MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

Order this document by: AN1254/D

Using the MC68HC16Z1 for Audio Tone Generation

By Scott Howard

INTRODUCTION

There are many applications where a microcontroller is required to generate audio-frequency tones as part of a product's function. Audio tones can be used to communicate data, interact with the user of the product, or to perform other functions. **Table 1** shows typical applications.

Table 1 Tone Generation Applications

Application	How Used
Security Systems	Communication between system and remote monitoring site
Telephone Products	DTMF (Dual Tone Multi Frequency) transmits digits to central office MF (Multi Frequency) used between central offices
Instrumentation and Data Acquisition	Data transmission and remote control
MODEMs	Transmitting data over the telephone network between computers

Audio tones may be simple, such as a square wave produced by toggling a single output bit, or very complex, such as computer-generated music. Most microcontroller applications involve waveforms of low to moderate complexity. Typical tones consist of square waves, sine waves, or other arbitrary patterns such as triangles and ramp waveforms, as well as combinations of these.

It is feasible and in fact quite simple to generate audio tones in software using a microcontroller. But, since there are a number of analog integrated circuits available that can perform this function, why use a microcontroller?

There are two important reasons for considering a software approach:

1. The microcontroller is already part of the product, and hardware costs can be reduced elsewhere in the design by using tone-generating software
2. Software offers flexibility which is unavailable (or expensive) if implemented in hardware; e.g., output waveforms, frequencies, and output levels can be changed easily in software

This application note examines generating arbitrary waveform using software techniques, and shows how to generate DTMF tones used on the public switched telephone network.

Source code for software discussed in this note is available from Motorola Freeware Data Systems. For modem access to the Freeware BBS, dial (512) 891-3733. For Internet access, use freeware@mot.sps.com. For WWW access, use http://freeware.aus.sps.mot.com/.

© MOTOROLA INC, 1996

HOW TONE GENERATION IS DONE

A table of data representing the output waveform is stored in memory. A pointer is used to access the table. Initial pointer value is the address of the first data point. Hardware and software are set up to generate waveform samples at a constant rate. During each sample period, data is read from the address specified by the pointer and sent to the output hardware. The pointer value is then incremented and compared to the last address in the table. When the pointer is incremented past the end of the table, the initial value is restored.

In **Figure 1**, a table of 18 samples in memory represent a sine wave. If one sample is output each millisecond through a digital to analog converter (DAC), then a sine wave of 55.6 Hz (1000 ÷18) is generated.

To generalize, the output frequency is calculated as follows:

$$F_{out} = \frac{\text{Sample Rate in Hz}}{\text{Samples per Wave}}$$

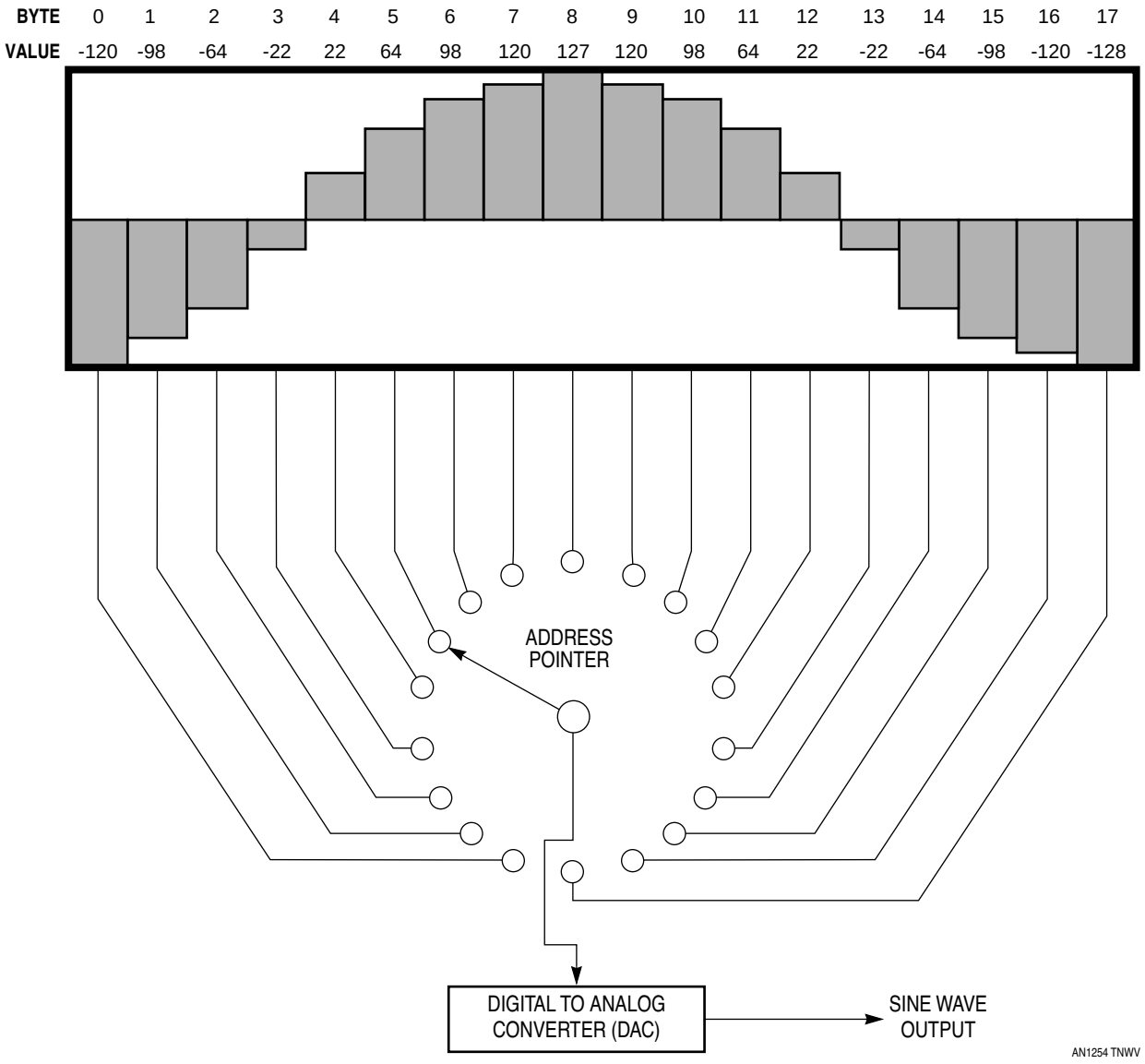

Figure 1 Waveform Generation

Tone Generation Code Example

CPU16 code to implement tone generation could look like this:

```
               PSCT                        Switch to ROM section
INIT:          LDX      #TABLE             point to start of table
               STX      POINTER            save in memory
               JSR      SET_TIMER          set up GPT hardware to generate
*                                          an interrupt every millisecond

* The following routine is called by the timer interrupt

ISR:           PSHM     X,D                save X and D registers on stack
               JSR      RESET_TIMER        set up timer for next interrupt
               LDX      POINTER
               LDAA     0,X                Get next sample
               STAA     DAC                write to D/A Converter
               AIX      #1                 step to next address in table
               CPX      #TABLE_END         stepped past end of table yet?
               BNE      ISR_1              branch if not
               LDX      #TABLE             else reset pointer to start
ISR_1:         STX      POINTER            save new pointer back to memory
               PULM     D,X                restore CPU registers from stack
               RTI                         done

*              Table of sine wave samples in memory

TABLE:         FCB      -120,-98,-64,-22
               FCB      22,64,98,120,127
               FCB      120,98,64,22
               FCB      -22,-64,-98,-120,-128
TABLE_END:     EQU      *

               DSCT                        Switch to RAM section
POINTER        RMB      2                  reserve memory to store pointer
```

QUESTIONS AND ANSWERS

How Can The Waveform Be Changed?

Any arbitrary waveform may be generated by changing the values stored in the memory table. This is one of the great advantages of this method of digital waveform generation.

Can Multiple Signals Be Generated?

Multiple signals can be generated by performing multiple table lookup operations, adding the samples, and sending the sum to the DAC. Different tables can be used for each lookup operation, or a single table can be used, to save space. Waves of different frequencies can be generated from the same memory table using the techniques discussed below.

Do not allow results of addition to overflow. For example, if the two samples retrieved from the 8-bit table shown in Figure 1 were both 126, adding them would cause accumulator overflow, and the result stored in the DAC would be –2. To avoid this, software must scale the samples before adding them. In the example, both samples must be divided by two, so that the value 126 is changed to 63, and the sum becomes the true value of 126.

Can A Waveform Be Generated Without Using A Lookup Table?

A subroutine can be used to generate the data points, but there are significant trade-offs to be considered. A subroutine cannot be changed as easily as a lookup table, and algorithms for some commonly-used waveforms (such as the sine wave in the example) are difficult to implement. However, algorithms for certain other waveforms are very simple: a ramp can be implemented by repetitively incrementing or decrementing a value stored in memory, then sending each new result to the DAC; and white noise can be simulated with a random number generator.

How Can The Amplitude Be Varied?

Amplitude can be varied digitally, by multiplying each sample by a scaling factor before it is sent to the DAC. The CPU16 instruction set includes a number of multiplication instructions, including multiply-and-accumulate (MAC) and repetitive multiply-and-accumulate (RMAC) instructions, that make this type of operation fast and simple. Multiplication can be done as a series of adds and shifts on microcontrollers that do not support multiplication directly.

Amplitude can also be controlled in external hardware, either by the DAC or by analog circuitry further downstream. There are associated hardware costs, but this method may be effective in particular applications.

What Are The Side Effects Of Digital Amplitude Control?

Varying the amplitude digitally also varies the signal to noise ratio (SNR) of the outgoing signal because digital noise remains at a constant level of one LSB, while the amplitude of the outgoing signal varies.

The digital representation is an approximation of the true analog signal, and can have as much as $\pm$ 1/2 LSB of error, or quantization noise. It follows that, the more bits used to represent a signal (i.e., the larger the word), the smaller quantization error and noise are in relation to the full scale value of the signal. This noise decreases by six db for each bit added to the word width — in a16-bit M68HC16 device, the digital noise floor is 96 db down, and the SNR is 96 db at full amplitude. When the signal amplitude is reduced, the noise floor remains constant at –96 db, so the signal to noise ratio is reduced accordingly.

When digital gain control is used, the effective range of gain control is bounded by the specified minimum SNR. For example, if the SNR of the tone output must be at least 40 db, approximately 50 db of gain control can be used without exceeding the specification.If more gain control than can be accommodated by the available word width is needed, then the designer should consider the hardware approach described earlier. For example, if the design were implemented in an 8-bit machine, which has a digital noise floor of –48 db, gain control could not be implemented digitally if a 40 db SNR was needed.

What About Changing The Frequency Of The Output Signal?

Changing output signal frequency is a complex issue. One approach is to alter the sample rate. This is not always the best approach, particularly in the telecommunications arena, where many systems require a fixed sample rate. Even when the system design will accommodate changes in sample rate, the software may rely on constant timing for other functions, thus making rate changes difficult to implement.

One way to change frequency is to store more samples than needed in the table, then skip a specific number of samples for each value sent to the DAC. The number of skipped samples is referred to as the pointer interval. Output frequency can be varied by changing the pointer interval, and sample rate remains constant.

Since output frequency is equal to sample rate divided by the number of samples per wave, skipping a number of table entries for each sample (i.e., interval > 1) has the effect of multiplying output frequency.

$$F_{out} = \frac{\text{Interval} \times \text{Sample Rate}}{\text{Samples per Wave}}$$

If the number of samples in the example in Figure 1 is increased by four (to 72), but four table entries are skipped for each sample output, then the output frequency remains the same. if the pointer interval is changed to 3, then the frequency becomes:

$$F_{out} = \frac{3 \times 1000\ \text{Hz}}{72} = 41.7\ \text{Hz}$$

The output frequency has changed, but the sample rate remains constant. Frequency resolution is found by substituting an interval of 1:

$$F_{interval} = \frac{\text{Sample Rate}}{\text{Table Size}}$$

In this example, the frequency can be controlled in units of 13.9 Hz:

$$F_{interval} = \frac{1000\ \text{Hz}}{72} = 13.9\ \text{Hz}$$

Since frequency is a ratio of sample frequency and table size, increased frequency resolution can be achieved by leaving sample rate constant and increasing sample table size, as shown in **Figure 2**.

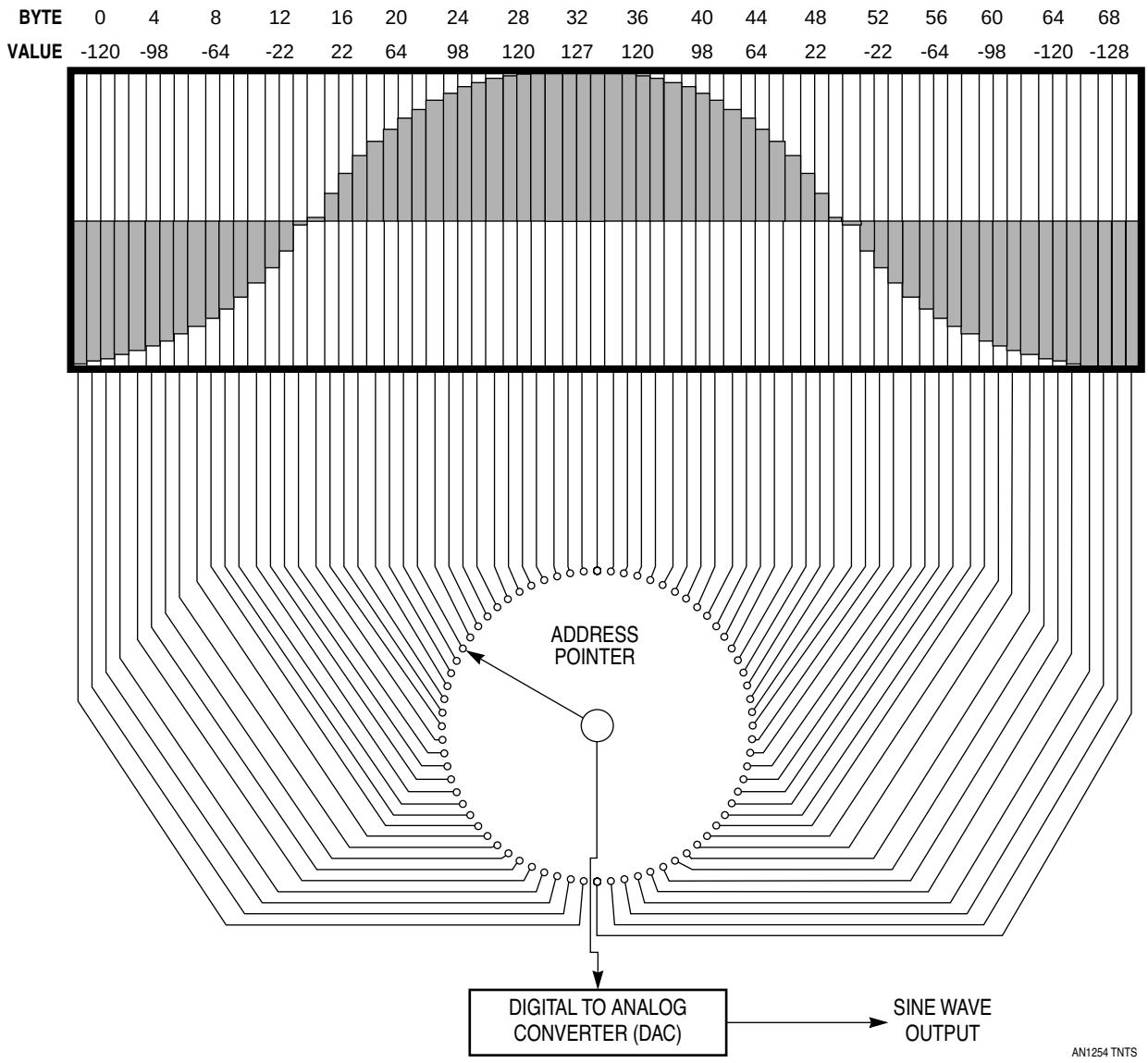

Figure 2 Increasing Table Size increases Frequency Resolution

Frequency Change Code Example

The following code could be used to implement the pointer interval.

```
                PSCT                        Switch to ROM section
INIT:           CLRW    POINTER             initialize pointer
                LDD     #4                  set up initial interval
                STD     INTERVAL
                JSR     SET_TIMER           set up GPT hardware to generate
*                                           an interrupt every millisecond
* The following routine is called by the timer interrupt
ISR:            PSHM    X,D,E               save registers on stack
                JSR     RESET_TIMER         set up timer for next interrupt
                LDX     #TABLE              point to table in memory
                LDE     POINTER
                LDAA    E,X                 Get next sample
                STAA    DAC                 write to D/A Converter
                ADDE    INTERVAL            Add interval to pointer
                CPE     #TABLE_SIZE         stepped past end of table yet?
                BLO     ISR_1               branch if not
                SUBE    #TABLE_SIZE         Reset pointer modulo table size
ISR_1:          STE     POINTER             save new pointer back to memory
                PULM    E,D,X               restore CPU registers from stack
                RTI                         done
* Table of sine wave samples in memory
TABLE:          FCB     -120,-100,-64,-20
                FCB     20,64,100,120,127
                FCB     120,100,64,20
                FCB     -20,-64,-100,-120,-128
TABLE_SIZE:     EQU     *-TABLE

                DSCT                        Switch to RAM section
INTERVAL        RMB     2                   storage for pointer interval
```

What Else Can Be Done With This Technique?

Possibilities include:

- Multiple tones — Performing multiple table lookups, summing the samples, and sthen ending the sum to the DAC.
- Amplitude modulation — Performing two table lookups, multiplying the two values, and then sending the product to the DAC.
- Frequency modulation — Performing two table lookups, then using one lookup value to modulate the interval of the other.

GENERATING DTMF TONES

DTMF (Dual Tone Multi-Frequency) signalling is used to transmit phone numbers on the public telephone network. Generation of DTMF illustrates all of the concepts discussed.

In this encoding scheme, 16 binary digit codes are represented by means of sine wave tone pairs, organized into a high group (1200-1700 Hz) and a low group (600-1000 Hz). There are four tones in each group.

As shown **Table 2**, the tones are associated with a particular row or column on the telephone keypad. Column four is defined but is not usually implemented on a telephone. Signalling is accomplished by transmitting one tone from each group for a minimum of 50 ms, followed by a silent period of at least 50 msec.

Table 2 DTMF Row And Column Frequencies

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

Industry specifications for DTMF generally require frequency errors to be less than 1%, and total harmonic distortion (THD) to be less than 10%. Additionally, the frequency response of the telephone line generally rolls off at high frequencies, requiring the high group of tones to be transmitted at a higher amplitude than the low group. The telephony buzzword for this characteristic is twist.

Software Approach

This example uses a single sine table with two pointers, one for the column tone and one for the row tone. Each pointer has its own interval value, so that different frequencies can be generated.

In order to reduce harmonic content to a minimum, the output must be sampled at a high enough rate to filter out the noise introduced at the sampling frequency. A sample period of 125 μsec was chosen for this example; this is a standard sample rate in the telecommunications industry.

Calculating the Pointer Intervals

The general form of the equation is:

$$F_{out} = \frac{\text{Interval} \times \text{Sample Rate}}{\text{Samples per Wave}}$$

Rearrange the equation to calculate the interval:

$$\text{Interval} = \frac{F_{out} \times \text{Samples per Wave}}{\text{Sample Rate}}$$

The frequency for row 1 on the DTMF keypad is 697 Hz. If a sine wave table of 512 entries is used,

$$\text{Interval} = \frac{697\ \text{Hz} \times 512}{8000\ \text{Hz}} = 44.6$$

Fractional intervals can't be used to step through the table, so round the interval to 45. Plug that number back into the frequency equation, and the actual F_{out} is:

$$F_{out} = \frac{\text{Interval} \times \text{Sample Rate}}{\text{Samples per Wave}} = \frac{45 \times 8000\text{Hz}}{512} = 703\text{Hz}$$

This gives a frequency error of 0.88%, which is acceptable.

When multiple tones based on a single sine wave table are used, performing the calculations can be tedious. A spreadsheet which calculates intervals based on the sine wave table size, sample period, and desired frequency, as well as showing actual frequencies and error, is available through the Freeware system. The file name is DTMF.WKS. DTMF frequencies shown in **Table 3** were calculated using the spreadsheet.

Table 3 DTMF Tone Calculations

Sample Period	125 μΣ			
Sample Table Size	512			
Frequency	697	770	852	941
Sample Interval	45	49	55	60
Actual Frequency	703.13	765.63	859.38	937.50
% Error	0.88	– 0.57	0.87	– 0.37
Frequency	1209	1336	1477	1633
Sample Interval	77	86	95	105
Actual Frequency	1203.13	1343.75	1484.38	1640.63
% Error	– 0.49	0.58	0.50	0.47

Implementing High Tone Pre-Emphasis

In order to compensate for the high-frequency rolloff characteristic of most telephone lines, the high group of tones must be approximately 1 to 3 db higher power than the low group. This equates to an amplitude multiplication of 1.12 to 1.41. Since 0.25 is a binary fraction which can be obtained by shifting instead of using a full multiply, 1.25 is a convenient value to use.

The sine table contains values for the low frequency row tones. When the high group sample is read from the sine table, it is shifted right 2 bits (divide by 4), then the same sine value is added again into the accumulator, producing the 1.25 multiplication. The pre-emphasis is:

$$\text{db} = 20 \log \frac{V1}{V2} = 20 \log (1.25) = 1.94 \text{ db}$$

A CPU16 code sequence to implement pre-emphasis is shown below.

```
ldd     e,y             ;get the sample
asrd                    ;divide by four (column * 0.25)
asrd
addd    e,y             ;add sample again (column * 1.25)
```

Calculating Sine Values to Avoid Overflow

The values in the sine wave table must calculated to avoid overflow errors when the two samples are summed. In this example, 16 bits are used to store the samples, so the maximum data values are +32767 and –32768. To avoid overflow, the values in the table must be between +16383 and –16384. Since the column tones are pre-emphasized, the actual values must be somewhat less than the maximum.

The DAC output value is calculated by

$$\text{MaxOutput} = \text{MaxSineValue} \times (1 + \text{PreEmphasis})$$

To solve for the MaxSineValue, rearrange the equation:

$$\text{MaxSineValue} = \frac{\text{MaxOutput}}{(1 + \text{PreEmphasis})} = \frac{32767}{(1+1.25)} = 14563.11$$

Sine values in the table must vary between ±14563.

A C program called MAKESINE.C which generates a sine table based on user-defined table size and maximum output values is available from Freeware data systems. The output format is compatible with most cross assemblers for Motorola microcontrollers.

Hardware Design

The M68HC16Z1EVB is used to implement the tone generation hardware. As shown in **Figure 3**, GPT PWM channel A is used as an 8-bit DAC, outputting DTMF tones through a low-pass RC filter to an external audio amplifier and speaker. A 3 x 4 keypad is connected to the GPT I/O pins.

A software driver scans the keypad and enables the appropriate DTMF tones when a key press is detected. The driver which performs the sine wave lookup and pointer increment is configured as an interrupt routine, using the GPT Output Compare channel 2 to generate a regular 125 µsec interrupt.

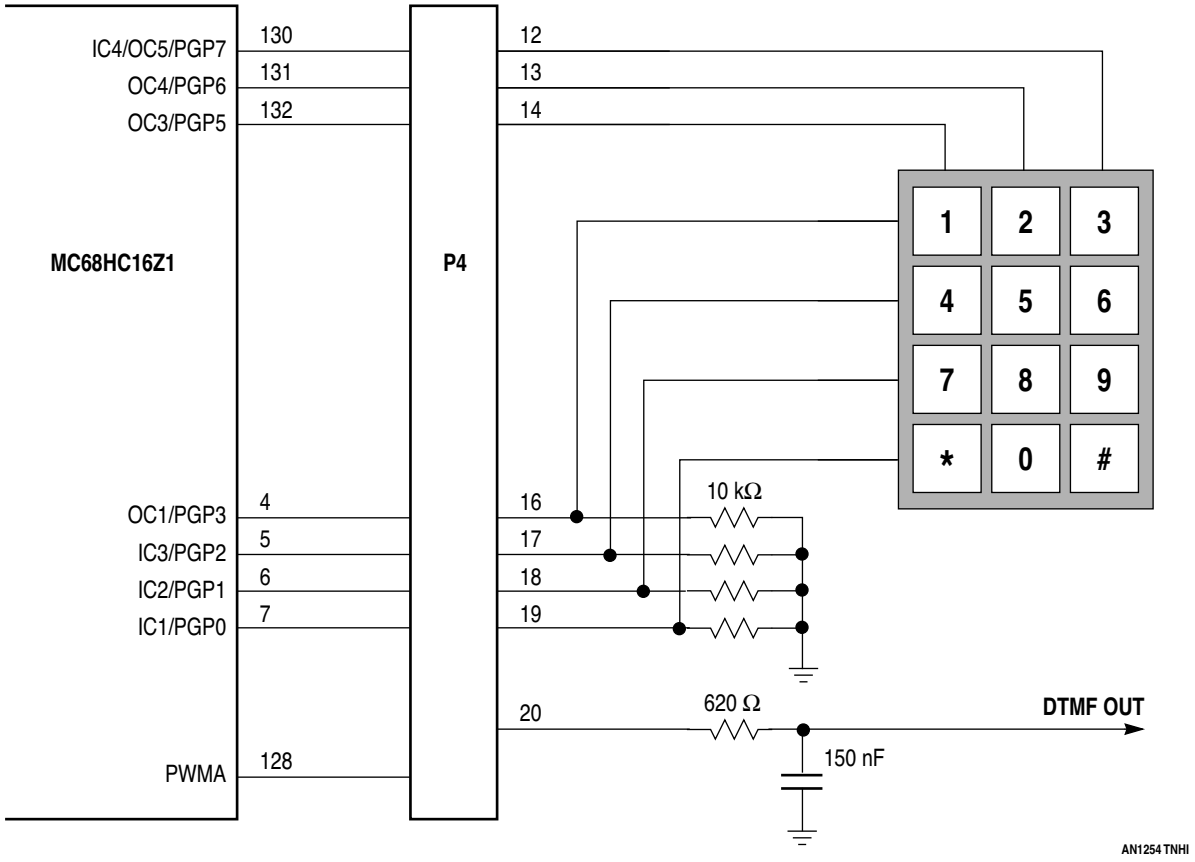

Figure 3 DTMF Hardware Interface

BENCHMARKING THE ASSEMBLY LANGUAGE CODE

Programs that implement the algorithm for M68HC05 and M68HC11 processors, DTMF05.ASM and DTMF11.ASM, are available through Freeware Data Systems. Both use an on-board timer to generate an interrupt at 128 μsec intervals, rather than 125 μsec, which allows them to meet the 1.0% frequency error specification while using a 256-byte sine table.

System performance is shown in **Table 1**. The M68HC05 takes 116 clocks and 464 bytes of code to generate DTMF; the M68HC11 takes 119 clocks and 457 bytes of code. The HC11 requires extra clock cycles to manipulate 16-bit addresses, whereas the HC05 can use byte-sized address calculations applied to a 16-bit offset. The HC05, operating at the standard 2 MHz bus speed, uses 116/256 clock cycles (45%) to service the DTMF interrupt. The HC11 is slightly higher at 46%.

The CPU16 can process the tone interrupt routine in 9.12 μsec, representing an overhead of 7.3%. This is mainly due to increased data transfer capacity provided by the16-bit data bus. Overall, The M68HC16 is approximately 6.5 times faster than the M68HC11 in this application. Increased performance allows the M68HC16 to generate tones at a higher sampling rate than the 8-bit microcontrollers, and the device has enough additional bandwidth to perform amplitude and frequency modulation.

Table 4 Performance results for M68HC05, M68HC11, and M68HC16

Device	Code Size	Table Size	Sample Period	CPU Clock Speed	Interrupt Execution Speed	Execution Time	CPU Bandwidth
M68HC05	208 Bytes	256 Bytes	128 μsec	2.0 MHz	116 Cycles	58.0 μsec	45%
M68HC11	201 Bytes	256 Bytes	128 μsec	2.0 MHz	116 Cycles	59.9 μsec	46%
M68HC16	388 Bytes	512 Bytes	125 μsec	16.78 MHz	152 Cycles	9.12 μsec	7.3%

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters can and do vary in different applications. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. MOTOROLA is a registered trademark of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

TO OBTAIN ADDITIONAL PRODUCT INFORMATION:

USA/EUROPE: Motorola Literature Distribution;
P.O. Box 20912; Phoenix, Arizona 85036. 1-800-441-2447
JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, Toshikatsu Otsuki,
6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 03-3521-8315
HONG KONG: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,
51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298
MFAX: RMFAX0@email.sps.mot.com - TOUCHTONE (602) 244-6609
INTERNET: http://www.mot.com

MOTOROLA