Texas Instruments

DTMF Tone Generation and Detection

An Implementation Using the TMS320C54x

Application Report

1997

Digital Signal Processing Solutions

Texas Instruments

Printed in U.S.A., June 1997

SPRA096

DTMF Tone Generation and Detection

An Implementation Using the TMS320C54x

SPRA096
June 1997

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

1 Introduction 1

2 DTMF Tone Generator 2

3 DTMF Tone Detector 5
3.1 Collecting Spectral Information 5
3.2 Modifications to the Goertzel Algorithm 7
3.3 Validity Checks 8
3.4 Program Flow Description of the DTMF Detector 10
3.5 Multichannel DTMF Tone Detection 13

4 Speed and Memory Requirements 14

5 Performance 15
5.1 MITEL Tests 15
5.2 BELLCORE Talk-Off Test 18

6 Summary 19

References 20

Appendix A Background: Digital Oscillators A-1
A.1 Digital Sinusoidal Oscillators A-1

Appendix B Background: Goertzel Algorithm B-1
B.1 Goertzel Algorithm B-1

Appendix C DTMF Tone Generator Code Listing C-1
C.1 DTMF Tone Generator Executable on a TMS320C54x EVM C-1

Appendix D DTMF Tone Detector Code Listing D-1
D.1 DTMF Tone Detector Executable on a TMS320C54x EVM D-1

List of Figures

1 Touch-Tone Telephone Keypad
(A row tone and column tone are associated with each digit.) 1

2 Two Second-Order Digital Sinusoidal Oscillators
(Program-flow description of the DTMF generator) 3

3 Flowcharts of the DTMF Encoder Implementation 4

4 Short Mathematical Description of Goertzel Algorithm 6

5 Mathcad Simulation (Case 1: 858 Hz, 1483 Hz → digit = 9)
(Case 2: 800 Hz, 1483 Hz → digit = undefined) 7

6 Flowcharts of the DTMF Decoder Implementation (Part 1 of 2) 11

7 Flowcharts of the DTMF Decoder Implementation (Part 2 of 2) 12

List of Tables

1 Coefficients and Initial Conditions for Sinusoidal Oscillators 2
2 List of Frequencies and Filter Coefficients 6
3 List of Tuned Frequencies and Modified Filter Coefficients 8
4 Speed and Memory Requirements for the DTMF Encoder 14
5 Speed and Memory Requirements for the DTMF Decoder 14
6 MITEL Test Results 17
7 Bellcore Talk-off Test Results 18

DTMF Tone Generation and Detection

ABSTRACT

This application report describes the implementation of a dual-tone multiple frequency (DTMF) tone generator and detector for the TMS320C54x. The author provides some theoretical background on the algorithms used for tone generation and detection, and documents the actual implementation in detail. Finally, code is benchmarked in terms of speed and memory requirements**.**

1 Introduction

A DTMF codec incorporates an encoder that translates key strokes or digit information into dual-tone signals, as well as a decoder that detects the presence and the information content of incoming DTMF tone signals. Each key on the keypad is identified uniquely by its row frequency and its column frequency (see Figure 1). The DTMF generating and decoding scheme is not very computationally extensive and can be handled easily by a DSP concurrently with other tasks. This report describes an implementation of the DTMF codec on the TMS320C54x, a fixed-point DSP designed by Texas Instruments (TI™)† especially for telecommunication applications.

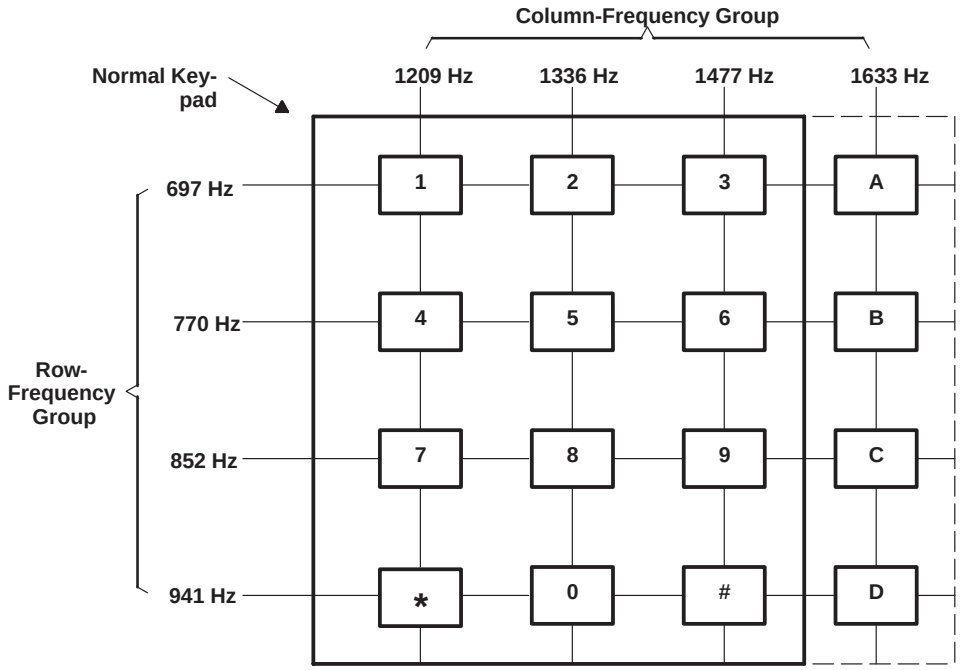

Figure 1. Touch-Tone Telephone Keypad
(A row tone and column tone are associated with each digit.)

† TI is a trademark of Texas Instruments Incorporated.

2 DTMF Tone Generator

The encoder portion and tone generation part of a DTMF codec are based on two programmable, second-order digital sinusoidal oscillators, one for the row tone and one for the column tone. Two oscillators, instead of eight, facilitate the code and reduce the code size. Of course, for each digit that is to be encoded, each of the two oscillators needs to be loaded with the appropriate coefficient and initial conditions before oscillation can be initiated. Since typical DTMF frequencies range from approximately 700 Hz to 1700 Hz, a sampling rate of 8 kHz for this implementation is within a safe area of the Nyquist criteria. Table 1 specifies the coefficients and initial conditions necessary to generate the DTMF tones. Figure 2 displays the block diagram of the digital oscillator pair.

For more detail, see Appendix A, which provides some theoretical background and a guideline for determining coefficients and initial conditions for digital sinusoidal oscillators.

Tone duration specifications by AT&T state the following: 10 digits/sec is the maximum data rate for touch-tone signals. For a 100-msec time slot, the duration for the actual tone is at least 45 msec and not longer than 55 msec. The tone generator must be quiet during the remainder of the 100-msec time slot.

Table 1. Coefficients and Initial Conditions for Sinusoidal Oscillators

f(Hz)	*a*1	*y*(–1)	*y*(–2)/*A*
697	0.85382	0	–0.52047
770	0.82263	0	–0.56857
852	0.78433	0	–0.62033
941	0.73911	0	–0.67358
1209	0.58206	0	–0.81314
1336	0.49820	0	–0.86706
1477	0.39932	0	–0.91680
1633	0.28424	0	–0.95874

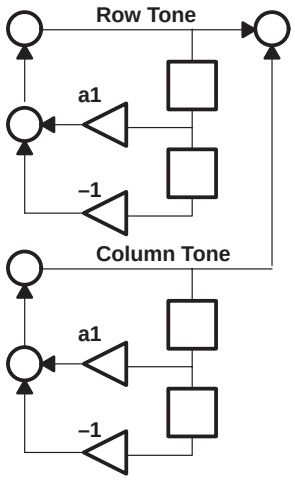

Figure 2. Two Second-Order Digital Sinusoidal Oscillators (Program-flow description of the DTMF generator)

For the following description of the program flow, it is helpful to consult the flowchart shown in Figure 3. Essentially, the series of keypad entries (digits) are translated into a series of dual-tones of certain duration that are interrupted by pauses of certain duration. Later, the dual-tones enable the decoder to identify the associated digits. But, the pauses also are necessary to discriminate between two or more identical digits entered successively.

The program flow, therefore, incorporates two tasks that are swapped after certain time intervals. One task (the “tone task”) generates dual-tone samples and the other (the “quiet task”) generates pause samples. Each task is assigned a certain duration that is controlled by a timer variable. At the end of each task, the task has to initialize the timer variable and the task-name (tone or quiet) for the next task to be invoked. At the end of the quiet task, one very important component is added: A new digit is retrieved from the digit buffer and is unpacked. Unpacking means that the digit is mapped to the row/column tone properties (oscillator coefficients, initial conditions) and pointers are loaded, pointing to the appropriate locations in the oscillator property table.

The entire program flow is synchronized to the receive-interrupt service routine, which provides a perfect clock for real-time processing and constant sample output. On completion of the RINT_ISR, the task scheduler is invoked, which determines the particular task (tone or quiet) that needs to be executed. Both tone task and quiet task check on the timer variable to determine if the end of the task duration is already reached. If not, a tone or quiet sample, respectively, is generated. If the end of the task duration is reached, the next task name and duration is initialized and starts to execute with the completion of the next RINT_ISR. The quiet task, additionally, unpacks the next digit at the end of its duration.

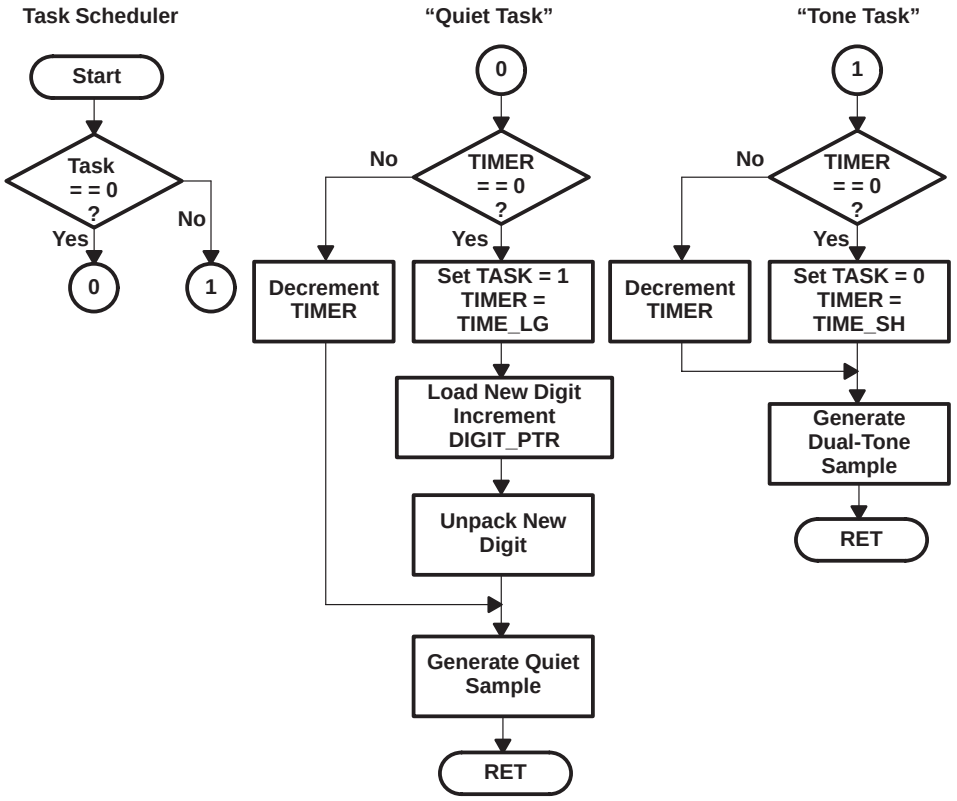

Figure 3. Flowcharts of the DTMF Encoder Implementation

3 DTMF Tone Detector

The task to detect DTMF tones in an incoming signal and to convert them into actual digits is certainly more complex than the encoding process. The decoding process is by its nature a continuous process, meaning it needs to continually search an incoming data stream for the presence of DTMF tones.

3.1 Collecting Spectral Information

The Goertzel algorithm is the basis of the DTMF detector. This method is a very effective and fast way to extract spectral information from an input signal. This algorithm essentially utilizes two-pole IIR type filters to compute DFT values effectively. It is, thereby, a recursive structure (always operating on one incoming sample at a time), as compared to the DFT (or FFT) which needs a block of data before being able to start processing. The IIR structure for the Goertzel filter incorporates two complex-conjugate poles and facilitates the computation of the difference equation by having only one real coefficient. For the actual tone detection, the magnitude (here, squared magnitude) information of the DFT is sufficient. After a certain number of samples N (equivalent to a DFT block size), the Goertzel filter output converges towards a pseudo DFT value vk(n), which can then be used to determine the squared magnitude. See Figure 4 for a short mathematical description of the algorithm. More detail is provided in Appendix B.

Goertzel Algorithm in short:

1. Recursively compute for n = 0 ..*N*

$$v_k(n) = 2\cos\left(\frac{2\pi}{N}K\right) \cdot V_k(n-1) - V_k(n-2) + x(n)$$

where $V_k(-1) = 0 \qquad V_k(-2) = 0$

$x(n) = input$

2. Compute once every *N*

$$|X(k)|^2 = y_k(N)\,y_k^*(N)$$

$$= v_k^2(N) + v_k^2(N-1) - 2\cos\left(\frac{2\pi}{N}k\right) v_k^2(N)\, v_k^2(N-1)$$

Figure 4. Short Mathematical Description of Goertzel Algorithm

The Goertzel algorithm is much faster than a true FFT, as only few of the set of spectral line values are needed and only for those values are filters provided. Squared magnitudes are needed for eight row/column frequencies and for their eight-second harmonics. The second harmonics information later enables discrimination of DTMF tones from speech or music. Table 2 contains a list of frequencies and filter coefficients. The choice of *N* is mainly driven by the frequency resolution needed, which sets a lower boundary. *N* also is chosen so that *(k/N)fs* most accurately coincides with the actual DTMF frequencies (see Table 1) assuming *k*s are integer values and *f*s is a sampling frequency of 8 ksps.

Table 2. List of Frequencies and Filter Coefficients

1st HARMONICS (*N* = 102) *fs* = 8 ksps				2nd HARMONICS (*N*=102) *fs* = 8 ksps			
DTMF	**k**	**FREQUENCY (k/N)fs/Hz**	**COEFFICIENT cos(2pi k/N)**	**2nd harm.**	**k**	**FREQUENCY (k/N)fs/Hz**	**COEFFICIENT cos(2pi k/N)**
697	9	706 (+ 1.2%)	27860	1394	18	1412 (+ 1.2%)	14606
770	10	784 (+ 1.9%)	26745	1540	20	1569 (+ 1.9%)	10891
852	11	863 (+ 1.3%)	25529	1704	22	1725 (+ 1.3%)	7010
941	12	941 (0.0%)	24216	1882	24	1882 (+ 0.0%)	3023
1209	15	1176 (– 2.7%)	19747	2418	31	2431 (+ 0.5%)	–10891
1336	17	1333 (– 0.2%)	16384	2672	34	2667 (– 0.2%)	–16384
1477	19	1490 (+ 0.9%)	12773	2954	38	2980 (+ 0.9%)	–22811
1633	21	1647 (+ 0.9%)	8967	3266	42	3294 (+ 0.9%)	–27860

The graphs of Figure 5 (from a Mathcad simulation) show how the Goertzel filters resonate with an input signal of matching row/column frequency (case 1) and how they do not resonate when the input signal does not match their frequency (case 2). Appendix C contains the Mathcad simulation.

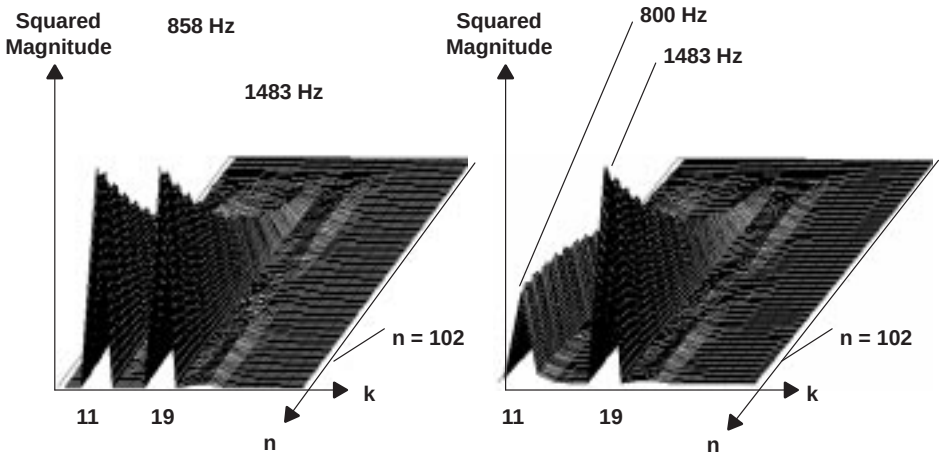

Figure 5. Mathcad Simulation
(Case 1: 858 Hz, 1483 Hz → digit = 9)
(Case 2: 800 Hz, 1483 Hz → digit = undefined)

3.2 Modifications to the Goertzel Algorithm

The coefficients of Table 2 reflect the coefficients needed to recursively compute the true DFT. The evenly spaced frequency bins of a true DFT present an inherent drawback in the DTMF tone-detection process. The DFT frequency bins mostly deviate from the true DTMF frequency by an amount in the range of up to 2% off center frequency. To be able to meet the acceptable bandwidth specifications, a modification of the algorithm departs from the true DFT and tunes frequency bins exactly with the DTMF tone frequencies. This modification gives up the DFT property of evenly spaced frequency bins; and with that, takes two calculated risks: (1) A frequency bin is possibly moved inside of the mainlobe of its neighboring frequency bin. Therefore, neighboring frequency bins can affect each other. Note that the mainlobe of the continuous magnitude spectrum of a rectangular windowed sinewave (window is N wide) is exactly the distance of 2 DFT frequency bins. (2) This is especially true when column frequencies and 2nd harmonics of row frequencies lie close to one another. Note that column frequencies and 2nd harmonics of row frequencies share the same frequency band.

To minimize these risks, two rules were applied: (1) Frequency bins are spaced apart by more than 0.9/fs. (2) If rule 1 cannot be achieved, 1st harmonics frequency bin placement has a higher priority than 2nd harmonic bin placement. With these rules applied, the coefficient table given in Table 2 is modified. All necessary fundamental frequency bins essentially can be tuned to their DTMF frequencies. A small frequency deviation is needed to be accepted only in two cases of second harmonics. The new method of using tuned frequency bins facilitates the checking of acceptable bandwidths of DTMF tones and helps meet the acceptable bandwidth specifications. The modified table of coefficients is given in Table 3.

Table 3. List of Tuned Frequencies and Modified Filter Coefficients

1st HARMONICS (*N* = 102) *fs* = 8 ksps				2nd HARMONICS (*N*=102) *fs* = 8 ksps			
DTMF	**k**	**FREQUENCY (k/N)fs/Hz**	**COEFFICIENT cos(2pi k/N)**	**2nd harm.**	**k**	**FREQUENCY (k/N)fs/Hz**	**COEFFICIENT cos(2pi k/N)**
697	8.88	697	27980	1394	17.93	1406 (+ 0.9%)	14739
770	9.82	770	26956	1540	19.72	1546 (+ 0.4%)	11414
852	10.86	852	25701	1704	21.72	1704	7549
941	12.00	941	24216	1882	24.00	1882	3032
1209	15.42	1209	19073	2418	30.83	2418	–10565
1336	17.03	1336	16325	2672	34.07	2672	–16503
1477	18.83	1477	13085	2954	37.66	2954	–22318
1633	20.82	1633	9315	3266	41.64	3266	–27472

3.3 Validity Checks

Once the spectral information (in the form of squared magnitude at each of the row and column frequencies and their second harmonics) is collected, a series of tests need to be executed to determine the validity of tone and digit results.

A first check makes sure the signal strength of the possible DTMF tone pair is sufficient. The sum of the squared magnitudes of the peak spectral row component and the peak spectral column component needs to be above a certain threshold (THR_SIG). Since already small twists (row and column tone strength are not equal) result in significant row and column peak differences, the sum of row and column peak provides a better parameter for signal strength than separate row and column checks. Tone twists are investigated in a separate check to make sure the twist ratio specifications are met.

The spectral information can reflect the types of twists. The more likely one, called “reverse twist”, assumes the row peak to be larger than the column peak. Row frequencies (lower frequency band) are typically less attenuated as than column frequencies (higher frequency band), assuming a low-pass filter type telephone line. The decoder, therefore, computes a reverse twist ratio and sets a threshold (THR_TWIREV) of 8 dB acceptable reverse twist. The other twist, called “standard twist”, occurs when the row peak is smaller than the column peak. Similarly, a “standard twist ratio” is computed and its threshold (THR_TWISTD) is set to 4 dB acceptable standard twist.

The program makes a comparison of spectral components within the row group as well as within the column group. The strongest component must stand out (in terms of squared amplitude) from its proximity tones within its group by more than a certain threshold ratio (THR_ROWREL, THR_COLREL).

Finally, the program checks on the strength of the second harmonics in order to be able to discriminate DTMF tones from possible speech or music. It is assumed that the DTMF generator generates tones only on the fundamental frequency; however, speech will always have significant even-order harmonics added to its fundamental frequency component. This second harmonics check, therefore, makes sure that the ratio of the second harmonics component and the fundamental frequency component is below a certain threshold (THR_ROW2nd, THRCOL2nd). If the DTMF signal pair passes all these checks, we say a valid DTMF tone pair, which corresponds to a digit, is present.

We now need to determine if the valid DTMF tone information contains stable digit information. This is done by mapping the tone-pair to its corresponding digit and comparing it with the previously detected digit. We call the digit information stable if it has been detected twice successively.

Finally, we compare the detected digit with the previous-to-last digit. Only if the last digit was preceded by a pause do we accept the current digit as a valid digit. The detector is then forced into a state where it waits for a pause before being able to accept a new digit. This last step is necessary to ensure the discrimination of identical keystrokes succeeding one another.

3.4 Program Flow Description of the DTMF Detector

This DTMF decoder implementation uses a task scheduler in a similar fashion as the encoder, again synchronized to the completion of the receive interrupt service routine (ISR). The receive ISR is continuously filling a buffer of N = 102 words with incoming data. Once the buffer is full, it signals the full buffer by setting the input data status word, *indatastat*, to 1. The task scheduler essentially keeps polling indatastat; and once it recognizes a full input data buffer (indatastat = 1), it initiates the DTMF detection process by calling the appropriate functions. (Consult the flowcharts in Figure 6 and Figure 7 might be very helpful at this point.)

First, the input data status word is reset to signal that the detection process has been initiated. Then, the content of the input data buffer is copied into an intermediate buffer for processing (double buffering). All the detection functions will then operate on the intermediate buffer. The gain control function attenuates strong signal inputs and protects the succeeding functions from the overflow of the accumulators. Next, the Goertzel filters are executed. Since the preceding gain control ensures that overflow cannot occur, overflow checking is removed and optimized loops allow fast execution. The output of the Goertzel function are the delay states of the 16 filters which are collected in an array.

On completion of the Goertzel function, the digit validation checks are invoked. The spectral information is computed from the filter delay states and collected in an energy template. For the next round of execution, the filter delay states are initialized to zero. The energy template is then searched for row and column energy peaks. From then on, the detector essentially operates in two modes: the tone/digit detection mode or the pause detection mode. In the tone/digit detection mode, the detector searches for DTMF tone presence and executes all the digit validation tests. In the pause detection mode, DTMF tone detection is disabled and the decoder first has to await a pause signal. Tone/digit or pause modes are controlled by the *detectstat* variable. Digit validation checks include signal strength, reverse and standard twist, relative peaks, second harmonics and digit stablility. With the successful completion of these tests, the valid digit is stored into the digit output buffer.

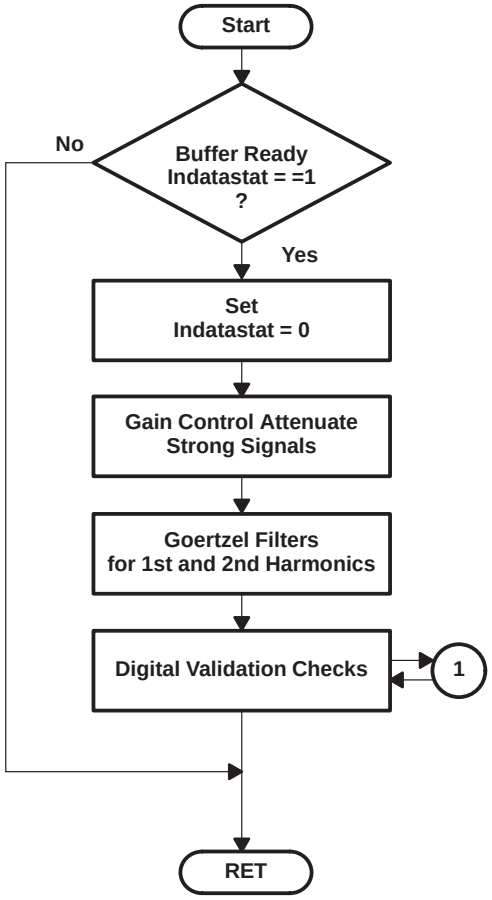

Figure 6. Flowcharts of the DTMF Decoder Implementation (Part 1 of 2)

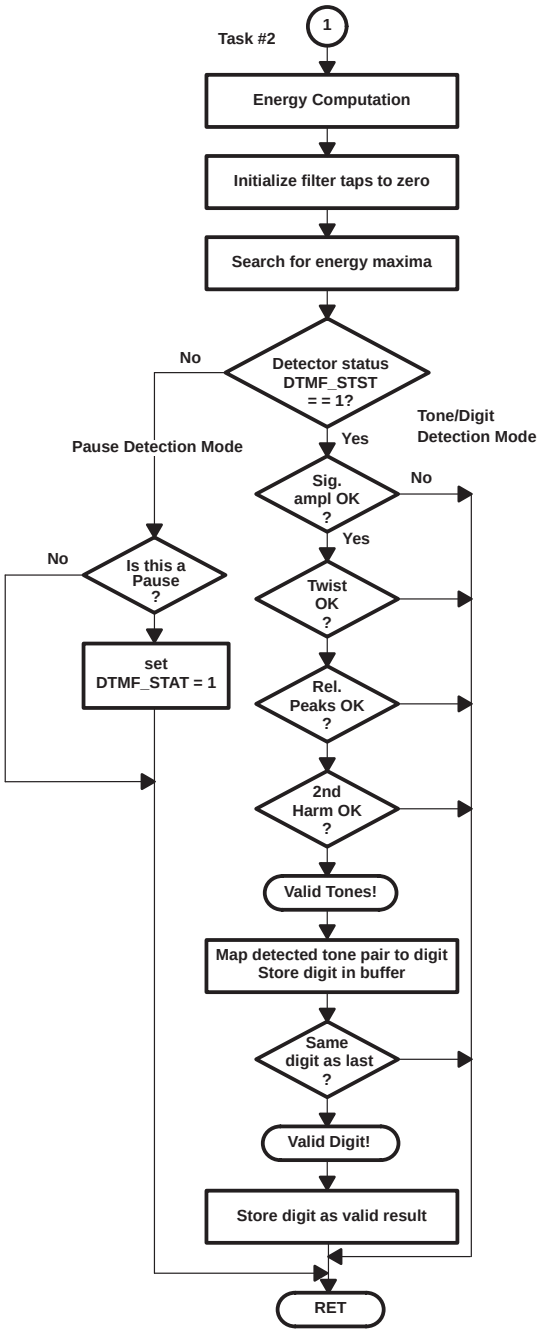

Figure 7. Flowcharts of the DTMF Decoder Implementation (Part 2 of 2)

3.5 Multichannel DTMF Tone Detection

The software is written as C-callable reentrant functions. This enables the user to set up multichannel tone detectors in C without adding significant additional code. In order to facilitate and structure multichannel applications, the code uses a structure to hold all the global variables and pointers to various arrays for a single channel. All the user has to do is to define a structure for each channel and initialize it properly. A pointer to the structure of a desired channel is passed to the C-callable functions and the detection takes place.

4 Speed and Memory Requirements

Table 4 and Table 5 summarize the speed and memory requirements of the DTMF encoder/decoder implementation. The encoder, as well as the decoder, occupies only a very small portion of the computing capability of this 50-MIPS DSP. The maximum MIPS count for the actual DTMF encoder is approximately 0.45 MIPS. The isolated DTMF decoder uses approximately 0.82 MIPS. These speed specifications include all processing necessary after the completion of the receive interrupt service routine. Also, a sampling rate of 8000 samples/sec is assumed. The MIPS performance of the DTMF decoder is achieved using a buffered concept instead of a sample-by-sample concept. The speed-critical Goertzel-DFT function has the 16 filters coded inline for fast execution. Due to an improved gain control, the typical overflow check within the goertzel routine was omitted. A drawback of the buffered concept is the need for additional data memory due to double buffering of data. The amount of data memory is given, to a large degree, by the size of the input data buffers, which are set to N = 102 length in this application. Table 4 and Table 5 summarize the benchmarks for the DTMF encoder and decoder.

Table 4. Speed and Memory Requirements for the DTMF Encoder

MODULE NAME	TASKS INCLUDED	PROGRAM MEMORY	DATA MEMORY	MAX CYCLES PER SAMPLE	MIPS
DTMF Encoder (sample by sample)	Task Scheduler Tone Task Quiet Task	129	49	57	0.45
HW/SW Initialization I/O	hwinit swinit RINT_ISR	90	—	—	—
TOTAL		219	49	57	0.45

Table 5. Speed and Memory Requirements for the DTMF Decoder

MODULE NAME	C FUNCTIONS	PROGRAM MEMORY	DATA MEMORY (n CHANNELS)	MAX CYCLES PER 102 SAMPLES	MIPS
DTMF Decoder (buffered)	some overhead Gain Control Goertzel-DFT DTMFChecks	572	260*n + 32	100 705 8802 788	0.008 0.055 0.690 0.062
HW/SW Initialization I/O	ISRs inits C-environment	351	—	—	—
TOTAL		923	260*n + 32	10395	0.82

5 Performance

Two well-known tests to evaluate the performance of DTMF decoders are available through MITEL and Bell Communications Research (Bellcore), both of which supply the associated test tapes and test procedures.

5.1 MITEL Tests

The DTMF tone decoder has been tested using the MITEL test procedures. For a preliminary test, a set of files with the digitized signal data for the various MITEL tests was acquired through TI internal sources. The files essentially contain a subset of the signal data of the real MITEL test tapes. Since none of the files contained more than 32K words of data, the files were reformatted as ".inc" files and then included in the source code. At link time, the data of a given test file was mapped into the external memory of the TMS320C54xEVM, occupying at maximum 32K words of data space. The code was then tested in real-time using the contents of the 32K-word test buffer as its input source. During this preliminary test, the various decoder thresholds were set to proper levels.

The complete MITEL test was then executed according to the test procedures specified in the MITEL test document. A digital audio tape (DAT) recording of the test tapes was used as input signal source. The decoder was executed on the TMS320C54xEVM utilizing a TMS320AC01 analog interface to convert the incoming signal into the digital domain. The MITEL test essentially has two sections. The first section measures the DTMF tone decoder in terms of recognition bandwidths (RBW), recognition center frequency offset (RCFO), standard twist, reverse twist, dynamic range (DR), guard time and signal-to-noise ratio (SNR). The second section, called "talk-off test", consists of recordings of conversations on telephone trunks made over a long period of time and condensed into a 30-minute period. In this section, the decoder's capabilities to reject other sources such as speech and music is measured. MITEL specifies a maximum of 30 responses of the DTMF decoder as acceptable speech-rejection.

Table 6 summarizes the MITEL test results. With the exception of the recognition bandwidth results for the low-band frequencies, the DTMF decoder passes all the tests and exceeds the specifications. The slightly higher relative bandwidths for the low-band frequencies were accepted in favor of a higher robustness. Essentially, the signal strength threshold THR_SIG2 determines to a large part the recognition bandwidths. THR_SIG2 has been set to leave slightly more room for digit acceptance and higher robustness. The threshold settings for acceptable twists helped pass the twist test and exceed the specifications. The dynamic range of the decoder of 28 dB is better than the specification. The guard time of 30 ms leaves enough head room to safely detect digits with a specified 45-ms tone duration. The decoder was able to correctly detect all 1000 tone bursts for each of the given noise environments of –24 dBV, –18 dBV and –12 dBV AWGN. When the decoder was exposed to the 30 minutes of speech and music samples, it did not respond a single time and, thereby, exceeded the MITEL talk-off specification of 30 permissible responses by a significant margin.

Table 6. MITEL Test Results

BW TESTS	FREQUENCY	RBW%	RCFO%
specification		1.5% < RBW < 3.5%	
Low band	697 Hz	5.8%	0.10%
	770 Hz	5.5%	0.05%
	852 Hz	5.1%	0.05%
	941 Hz	4.5%	0.05%
High band	1209 Hz	2.8%	0.10%
	1336 Hz	2.5%	0.05%
	1477 Hz	2.3%	0.15%
	1633 Hz	2.4%	0.00%
TWIST TESTS	**STD TWIST**	**REV TWIST**	
specification	> 4 dB	> 8 dB	
DIGIT 1	4.7 dB	8.5 dB	
DIGIT 5	4.6 dB	8.6 dB	
DIGIT 9	4.5 dB	8.3 dB	
DIGIT 16	4.6 dB	8.5 dB	
DR TESTS	**DYN RANGE**		
specification	> 25 dB		
DIGIT 1	28 dB		
DIGIT 5	28 dB		
DIGIT 9	28 dB		
DIGIT 16	28 dB		
GUARD TIME	**MIN PAUSE TIME**	**MIN TONE TIME**	
specification		45 ms	
DIGIT 1	13 ms	30 ms	
SNR TESTS	**NOISE**	**RESULT**	
specification	–24 dBV		
DIGIT 1	–24 dBV	passed	
DIGIT 1	–12 dBV	passed	
DIGIT 1	–18 dBV	passed	
TALK-OFF TEST	**DECODER RESPONSES**	**RESULT**	
specification	< 30		
result	none	very robust	

5.2 BELLCORE Talk-Off Test

Through TI internal sources, the Bellcore series-1 Digit Simulation Test Tapes for DTMF receivers were available for testing. These tapes consist of six half-hour sequences of speech samples, designated part 1 through part 6, which are known to contain energy at or near valid DTMF frequency pairs. This test exhaustively measures the speech-rejection capabilities of DTMF receivers in telecommunication systems. There are over 50,000 speech samples, some of which are music samples. It is estimated that the six parts of the series-1 tapes are equivalent to the exposure of one million customer dialing attempts in a local central office. In other words, exposing a DTMF receiver to all the speech samples in series-1, will produce the same number of digit simulations that the receiver would experience if it were exposed to customer speech and room noise present during network control signaling on one million calls. The Bellcore talk-off test is far more exhaustive than the MITEL talk-off test.

The test setup was identical to the one used for the MITEL talk-off. Table 7 summarizes the test results for the Bellcore talk-off. In the three hours of testing, the decoder responded to digit simulations only in six cases. This is far less than the specifications required to pass the talk-off. The decoder proved to be very robust in terms of its speech-rejection capabilities.

Table 7. Bellcore Talk-off Test Results

TEST	DIGITS	SPECIFICATION	RESULTS
parts 1 through 6 (3 hours)	0–9	per 1,000,000 calls < 333 responses	6 responses
parts 1 through 6 (3 hours)	0–9, *, #	per 1,000,000 calls < 500 responses	6 responses
parts 1 through 6 (3 hours)	0–9, *, #, A, B, C, D	per 1,000,000 calls < 666 responses	6 responses

6 Summary

DTMF tone encoding and decoding concepts and algorithms were described in detail. Further theoretical background is provided in the appendix. The DTMF encoder and decoder implementations were explained, and the associated speed and memory requirements were presented. The DTMF tone decoder has been tested according to the MITEL and BELLCORE test specifications and the results are documented. It is important to note that the decoder was implemented as reentrant, C-callable functions, which facilitate setting up a multi-channel DTMF decoder system. The code is modular and easy to integrate into any given telephony application. The decoder algorithm was optimized to meet the test specifications as well as offer a very attractive MIPS count far below 1 MIPS per channel.

References

1. Proakis, J.G., Manolakis, D.G., *Digital Signal Processing*, Macmillan Publishing Company, New York, 1992, pages 737–739.
2. Mock, P., “Add DTMF Generation and Decoding to DSP-P Designs,” *DSP Applications with the TMS320 Family, Vol. 1*, Texas Instruments, 1989.
3. Ziemer, R.E., Tranter, W.H., *Principles of Communications*, Houghton Mifflin Company, Boston, 1995.
4. MITEL Technical Data, *Tone Receiver Test Cassette CM7291*, 1980.
5. Bell Communications Research, *Digit Simulation Test Tape*, Technical Reference TR-TSY-000763, Issue 1, July 1987.

Appendix A Background: Digital Oscillators

A.1 Digital Sinusoidal Oscillators

A digital sinusoidal oscillator can, in general, be viewed as a form of a two-pole resonator for which the complex-conjugate poles lie on the unit circle. It can be shown that the poles of a second-order system with system function

$$H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{A.1}$$

with parameters

$$\begin{aligned} b_0 &= A \sin \omega_0 \\ a_1 &= -2 \cos \omega_0 \\ a_2 &= 1 \end{aligned} \tag{A.2}$$

are exactly located at the unit circle. That is

$$P_{1,2} = e^{\pm j \omega_0} \tag{A.3}$$

The discrete-time impulse response

$$h(n) = A \sin \left((n + 1)\ \omega_0 \right) \cdot u(n) \tag{A.4}$$

corresponding to the above second-order system clearly indicates a clean sinusoidal output due to a given impulse input. Therefore, this system can be termed a digital sinusoidal oscillator or digital sinusoidal generator.

For the actual implementation of a digital sinusoidal oscillator, the corresponding difference equation is the essential system descriptor, given by

$$y(n) = -a_1 y(n-1) - a_2 y(n-2) + b_0 \delta(n) \tag{A.5}$$

where initial conditions $y(-1)$ and $y(-2)$ are zero. Note that the impulse applied at the system input serves the purpose of beginning the sinusoidal oscillation. Thereafter, the oscillation is self-sustaining, as the system has no damping and is exactly marginally stable. Instead of applying a delta impulse at the input, let the initial condition $y(-2)$ be the systems oscillation initiator and remove the input. With this in mind, the final difference equation is given by

$$y(n) = 2 \cos\ \omega_0 \cdot\ y(n-1) - y(n-2) \tag{A.6}$$

where

$$y(-1) = 0$$
$$y(-2) = -A \sin \omega_0 \qquad \textbf{(A.7)}$$
$$\omega_0 = 2\pi f_0 / f_s$$

with f_S being the sampling frequency, f_0 being the frequency and A being the amplitude of the sinusoid to be generated. Note that the initial condition $y(-2)$ solely determines the actual amplitude of the sinewave.

Appendix B Background: Goertzel Algorithm

B.1 Goertzel Algorithm

As the first stage in the tone-detection process, the Goertzel algorithm is one of the standard schemes used to extract the necessary spectral information from an input signal. Essentially, the Goertzel algorithm is a very fast way to compute DFT values under certain conditions. It takes advantage of two facts:

1. The periodicity of phase factors $\left\{W_N^k\right\}$ allows the expression of the computation of the DFT as a linear filter operation utilizing recursive difference equations.
2. Only a few of the spectral values of an actual DFT are needed (in this application, there are eight row/column tones plus an additional eight tones or corresponding 2nd harmonics).

Keeping in mind that a DFT of size N is defined as

$$X(k) = \sum_{m=0}^{N-1} x(m)\, e^{-j\frac{2\pi}{N}km} \qquad \textbf{(B.1)}$$

it is possible to find the sequence of a one-pole resonator

$$y_k(n) = \sum_{m=0}^{N-1} x(m)\, e^{j\frac{2\pi}{N}k(n-m)} \qquad \textbf{(B.2)}$$

which has a sample value at $n = N$ coinciding exactly with the actual DFT value. In other words, each DFT value $X(k)$ can be expressed in terms of the sample value at $n = N$ resulting from a linear filter process (one-pole filter).

It can be verified that

$$\begin{aligned} X(k) = y_k(N) &= \sum_{m=0}^{N-1} x(m)\, e^{j\frac{2\pi}{N}k(N-m)} \\ &= \sum_{m=0}^{N-1} x(m)\, e^{j\frac{2\pi}{N}kN}\ e^{-j\frac{2\pi}{N}km} \qquad \textbf{(B.3)} \\ &= \sum_{m=0}^{N-1} x(m)\, e^{-j\frac{2\pi}{N}km} \end{aligned}$$

The difference equation corresponding to the above one-pole resonator sequence (B.2), which is essential for the actual implementation, is given by

$$y_k(n) = e^{j\frac{2\pi}{N}k}\ y_k\ (n–1) + x\ (n) \qquad \textbf{(B.4)}$$

with $y(–1) = 0$ and pole location $p = e^{-j\frac{2\pi}{N}k}$. Being a one-pole filter, this recursive filter description yet contains complex multiplications, not very convenient for a DSP implementation. Instead, by using a two-pole filter with complex conjugate poles $p_{1,2} = e^{\pm j\frac{2\pi}{N}k}$ and only real multiplications in its difference equation,

$$v_k\ (n) = 2\cos\left(\frac{2\pi}{N}k\right) \cdot v_k\ (n–1)–v_k\ (n–2) + x(n) \qquad \textbf{(B.5)}$$

where $v_k(–1)$ and $v_k(–2)$ are zero.

In the Nth iteration, only a complex multiplication is needed to compute the DFT value, which is

$$X(k) = y_k(N) = v_k(N)–e^{-j\frac{2\pi}{N}k}\, v_k(N–1) \qquad \textbf{(B.6)}$$

However, the DTMF tone-detection process does not need the phase information of the DFT; squared magnitudes of the computed DFT values, in general, suffices. After some mathematical manipulation, it is found that

$$\begin{aligned} |X(k)|^2 &= y_k(N)\, y_k{}^*(N) \\ &= v_k{}^2(N) + v_k{}^2\ (N–1) –2\cos\left(\frac{2\pi}{N}k\right) v_k{}^2(N)\, v_k{}^2(N–1) \end{aligned} \qquad \textbf{(B.7)}$$

In short: For the actual DSP implementation, equations (B.5) and (B.7) are used to retrieve the spectral information from the input signal $x(n)$ for further evaluation. Note that equation (B.5) is the actual recursive linear filter expression, which is looped through for $n = 0$ .. N. Equation (B.7) is computed only once every N samples to determine the squared magnitudes.

Appendix C DTMF Tone Generator Code Listing

C.1 DTMF Tone Generator Executable on a TMS320C54x EVM

```
************************************************************
*   (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1996             *
************************************************************
*   Program Name:     DTMF tone generator                  *
*   File Name:        dtmf100e.asm                         *
*   File Description: This file contains the code for a    *
*                     DTMF tone generator                  *
*                     (TMS320C54x EVM version)             *
*                                                          *
*   Author:   Gunter Schmer                                *
*   Date:     08/20/96                                     *
*   Revision: 2.0                                          *
*   Latest working date:  08/20/96                         *
************************************************************
        .title ”dtmf encoder”
        .mmregs
        .include  ”varse.inc”         ; define global vars and consts
        .include  ”sectse.inc”        ; define global tables and buffers
        .include  ”globalse.inc”      ; globalized labels
****************************************
*       CODE STARTS HERE
****************************************
        .text
START
        ssbx    INTM            ; global interrupt disable
        ld      #0,DP           ; initialize data pointer
                                ; Initialize Hardware
        call    hwinit          ; hardware initialization subroutine
                                ; Initialize Software
        ssbx    SXM             ; data sign ext. before usage
        rsbx    OVM             ; no saturation of accu if overflow
        ssbx    FRCT            ; fractional mode bit, left shift of
                                ; multiplier to compensate for extra sign bit
        stm     #0040h,IMR      ; enable RINT1 interrupt
        stm     #00c8h,IFR      ; clear all pending serial port or timer interrupts
        rsbx    INTM            ; global interrupt enable
        ld      #aic_conf,DP    ; DP to variables
        mvmd    DRR1,rcv        ; dummy read
        mvdm    tra,DXR1        ; dummy write
****************************************
*       Main Program
****************************************
Main
        st      #PH_NBR,DIGIT_PTR      ; initialize pointer to PHNBR table
        st      #0,TASK                ; initialize beginning task
        st      #0,TIMER               ; initialize timer to zero
        stm     #DATA,AR5              ; AR5 points to testbuffer
next    idle    1
        call    tasks                  ; actual processing following RINT interrupt
        nop
                                       ; store to test buffer
                                       ; stm   #7fffh,BK
                                       ; ld    tra,A
                                       ; stl   A,*AR5+%
        b       next
```

```
done    b       done
*************************************************************
* Task Scheduler:
* Run this task scheduler section at the beginning of
* each RINT_ISR process
*
*
*************************************************************
tasks   cmpm    TASK,#00h
        bc      task1,NTC       ; if(TASK!=0) branch to task1
                                ; else branch to task0
task0   cmpm    TIMER,#00h
        bc      task01,NTC      ; branch if timer not zero
        mvdm    DIGIT_PTR,AR1   ; AR1 points to digit in PHNBR table
        st      #01h,TASK       ; TASK of next RINT_ISR is task1
        st      #TIME_LG,TIMER  ; set timer to long duration
        ld      *AR1+,A         ; load A with digit
        bc      done,alt        ; done if digit is –1 !
        mvmd    AR1,DIGIT_PTR   ; save new pointer to digit in PHNBR table
        call    unpack          ; unpack digit: A –> T1_OFS, T2_OFS
        b       task02
task01  ld      TIMER,A
        sub     #1,A
        stl     A,TIMER         ; decrement timer
task02  call    quiet
        b       task3           ; branch to task3
task1   cmpm    TIMER,#00h
        bc      task11,NTC      ; branch if timer not zero
        st      #00h,TASK       ; TASK of next RINT_ISR is task0
        st      #TIME_SH,TIMER  ; set timer to short duration
        b       task12
task11  ld      TIMER,A
        sub     #1,A
        stl     A,TIMER         ; decrement timer
task12  mvdm    T1_OFS,AR2      ; AR2 is offset for row–tone
        mvdm    T2_OFS,AR3      ; AR3 is offset for column–tone
        call    tone
task3   ret

****************************************
* Interrupt Service Routines
****************************************
RINT1_ISR:
        ld      #rcv,DP
        mvmd    DRR1,rcv
        andm    #0fffch,tra
        mvdm    tra,DXR1
        popm    ST1
        popm    ST0
        rete
         .end
;varse.inc
*************************************
***   Constants
*************************************
TIME_SH .set    440     ; pause duration (55 msec)
TIME_LG .set    360     ; tone duration  (45 msec)
DAC_OFS .set    000h    ; DAC offset
*************************************
***   Variables
```

```
************************************
        .bss    aic_conf,1
       .bss    rcv,1          ; receive variable
       .bss    tra,1          ; transmit variable
       .bss    DIGIT,1        ; current digit
       .bss    DIGIT_PTR,1    ; points to current digit in PHNBR table
       .bss    TIMER,1        ; timer counter
       .bss    T1_OFS,1       ; offset for tone 1 (row tone)
       .bss    T2_OFS,1       ; offset for tone 2 (column tone)
       .bss    TASK,1         ; holds the nbr of the task that is to be executed
;sectse.inc
************************************
***  Tables
************************************
*******************************************************
* The following table assembles the coefficients and  *
* initial conditions for the difference equations of  *
* the digital sinusoidal oscillators.                 *
*                                                     *
* In general:                                         *
*   DEQ:    y(n) = 2*cos(2pi*f/fs)*y(n–1) – y(n–2)    *
*   I.C.:   y(–1) = 0                                 *
*           y(–2) = –A*sin(2pi*f/fs)                  *
*   where   A  = desired amplitude of sine wave       *
*           f  = desired frequency of sine wave       *
*           fs = sampling frequency                   *
*                                                     *
* Example:                                            *
*       .word    cos(2pi*f/fs)*32768  ;coefficient    *
*       .word    0                    ;y(–1)          *
*       .word    –A*sin(2pi*f/fs)*32768      ;y(–2)   *
*                                                     *
*******************************************************
        .sect   ”tbl_tone”
TONES   .word  27980                ; row 1 coef
        .word  0                    ; y(n–1)
        .word  –1024*5204/10000     ; y(n–2)
        .word  26956                ; row 2 coef
        .word  0                    ; y(n–1)
        .word  –1024*5686/10000     ; y(n–2)
        .word  25701                ; row 3 coef
        .word  0                    ; y(n–1)
        .word  –1024*6203/10000     ; y(n–2)
        .word  24219                ; row 4 coef
        .word  0                    ; y(n–1)
        .word  –1024*6736/10000     ; y(n–2)
        .word  19073                ; col 1 coef
        .word  0                    ; y(n–1)
        .word  –1024*8132/10000     ; y(n–2)
        .word  16325                ; col 2 coef
        .word  0                    ; y(n–1)
        .word  –1024*8671/10000     ; y(n–2)
        .word  13085                ; col 3 coef
        .word  0                    ; y(n–1)
        .word  –1024*9168/10000     ; y(n–2)
        .word  9315                 ; col 4 coef
        .word  0                    ; y(n–1)
        .word  –1024*9587/10000     ; y(n–2)
```

```
*****************************************************************
* The following table contains offsets into the                 *
* tone table tbl_tone.                                          *
* The offset for the row–tone is the (upper byte) of each word  *
* The offset for the column–tone is the (lower byte + #12)      *
*                                                               *
*****************************************************************
       .sect   ”tbl_keys”
KEYS   .word   0903h         ; ’0’  example: 0903h ––> 3.row, 1.column (*3)
       .word   0000h         ; ’1’
       .word   0003h         ; ’2’
       .word   0006h         ; ’3’
       .word   0300h         ; ’4’
       .word   0303h         ; ’5’
       .word   0306h         ; ’6’
       .word   0600h         ; ’7’
       .word   0603h         ; ’8’
       .word   0606h         ; ’9’
       .word   0009h         ; ’A’
       .word   0309h         ; ’B’
       .word   0609h         ; ’C’
       .word   0909h         ; ’D’
       .word   0900h         ; ’E’ = ’*’
       .word   0906h         ; ’F’ = ’#’
****************************************************************
* The following table contains the phone number to be          *
* encoded. The format is as follows                            *
*                                                              *
* PH_NBR   .word   4,2,7,0Eh,0Fh,0Ah,3,–1                      *
*                                                              *
* is the foll nbr: 4 2 7 * # A 3   (–1 terminates the encoding)*
****************************************************************
       .sect   ”tbl_phnr”
PH_NBR .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   1,2,3,4,5,6,7,8,9,0,0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
       .word   –1
******************************************
* TEST BUFFER
******************************************
DATA   .usect ”testbuf”,8000h
******************************************
*       Stack setup
*****************************************
BOS     .usect  ”stack”,20h   ; setup stack
TOS     .usect  ”stack”,1     ; Top of stack at reset
**************************************************************
```

```
*   (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1996                *
*************************************************************
*   Program Name:     DTMF tone generator                     *
*   File Name:        dtmfsube.asm                            *
*   File Description: This file contains subroutines for a    *
*                     DTMF tone generator                     *
*                     (TMS320C54x EVM version)                *
*                                                             *
*   Author:   Gunter Schmer                                   *
*   Date:     08/20/96                                        *
*   Revision: 2.0                                             *
*   Latest working date:  08/20/96                            *
*************************************************************
        .mmregs
        .include  "globalse.inc"
*************************************************************
* SUBROUTINE:  unpack                                       *
* Description: Maps the key value (in A) into two offsets for   *
*              two tones and writes results into variables      *
*              T1_OFS, T2_OFS                                   *
* Uses:        AR2                                              *
* Input:       A                                                *
* Output:      none                                             *
*************************************************************
unpack  stlm    A,AR2           ; AR2 keys-map offset for unpacking
        ld      #DIGIT,DP       ; DP to variables
        nop                     ; pipeline conflict(1) + latency(1) with AR2
        ld      *AR2(KEYS),A    ; load A with keys-map value
        ld      *AR2(KEYS),B    ; load B with keys-map value
        and     #0f00h,A        ; mask out the row portion
        and     #000fh,B        ; mask out the column portion
        sftl    A,-8            ; right shift A by 8
        add     #12,B           ; add 4*3 words offset to point into column portion
        stl     A,T1_OFS        ; store tone 1 offset (row tone)
        stl     B,T2_OFS        ; store tone 2 offset (column tone)
        ret
*************************************************************
* SUBROUTINE:  tone                                         *
* Description: Generates the dual tone samples for DTMF         *
*              using offsets T1_OFS, T2_OFS                     *
* Uses:        AR2, AR3                                         *
* Input:       none                                             *
* Output:      none                                             *
*************************************************************
tone    ld      #0,B                ; clear B  ----ROW TONE----
        sub     *AR2(TONES+2),15,B  ; (B) = -(1/2)y(n-2), in high accumulator
        ltd     *AR2(TONES+1)       ; load (T) with y(n-1) and y(n-1) --> y(n-2)
        mpy     *AR2(TONES),A       ; (A) = coef*y(n-1)
        add     A,B                 ; (B) = coef*y(n-1) - (1/2)y(n-2)
        sth     B,1,*AR2(TONES+1)   ; 2*(B) --> y(n-1)
        ld      #0,B                ; clear B  ----COLUMN TONE----
        ; b     tone1               ; test only one oscillator
        sub     *AR3(TONES+2),15,B  ; (B) = -(1/2)y(n-2), in high accumulator
        ltd     *AR3(TONES+1)       ; load (T) with y(n-1) and y(n-1) --> y(n-2)
        mpy     *AR3(TONES),A       ; (A) = coef*y(n-1)
        add     A,B                 ; (B) = coef*y(n-1) - (1/2)y(n-2)
        sth     B,1,*AR3(TONES+1)   ; 2*(B) --> y(n-1)
tone1   add     *AR2(TONES+1),15,B  ; add two tone samples
        add     #DAC_OFS,B          ; add DAC offset
        sth     B,tra               ; write to transmit variable
```

```
       ret
*****************************************************************
* SUBROUTINE:  quiet                                            *
* Description: Generates a pause of specified duration          *
*                                                               *
* Uses:        AR2, AR3                                         *
* Input:       TIME_SH                                          *
* Output:      none                                             *
*****************************************************************
quiet  ld     #0,B           ; clear B
       add    #DAC_OFS,B     ; add DAC offset
       sth    B,tra          ; write to transmit variable
       ret
```

Appendix D DTMF Tone Detector Code Listing

D.1 DTMF Tone Detector Executable on a TMS320C54x EVM

```
/************************************************************
*   (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1996              *
************************************************************
*   Program Name:     DTMF tone decoder                     *
*   File Name:        globals.h                             *
*   File Description: defines channel information           *
*                                                           *
*   Author:   Gunter Schmer                                 *
*   Date:     10/15/96                                      *
*   Revision: 2.0                                           *
*   Latest working date: 10/15/96                           *
************************************************************/
/* globals */
/* struct used to pass on channel info to functions     */
typedef struct
{
    int *indata;          /* ptr to indata                        */
    int *taps;            /* ptr to filter states                 */
    int *energy;          /* ptr to energy template               */
    int *digitptr;        /* ptr to digit output array            */
    int digitlast;        /* last detected digit                  */
    int detectstat;       /* status of detector                   */
                          /* 0 – detector waiting for pause       */
                          /* 1 – detector ready to detect digit   */
    int err_flags;        /* error flags for dtmf checks          */
                          /* b0 – error signal strength           */
                          /* b1 – error reverse twist             */
                          /* b2 – error standard twist            */
                          /* b3 – error row's relative peak       */
                          /* b4 – error col's relative peak       */
                          /* b5 – error row's 2nd harmonic        */
                          /* b6 – error col's 2nd harmonic        */
                          /* b7 – error OVERFLOW of ACCA          */
                          /* b8..b15 are zero                     */
} DTMFCHANNEL;
/* --------------- channel#1 information ------------------------------*/
int  indata1[102];               /* receive buffer                    */
int  indata2[102];               /* process buffer                    */
int  digits[128];                /* output buffer for digit results   */
int  taps[32];                   /* filter states                     */
int  energy[16];                 /* energy template                   */
/* used for interrupt servicing and signaling a full buffer           */
int  *indataptr = &indata1[0];   /* ptr to receive buffer             */
int  indatastat = 0;             /* status of receive buffer          */
                                 /* 0 = buffer not ready to process   */
                                 /* 1 = buffer ready to process       */
/* initialized DTMFCHANNEL */
```

```
DTMFCHANNEL    channel1 = {   &indata2[0],
               &taps[0],
               &energy[0],
               &digits[0],
               0xFFFF,
               1,
               0   };
DTMFCHANNEL    *ptrchannel1 = &channel1;
/* --------------- end of channel#1 ------------------------*/
/* used for testing */
volatile unsigned int *testdataptr = (volatile unsigned int *) 0x8000;
/************************************************************
*   (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1996              *
************************************************************
*   Program Name:     DTMF tone decoder                     *
*   File Name:        main.c                                *
*   File Description: calls init functions                  *
*                     calls decoder functions               *
*                                                           *
*   Author:   Gunter Schmer                                 *
*   Date:     10/15/96                                      *
*   Revision: 2.0                                           *
*   Latest working date: 10/15/96                           *
************************************************************/
#include ”init.h”
#include ”globals.h”
#include ”mmregs.h”
#include ”dtmfsub.h”
/* prototypes */
void tasks(void);
void initArrays(void);
main()
{

  /*************** init HW and SW *****************/
  initWSGen();        /* initialize WS generator*/
  initArrays();       /* initialize arrays      */
  initSP1();          /* initialize SP1         */
  initAC01();         /* initialize AIC         */
  initInts(0x0040);   /* allow only RINT1       */
  /************** synch tasks() to RINT0 **********/
    for(;;)   {
        asm(”  IDLE   1”);    /* wait for interrupts           */
        tasks();              /* run tasks on completion of ISR */
  }
}
void tasks(void)
{
    int n;
        if(indatastat==1)  {
        /* reset indata status bit */
```

```
        indatastat = 0;
        /* copy indata1 to indata2 */
        for(n=0;n<102;n++)
        indata2[n] = indata1[n];
        /* perform gain control */
        gaincntrl(ptrchannel1);
        /* perform goertzel     */
        goertzel(ptrchannel1);
        /* perform dtmf digit validation */
        dtmfchecks(ptrchannel1);
    }
}
void initArrays(void)
{
  int n;
  for(n=0;n<102;n++)   {
      indata1[n] = 0;
      indata2[n] = 0;
  }
  for(n=0;n<128;n++)
      digits[n] = 0;
  for(n=0;n<32;n++)
      taps[n] = 0;
  for(n=0;n<16;n++)
      energy[n] = 0;
}

*************************************************************
*   (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1996              *
*************************************************************
*   Program Name:     DTMF tone decoder                     *
*   File Name:        dtmfsub.asm                           *
*   File Description: This file contains the subroutines    *
*                     used for a DTMF tone detector         *
*                     (TMS320C54x EVM version)              *
*   Author:   Gunter Schmer                                 *
*   Date:     10/15/96                                      *
*   Revision: 2.0                                           *
*   Latest working date: 10/15/96                           *
*************************************************************
    .mmregs
    .include   "globals.inc"      ;global labels
    .include   "tables.inc"       ;various tables
*****************************************************************
* SUBROUTINE: gaincntrl (approx. 705 cycles)
*
* Input: none
* Output: none
* Uses:
*
* Description: Scales a block of input data in order to ensure
```

```
*              no overflow when running the goertzel routine
*              approx. 490 cycles per N=102 sample frame
*              approx. 5 cycles   per sample (negligable)
*
*              gain_power =   (1/64) * SUM{0,N-1}[x(n)*x(n)]
*                         =   (1/64) * N * signalpower
*
*              gain_lim   =   (1/64) * (1/N)
*                         =   minimum signalpower (-..dB)
*
*              gain_const =   0.5*sqrt(2)*sqrt(64/N)
*              gain_amp   =   gain_const * sqrt(gain_power)
*              gain_lev   =   1/N
*              gain_scale =   gain_lev/gain_amp
*
*              x(n) = x(n) * gain_scale
*
* FRCT=1, SXM=1
*****************************************************************
    .text
_gaincntrl:
    nop
    pshm    ST0
    pshm    ST1
    pshm    AR1
    pshm    AR6
    pshm    AR7
    nop
    frame   #-10    ;---- local variables -----
                    ; *SP(0) = gain_pow (HI)
                    ; *SP(1) = gain_pow (LO)
                    ; *SP(2) = gain_lim (HI)
                    ; *SP(3) = gain_lim (LO)
                    ; *SP(4) = gain_amp
                    ; *SP(5) = gain_lev
                    ; *SP(6) = gain_scale
                    ; *SP(7) = gain_const
                    ; *SP(8) = gain_delta
                    ; *SP(9) = ptr to indata
                    ;---- arguments -----------
                    ; (A) = ptr to dtmfchannel
    stlm    A,AR5
    nop
    ld  *AR5,A
    stl A,*SP(9)
                      ;value initializations
    ld  #0005h,16,A   ;(A)  = 00050000h
    or  #0505h,A      ;(A)  = 00050505h = 1/(64*N) (Q31)
    sth A,*SP(2)      ;(A)  --> gain_lim  (32bit)
    stl A,*SP(3)
    ld  #00E3h,A      ;(AL) = 00E3h = (0.5*sqrt(2))/N
```

```
    stl A,*SP(5)       ;(AL) --> gain_lev  (16bit)
    ld  #6564h,A       ;(AL) = 6564h = sqrt(64/102)
    stl A,*SP(7)       ;(AL) --> gain_const
    ld  #4000h,A       ;(AL) = 4000h = 0.5
    stl A,*SP(8)       ;(AL) --> gain_delta

    ssbx   FRCT
                       ;compute signal power
    ld  *SP(9),A
    stlm   A,AR2       ;AR2 points to indata2 buffer
    rptz   B,#(N-1)    ;for(n=0;n<N-1;n++)
     squra  *AR2+,B    ;  (B)=(B)+x(n)*x(n)
    sfta   B,-6        ;(B)=(1/64)*(B)
    sth B,*SP(0)       ;(B) --> gain_pow   (32bit)
    stl B,*SP(1)       ;        gain_pow = (N/64)*sigpower
    ld  *SP(2),16,A    ;(A) = gain_lim = 1/(64*N)  (32bit)
    or  *SP(3),A
    sub A,B            ;(B) = gain_pow - gain_lim
    bc  gain2,BLEQ     ;if(gain_pow > gain_lim)
                       ;{
                       ; compute gain_amp = sqrt(gain_pow)
    st  #4000h,*SP(4)  ;   initialize gain_amp estimate
    st  #4000h,*SP(8)  ;   initialize gain_delta to 0.5
    mvmm   SP,AR2
    mar *+AR2(8)       ;   AR2 points to gain_delta
    stm #(sqrt_iterations-1),BRC
    rptbd  sqrtloopend-1
    squr   *SP(4),A    ;   (A) = yold*yold
    dsub   *SP(0),A    ;   (A) = yold*yold - x
sqrtloop
    bcd sqrt1,AGT
    ld  *SP(4),16,B    ;   (B) = yold
    sub *AR2,15,B      ;   (B) = yold - (1/2)*delta
                       ;   if(y needs to be larger)
    add *AR2,16,B      ;     (B) = yold + (1/2)*delta
sqrt1  sth B,*SP(4)    ;   (BH) --> ynew
    ld  *AR2,15,A      ;   (AH) = (1/2)delta
    sth A,*AR2         ;   (AH) --> delta new
    squr   *SP(4),A
    dsub   *SP(0),A    ;   (A) = yold*yold - x
sqrtloopend
                       ; compute gain_amp = sqrt(64/N)*gain_amp
    ld  *SP(7),T       ;   (T) = gain_const
    mpy *SP(4),A       ;   (A)=sqrt(64/102)*gain_amp
    sth A,*SP(4)       ;   (AH) --> gain_amp
                       ; compute gain_lev/gain_amp
    ld  *SP(5),16,A    ;   (AH) = gain_lev
    rpt #(16-1)
     subc  *SP(4),A    ;   compute (gain_lev/gain_amp)
    and #0ffffh,A      ;   retain quotient (AL), mask remainder
    sfta   A,15        ;   shift quotient into high accu
```

```
    sth A,*SP(6)       ;   (AH) –> gain_scale
                       ; scale indata with (gain_lev/gain_amp)
    ld  *SP(9),A       ;
    stlm    A,AR2      ;   AR2 points to indata
    mvmm    SP,AR3     ;
    mar *+AR3(6)       ;   AR3 points to gain_scale
    stm #(N–1),BRC     ;
    rptb    gain1–1    ;   for(n=0;n<N;n++)
    mpy *AR2,*AR3,A    ;      (A) = x(n)*(gain_lev/gain_amp)
    sth A,*AR2+        ;      (AH) –> x(n)
gain1                  ;}

gain2  frame  #10
    nop
    popm    AR7
    popm    AR6
    popm    AR1
    popm    ST1
    popm    ST0
    ret
*************************************************************************
* SUBROUTINE: goertzel         (w/o OV check approx. 8861 cycles)
*
* Input:  none
* Output: none
* Uses:
*
* Description:
* This new version of the subroutine operates on a block of data
* 102 samples long and computes the following difference equation
* It does thereby not check for overflow and assumes a prescaling
* gain control function to protect from possible overflow.
*
* vk(n) = 2*coef*vk(n–1) – vk(n–2) + x(n)
*
* Coefficients are in order starting with location COEF1st
* COEF1st .word    xxxx    ;coef1
*     .word yyyy           ;coef2
*     ....
*
* Data vk(n–1), vk(n–2) is ordered as follows
*   v15(n–1)
*   v15(n–2)
*   v14(n–1)
*   v14(n–2)
*   ....
*   v0(n–1)
*   v0(n–2)  <––AR2
*
* FRCT=1, SXM=1
*
```

```
*********************************************************************
_goertzel:
    nop
    pshm    ST0
    pshm    ST1
    pshm    AR1
    pshm    AR6
    pshm    AR7
                          ;---- local variables -----
                          ; none
                          ;---- arguments -----------
                          ; (A)    = ptr to DTMFCHANNEL
    rsbx    OVA           ;clear overflow bit for A
    ssbx    FRCT          ;set fractional mode
    stlm    A,AR5         ;AR5 is used as pointer to struct elements
                          ; *AR5(0) = ptr to indata
                          ; *AR5(1) = ptr to taps
                          ; *AR5(2) = ptr to energy template
                          ; *AR5(3) = digitptr
                          ; *AR5(4) = digitlast
                          ; *AR5(5) = detectstat
                          ; *AR5(6) = err_flags
    ld  *AR5,A            ;
    stlm    A,AR1         ;AR1 points to indata buffer
    ld  *AR5(1),A         ;
    add #31,A             ;
    stlm    A,AR6         ;AR6 points to end of taps block
    stm #(N-1),BRC

goer1   rptb    goer2-1   ;for(n=0;n<N;n++)     {
    mvmm    AR6,AR2       ;   AR2 points to end of taps block
    stm #COEF1st,AR3      ;   AR3 points to Coefficients
    ld  *AR1+,16,A        ;   (A) = x(n)
                          ;   ------ filter k=0 -----------------
    sub *AR2-,16,A,B      ;   (B) = -vk(n-2) + x(n)
    mac *AR2,*AR3,B       ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
    mac *AR2,*AR3+,B      ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2          ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-           ;   (B)     ---> vk(n-1)
                          ;   ------ filter k=1 -----------------
    sub *AR2-,16,A,B      ;   (B) = -vk(n-2) + x(n)
    mac *AR2,*AR3,B       ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
    mac *AR2,*AR3+,B      ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2          ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-           ;   (B)     ---> vk(n-1)
                          ;   ------ filter k=2 -----------------
    sub *AR2-,16,A,B      ;   (B) = -vk(n-2) + x(n)
    mac *AR2,*AR3,B       ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
    mac *AR2,*AR3+,B      ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2          ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-           ;   (B)     ---> vk(n-1)
```

```
                        ;   ------ filter k=3 -----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=4 -----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=5 -----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=6 -----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=7 -----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=8 -----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=9 -----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=10 ----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
mac *AR2,*AR3+,B        ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
delay   *AR2            ;   vk(n-1) ---> vk(n-2)
sth B,*AR2-             ;   (B)     ---> vk(n-1)
                        ;   ------ filter k=11 ----------------
sub *AR2-,16,A,B        ;   (B) = -vk(n-2) + x(n)
mac *AR2,*AR3,B         ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
```

```
    mac *AR2,*AR3+,B       ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2           ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-            ;   (B)     ---> vk(n-1)
                           ;   ------ filter k=12 -----------------
    sub *AR2-,16,A,B       ;   (B) = -vk(n-2) + x(n)
    mac *AR2,*AR3,B        ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
    mac *AR2,*AR3+,B       ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2           ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-            ;   (B)     ---> vk(n-1)
                           ;   ------ filter k=13 -----------------
    sub *AR2-,16,A,B       ;   (B) = -vk(n-2) + x(n)
    mac *AR2,*AR3,B        ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
    mac *AR2,*AR3+,B       ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2           ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-            ;   (B)     ---> vk(n-1)
                           ;   ------ filter k=14 -----------------
    sub *AR2-,16,A,B       ;   (B) = -vk(n-2) + x(n)
    mac *AR2,*AR3,B        ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
    mac *AR2,*AR3+,B       ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2           ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-            ;   (B)     ---> vk(n-1)
                           ;   ------ filter k=15 -----------------
    sub *AR2-,16,A,B       ;   (B) = -vk(n-2) + x(n)
    mac *AR2,*AR3,B        ;   (B) = coef*vk(n-1) - vk(n-2) +x(n)
    mac *AR2,*AR3+,B       ;   (B) = 2*coef*vk(n-1) - vk(n-2) +x(n)
    delay   *AR2           ;   vk(n-1) ---> vk(n-2)
    sth B,*AR2-            ;   (B)     ---> vk(n-1)
goer2                      ;}
goer3   popm    AR7
    popm    AR6
    popm    AR1
    popm    ST1
    popm    ST0
    ret
*******************************************************************************
* SUBROUTINE: dtmf_checks             (approx 788 cycles)
*
* Input:  none
* Output: A (status if valid digit)
* Uses:
*
* Description:
*  - energy computation
*  - initialize filter taps to zero
*  - maximum search
*  - DTMF status check
*  - signal strength check
*  - twist check
*  - relative peak check
*  - 2nd harmonic check
*  - digit validation
```

```
*
************************************************************************
_dtmfchecks:
    nop
    pshm    ST0
    pshm    ST1
    pshm    AR1
    pshm    AR6
    pshm    AR7
    nop
    frame   #0          ;---- local variables -----
                        ; none
                        ;---- arguments -----------
                        ; (A)    = ptr to dtmfchannel struct
    ssbx    FRCT
    stlm    A,AR5       ;AR5 is used as ptr to dtmfchannel struct
                        ; *AR5(0) = ptr to indata
                        ; *AR5(1) = ptr to taps
                        ; *AR5(2) = ptr to energy template
                        ; *AR5(3) = digitptr
                        ; *AR5(4) = digitlast
                        ; *AR5(5) = detectstat
                        ; *AR5(6) = err_flags
;-----------------------------------------------------------------------
; Energy computation
;
;  y(N)y*(N) = vk(N)*vk(N) – 2*coef*vk(N)*vk(N–1) + vk(N–1)*vk(N–1)
;
; Coefficients are in pmem in order starting with location COEF1st
; COEF1st .word    xxxx    ;coef0  <––AR3
;          .word   yyyy    ;coef1
;     ....
;
; Data vk(n–1), vk(n–2) is ordered as follows in dmem
;   v15(n–1)
;   v15(n–2)
;   v14(n–1)
;   v14(n–2)
;   ....
;   v0(n–1)
;   v0(n–2)  <––AR2
;
; FRCT = 1 , OVLY = 1
; AR2: points to v1(N–1)
; AR3: points to COEF1st
;
;-----------------------------------------------------------------------
ener    ld  *AR5(1),A
    add #31,A
    stlm    A,AR2           ;AR2 points to end of filter taps block
    stm #COEF1st,AR3        ;AR3 points to beg of coefficient table
```

```
    ld  *AR5(2),A
    stlm    A,AR4       ;AR4 points to energy template
    stm #(16–1),BRC
    rptb    ener1–1     ;for(k=0;k<16;k++)   {
     ld *AR2,16,A       ; (A) = vk(N–1), A(32:16) loaded
     mpya   *AR2–       ; (B) = vk(N–1)*vk(N–1), (T) = vk(N–1)
     mpy    *AR2,A      ; (A) = vk(N)*vk(N–1)
     ld *AR3+,T         ; (T) = coef
     mpya   A           ; (A) = coef*vk(N)*vk(N–1)
     sub    A,1,B       ; (B) = –2*coef*vk(N)*vk(N–1) + vk(N–1)*vk(N–1)
     ld *AR2,T          ; (T) = vk(N)
     mac    *AR2–,B     ; (B) = vk(N)*vk(N) – 2*coef*vk(N)*vk(N–1) + vk(N–1)*vk(N–1)
     sth    B,*AR4+     ; store result E[k] into energy data block
ener1                   ;}
;-------------------------------------------------------------------------
; Initialize Filter Taps to zero for next round
;
;-------------------------------------------------------------------------
    ld  *AR5(1),A
    stlm    A,AR2       ;AR2 points to beginning of filter taps
    ld  #0,A
    rpt #(32–1)         ;for(i=0;i<32;i++)
     stl    A,*AR2+     ;   initialize filter taps for next round
BREAK   nop             ;<--- breakpoint for testing
;-------------------------------------------------------------------------
; Alternative Maximum search
; result: AR3 points to max row energy ROWMAX
;         AR4 points to max col energy COLMAX
;-------------------------------------------------------------------------
max_search:
    ld  *AR5(2),A
    stlm    A,AR2       ;AR2 points to ENERGY table (row energies)
    mvmm    AR2,AR3     ;AR3 points to ENERGY table
    stm #(4–1),BRC
    rptb    max1–1      ;for(k=0;k<4;k++)
    ld  *AR3,A          ;   (A)=oldmax
    sub *AR2,A          ;   (A)=oldmax–E[k]
    bc  max11,AGT       ;   if(oldmax < E[k])
    mvmm    AR2,AR3     ;      AR3 points to newmax
max11   mar *AR2+       ;   increment pointer
max1                    ;result: AR3 points to ROWMAX
                        ;        AR2 points to column energies
    mvmm    AR2,AR4     ;AR4 points to ENERGY table (column energies)
    stm #(4–1),BRC
    rptb    max2–1      ;for(k=0;k<4;k++)
    ld  *AR4,A          ;   (A)=oldmax
    sub *AR2,A          ;   (A)=oldmax–E[k]
    bc  max21,AGT       ;   if(oldmax < E[k])
    mvmm    AR2,AR4     ;   AR4 points to newmax
max21   mar *AR2+       ;   increment pointer
max2                    ;result: AR4 points to COLMAX
```

```
              ;        AR2 points to column energies
;-------------------------------------------------------------------------
; DTMF Status Check:
;   if(detectstat == 0)
;      if( max(ROWMAX,COLMAX) < THR_PAU )
;         detectstat = 1, detector enabled
;      terminate all checks
;   else
;      continue
;
; AR3 points to ROWMAX
; AR4 points to COLMAX
;-------------------------------------------------------------------------
stat_check
    andm   #0080h,*AR5(6) ;clear error flags, except OVA
    bitf   *AR5(5),#0001h ;
    bc  sig_check,tc      ;if(detectstat == 1) continue with sig_check
    ld  *AR3,A            ;
    ld  *AR4,B            ;else
    max A                 ;  (A)=max(ROWMAX,COLMAX)
    sub #THR_PAU,A        ;  (A)=max(ROWMAX,COLMAX) – THR_PAU
    bc  stat1,ageq        ;  if((A)<0)
    st  #0001h,*AR5(5)    ;     detectstat = 1
stat1   b   ende          ;  terminate all checks
;-------------------------------------------------------------------------
; Signal strength check:
;    if(ROWMAX+COLMAX > THR_SIG)  continue
;    else  terminate all checks
;
; AR3 points to ROWMAX
; AR4 points to COLMAX
;-------------------------------------------------------------------------
sig_check:
    ld  *AR3,A
    add *AR4,A         ;(A) = ROWMAX+COLMAX
    sub #THR_SIG1,A,B  ;
    bc  ende,BLEQ      ;if(ROWMAX+COLMAX <= THR_SIG1)
                       ;    signal is not possible tone
                       ;    terminate checks
    sub #THR_SIG2,A,B  ;
    bc  err_sig,BLEQ   ;if(ROWMAX+COLMAX <= THR_SIG2)
                       ;    signal is possible tone
                       ;    however does not meet THR_SIG2
                       ;    branch to err_sig
    b   twist_check
err_sig:               ;**** record signal strength error ****
    orm #0001h,*AR5(6) ; set signal strength error flag (b0)
    b   ende           ; terminate checks
;-------------------------------------------------------------------------
; Twist check:
;    if(ROWMAX > COLMAX)
```

```
;       check reverse twist:
;       if((COLMAX/ROWMAX) > THR_REVTWI)   continue
;       else terminate all checks
;    else
;       check standard twist:
;       if((ROWMAX/COLMAX) > THR_STDTWI)   continue
;       else  terminate all check
;
; AR3 points to ROWMAX
; AR4 points to COLMAX
;-----------------------------------------------------------------------
twist_check:
    ld  *AR3,A
    sub *AR4,A
    bc  twist2,ALEQ        ;if(ROWMAX > COLMAX)
twist1  ld  *AR4,16,A      ;   (AH)=COLMAX
    rpt #(16-1)            ;
     subc   *AR3,A         ;   compute (COLMAX/ROWMAX)
    and #0ffffh,A          ;   retain quotient (AL), mask remainder
    sfta    A,15           ;   shift quotient into AH
    sub #THR_REVTWI,16,A;
    bc  err_revtwi,ALEQ    ;   if((COLMAX/ROWMAX) <= THR_REVTWI)
                           ;      branch to err_revtwi
    b   rel_check
                           ;else
twist2  ld  *AR3,16,A      ;   (AH)=ROWMAX
    rpt #(16-1)            ;
     subc   *AR4,A         ;   compute (ROWMAX/COLMAX)
    and #0ffffh,A          ;   retain quotient (AL), mask remainder
    sfta    A,15           ;   shift quotient into AH
    sub #THR_STDTWI,16,A   ;
    bc  err_stdtwi,ALEQ    ;   if((COLMAX/ROWMAX) <= THR_STDTWI)
                           ;      branch to err_stdtwi
    b   rel_check
err_revtwi:                ;**** record reverse twist error ****
    orm #0002h,*AR5(6)     ; set reverse twist error flag (b1)
    b   ende               ; terminate checks
err_stdtwi:                ;**** record standard twist error ****
    orm #0004h,*AR5(6)     ; set standard twist error flag (b2)
    b   ende               ; terminate checks
;-----------------------------------------------------------------------
; Relative peak check:
;
;    find rel row peak --> RELPEAK
;    if(RELPEAK/ROWMAX < THR_ROWREL)  continue
;    else terminate all checks
;
;    find rel col peak --> RELPEAK
;    if(RELPEAK/COLMAX < THR_ROWREL)  continue
;    else terminate all checks
;
```

```
; AR3 points to ROWMAX
; AR4 points to COLMAX
;-------------------------------------------------------------------------
rel_check:                ;------ relative row peak ratio check -------
    ld  *AR5(2),A         ;
    stlm    A,AR0         ;AR0 points to ENERGY table (row frequencies)
    ld  #0,B              ;(BH)=0
    stm #(4-1),BRC        ;
    rptb    rel12-1       ;for(k=0;k<4;k++)
     ld *AR0,16,A         ;  (AH)=E[k]
     cmpr   0,AR3         ;
     bc rel11,TC          ;  if(AR0==AR3)
     max    B             ;    (B)=newmax=max(oldmax,E[k])
rel11    mar    *AR0+     ;

rel12   rpt #(16-1)       ;
     subc   *AR3,B        ;compute (RELPEAK/ROWMAX)
    and #0ffffh,B         ;retain quotient (BL), mask remainder
    sfta    B,15          ;shift quotient into BH
    sub #THR_ROWREL,16,B  ;
    bc  err_rowrel,BGEQ   ;if((RELPEAK/ROWMAX) >= THR_ROWREL)
                          ;      branch to err_rowrel
                          ;------ relative column peak ratio check ----
rel2    ld  #0,B          ;(BH)=0
    stm #(4-1),BRC        ;
    rptb    rel22-1       ;for(k=4;k<8;k++)
     ld *AR0,16,A         ;  (AH)=E[k]
     cmpr   0,AR4         ;
     bc rel21,TC          ;  if(AR0==AR4)
     max    B             ;    (B)=newmax=max(oldmax,E[k])
rel21    mar    *AR0+     ;

rel22   rpt #(16-1)       ;
     subc   *AR4,B        ;compute (RELPEAK/COLMAX)
    and #0ffffh,B         ;retain quotient (BL), mask remainder
    sfta    B,15          ;shift quotient into BH
    sub #THR_COLREL,16,B  ;
    bc  err_colrel,BGEQ   ;if((RELPEAK/COLMAX) >= THR_COLREL)
                          ;      branch to err_colrel
    b   sec_check
err_rowrel:               ;**** record row's rel peak error ****
    orm #0008h,*AR5(6)    ; set row's rel peak error flag (b3)
    b   ende              ; terminate checks
err_colrel                ;**** record col's rel peak error ****
    orm #0010h,*AR5(6)    ; set col's rel peak error flag (b4)
    b   ende              ; terminate checks
;-------------------------------------------------------------------------
; Second Harmonics check
;    if( (ROW2nd/ROWMAX) < THR_ROW2nd )  continue
;    else terminate all checks
;
```

```
;    if( (COL2nd/COLMAX) < THR_COL2nd )  continue
;    else terminate all checks
;
; AR3 points to ROWMAX
; AR4 points to COLMAX
;-----------------------------------------------------------------------
sec_check:
    stm #8h,AR0           ;offset into 2nd harmonics energies
                          ;------ 2nd harmonics check for rows -------
    mar *AR3+0            ;AR3 points to corresponding ROW2nd
    ld  *AR3-0,16,A       ;(AH)=ROW2nd, AR3 points to ROWMAX
    sub *AR3,16,A,B       ;(BH)=ROW2nd-ROWMAX
    bc  err_row2nd,BGEQ   ;if(ROW2nd<ROWMAX)
    rpt #(16-1)           ;
     subc   *AR3,A        ;   compute (ROW2nd/ROWMAX)
    and #0ffffh,A         ;   retain quotient (AL), mask remainder
    sfta    A,15          ;   shift quotient into AH
    sub #THR_ROW2nd,16,A  ;
    bc  err_row2nd,AGEQ   ;   if((ROW2nd/ROWMAX) >= THR_ROW2nd)
                          ;      branch to err_row2nd
                          ;------ 2nd harmonics check for columns ----
sec2    mar *AR4+0        ;AR4 points to corresponding COL2nd
    ld  *AR4-0,16,A       ;(AH)=COL2nd, AR4 points to COLMAX
    sub *AR4,16,A,B       ;(BH)=COL2nd-COLMAX
    bc  err_col2nd,BGEQ   ;if(COL2nd<COLMAX)
    rpt #(16-1)           ;
     subc   *AR4,A        ;   compute (COL2nd/COLMAX)
    and #0ffffh,A         ;   retain quotient (AL), mask remainder
    sfta    A,15          ;   shift quotient into AH
    sub #THR_COL2nd,16,A  ;
    bc  err_col2nd,AGEQ   ;   if((COL2nd/COLMAX) >= THR_COL2nd)
                          ;      branch to err_col2nd
    b   digit_map
err_row2nd:               ;**** record row's 2nd harm error ****
    orm #0020h,*AR5(6)    ; set row's 2nd harm error flag (b5)
    b   ende              ; terminate checks
err_col2nd:               ;**** record col's 2nd harm error ****
    orm #0040h,*AR5(6)    ; set col's 2nd harm error flag (b6)
    b   ende              ; terminate checks
;-----------------------------------------------------------------------
; Map detected tone pair to digit
;
; AR3 points to ROWMAX
; AR4 points to COLMAX
;-----------------------------------------------------------------------
digit_map:
;---------- find row and col numbers ----------------------------
    ld  *AR5(2),A
    stlm    A,AR0         ;load offset
    nop
    nop
```

```
    mar *AR3-0            ;AR3 contains row#
    mar *+AR0(4)
    mar *AR4-0            ;AR4 contains col#

;---------- map row and column numbers to actual digit -------------
    ld  *(AR3),8,A        ;load row nbr in upper byte of AL
    add *(AR4),A          ;load column nbr in lower byte of AL
    stm #KEYS,AR2         ;AR2 points to key mapping table
    stm #0,AR3            ;AR3 is used as counter
    stm #(16-1),BRC
    rptb   digit3-1       ;for(k=0;k<16;k++)   {
     sub   *AR2+,A,B
     bc digit31,bneq      ;   if(A == KEYS[k])  insert delay
      b digit4            ;       branch to ende
digit31  mar   *AR3+      ;  increment counter
digit3                    ;}

;-------------------------------------------------------------------------
; Validate digit from previous digit
;
;-------------------------------------------------------------------------
digit4                    ;now AR3 contains decoded digit
    ld  *AR5(3),A
    stlm   A,AR4          ;AR4 used as digitptr
    ldm AR3,A             ;load current digit
    sub *AR5(4),A,B       ;compare with last digit
    bc  digit5,bneq       ;if current digit and last digit are the same
     stl   A,*AR4+        ;   store VALID DIGIT into DIGITS
     ldm   AR4,A          ;
     stl   A,*AR5(3)      ;   update pointer
     st #0,*AR5(5)        ;   disable detector and wait for pause
digit5  ldm AR3,A         ;store decoded digit
    stl A,*AR5(4)         ; into digitlast

ende   frame  #0
    popm   AR7
    popm   AR6
    popm   AR1
    popm   ST1
    popm   ST0
    nop
    ret
```

```
*************************************************************
*   (C) COPYRIGHT TEXAS INSTRUMENTS, INC. 1996              *
*************************************************************
*   Program Name:     DTMF tone decoder                     *
*   File Name:        tables.inc                            *
*   File Description: tables for dtmfsub.asm                *
*                                                           *
*   Author:   Gunter Schmer                                 *
*   Date:     10/15/96                                      *
*   Revision: 2.0                                           *
*   Latest working date: 10/15/96                           *
*************************************************************
************************************
***   Constants
************************************
N       .set    102                 ;max count for DFT loops
THR_SIG1    .set    1900*32768/10000   ;threshold for possible tone
THR_SIG2    .set    2150*32768/10000   ;threshold for definite tone
THR_PAU     .set    458*32768/10000    ;threshold for pause energy
THR_STDTWI  .set    3162*32768/10000   ;threshold for standard twist (–5dB)
THR_REVTWI  .set    1259*32768/10000   ;threshold for reverse twist (–9dB)
THR_ROWREL  .set    2000*32768/10000   ;threshold for row's relative peak ratio
THR_ROW2nd  .set    2000*32768/10000   ;threshold for row's 2nd harmonic ratio
THR_COLREL  .set    2000*32768/10000   ;threshold for col's relative peak ratio
THR_COL2nd  .set    2000*32768/10000   ;threshold for col's 2nd harmonic ratio
sqrt_iterations .set 16              ;iterations for sqrt in gain control function
************************************
***   Tables
************************************
*******************************************************
* The following table holds the coefficients for      *
* the 16 Goertzel filters, computed as follows        *
*                                                     *
*       .word    cos(2pi*k/N)*32768  ;coefficient     *
*                                                     *
* with  N = 102  (1st and 2nd Harmonics)              *
*       k =  9, 10, 11, 12, 15, 17, 19, 21  (1. Harm) *
*       k = 18, 20, 22, 24, 31, 34, 38, 42  (2. Harm) *
*                                                     *
*******************************************************
        .sect   "tbl_coef"
COEF1st .word   27980       ; 1st Harmonics
    .word   26956
    .word   25701
    .word   24219
    .word   19073
    .word   16325
    .word   13085
    .word   9315
COEF2nd .word   14739       ; 2nd Harmonics
    .word   11414
```

```
    .word   7549
    .word   3032
    .word   –10565
    .word   –16503
    .word   –22318
    .word   –27472
*****************************************************************
* Digit mapping table                                           *
*   Low byte : row number                                       *
*   High byte: column number                                    *
*****************************************************************
    .sect   ”tbl_keys”
KEYS    .word   0301h           ; ’0’
    .word   0000h               ; ’1’
    .word   0001h               ; ’2’
    .word   0002h               ; ’3’
    .word   0100h               ; ’4’
    .word   0101h               ; ’5’
    .word   0102h               ; ’6’
    .word   0200h               ; ’7’
    .word   0201h               ; ’8’
    .word   0202h               ; ’9’
    .word   0003h               ; ’A’
    .word   0103h               ; ’B’
    .word   0203h               ; ’C’
    .word   0303h               ; ’D’
    .word   0300h               ; ’E’ = ’*’
    .word   0302h               ; ’F’ = ’#’
*****************************************************************
* Test Buffer for encoding results                             *
*****************************************************************
DATA    .usect  ”buffer”,8000h ;length 32k words
```