the **difference** of 0 to 1.890 degrees using 8-bit values. For a roughly 6:1 resolution improvement. But on top of this, we can add...

Ploy #4: Delay Scaling

The original quantization calculations assumed that "full scale" for 8-bits was 20 degrees, or somewhat more than the fattest needed pulse. By **scaling** to different delay routines for different pulse positions, we can gain quite a bit. Ferinstance, in the case of **p1S,** we need only use an 8-bit full scale value of two degrees. Giving us a further 10:1 improvement, for a combined whopping 60:1. Or nearly a 14 bit accuracy from a stored 8-bit amplitude value!

The benefits of offset and scaling do drop as you go further into the quadrant. A separate small table could hold the needed offset and scaling values. Note that these would apply to **all** amplitudes in the sequence. And thus would add only a few additional bytes to total storage needs.

Ploy #5: Linear Differencing

In general, the pulse widths increase more or less linearly with amplitude. If a guess was made that an 0.53 pulse width was 53/100ths the corresponding 1.00 pulse width, then only the actual **difference** need be stored. The difference would then be summed with the guess. Since this difference is typically only a fraction of a degree, table lookup storage accuracy would dramatically increase. Typically giving four or five bits of improvement.

This may require a micro with a multiply instruction. An approximation might involve splitting the amplitudes into four or eight groups and using shifts or whatever to properly deal with them.

Ploy #6: Quadratic Interpolation

With **quadratic interpolation**, only **one-tenth** or fewer amplitudes need be stored along with an **increment** and an **increment delta** value. Details of this method appear in **MUSE153.PDF**.

If 0.50 is stored as **x**, 0.51 will be **y=x+increment**. 0.52 would be **y+increment+delta**, and so on. Each new interval would get **fatter** by delta. Per the detailed examples in the tutorial.

Ploy #7: Plot Viewing

Actually viewing the **delay-vs-amplitude** magic sinewave plots can reveal other avenues for substantial error reduction. As in this crude **Delta Friendly 28** plot and its **sourcecode**. Or this **Best Distortion 28** and its **sourcecode**.

In which we see several rather surprising features. Of the eight delay values useful for this particular delta magic sinewave, three appear to be perfectly linear with amplitude. And thus can be **eliminated entirely** from table lookup storage! Provided a multiply function is available. Three more deviate from linearity by only a small fraction of a degree. And easily handled by linear differencing.

The final two delay curves only require two degrees of full scale accuracy, and thus give 10X over the 20 degree quantizations used in the **calculators**. Interestingly, most of the "bent" part of these curves takes place above 0.9 amplitude. Which leaves options of not using the highest possible amplitudes or using a few 16-bit double words to deal with these small and specific areas.

Ploy #8: Progressive Builds

When you plot view a best distortion magic sinewave, the widest pulse is also seen to be the most nonlinear. Giving a "double whammy" to the needed resolution. But note that **best distortion pulse width is always progressive**. Suggesting that we **sum** previous pulse subroutines to increase build accuracy.

Ferinstance, on a best distortion 28, the critical pulse seven can be generated by doing a pulse **six** delay and then adding a differential new delay to it. Pulse six can be built by doing a pulse **five** delay and then adding its differential.

Setting Frequency

These more precise delay schemes could make setting the overall frequency trickier. The best way to deal with this would appear to be a two-step process where one microcontroller generates a variable reference **clock** frequency and a second one uses that frequency to generate the desired **magic sinewave**. This also nicely avoids the need for excessively high clock frequencies.

For More Help

Additional **Magic Sinewave** help is available per the previously shown web links. Detailed analysis of distortion improvements are available at our standard consulting rates. Per our **Magic Sinewave Development Proposal** and our **Consulting Services**.

Additional **GuruGrams** await your ongoing support as a **Synergetics Partner**.