A new class of math functions called Magic Sinewaves now 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. Two new intros appear here and here, along with a development proposal here, a tutorial here, visualizations here, jitter and distortion analysis here, lots of calculators here, some evaluation chips here and seminars and workshops here.

Successful magic sinewave sourcecode development seems to need a unique programming style. Combined with non-obvious and often non-mainstream techniques required to create useful real world devices. In this GuruGram, we will be looking at my programming techniques for PIC based magic sinewave sourcecode development and evaluation.

The latest fully tested and professional sourcecode is now readily available to commercial users at quite reasonable pricing. As are custom new designs. Free older sourcecode has been posted to my website for possible limited demo use by students or low budget home experimenters.

Why a PIC?

The PIC family of microprocessors from Microchip Technology are an obvious and economical choice for low end magic sinewave development. The 16F628 easily holds a Delta Friendly Delta 28 generator, and the 16F648 nicely accommodates Delta 44, Delta 60, Best Efficiency 28, and possibly a Best Efficiency 44 design.

But we will note in passing that programming of faster premium DSP or higher level microcomputer chips that can directly work with 12-bit or even 16-bit data words would likely lead to simpler and cleaner code. The reason being first that the fully expanded delay values required for magic sinewaves often demand a full 12-bit resolution. And second that coding and timing has to be extremely tight, ultra precise, and perfectly equalized for useful results.
Getting Started

At first glance, Magic Sinewave sourcecode seems trivially easy: For all the best efficiency designs, you set a pair of port lines, delay an interval, clear the port lines, delay some more, and repeat for full cycles. For delta friendly designs, you do the same thing using port patterns instead.

But, as always, the devil is in the details...

Magic sinewave timing must be exceptionally precise and perfectly equalized for useful low harmonic rejection!

Your first problem in any Magic Sinewave development is to create a list of delay values that establish the position and spacing of each Magic Sinewave pulse. You can start with our Magic Sinewave Calculators, such as this 28-bit one.

From the calculators, you extract a list of values (auto extraction features and custom services are available to Synergetics Partners). You then quantize these values to your available accuracy as set by your clock to output frequency ratio. A Delta-28 might involve 3472 counts per 30 degrees or 41,664 instruction cycles per output cycle. Or 166,656 PIC clock cycles.

Next, you "shake the box" and check the nearest 78,125 or so quantized magic sinewaves to find the one with the best mix of lowest distortion and amplitude accuracy. Per these guidelines. (again, auto quantizer code and custom services are available to Synergetics Partners). To prevent any delay versus amplitude sensitivity, the totals of delay values for any amplitude must be a constant.

Performance of your final delay choices can be verified by simulation. You can use Sigview 32 per this tutorial.

As typical examples, a Delta 28 Magic sinewave might need eight delay values per amplitude or 808 total values for amplitudes 0-100. As we will shortly see, these values may range from very low integers to several thousand counts each.

Thankfully, there are tricks we can pull so we can store them as 8-bit data values. At present, our goal is simply to gain a raw list of the uncompressed delay values. Note that Best Efficiency magic sinewaves will require double the storage of their Delta Friendly equivalents. But will reject more harmonics and use fewer switching transitions to do so. For a correspondingly higher efficiency.

A second task after your desired delay tables are complete is to generate a list of patterns needed for delta friendly output. These are not in the least obvious and can be quite subtle, but they are detailed here, here, here, and here.

Only after the delay list and the pattern list are created and simulator checked can you even begin to worry about programming any actual hardware.

— 40.2 —
Which Tools?

The ideal way to develop Magic Sinewaves is with a real time ICE in circuit PIC emulator. Preferably one that includes internal high performance Fourier Spectral Analysis. Since such tools are usually very costly, here instead is my bare minimum recommendation for low end Magic Sinewave development tools...

<table>
<thead>
<tr>
<th>Oshensoft</th>
<th>PIC development system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transtronics</td>
<td>Pocket Programmer II</td>
</tr>
<tr>
<td>Sigview 32</td>
<td>Fourier Analysis Simulator</td>
</tr>
<tr>
<td>Synergetics</td>
<td>JavaScript calculators</td>
</tr>
<tr>
<td>Synergetics</td>
<td>Gonzo analysis routines</td>
</tr>
</tbody>
</table>

In addition, a 20 MHz breadboarding system will be required, combined with methods of accurately observing and testing the generated waveforms. Minimum gear here might include a Tektronix 2246 oscilloscope and a Hewlett Packard 3581C Frequency Selective Voltmeter. A quality audio spectrum analyzer, of course, is also highly desirable.

Some First Principles

Here are some of the concepts that can make Magic Sinewave development rather tricky...

**Exact Timing**— Coding must be precise to within one instruction cycle to fully guarantee proper rejection of low harmonics.

**Full Equalization**— Each and every path through the updating process and all other code must take exactly the same number of instruction cycles. No interrupts or side projects are allowed. Chip is 100% committed.

**Variable Clocking**— The frequency reference clock is a data input that varies over a mid to wide range. Certain chip features such as baud rates, timers, or A/D converters may end up restricted or useless.

**High Clock to Output Ratio**— As many as 41,664 instruction cycles are typically required per single output cycle to guarantee proper low harmonic rejection. In certain Magic Sinewave "pinch points", few instruction cycles will be available.

**Machine Language Only**— Magic Sinewaves demand the utmost in instruction set organization and use. Compiling assembly code from a higher language flat out ain’t gonna happen. Bare metal programming is a must.

**Special Speed Tricks**— Required minimum instruction times trade against code length, complexity, and obtuseness. Techniques here include table lookup, pipelining, splitting, linear “unwrapped” coding, and limited subroutine use.
Understanding RETLW — The usual mechanism for a PIC table lookup involves jumping to a calculated subroutine, followed by a RETLW that returns a literal value to the W register.

Understanding PCLATH — Modifying the program counter is a useful way of indexing table lookups and precision delays. The high bytes are preset by tricky and obscure register PCLATH. Which must be carefully reset before reuse.

Wide Delay Range — Needed time delay values go from very few to many thousands of cycles. Special expansion techniques (covered below) permit 8-bit table lookup storage of up to 12-bit data.

Pipelining — Slower delay table lookups are best done early, prestashing their values in registers for later fast access. There simply is not enough time for multiple table lookups and PCLATH mods at certain code pinch points.

Split Code — Separate code is best done for zero amplitude which still has to maintain sync. Because pinch points are different for low and high amplitudes, a pair of separate low and high amplitude routines may prove useful for fancier Magic Sinewave sequences.

Compromise — A sense of balance must be maintained. Certain low amplitudes of extremely low power may not be initially realizable. And an occasional one or two byte timing error may be necessary.

A Precision Time Delay

An essential routine for Magic Sinewave code is a precision wide range time delay having minimum overhead. This module delays from 0 to 255 machine cycles with a mere six bytes of overhead, including its calling subroutine...

```
CALL as subroutine with delay value in W and PCLATH preset to page three...
    02FE SUBWF PCL,1    ; move pc to needed delay
    02FF NOP           ; padding - never accessed
    0300 NOP           ; burn one cycle if used
    .................  ; .................
    03FD NOP           ; burn one cycle if used
    03FE NOP           ; burn one cycle if used
    03FF RETLW 0x00    ; return to calling routine
```

Yeah, this uses up a whole page of NOP commands. But otherwise is fast, clean, and convenient. A subtraction gets used so that more cycle-burning NOP instructions are used for longer time delays. Note that the code must begin two bytes before page three. Note further that an input delay value of zero goes directly to a RETLW return.

— 40.4 —
Delay Value Expansion

The needed delay values for Magic Sinewave pulse widths and interpulse spacings vary from a very few to many thousands of instruction cycles. Thus, they cannot be directly stored as single 8-bit words. Instead, a two step process gets used...

To gain the needed resolution, delay values can consist of an expansion calculation that is combined with a residue error correction.

The amplitude versus delay curves of VISMAGSN.PDF show that most delays are more or less linear with amplitude. This suggests using an amplitude ramp or its inverse to approximate most of the delay. Any residue can then be reduced to 8-bit values in the 0-255 range and use the above precision time delay to exactly make up any remaining error.

All of which may need a multiply. By far the simplest and fastest 8-bit multiply is to run around a loop an integral number of times. Here is an example routine that delays five times the amplitude plus six overhead cycles...

```
EXPANDG  MOVF   AMPLIT,0 ; get amplitude
LOOPG    BTFSC  STATUS,Z ; check zero flag
         RETLW  00       ; return when done
         ADDLW  0xFF     ; count down
         GOTO   LOOPG    ; repeat for five more
```

Thus, two steps are used for most time delays. Most of the delay is taken out with an amplitude dependent calculation. Any residue error is then taken out with a precision delay that is wholly within an 8-bit range. At certain code pinch points, it may be necessary to truncate or threshold the amplitude delay loops. See the actual sourcecode for specific examples.

Updating

The updating code can be more or less conventional, except that...

Update code SHOULD be reasonably compact and fast.
Update code MUST be inside the normal delay timing.
Update code MUST be fully equalized for ALL paths.

Present thinking is to have seven input lines capable of 128 logic states.
101 of these states can be used to directly parallel input amplitudes 0-100 for immediate updates. Codes X111 XXXX can be used to sense four pushbuttons for step down, step up, slew down, and slew up.

The step buttons will advance or retard one count per pressing and must be released before an additional advance can be made. Pressed slew buttons will continually advance or retard at a preprogrammed slew rate. The down buttons "stick" at amplitude 0, while the up buttons "stick" at amplitude 100. All buttons up is a "make no changes" state, as are any multi-button down lockout states.

Eleven states remain for special functions. Exciting possibilities here include...

- soft or slow starts
- advancing or retarding output phase
- increasing or decreasing slew rate
- speed steps for a blender
- intensity presets for stage lighting
- direct analog input modes
- flame flicker or random effects
- power factor correction or ups
- mood lighting ultra slow dimming
- phase sequence or direction reversal
- custom system control features
- parallel input change slewing

The MSD28-5X evaluation chip presently attempts an update six times per cycle, or every 60 degrees of phase. Giving a thirty degree average latency. This seems to be a useful compromise between complexity and transient behavior.

Synchronization

Output synchronization pulses can be enormously useful. Besides being handy for scope viewing, they are essential for solar power and similar synchronous inverter apps where the output is to be phase and/or frequency locked to an existing power line reference.

A square wave is presently provided precisely at the first zero degree phase transition of phase "A". Its position is slightly adjustable to match system needs. Square wave is high for 0-180 degrees and low for 180-360 degrees.

Quite a bit has to happen inside any 60 degree update cycle interval. These start with an UPDATE routine that finds and stores a possible new amplitude but does not immediately use it. An equalizing delay follows to center any possible sync pulse. An amplitude dependent delay follows that still uses the old delay values needed to exactly complete the previous cycle.

At this point, the start of a "real" sync pulse will be output if at phase zero, and "fake" sync pulses will be output at phases 60, 120, 180, 240, and 300.
A HANDSH,2 steering flag bit decides when a real sync pulse is to be output.

The update cycle interval continues by pipelining the delay values for this and future cycles into appropriate registers using an GRabdly routine. This is followed by an amplitude dependent delay that now uses the new delay values.

There are presently two possible routes through the update cycle interval. The "normal" route is followed by amplitudes 1-100 as described above. The "zero" route is followed only by amplitude zero. It eliminates any amplitude dependent delays and substitutes the rest of the output code with a long fixed delay.

This maintains exact sync and frequency during zero amplitude output that exactly matches that of the higher amplitude outputs. The zero flag does the steering between zero and "live" amplitude outputs.

**Organization**

Here is how the MSD28D-05X is arranged...

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page ZERO</td>
<td>Setup and main output loop</td>
</tr>
<tr>
<td>Page ONE</td>
<td>Main output loop</td>
</tr>
<tr>
<td>Page TWO</td>
<td>Utilities and expansion</td>
</tr>
<tr>
<td>Page THREE</td>
<td>Precision time delay</td>
</tr>
<tr>
<td>Page FOUR</td>
<td>Delay tables A and B</td>
</tr>
<tr>
<td>Page FIVE</td>
<td>Delay tables C and D</td>
</tr>
<tr>
<td>Page SIX</td>
<td>Delay tables E and F</td>
</tr>
<tr>
<td>Page SEVEN</td>
<td>Delay tables G and H</td>
</tr>
</tbody>
</table>

While considerable room remains in the chip, fancier Magic Sinewave devices should almost certainly start with the larger sixteen page 16F648. We’ll note in passing that five delay tables can be placed on two pages by splitting by 0-50 and 51-100 amplitude and properly incrementing PCLATH.

**For More Help**

The MS28D-05X chips are available at $19.63 each plus shipping. Sourcecode and one hour of consulting is separately available for $89 additional.

You can order your samples and sourcecode here. Or via our eBay store.

Licensing arrangements for your own chip production using our sourcecode or any of its derivatives or variants are available and are quite reasonably priced. You can email me for further details.

Additional Magic Sinewave services, programming, seminars, training and project development is available here and here. Further GuruGrams columns await your ongoing support as a Synergetics Partner.