Order this document
by AN1771/D

Motorola Semiconductor Application Note

AN1771

Precision Sine-Wave Tone Synthesis Using 8-Bit MCUs

By Joe Haas
TSG Body Electronics and Occupant Safety Division
Austin, Texas

Introduction

The pervasive nature of the modern microcontroller (MCU) has resulted in numerous products that now contain one or more MCUs as central subsystems. Cell phones, base stations, repeaters, SLICs (subscriber line interface cards), and cordless telephones are just a few of the many products which have MCUs at the center of their functionality.

These products also require precision tone generators for functions such as dual-tone-multi-frequency signaling (DTMF), call progress tones, continuous tone-coded squelch system encode (CTCSS), digital continuous tone-coded squelch system encode (DCTCSS), and user interface chimes.

While off-the-shelf components generally are available for these functions, the added cost can be greatly reduced by using the already present MCU to synthesize the desired tones. This benefit is multiplied in systems where many unrelated tone protocols are required, since the same synthesis firmware/hardware can be used across a wide range of frequencies.

This application note presents basic tone synthesis techniques and illustrates their implementation using the HC08, HC05, HC11, and HC12 Families of MCUs.

MOTOROLA

Tone Synthesizer Basics

When an analog signal is stored in digital memory, an A/D (analog-to-digital) converter is used to provide quantized samples at a specific data rate (known as the sample rate or F_S) to be stored in memory as binary values. To retrieve the stored signal, the binary samples are extracted from memory and sent to a D/A (digital-to-analog) converter at the same rate at which they were stored. As long as the analog signal has no frequency components greater than half the sample rate (as per the Nyquist criteria), the reconstructed signal will appear to closely follow the original waveform. (Quantization effects in the A/D will introduce some errors.)

To generate a tone at a specific frequency, one can simply digitize a sample of the tone to be reconstructed and store the sample in the system memory for later recall. However, for a multi-tone system, each tone requires a separate sample and thus its own memory storage. The more tone frequencies required, the more storage needed to hold the samples. In addition, the sample lengths for different frequencies will not be consistent, since each stored sample must continue until the signal repeats. This method would be tedious to maintain, use large amounts of memory to store relatively few tones, and would be limited to only those tones which were stored previously.

Another reconstruction method would be to generate a single sample and vary the reconstruction sample rate. This would produce a signal with a variable frequency with only one stored cycle, but it would yield a variable and non-linear Fstep (Fstep is the smallest, non-zero increment of frequency).

As an example, consider an 8-MHz master clock and a 256-byte sine sample. The 8-MHz master clock is applied to a programmable 16-bit divider which is used to set the sample rate. To obtain reconstructed tones from near-DC to 3 kHz, the divider would range from 65535 (8E6 / 65535 / 256 = 0.477 Hz) to 10 (8E6 / 10 / 256 = 3.125 kHz).

At the low end of the frequency range, the Fstep would be:

$$\begin{aligned}\text{Fstep} &= \text{Fdiv2} - \text{Fdiv1}\\ &= (8E6 / 65534 / 256) - (8E6 / 65535 / 256)\\ &= 0.47685 - 0.47684\\ &= 0.00001\ \text{Hz}\end{aligned}$$

While at the high end:

$$\begin{aligned}\text{Fstep} &= \text{Fdiv2} - \text{Fdiv1}\\ &= (8E6 / 10 / 256) - (8E6/11 / 256)\\ &= 3125 - 2841\\ &= 284\ \text{Hz}\end{aligned}$$

This illustrates that the example would exhibit an Fstep variation of several orders of magnitude across the signal passband. Not only would this complicate real-time frequency calculations on the target system, but the Fstep granularity at the higher frequencies would severely limit the utility of the system. (Typically, Fstep should be at least 0.5 Hz across the passband for most applications.)

Filtering this system would also pose some problems. A reconstruction filter (for instance, a low-pass filter with a cutoff frequency, Fc, just below the Nyquist rate of Fs / 2) is used to remove the PWM (pulse width modulation) sample frequency and higher order harmonics. If the sample rate is varied, the user must undertake the difficult and expensive task of designing a tunable filter that can track the sample rate so that the reconstructed signal can have a flat response in the passband. This would require additional hardware, MCU resources, and firmware support which would increase the cost of both development and production.

Direct Look-Up Synthesis

The direct-look-up synthesis algorithm described here uses a combination of the aforementioned schemes to produce precision waveforms across a specific frequency band. A look-up table holds a replica of the waveshape which is to be generated. (Typically, this is a mathematically generated sine table with N entries.) At every sample point, the algorithm uses the value of a phase accumulator to extract data from the table which is sent to the D/A. The phase accumulator is a software register used to keep a "running total" of the current phase valve of the synthesized signal. The algorithm also updates the phase accumulator to be used at the next sample point by adding a "delta phase" value, or Delta.

NOTE: *Look-up table accesses are modulo-N, such that any access beyond the end of the table will wrap-around to the beginning.*

To obtain finer Fstep granularity, Delta and the phase accumulator are represented as fractional quantities with the integer portion being used as the index into the sine table.

The frequency of the resulting tone can be deduced by setting Delta = 1. At every sample point, the integer portion of the phase accumulator is incremented by exactly 1. Since this corresponds to the index into the sine table, the D/A output simply will follow the sine table. Since the table holds one cycle, the frequency of the output will be 1/tgen, where tgen is the time required for one full cycle.

With N table entries sent at 1/Fs per entry:

tgen = N * 1 / Fs.

If Delta is doubled, the table will be cycled in half the samples, which results in:

tgen = N / (2Fs)

Thus, tgen is inversely proportional to the value of Delta. Since F = 1/t, the frequency of the generated signal is given by this equation:

(1) Fgen = (Fs * Delta) / N

As noted, Delta is a fractional quantity valid in this range:

0 <= Delta < N / 2

For microcontroller applications, Delta is most easily represented as a 2-byte quantity (referred to here as Dreg) with the upper byte holding the integer portion and the lower byte holding the fractional portion (thus, the radix lies between bits 7 and 8). The decimal value of Delta would be represented as:

(2) Delta = Dreg[15:0] / mod(fractional)

Since the fractional portion is represented here as an 8-bit value, mod(fractional) = 256 which yields:

(3) Delta = Dreg[15:0] / 256

and

Dreg[15:0] = 256 * Delta

The 16-bit Dreg value is thus added to the 16-bit phase accumulator at each sample period to generate the table index and running phase reference. The table index is extracted from the phase accumulator by masking the integer portion with N – 1 (valid for N = 2^x, where x is a positive integer). For an 8-byte table, the mask would be $07 (the lower three bits) and for a 256-byte table the mask would be $FF (all eight bits of the integer portion of Delta). This provides a simple and efficient method of implementing the numerical values used to represent Delta.

Example:

Given: N = 8, Fs = 8 kHz, and Fgen = 800 Hz

From equation 1, solve for Delta,

Delta = (N * Fgen) / Fs

= (8 * 800) / 8000

= 0.8

The integer and fractional parts (high byte/low byte) are represented as:

Integer = 0

Fractional = 0.8 * 256 = 204.8 (round to the nearest integer) = $CD

Dreg = $00CD

The pointer mask, as noted, would be Accum[10:8] = [111], which is used as an offset into the 8-byte sine table.

Table 1. Example of a 4-Bit, Unsigned Sine Table (D/A = 8 + int(sin(2*pi*x / 16) * 15)

Offset, x	D/A	Degrees
0	8	0
1	13	45
2	15	90
3	13	135
4	8	180
5	2	225
6	0	270
7	2	315

Table 2. Example of Phase Accumulator History (Each Line = 1 Sample Period)

Accum [15:0]	Accum [10:8]	D/A Value from Table
$0000	$00	$08
$00CD	$00	$08
$019A	$01	$0D
$0267	$02	$0F
$0334	$03	$0D
$0401	$04	$08
$04CE	$04	$08
:	:	:
:	:	:

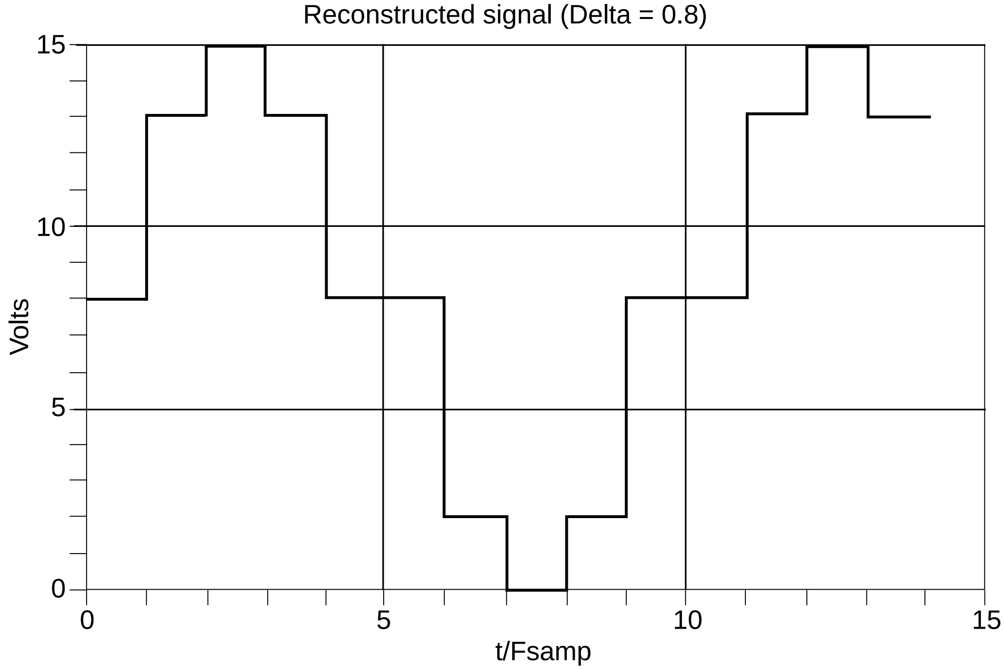


Figure 1. Delta = 0.8 (800 Hz) Example Using 8-Byte Table

Figure 1 illustrates a full cycle of the reconstructed signal, with each horizontal division representing one sample period (1 / Fs = 1 / 8000 = 125 µs). From this, the period of the waveform can be calculated by counting the number of sample periods for a full cycle and multiplying by the sample period (in this case, 10 samples = 1.25 ms = 1 / 800 Hz).

As is apparent from the plot of **Figure 1**, a table length of 8 results in a coarse reconstruction; a longer sine table gives more resolution and reduces harmonic distortion. Since the integer portion of Delta is eight bits, a 256-byte table is easily indexed while not reserving an excessive amount of memory. Linear interpolation can be used to increase accuracy with a shorter table, but this is generally not feasible on most MCUs due to processor bandwidth limitations. (However, the HC12 can support this method as is described later in this application note.)

An interesting result of this reconstruction method is that the relationship between Delta and Fs is linear, with each unit change in Delta resulting in the same change in Fgen across the entire pass-band. This value was referred to earlier as Fstep and represents the smallest possible change in Fgen.

Fstep can be found from equations 1 and 3 by:

Fstep = F1 – F2

Choose F1 and F2:

F1 = (Fs * Dreg / 256) / N

F2 = (Fs * (Dreg – 1) / 256) / N

Fstep = ((Fs * Dreg / 256) / N) – (Fs * (Dreg – 1) / 256) / N

= Fs / (256 * N) * (Dreg – (Dreg – 1))

(4) Fstep = Fs / (256 * N)

Thus, for any value of Dreg, the Fstep is always equal to Fs / (256 * N). One result worth consideration is that for a given sample rate, the only remaining variable to determine Fstep is the table length.

From the previous example, Fs = 8000 Hz and N = 8, which gives Fstep = 3.906 Hz. Increasing the table size to N = 256 results in Fstep = 0.122 Hz. Fstep specifies the maximum gross frequency error for any given tone frequency allowing system accuracy within ±Fstep/2 of any desired frequency.

After signal purity considerations, Fstep typically is the next most important design parameter as it determines how accurately generic tone frequencies can be generated. Generally, a designer is faced with the need to generate tones over a specific frequency range with some degree of accuracy. Typically, this is specified in terms of %error (plus or minus) of the desired frequency, but also may be expressed as +/–ΔF(Hz). (Of course, specifying the error in this manner is trivial because Fstep < 2ΔF is all that is required for the design to meet the specification.)

For systems that express error in terms of percent, use this equation to determine the maximum allowed Fstep:

(5) Fstep(max) = (Fmin * %error) / 2

Where Fmin is the minimum desired frequency to be generated

Of course, this equation represents the design minimum, and usually it is desirable to choose as small an Fstep as is practical. Actual Fstep should be at least 50 percent of Fstep (max) from equation (5) to allow for round-off errors and normal variations in system clock frequency.

Dual Tone (Chord) Synthesis

Applications such as DTMF and call progress signaling require dual tone synthesis which is simply the generation of two mixed tones of unrelated frequencies. The term "chord" is sometimes used to describe this technique, even though the two tones are not necessarily related by harmonics. In direct look-up synthesis, dual tone generation is a straightforward extension of the single tone case described earlier. Two separate tones can be generated by maintaining two separate Dreg and phase accumulator registers. For each sample period, the system adds Dreg1 to accumulator1 and Dreg2 to accumulator2. The index extracted from each accumulator is used to separately extract D/A values from the same look-up table. Before sending to the D/A, however, these two values are added in software, with the resulting D/A output representing the algebraic sum of the two unrelated tones.

When mixing two signals on the same D/A channel in this manner, it is important to avoid overflow. Overflow occurs when a value is calculated that exceeds the D/A maximum range. If the two signals are of the same amplitude, the range of instantaneous amplitude can vary from a minimum of 0 to a maximum of 2A, where A is the maximum amplitude of the individual signals. Thus, the maximum allowed value is D/A(max) = 2A, or A = D/A(max) / 2.

This can most easily be accomplished by "pre-dividing" the sine table values by 2 so that when any two values are summed, the result won't overflow the D/A.

While pre-division minimizes the real-time effort required by the firmware, it also increases the round-off error (because the D/A LSB (least significant bit) of the original sine table values are lost). A better method is to use the original table and perform the division in real time (post-division). While this adds some overhead to the system, it reduces round-off error which results in improved dynamic range.

With an 8-bit D/A implementation on an 8-bit MCU, the most efficient way to implement post-division is to simply add the byte values and perform an ROR instruction on the result (divide by 2). When the two 8-bit values are added, the carry becomes the ninth bit. The effect of the ROR instruction is to divide this 9-bit value by two with the 8-bit result being the desired D/A value. While the LSB of the final D/A result is lost, it should be noted that this represents only one round-off error instead of the two errors introduced by the pre-division method.

Look-Up Table Requirements

The length of the look-up table is a primary design variable and is determined by available memory and desired Fstep resolution. D/A dynamic range also contributes to the length of the table as some systems can accommodate 10-, 12-, or 16-bit D/A sub-systems. This mandates more memory to hold the longer D/A values in the look-up table.

Another factor in determining table length derives from the nature of the accumulator/pointer system employed. To reduce firmware overhead, the look-up table length should be an exponential multiple of 2 (given earlier as N = 2^x). This simplifies the modulo bit mask to extract the D/A pointers which can save several execution cycles in code that is typically very time sensitive. Optimally, an 8-bit mask is chosen because this requires no extra cycles to extract the pointer which results in a code-optimal table length of 2^8 or 256 bytes. While this may result in an Fstep which is much smaller than required for some applications and increase the table memory required, the reduction in execution cycles can overshadow memory availability concerns in systems where ancillary firmware load is high.

D/A Methods

Two of the most popular D/A methods are direct conversion and pulse width modulation (PWM, also referred to as pulse length modulation, PLM). While direct D/A is the easier to implement (in terms of firmware support) and can result in less distortion and noise than PWM methods, typically, it is more expensive and therefore not as desirable in cost-sensitive systems.

For this reason, the bulk of the following discussion focuses on PWM methods for some of the 8- and 16-bit Motorola microcontrollers. In general, buffered PWM is preferred over non-buffered because the signal-to-noise ratio of the output can be adversely affected by even slight timing variations in the PWM signal.

Filtering

The sample frequency should be as high as possible (relative to the reconstructed signal) to relax the filtering requirements. The lower the sample frequency, the sharper the filtering required to effectively eliminate the stop-band frequency components. Some of the PWM methods described here are limited to carrier frequencies of around 8 kHz or less (due to timer and/or MCU clock speed limits), which can require very sharp filtering to sufficiently remove the PWM carrier and signal aliases from the D/A output for some applications.

Sample rate and filter order are the prime cost factors in a synthesis system. As the sample rate is increased, more D/A performance is required which typically increases costs by forcing the designer to exercise one or more of these choices:

- Use a higher frequency crystal
- Use a PWM module only available on a more expensive MCU
- Use an external D/A

The filter costs also are related to sample rate, but are inversely proportional, which has the effect of countering the cost issues. Thus, it usually is possible for the designer to reach a cost compromise which allows the system performance specifications to be met.

To approach the issue of filtering, the user first must consider the spectral content of the signal that is to be filtered. Sampling theory dictates that when a continuous time signal is sampled at a regular rate (for example, a sine table), the spectrum of the reconstructed signal will be comprised of the spectrum of the original signal plus the original spectrum translated to harmonics of the sample frequency as illustrated in **Figure 2**. To recover the original signal, minus the translated spectra, a reconstruction filter is needed as indicated in the figure.

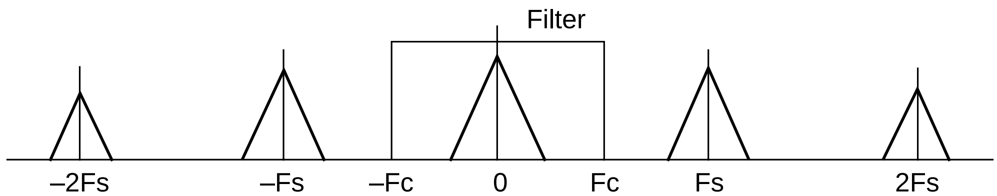


Figure 2. Reconstructed Signal Spectra and Filter Response (Fc = ±Fs/2)

The ideal filter described in **Figure 2** would pass all signals below Fc, and reject all signals above Fc. Unfortunately, it is impossible to construct an ideal filter, which forces the designer to consider real filter performance when designing a synthesis system. The impact of this can be seen in **Figure 3** which shows a synthesized signal, Fgen (Fgen < Fs / 2), inside a real filter passband. The real filter has a cutoff frequency (Fc) that is less than the Nyquist rate, Fs/2. The stop-band aliases Fa = Fs ±Fgen and sample clock are also shown. The intersection of the filter curve with that of the stop-band alias determines the degree of attenuation of the alias component.

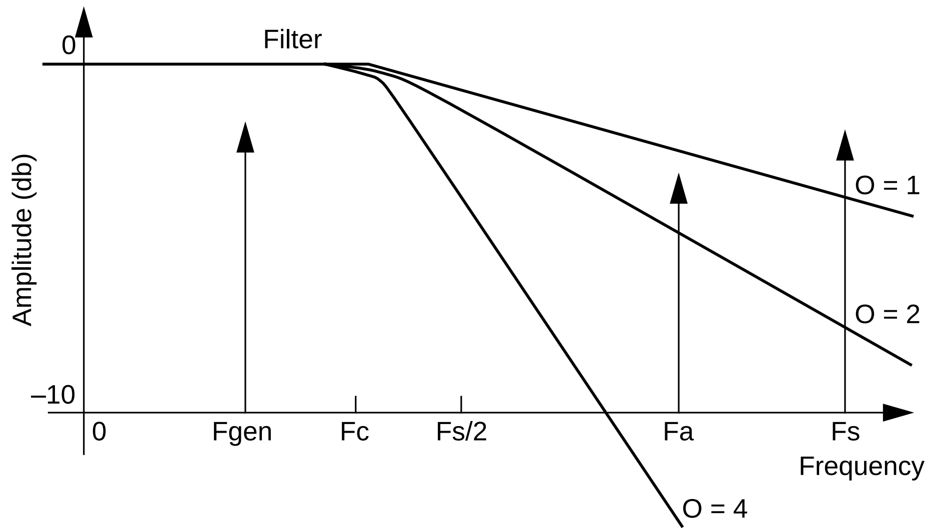


Figure 3. Example Signal and Real Filter Response
O = Order of Filter (1st, 2nd, and 4th Shown)

Filters for signal reconstruction have three important design rules:

1. The passband response should be reasonably flat.
2. The filter cutoff must be somewhat less than the Nyquist rate, but greater than Fgen(max).
3. The required filter order is determined by the separation between Fgen(max) and Fs – Fgen(max).

The flat passband requirement is dictated by the application. Most applications require that signal amplitudes only vary by a small amount across the passband. Typically, Butterworth response is preferred as it has essentially no amplitude ripple in its passband. If the cutoff frequency is chosen too far inside the desired passband (for example, to increase the stop-band attenuation), amplitude distortion (known as twist) can also result which can disrupt the function of tone receivers or detectors (particularly important for dual tone systems).

Once the cutoff frequency is chosen so as to minimize the pass-band distortion, the filter order (for example, the slope of the stop-band attenuation) can be determined by the amount of stop-band alias attenuation required and the system parameters. Better than 40db attenuation in the stop-band is generally a safe figure, although more or less attenuation may be appropriate for a particular system design.

Each order of filtering results in an attenuation slope of approximately 6db/octave in the filter stop-band. Given filter cutoff, Fc, and a target frequency, F, the following equation relates Fc and F in terms of octaves:

(7) Fc * 2^x = F, or

2^x = F/Fc

where x = number of octaves of separation.

To solve for x, the log function is used:

(8) x = log (F / Fc) / log(2)

For a given filter order, O, and cut-off frequency, Fc, the attenuation at a particular frequency, A(F), can be calculated from this formula:

(9) A(F) = (O * 6db / octave) * x octaves

= (6 db * O) * (log (F / Fc) / log(2))

Which can be quickly re-arranged to solve for O:

(10) O = A(F) * log(2) / (6 db * log (F / Fc))

O is a unitless quantity and is rounded to the nearest integer.

If the user assumes that the alias and Fs components are approximately equal to the amplitude of the fundamental signal (This is generally true ± a few db for PWM and DAC systems.), A(F) can be taken as the absolute desired attenuation floor and equation 10 can be used to determine the required filter order based on the fundamental stop-band alias, Fs–Fgen(max) (which is typically the most important component to eliminate).

Simple RC stages can be used for applications where order is calculated at 2 or less. However, higher order filters usually require an active design (such as switched capacitor or op-amp based filters) to reduce the pass-band attenuation inherent in passive RC filters.

For most of the firmware examples presented here, these parameters were used:

Fs = 7.812 kHz

Fgmax = 2.6 khz

Fc = 3 kHz

A(Fmas) = 40 db

from equation 10,

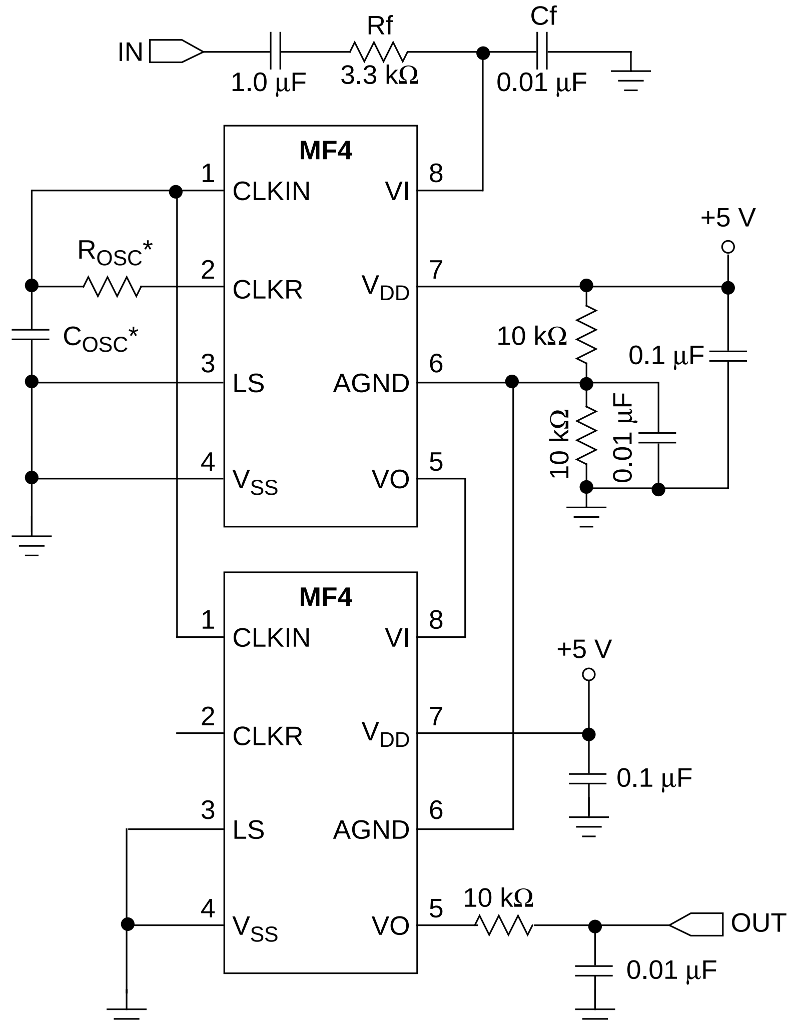
O = A(Fmax) * log(2) / (6 db * log((Fs – Fgmax) / Fc)

= 40 db * log(2) / (6 db * log [(7812 – 2600) / 3000)]

= 8.36

Thus, an eighth order filter would ensure that the stop-band aliases would be better than 40 db below the fundamental. The most effective filter method for higher order designs is a switched capacitor filter such as the MF-4. These devices allow relatively high filter orders with few parts.

The schematic of **Figure 4** shows an eighth order filter with RC input and output filters (needed to remove high frequency noise) for a total filter order of 10, or about 60db/octave. This is the reconstruction filter used with the all of the following examples.



* R_{OSC} and C_{OSC} set Fc

Figure 4. Example Filter Based on the MF-4 Switched Capacitor Building Clock

One of the results of equation 10 (with respect to the primary stop-band alias, Fs - Fgen) is that the filter order can be reduced by increasing Fs.

If Fs from the previous example is increased to 31.2 kHz:

Fs = 31.2 kHz

Fgmax = 2.6 kHz

Fc = 3 kHz

A(Fmax) = 40 db

O = A(Fmax) * log(2) / (6 db * log ((Fs-Fgmax) / Fc)

= 40 db * log(2) / (6 db * log((31200 – 2600) / 3000))

= 2.05

Thus, by simply increasing the sample rate by a factor of 4, the two MF-4s in the example filter can be eliminated. This greatly reduces the filter cost.

Sine Table

Each of the following examples uses a unique sine table. While some effort was made to keep the examples consistent, subtle variations from one MCU implementation to the next can impact the data contained in the sine table. Most of this variation is due to PWM latencies in some of the implementations. The D/A code used also can have a drastic impact on the composition of the sine table (a codec versus a linear D/A, for example).

In general, all of the examples presented here follow the same basic format: The sine table varies between a min and max binary value with a mid-point (or 0) reference that lies at:

D/A(0) = min + ((max – min) / 2)

Thus, all of the tones generated will have a DC offset. Since min and max typically are close to 0 and 255, respectively, the 0 reference will generally be close to D/A (255) / 2.

Since buffered PWM and direct D/A systems generally don't exhibit latency problems, the examples here use a sine table that varies from 1 to 255 (or 0 to 254 for the HC12 PWM) with the 0 reference at 128.

However, unbuffered PWM systems can have min/max values that are not so straightforward and require a different sine table. The C program in **Sine Table Generator C Program** illustrates a simple method of generating a generic sine table given minimum, maximum, and number of entries and formats it for assembly as an include file.

Tone Generator Algorithm

Each of the D/A examples to follow are shaped by the subtleties of the particular MCUs chosen for this application note. However, the central tone generator algorithm is substantially similar in all cases. Some MCUs require more memory and/or execution time to code and execute, but they all perform the same tasks in the same fashion to generate the sine wave signal. **Figure 5** illustrates the flowchart for this algorithm which is the basis for all of the following examples.

The flow chart has two basic variations. **Figure 5A** is for non-buffered systems and uses a temporary holding register for the D/A value. The previously calculated D/A is loaded from the temporary register at the start of the interrupt and immediately transferred to the PWM duty cycle register.

In **Figure 5B**, for buffered systems, this value can be stored as soon as it is calculated.

HC05 Family

Two different PWM modules are available in the HC05 Family. The HC05B16, HC05B32, and HC05X32 variants have a simple PLM module that can provide an 8-bit PWM output at one of two rates, fast and slow.

At maximum MCU clock rates, the fast mode allows only a 1.95-kHz PWM rate, which limits the utility of tone synthesis since the maximum allowed tone frequency would be only Fs / 2 = 975 Hz. Still, this might prove useful in several applications, especially in the generation of CTCSS tones. (The highest CTCSS tone is approximately 250 Hz.)

Another HC05 variant, the MC4, has a more flexible PWM module which can generate buffered PWM at rates of up to about 24 kHz and is buffered.

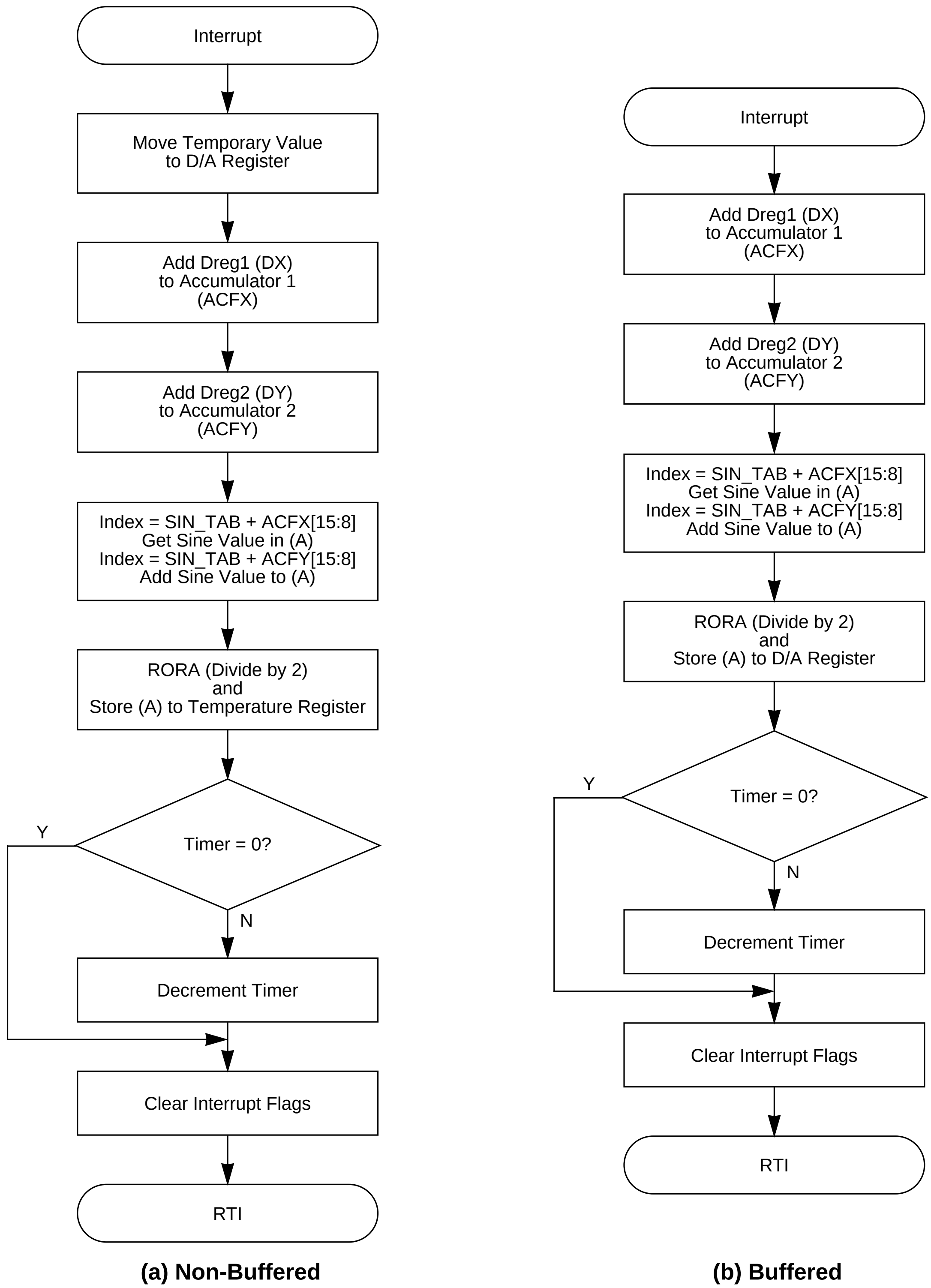


Figure 5. Tone Generator Interrupt Service Flowchart

HC05 PLM

Since the PLM system is not buffered, a crude yet effective technique is used to provide a synchronous interrupt to service the tone generator algorithm. The PWM output is simply connected to one of the input captures which is then configured for falling edge operation. This configuration is effective, but care must be taken to ensure that the PWM avoid 0 percent and 100 percent duty cycles. The PLM does not allow a 100 percent duty cycle, but 0 percent is achievable and must be avoided. If 0 percent is generated by the PLM, the output is a steady logic 0, which effectively disables the tone interrupt. The easiest method to address this situation is to code the sine table so that the min value is at least 1.

NOTE: *It should be noted that, due to interrupt latency, the full 8-bit dynamic range of the PLM is not available.*

The amount of degradation is determined by the interrupt latency, and the amount of time it takes for the interrupt routine to write a new D/A value to the PWM duty cycle register. Because of this requirement, the flowchart of **Figure 5A** is used for this example. Since the PLM rate is so low, the MCU latency does not significantly impact the sine table min value. The interrupt latency is 10 cycles, plus a maximum instruction latency of 11 cycles, plus seven cycles of transfer latency equals 28 cycles of latency. However, at a 1.95-kHz PLM rate, it takes four MCU cycles for every PLM counter tick, so the minimum PLM duty cycle is latency / 4 = 7.

HC05MC4 PWM

The MC4 implementation is similar to that of the PLM version in that an input capture is used to source the tone generator interrupt service routine. The MC4 PWM setup is somewhat more complicated in that it offers several features that are targeted at motor applications. For this application, however, we simply want a buffered PWM at a single port pin, which is easily configured as shown in **MC4 PWM**. Since the PWM is buffered, the D2A temp register that was used in the PLM version can be eliminated and the new D/A value can be written directly to the duty cycle register (PWMAD).

HC08 Buffered PWM

The HC08 PWM module offers a buffered mode by linking two PWM duty cycle registers. Application firmware must track which register was last written to maintain the buffered operation, but this is easily accomplished with a simple counter which is incremented each time a duty cycle register is written. Bit 0 of this counter is used to select which duty cycle register is to be written during any particular interrupt cycle. Since the HC08 PWM uses timer overflow to operate its PWM, it serves as the obvious choice to source the interrupt which drives the tone generator service routine.

HC11 Synchronous PWM

While there are HC11 variants with PWM modules, this example uses two output compares to generate the PWM signal and is thus applicable to all HC11 variants. It is synchronous because the update operation is integrated into the OC interrupt which forces the update to be synchronized with the start of the PWM cycle. However, since the operation is not buffered, dynamic range is affected by response latency (**Figure 5A** applies).

On the HC11, only one output compare, OC1, can affect any of the OC port pins. All other output compares are tied to a dedicated pin so that the selection of the second OC is tied to a port pin selection and vice versa. For this example, OC1 generates the main interrupt and sets the PWM port pin (PA6) while OC2 clears the port pin.

As illustrated in **Figure 6**, the OC1 interrupt routine sets both the OC2 and OC1 time-outs and updates the D/A value to be used for the next cycle.

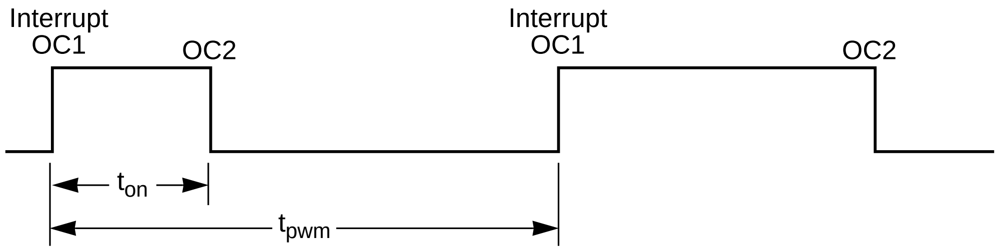


Figure 6. OC1 and OC2 PWM Timings

As indicated in **HC11 PWM Listing**, the OC1 interrupt requires 29 MCU cycles to stack the registers and update the OC2 timer, which dictates the minimum pulse width. Proper use of the WAI instruction (which pre-stacks the registers on the HC11) can save up to 11 cycles. (WAI takes 14 cycles: 11 cycles to stack registers, plus 3 cycles to fetch the interrupt vector.) Since the vector fetch comes after the interrupt, it gets counted as latency in this example, which reduces the minimum pulse width to 29 – 11 = 18 cycles. The only restriction on the high end of duty cycle is that the OC2 time-out be less than (for instance, occur prior to) the OC1 time-out value.

NOTE: *The interrupt latency does not account for the instruction that is executing at the time of the interrupt.*

For applications where WAI can not be used or guaranteed, the wide variation in instruction cycles can make the latency calculation a difficult task. Worst case instruction latency would add an additional 41 cycles (IDIV and FDIV) but this can be an excessive step as these instructions are not encountered often in real applications. If the IDIV and FDIV instructions are not used, the figure can be reduced to 10 cycles which will cover all of the remaining instructions while only adding a moderate degree of overhead to the PWM duty cycle.

The following equations determine the critical design constants:

TSAMP = PWM cycle time (cycles)
= (XTAL / 4) / Fsamp
= E / Fsamp

TMIN = minimum pulse width (cycles)
= Tint_resp + Tinstr + Toc2_update + 1
= 14 + 10 + 15 + 1 = 40 (no WAI)
= 3 + 0 + 15 + 1 = 19 (guaranteed WAI)

TMAX = maximum pulse width (cycles)
= TSAMP – 1

RANGE = # discrete steps from min to max
= TMAX – TMIN

DUTY = duty cycle (%)
= D2A / TSAMP

For this example, a 9.83-MHz crystal was used which gives the following values. The 8-MHz case is also shown.

Table 3. HC11 Design Examples

9.83-MHz Crystal	8-MHz Crystal
TSAMP = (9.8304E6/4) / 7.812 kHz = 314cycles TMIN = 40 cycles (worst case) TMAX = 313 cycles RANGE = 313 – 40 = 273	TSAMP = (8E6/4) / 7.812 kHz = 256 cycles TMIN = 40 cycles (worst case) TMAX = 255 cycles RANGE = 255 – 40 = 215

While this example limits the maximum "on" time to eight bits, or 255 timer cycles, the above calculations indicate that greater than eight bits of dynamic range are possible for E > 2.32 MHz (for Fsamp as shown). If maximum dynamic range is of importance and the MCU oscillator design will allow higher crystal frequencies to be selected, the excess RANGE value can be used to absorb the latency figure. This is done by adding the latency into the updated TOC2 value at the end of the OC1 interrupt routine. This method would add nine cycles to the length of the interrupt routine, but would allow a full 8-bit D/A implementation. In this case, the sine table could be calculated to swing from 1 to 255.

HC12 Buffered PWM

For this example (see **HC12 PWM Listing**), the HC12 PWM is operated in 8-bit buffered mode. The original design used an output compare interrupt to update the PWM where the OC period was an integer multiple of the PWM period. However, this design exhibited noise problems at high values of PWDT0 and the system was re-worked to follow the HC05 PLM case where the PWM drives an input capture. (PP0 is connected to TC7 as a falling edge triggered interrupt.) For the HC12 PWM module, the duty cycle ranges from 1 / 256 to 256 / 256 for values of PWDT0 that range from 0 to 255. Since the input capture system cannot tolerate duty cycles of 0 percent or 100 percent, these values must be eliminated from the sine table, thus the HC12 PWM sine table should range from 0 to 254 for proper operation.

One difference worthy of note in the HC12 allows the reduction in the length of the sine table. In systems where memory must be conserved, the addition of the linear interpolate instruction, TBL, can greatly reduce the size of the 256-byte sine table of the previous examples without seriously impacting signal quality. A reduction in N by a factor of 4 or 8 (64- or 32-byte sine length) can be achieved by using the fractional portion of the phase accumulators to supply the interpolation operator used by the TBL instruction. This is a direct extension of the indexing principal defined for the phase accumulators. If the integer portion of the accumulator determines the position in the sine table, the fractional portion determines the fractional phase distance to the next entry.

To keep the system parameters the same as the 256-byte case (same Fstep, Fsamp, Fgen, etc.), the decimal radix for Dreg and the phase accumulators are moved up rather than reducing the range of the integer portion. Since the interpolate operation has the effect of "filling in" the "missing" table entries, the position of the radix is chosen to yield an effective table length of 256 (which simply allows the same Dreg values to be used).

This is accomplished by moving the radix in proportion to the factor of reduction in table length. If the table is divided by a factor of 2^x, then the radix is moved up "x" bits. The example in **Interpolated Table Lookup** uses a 32-byte table, which is a factor of 2^3 reduction, thus moving the radix to lie between bits 10 and 11. Shift instructions are used to byte align the radix when extracting the table index and interpolate values.

Direct D/A

A direct D/A interface is a worthwhile alternative to PWM methods in those situations where PWM is not suitable and the additional cost is justified. (See **HC12 DAC Listing**.) Signal-to-noise improvements can be achieved over most PWM methods, and system clock frequencies can be reduced in some cases to reduce power consumption. There are several well documented methods that can be employed for direct D/A; for this reason, the discussion here focuses on the importance of timing in writing the D/A value to the D/A sub-system.

As mentioned earlier regarding PWM systems, buffered operation is preferred over non-buffered due to the way in which changes in the duty cycle (for instance, new D/A values) are synchronized to the sample clock. This is also important in direct D/A sub-systems because a statistical variation of even a single CPU clock cycle can result in significant noise in the output. For interrupt-driven systems, instruction latencies introduced in the interrupt dispatch can easily account for several CPU cycles of variation in the timing of the D/A update. A simple mechanism for precisely controlling the D/A update is needed.

The simplest approach is to use the WAI (or WAIT, depending on the processor source form) instruction to ensure that the CPU has been configured in anticipation of the coming interrupt. Once the wait instruction is complete, the subsequent interrupt response latency will be consistent for each iteration of the interrupt.

This approach has two basic difficulties:

1. The designer must make sure that a wait instruction is executed prior to each and every interrupt. While this is relatively straightforward for simple systems, it may not be feasible to maintain for more complicated systems, especially if interrupt recursion is used.
2. Other interrupt sources may disrupt the D/A update process which dictates, in general, that other interrupts must be disabled during tone generation.

Another approach requires the addition of a latch and the use of an output compare signal to latch the new value into the D/A after the interrupt firmware has written the D/A update. The output compare will then be synchronized to the CPU clock with no excessive firmware maintenance issues. As long as the tone generator interrupt can be adequately serviced, the D/A latch can be precisely synchronized to the CPU clock. The external latch approach also allows I/O (input/output) expansion to reclaim the bits used to drive the D/A for other I/O functions.

This method is illustrated in **Figure 7**. The DAC0832 is designed to interface to a processor bus and features a built-in double-buffered latch. One interface signal (~WR) latches the initial write, while another interface signal (~XFER) transfers the latched data to the D/A.

An output compare signal drives the ~XFER signal which assures that the data is always presented to the D/A at the exact sample point relative to the previous sample period.

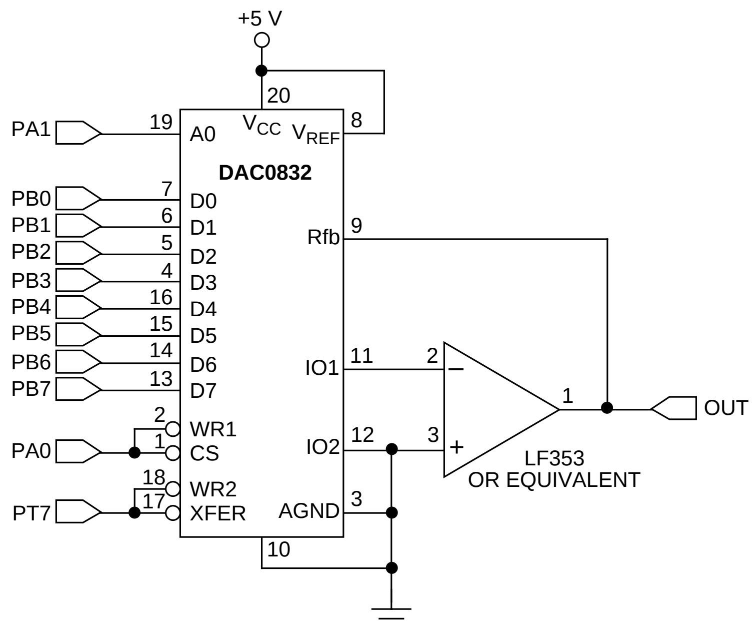


Figure 7. DAC MCU Connections

The output compare also serves as the tone generator interrupt source as it occurs at the sample rate. Once the interrupt is processed, the code clears the XFER signal and updates the phase accumulators. The updated values are then used to calculate the new D/A value which is then written to the D/A port which arms the D/A transfer mechanism. When the next output compare is issued, the D/A will transfer the value previously written and repeat the procedure.

DTMF and Call-Progress Tones

TELCO and wireless applications are two areas which make wide use of DTMF and call-progress signaling. Both DTMF and call-progress signaling systems make use of dual tones to signify a unique system state.

Tone Definitions

Table 4 lists the tone formats for the various signaling states.

Table 4. TDMF and Call Progress Frequency List

State Description	High Tone	Low Tone (Hz)	Fs = 7.812 kHz	
			High Tone Dreg (Decimal)	Low Tone Dreg (Decimal
Dial tone	440 ±5%	350 ±0.5%	3691	2936
Busy *	620 ±5%	480 ±0.5%	5201	4026
Ringback *	480 ±5%	440 ±0.5%	4026	3691
Note: All DTMFs ±0.5%				
DTMF "1"	1209 ±5%	697 ±5%	10142	5847
DTMF "2"	1336	697	11207	5847
DTMF "3"	1477	697	12316	5847
DTMF "4"	1209	770	10142	6459
DTMF "5"	1336	770	11207	6459
DTMF "6"	1477	770	12316	6459
DTMF "7"	1209	852	10142	7147
DTMF "8"	1336	852	11207	7147
DTMF "9"	1477	852	12316	7147
DTMF "0"	1336	941	11207	7894
DTMF "*"	1209	941	10142	7894
DTMF "#"	1477	941	12316	7894
DTMF "A"	1633	697	13698	5847
DTMF "B"	1633	770	13698	6459
DTMF "C"	1633	852	13698	7147
DTMF "D"	1633	941	13698	7894

* Busy tone cycles on/off at 0.5 s/0.5s, ringback tone cycles on/off at 2 s / 4 s.

To calculate the absolute frequency tolerance one must take the lowest frequency in the table, 350 Hz, and apply equation 6:

Fstep(max) = Fmin * %error
= 350 * 0.005
= 1.75 Hz

All of the examples presented here meet this Fstep specification with no difficulty (although the HC05 PWM example would not be able to generate the DTMF tones due to its limitation on Fs).

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
752 Hz	7	8	9	C
941 Hz	*	0	#	D

Figure 8. Standard DTMF Keypad Layout and Frequency Matrix

Due to the legacy of the original Bell Telephone DTMF keypad layout, it is still common to depict the DTMF row/column format as shown in **Figure 8**. This layout is helpful in that the intersecting rows and columns correspond to the frequencies of each signal. A binary "2 of 8" code is often used to represent DTMF digits as the row and column frequencies can be easily extracted. In the 2 of 8 code, four bits are used to represent the 16 DTMF signals. The upper two bits specify the row frequency, while the lower two bits specify the column frequency as illustrated in **Table 3**.

Table 5. ASCII to 2 of 8 Conversion Matrix

2 of 8	ASCII
0000	1
0001	2
0010	3
0011	A
0100	4
0101	5
0110	6
0111	B
1000	7
1001	8
1010	9
1011	C
1100	*
1101	0
1110	#
1111	D

Sample TELCO Routines

TELCO Subroutines shows the HC11/HC12 routines that are used to demonstrate the DTMF and call-progress tones. The main subroutine is DTMFstr which takes an EOL ($0D) terminated ASCII string and converts it to the DTMF equivalents for each tone using ASCdtmf. The constants "toneon" and "toneoff" specify the on and off timings for the DTMF signals and are shown at their typical values in this listing (40 ms on/off).

ASCdtmf converts the ASCII character in (A) to a 2 of 8 code using the ordered ASC_T look-up table. The 2 of 8 code is then used to access the DTMFlo and DTMFhi look-up tables to extract the desired Dreg values which are copied to the DX and DY registers. Lastly, the ASCdtmf uses the tontimer to time the on and off portions of the tone before exiting.

Since most of the MCU execution time is spent waiting for tontimer to count down to 0, these loops can contain a JSR to a system polling subroutine to perform non-critical real-time system functions. As long as the polling routine takes less than (1 / Fs) – Tinterrupt, the system throughput will not be impacted inversely.

The call-progress tones are generated by CPsub. The tone generated is determined by the contents of the (A) register upon entry into the routine. (A) = "D" generates a dial tone, (A) = "B" generates a busy tone, while (A) = "R" generates a ringback tone. All of the call progress tones continue until an SCI character is detected. In a real-world application, an I/O signal and/or timer combination likely would be used to terminate these tones.

Conclusion

The techniques described herein demonstrate the feasibility of implementing a sine-wave-based tone generation system on a variety of Motorola microcontroller families. By using interrupts to synchronize the tone generation algorithm, the system may be integrated easily in to any system without having to re-calibrate machine cycles in timing loops. The interrupt nature of the system also allows for real-time I/O service for application specific functions. This allows a wide variety of tone signaling protocols to be supported easily with a minimum of code and data overhead.

Listings

HC05 PWM Listings

HC05 PLM

Setup:

```
                ; init pwm (PLMA)
                ; NOTE: MOR must select /1 clock prescale

0401    B60C    LDA    MISC
0403    A4F5    AND    #$FF^(SFA|SM)        ; set pwm period = fast
0405    B70C    STA     MISC                ; = 1.92 kHz @ X = 8 Mhz
0407    A680    LDA     #$80                ; preset @50% duty
0409    B70A    STA     PLMA

                ; init IC1

040B    B612    LDA     TCR
040D    AA82    ORA     #ICIE|IEDG1
040F    B712    STA     TCR

0411    9A      CLI
```

Interrupt service:

```
                ;  icii traps PLM edges to synch the PWM update
                ;  fsamp rate is determined by PLM period ...
                ;  new SIN_TAB pointers are calculated for next
                ;  sample period.  D2A is < 8 bits due to
                ;  response latency of IC interrupt.

0430    B65A    icii   LDA     D2A
0432    B70A    STA    PLMA                 ; update PLM
0434    B613    LDA    TSR                  ; clear interrupt flags
0436    B615    LDA    TIC1L
0438    B61D    LDA    TIC2L
043A    B651    LDA    DX+1                 ; do accum for tone 1
043C    BB57    ADD     ACFX+1
043E    B757    STA     ACFX+1
0440    B650    LDA     DX
0442    B956    ADC     ACFX
0444    B756    STA     ACFX
0446    B653    LDA     DY+1                ; do accum for tone 2
0448    BB59    ADD     ACFY+1
044A    B759    STA     ACFY+1
044C    B652    LDA     DY
044E    B958    ADC     ACFY
```

```
0450    B758         STA     ACFY
0452    BE56         LDX     ACFX                 ; lookup tone 1
0454    D60474       LDA     SIN_TAB,X
0457    BE58         LDX     ACFY                 ; lookup tone 2
0459    DB0474       ADD     SIN_TAB,X
045C    46           RORA                         ; div by 2 to get 8 bits
045D    B75A         STA     D2A                  ; store for next update
045F    B654         LDA     tontimer             ; update duration count
0461    2604         BNE     loop4                ; done,
0463    B655         LDA     tontimer+1
0465    270C         BEQ     icix                 ; done,
0467    B655       loop4   LDA     tontimer+1     ; tontimer--
0469    A001         SUB    #$01
046B    B755         STA    tontimer+1
046D    B654         LDA    tontimer
046F    A200         SBC    #$00
0471    B754         STA    tontimer
0473    80           icix   RTI
```

MC4 PWM

Setup:

```
                     ; init pwm

0101    A641         LDA     #CSA1+POLA           ; enable pwm1
0103    B714         STA     CTLA
0105    A690         LDA     #9*10                ; set 7.8 kHz pwm rate
0107    B716         STA     RATE
0109    A680         LDA     #$80                 ; preset D/A @ zero
010B    B710         STA     PWMAD

                     ; init IC1

010D    A682         LDA     #ICIE2|IEDG2         ; ic2 on, rising edge
0111    B717         STA     TCR
```

Interrupt service:

```
                     ; ic1ii traps PLM edges to synch the PWM update
                     ;  fsamp rate is determined by PWM period ... new
                     ;  SIN_TAB pointers are calculated for next
                     ;  sample period.

0132    B618       ic1ii   LDA     TSR            ; clear int flags
0134    B61C         LDA    TIC1L
0136    B61A         LDA    TIC2L
0138    B651         LDA    DX+1                  ; do accum for tone 1
013A    BB57         ADD    ACFX+1
013C    B757         STA    ACFX+1
013E    B650         LDA    DX
0140    B956         ADC    ACFX
```

```
0142    B756        STA    ACFX
0144    B653        LDA    DY+1                     ; do accum for tone 2
0146    BB59        ADD    ACFY+1
0148    B759        STA    ACFY+1
014A    B652        LDA    DY
014C    B958        ADC    ACFY
014E    B758        STA    ACFY
0150    BE56        LDX    ACFX                    ; lookup tone 1
0152    D60172      LDA    SIN_TAB,X
0155    BE58        LDX    ACFY                    ; lookup tone 2
0157    DB0172      ADD    SIN_TAB,X
015A    46          RORA                           ; div by 2 to get 8 bits
015B    B710        STA    PWMAD                   ; store to d/a
015D    B654        LDA    tontimer                ; update duration count
015F    2604        BNE    loop4                   ; done,
0161    B655        LDA    tontimer+1
0163    270C        BEQ    icix                    ; done
0165    B655      loop4    LDA     tontimer+1      ; tontimer--
0167    A001        SUB    #$01
0169    B755        STA    tontimer+1
016B    B654        LDA    tontimer
016D    A200        SBC    #$00
016F    B754        STA    tontimer
0171    80        icix     RTI
```

HC08 PWM Listing

Setup:

```
                 ; init pwm

6E07    B620     LDA     TSC                    ; stop timer
6E09    AA30     ORA     #TSTOP|TRST
6E0B    B720     STA     TSC
6E0D    4500FF   LDHX    #pwper                 ; set pwm period
6E10    3524     STHX    TMOD
6E12    450080   LDHX    #$0080                 ; init duty cycle
6E15    3527     STHX    TCH0
6E17    A601     LDA     #1                     ; init tracking register
6E19    B75A     STA     track
6E1B    A62A     LDA     #MS0B|TOV0|ELS0B
6E1D    B726     STA     TSC0                   ; init ch1 = buffered
6E1F    B620     LDA     TSC                    ; stop timer
6E21    A4DF     AND     #$FF^TSTOP
6E23    AA40     ORA     #TOIE
6E25    B720     STA     TSC
6E27    9A       CLI
```

Interrupt service:

```
                 ;  tovi sets the fsamp rate and calculates new
                 ;  SIN_TAB pointers for next sample period.  D2A
                 ;  is 8 bits only!

6E44    B620     tovi   LDA     TSC             ; clear int flag
6E46    A47F     AND    #$FF^TOF
6E48    B720     STA    TSC
6E4A    B651     LDA    DX+1                    ; do accum for tone 1
6E4C    BB57     ADD    ACFX+1
6E4E    B757     STA    ACFX+1
6E50    B650     LDA    DX
6E52    B956     ADC    ACFX
6E54    B756     STA    ACFX
6E56    B653     LDA    DY+1                    ; do accum for tone 2
6E58    BB59     ADD    ACFY+1
6E5A    B759     STA    ACFY+1
6E5C    B652     LDA    DY
6E5E    B958     ADC    ACFY
6E60    B758     STA    ACFY
6E62    8C       CLRH
6E63    BE56     LDX    ACFX                    ; lookup tone 1
6E65    D66E84   LDA    SIN_TAB,X
6E68    8C       CLRH
6E69    BE58     LDX    ACFY                    ; lookup tone 2
6E6B    DB6E84   ADD    SIN_TAB,X
6E6E    46       RORA                           ; div by 2 to get 8 bits
6E6F    450028   LDHX   #TCH0L                  ; test which pwm to write
6E72    015A03   BRCLR  0,track,loop3           ; is ch0,
6E75    45002B   LDHX   #TCH1L                  ; switch to ch1
6E78    F7      loop3   STA     ,X              ; set for next cycle
6E79    3C5A     INC    track                   ; update tracking reg
6E7B    5554     LDHX   tontimer                ; update duration count
6E7D    2704     BEQ    loop4                   ; done,
6E7F    AFFF     AIX    #-1t                    ; x--
6E81    3554     STHX   tontimer
6E83    80       loop4  RTI
```

HC11 PWM Listing

```
Setup:

                ; TON enables oc1 tone generator

        8064   8640TON LDAA   #OC1M6               ; oc1 sets PA6
        8066   B7100C  STAA   OC1M
        8069   8640    LDAA   #OC1D6
        806B   B7100D  STAA   OC1D
        806E   B61020  LDAA   TCTL1                ; oc2 clears PA6
        8071   843F    ANDA   #~(OM2|OL2)
        8073   8A80    ORAA   #OM2
        8075   B71020  STAA   TCTL1
        8078   FC100E  LDD    TCNT                 ; init oc1 rate
        807B   C30133  ADDD   #TSAMP
        807E   FD1016  STD    TOC1
        8081   961E    LDAA   TMIN                 ; init d2a
        8083   9708    STAA   D2A
        8085   FC100E  LDD    TCNT                 ; preset OC2 near bottom
        8088   D31E    ADDD   TMIN
        808A   D31E    ADDD   TMIN
        808C   FD1018  STD    TOC2
        808F   86C0    LDAA   #OC1F|OC2F           ; pre-clear oc flags
        8091   B71023  STAA   TFLG1
        8094   B61022  LDAA   TMSK1                ; enable oc1 interrupt
        8097   8A80    ORAA   #OC1F
        8099   B71022  STAA   TMSK1
        809C   39             RTS
                ;
                ;
                ; TOFF disables oc1 tone generator

        809D   B61022  TOFF    LDAA TMSK1          ; disable oc1 interrupt
        80A0   847F    ANDA   #~OC1F
        80A2   B71022  STAA   TMSK1
        80A5   7F100C  CLR    OC1M                 ; disconnect timer pins
        80A8   B61020  LDAA   TCTL1
        80AB   843F    ANDA   #~(OM2|OL2)
        80AD   B71020  STAA   TCTL1
        80B0   39      RTS
```

Interrupt service:

```
                ; OC1II handles oc1 interrupts by setting fsamp
                ;  pace and calculating new SIN_TAB pointers for
                ;  next sample period. D2A is 8bits only! Cycle
                ;  times assume DIR addressing for non-MCU
                ;  locs, & EXT addressing for all other locs.

                                                   ;~14 for interrupt
       80B1   DC08    OC1I    LDD D2A              ;~4 get pwm from last d/a
       80B3   F31016 ADDD    TOC1                  ;~6
       80B6   FD1018 STD     TOC2                  ;~5
                                                   ;  ~~= ~14 + ~15 = ~29

       80B9   FC1016 LDD     TOC1                  ;~5 set pwm rate
       80BC   C30133 ADDD    #TSAMP                ;~4
       80BF   FD1016 STD     TOC1                  ;~5
       80C2   DC00   LDD     DX                    ;~4 do accum for tone 1
       80C4   D304   ADDD    ACFX                  ;~5
       80C6   DD04   STD     ACFX                  ;~4
       80C8   DC02   LDD     DY                    ;~4 do accum for tone 2
       80CA   D306   ADDD    ACFY                  ;~5
       80CC   DD06   STD     ACFY                  ;~4
       80CE   CE80F3 LDX     #SIN_TAB              ;~3 lookup tone 1
       80D1   D604   LDAB    ACFX                  ;~3
       80D3   3A     ABX                           ;~3
       80D4   A600   LDAA    0,X                   ;~4
       80D6   CE80F3 LDX     #SIN_TAB              ;~3 lookup tone 2
       80D9   D606   LDAB    ACFY                  ;~3
       80DB   3A     ABX                           ;~3
       80DC   AB00   ADDA    0,X                   ;~4 add to 1st tone
       80DE   46     RORA                          ;~2 div by 2 to get 8 bits

                     IF      BIT8                  ; slower method (8 bit d/a)
       80DF   16     TAB                           ;~2
       80E0   4F     CLRA                          ;~2
       80E1   C3001E ADDD    #TMIN                 ;~4 add TMIN to d/a
       80E4   DD08   STD     D2A                   ;~4 save for next sample

                     ELSE                          ; quick method( < 8 bit d/a)
                     ENDIF

       80E6   DE0A   LDX     tontimer              ;~5 update tone duration
       80E8   2703   BEQ     :03                   ;~3 done,
       80EA   09     DEX                           ;~3
       80EB   DF0A   STX     tontimer              ;~5
       80ED   86C0:03LDAA    #OC1F|OC2F            ;~2
       80EF   B71023 STAA    TFLG1                 ;~4
       80F2   3B     RTI                           ;~12
                                                   ;  ~~ = 134 (BIT8 = false)
                                                   ;  ~~ = 143 (BIT8 = true)
```

HC12 PWM Listing

Setup:

```
                                          ; timer inits

0820   8600    LDAA    #0                 ; TC7 = IC
0822   5A80    STAA    TIOS
0824   8680    LDAA    #EDG7B             ; falling edge
0826   5A8A    STAA    TCTL3
0828   8680    LDAA    #TEN               ; enable timer
082A   5A86    STAA    TSCR
082C   8608    LDAA    #TCRE
082E   5A8D    STAA    TMSK2
0830   8680    LDAA    #C7I
0832   5A8C    STAA    TMSK1
0834   CC0871  LDD     #tc7ii             ; init the interrupt vector
0837   7C0B20  STD     tc7vec

                                          ; init pwm channel 0

083A 8600      LDAA    #0                 ; 32 kHz sample rate ;PCKA1
083C   5A40    STAA    PWCLK              ; separate PWMs, /1 prescale
083E   790041  CLR     PWPOL              ; clock A for PWM0
0841   790054  CLR     PWCTL              ;non-center,PWM runs in wait
0844   86FF    LDAA    #255
0846   5A4C    STAA    PWPER0             ; set pulse period
                                          ; = (chA period) * (255 + 1)
                                          ; = 1/E * 256
                                          ; = 32 µS (31.25 kHz) @ E = 8 MHz
                                          ; this is exactly 4x Fsamp
0848   8601    LDAA    #PWEN0             ; enable PWM0
084A   5A42    STAA    PWEN
084C   CC0000  LDD     #0
084F   8680    LDAA    #$80
0851   5A50    STAA    PWDTY0             ; init d/a register

0853   10EF    CLI
```

Normal Table Lookup

```
Interrupt service:

                     ; tc7ii sets the fsamp rate and calculates new
                     ;  SIN_TAB pointers for next sample period.
                                                  ;~9 for interrupt
      0871   8680 tc7iiLDAA  #C7F                 ;~1
      0873   5A8E   STAA    TFLG1                 ;~3
      0875   FC0800 LDD     DX                    ;~3 do accum for tone 1
      0878   F30806 ADDD    ACFX                  ;~3
      087B   7C0806 STD     ACFX                  ;~2
      087E   FC0802 LDD     DY                    ;~3 do accum for tone 2
      0881   F30808 ADDD    ACFY                  ;~3
      0884   7C0808 STD     ACFY                  ;~2
      0887   CE0D00 LDX     #SIN_TAB              ;~2 lookup tone 1
      088A   F60806 LDAB    ACFX                  ;~3
      088D   1AE5   ABX                           ;~2
      088F   A600   LDAA    0,X                   ;~3
      0891   CE0D00 LDX     #SIN_TAB              ;~2 lookup tone 2
      0894   F60808 LDAB    ACFY                  ;~3
      0897   1AE5   ABX                           ;~2
      0899   AB00   ADDA    0,X                   ;~3 add to 1st tone
      089B   46     RORA                          ;~1 div by 2 to get 8 bits
      089C   5A50   STAA    PWDTY0                ;~3 save to d/a
      089E   FE0804 LDX     tontimer              ;~3 update tone duration?
      08A1   2704   BEQ     L3                    ;~3 no, done,
      08A3   09     DEX                           ;~1 decrement tone timer
      08A4   7E0804 STX     tontimer              ;~2
      08A7   0B L3  RTI                           ;~8
                                                  ;  ~~ = 70
```

Interpolated Table Lookup

```
Interrupt service:

                     ; tc7ii sets the fsamp rate and calculates new
                     ; SIN_TAB pointers for next sample period.
                                                  ;~9 for interrupt
      0D4E   8680 tc7ii LDAA #C7F                 ;~1
      0D50   5A8E    STAA    TFLG1                ;~3
      0D52   FC0800  LDD     DX                   ;~3 do accum for tone 1
      0D55   F30806  ADDD    ACFX                 ;~3
      0D58   7C0806  STD     ACFX                 ;~2
      0D5B   FC0802  LDD     DY                   ;~3 do accum for tone 2
      0D5E   F30808  ADDD    ACFY                 ;~3
      0D61   7C0808  STD     ACFY                 ;~2
```

```
                   ; interpolate lookup goes here
                   ; INCLUDE  "LOOKUP.ASM"    ;~20
                     INCLUDE "INTERP.ASM"     ;~45
                   ; interpolate table lookup code
                   ;  Adds 25 MCU cycles over standard version
                   ;  uses 32 byte sine table.
0D64   FC0806 LDD     ACFX                    ;~3 move radix (tone 1)
0D67   49     LSRD                            ;~1
0D68   49     LSRD                            ;~1
0D69   49     LSRD                            ;~1
0D6A   B781   EXG     A,B                     ;~1 calculate table address
0D6C   CE0D9B LDX     #SIN_TAB                ;~2
0D6F   1AE5   ABX                             ;~2
0D71   B781   EXG     A,B                     ;~1 B = fractional phase
0D73   183D00 TBL     0,X                     ;~8 interpolate
0D76   7A080A STAA    temp                    ;~2
0D79   FC0808 LDD     ACFY                    ;~3 move radix (tone 2)
0D7C   49     LSRD                            ;~1
0D7D   49     LSRD                            ;~1
0D7E   49     LSRD                            ;~1
0D7F   B781   EXG     A,B                     ;~1 calculate table address
0D81   CE0D9B LDX     #SIN_TAB                ;~2
0D84   1AE5   ABX                             ;~2
0D86   B781   EXG     A,B                     ;~1 B = fractional phase
0D88   183D00 TBL     0,X                     ;~8 interpolate
0D8B   BB080A ADDA    temp                    ;~3 add to 1st tone
                                              ; ~~ = 45
              ; end of lookup, (A) = PWM value
0D8E   46     RORA                            ;~1 div by 2 to get 8 bits
0D8F   5A50   STAA    PWDTY0                  ;~3 save to d/a
0D91   FE0804 LDX     tontimer                ;~3 update tone duration?
0D94   2704   BEQ     L8                      ;~3 no, done,
0D96   09     DEX                             ;~1 decrement tone timer
0D97   7E0804 STX     tontimer                ;~2
0D9A   0B L8  RTI                             ;~8
              ;  ~~ = 70 (~95 for interpolate version)
              ;
```

HC12 DAC Listing

```
Setup:

                                        ; timer inits

0820   8680   LDAA    #IOS7
0822   5A80   STAA    TIOS
0824   8680   LDAA    #TEN
0826   5A86   STAA    TSCR
0828   8608   LDAA    #TCRE
082A   5A8D   STAA    TMSK2
082C   8680   LDAA    #OM7
082E   5A88   STAA    TCTL1

0830   8680   LDAA    #C7I
0832   5A8C   STAA    TMSK1
0834   CC0400 LDD     #1024             ; 7.8125 kHz fsamp
0837   5C9E   STD     TC7
0839   CC0870 LDD     #tc7ii
083C   7C0B20 STD     tc7vec

                                        ; init DAC port I/O

083F   86FF   LDAA    #$FF
0841   5AAF   STAA    DDRT
0843   86FF   LDAA    #$FF
0845   5A03   STAA    DDRB
0847   86FF   LDAA    #$FF
0849   5A02   STAA    DDRA
084B   8606   LDAA    #$06
084D   5A00   STAA    PORTA
084F   8680   LDAA    #DACXFR
0851   5AAE   STAA    PORTT

0853 10EF     CLI
```

Interrupt service:

```
                        ;  tc7ii sets the fsamp rate and calculates new
                        ;  SIN_TAB pointers for next sample period.
                        ;  D2A is 8 bits only!

086C   8680 tc7iiLDAA    #C7F
086E   5A8E   STAA    TFLG1
0870   86C0   LDAA    #OM7+OL7             ; reset XFER pin to "1"
0872   5A88   STAA    TCTL1
0874   8680   LDAA    #FOC7
0876   5A81   STAA    CFORC
0878   A7     NOP
0879   8680   LDAA    #OM7
087B   5A88   STAA    TCTL1
087D   FC0800 LDD     DX                   ; do accum for tone 1
0880   F30806 ADDD    ACFX
0883   7C0806 STD     ACFX
0886   FC0802 LDD     DY                   ; do accum for tone 2
0889   F30808 ADDD    ACFY
088C   7C0808 STD     ACFY
088F   CE0D00 LDX     #SIN_TAB             ; lookup tone 1
0892   F60806 LDAB    ACFX
0895   1AE5   ABX
0897   A600   LDAA    0,X
0899   CE0D00 LDX     #SIN_TAB             ; lookup tone 2
089C   F60808 LDAB    ACFY
089F   1AE5   ABX
08A1   AB00   ADDA    0,X
08A3   46     RORA                         ; div by 2 to get 8 bits
08A4   5A01   STAA    PORTB                ; write data to port
08A6   84FD   ANDA    #$FD                 ; strobe write
08A8   5A00   STAA    PORTA
08AA   A7     NOP
08AB   A7     NOP
08AC   8A02   ORAA    #DACS
08AE   5A00   STAA    PORTA
08B0   FE0804 LDX     tontimer             ; update tone duration
08B3   2704   BEQ     L3                   ; done,
08B5   09     DEX
08B6   7E0804 STX     tontimer
08B9   0B L3  RTI
```

TELCO Subroutines

```
0F42   halfsecEQU     3906                 ; = time * Fsamp
04B0   toneon EQU     1200                 ; = time * Fsamp
0258   toneoffEQU     600                  ; = time * Fsamp
000D   EOL    EQU     $0D                  ; end of line
0B78   dialow EQU     2936                 ; dial low tone
0E6B   dialhi EQU     3691                 ; dial high tone
0032   ringcount EQU   50                  ; max # ring cycles
0E6B   ringlowEQU     3691                 ; ring low tone
0FBA   ringhi EQU     4026                 ; ring high tone
3D09   ringon EQU     15625                ; ring ton = 2 s
7A12   ringoffEQU     31250                ; ring toff = 4 s
0032   busycount EQU   50                  ; max # busy cycles
0FBA   busylowEQU     4026                 ; busy low tone
1451   busyhi EQU     5201                 ; busy high tone
0F42   busyon EQU     3906                 ; busy ton = 0.5 s
0F42   busyoffEQU     3906                 ; busy toff = 0.5 s
       ;
       ;

0823   doapp

       ; The following demonstration code sends the test_str
       ;  string as DTMF signals. Tone on/off times are
       ;  40ms/40ms.  Any SCI character received aborts

0823   TESTDTMF
0823   CC0000 LDD     #0
0826   7C0806 STD     ACFX                 ; clear phase accumulator
0829   7C0808 STD     ACFY                 ; clear phase accumulator
082C   7C0804 STD     tontimer             ; clear tone timer
082F   7C0800 STD     DX                   ; init tone 1 (off)
0832   7C0802 STD     DY                   ; init tone 2 (off)

                                           ; send a dial tone

0835   8644   LDAA    #'D'
0837   1608E9 JSR     CPsub

                                           ; send some DTMFs

083A   CE085C LDX     #test_str            ; get test string
083D   072B   BSR     DTMFstr              ; send it

                                           ; send ring back

083F   8652   LDAA    #'R'
0841   1608E9 JSR     CPsub

```

```
                                               ; send busy

      0844   8642   LDAA    #'B'
      0846   1608E9 JSR     CPsub

      0849   CC0000 LDD     #0
      084C   7C0800 STD     DX
      084F   7C0802 STD     DY
      0852   86C0   LDAA    #$C0
      0854   7A0806 STAA    ACFX
      0857   7A0808 STAA    ACFY
      085A   20FE L1BRA     L1                     ; stop execution


 085C 392C35353537test_str  FCB                    "9,5557579,,,,",EOL

                    ;  DTMFstr sends the string pointed at (X) as
                    ;  DTMF digits until EOL is detected.
                    ;  USES: A,B,X,Y

      086A   A600 DTMFstr  LDAA    0,X             ; get string character
      086C   08     INX
      086D   810D   CMPA    #EOL                   ; end of string?
      086F   2704   BEQ     L2                     ; yes,
      0871   0703   BSR     ASCdtmf                ; send the tone
      0873   24F5   BCC     DTMFstr                ; no interrupt,
      0875   3D  L2 RTS
                    ;
                    ;
             ;  ASCdtmf converts the ASCII in (A) to DTMF frequencies in DX
             ;  & DY and times t-on.  NON-DTMF characters result in a 0.5
             ;  sec pause.  DTMF characters are: {0-9}, {A-D}, {*}, and {#}
             ;  USES: A,B,Y

      0876   C7 ASCdtmf  CLRB
      0877   CD08C9 LDY     #ASC_T                 ; init table index
      087A   A140 L3CMPA    0,Y                    ; in table?
      087C   270B   BEQ     gotASC                 ; yes,
      087E   02     INY
      087F   52     INCB
      0880   C10F   CMPB    #maxDTMF               ; end of table?
      0882   23F6   BLS     L3                     ; no,
      0884   CC0F42 LDD     #halfsec               ; delay 1/2 sec
      0887   2034   BRA     waitone

      0889   37 gotASC PSHB                        ; save for later
      088A   C403   ANDB    #$03                   ; mask hi tone
      088C   58     LSLB                           ; construct index (*2)
      088D   CD08E1 LDY     #DTMFhi
      0890   19ED   ABY
      0892   ED40   LDY     0,Y                    ; get hi tone
```

```
0894   7D0800 STY     DX
0897   33     PULB
0898   C40C   ANDB    #$0C                 ; mask low tone
089A   54     LSRB                         ; = hinyb * 2
089B   CD08D9 LDY     #DTMFlo
089E   19ED   ABY
08A0   ED40   LDY     0,Y                  ; get low tone
08A2   7D0802 STY     DY
08A5   CC04B0 LDD     #toneon              ; set on time
08A8   7C0804 STD     tontimer
08AB   3E L4  WAI
08AC   FC0804 LDD     tontimer
08AF   26FA   BNE     L4                   ; not done yet,
08B1   CC0000 LDD     #0                   ; tones off
08B4   7C0800 STD     DX
08B7   7C0802 STD     DY
08BA   CC0258 LDD     #toneoff             ; set off time
08BD   7C0804 waitone  STD     tontimer
08C0   3E L5  WAI
08C1   FC0804 LDD     tontimer
08C4   26FA   BNE     L5                   ; not done yet,
08C6   10FE   CLC
08C8   3D     RTS

                                            ; Table of ASCII DTMF digits

08C9 313233413435 ASC_T  FCB               "123A456B789C*0#D"
000F          maxDTMF   EQU                15

              ;  table of high tones for each DTMF character.
              ;  Tone values calculated from:
              ;  D = (Fgen * 65536) / Fsamp = Fgen * 8.3886

08D9   16D5 DTMFlo   FDB     5845          ; 697 Hz
08DB   193B   FDB    6459                  ; 770 Hz
08DD   1BEB   FDB    7147                  ; 852 Hz
08DF   1ED6   FDB    7894                  ; 941 Hz

08E1   279E   DTMFhi FDB     10142         ; 1209 Hz
08E3   2BC7   FDB    11207                 ; 1336 Hz
08E5   3066   FDB    12390                 ; 1477 Hz
08E7   3583   FDB    13699                 ; 1633 Hz
              ;
              ;
              ;
              ; CPsub uses (A) to select one of the following call
              ; progress tone pairs:
              ; (A) Signal state
              ; "D" dial tone (8 sec max)
              ; "R" ring back tone (100 rings)
              ; "B" Busy tone (50 burst cycles)
```

```
                        ; USES: A,B,Y

        08E9   D6C4  CPsub    LDAB   SC0SR1         ; preclear sci
        08EB   D6C7           LDAB    SC0DRL
        08ED   8144           CMPA    #'D'          ; dial tone?
        08EF   262E           BNE     nodial        ; no,
        08F1   CC0B78         LDD     #dialow       ; set tones
        08F4   7C0800         STD     DX
        08F7   CC0E6B         LDD     #dialhi
        08FA   7C0802         STD     DY
        08FD   CCFFFF         LDD     #$FFFF        ; set maximum duration
        0900   7C0804         STD     tontimer
        0903   3E  waitall    WAI
        0904   96C4           LDAA    SC0SR1
        0906   8520           BITA    #RDRF
        0908   1401           SEC                   ; preset SCI detect flag
        090A   2607           BNE     killall       ; got an SCI chr,
        090C   FC0804         LDD     tontimer
        090F   26F2           BNE     waitall       ; keep a' goin'...
        0911   10FE           CLC                   ; clear SCI detect flag
        0913   96C7  killall  LDAA    SC0DRL
        0915   CC0000 LDD     #0                    ; turn off tones
        0918   7C0800 STD     DX
        091B   7C0802 STD     DY
        091E   3D     RTS

        091F   8152 nodial CMPA #'R'                ; ring-back tone?
        0921   262D   BNE     noring                ; no,
        0923   8632   LDAA    #ringcount            ; set ring counter
        0925   7A080B STAA    count
        0928   CC0E6B  ringlp LDD    #ringlow       ; set tones
        092B   7C0800 STD     DX
        092E   CC0FBA LDD     #ringhi
        0931   7C0802 STD     DY
        0934   CC3D09 LDD     #ringon               ; set ring on time
        0937   7C0804 STD     tontimer
        093A   07C7   BSR     waitall               ; wait...
        093C   2511   BCS     CPexit                ; got an SCI, quit
        093E   CC7A12 LDD     #ringoff              ; set ring off time
        0941   7C0804 STD     tontimer
        0944   07BD   BSR     waitall               ; wait again...
        0946   2507   BCS     CPexit                ; got an SCI, quit
        0948   73080B DEC     count                 ; done 'em all yet?
        094B   26DB   BNE     ringlp                ; no,
        094D   10FE   CLC                           ; no SCI detected
        094F 3D CPexit RTS

        0950   10FE  noring   CLC                   ; preclear SCI detect
        0952   8142   CMPA    #'B'                  ; busy tone?
        0954   26F9   BNE     CPexit                ; no,
        0956   8632   LDAA    #busycount            ; set ring counter
```

```
0958   7A080B STAA    count
095B CC0FBA busylp LDD  #busylow          ; set tones
095E   7C0800 STD     DX
0961   CC1451  LDD     #busyhi
0964   7C0802 STD     DY
0967   CC0F42 LDD     #busyon             ; set ring on time
096A   7C0804 STD     tontimer
096D   0794   BSR     waitall             ; wait...
096F   25DE   BCS     CPexit              ; got an SCI, quit
0971   CC0F42 LDD     #busyoff            ; set ring off time
0974   7C0804 STD     tontimer
0977   078A   BSR     waitall             ; wait again...
0979   25D4   BCS     CPexit              ; got an SCI, quit
097B   73080B DEC     count               ; done 'em all yet?
097E   26DB   BNE     busylp              ; no,
0980   10FE   CLC                         ; no SCI detected
0982   3D     RTS
```

Sine Table Generator C Program

```
#include <stdio.h>
#include <math.h>

// This program constructs a sine table as specified by the user.
// min, max, and size are provided at run time with the output
// going to the display and a file named "SINE.ASM."
//
// Table entries are defined by the following:
// sin,x = int(MIDP + (swing * SIN (360 * x / 256)))
//
// where x = table offset

FILE *fi;
float max = 255;
float min = 1;
float size = 256;
const float pie = 3.141592654;
float x, y, MIDP, SWING, t;

void main(void)

{
```

```
printf("Sine table compiler, v1.00\n");
printf("Sending output to \"SINE.ASM\"...\n");
if ( (fi = fopen("SINE.ASM", "w")) != NULL)
        {

        // get table parameters

        printf("Enter table size (256 max): ");
        scanf("%f", &size);
        printf("Enter table min value (0-255): ");
        scanf("%f", &min);
        printf("Enter table max value (0-255): ");
        scanf("%f", &max);
        SWING = (max - min) / 2;
        MIDP = min + SWING;

        // put descriptor header in .asm file

        printf("; sine lookup table\n");
        fprintf(fi, "; sine lookup table\n");
        printf("; size = %5.0f, min = %5.0f, max = %5.0f \n", size, min, max);
        fprintf(fi,";size = %5.0f,min = %5.0f,max = %5.0f \n",size,min, max);
        printf("; MID = %f   SWING = %f\n",MIDP,SWING);
        fprintf(fi, "; MID = %f   SWING = %f\n",MIDP,SWING);
        fprintf("SIN_TAB\n");// place table lable

        // put table data as assembly source.

        x = 0;
        while (x <= size)
          {
          y = MIDP + (SWING * (sin (2 * pie * x / size)));
          printf("\tFCB\t");// display source
          printf("%5.0f",y);
          printf("\n");
          fprintf(fi, "\tFCB\t");// write source file
          fprintf(fi, "%5.0f",y);// for casm0x use "%5.0ft"
          fprintf(fi, "\n");
          x++;
          }
        fclose(fi);
        printf("Done.\n");
        }
else
        {
        printf("File error.\n");
        }
}
```

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution, P.O. Box 5405, Denver, Colorado 80217, 1-800-441-2447 or 1-303-675-2140. Customer Focus Center, 1-800-521-6274

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 141, 4-32-1 Nishi-Gotanda, Shinagawa-ku, Tokyo, Japan. 03-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/;
TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: http://motorola.com/sps/

MOTOROLA