Application Note

DTMF Tone Generation Using the Z86E04 MCU

AN003901-Z8X1199

ZiLOG WORLDWIDE HEADQUARTERS • 910 E. HAMILTON AVENUE • CAMPBELL, CA 95008
TELEPHONE: 408.558.8500 • FAX: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Windows is a registered trademark of Microsoft Corporation.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

Acknowledgements

Project Lead Engineer

Bob Bongiorno

Application and Support Engineer

Joe Rovito

System and Code Development

Bob Bongiorno

DTMF Tone Generation Using the Z86E04 MCU

Dual-Tone, Multi-Frequency (DTMF) tones are required to access telephone lines for communications and data exchange. This application note describes a method for generating DTMF tones using the Z86E04 microcontroller and a minimal amount of support circuitry. Taking advantage of the Z8's efficient table-lookup capabilities and its versatile timers, complex signals can be generated easily without a hardware DAC.

General Overview

DTMF signals are a combination of two sine waves of different frequencies that correspond to a row and column position on the standard telephone keypad. To produce the twelve key tones of this keypad, seven sine-wave frequencies are required. Three frequencies are associated with the column positions and four with the row positions. Each key is associated with a corresponding pair of tones that are produced when the key is selected. For example, assume the 0 key is pressed. The row frequency would be 941 Hz while the column frequency would be 1336 Hz. These two frequencies, or tones, are added together to produce the signal understood at the receiving end to be the 0 key.

The standard telephone keypad and the corresponding DTMF tone assignments are indicated in Table 1.

Table 1. DTMF Tone Assignments

		Columns		
		1209 Hz	1336 Hz	1477 Hz
Rows	697 Hz	1	2	3
	770 Hz	4	5	6
	852 Hz	7	8	9
	941 Hz	•	0	#

Additionally, column frequencies must be 3 dB higher in amplitude than the row frequencies. Because the telephone line acts like a low-pass filter, due to distributed capacitance, the column frequencies are attenuated more so than the row

frequencies. The 3-db column-frequency boost is a compensation for this line characteristic.

Sine Wave Generation

This application note uses a table lookup method for sine wave generation. A sine table is contained in program memory and is accessed by a pointer at a high enough rate (referred to as the Sample rate) to produce a reasonably pure sine wave at the required frequency. According to Nyquist, the Sample rate must be at least twice that of the highest frequency generated. In this case, a sample frequency of at least 2954 Hz is required. To improve accuracy and allow for a simpler, less-costly low-pass filter, a sample rate of 8 kHz is chosen. This sample rate is about five times higher in frequency but is easily achieved with an 8-MHz oscillator frequency. This oscillator frequency is divided by eight, thereby supplying a 1-MHz reference clock to the counter and timers.

The sine table itself was chosen to contain 256 values and was located at an upper byte boundary. This approach simplifies some calculations and makes the concept more straightforward. Only the upper byte of the pointer is used to access the next table value. The sine table itself was generated using a Basic program that selected a wide dynamic range for the sine values. This program produces a DTMF output level of sufficient amplitude for a host of applications.

There are two values that are required for each sine table access. One is the frequency desired, based on the key selected, which is represented as Frequency Increment or `FINCR`. The other is the previous table pointer value, which is represented as Increment or `INCR`. At each periodic interrupt of the sample rate, `FINCR` is added to `INCR` to produce a new `INCR`. This new increment value is a sixteen-bit value, but only the upper byte is used to point to the next sine value in the table.

The formula for the Frequency Increment is as follows:

```
FINCR = (Tablestep x 256 x Frequency) ÷ Sample
```

Where `FINCR` is the 16-bit value, `Tablestep` is the number of values in the sine table, `Frequency` is the desired frequency in hertz, and `Sample` is the sample rate in samples per second.

For this application note, `Tablestep` = 256, `Frequency` is the desired frequency for the DTMF tones, and `Sample = 8000`.

The collection of `FINCR` values is the `offset_table`, arranged for each key as a pair of frequencies.

The equation can be reduced to the relationship:

```
FINCR = 8.192 x Frequency
```

An intuitive way of thinking about sine wave generation is to think of the range of possible frequencies available from a given table. This application note uses 256 locations that represent one cycle of a sine wave. Assume every table location is to be accessed at the 8-kHz sample rate. (Access rate may be a better term than Sample rate. Because the 256-byte table represents one full 360 degree cycle of the sine wave, the frequency produced would be 8000 / 256 or 31.25 Hz. The sine wave quality would be the highest attainable for this system. The value of FINCR would be 256 or 0100h. (Remember that only the upper byte of INCR is used to access the next sine table value.) Because the lower byte of FINCR is '00', the lower byte of INCR always contains 00 after every sum of the two registers.

What frequency is attained for FINCR equal to 0101h or 257? Solving for Frequency, the above equation becomes:

```
Frequency = 0.1221 x FINCR
          = 0.1221 x 257
          = 31.3721 Hz
```

From this equation, each increment of the FINCR yields an increase in frequency of 0.1221 Hz.

For the second case, assume INCR is 0000 to begin. Now the low byte of FINCR contains 01. FINCR and INCR are continuously summed. The lower byte of INCR, again, does not effect the upper byte until the 256th sum. This most recent sum caused a carry into the upper byte. Instead of taking one table step, the 256th access takes two steps. The lower byte is 00 again so another 256 sums are required to take the two-step access.The sequence is one step for 255 accesses, then two steps for one access, continuously. Because there are 8000 samples per second, the table skip occurs 31 times per second. The effect is to increase the time through the table and therefore to increase the frequency by 0.1221 Hz.

Now consider what the highest frequency sine wave would be. According to Nyquist, there must be one point in less than a half-cycle of the sine wave, to describe it in terms of frequency. Assume the first point is accessed at the start of the table. Two additional points are required at just under one- half table intervals. Then, the next point completes the cycle and starts the beginning of the next cycle, just past the start of the table, and so on. Thus the highest frequency sine wave would be about 8000/2 or 4000 hz. The value of FINCR would be 32,768 or 8000h. As the synthesized sine wave goes from the lowest frequency to the highest, purity of the sine wave is compromised for speed, as the number of points describing the waveform is reduced.

DTMF Tone Generation

DTMF tones are specific pairs of sine waves, produced simultaneously, that represent the key positions of a common telephone keypad. Adding the individual

sine waves together produces the DTMF. In this case, the generation of each sine wave is related to a timer value, one for the column and one for the row frequency. Before the column sine value is added to the row sine value, the column value is doubled by shifting left one time. This new value provides the 3 dB edge over the row frequencies. The range of values in the sine table becomes more meaningful because, for the worst case, the sine value may be three times larger before using this value for the 8-bit counter.

PWM Using the Z8 Timers 0 and 1

The DTMF tones are produced by utilizing both Timer 0 and Timer 1 in this application. Timer 0 to sets the rate or frequency of the output, which, in this case, is 8 kH. Timer 0 raises the DTMF output each time its polled interrupt request is set. This same event loads and restarts Timer 1 in a “One-Shot” mode, with vectored interrupt processing. When T1 interrupts, the Interrupt Service Routine (ISR) pulls the DTMF output back down. The value loaded into T1’s counter is the most recently calculated sine value. The effect is that T1 controls the “On-Time” of the signal based on the sine value for each sample period. The duty cycle is based on the length of time T1 requires to count down before its interrupt occurs and pulls the line down for the balance of the T0 time-out. Thus, T0 remains a constant while T1 is re-loaded at the T0 Sample rate.

For simplification, consider the critical points along the sine wave as related to the PWM value. At 0 and 180 degrees, the PWM duty cycle is at 50%. At 90 degrees, the maximum amplitude, the duty cycle is 100%. At 270 degrees, the minimum amplitude, the duty cycle is 0%. To provide some margin for interrupt latency and register load times, the duty cycle minimum and maximum values are somewhat offset from 0 and 100%.

A simple RC network converts digital PWM output to a varying DC signal. This Low-Pass filter was chosen because its corner frequency is the lowest column frequency, or 1209 Hz. At this point, the column frequencies are at least 3 dB below the row frequencies. Telephone specifications require a 3 dB increase for the columns over the rows. By doubling the column sine value, the column receives a 6 dB increase, producing a net increase over the rows of the required 3 dB. The filter, as stated above, is a simple passive filter and may not meet the requirements of some systems. Because the filter is non-buffered, take care when driving external circuits. A high-impedance buffer may be required to isolate the filter from the load. If buffering is required, perhaps a dual or quad op amp is appropriate, because an active filter and buffer amplifier could be used to improve the filter characteristics as well as the output drive capability.

Summary

Many MCU applications require signal generation in the analog domain.This application note focuses on DTMF and sine wave generation, but other signal types are possible. The table method is not restrictive to sine waves. Triangle waves, ramps, pulse trains, and combinations can be synthesized with this approach.

This particular design, as implemented, features several unused inputs that can be utilized for table selection. Selecting from multiple offset tables to generate all sine waves or other frequency DTMF tones becomes a simple matter. The sine table itself can be altered to produce other basic wave shapes. By sampling at a faster rate, higher frequency or higher quality signals can be attained.

The PWM method used to generate analog signals is compelling. Using the Timers in this arrangement provides great flexibility and duty cycle range with very little software overhead with regard to timer register data or control updating. In the case of generating fixed duty cycles, Timer 1 can be used in a continuous mode, requiring an update only when the duty cycle requires changing.

The keyscan routine is simple and modular and easily adapted to various types of keypads.

Using the Z86E04 for the generation of DTMF tones provides a simple and cost effective approach with the additional flexibility a Microcontroller provides. Other standard signaling tones, like ringback and busy, as well as unique signals for non-standard applications, are readily achievable.

Technical Support

Source Code

```
;=============================================================================
;=          TITLE:            DTMF.asm                                       =
;=          DATE:             Started August 1999                            =
;=          PURPOSE:                                                         =
;=                                                                           =
;=          FILE TYPE:        STAND ALONE MODULE                             =
;=                                                                           =
;=          HEADER FILES:     equ.h, offset.s, sine.s                        =
;=                                                                           =
;=          HARDWARE:         NovaTech Z8 Proto. Board ZPCB18                =
;=                                                                           =
;=          ASSEMBLER:        ZiLOG ZDS / ZMASM                              =
;=          PROGRAMMER:       Bob Bongiorno                                  =
;=============================================================================
;
;           RELEASE HISTORY:
;
;     Version     Date        Description
;     1.00        8/15/99     Proto Release
;     1.10        9/15/99     Reasonably Functional Model
;     1.20        10/1/99     Commented and OTP Burned
;
;=============================================================================
;
;*****************************************************************************
;     I/O MAP
;*****************************************************************************
;P00  -->   PWM Output
;P01  -->   unused output, no connection
;P02  -->   unused output, no connection

;P20  -->   Row 0
;P21  -->   Row 1
;P22  -->   Row 2
;P23  -->   Row 3
;P24  -->   Col 0
;P25  -->   Col 1
;P26  -->   Col 2
;P27  -->   Col 3

;P31  -->   unused input, terminate to ground.
;P32  -->   unused input, terminate to ground.
;P33  -->   unused input, terminate to ground.

;*****************************************************************************
;*****************************************************************************
;
      GLOBALS     ON          ;Required for symbol table generation.


      include "equ.h"         ;Equate file.

;=============================================================================
;=    TITLE:      equ.h                                          =
```

```
;=    DATE:       October 1, 1999                          =
;=    PURPOSE:                                             =
;=
;=    FILE TYPE:  Included Header File                     =
;=
;=    HARDWARE:   NovaTech Z8 Proto. Board ZPCB18          =
;=
;=    ASSEMBLER:  ZiLOG ZDS / ZMASM                        =
;=    PROGRAMMER: Bob Bongiorno                            =
;=========================================================================
WORK_REG0   .equ  00h
;
p0                .equ  r0
;not used         .equ  r1
p2                .equ  r2
p3                .equ  r3
bounce            .equ  r4
row_cnt           .equ  r5
key_cnt           .equ  r6
key_temp          .equ  r7
temp_rows         .equ  r8
;not used         .equ  r9
;not used         .equ  r10
;not used         .equ  r11
;not used         .equ  r12
;not used         .equ  r13
;not used         .equ  r14
;not used         .equ  r15
;
;
WORK_REG1         .equ  10h
;
offset_hi         .equ  r0
offset_lo         .equ  r1
offset            .equ  r0
calc_sin_value    .equ  r2
temp              .equ  r3
row_inc_hi        .equ  r4
row_inc_lo        .equ  r5
pointer_hi        .equ  r6
pointer_lo        .equ  r7
pointer           .equ  r6
col_inc_hi        .equ  r8
col_inc_lo        .equ  r9
r_freq_hi         .equ  r10
r_freq_lo         .equ  r11
c_freq_hi         .equ  r12
c_freq_lo         .equ  r13
row_val           .equ  r14
col_val           .equ  r15
;
KEY_CNT           .EQU  06h
KEY_TEMP          .EQU  07h
R_FREQ_HI         .EQU  1Ah
;
xtal              .equ  8000000     ;8 Mhz
sample            .equ  8000        ;8000 samples/sec
ctval             .equxtal/8/sample ;Formula for Timer 0, Counter value.
```

```
tabstp            .equ  256         ;Number of entries in Sine table.

;*****************************************************************************
;             Interrupt Vectors /  Program Memory Address 0000 - 000Bh
;*****************************************************************************

      org   0000h

      .WORD DUMMY       ;IRQ 0 , Not Used
      .WORD DUMMY       ;IRQ 1 , Not Used
      .WORD DUMMY       ;IRQ 2 , Not Used
      .WORD DUMMY       ;IRQ 3 , Not Used
      .WORD KEY_SCAN    ;IRQ 4 , Timer 0 Interrupt
      .WORD TIMER_1     ;IRQ 5 , Timer 1 Interrupt

;*****************************************************************************
;               Start of Executed Code,  000Ch
;*****************************************************************************
;
;*****************************************************************************
;                 Initialization Begins
;*****************************************************************************

DTMF_INIT:
      DI                        ;Disable interrupts.

      CLR   P3M                 ;Port 3 = Digital Inputs, Port 2 = Open Drain
                                ;Outputs.
      LD    P01M,#00000100b     ;Port 0 = Outputs, Stack = Internal (Emulator
                                ;requirement).
      LD    P2M,#00001111b      ;Port 2 Mode: Output = 7,6,5 & 4. Input = 3,2,1 &
                                ;0.

      SRP   #70h                ;Setup to clear all RAM. Load a pointer (r15),
                               ;in working register group 7,to one below its own
      LD    r15,#7Eh            ;address of 7Fh.

CLR_RAM:                        ;This two instruction loop,(4 bytes) clears all
RAM !
      CLR   @r15                ;The DJNZ instruction requires a working regis-
                                ;ter.
      DJNZ  r15,CLR_RAM         ;(Ports 0 and 2 outputs are cleared here as
                                ;well.)
                                ;The last register cleared is r15 itself by the
                                ;last DJNZ !

      DEC   KEY_TEMP            ;Non-key value (FF) for first time through
                                ;KEY_SCAN.

      CLR   IRQ                 ;Clear spurious interrupt requests.
      LD    IMR,#00010000b      ;Allow IRQ 4 (Timer 0) interrupt.
      LD    IPR,#00001011b      ;Set interrupt priority for IRQ5 > IRQ4.

      LD    SPL,#80h            ;Stack = Top of Ram + 1 (Pre-decrementing stack
                                ;pointer).
      CLR   SPH                 ;(A good practice during emulation.)

      LD    PRE0,#00000001b     ;T0 prescaler = 00h for divide by 64 and contin-
```

```
                              ;uous mode.
      CLR   T0                ;T0 counter = 00h for count of 256.
      LD    TMR,#00000011b    ;Load and enable T0 for keyscan interrupts =
                              ;16.384 mSec.

      EI                      ;Enable global interrupts.

;*****************************************************************************
;                Initialization Ends
;*****************************************************************************

;*****************************************************************************
;                Foreground Wait-Loop
;*****************************************************************************

WAIT_HERE:                    ;Program loops here in the foreground
      NOP                     ;until the interrupt for Timer 0 /KEY_SCAN
      NOP                     ;comes along. (User's code could be
      NOP                     ;located here but must be tolerant of periods
      JR    WAIT_HERE         ;of inactivity while the DTMF signal is output.
                              ;Also, any modification of the register pointer
                              ;will require saving and restoring it in the
                              ;following routines.)


;*****************************************************************************
;                KEY_SCAN

;KEY_SCAN is an interrupt service routine driven by Timer 0. KEY_SCAN outputs
the ;column drive on ports 2-4, 2-5, 2-6, and 2-7.  Scan inputs are on ports 2-
0, 2-1, ;2-2 and 2-3. As each column is pulled low, the inputs are checked for
an active ;key and debounced if necessary. Once a key is debounced, DTMF_OUT is
called in ;order to output the appropriate frequencies.

;*****************************************************************************

KEY_SCAN:
      SRP   #WORK_REG0        ;Point to working reg. group 0.
      LD    P2M,#00001111b    ;Set Port 2 for upper nibble outputs,
                              ;lower nibble inputs.
COL_0:
      LD    p2,#11101111b     ;Column 0 driven low.
      CLR   key_cnt           ;Key 0-3.
      CALL  GET_KEY           ;This finds active key if any.
      JR    Z,KEY_FOUND       ;Jump to debounce key.
COL_1:
      LD    p2,#11011111b     ;Column 1 driven low.
      LD    key_cnt,#4        ;Key 4-7
      CALL  GET_KEY           ;
      JR    Z,KEY_FOUND       ;
COL_2:
      LD    p2,#10111111b     ;Column 2 driven low.
      LD    key_cnt,#8        ;Key 8-11
      CALL  GET_KEY           ;
      JR    Z,KEY_FOUND       ;
COL_3:
      LD    p2,#01111111b     ;Column 3 driven low.
      LD    key_cnt,#12       ;Key 12-15
```

```
      CALL  GET_KEY             ;
      JR    NZ,SCAN_EXIT        ;Jump for no active key found.

KEY_FOUND:                      ;Active key found.
      ADD   key_cnt,row_cnt     ;Add row value from GET_KEY to key base value.
      CP    key_cnt,key_temp    ;If key is same,
      JR    Z,KEY_SAME          ;go debounce it.
      CLR   bounce              ;Not same, so clear debounce counter.
      LD    key_temp,key_cnt    ;Make them the same for the next time around.
      JR    SCAN_EXIT           ;Exit with no action taken.
KEY_SAME:
      INC   bounce              ;Debounce counter.
      CP    bounce,#4           ;The same key for 4 reads ?
      JR    NZ,SCAN_EXIT        ;If not, exit with no action taken.
DEBOUNCED:
      LD    key_temp,#0FFh      ;Else debounced so set key_tmp = non-key value
                                ;and
      CLR   bounce              ;clear bounce counter for next key scan.
      JR    DTMF_OUT            ;Go output DTMF for key_cnt value.
SCAN_EXIT:
      LD    p2,#11111111b       ;All columns inactive.
      IRET                      ;Return to WAIT_HERE loop and reenable T0 int.


GET_KEY:
      LD    temp_rows,p2              ;Input port 2 and save in temp_rows.
      AND   temp_rows,#0Fh            ;Clear for Row data only.
ROW_0:
      CLR   row_cnt                   ;Set for Row 0.
      CP    temp_rows,#00001110b      ;Row 0 key ?
      JR    Z,KEY_RET                 ;Yes, return.
ROW_1:
      INC   row_cnt                   ;No. Set for Row 1.
      CP    temp_rows,#00001101b      ;Row 1 key ?
      JR    Z,KEY_RET                 ;Yes, return.
ROW_2:
      INC   row_cnt                   ;No. Set for Row 2.
      CP    temp_rows,#00001011b      ;Row 2 key ?
      JR    Z,KEY_RET                 ;Yes, return.
ROW_3:
      INC   row_cnt                   ;No. Set for Row 3.
      CP    temp_rows,#00000111b      ;Row 3 key ?
KEY_RET:
      RET                             ;Return.

;*******************************************************************************
;
;               DTMF_OUT

;Output DTMF signals based on the key value in key_cnt.
;Stay here till no key is active.
;Timer 0 runs continuously at the sample rate. The DTMF output
;is raised at start of its' cycle.
;T0 is a polled interrupt.
;Timer 1 runs as a one-shot timer whose counter is loaded with the Sine value.
;T1 is a vectored interrupt that pulls the DTMF line back down once per T0
cycle.
;T0 is the PWM frequency, T1 controls the ON time or Duty-Cycle.
```

```
;*******************************************************************************

DTMF_OUT:
      SRP   #WORK_REG1        ;Set up pointer for working register group 1.
      CLR   TMR               ;Stop timers.
      LD    pointer_hi,#>Sine ;Load the Sine table, high byte, base address
                              ;into pointer_hi.

DTMF_OFFSET:
      LD    offset_hi,#>offset_table      ;Load offset_table base address
      LD    offset_lo,#<offset_table      ;into offset register pair.

      RL    KEY_CNT           ;Rotate key_cnt, left twice, to multiply x 4.
      RL    KEY_CNT           ;RL works here because we know that bits 4 to 7
                              ;are always zero with 16 keys.
                              ;Add this value to the offset low byte pointer.
      ADD   offset_lo,KEY_CNT ;Each key corresponds to 4 successive entries
                              ;in the table.
      ;The upper byte, offset_hi, is 01h because the table is
      ;located at '01C0'h. The max key_cnt after shifting is 3Ch.
      ;(0000 1111 > 0011 1100)
      ;Adding to C0h yields FCh. Because no carry is generated,
      ;we don't have to increment offset_hi, it's always 01h.

      ;Each key is a designated row and column frequency pair.
      ;Get the table increment values based upon the current key.
      ;We need a row frequency word and column frequency word.
      ;These values are the step size for "walking through"
      ;the Sine table.  The larger the step, the higher the frequency.

      LD    temp,#R_FREQ_HI   ;Set indirect pointer to r_freq_hi register
address.
      LDCI  @temp,@offset     ;r_freq_hi
      LDCI  @temp,@offset     ;r_freq_lo
      LDCI  @temp,@offset     ;c_freq_hi
      LDCI  @temp,@offset     ;c_freq_lo

      ;Column and row frequency offsets are now in place.


      ;Set-up Timer 0 and 1 to prepare for PWM / DTMF signal generation on
      ;port 0-0.

DTMF_TIMERS:
      LD    PRE0,#00000101b   ;T0 prescaler = 1, continuous mode.
      LD    T0,#ctval         ;T0 is loaded with 7Dh for 8 khz sample rate (8
Mhz
                              ;xtal).
      LD    TMR,#00000011b    ;Load and enable T0 for the first time.

      LD    PRE1,#00000110b         ;T1 prescaler = 1, one shot mode.
      LD    calc_sin_value,#20      ;Load a dummy value for the first time.
      LD    IMR,#00100000b          ;Allow vectored T1 interrupt only.
      CLR   IRQ                     ;Clear all pending ints.
      EI                            ;Reenable global ints.

DTMF_LOOP:
      OR    P0,#00000001b           ;Raise DTMF output line, port 0-0.
```

```
      LD    T1,calc_sin_value       ;T1 counter loaded = last calc. value.
      OR    TMR,#00001100b          ;Start T1 (One shot mode.)

      TCM   P2,#00001111b           ;Check for any key still active.
      JR    Z,DTMF_EXIT             ;No active key, exit. Else, continue DTMF
                                    ;out.

;At some point Timer 1's interrupt will come along and set the output low.
;In the meantime, we are calculating Timer 1's counter value for the NEXT pass
thru
;the DTMF_LOOP.
;The DTMF signal is the sum of 2 sine waves.  The T1 value, or duty cycle, is
;changed 8000 times /sec. CALCULATE_PWM executes at this rate in order to
determine ;the next sum of the row and column value from the sine table. The
row_inc pair is ;the previous pointer for the row Sine table. Only the upper
byte is used
;to find the next sine value. Likewise, the column_inc pair works the same way
for
;the column frequency. Once we have a row and column sine value, they are added
;together.  The column frequency requires a 3 db higher level with respect to
;the row.
;This is accomplished by shifting the column left before adding to the row.
;Because the sine table has been located with the lower byte of its address at
00,
;and because it's 256 bytes long, (00 --> FF), we don't have to be concerned
with
;the upper byte of the sine table address; it’s always 02. This simplifies the
math
;each time the “calc_sin_value” is determined.

CALCULATE_PWM:
      ADD   row_inc_lo,r_freq_lo    ;The r_freq pair for the key is added to
                                    ;the
      ADC   row_inc_hi,r_freq_hi    ;last pointer value and stored. The high
                                    ;byte
      LD    pointer_lo,row_inc_hi   ;row_inc_hi is loaded into the sine table
                                    ;pointer_lo.
      LDC   row_val,@pointer        ;The new row value from the sine table.

      ADD   col_inc_lo,c_freq_lo    ;Now we do the same for the column.
      ADC   col_inc_hi,c_freq_hi    ;
      LD    pointer_lo,col_inc_hi   ;
      LDC   col_val,@pointer        ;We now have the column value as well.

      RL    col_val                 ;The column value is doubled
                            ;by shifting left before the sum. This is to
                            ;give
                            ;the column frequencies a 3 db gain over the
                            ;row frequencies. (This is refered to as "twist"
                            ;in the telephone industry.)

      ADD   row_val,col_val         ;2(col_val) + row_val --> row_val .
      LD    calc_sin_value,row_val  ;The calc_sin_value is this sum.
                                    ;Save it for the next T1 load in the
DTMF_LOOP.

WAIT_T0_POLLED:
      TM    IRQ,#00010000b    ;Test for T0 IRQ set.
```

```
      JR    Z,WAIT_T0_POLLED  ;If not, loop back to test it again.
      AND   IRQ,#11101111b    ;Else, T0 IRQ is set, so clear it and
      JR    DTMF_LOOP         ;jump to start next cycle of DTMF output.


DTMF_EXIT:                    ;The key is no longer active.
      DI                      ;(Always disable global interrupts while modify-
                              ;ing
                              ;Int regs.)
      CLR   TMR               ;Stop both timers.
      CLR   IRQ               ;Clear all pending interrupts.
      LD    IMR,#00010000b    ;Enable T0 interrupts only.
      LD    PRE0,#00000001b   ;Set T0 prescaler for divide by 64 (00) and
                              ;continuous mode.
      CLR   T0                ;Set T0 counter for divide by 256 (00); 16 mSec
                              ;int.
      LD    TMR,#00000011b    ;Load and enable T0 for KEY_SCAN work.

      CLR   row_inc_lo        ;Reset pointers so sinewaves will be
      CLR   row_inc_hi        ;in phase. This is for the optional 4 column
      CLR   col_inc_lo        ;keypad with additional offset table entries.
      CLR   col_inc_hi        ;This clearing is not necessary for 3 column
                              ;keypad.

      IRET                    ;Return to WAIT_HERE loop and reenable T0 int.

;*****************************************************************************
;               TIMER_1 Interrupt Service Routine
;*****************************************************************************
;
TIMER_1:                      ;Timer 1 interrupt service.
      AND   P0,#11111110b     ;Lower DTMF output line.
      IRET                    ;Reenable Timer IRQ's and return to DTMF work.

;*****************************************************************************
;                Offset Table
;*****************************************************************************

      org   01C0h             ;Locate offset_table


      include     "offset.S"  ;offset_table.
;
;=============================================================================
;=    TITLE:      offset_table.s                                           =
;=    DATE:       October 1, 1999                                          =
;=    PURPOSE:                                                             =
;=
;=    FILE TYPE:  Included Row and Column Frequency Lookup Table           =
;=
;=    HARDWARE:   NovaTech Z8 Proto. Board ZPCB18                          =
;=
;=    ASSEMBLER:  ZiLOG ZDS / ZMASM                                        =
;=    PROGRAMMER: Bob Bongiorno                                            =
;=============================================================================
;
;     Offset Table for Frequency Increment
;     ------------------------------------
;
```

```
offset_table:
;                                Key / Row Freq / Col Freq
;                                -------------------------
            .word 7709,9904    ; *   /   941 Hz / 1209 Hz
            .word 6980,9904    ; 7   /   852 Hz / 1209 Hz


            .word 6308,9904    ; 4   /   770 Hz / 1209 Hz
            .word 5710,9904    ; 1   /   697 Hz / 1209 Hz
;
            .word 7709,10945   ; 0   /   941 Hz / 1336 Hz
            .word 6980,10945   ; 8   /   852 Hz / 1336 Hz
            .word 6308,10945   ; 5   /   770 Hz / 1336 Hz
            .word 5710,10945   ; 2   /   697 Hz / 1336 Hz
;
            .word 7709,12100   ; #   /   941 Hz / 1477 Hz
            .word 6980,12100   ; 9   /   852 Hz / 1477 Hz
            .word 6308,12100   ; 6   /   770 Hz / 1477 Hz
            .word 5710,12100   ; 3   /   697 Hz / 1477 Hz
;
; last column for 16 keypad consists of pure sines for debugging only
;
            .word 12000,12000 ; D   /   1500 Hz Sinewave
            .word 8000,8000    ; C   /   1000 Hz Sinewave
            .word 4000,4000    ; B   /    500 Hz Sinewave
            .word 2000,2000    ; A   /    250 Hz Sinewave

;           Hex Conversion for Reference
;           ---------------------------

;           .byte 1Eh,1Dh,26h,0B0h
;           .byte 1Bh,44h,26h,0B0h
;           .byte 18h,0A4h,26h,0B0h
;           .byte 16h,4Eh,26h,0B0h

;           .byte 1Eh,1Dh,2Ah,0C1h
;           .byte 1Bh,44h,2Ah,0C1h
;           .byte 18h,0A4h,2Ah,0C1h
;           .byte 16h,4Eh,2Ah,0C1h

;           .byte 1Eh,1Dh,2Fh,44h
;           .byte 1Bh,44h,2Fh,44h
;           .byte 18h,0A4h,2Fh,44h
;           .byte 16h,4Eh,2Fh,44h

;     Optional Pure Sine Hex Codes for debugging only

;           .byte 2Eh,0E0h,2Eh,0E0h
;           .byte 1Fh,40h,1Fh,40h
;           .byte 0Fh,0A0h,0Fh,0A0h
;           .byte 07h,0D0h,07h,0D0h

;*****************************************************************************
;                Sine Table
;*****************************************************************************

      org   0200h             ;Locate Sine Table
```

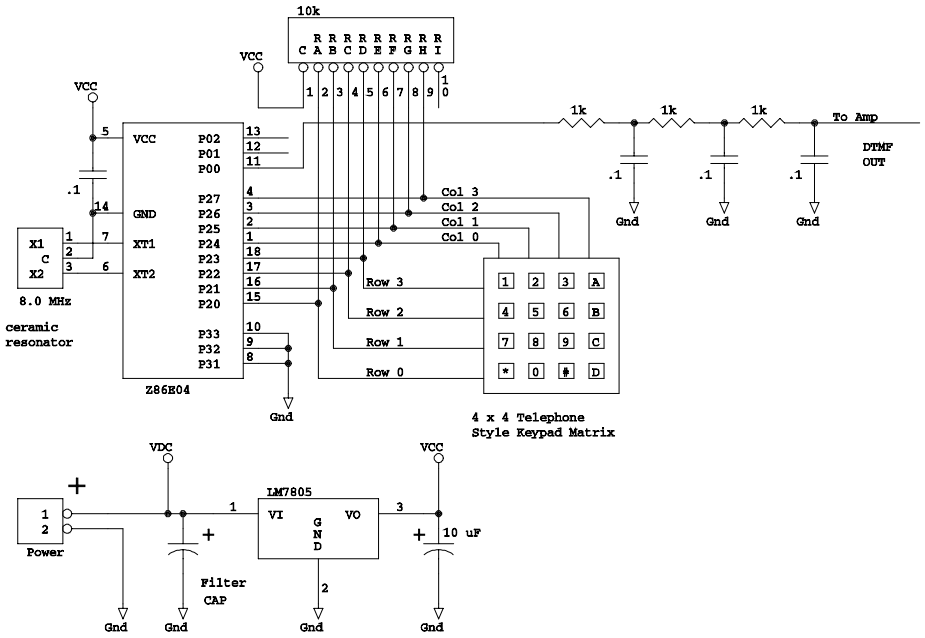
```
      include     "Sine.s"    ;Sine table.
;
;=====================================================================
;=    TITLE:            Sine.s                                       =
;=    DATE:       October 1, 1999                                    =
;=    PURPOSE:
;=
;=    FILE TYPE:  Included Sine Value Lookup Table                   =
;=
;=    HARDWARE:   NovaTech Z8 Proto. Board ZPCB18                    =
;=
;=    ASSEMBLER:  ZiLOG ZDS / ZMASM                                  =
;=    PROGRAMMER: Bob Bongiorno                                      =
;=====================================================================
;
Sine:
      .byte  17, 17, 18, 18, 18, 19, 19, 20, 20, 20, 21, 21, 21, 22, 22, 22
;
      .byte  23, 23, 23, 24, 24, 24, 25, 25, 25, 26, 26, 26, 27, 27, 27, 27
;
      .byte  28, 28, 28, 28, 29, 29, 29, 29, 29, 30, 30, 30, 30, 30, 31, 31
;
      .byte  31, 31, 31, 31, 31, 31, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32
;
      .byte  32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 31, 31, 31, 31, 31
;
      .byte  31, 31, 31, 30, 30, 30, 30, 30, 29, 29, 29, 29, 29, 28, 28, 28
;
      .byte  28, 27, 27, 27, 27, 26, 26, 26, 25, 25, 25, 24, 24, 24, 23, 23
;
      .byte  23, 22, 22, 22, 21, 21, 21, 20, 20, 20, 19, 19, 18, 18, 18, 17
;
      .byte  17, 17, 16, 16, 16, 15, 15, 14, 14, 14, 13, 13, 13, 12, 12, 12
;
      .byte  11, 11, 10, 10, 10,  9,  9,  9,  8,  8,  8,  7,  7,  7,  7,  7
;
      .byte   6,  6,  6,  6,  5,  5,  5,  5,  5,  4,  4,  4,  4,  4,  3,  3
;
      .byte   3,  3,  3,  3,  3,  3,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2
;
      .byte   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,  3
;
      .byte   3,  3,  3,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  6,  6,  6
;
      .byte   6,  7,  7,  7,  7,  8,  8,  8,  9,  9,  9, 10, 10, 10, 11, 11
;
      .byte  11, 12, 12, 12, 13, 13, 13, 14, 14, 14, 15, 15, 16, 16, 16, 17
;
;*********************************************************************
;                Dummy Interrupt Service Routine
;*********************************************************************
      org   03F8h             ;Locate Dummy interrupt handler to just
                              ;fit into top of memory for 1K ROM device.
                              ;This is a recommended routine for all Z8's.
                              ;Locate just inside available ROM space.
                              ;Fill unused ROM locations ahead of this
                              ;with NOP's (FF).
```

```
DUMMY:                          ;Vector here for spurious interrupts.
      DI                        ;Disable global,
      CLR   IMR                 ;potential and
      CLR   IRQ                 ;all pending interrupts.
      JP    DTMF_INIT           ;Jump to cold start / initialization.

;*****************************************************************************
;
;                 Program End
;*****************************************************************************
;
      end
==============================================================================
;==============================================================================
```

Schematic

Figure 1. DTMF Tone Generation Schematic



Note: VDC voltage is not to exceed the input voltage and / or power dissipation of the LM7805 regulator. The “Filter CAP” is selected based on the supply voltage and ripple and the minimum regulator drop-out voltage. All CAP

values are in µF with +80/-20% tolerance, except for the caps in DTMF OUT filter. They are +5/ -5% tolerance.

Flowcharts

Figure 2. DTMF (Initialization) and WAIT_LOOP

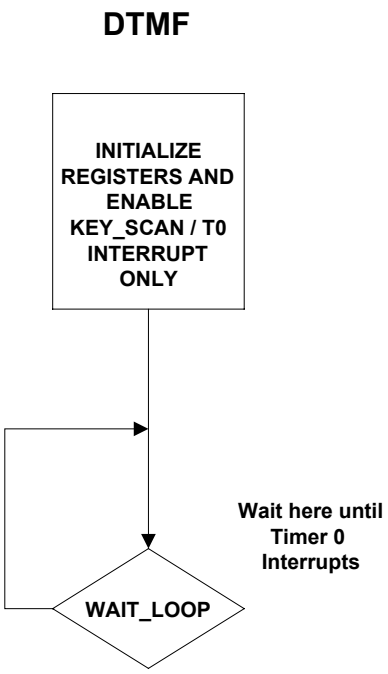


Figure 3. KEY_FOUND

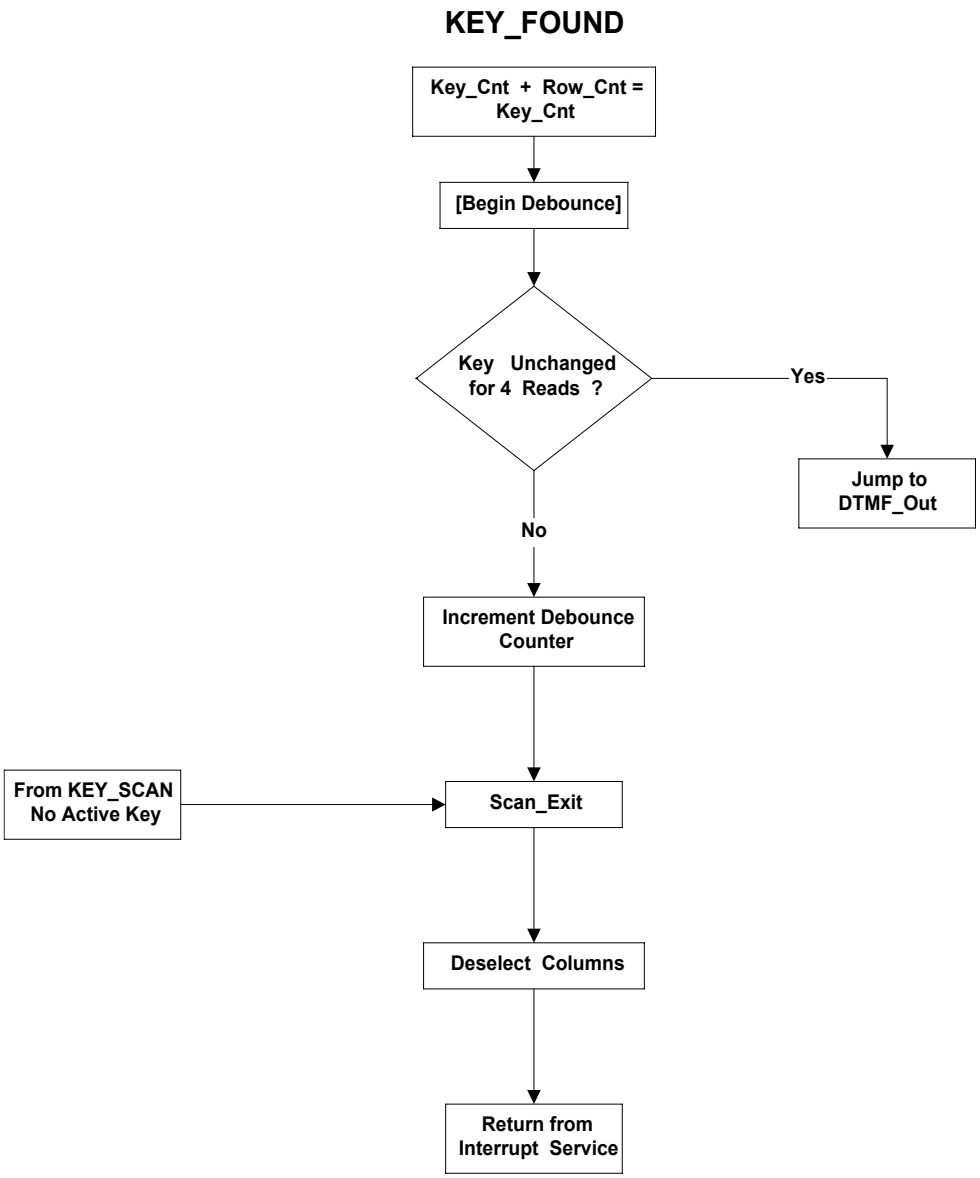


Figure 4. DTMF_OUT

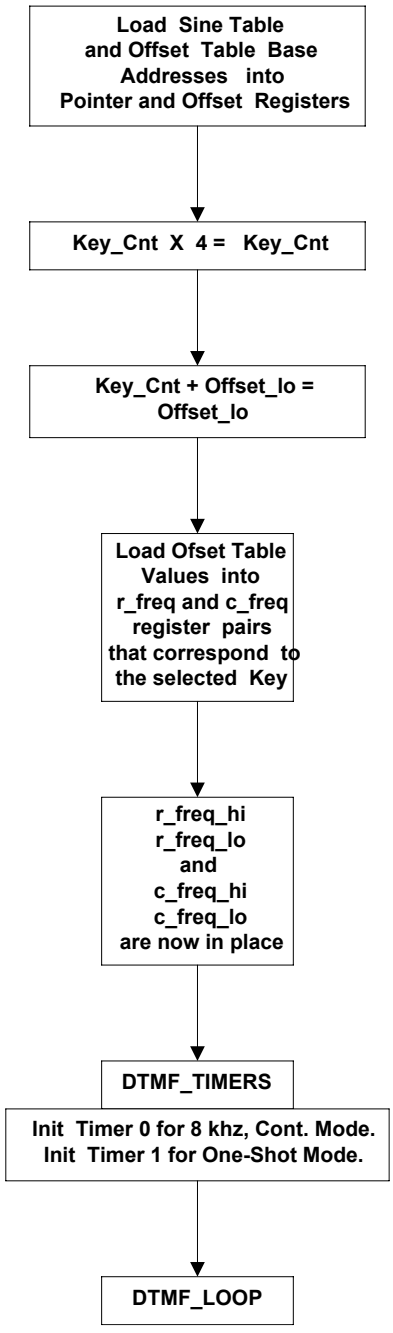


Figure 5. DTMF_LOOP

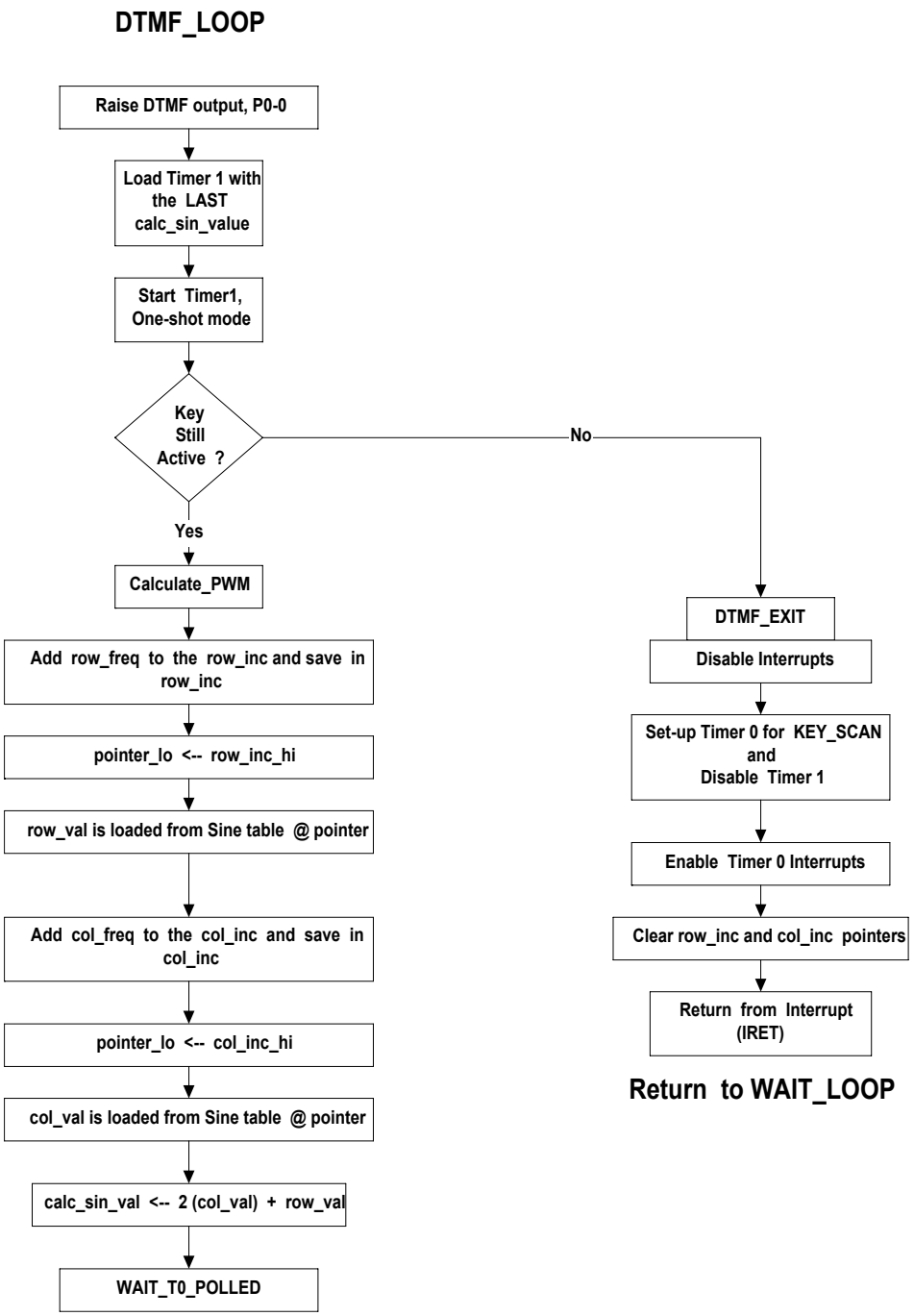
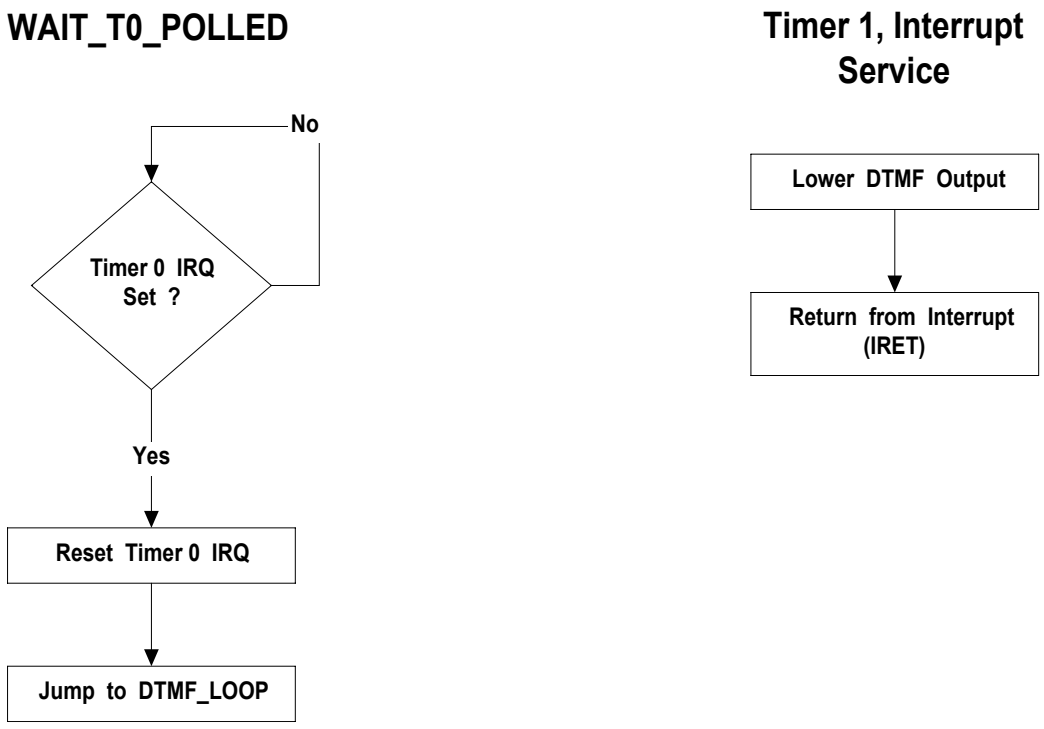


Figure 6. WAIT_TO_POLLED and Timer 1 Interrupt Service



Test Procedure

Equipment Used

Testing the DTMF Tone Generation program requires the following equipment:

- Windows 95/98/NT-based PC with ZDS 2.11 installed
- Z86CCP01ZEM and/or a Z86E04 programmed as specified
- Z86CCP00ZAC
- 8 Volt @ 1 Amp power supply and/or a 5 Volt @ 100-mAmp power supply
- Oscilloscope (required if viewing the DTMF waveforms)
- Audio amplifier and speaker (required if monitoring audio signals)
- Frequency counter (required if precise measurement of sine wave frequency is necessary)
- Breadboard of circuit.

General Test Setup and Execution

The testing was performed with ZDS 2.11 and the Z86E04 as the target chip. The 18-pin emulator target cable is connected from the emulator to the target proto board. The proto board used the Novatech Z8 prototyping board.Though specific MCU pin numbers are provided, this code works on almost any Z8, provided two timers are available. The pin numbers must be reviewed and some modifications would be required.

ZDS 2.11 is used to assemble the source program (`DTMF.s`) and monitor the Z86E04 register file memory windows, if required.The emulator runs the program when Reset GO is selected. Optionally, an OTP may be programmed and installed in the proto board, allowing operation without the PC and Emulator. The check-sum for the program is `59F7h` if assembled as is and all unused memory locations are filled with `FFh`.

Test Results

The program works as specified and the standard DTMF and optional sine waves are generated for the corresponding key selected.

References

DTMF Tone Generation Using the Z8 CCP, AP96Z8X1200, ZiLOG, Inc., 1997.