Application Note

AT&T

Dual-Tone Multifrequency Receiver Using the *WE*® DSP32 Digital Signal Processor

Contents

Contributed by: *J. Hartung*
S. L. Gay
G. L. Smith

Introduction

A common signaling method used in the telephone network is dual-tone multifrequency (DTMF) signaling. In this scheme, pairs of tones are used to signal the digits 0 through 9, pound (#), star (*), and the digits A, B, C, and D. For each pair, one of the tones is selected from a low group of four frequencies, and the other from a high group of four frequencies. The correct detection of a digit requires both a valid tone pair and the correct timing intervals.

DTMF signaling is used both to set up a call and to control features such as call forwarding and conference calling. In some applications, it is necessary to detect DTMF signaling in the presence of speech, so it is important that the speech waveform is not interpreted as a valid signaling tone. Standard test procedures have been published [1]* to verify the performance of DTMF receivers for valid tones and for the recognition of speech signals as tones.

The implementation of a DTMF receiver involves the detection of each of the signaling tones, validation of a correct tone pair, and timing to determine that a digit is present for the correct amount of time and with the correct spacing between tones. In addition, depending on the algorithm used to detect frequencies, it is sometimes necessary to perform additional tests to improve the performance of the decoder in the presence of speech. Current DSP technology allows several DTMF receivers to be implemented on a single device. A DSP

* [ ] indicates a reference listed at the conclusion of this application note.

implementation is useful in applications in which the digitized signal is available and several channels need to be processed, such as in a private branch exchange (PBX), or where other functions can be included on the same device.

This application note describes a DTMF receiver based on the discrete Fourier transform (DFT). Using this algorithm, 16 DTMF receivers can be implemented on a *WE* DSP32 Digital Signal Processor using internal RAM (for both data and program memory) and executing at a 160 ns instruction cycle time. The input data is assumed to be μ-law coded, and is received on the DSP32 serial interface. Detected digits can be read from the DSP32 parallel port.

DTMF Requirements

Figure 1 shows the matrix of frequencies used to encode the 16 DTMF symbols. Each symbol is represented by the sum of the two frequencies that intersect the digit. The row frequencies are in a low band, below 1 kHz, and the column frequencies are in a high band, between 1 kHz and 2 kHz. The digits are displayed as they would appear on a telephone's 4x4 matrix keypad (on standard telephone sets, the fourth column is omitted). DTMF receivers are required to detect frequencies with a tolerance of $\pm$1.5 percent as valid tones. Tones that are offset by $\pm$3.5 percent, or greater, must not be detected. This requirement not to detect tones is necessary to inhibit the detector from falsely detecting speech and other signals as valid DTMF digits. The receiver is required to work with a worst-case signal-to-noise ratio (SNR) of 15 dB, and with an attenuation of 26 dB.

Another requirement is the ability to detect DTMF signals when the two tones are received at different levels. The high-band tone may be received at a lower level than the low-band tone due to the attenuation characteristics of the telephone network. This level difference is called twist, and the situation described is called normal twist. Reverse twist occurs when the low-band tone is received at a lower level than the high-band tone. The receiver must operate with a maximum of 8 dB normal twist and 4 dB reverse twist.

In addition to the frequency, noise, and twist requirements, the DTMF signal must meet timing requirements for duration and spacing of digit tones. Digits are required to be transmitted at a rate of less than ten per second. A minimum spacing of 50 ms between tones is required, and the tones must be present for a minimum of 40 ms. Any tone-detection scheme used to implement a DTMF receiver must have a significant time resolution to verify correct digit timing.

A final requirement for the receiver is that it operates in the presence of speech without incorrectly identifying the speech signal as a valid DTMF symbol. This is referred to as talk-off performance. Although this requirement is not stated in strict numerical terms, standard recordings such as the Mitel DTMF test tape [2] contain speech segments that are used to test the receiver's performance under these conditions.

The remainder of this application note describes the algorithm, implementation, and performance of the DFT-based *WE* DSP32 Digital Signal Processor DTMF receiver.

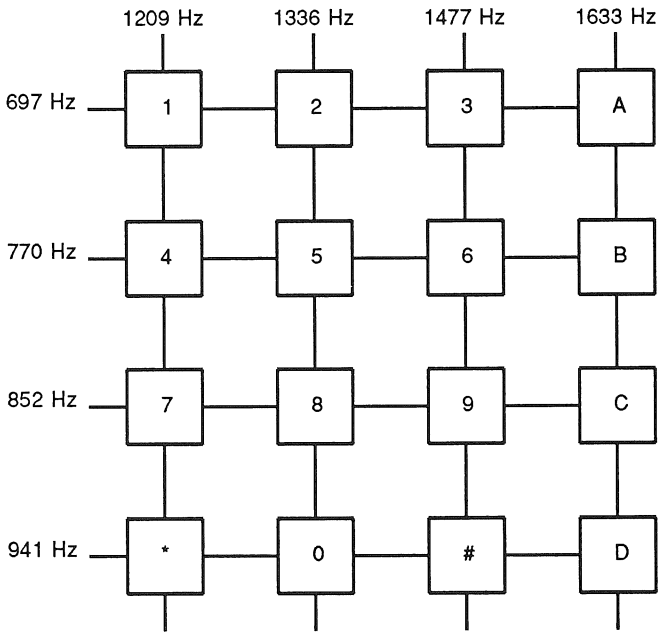

Figure 1. 4x4 Matrix Telephone Keypad

The Basic Algorithm

The general approach taken by this algorithm for DTMF tone detection is to take the Fourier transform of the observed signal and search for energy at the frequencies of interest. Since the algorithm is implemented on a DSP, a Discrete Fourier Transform (DFT) is used. The analysis frame must be long enough to resolve the DTMF frequencies, but short enough to detect the minimum length tone. A 12.75 ms

frame at a sampling rate of 8 kHz is a good choice [3].

To guard against talk-off, the total energy is calculated and compared to the sum of the largest magnitudes in the high- and low-frequency bands. If a valid tone pair is being transmitted, then the two values should be equivalent. But if there is speech present, the total energy should be much greater.

The algorithm also exploits the rich harmonic structure of the speech signal to assist in guarding against talk-off. If speech has energy at a frequency, f_1, it most likely has significant energy at twice the frequency, $2f_1$. Simulation of a DTMF tone by speech is prevented by calculating the energy at each tone and at its second harmonic. If the second harmonic is below a threshold proportional to the fundamental, a pure tone is present. Otherwise, it is assumed that a speech signal is present.

In calculating the DFT, the Goertzel algorithm [4], a method for calculating any single coefficient of a DFT, is chosen over a fast Fourier transform (FFT) algorithm. There are two reasons for this. In order to obtain the required frequency resolution at an 8 kHz sampling rate, a 256-point FFT would be required. Since the algorithm for tone detection requires knowledge of the energy at only 16 frequencies, it is more efficient to execute the Goertzel algorithm for these 16 frequencies than to perform the FFT for all 256 frequencies. In addition, the Goertzel algorithm is recursive, eliminating the need to store 256 samples for the FFT for each DTMF detector. This saves both real-time and data memory.

The Goertzel algorithm can be thought of as a matched filter for each DFT coefficient. The transfer function for the filter is:

$$H_k(Z)=\frac{1-W_N^k Z^{-1}}{1-2\cos(\frac{2\pi k}{N})Z^{-1}+Z^{-2}}$$

where $W_N = \exp(-j2\pi/N)$, and N is the length of the observation window for the DFT. The flow graph of this transfer function is shown in Figure 2.

Initially, the state variables of the filter are set to zero. Then, the filter is executed N times. The output at this point, $y_k(N)$, is the k'th coefficient of a length N DFT. Notice that the filter is implemented as a direct form II second-order section. The recursive part of the filter is on the left-hand side of the delay elements, and the nonrecursive part is on the right. Since only the output at time N is needed, it is only necessary to compute the nonrecursive part of the filter after the last iteration of the recursive part. A further simplification in the algorithm is made by observing that only the square of the magnitude of the DFT coefficient is needed. The nonrecursive calculation of the DFT coefficient is:

$$y_k(N)=S(N)-W_N^k S(N-1)$$

where S(N) and S(N–1) represent the value of the state variables at times N and N–1. It can be shown that:

$$|y_k(N)|^2=$$

$$|S(N)|^2-2\cos(\frac{2\pi k}{N})S(N)S(N-1)+|S(N-1)|^2$$

Therefore, it is only necessary to store the value, 2 cos (2πk/N), for each coefficient to be evaluated.

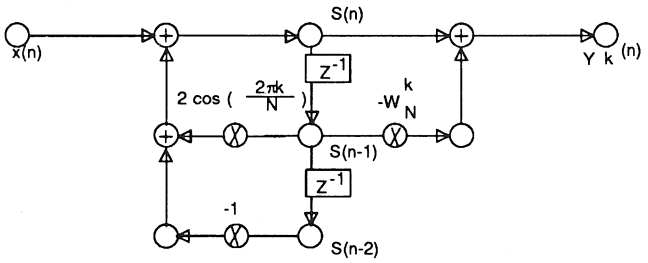

Figure 2. Flow Graph of the Transfer Function

Implementation

The flow chart for the DTMF tone detector is shown in Figure 3.

At the beginning of each frame, the state variables of each of the 16 Goertzel filters are set to zero. Then, for 12.75 ms (102 samples at a sampling frequency of 8 kHz), the recursive part of the Goertzel algorithm is executed. At the end of each frame, the square of the magnitude of the coefficients at each frequency are calculated. The following five tests are then performed on these 16 values:

1. **The Magnitude Test.** In each group of four tones (the row and column tones), the frequency whose coefficient has the largest

magnitude is found. This frequency is tentatively referred to as the detected "tone" for that band. The detected tones' magnitudes are compared to a threshold. If either tone's magnitude is smaller than the threshold, then this test has failed.

2. **The Twist Test.** Twist is the ratio of the energy in the lower-band tone to the energy in the upper-band tone. If the measured twist falls outside the region of −4 dB to +8 dB, the test has failed.
3. **The Frequency Offset Test.** The energy of the largest tone in each band is compared to the energies of the other frequencies in that band. If the difference falls below a threshold in either band, the test fails.
4. **The Tone-to-Total Energy Test.** Let c1, c2, and c3 be three different constants, each greater than one. The energy of the low-band detected tone is weighted by c1, the energy of the high-band detected tone is weighted by c2, and the sum of the low- and high-band tones' energies are weighted by c3. If any one of these terms is smaller than the total energy, then the test has failed.
5. **The Harmonic Ratio Test.** The energy of the detected tone is compared to the energy of its second harmonic. If the energy of the second harmonic is not sufficiently smaller than the energy of the detected tone, the test has failed.

If all of the above tests pass, the tone pair is decoded as an integer between 0 and 15, representing the digits 0 through 9, A through D, * and #, respectively. This value is placed in a memory location designated D(k) and is the digit at frame k. If any of the tests fail, then −1, representing "no detection," is placed in D(k). For a new valid digit to be declared and sent to the parallel port, D(k) must be the same for two successive frames. If it is valid for more than two successive frames, the receiver is detecting the continuation of the previously validated digit, and a new digit is not output.

Performance

The DTMF detector described above has been verified by using the Mitel DTMF test tape.

This section documents the results for each test included on the tape.

Test 1 – This test determines the frequency offset range of the receiver. Acceptable performance requires detection of the tone when the magnitude of the frequency offset is less than ±1.5 percent. When the magnitude of the frequency offset is greater than ±3.5 percent, the receiver is required not to detect the tone (see Table 1).

Table 1. Frequency Offset Range of the Receiver

Frequency Deviation	Percent Deviation at Cutoff Digit			
	1	5	9	D
Low group	2.4 −3.5	2.4 −2.7	2.7 −2.2	2.7 −2.3
High group	2.3 −2.4	2.1 −2.0	2.0 −1.9	1.8 −1.7

Test 2 – This test verifies that the receiver operates with a normal twist of up to 8 dB and a reverse twist of up to 4 dB (see Table 2).

Table 2. Verification of Receiver's Twist Operations

Twist	Attenuation at Cutoff (in dB) Digit			
	1	5	9	D
Reverse	5.3	4.3	4.4	4.1
Normal	8.2	8.1	8.3	8.4

Test 3 – Dynamic range test. A receiver must detect touch-tones with a dynamic range of 26 dB. The receiver detects touch-tones with a dynamic range of 34 dB.

Test 4 – Guard time test. A receiver must detect DTMF pulses as short as 40 ms. The receiver detects pulses as short as 34 ms.

Test 5 – Signal-to-noise ratio test. A receiver must detect touch-tones with a SNR of 15 dB. The receiver detects all touch-tones at this level.

Test 6 – Talk-off test. Acceptable talk-off performance requires less than 30 false digit detections in the talk-off section of the Mitel tape. The receiver detected 5 false digits.

Hardware Description

The *WE* DSP32 Digital Signal Processor receives its sampled data input from a time-division multiplexed (TDM) serial bit stream

which contains 16 channels of 8-bit μ-law samples. A sample timing diagram is shown in Figure 4. The rate of the input load clock (ILD) must be 128 kHz (one clock cycle for each channel). The input clock (ICK), which is the rate at which the bits are input to the DSP32, should equal 2.048 MHz. The signal SY, which is the external synch pulse, tells the DSP32 when the first channel is being transmitted. When the ILD corresponding to channel 16 goes high, the SY should go low coincidentally and go high 800 ns before the ILD corresponding to channel 1 goes high. The DSP32 DMA hardware handles the loading of input data to a buffer in RAM.

The output of the DSP32 is connected to an external microprocessor. When a digit has been validated, it is written to the lower 8 bits of the parallel I/O data register (PDR). The lower 4 bits contain the decoded digit, and the

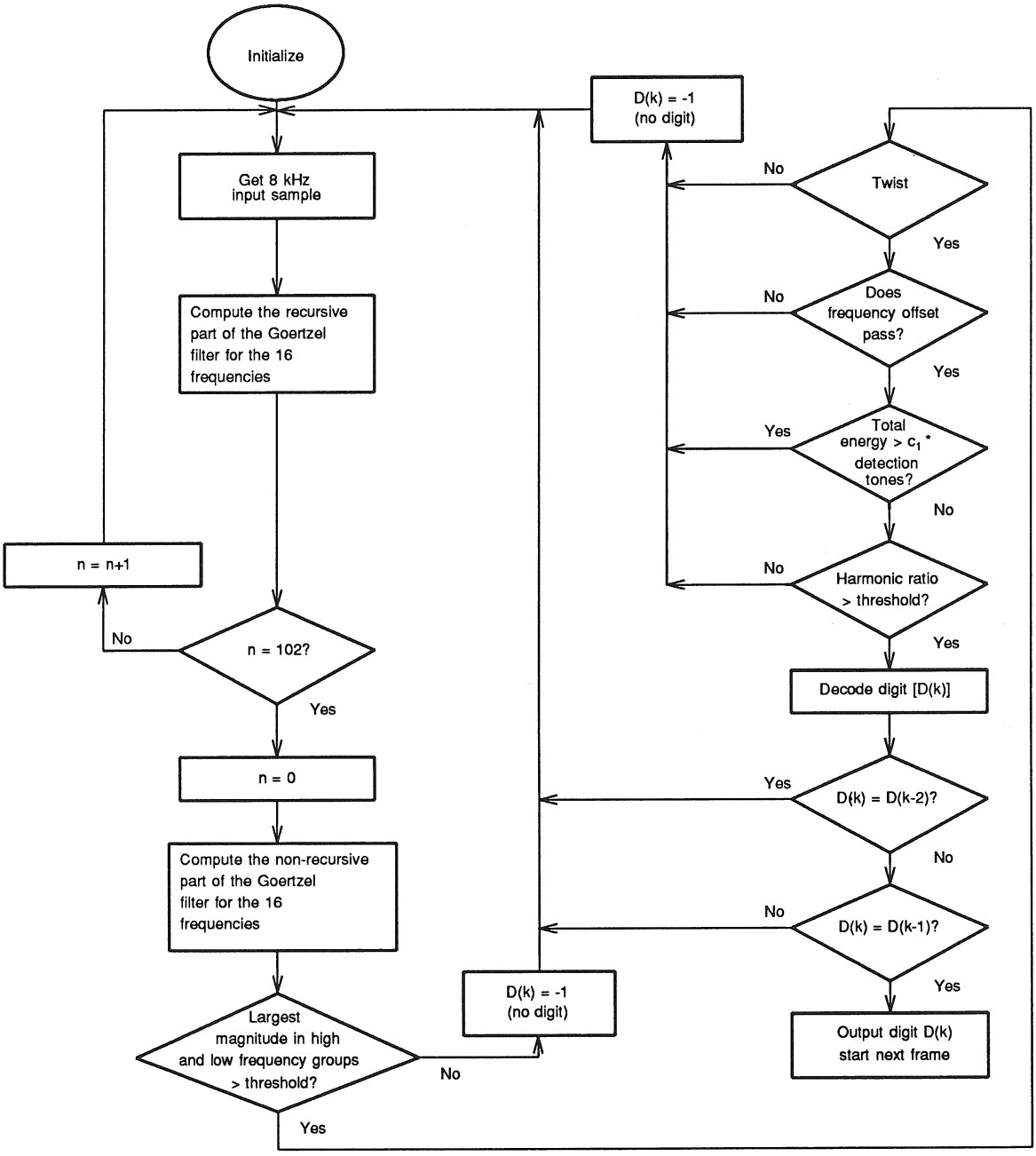

Figure 3. Flow Chart for the Dual-Tone Multifrequency Detector

upper 4 bits of the byte contain the channel number. When the PDR is written, the Parallel data full (PDF) pin goes high, which can be used to interrupt the external microprocessor. The microprocessor then has a maximum of 40 μs to read the data before it is overwritten.

Source Code

Program 1 implements 16 DFT-based DTMF receivers on the AT&T *WE* DSP32 Digital Signal Processor operating with a 160 ns instruction rate. Both the program and data are stored in on-chip RAM. The state variables for the DFT are in bank 1, and everything else is in bank 0. During the 125 μs sample period, there is time to execute the 16 receivers and an additional 130 instructions.

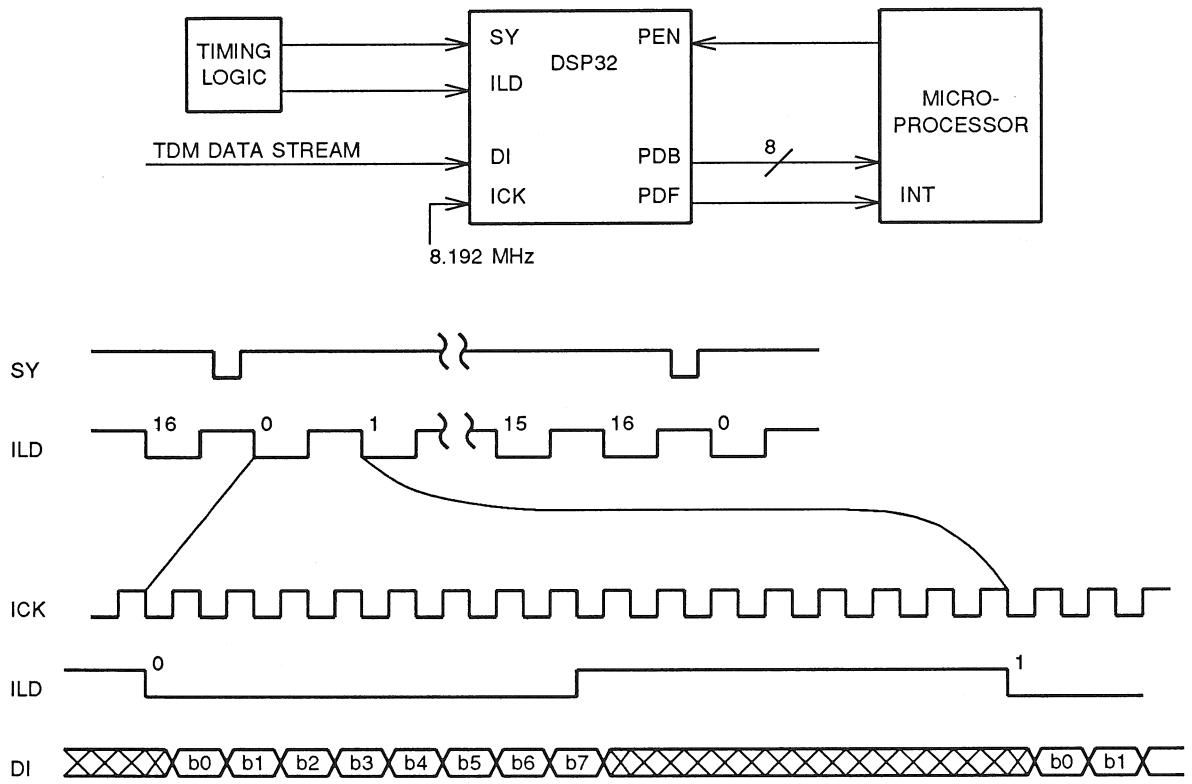

Figure 4. DSP32 TDM Data and Microprocessor Interfaces

Program 1. DTMF Receiver Source Code

```
.global _iirstvr,_iirincr,_ttdigit,inbuf,bufio

        dauc=0x0
        ioc=0x40                        /* 8-bit input, passive clks, DMA on IBF */
        r15=12
        r16=4
begin:if (syc) goto begin              /* wait for sync to go high before */
        nop                             /* beginning DMA                   */
        ioc=0x2040                      /* 8-bit input, passive clks, DMA on IBF */
        pin=inbuf
        r17=-60
        r5=102-2                        /* DFT frame  size - 2 */
        r6=energy
        r10=_iirzero
        r7=16-2                         /* 16 receivers - 2 */
clrnrg:if (r7-- >= 0) goto clrnrg       /* zero the energy locations on a */
        *r6++ = a0 = *r10               /* frame by frame basis. */
        r7=32-2
        r2=_iirco                       /* pointer to coefficients */
        r3=_iirstvr                     /* pointer to state variables */
clear:*r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        *r3++ = a0 = *r10
        if (r7-- >= 0) goto clear       /* zero the state variables on a */
        *r3++ = a0 = *r10               /* frame by frame basis. */

/*************************************************************************/
/*                                                                       */
/*      This section sets up the DMA for each receiver.  After 16 samples    */
/*      (one for each receiver) have been received, they are converted to    */
/*      floating point representation and stored at bufio.  DMA is then      */
/*      re-initialized to continue receiving data.                           */
/*                                                                       */
/*************************************************************************/
sample: pin-ilimit
iwait:if (le) goto iwait                /* wait for input data to be gathered */
        pin-ilimit
        pin=inbuf
        r1=inbuf
        r4=bufio
        r6=energy
        r3=14                           /* 16 receivers -2 */
ioxfr:if (r3-- >= 0) goto ioxfr
        *r4++ = a1 = ic(*r1++r16)       /* move data from inbuf to bufio */
        r11=14                          /* 16 receivers - 2 */
```

```
/*************************************************************************/
/*                                                                       */
/*      This section computes the LHS of the DFT for the 1st and 2nd order  */
/*      harmonics of each tone for each sample. It is performed 16 times,   */
/*      once for each receiver. The state variables for each receiver are   */
/*      stored in a sequential stack on top of each other.                  */
/*                                                                       */
/*************************************************************************/
        r1=bufio
        r3=_iirstvr                         /* pointer to state variables */
        r4=r3+4                             /* stores updated state variables */
dft:a1=*r1
/* 697hz    1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++   /* 2nd state var. d[1,1] */
        *r4++r15 = a0 = a0 - *r3++          /* 1st state var. d[1,2] */
/* 770hz    1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 852hz    1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 941hz    1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1209hz   1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1336hz   1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1477hz   1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1633hz   1st harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 697hz    2nd harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 770hz    2nd harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 852hz    2nd harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 941hz    2nd harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1209hz   2nd harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1336hz   2nd harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1477hz   2nd harmonic  LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++               /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
/* 1633hz   2nd harmonic   LHS        */
        a0 = a1 + (*r4-- = *r3++) * *r2++r17            /* d[1,1] */
        *r4++r15 = a0 = a0 - *r3++                      /* d[1,2] */
        if (r11-- >= 0) goto dft            /* 16 receiver counter */
        *r6++ = a2 = *r6 + a1 * *r1++       /* update energy */
        if (r5-- >= 0) goto sample          /* 205 sample frame counter */
        nop
```

```
/*************************************************************************/
/*                                                                       */
/*      This section computes the RHS of the DFT for the 1st harmonics   */
/*      of each tone.  The magnitude is squared and stored in consecutive */
/*      locations starting at _iirstvr.                                  */
/*                                                                       */
/*************************************************************************/

        ioc=0x40
        r16=0
        r12=energy
        r18=_iirstvr
        r19=_ttdigit
        r11=14                          /* 16 receivers - 2 */
timing:r2=_iirco
        r4=r18
        r15=8
        r1=6
        r3=r18                   /* points to location of mag^2 of each tone */
right:a2= *r4 * *r4++           /* Sn * Sn-1 */
        a1=*r4-- * *r4                  /* Sn-1^2 */
        a0 = a1 + *r4++r15 * *r4        /* Sn^2 + Sn-1^2 */
        if (r1-- >= 0) goto right
        *r3++ = a0 = a0 - a2 * *r2++  /* |Y|^2          */

/*************************************************************************/
/*                                                                       */
/*      This section finds the largest value in the low and high frequency */
/*      band and stores the 2 values at _iirstvr+32. It also computes their */
/*      location in the low and high frequency band and stores these 2   */
/*      values at _iirincr.                                              */
/*                                                                       */
/*************************************************************************/

        r8=_iirincr
        r4=r18+32                       /* loc. of largest mag. in each band */
        r3=r18                          /* loc. of mag^2 of each tone */
        r7=0
lrgmag3:r6=1
        r5=0
        r1=0
        r2=0
        *r4=a0=*r3++            /* writes possible large mag. temporarily *r4 */
lrgmag4:a1=*r3+
        nop
        a2=a1-a0
        3*nop
        if (ale) goto lrgmag1           /* test which value is larger */
        nop
        *r4=a0=a1
        r2=r2+1
        r1=r2
        goto lrgmag2
        r1=r1+r5                        /* contains the row & col increments */
lrgmag1:r5=r5+1
lrgmag2:if (r6-- >= 0) goto lrgmag4
        nop
        *r8++=r1                        /* contains the row & col increments */
        if (r7-- >= 0) goto lrgmag3
        r4=r4+4
```

```
/*************************************************************************/
/*                                                                       */
/*      This section compares the lower of the high and low frequency    */
/*      band magnitudes to a threshold level.  If it is lower than the   */
/*      threshold the program updates the valid digit stack with a -1    */
/*      and retrieves another frame.  Otherwise, it continues and checks */
/*      the twist.                                                       */
/*                                                                       */
/*************************************************************************/

        r1=r18+32                       /* loc. of max row & col mag's */
        a0= *r1 - *r1++                 /* high mag. - low mag. */
        r10=_ttrlevl
        a2 = *r1
        a1 = *r1-- - *r10               /* lower value - 8e7 */
        if (alt) goto _ttrlow
        r2=_twisthg
_ttrhigh:a2 = *r1
        a1 = *r1++ - *r10               /* lower value - 8e7 */
        r2=_twistlw
        a1 = -*r1 + a2 * *r2            /* low mag. * 2.5 - high mag. */
        nop
        if (alt) goto notone            /* threshold check        */
        nop
        if (alt) goto notone            /* reverse twist test */
        nop
        goto frqofset                   /* continue */
        nop
_ttrlow:a1 = -*r1 + a2 * *r2            /* high mag. * 9.2 - low mag. */
        if (alt) goto notone            /* threshold check */
        2*nop
        if (alt) goto notone            /* normal twist test */
        nop

/*************************************************************************/
/*                                                                       */
/*      Do largest frequencies stand out?                                */
/*      This section compares the largest magnitude (detected "tone") in */
/*      each frequency band with the other magnitudes in that band. If the */
/*      "tone" is not significantly larger then the other magnitudes the */
/*      program returns to begin and samples another frame.              */
/*                                                                       */
/*************************************************************************/

frqofset:r4=0
        r10=_vldtonl
        r1=r18+32                       /* loc. of largest mag. in each band */
        r2=r18                          /* loc. of mag^2 of each tone */
freqoff2:r3=2
freqoff1:a2 = *r2
        a1 = *r1 - *r2++                /* check if mag's are the same */
        nop
        a0 = *r1 - a2 * *r10            /* highest mag. - other mags. * coeff */
        nop
        if (aeq) goto freqoff           /* if equal mag's get next mag */
        nop
        if (age) goto freqoff           /* if highest mag < other mag * coeff */
        nop                             /* tone is not valid */
        goto notone                     /* Therefore, update _ttdigit with -1 */
        nop                             /* and get another frame of samples. */
freqoff:if (r3-- >= 0) goto freqoff1
        nop
        r10=_vldtonh
        if (r4-- >= 0) goto freqoff2
        r1=r1+4
```

```
/*************************************************************************/
/*                                                                       */
/*      This section compares the total inband energy to the largest mag.  */
/*      in the low band, the largest mag. in the high band, and the sum of  */
/*      the two magnitudes.                                              */
/*                                                                       */
/*************************************************************************/

energ:r1=r18+32
        a2=*r1++                        /* a2=largest mag in low band */
        a1=*r1                          /* a1=largest mag in high band */
        r3=_nrgcoef
        a3 = a2 - *r12 * *r3++          /* a3=low band mag-energy*lowband coef */
        a2=a1+a2
        a3 = a1 - *r12 * *r3++          /* a3=highband mag-energy*highband coef*/
        a3 = a2 - *r12++ * *r3          /* a3=sum of mags-energy*sum coef */
        if (alt) goto notone
        nop
        if (alt) goto notone
        nop
        if (alt) goto notone
        nop

/*************************************************************************/
/*                                                                       */
/*      Is the harmonic ratio > threshold?                               */
/*      This computes the magnitudes squared of the "tones" second       */
/*      harmonics and compares it with the "tone". If the magnitudes are  */
/*      within a certain ratio, the program returns to begin and samples  */
/*      another frame.                                                   */
/*                                                                       */
/*************************************************************************/

        r3=r18+32                       /* loc. of largest mag. in each band */
        r1=0
        r5=0
        r10=_iirincr
        r7=_hrmcoef
_2ndharm:r6=*r10++                      /* compute the location of the 2nd harm */
        r4=r18                          /* coefficients and state variables */
        r8=r6*2
        r8=r8*2
        r2=r8+32
        r2=r2+r5
        r8=r2*2
        r2=r2+_iirco                    /* loc. of the 2nd harmonic coeff's */
        r4=r4+r8
        a2= *r4 * *r4++                 /* Sn * Sn-1 */
        a1=*r4-- * *r4                  /* Sn-1^2 */
        a0 = a1 + *r4 * *r4             /* Sn^2 + Sn-1^2 */
        a0 = a0 - a2 * *r2              /* |Y|^2 */
        2*nop
        a3 = *r3++ - a0 * *r7++         /* 1st harm - 2nd harm * 2nd harm coeff */
        3*nop
        if (ale) goto notone
        r5=16
        if (r1-- >= 0) goto _2ndharm
        r9=_iirincr                     /* used in decode digit section */
```

```
/*****************************************************************************/
/*                                                                           */
/*      Decode digits                                                        */
/*      pdr's lower byte contains the valid decoded digit in its lower 4     */
/*      bits and the receiver number in its upper 4 bits.                    */
/*                                                                           */
/*****************************************************************************/

        r1=*r9++                        /* location of _iirincr */
        r7=r16*2
        r2=r1*2
        r2=r2*2
        r3=*r9
        r2=r2*2
        r3=r3*2
        r2=r2+r3
        r2=r2+_romtbl                   /* loc. of decoded digit */
        r10=*r2                         /* present digit D(k) */
        r4=r19+2
        r5=*r4                          /* 2nd to last digit D(k-2) */
        r7=r7*2
        r5=r5-r10
        if (eq) goto tone1              /* D(k)=D(k-2) ? */
        r4=r19
        r5=*r4                          /* last digit decoded D(k-1) */
        r7=r7*2
        r5=r5-r10
        if (ne) goto tone1              /* D(k)=D(k-1) ? */
        r8=r7*2
        r8=r8^r10
        pdr=r8
        goto tone1
        nop
notone:r10=-1                           /* set digit to -1 and update stack */
tone1:r1=r19
        r3=*r1
        *r1++ = r10                     /* update D(k-1)=D(k) */
        *r1 = r3                        /* update D(k-2)=D(k-1) */
        r19=r19+4
        r16=r16+1
        if (r11-- >= 0) goto timing     /* process timing for next receiver */
        r18=r18+128
        goto begin
        r15=12
```

```
/*************************************************************************/
/*                                                                       */
/*                              PROGRAM VARIABLES                        */
/*                                                                       */
/*  First order harmonic coefficients                                    */

_iirco: float        1.70773781, 1.645281, 1.568687, 1.4782
        float   1.1641, .99637, .7986184, .5685327

/*  Second order harmonic coefficients                                   */

        float        .916368, .70695, .46077885, .1851
        float         -.644862, -1.0072464, -1.36221, -1.676771
_iirzero: float 0.0
_ttrlevl: float 4e7
_twisthg: float 10.5
_twistlw: float 7.4
_vldtonl: float 5.0
_vldtonh: float 5.0
_nrgcoef: float 7.0, 5.6, 36.0
_hrmcoef: float 1.6, 60.0
_romtbl:  int 1, 2, 3, 10
           int 4, 5, 6, 11
           int 7, 8, 9, 12
           int 14, 0, 15, 13

/*                                                                       */
/*************************************************************************/

/*************************************************************************/
/*                                                                       */
/*                              STATE VARIABLES                          */
/*                                                                       */

inbuf:   15*float 0.0
ilimit:  float 0.0
dummy:   100*float 0.
bufio:   16*float 0.0
_iirincr:2*int 0
energy:  16*float 0.0
_ttdigit:32*int 0
.rsect ".hi_ram"
_iirstvr:512*float 0.0              /* state variables */

/*                                                                       */
/*************************************************************************/
```

References

1. *Touch-Tone Calling–Requirements for Central Office*, AT&T Compatibility Bulletin No. 105, August 8, 1975.
2. *Tone Receiver Test Cassette #CM7291*, Mitel Technical Data Manual, Mitel Semiconductor, 2321 Morena Blvd. Suite M, San Diego, CA 92110.
3. Mock, Patrick, "Add DTMF Generation and Decoding to DSP -μP Designs," *EDN*, March 21, 1985.
4. Oppenheim, A.V. and Schaeffer, R.W., *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

NOTES:

For additional information, contact your AT&T Account Manager, or call:

- AT&T Microelectronics
  Dept. 51AL230230
  555 Union Boulevard
  Allentown, PA 18103
  1-800-372-2447

  In Canada, call:
  1-800-553-2448

- AT&T Microelectronics GmbH
  Freischützstrasse 92
  D-8000 Munich 81
  West Germany
  Tel. 0 89/95 97 0
  Telex 5 216 884

- AT&T Microelectronics Asia/Pacific
  4 Shenton Way #20-00
  Shing Kwan House
  Singapore 0106
  Tel. 225-5233
  Telex RS 42898
  FAX 225-8725

- AT&T Microelectronics Japan
  7F, Fukoku Seimei Bldg.,
  2-2-2, Uchisaiwai-cho,
  Chiyoda-ku, Tokyo 100 Japan
  Tel. (03) 502-3055
  Telex J32562 ATTIJ
  FAX (03) 593-3307

June 1988

AP88-08DMOS