General-Purpose Tone Decoding and DTMF Detection

APPLICATION REPORT: SPRA082

Craig Marven
Regional Technology Center
Bedford, England
Texas Instruments

Digital Signal Processing Solutions

TEXAS INSTRUMENTS

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

General-Purpose Tone Decoding and DTMF Detection

Abstract

This report describes a single-chip solution for concurrent DTMF and general-tone decoding or expanded, general-tone decoding only. These facilities are provided on a special program on the TMS320C17 or TMS320E17. The term TMS320C17 applies to both throughout the report.

The report covers the following topics:

- ❑ Theory of Operation
  - ■ Software
  - ■ TMS320C17 Features
  - ■ Hardware
- ❑ Implementation
  - ■ Software Implementation
  - ■ Utilization of TMS320C17 Resources
  - ■ Hardware Implementation
- ❑ Host Interface
  - ■ Host Write Cycle
  - ■ Host Access Considerations
  - ■ Host Interface Registers
  - ■ Register Read Functions

- Applications and Customization
  - Secure Off-Site Control
  - Call Monitoring
  - DTMF Telephone Tester
  - Customization for User Applications
  - Flexibility through Programmability
- Conclusions and References

The report also includes the following appendixes:

Appendix A	Tone Detector Source Code
Appendix B	PC Application Program
Appendix C	Power Detector Operational Considerations

Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to *dsph@ti.com.* Questions receive prompt attention and are usually answered within one business day.

Introduction

The use of the Dual-Tone Multi-Frequency (DTMF) signaling scheme within telecommunications systems has become widespread over the past few years. It is replacing the older type of pulse oriented dialing methods in telephones worldwide, and also finds application in a number of other equipment types, such as personal computer (PC) telephone peripherals, remote signaling schemes etc.

In parallel with the universal DTMF standard, the various telecommunications companies or public authorities (PTTs) around the world use a number of different tones to signal call progress parameters. Examples include busy tones, number unobtainable, timing tones, etc. Although DTMF operates to an internationally recognized standard, these additional tones do not. Therefore, there is often a need for a programmable tone detection capability operating concurrently with standard format DTMF decoding. Alternatively, there are also many possible areas of application for an expanded programmable tone decoding facility without DTMF capability.

This document describes a single-chip solution to fulfill the requirement for concurrent DTMF and general-tone decoding or expanded, general-tone decoding only. These facilities are provided by a special program on the TMS320C17 or TMS320E17 first-generation digital signal processor (DSP). The term TMS320C17 should be taken to apply to both the TMS320C17 and TMS320E17 for the remainder of this report. See Reference [6] for full information on these devices.

The TMS320C17 is particularly suited to tone detection as it possesses on-chip serial ports, a hardware multiplier and a 200 nanosecond (ns) instruction cycle time. These last two features allow high-speed calculation of the digital filter equations which implement the core of the tone decoding function.

The main functions of the tone detector described in this report are as follows:

1. DTMF tone decoding to international standards
2. Power measurement at six selectable frequencies in the band 300-3400 Hz
3. Power measurement at three selectable frequencies simultaneously with DTMF tone decoding
4. Selectable bandwidth and resolution of frequency selection
5. Timestamping of tone arrival and departure
6. Selectable thresholds to define tone arrival and departure
7. Interrupt generation on tone arrival, departure or change
8. Interrupt generation on unidentified tone
9. Interrupt generation on validation of DTMF digits

10. Variable gain setting on input to receivers

11. Self test

In addition to a detailed description of the operation of the software within the TMS320C17, a complete solution to a tone detection peripheral for an IBM XT or AT compatible PC is presented. Remember that this is just one possible application for the tone detection TMS320C17, it could equally be paired with any other host CPU.

This report is divided into seven sections and three appendices. A brief outline of the contents of each section serves as a useful guide. Although some sections refer to general principles of DTMF and tone decoding, keep in mind that the primary objective is to discuss a particular implementation of a tone detector.

Theory of Operation

Describes the basic theory of operation of the tone detector, describing total system scope and functionality, and giving a brief introductory description of each functional block. For this purpose the tone detector is considered as a set of software functions with supporting hardware. The high suitability of the TMS320C17 DSP for tone detection is also discussed.

Implementation

Deals in detail with the implementation of both the software within the TMS320C17, and its supporting hardware. Each is split into its main functional blocks and then further subdivided into individual tasks. The description of software implementation is accompanied by a series of flow charts, allowing the reader to follow the description from the top functional level right down to the detail of individual tone detector features. This section also covers in detail how the tone detector program controls, and benefits from, some of the resources provided by the TMS320C17.

Host Interface

Describes the host interface of the tone detector. This has been designed for easy connectability to a variety of host CPUs, and is essentially a single physical 8-bit read/write register. The host interface software is implemented by an interrupt routine in the TMS320C17, allowing host access at any time as required.

Applications and Customization

Briefly outlines some possible applications for the tone detector including traditional telephony applications along with some innovative approaches. These include a method for secure off-site remote control of equipment via telephone lines, a tester for telephone equipment, etc. For many applications it may be necessary to customize the program to some extent. A number of examples of this are discussed.

Conclusion

Within the appendices are a full listing of the source code for the tone detector in COFF (common object file format) source format, and a demonstration program for IBM or compatible PCs. This program is written in Turbo Paşcal and is for use with the design example included in this report.

History of DTMF

There are two standard dialing conventions used in telephone systems throughout the world. The most common, and by far the oldest is known as pulse or loop-disconnect dialing. DTMF is a relatively new all-electronic method which is rapidly replacing the older electro-mechanical system. Figure 1 represents a highly simplified pulse dialing telephone terminal. There are other circuits required to make a practical telephone, but this diagram serves to illustrate several key points.

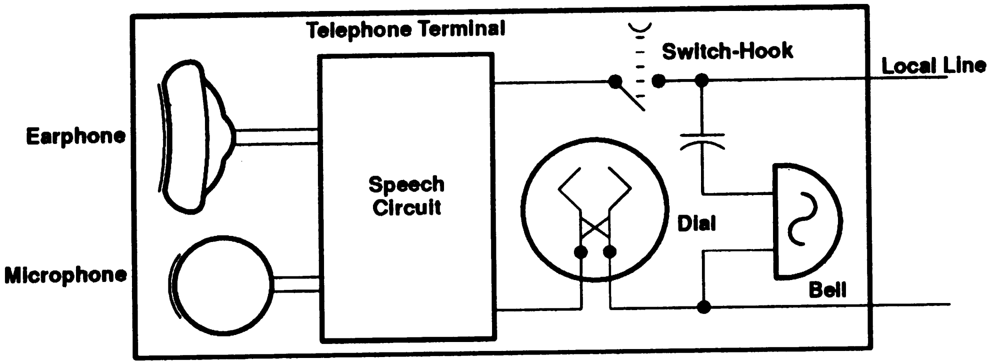


Figure 1. Pulse Dialing Telephone

When the receiver of a pulse dialing telephone is lifted, the hook-switch closes and a DC loop current of a few milliamperes flows from the central office or local exchange. The dial is arranged so that the switch within it opens and closes as it returns to its rest position. When the switch opens it causes the loop current to be interrupted, hence the alternative name of loop-disconnect dialing. The dial is arranged so that one disconnect period or pulse is created for the digit 1, two for the digit 2, up to ten pulses for the digit 0.

Dial pulses originally operated electromechanical switching systems, and still do in many countries. These systems have an upper limit of about ten operations per second and pulse dialing systems therefore produce pulses of a 100 millisecond (ms) duration. Nominal operation in the U.S. gives a break period of 61 ms and a make period of 39 ms. This is different from other countries which use a 2:1 ratio (67 ms break, 33 ms make). An inter-digit pause is indicated by an absence of pulses of nominally 700 ms for U.S. systems, or as short as 200 ms in other countries.

The time required to send the dial pulses needed for one digit can be up to 1.7 seconds (ten pulses for the digit 0 and a 700 ms inter-digit pause) which can make the dialing of a long international number very time consuming. For example, the international number (from the U.S.) for Texas Instruments in Bedford, England is:

0 1 1 4 4 2 3 4 2 7 0 1 1 1

This would take 15.1 seconds to dial with a U.S. pulse dialing system. It is not difficult to see why the method is now regarded as out-dated.

In order to reduce costs, increase reliability, and improve service, the electromechanical switching systems used at central offices or local exchanges are being replaced with fully electronic systems. In most advanced countries this upgrading process is virtually complete. With the new equipment it is no longer necessary to have a slow dialing mechanism to accommodate the response time of the old switching mechanisms. A new dialing scheme thus becomes possible using purely electronic means. The DTMF system has been adopted as the universal standard through the CCITT (Comite Consultatif International de Telephonie et de Telegraphie) which is a committee of the International Telecommunication Union (ITU), now part of the United Nations.

The Use and Characteristics of DTMF

The full name for DTMF is Dual-Tone Multi-Frequency which describes its operating characteristics very well. Consider that a telephone is equipped with a keypad as shown in Figure 2, instead of a dial. The A,B,C and D keys are usually not present, but are part of the full CCITT specification and can be decoded by the programmed TMS320C17 used here.

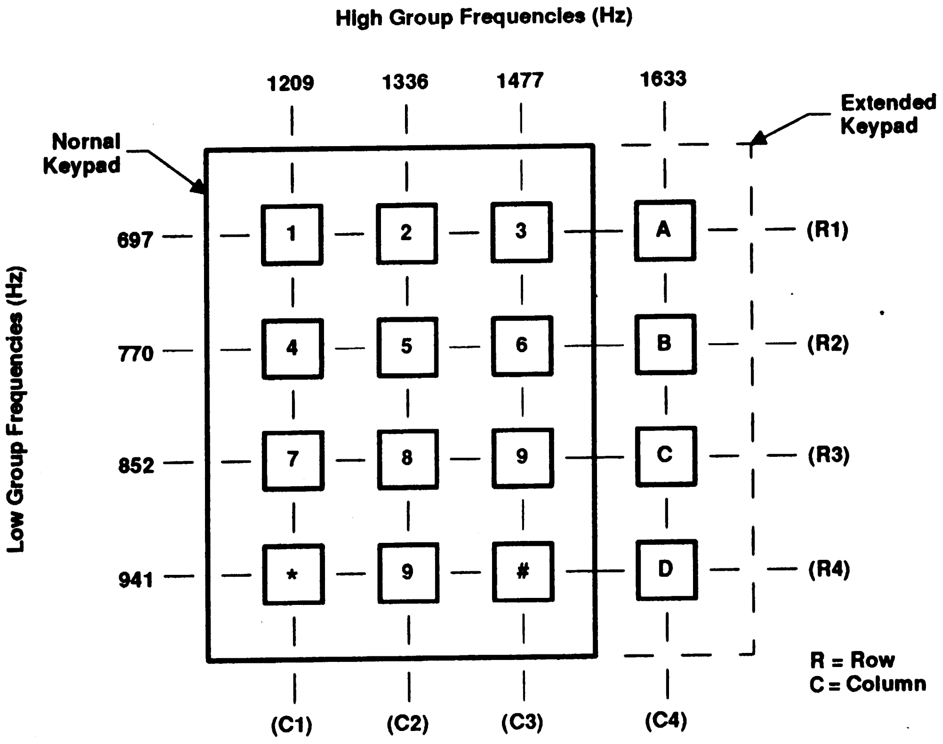


Figure 2. DTMF Keypad

Pressing any key causes an electronic circuit to generate a tone which is a summation of the two individual frequencies related to the row and column of that key.

The frequencies used in DTMF dialing have been carefully selected so that any DTMF decoding circuit will not confuse them with other tones that may occur on the line. As the tone generation does not involve a disconnect of the telephone circuit, DTMF tones may be sent down the line during a call just by pressing any key on the keypad. When this method is used as a form of low speed data transmission, it is important that speech is not accidentally interpreted as a DTMF tone. In order to reduce the risk of this happening, tones must be present continuously for a minimum period of about 50 ms, with an interdigit pause of similar length.

With a minimum dialing time of 100 ms per digit, irrespective of its value, our previous example number would take 1.4 seconds to dial. This represents a saving of 13.7 seconds or 91% of the time taken by a pulse dialer. Additional advantages of DTMF dialing include the use of solid-state electronic circuits and compatibility with electronically controlled exchanges.

Theory of Operation

This section briefly describes the operation of the tone detection system presented in this report. A functional block diagram for the complete system is shown in Figure 3.

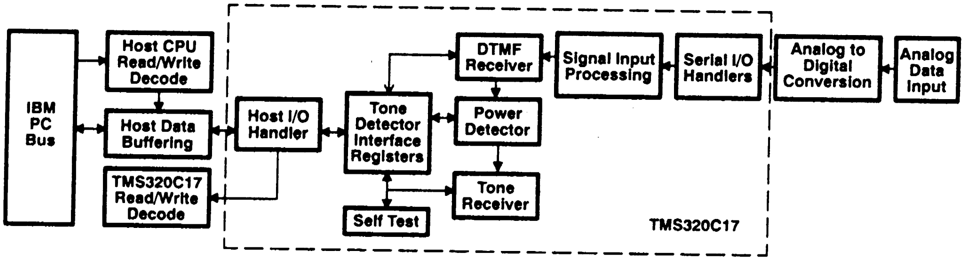


Figure 3. Tone Detector Functional Block Diagram

As is clear from examination of Figure 3, the tone detector may be viewed as comprising a set of software routines within the TMS320C17, plus associated external hardware to provide interfaces between the TMS320C17 and both the incoming analog signal and a host CPU.

The following paragraphs briefly describe the major software and hardware features of the tone detection system, and some of the features of the TMS320C17 which are of special benefit to this application.

Software

The tone detection system described in this report comprises six groups of functions within the TMS320C17. These provide a powerful tone detection capability for either DTMF decoding, general tone identification or a combination of both. These six functional groups are as follows:

1. Input signal processing
2. DTMF receiver
3. Power (envelope) detector
4. Tone receiver - comprising five sub-sections
5. I/O routines (Interrupt Handler)
6. Self test

Figure 4 shows how the first four of these functions interrelate during normal operation of the the tone detector. Each block within Figure 4 is explained in detail in the Implementation section and each also has a detailed flowchart associated with it. The number of the figure for the associated detailed flow chart is shown inside each block in Figure 4.

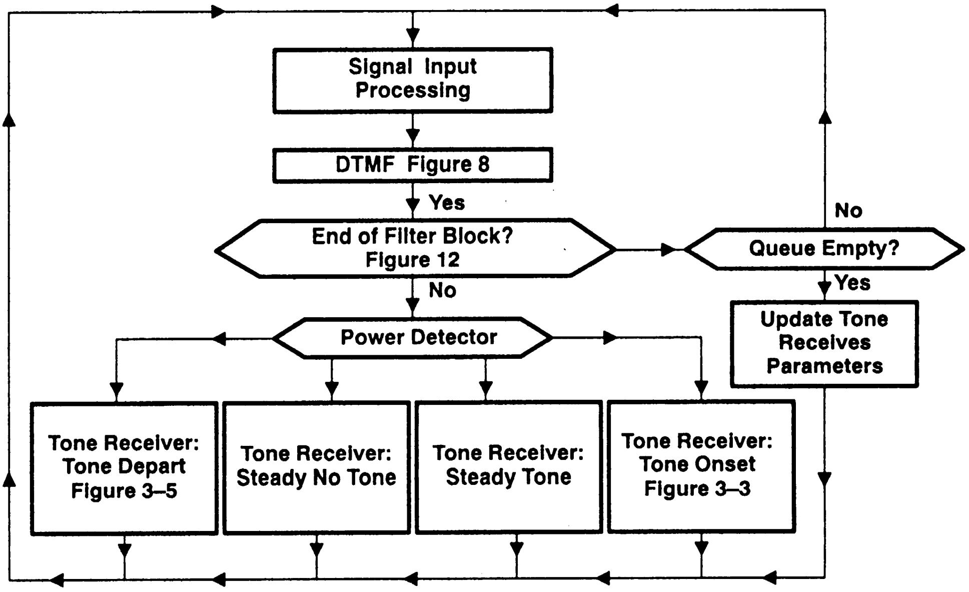

Signal Input Processing
DTMF Figure 8
Yes
End of Filter Block? Figure 12
No
Queue Empty?
No
Yes
Update Tone Receives Parameters
Power Detector
Tone Receiver: Tone Depart Figure 3–5
Tone Receiver: Steady No Tone
Tone Receiver: Steady Tone
Tone Receiver: Tone Onset Figure 3–3

Figure 4. Tone Detector Flow Chart - Top Level

Program execution remains within the flow shown in Figure 4 unless interrupted by either an I/O request or a self-test command, which are independent functions. Self-test is merely a special case of a host CPU I/O request. Both serial I/O and host CPU I/O cause an interrupt to the TMS320C17 and therefore function outside the normal program flow. Self-test additionally destroys all temporary data storage, leaving the tone detector in the same state as after a hardware reset (see the Register Read Functions section). The following sections briefly describe the relationship betwen the above six functional groups. A more detailed description of the operation of each is contained in the Implementation section.

Input Signal Processing

This ensures that the incoming data samples are within the optimum working range of the tone detector. Software limiting of the incoming signal is applied if it exceeds the maximum signal input level (see the Signal Input Processing section). Program control passes to the DTMF receiver if it is enabled, otherwise control passes to the power detector.

DTMF Receiver

Using the signaling plan outlined in CEPT (Conference Europeenne des Administrations des Postes et des Telecommunications) recommendations T/CS 46-02, the DTMF receiver validates and decodes DTMF tone pairs against a template of acceptable frequency deviation. The DTMF receiver may be enabled or disabled under software control by the host CPU. Once the operation of the DTMF receiver is complete, program flow passes to the power detector.

Power (Envelope) Detector

The power detector performs a simple smoothing operation on the incoming signal and, using thresholds programmed by the user, directs program flow among one of the four possible tone receiver flow paths shown in Figure 4:

1. Tone onset
2. Tone depart
3. Steady no tone
4. Steady tone

Separate threshold levels may be programmed for detection of the onset and departure of the input signal.

Tone Receiver Power Level Determination

The tone receiver determines the overall power level of the incoming signal and the individual power level at up to six selectable frequencies. In addition, it validates the signal onset or signal departure indication from the envelope detector to change the tone arrival or tone departure status bits (see Status section). The tone receiver operates independently of the DTMF receiver and provides programmable center frequency, bandwidth, resolution and thresholds for the recognition of general tones in the band 300 Hz to 3400 Hz (e.g., call progress tones).

When the DTMF receiver is disabled the tone receiver monitors six programmable frequencies in the range 300-3400 Hz and reports the power levels received at each of those frequencies. When the DTMF receiver is enabled the tone receiver monitors only three frequencies. The power level of the three unused frequencies is registered as zero. The tone receiver also has an additional power measurement which reports the received power across the telephony band of 300-3400 Hz allowing the system to detect the presence of frequencies outside those programmed individually.

When the tone receiver is enabled, filtering begins upon the recognition of a tone by the envelope detector. The host may be interrupted at the end of the first block of filtering as a result of the tone arrival bit in the status register being set. At this time level information for the new tone is available at each of the search frequencies. The host may also be interrupted by tone departure. The tone receiver is also able to detect any change in signal content and may optionally generate an interrupt as a result. Host interrupt is described in detail in Host Interrupt section.

The flow of program execution around the tone receiver is dependent upon the results of tests at a number of points. The most important of these is at the output of the power detector. As mentioned above there are four possible conditions the power detector can indicate:

1. Tone onset
2. Tone depart
3. Steady no tone
4. Steady tone

The operations performed within these blocks are described in detail in Software Implementation section.

The second most important decision point in the tone receiver program flow is represented by the end of filter block test. When the tone receiver is enabled, incoming samples are filtered in blocks. The block size is dependent upon the value written to the filter length register (see Filter Length section). If a filtering block has been completed, housekeeping functions must be performed.

I/O Handler (Serial and Parallel)

Any external I/O access will cause an interrupt to the TMS320C17. External I/O can come from one of three possible sources:

- A new data sample being input from the serial port
- A host CPU write access
- A host CPU read access

The source of the interrupt is checked by the program and control passed to the appropriate portion of the interrupt handler code. A comprehensive discussion on the use of interrupts within the tone detector is given in Hardware Implementation section, including a detailed examination of some parts of the interrupt handler code.

Self-Test

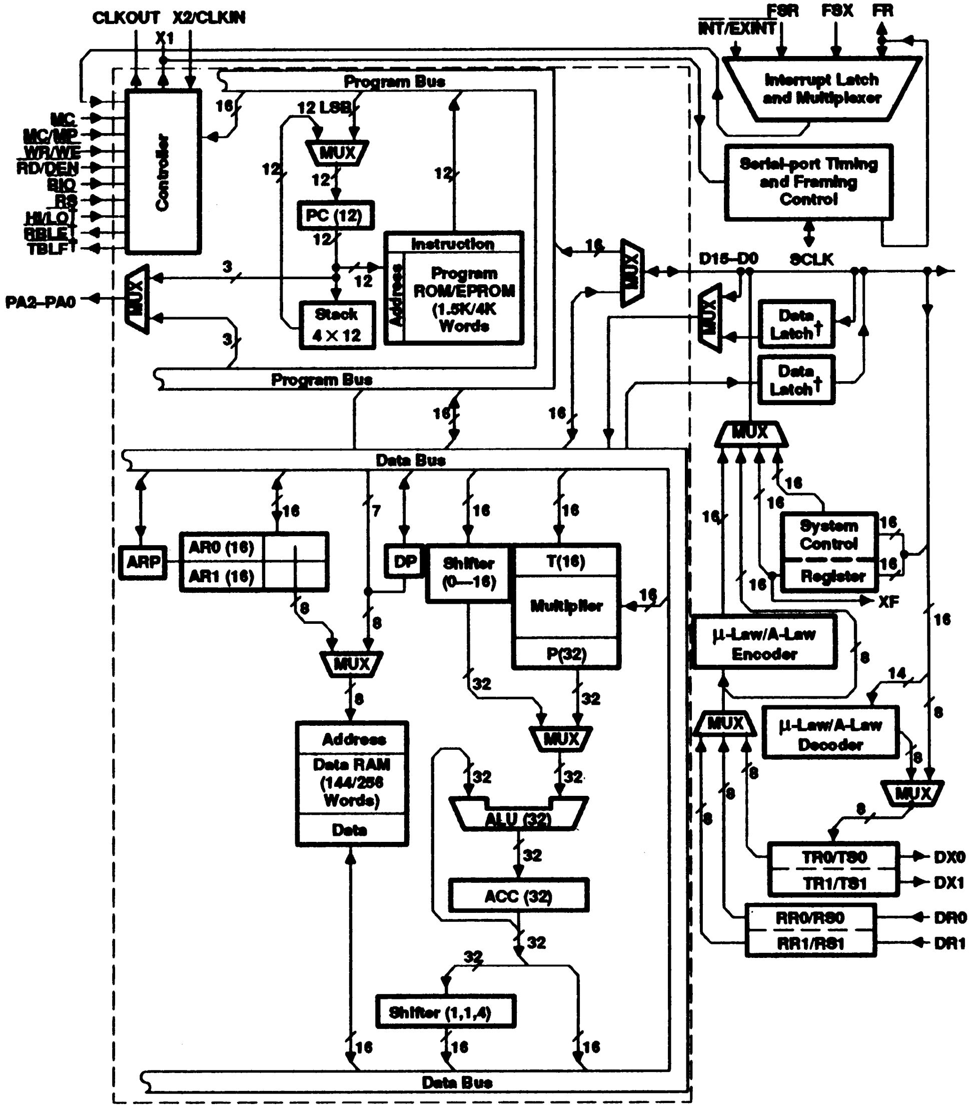
One special case of a host CPU write access is a self-test request. The TMS320C17 responds to this by immediately performing a ROM checksum test, a RAM data test and a codec interrupt check. After these have been performed the host CPU may release the TMS320C17 from self-test mode. The TMS320C17 is then left in a state similar to that after a hardware reset (see Register Read Functions section).

TMS320C17 Features

The TMS320 family utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture, program and data memory lie in two separate spaces, permitting a full overlap of instruction fetch and execution. The TMS320 family's modification allows transfers between program and data spaces. This permits coefficients stored in program ROM to be read into RAM, eliminating the need for a separate coefficient ROM. It also makes available immediate operand instructions and subroutine calls to computed addresses.

The TMS320C17 provides all the basic features of the industry-standard TMS320C10. Two serial ports, expanded data memory to 256 words, expanded program memory to 4K words on-chip, and a coprocessor mode are added to provide a powerful processor for a variety of communications-oriented applications. The TMS320C17 is a microcomputer device only, with no external program memory facility. The TMS320E17, a 4K-word EPROM version of the TMS320C17 is available for prototyping or low volume production.

The Tone Detection application takes advantage of the full set of processor resources shown in Figure 5. A few examples from the code, and a description of each, are given in Utilization of TMS320C17 Resources section to illustrate this.



Legend:

ACC	=	Accumulator	PC	=	Program Counter
ARP	=	Auxiliary Register Pointer	P	=	P Register
AR0	=	Auxiliary Register 0	T	=	T Register
AR1	=	Auxiliary Register 1	TR	=	Transmit Register
DP	=	Data Page Pointer	RR	=	Receive Register

†TMS320C17/E17 only

Figure 5. TMS320C17/E17 Block Diagram

The Tone Detector program uses less than 50% of the available 4K-words of program memory and less than 70% of the available 256 words of data memory within the TMS320C17. Of the 174 words of data memory used, 75 are in page 0, and 99 in page 1. A detailed list of program and data memory utilization is shown in Table 1.

Table 1. Program and Data Memory Utilization

Routine	Code Listing Page	Description	Program Memory Locations	Data Memory Locations
	489	Reset and interrupt vectors	4	
	490	DTMF Constants and filter coefficients	28	
	490	Tone detector constants	62	
	491	Tone detector filter coefficients	129	
MAIN	492	Read sample from input queue and update current time, scale the input sample and call DTMF if it is switched on.	55	14
ENVDET	494	Detect changes in signal envelope relative to user-programmed upper and lower thresholds	41	4
TONSET	495	Handle occurrence of tone onset	11	2
TDEPT	495	Handle tone departure	41	1
FILTER	496	Routine for filtering and accumulating the input samples	172	52
LEVCAL	499	Calculates the levels at the end of each block of filtering	109	1
CHNGS	501	Check for level changes during a toneburst	31	0
LVLS	501	Write levels into registers	13	3
COMPLT	502	Complete operations ready for next filtering operation	39	1
RSTFIL	502	Clear down filter accumulators and reset pointers ready for another filter operation	61	
SQRT	504	Generates the square root of an integer	32	
DTMF	504	Detect DTMF digits	508	83
INTHDL	510	Interrupt handler	194	8
CRESET	514	Cold reset handler	14	1
WRESET	514	Warm reset handler	33	3
ATTEN	515	Write out status to draw attention to change in one or more of the status bits	4	
XFUPD	515	Update the XF flag	17	
SLFTST	516	Self test of processor	111	4
Total			1709	177

Hardware

In order for the TMS320C17 to receive its input signal and communicate with a host CPU it requires a small amount of support circuitry. This comprises just three devices, as shown in Figure 6. This example is specifically for interfacing the tone detection system to an IBM XT or AT compatible PC bus. A detailed description of this circuit is given in Hardware Implementation section.

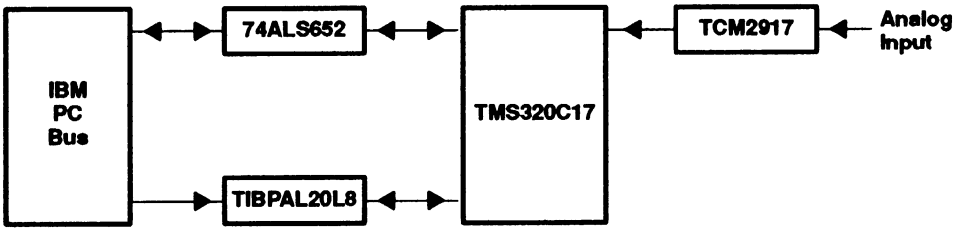


Figure 6. PC Tone Detector Circuit Diagram—Block Level

Analog to Digital Conversion

The analog signal is converted to a serial pulse code modulated (PCM) serial data stream by an industry standard combined codec and line filter (COMBO), the TCM2917. This interfaces directly to the TMS320C17 with no support circuitry.

Host Interface

A programmable logic array (PAL) provides read and write decoding for both the host CPU and the TMS320C17, including full address decoding of the host CPU bus. A 74ALS652 provides a two way latched data buffer between the host CPU and the TMS320C17. The TMS320C17 has a special coprocessor mode which can also perform the latched data buffer function in a wide variety of applications. The coprocessor mode is described in greater detail in Use of Coprocessor Port for Parallel I/O section.

Implementation

This section describes in greater detail how the tone detector functions described in the Theory of Operation section are implemented. It is intended for non-mathematical readers, and equations have only been included where they can aid understanding for readers familiar with general DSP techniques. It is not necessary to understand the derivation or purpose of these equations in order to gain a basic understanding of system operation.

Software Implementation

As described in Software section the software within the tone detector may be conveniently split into the following six groups:

1. Signal input processing
2. DTMF receiver
3. Power detector
4. Tone receiver comprising five sub-sections
5. I/O routines (Interrupt Handler)
6. Self test

A detailed description of the performance and implementation of these functions follows.

In all of the detailed explanations in this section of the report, references are provided to a page of the program listing included as Appendix A.

Signal Input Processing

This block contains only two straightforward tasks:

1. Read queue, increment time (program listing page 492)—Codec samples sent to the TMS320C17 are received via its serial port and then queued. The maximum queue length is eight samples. Under normal circumstances the queue will not contain more than one sample. However, at the end of each block of filtering or DTMF detection, there is a series of computations which must be completed before the handling of the next codec sample. Operation of both the DTMF code and the tone filtering code are suspended during this period and new codec samples accumulate on the queue. At all times, information arriving at the TMS320C17 via its serial port is handled with first priority, so that no samples or requests are missed.

2. Scale and limit (program listing page 492)—In this report the TMS320C17 is programmed to accept A-law input samples. The TMS320C17 can also be programmed to accept the u-law samples in North American applications. The output from the on-board compander is scaled to a number range which affords the maximum precision for the range of signal magnitudes allowed. The tone receiver is specified to provide linear detection of tones in three ranges. The dynamic range of the tone receiver is between 35 and 40 decibels (dB). Provision of three software selectable scale factors allows this dynamic range to be shifted so that the top of the range is at either +2, −10 or −22 dBmO. Where dBmO is defined as the zero reference point of the channel. The overall detection range is thus +2 to −60 dB approximately (see Figure 7).

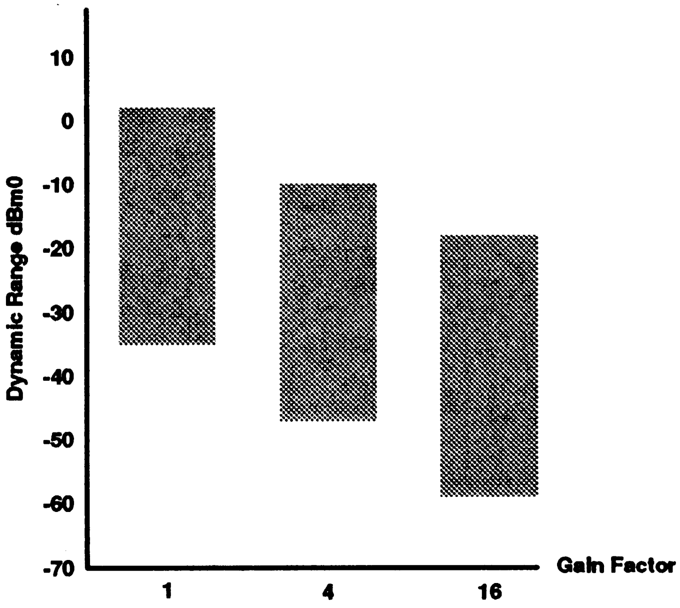


Figure 7. Tone Detector Active Dynamic Range vs Gain Factor

The output from this block is the next sample to be dealt with by the DTMF code and the power detector.

DTMF Receiver

A brief specification is given in Table 2. For full details, refer to CEPT recommendation T/CS 46-02. The operation of the TMS320C17 algorithm to this specification has been verified by use of the standard Mitel DTMF test tape.

Table 2. DTMF Decoder Specification

Measurement	Breakdown	Value
Signal frequencies	Low Group	697, 770, 852, 941 Hz
	High Group	1209, 1336, 1477, 1633 Hz
Frequency deviation for correct operation		≤1.9%
Power levels per frequency	Operation	(−6 dBm0 − G dB) to (−36 dBm0 − G dB)*
	Non-operation	−45 dBm0 − G dB*
Power level difference between frequencies for operation		0 dB to 10 dB
Tone duration	Recognition	≥40 mS
	Non-Recognition	≤20 mS
Silence duration	Recognition	≥40 mS
	Non-Recognition	≤20 mS
Signal to noise ratio required for correct operation		12 dB
Talk-off performance		15 hits in 30 minutes of condensed speech

*See Mode subsection in Host Interface section for an explanation of the gain control factor GdB.

The DTMF receiver may be used to receive and recognize tones from a remote handset, e.g. in a PABX, or from a telephone set at a remote point on the public telephone network. The distortion of tones over the public network is often severe; for example, the attenuation of the signal from the remote transmitter could vary from 0 dB to 30 dB or more. The specification shown in Table 2 provides correct operation across the normal range of signals received over the public network.

The range of received signal levels at which the DTMF receiver will correctly decode signals can be varied by altering the gain of the tone detector module under software control (see Mode section).

Validation of a DTMF digit while the DTMF receiver is enabled (see Mode section) causes a DTMF interrupt to be generated and suppresses the generation of any short tone interrupt which might otherwise have been generated by the tone receiver code. The arrival time of the tone is stored for the host to read if required.

The following description of the operation of the DTMF block relates directly to the detailed flow chart shown in Figure 8.

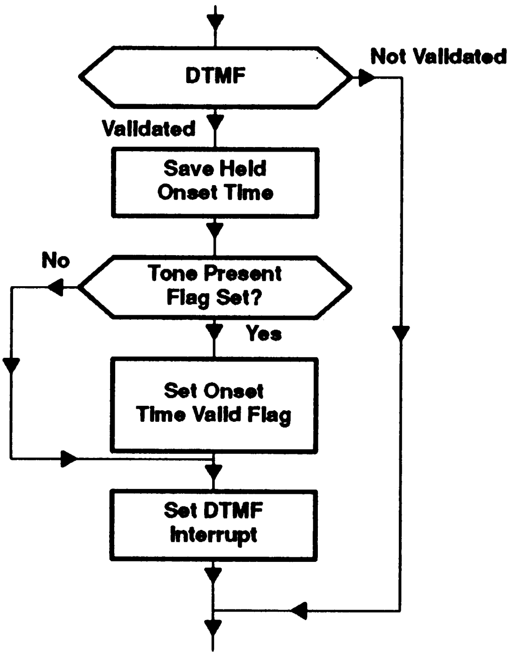


Figure 8. DTMF Receiver Flow Chart

DTMF (program listing page 508)—This revolves around a set of eighth order narrow bandpass filters at each of the individual tone frequencies which may be combined to produce a DTMF digit.

The simple eighth order filtering process is executed on the incoming sample automatically when the DTMF receiver is enabled. If a valid DTMF digit is found, its value is stored in the DTMF digit register and execution passes along the 'validated' path. If the DTMF receiver is not enabled, program execution passes onto the tone receiver.

Save Held Onset Time—The onset time of all detected signals is saved in a holding register. This is transferred to the tone arrival register only if the tone receiver is not already indicating the presence of a tone, in which case the tone arrival register will already have been loaded.

Set Onset Time Valid Flag, Set DTMF Interrupt—The DTMF tone onset time is saved in a register for the host to read. The host is informed by interrupt (if implemented) that a tone onset has occurred and that timer registers containing information about the tone are available to be read.

Power (Envelope) Detector (see program listing page 494)

As described above the power detector performs an envelope detection operation on the incoming signal, and directs flow to one of four tone receiver paths.

The smoothing filter applied to the incoming signal has the form:

$$\text{ENVEL} = \frac{((2^{15} \times \text{ENVEL}) + \text{ABS}(32 \times \text{EDF} \times \text{SAMPLE}) - (32 \times \text{EDF} \times \text{ENVEL}))}{2^{15}}$$

Where EDF is the user programmed envelope decay factor (see Envelope Decay Factor section). This is equivalent to:

$\text{ENVEL} = ((1-k) \times \text{ENVEL}) + (k \times \text{ABS(SAMPLE)})$ where EDF is k X 2^{10} where EDF is k $\times$ 2^{10} k positive.

The envelope decay factor may be programmed to provide a range of time constants for the envelope detector. There is generally a trade-off between the rejection of a glitch if a long time constant is used and increased accuracy of time-stamping with a short time constant.

When the power detector identifies the departure of the input signal, a status register bit (see Status section) may be set, and the time of departure written into a register. This depends upon the signal having been recognized as a DTMF digit or a valid tone within the tone receiver search bands.

Due to the method of implementation of the envelope detector, it should be kept in mind that there are two areas of operation when using the tone receiver: the arrival and departure time skew and the sampling frequency. These are explained in detail in Appendix C.

Tone Receiver Band Pass Filter Generation

The tone receiver generates a band pass filter for each of the chosen frequencies and uses these filters to select the desired frequencies from the incoming signal. The steepness of cut-off of each bandpass filter is defined by the length of time over which the received signal is filtered. This is programmed via a register and applies to all the filters in operation. The passband width of each filter is specified via a separate register, and the maximum value for passband width for any single filter is 492 Hz. Each of the filters in use may be selected to adopt either the passband width specified in the register (wide filter) or a passband width of zero (narrow filter).

As described in the Tone Receiver Power Level Determination section, the power detector directs the flow of the tone receiver along one of four paths:

1. Tone onset
2. Tone departure
3. Steady tone
4. Steady no tone

A detailed description of the operation of each of these follows.

Tone Onset

Figure 9 shows the flow chart associated with a tone onset indication from the power detector.

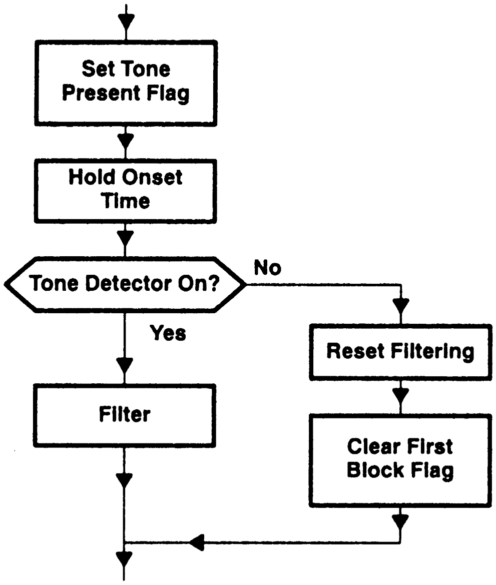


Figure 9. Power Detector Flow Chart—Tone Onset

Set Tone Present Flag—This flag is used to indicate the presence or absence of a tone on the line.

Hold Onset Time—The onset time of all detected signals is saved in a holding register.

Filter (program listing page 496)—This routine is the heart of the tone receiver algorithm. The FIR filters are of the lowpass type and there is one for each of the six search frequencies. A range of filter lengths may be specified, from 61 to 1025 samples, allowing filters of extremely steep cut-off to be implemented. With the maximum filter length of 1025 samples, the shortest quantifiable tone is one of at least 128 ms duration. The input signal is demodulated using a sine and cosine wave at each of the six search frequencies. The result of the demodulation is that any signal present at one of the search frequencies is transposed into the passband of the lowpass filter. Figure 10 shows the filter structure.

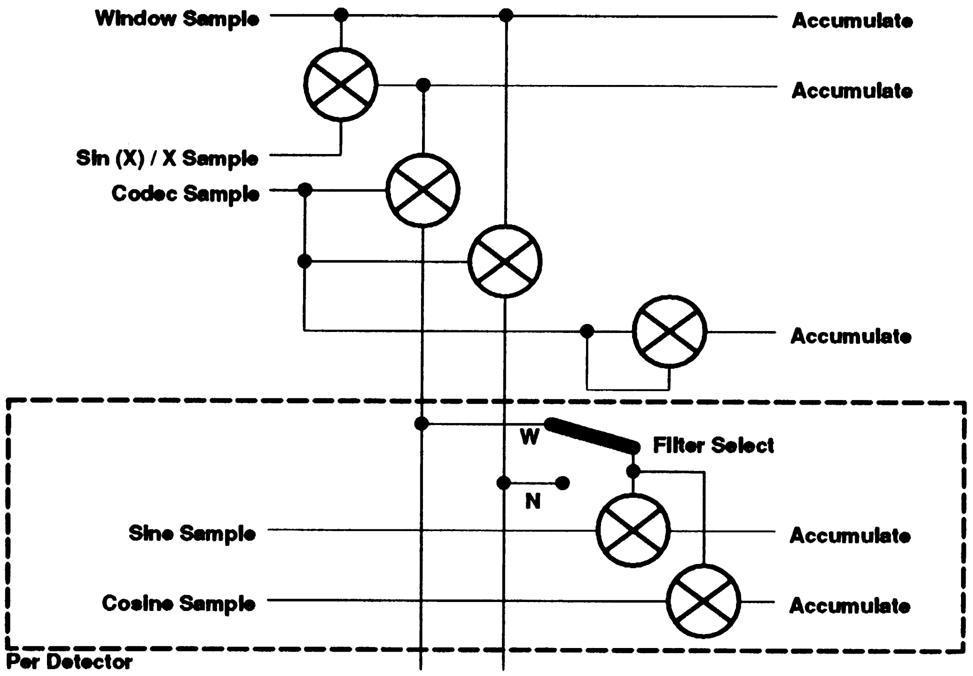


Figure 10. FIR Filter Structure

The coefficients of the filter are samples taken from a window function stored in ROM. The function is a Kaiser window, chosen to give the narrowest lowpass response with the given stopband rejection. Where a wide filter response is specified, each filter coefficient is multiplied by a sample of a sin(×)/× function to provide a second wide filter coefficient. This has the effect of widening the filter passband in a definable and convenient manner. The input sample is multiplied by the normal (narrow) and wide filter coefficients to produce both a narrow and wide intermediate sample. Each of the six filters is specified to be either narrow or wide according to the value in the filter select register. Depending on this value, the appropriate intermediate sample is multiplied by a sine sample and cosine sample at the required search frequency. The sine and cosine samples are generated as required by a special routine. The twelve products are separately accumulated to 32-bit accuracy.

In addition to this, accumulations are kept of the wide and narrow filter coefficients so that the filter accumulations can later be normalized. An accumulation is also kept of the square of the input sample, so that the total signal level in the telephony band can be calculated.

Reset Filtering—Clears down all the accumulators and registers used by the filters.

Clear First Block Flag—Clears a flag set to indicate that the first block of data was being filtered.

Tone Depart

Figure 11 shows the flow chart associated with a tone departure indication from the power detector.

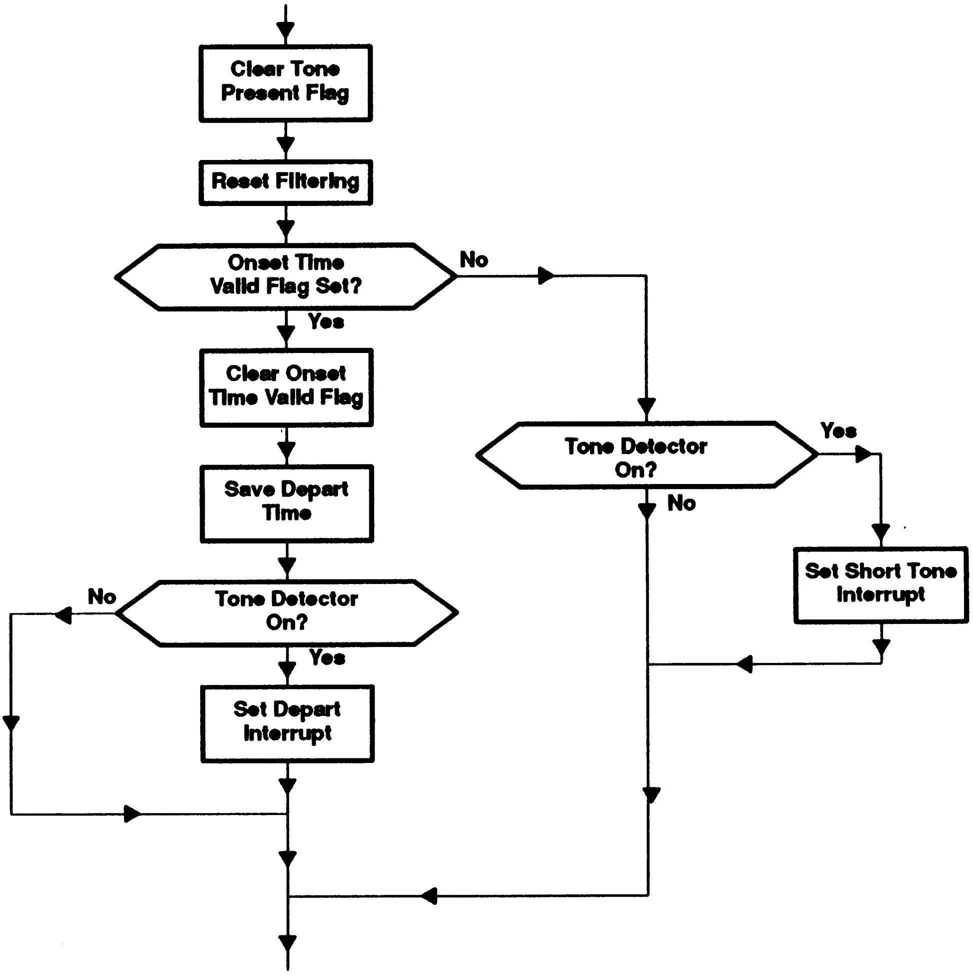


Figure 11. Power Detector Flow Chart—Tone Departure

Clear Tone Present Flag—This flag is used to indicate the presence or absence of a tone on the line.

Reset Filtering—Clears down all the accumulators and registers used by the filters.

Onset Time Valid Flag Set?—The program tests to see if a flag has been set at this point to indicate that the stored onset time is valid. This will be the case only if a complete block of filtering has been performed on the tone, or the tone has been recognized as a DTMF digit. If the flag is not set the program further checks to see if the tone detector is enabled. If not this section terminates. Timer registers are not updated and contain onset and departure times for the previous valid tone or digit. However, the current time register is available for the host to read if it wishes to timestamp the short tone. If the tone detector is on, the short tone bit in the status register is set which can optionally generate an interrupt (see Status section).

Clear Onset Time Valid Flag—Clears the above flag.

Save Depart Time—Provided that a valid tone or digit has been recognized, the current time is saved directly into the tone departure register.

Set Depart Interrupt—If the tone detector is enabled, the tone depart bit in the status register is set. This may optionally generate an interrupt.

Steady No Tone

In this case, the only operation performed is Reset Filtering which clears down all the accumulators and registers used by the filters.

Steady Tone

This condition causes execution from just above the "Tone Detector On" decision point in the tone onset flow chart (Figure 9).

End of Filter Block?

When the tone receiver is enabled, incoming samples are filtered in blocks. The number of samples in a block is set by the filter length selected, and may be between 61 and 1025 samples. After each complete block of filtering, much housekeeping must be done. Figure 12 shows the flow chart for this process.

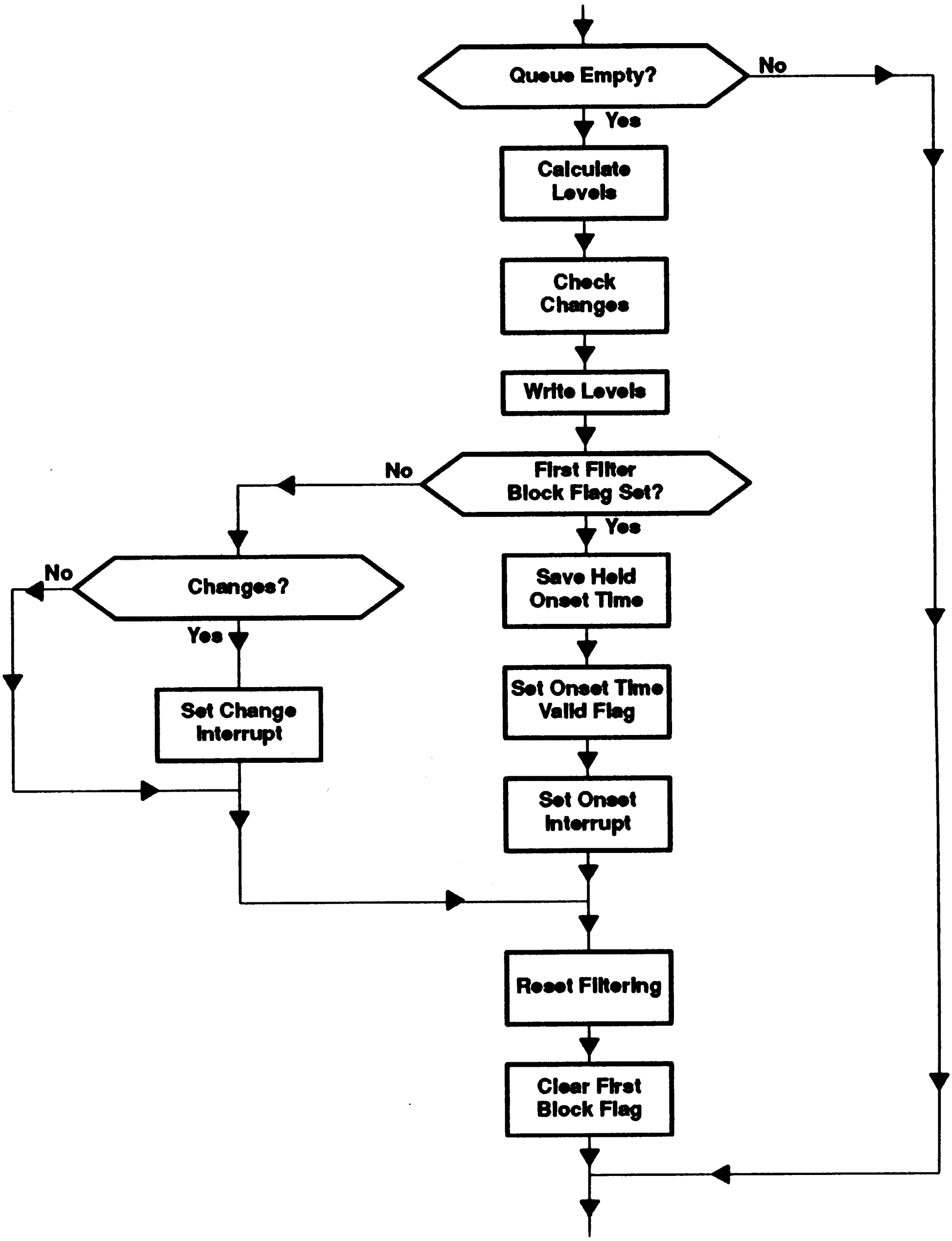


Figure 12. Tone Receiver Flow Chart—End of Filter Block

Calculate Levels—For each filter, the root of the sum of the squares of the corresponding sine and cosine accumulations is calculated and normalized using the appropriate filter-coefficient accumulation. The result represents the signal level falling within the passband of the filter. The square root of the signal-squared accumulator represents the total signal level present within the telephony band. Provided that the filters have been correctly placed, the root of the sum of the squares of the filter outputs should equal the total signal level. This allows a check to be made for tones present but not registered by the filters in use.

Check Changes, Write Levels—The output level of each of the six filters is checked to see whether any of them has crossed the change threshold programmed by the user. The signal levels in the six bands are then written to registers for the host to read. The second three filters will be zero if DTMF is switched on.

Save Held Onset Time, Set Onset Time Valid Flag, Set Onset Interrupt—If the block of filtering that has just been completed was the first one performed on the current tone there are a few other tasks to perform. The tone onset time is saved in a register for the host to read and then the host is informed by interrupt that a tone onset has occurred and that timer registers containing information about the tone are available to be read.

Changes?, Set Change Interrupt—If the completed filter block was not the first block after tone arrival, it is necessary to check for any changes to the tone. If any signal levels have crossed the change threshold in a filtering block other than the first block, then a change interrupt is asserted. Registers containing information about the tone may contain misleading information due to the likelihood of the change having occurred in the middle of a filtering operation.

Reset Filtering—Clears down all the accumulators and registers used by the filters.

Clear First Block Flag—Clears a flag set to indicate that the first block of data was being filtered.

I/O Routines (Interrupt Handler)

Both host and signal (serial) I/O are dealt with by the interrupt handler. Host read or write accesses cause an external hardware interrupt to the TMS320C17. The availability of a new codec sample within the serial port receive register causes an internal hardware interrupt. A flow chart of the interrupt handler is shown in Figure 13. A detailed description of some parts of the code within the interrupt handler are contained in Interrupts section.

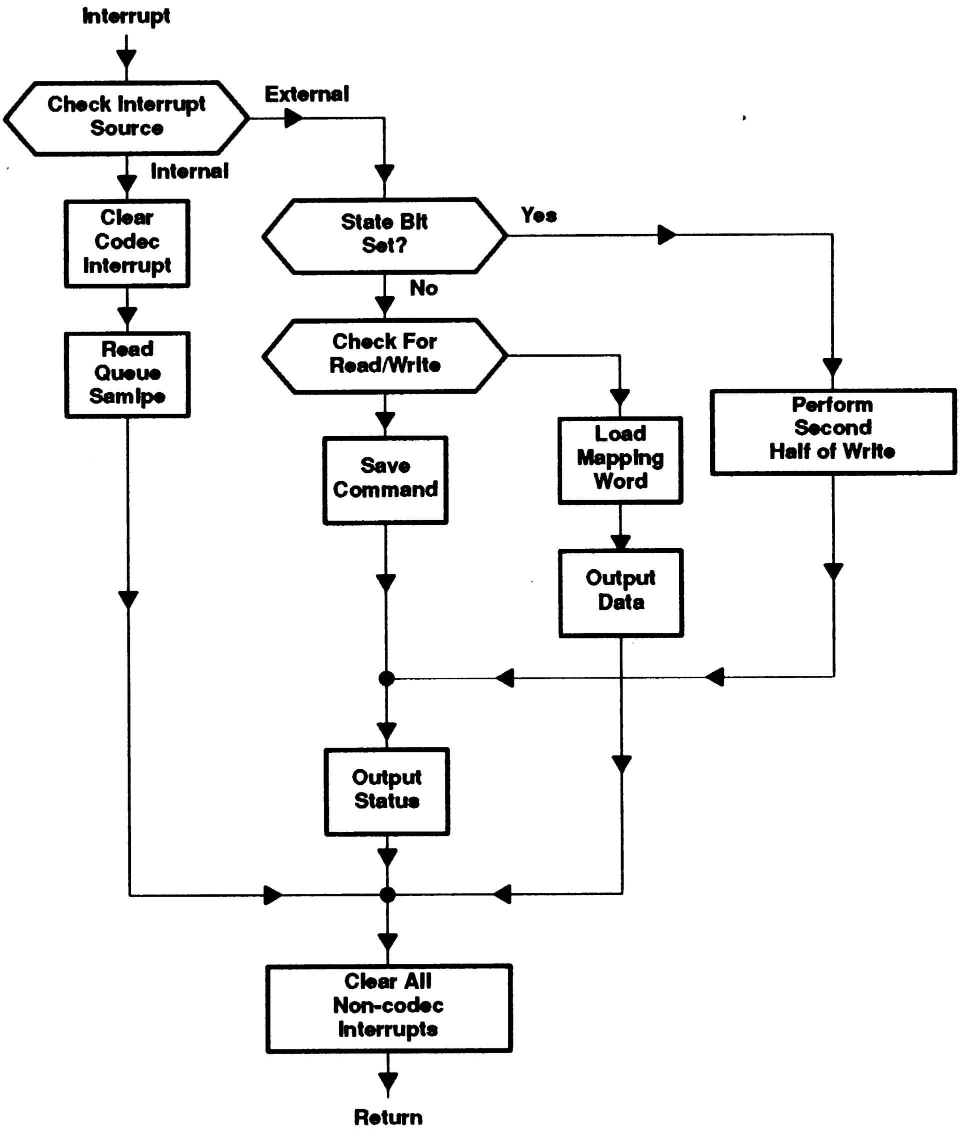


Figure 13. I/O (Interrupt Handler) Flow Chart

Self Test

The tone detector system can be instructed to carry out a self-test operation at any time by writing to a bit in the mode register. The flow chart for the self test routine is shown in Figure 14. The duration of the test is 6 ms. No access should be made to the tone detector until the end of this period when the result of the self test is available in the mode register.

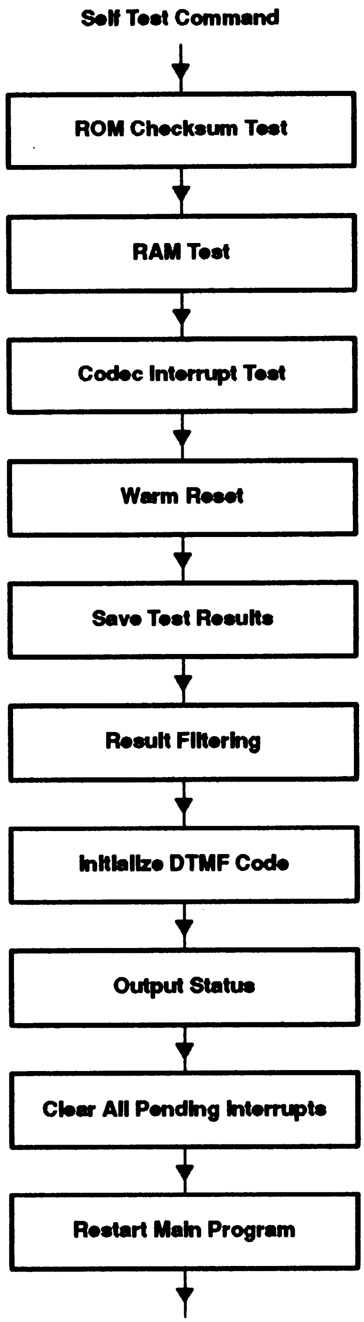


Figure 14. Tone Receiver Flow Chart - Self Test

Once the self-test is complete the tone detector enters a state where normal functions are inoperative, but the host data path may be tested. In this mode a write to any register other than mode or control will access a holding register inside the tone detector, rather than the register specified. This holding register may then be read by accessing any register other than mode or status, thus checking the integrity of the host data path.

Self-test is terminated by a further write to the mode register. When this has been done, the tone detector is left in the default state as though it had received a hardware reset.

Program Overview

An integrated flowchart for the tone detector program is shown in Figure 15. I/O routines and self test are not included as they do not form part of the normal tone detector program flow.

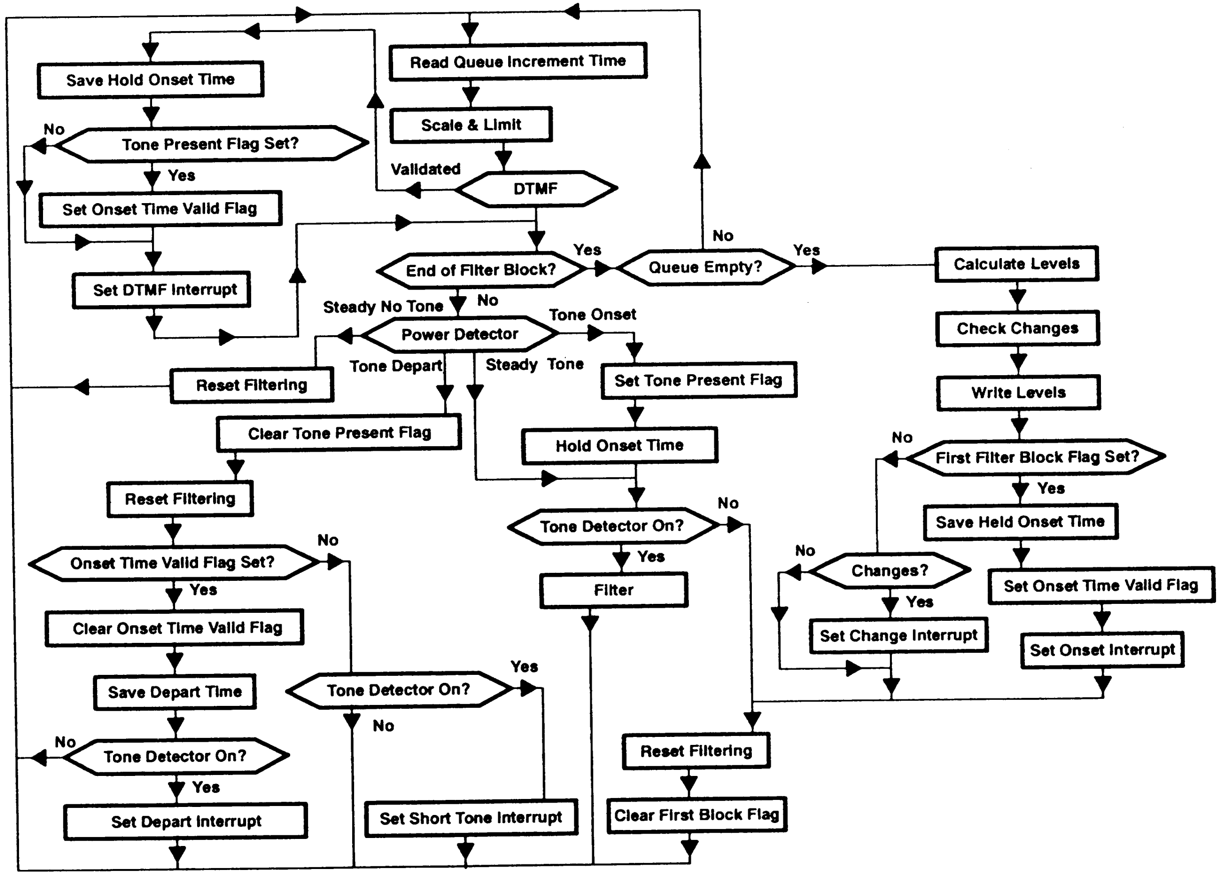


Figure 15. Tone Detector Flow Chart (Detailed)

Utilization of TMS320C17 Resources

Central Arithmetic Logic Unit (CALU)

The throughput capability of the CALU is one of the keys to the success of the TMS320 family. At the center of the CALU is a two's-complement 16 by 16 hardware multiplier with a 32-bit product register, which provides a result in a single cycle. Other features interfacing directly to the multiplier are the 32-bit ALU, 32-bit accumulator (ACC), two shifters and the data bus as shown in Figure 16. One input of the multiplier is provided directly from data memory via the data bus, the other is from the previously loaded temporary (T) register.

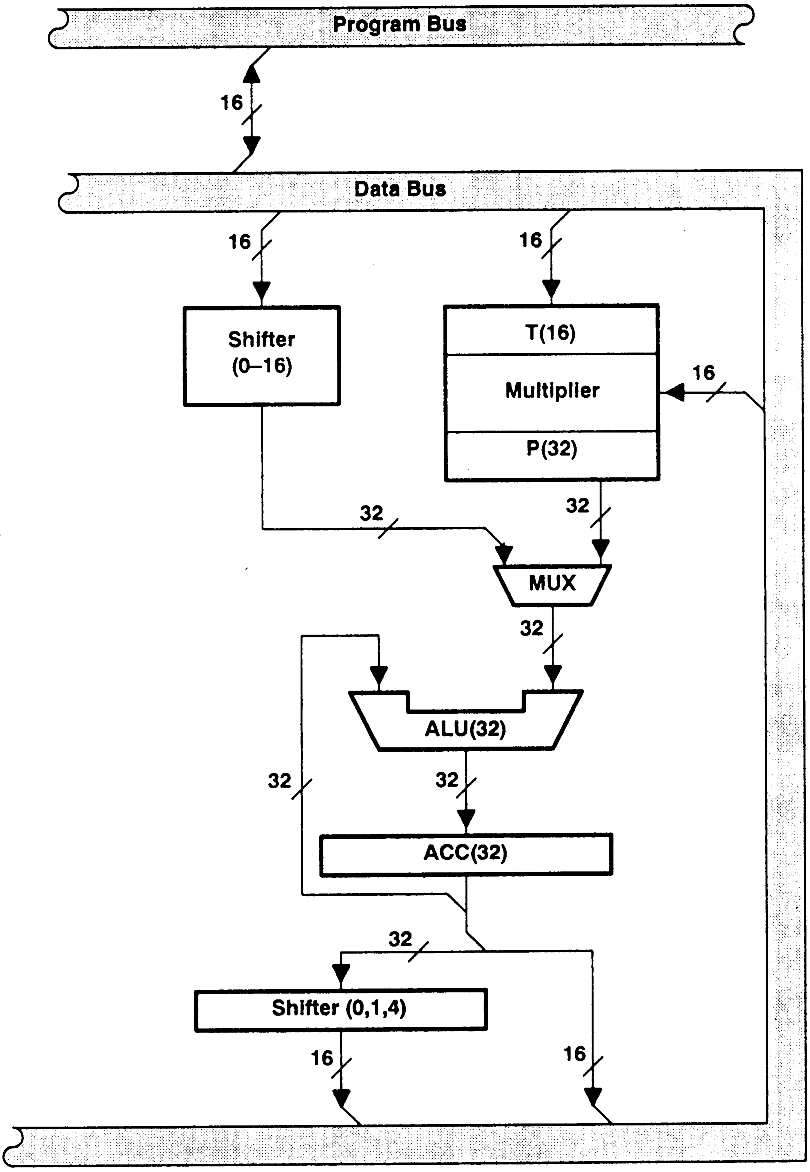


Figure 16. Central Arithmetic Logic Unit (CALU)

The hardware intensive approach of the CALU allows mathematically intensive algorithms to be performed very efficiently. To show its performance, the following example is taken from the ENVDET (envelope detector) routine in the source listing. Its function is to implement a smoothing filter of the form:

$$\text{ENVEL} = \frac{((2^{15} \times \text{ENVEL}) + \text{ABS}(32 \times \text{EDF} \times \text{SAMPLE}) - (32 \times \text{EDF} \times \text{ENVEL}))}{2^{15}}$$

Initial conditions are that EDF is stored in data memory location TEMP and the current envelope detector output is stored in ENVEL.

LAC	TEMP,5	Puts EDF $\times$ (2^5) into the accumulator, using the barrel shifter to shift EDF from data RAM location TEMP left by 5 bits.
SACL	TEMP	Stores 32 $\times$ EDF back into TEMP.
LT	TEMP	Loads 32 $\times$ EDF from TEMP into T register.
MPY	SAMPLE	Multiplies the data value from SAMPLE by 32 $\times$ EDF and puts result into the P register.
PAC		Copies P register result into accumulator. Note that an instruction which transfers the P register into the accumulator must always follow a multiply in order to ensure the contents of the P register are not lost if an interrupt occurs during the multiply instruction. ACC = 32 $\times$ EDF $\times$ SAMPLE
ABS		The absolute value (magnitude) of the result is left in the accumulator.
MPY	ENVEL	Multiplies the data value from ENVEL by 32 $\times$ EDF and puts result into P register. Note that it is not necessary to reload the T register.
SPAC		Subtracts P register contents from accumulator. ACC = ABS(32 $\times$ EDF $\times$ SAMPLE) $-$ (32 $\times$ EDF $\times$ ENVEL)
ADD	ENVEL,15	Adds current value from ENVEL to accumulator with a left shift of 15 (i.e. multiplied by 2^{15}). ACC = ABS(32 $\times$ EDF $\times$ SAMPLE) $-$ (32 $\times$ EDF $\times$ ENVEL) + (EDF $\times$ 2^{15})
ADD	ONE,14	Adds the value 2^{14} to the accumulator to round up the result.

SACH	ENVEL,1	Stores the upper 16 bits of the accumulator in ENVEL with a left shift of one to remove the extra sign bit (caused by multiplying two two's-complement numbers). As it is storing the high-order accumulator, the result is effectively divided by 2^{15}.

Thus we now have the result:

$$\text{ENVEL} = \frac{((2^{15} \times \text{ENVEL}) + \text{ABS}(32 \times \text{EDF} \times \text{SAMPLE}) - (32 \times \text{EDF} \times \text{ENVEL}))}{2^{15}}$$

This calculation takes 11 instructions and executes in 11 cycles or approximately 2.15 μs with a 20.48 MHz operating frequency.

Interrupts

The TMS320C17 has an extended interrupt capability to handle a number of possible sources. These are external interrupt and serial port interrupts for any of FSR (external receive framing input), FSX (external transmit framing input) and FR (internal framing output).

Two steps are required to enable an active interrupt to the device. First, the individual interrupt must be enabled by writing to the appropriate bits in the system control register. Secondly the master interrupt circuitry should be enabled by the EINT instruction.

When an interrupt occurs, its source can be determined by reading the interrupt flag bits in the system control register. Program control can then branch to the appropriate interrupt handler.

For a full explanation of TMS320C17 interrupts refer to sections 3 and 5 of the *First-Generation TMS320 User's Guide* (Reference [6]).

Interrupt Initialization

In our example interrupts are initialized by the WRESET (warm reset handler) routine as follows. CTLPRT and CTLUPR are equated to 0 and 1 respectively to point to the I/O locations of the lower and upper 16 bits of the 32-bit system control register. Some data RAM locations are also previously set up as shown.

CTL320	contains	FD9Fh
MS00FF	contains	00FFh
ONE	contains	0001h

The interrupt initialization code also includes the serial port initialization. The use of the serial ports within this application is covered briefly in DTMF Telephone Tester section. The following listing should also be referred to when reading that section.

OUT	CTL320,CTLPRT	Sets lower 16 control bits to FD9Fh. This resets all interrupt flags, enables external and FR interrupts only, connects I/O port 1 to the upper control register, sets the XF output low, enables the serial port, selects and enables A-law encoding/decoding and selects SCLK (serial clock) as an input.
OUT	CTL32U,CTLUPR	Sets upper control bits to 0CFEh. This sets SCLK to 2.048MHz, sets FR to 8KHz, selects sign magnitude companding and selects FR for fixed data rate operation.
LAC	CTL322	ACC = 7C90h.
SACL	CTL320	Stores 7C90h back into CTL320, for future use.
OUT	CTL320,CTLPRT	Sets lower control bits to 7C90h. This sets SCLK to be an output, connects I/O port 1 to the serial port companding hardware, selects internal framing and leaves other options unchanged. Note it does not clear interrupt flags.

Interrupt Handler - Entry

When a valid enabled interrupt is received, program execution jumps to program memory location 2. In our code, this contains a branch to label INTHDL which is at the start of the Interrupt Handler routine.

This routine contains the detailed steps for handling a serial port interrupt or an external (host interface) interrupt. All that is explained here is the code concerned with interrupt management.

SST	SRSAVE	Saves the current contents of the status register in data memory location SRSAVE. This is automatically in page 1 of data RAM, regardless of the value of the data page pointer.
LDPK	1	Sets the data page pointer to page 1.
SACH	ACCUHI	Saves the current contents of the accumulator in data memory location ACCUHI (data page 1).
SACL	ACCULO	As above.
LDPK	0	Resets the data page pointer to page 0.
SAR	AR0,ARSAVE	Saves the contents of AR0 in ARSAVE (data page 0).
LARP	0	Ensures auxillary register pointer is 0 for future indirect memory accesses.
IN	ITEMP,CTLPRT	Stores lower order system control register in data memory location ITEMP (data page 1).

LAC	ONE,3	Loads 2^3 into accumulator, ACC = 0004h.
AND	ITEMP	ANDs data in ITEMP with 0004h in order to test whether bit 2 in system control register is 1, (i.e. is it a serial port interrupt?).
BZ	NOTCDC	If bit 2 not set, it is not a serial port (codec) interrupt and execution branches to the routine for external (host interface) interrupts.

Interrupt Handler - Exit

All external interrupts return through the following path

LACK	7	Loads 7 into accumulator.
ADDS	CTL320	Adds CTL320 (7C90h) to accumulator with sign extension suppressed as we are not dealing with two's-complement numbers. ACC = 7C97h
SACL	ITEMP	Store accumulator into ITEMP.
OUT	ITEMP,CTLPRT	Clears all interrupts except internal framing, leaves all other bits in system control register unchanged.

Note only non-codec interrupts are cleared here. Codec (serial port) interrupts are cleared at the start of the codec interrupt routine. This is because the two interrupt sources are asynchronous. Thus it is quite possible for a serial port interrupt to occur during the external interrupt routine and vice-versa. It is essential that these "pending" interrupts are not lost during the handling of the previous interrupt.

The codec interrupts join the external interrupt exit path here

LAR	AR0,ARSAVE	Restores AR0 value to that prior to entering interrupt routine.
LDPK	1	Sets data page pointer to page 1.
ZALH	ACCUHI	Loads high accumulator with exact copy of ACCUHI.
ADDS	ACCULO	Loads low accumulator with exact copy of ACCULO with sign extension suppressed to leave high accumulator unaffected.
LST	SRSAVE	Restores status register value with that prior to entering interrupt routine.
EINT		Enables interrupts. This instruction always waits until the following instruction has completed execution so that interrupts are not nested.

RET	Returns program control to the point at which the interrupt occurred.

Serial Ports

Serial port initialization occurs at the same time as interrupt initialization as both involve the use of the TMS320C17 Control Registers. This is covered in detail in the interrupt section above.

This application uses a single serial input only. A TCM2917 codec chip operated in the fixed data rate mode is used to provide analog to digital conversion. A 2.048 MHz clock (SCLK) is provided by the TMS320C17 along with a framing signal (FR) giving a sampling rate of 8 KHz. With CDCPRT having been equated to one, data transfer is simply by the use of the following instruction

IN	*,CDCPRT	Inputs data from I/O port 1 which has been switched to accept serial input from the companding hardware by a previous write of a one to control register bit 8.

Hardware Implementation

The example outlined below is a possible design for a tone detection system as a peripheral to an IBM XT or AT compatible PC bus. Figure 17 shows the complete circuit schematic for this design. The circuit uses only four integrated circuits to implement a full-functionality tone detector. The signals required from the PC bus are SA0 - SA9 (latched address bus), D0 - D7 (8-bit data bus), $\overline{\text{IOW}}$ (I/O Write), $\overline{\text{IOR}}$ (I/O Read), RESET DRV (System Reset), and AEN (Address enable for DMA). Figure 18 shows the PC bus activity for these signals during an I/O operation. For more detailed information on the function and behaviour of these signals see References [3] and [4].

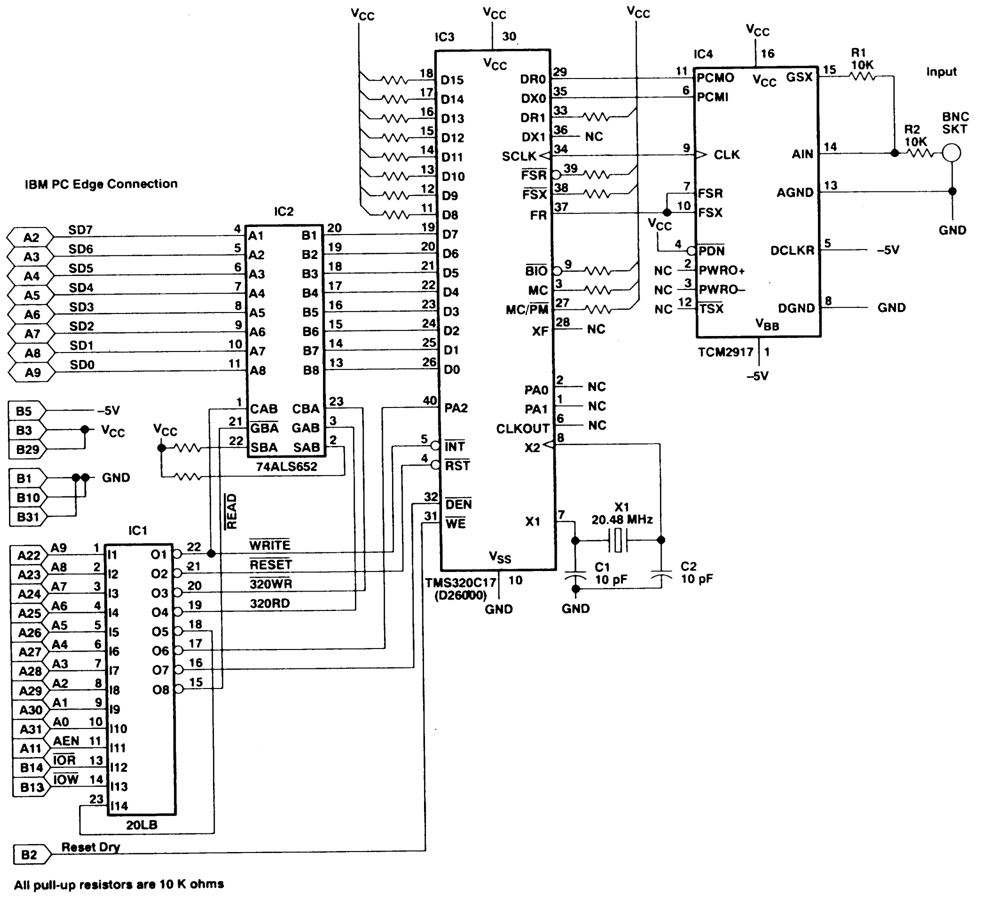


Figure 17. Tone Detector PC Application Circuit Diagram

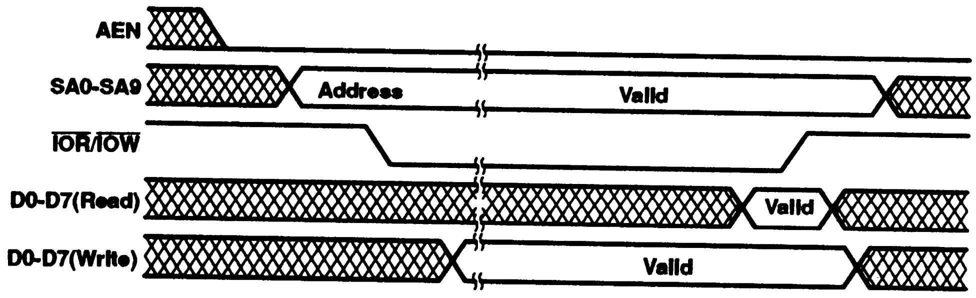


Figure 18. PC Bus Activity I/O Read or Write

The XF (external flag) pin of the TMS320C17 may also be used to signal an interrupt on one of the PC bus lines IRQ3 - IRQ7 (Interrupt requests), if it is desired to have an interrupt driven and not a polled interface. The example shown is based on a polled interface and does not utilize host interrupt.

Host Read/Write Decode

The PAL (programmable logic array) can give a host read or write function at any address in the range 0 to 03FFh (hexadecimal). Only one I/O address is used by the tone detection system in this example. For use in a PC, any free address in the I/O space could be chosen. The AEN signal is also passed to the PAL to ensure that the system is not mistakenly accessed during a direct memory access (DMA) cycle.

Assume that the I/O address of the tone detector is 0300h. The equations for the host read and write strobes would be as follows:

$$\overline{\text{READ}} = \overline{\text{A9}} \# \overline{\text{A8}} \# \text{A7} \# \text{A6} \# \text{A5} \# \text{A4} \# \text{A3} \# \text{A2} \# \text{A1} \# \text{A0} \# \overline{\text{IOR}} \# \text{AEN}$$

$$\overline{\text{WRITE}} = \overline{\text{A9}} \# \overline{\text{A8}} \# \text{A7} \# \text{A6} \# \text{A5} \# \text{A4} \# \text{A3} \# \text{A2} \# \text{A1} \# \text{A0} \# \overline{\text{IOW}} \# \text{AEN}$$

where # represents the logical OR function.

TMS320C17 I/O Read/Write Decode

The PAL also provides the decode function for TMS320C17 IN and OUT (read and write) operations. A TMS320C17 read and a data write always use I/O port 4. A status write is made to port 6. Ports 0 and 1 are reserved for internal functions of the TMS320C17. Other ports are not implemented in this system.

The equations for a TMS320C17 read and write are as follows:

$$\text{320RD} = \text{DEN} \,\&\, \text{PA2}$$

$$\overline{\text{320WR}} = \overline{\text{WE}} \# \overline{\text{PA2}}$$

Host Data Write

Upon receipt of the correct I/O address and the I/O Write strobe, the data present on the PC bus is latched into the 74ALS652 on the rising edge of I/O Write. Simultaneously, an interrupt is given to the TMS320C17. As previously described, the TMS320C17 responds to this interrupt by performing a read operation from its input port 4.

The TMS320C17 read is implemented by PA2 being set high and $\overline{\text{DEN}}$ (data enable) acting as a read strobe. While data enable is low, the high-impedance outputs of the 74ALS652 are enabled and the TMS320C17 reads an 8-bit value. This contains the address of the register to be accessed and the read/write bit which is set to indicate a host write in this case.

The read of port 4 is then followed by a write of the current contents of the status register from the TMS320C17 to output port 6. This is implemented by PA2 and PA1 being set high and $\overline{\text{WE}}$ (write enable) being used as a write strobe. When write enable goes high to signify the end of the write, the data on the low order data bus (D7 to D0) of the TMS320C17 is latched into the 74ALS652.

The second part of the host data write operation is an exact duplication of the above sequence of events. It would then be normal to read the status information returned at the end of the cycle. This is done by a simple I/O read from the address of the board which enables the contents of the 74ALS652 onto the PC data bus.

Host Data Read

This operation is based on the same sequence of events as above, as indicated in Host Read Cycle section.

Host Reset

The active high RESET DRV signal is taken from the PC bus, inverted and applied to the TMS320C17 RS input (pin 4).

Host Interrupt

As mentioned briefly above, the TMS320C17 uses the external flag (XF) pin (pin 28) to signal an interrupt to the host. This interrupt may come from a number of sources as described in Control section. This signal is active low and is set to a high level after a reset to the TMS320C17. There is a period of 2 ms after the release of reset for which the state of the interrupt should be ignored, as it is set inactive only by execution of the appropriate instruction. The state of XF is therefore undefined for the period between the application of reset and the execution of the instruction which initializes it to the inactive state.

The easiest method to overcome this would be only to enable the appropriate host interrupt line at least 2 ms after the release of reset.

Host Handshake

There is no host handshake implemented on the example application described here. The maximum length of time which a single read or write can occupy is 20 μs. The host should ensure that consecutive accesses do not occur more closely than this.

Analog Interface

This function is performed by an industry-standard combined PCM codec and line filter (COMBO), the TCM2917 (see Reference [5] which provides A/D and D/A conversion as well as transmit and receive filtering. In this application the codec is set to a gain of 1. The TCM2917 performs A-law companding and operates in this circuit in the fixed data rate mode of 2.048 MHz. As this application was developed in Europe, the A-law companding TCM2917 was used. For applications in North America this may be replaced with the TCM2916 which provides μ-law companding and is pin-for-pin compatible with the TCM2917. There is a small change to be made to the area of program which initializes the control registers in the TMS320C17. This is covered in detail in Substitution of TCM2916 for TCM2917 subsection.

The TCM2917 interfaces directly to one of the two serial ports on the TMS320C17 which were designed to facilitate the use of this type of device (see References [1] and [6] for further information).

Host Interface

The tone detector function described in this application note appears to the host CPU bus as a single 8-bit parallel port. This port is used as shown below to give access to the sixteen read and write registers within the TMS320C17.

In the particular example presented here the interface is of the polled access type. An interrupt driven interface can be implemented by setting the appropriate bits of the tone detector control register and connecting the XF pin of the TMS320C17 (pin 28) to a host interrupt input.

Host Write Cycle

The host CPU writes to one of the 16 available registers by a four step process as shown in Figure 19.

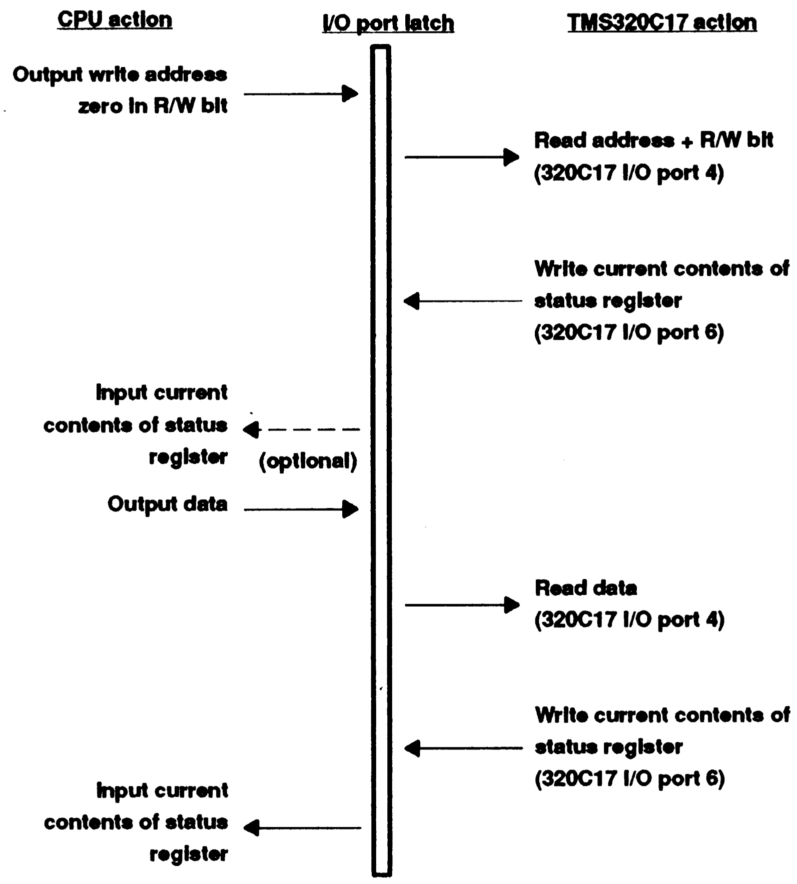

CPU action
I/O port latch
TMS320C17 action
Output write address zero in R/W bit
Read address + R/W bit (320C17 I/O port 4)
Write current contents of status register (320C17 I/O port 6)
Input current contents of status register
(optional)
Output data
Read data (320C17 I/O port 4)
Write current contents of status register (320C17 I/O port 6)
Input current contents of status register

Figure 19. Host Write Cycle

The write cycle is initiated by an output from the host CPU to the I/O port or memory location occupied by the tone detector. The first byte of data transferred is a command byte which contains the address of the register to be written to and the read/write bit set to a zero to indicate a write operation. The bit assignment is as shown below.

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	A3	A2	A1	A0
			R/W	——— Register address ———			

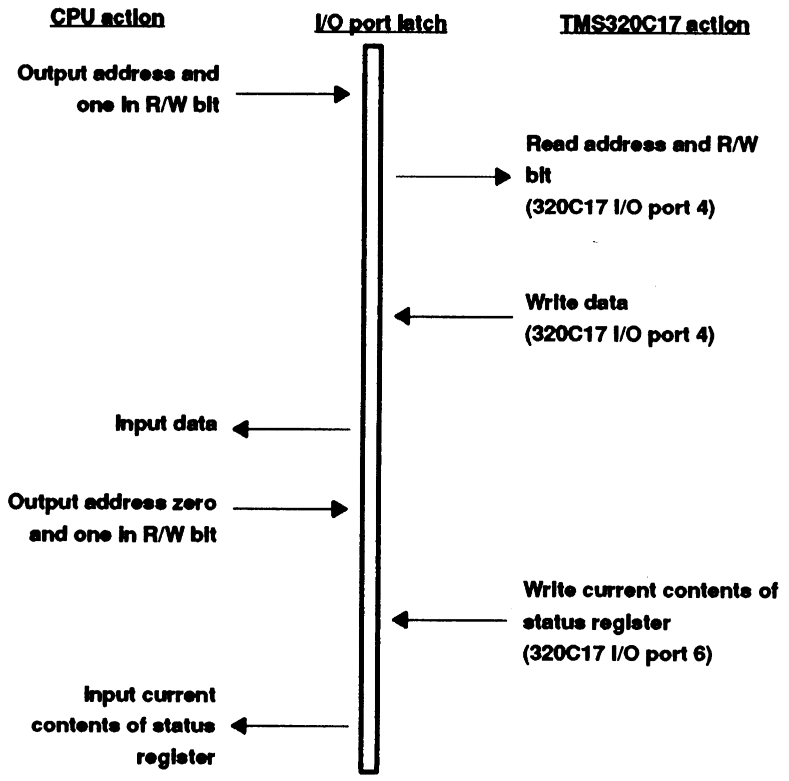
A3 is the most significant bit of the tone detector register number and A0 is the least significant bit.

Following this host CPU command the TMS320C17 will make the current contents of its status register available for input by the CPU. It is not usual for the host CPU to read the status information at this point.

This is followed by a host CPU write of the data to be transferred into the tone detector register. The operation is completed by the TMS320C17, which again makes the current contents of its status register available. It would be normal for the host CPU to read this status byte from the I/O port at this time.

Host Read Cycle

The read cycle is initiated by the host CPU in a similar way to the write cycle above, and is shown in Figure 20.



CPU action
I/O port latch
TMS320C17 action
Output address and one in R/W bit
Read address and R/W bit (320C17 I/O port 4)
Write data (320C17 I/O port 4)
Input data
Output address zero and one in R/W bit
Write current contents of status register (320C17 I/O port 6)
Input current contents of status register

Figure 20. Host Read Cycle

In this case, the read/write bit is set to a one to indicate a read. Following the initial host CPU write of the address, the TMS320C17 makes the contents of the addressed register available for the host CPU to read. The cycle is completed when the host CPU issues a second register read request with an address of zero (status register) and the TMS320C17 makes available the current contents of its status register for the host CPU to read.

Host Access Considerations

The host CPU may not attempt to perform any new access of a tone detector register before the previous access is complete. A read operation must be fully completed before a write is initiated and vice versa. Additionally, neither read nor write operations should be nested. Both the host read and host write should be regarded as discrete tasks to be executed in isolation from any other host access.

A delay should be allowed between the host CPU writing the register address to the tone detector and reading the subsequent response (data for a read cycle, status for a write cycle). This delay should be a minimum of 20 μs. No delay is necessary between reading the response and performing a subsequent write operation, but a further minimum 20 μs delay should be allowed prior to the next read. This delay allows the TMS320C17 to retrieve the correct data from its data memory, perform any necessary calculations and output it to the interface latch.

Host Interface Registers

Although the tone detector only occupies one physical 8-bit read/write host location, the full interface is implemented by sixteen read and write registers within the TMS320C17. Their allocation is shown below:

Address	Read Register	Write Register
0	Status	Control
1	Mode	Mode
2	DTMF digit	Envelope decay factor
3	Tone arrival (MS byte)	Upper threshold
4	Tone arrival (LS byte)	Lower threshold
5	Tone departure (MS byte)	Filter length
6	Tone departure (LS byte)	Passband width
7	Current time (MS byte)	Change threshold
8	Current time (LS byte)	Frequency (MS byte)
9	Band 1 signal level	Band 1 frequency (LS byte)
A	Band 2 signal level	Band 1 frequency (LS byte)
B	Band 3 signal level	Band 1 frequency (LS byte)
C	Band 4 signal level	Band 4 frequency (LS byte)
D	Band 5 signal level	Band 5 frequency (LS byte)
E	Band 6 signal level	Band 5 frequency (LS byte)
F	Total signal level	Filter select

Where MS byte refers to the most significant (upper) byte of a 16-bit word, and LS refers to the least significant (lower) byte of a 16-bit word.

Register Read Functions

Except where specified all of the following read registers are set to zero by a hardware reset.

Status

D7	D6	D5	D4	D3	D2	D1	D0
	ST	DT	TC	TA	TD		

ST— This bit is set to zero when the departure of a tone is detected by the envelope detector before it has been validated as a DTMF tone or a filtering operation has been completed on the tone.

DT— This bit is set to zero when the occurence of a valid DTMF tone pair is detected.

TC— This bit is set to zero when a change in tone is detected.

TA— This bit is set to zero when the arrival of a tone is detected by the envelope detector, and the tone has been validated as a DTMF digit or a filtering operation has been completed on the tone.

TD— This bit is set to zero when the departure of a tone is detected by the envelope detector.

Each of the bits in the register are set to one by writing the appropriate value to the IACK bits (bits 2-4) of the MODE register (see Mode section).

A reset will cause all of the bits of the status register to be set to one.

Mode

D7	D6	D5	D4	D3	D2	D1	D0
TEST	DTMF	TONE	IACK2	IACK1	IACK0	RC1	RC0

Each of the bits in this register, except RC1 and RC0, simply reflect the last value written to the corresponding bit in the mode register.

RC1 These two bits together form the result code generated by a self
RC0 test operation by the tone detector.

The meanings of the result codes are as follows:

RC1	RC0	Meaning
0	0	Clock failure
0	1	Test successfully completed
1	0	RAM failure detected
1	1	ROM failure detected

DTMF Digit

D7	D6	D5	D4	D3	D2	D1	D0
OVRUN				DD3	DD2	DD1	DD0

OVRUN— This bit is set to one when there has been an overrun of received DTMF digits, ie. a new digit has been received when the DT bit in the status register was set to zero (before the host has acknowledged the receipt of a previous digit). OVRUN remains set to one until a DTMF digit is received while the DT bit in the status register has a value of one. The digit indicated by DD3-DD0 is the last received digit regardless of the state of OVRUN.

DD3 to— DD0 These four bits together identify the last valid received DTMF digit.

The digits are identified as follows:

DD3	DD2	DD1	DD0	Received Digit
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	A
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	B
1	0	0	0	7
1	0	0	1	8
1	0	1	0	9
1	0	1	1	C
1	1	0	0	*
1	1	0	1	0
1	1	1	0	#
1	1	1	1	D

Tone Arrival (MS Byte and LS Byte)

The two tone arrival registers are read by the host CPU in conjunction. They report the time at which the arrival of a tone was detected. The 16-bit value formed by (256 × MS) + LS is treated as an unsigned integer giving the time at which tone arrival was detected in milliseconds. This time is taken from the contents of the current time register (see Current Time section) at the moment of the tone arrival being recognised by the power detector.

The tone arrival registers are updated when either a DTMF digit is detected, or a filtering operation is completed.

Tone Departure (MS Byte and LS Byte)

The two tone departure registers are read by the host CPU in conjunction. They report the time at which the departure of a tone was detected. The 16-bit value formed by (256 × MS) + LS is treated as an unsigned integer giving the time at which tone departure was detected in milliseconds, as taken from the current time register.

Neither the tone arrival or tone departure registers are updated by the arrival or departure of a short tone, i.e. one which had departed before being recognised as a DTMF digit, and before a tone receiver filtering operation had been completed on it.

Current Time (MS Byte and LS Byte)

The two current time registers are read by the host CPU in conjunction. They report the current time indicated by the tone detector module. The 16-bit value formed by (256 × MS) + LS is treated as an unsigned integer giving the current time in milliseconds. Reading the current time (MS byte) register causes the value of the current time (LS byte) register to be copied into a holding register. In order to get a correct reading of the full 16-bit value of the current time the MS byte should therefore be read first.

When current time reaches the maximum value of 65535, the next increment takes it to zero. The current time increments every millisecond upon release of hardware reset.

Band 1-6 Signal Level

The signal levels received in each of the frequency bands specified are reported in these six frequency band signal level registers. The values read from these registers are to be interpreted as 8-bit unsigned integers, SL. If a value of SL is read from a register, then the signal level represented is:

$$\frac{(5.30 \times \mathrm{SL})}{\mathrm{GAIN}} \text{ mV rms (root mean square)}$$

See Mode subsection in Host Interface section for a description of the gain factor (GAIN).

Typical values which may be read are as follows:

SL	Signal Level	Codec Level
40	212.0/GAIN mV rms	−14.1 dBm0 − G dB
254	1346.0/GAIN mV rms	+ 2.0 dBm0 − G dB

An input signal level of greater than 1346/GAIN mV rms will result in a value of SL = 255.

When the DTMF bit in the mode register is set to one, the values read from Band Signal Level Registers 4 to 6 are all zero as only three frequency bands can be monitored while the DTMF receiver is active. The DTMF bit must be set to a zero if bands 4 to 6 are to be monitored.

Total Signal Level

The signal level received over the frequency range 300 Hz to 3400 Hz is reported in the total signal level register. The number format is identical to that described for the band 1-6 signal level registers.

Register Write Functions

Except where explicitly stated, a hardware reset will set each register to zero and, when the contents of any register are changed, the tone detector uses the new value immediately.

Control

D7	D6	D5	D4	D3	D2	D1	D0
	INTST	INTDT	INTTC	INTTA	INTTD		

Writing a one to any of the bits in the control register enables an interrupt to be signalled on the XF pin of the TMS320C17 when the corresponding bit in the status register is set to zero.

Mode

D7	D6	D5	D4	D3	D2	D1	D0
TEST	DTMF	TONE	IACK2	IACK1	IACK0	GF1	GF0

The functions of the bits in the mode register are as follows:

TEST—	Writing a one to this bit starts a self test operation. The result of the test is reported in the lower bits of the mode register. As long as TEST is a one the tone detector remains in the TEST mode and no register accesses may take place. The self-test is terminated by writing a zero to TEST after which the tone detector is left in the default state assumed after a reset. A self test operation takes approximately 6 ms.
DTMF—	Writing a one to this bit enables the detection of DTMF digits. On entering the active state, the DTMF receiver begins looking for DTMF digits as though it had been monitoring a silent line in the recent past.
TONE—	Writing a one to this bit enables the detection of tones. When the tone detector is turned on, it will wait for the envelope detector to indicate that a tone is present before starting filtering operations.

IACK0— to IACK2 — The pattern written to these bits selects which of the five possible interrupt conditions from the tone detector module is being acknowledged. The acknowledgement of an interrupt causes the corresponding status bit to be set to the one state.

The selection patterns are as follows:

IACK2	IACK1	IACK0	Interrupt to be acknowledged	
0	1	0	Tone Departure	(TD)
0	1	1	Tone Arrival	(TA)
1	0	0	Tone Change	(TC)
1	0	1	DTMF Digit Arrival	(DT)
1	1	0	Short Tone	(ST)

Other patterns have no effect

GF1-GF0— The two bit pattern written to these bits selects which of three gain factors is applied to the input signal before it is passed to the DTMF and tone receivers and the envelope detector. By writing a suitable value to these bits, it is possible to adjust the tone detector module to accommodate very loud or very quiet signals. The selection patterns are as follows:

GF1	GF0	Gain Factor (GAIN)	Relative Gain (G dB)
0	X	4	12
1	0	1	0
1	1	16	24

Envelope Decay Factor

The time constant of the envelope detector is the time taken for the output of the detector to reach 63% of its final value. The value written to the envelope decay factor register is treated as an 8-bit unsigned integer, EDF. If the time constant required for the envelope detector is t, then EDF should be specified as

$$EDF = 1024 \times [1 - \exp(-1/(8000t))].$$

For example for a time constant of 1.0 ms, EDF should be set to 120.

A reset will cause this register to be set to a value of 120.

Upper Threshold

The upper threshold is the signal level at the output of the envelope detector at which the arrival of a tone is recognized. The number written to the upper threshold register is treated as an 8-bit unsigned integer, UT. If the signal level required for this threshold is V_{ut} Volts rms, then UT should be specified as

$$UT = 254 \times (0.743 \times GAIN \times V_{ut})$$

For example for an upper threshold of 425/GAIN mV, UT should be set to a value of 80. This represents a codec input of −8.0 dBmO − G dB.

A reset will cause this register to be set to a value of 255.

Lower Threshold

The lower threshold is the signal level at the output of the envelope detector at which the departure of a tone is recognized. The lower threshold is specified in exactly the same way as the upper threshold described above.

If the value programmed into the lower threshold register is larger than the value programmed to the upper threshold register, the value in the lower threshold register is taken as the threshold for both tone arrival and tone departure.

A reset will cause this register to be set to a value of 255.

Filter Length

The filter length register defines the number of samples of the input signal which are required to produce one result from the tone detector. The rate at which the codec feeds samples to the tone detector is 8000 samples per second, or one sample every 125μs. The value which is written to this register is treated as an 8-bit unsigned integer, FL. The length of filter specified by the value FL is

$$\frac{16384}{FL + 16} + 1 = N \text{ samples}$$

For example, for a filter length of 410 samples, FL should be set to 24, giving a filter duration of 51.3 ms.

The filter length defines the steepness of cutoff at the filter band edge. Figures 21 and 22 give an indication of the filter band edge shape for both wide filters and narrow filters of different lengths. They should be treated as indicative of the performance of the tone detector.

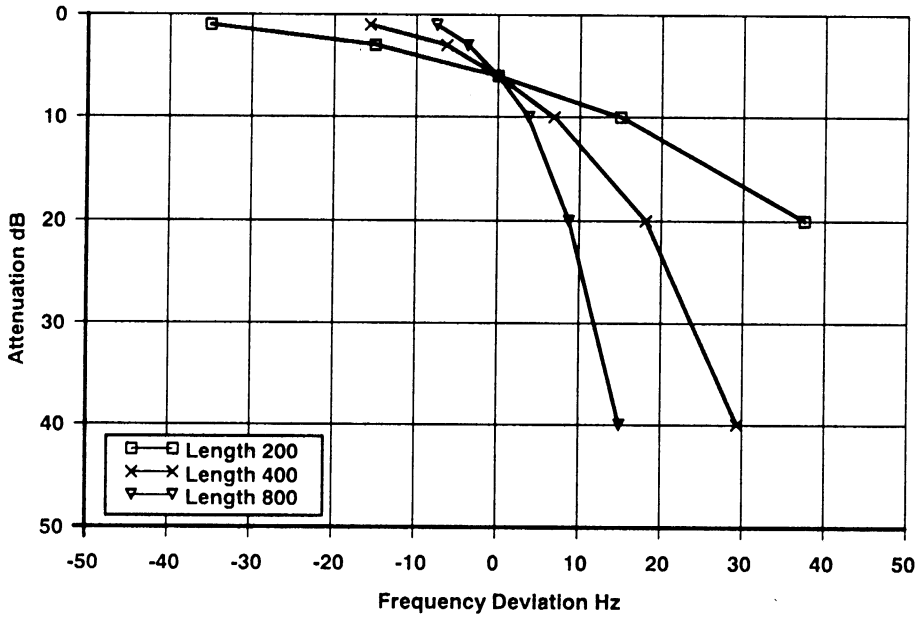


Figure 21. Filter Band Edge Shape - Wide Filter

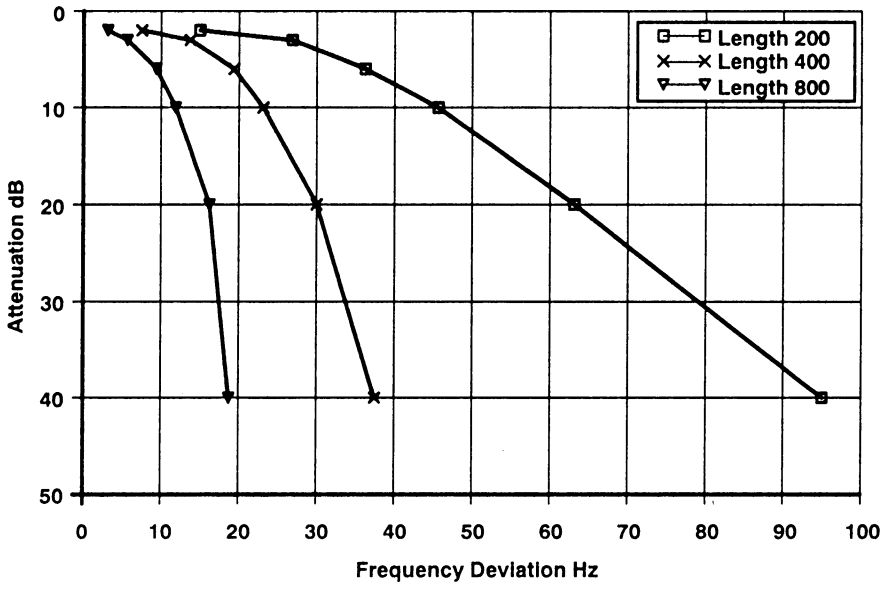


Figure 22. Filter Band Edge Shape - Narrow Filter

When contents of this register are changed, the tone detector waits until the start of the next filtering operation before using a new filter-length value.

A reset will cause this register to be set to a value of 50 (250 samples).

Passband Width

The bandwidth of the bandpass filters used by the tone detector is specified by the passband width register. The value which is written to this register is treated as a 6-bit unsigned integer, PW. If a bandwidth of Y Hz is required, then PW should be specified as:

$$PW = Y \times 0.128$$

The maximum permitted value for PW is 63, giving a passband width of 492 Hz.

The bandpass filters used by the tone detector are symmetrical about the center frequency, i.e. a bandwidth of X Hz defines that frequencies which deviate by up to X/2 Hz from the center frequency fall within the passband.

Change Threshold

At the end of each filtering operation (except the first after tone onset) the signal received at each of the monitored frequencies is compared against the signal received during the previous filtering operation. If the signal level at any one of the monitored frequencies has crossed the signal level threshold defined in the change threshold register, then the Tone Change status bit is set in the status register. The change threshold is defined in an identical manner to the upper threshold described above.

When the contents of this register are changed, the tone detector uses the new value of change threshold on the next signal level comparison.

Frequency (MS Byte)

The 8-bit value, FMS, written to the frequency register (MS byte) forms the most significant byte of the 16-bit specifier of a filter center frequency. When a value is written to one of the frequency (LS) registers, the current contents of frequency (MS) is concatenated with the 8-bit LS value defined below to form the 16-bit frequency specifier. The frequency (MS byte) register must therefore contain the desired value when the LS byte is written.

Band 1-6 Frequency (LS Byte)

The 8-bit value, FLS, written to one of the band 1-6 frequency (LS byte) registers is concatenated with the 8-bit value most recently written to the frequency (MS byte) register to form the 16-bit specification for the filter center frequency. If a center frequency of G Hz is required, then FMS and FLS should be specified as follows:

$$(\text{FMS} \times 256) + \text{FLS} = 8.192 \times G$$

or $\text{FMS} = (8.192 \times G) \text{ div } 256$
and $\text{FLS} = (8.192 \times G) \text{ mod } 256$

Filter Select

D7	D6	D5	D4	D3	D2	D1	D0
		F1	F2	F3	F4	F5	F6

Writing a one to any of the bits in the filter select register causes the corresponding filter to adopt the passband width specified by the passband width register (wide filter). Writing a zero causes the filter to adopt a zero passband width (narrow filter).

A reset will cause each of the bits in this register to be set to one.

Applications and Customization

The combination of a programmable tone receiver and a CEPT DTMF decoder in a single chip opens up a wide range of potential applications. The operation of the device across the 300-3400 Hz band targets its use towards telephony, but this is by no means the only area to which it can be applied.

The examples shown here are chosen from the more obvious potential applications. Some examples do not utilize the full power of the system, but they will hopefully serve to illustrate the capabilities of the tone detector and act as a stimulus for the development of innovative designs.

Secure Off-Site Control

The tone detection system described may be used within a secure off-site control system. An increasing amount of such equipment is now available, designed to respond to commands given remotely via a telephone line, as shown in Figure 23. These commands are typically a single or a sequence of DTMF tone(s), and may be supplemented by special tones.

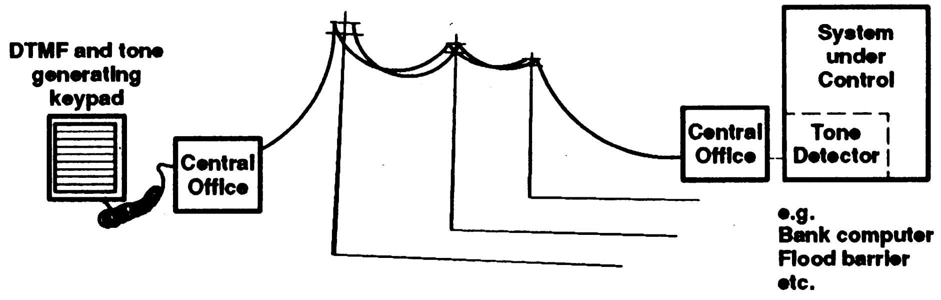


Figure 23. Secure Remote Controller

The level of security required varies with each type of equipment depending upon its function. For example, a home answering machine does not require a high level of security to protect its stored messages from being replayed to a remote telephone. At the other end of the scale, it is clearly important that financial information or transactions be heavily protected in the new remote banking systems now becoming available.

Sequences of DTMF tones of varying lengths with various intervals provide one level of security which would be more than adequate for remote activation in the case of the home answering machine. However, DTMF tones are limited by definition to a set of sixteen tones making computer controlled attack (hacking) of any equipment relying on them for protection relatively easy. The method of protection used for cash-cards, etc., where three unsuccessful attempts at breaking a code (the personal identification number, or PIN) result in a machine refusing to return the card is not feasible in that any remotely accessed system must be ready to respond to its authorized user at all times.The system cannot just shut down if it suspects it is under attack from an unauthorized source.

One way of providing the protection needed would be to make the number of possible combinations of activating tones impractically large for any systematic hacking. This could easily be achieved by extending the number of tones capable of detection beyond the sixteen provided by DTMF.

The tone detector presented here makes just such a scheme possible by providing capability for the accurate detection of a single frequency or any combination of up to six simultaneous frequencies within the telephony band. With the added variety of variable lengths of tone presence and absence, and sequential combinations of different tones, it is clear that a very high level of security can be offered. The tone detector offers time stamping of tone arrival, tone change, and tone departure and would thus make it easy for any equipment to which it is attached to decide whether or not to allow access.

Call Monitoring

Call monitoring functions may be implemented using the tone detection system described here. Across the various telephone companies in the world, there is a large variety of call progress tones used. It may also be useful to decode other tones received down a telephone line. An example might be for an answering machine to detect the fact that it is accidentally being called by a modem, or for auto dialing equipment to detect that it has accidentally called a modem. The ability to decode national call progress tones and other random tones received is of particular use in, for example, a PC with an integral telephony function. Here a range of actions may be expected of the PC depending upon the exact nature of the received tone. This application relates directly to the design example presented in the Host Interface section where a four-chip solution is shown for a PC tone detection peripheral.

DTMF Telephone Tester

Using the general purpose tone detection function, a low-cost DTMF telephone tester could be built to check the conformance of a telephone, or any other fixed tone generator, to a particular standard.

With programmable center frequency (to a resolution of 0.12 Hz), programmable passband width and filter cut-off, a precise measure of an incoming tone for conformity is easily made. In a laboratory environment this could again be implemented as a peripheral to a PC. If required, the TMS320C17 could also easily be controlled by any general-purpose 8-bit microcomputer to provide a low cost portable programmable tone tester.

Customization for User Applications

The source code for the TMS320C17 program described here is presented as Appendix A. The code takes up less than half of the on-chip ROM and allows space for user application code to be included on-chip for low chip-count solutions to a number of complex tone decoding tasks.

The TMS320E17 EPROM digital signal processor can be used for the development phase and low-volume manufacturing. For high-volume production, code can be masked onto the TMS320C17 to provide a custom DSP.

To aid integration of additional application code, certain functions of the device are not utilized by the existing source code. Of most importance is the BIO pin (pin 9) which is effectively a software interrupt. By simple insertion of a BIOZ instruction, code execution could branch to special application routines. The XF (external flag) pin of the device is used to signal an interrupt to the host. If (as is the case in the design example in this report) this function is not used, it is simple to reprogram the function of this pin for any desired purpose.

The following notes apply to any customization of the tone detector source code:

1. The correct execution of both the DTMF receiver and tone receiver functions is dependent upon certain time critical functions. Care should be taken to ensure that any change made to the code does not affect the clean handling of the continuous stream of samples from the codec.

2. Any change to the ROM code will require a corresponding change to the checksum word at program memory location 0004h (label CHECKS at bottom of page 489 of the source listing in Appendix A). The checksum test routine (see page 516 of Appendix A) sums all the program memory locations in the code and tests the lower 16 bits of the final sum for zero. It is important to maintain this zero result by adjusting the checksum word.

Alterations that may be made to the tone detector include:

- Substitution of TCM2916 for TCM2917 in North American applications
- Use of the coprocessor port for parallel I/O
- Use of either DTMF or tone receiver code in isolation

Substitution of TCM2916 for TCM2917

To change the TCM2917 codec for a TCM2916 requires only a small alteration in the program code. The only difference between the TCM2917 and the TCM2916 is that the TCM2917 performs A-law compression of its serial PCM data prior to output, and the TCM2916 performs μ-law compression. The TMS320C17 can decode either μ-law or A-law encoded data. The choice between μ-law and A-law is made by the value written to bit 14 in the TMS320C17 control register.

The lower 16 bits of the control register are set by writing data memory location CTL320 to output port zero. CTL320 is initialised with a value of FD9Fh in the existing code, with bit 14 set to a one (A-law conversion). Changing this initial value to BD9Fh will ensure bit 14 is set to a zero (μ-law conversion).

The change to a value of BD9Fh should be made by altering the statement

```
.word       FD9Fh       ; CTL320
```

(second statement below label CONST1 at the bottom of page 490 in Appendix A) to

```
.word       BD9Fh       ; CTL320
```

within the source file.

Use of Coprocessor Port for Parallel I/O

The TMS320C17 features a coprocessor port which provides a direct interface to most 4/8-bit microcomputers and 16/32-bit microprocessors. It is possible for the tone detection system to make use of this port for connection to a variety of possible host CPUs. Figure 24 shows a simplified logic diagram for the coprocessor port. Note that RBLE, TBLF and BIO are not necessary to the tone detector interface as it uses single byte transfers only.

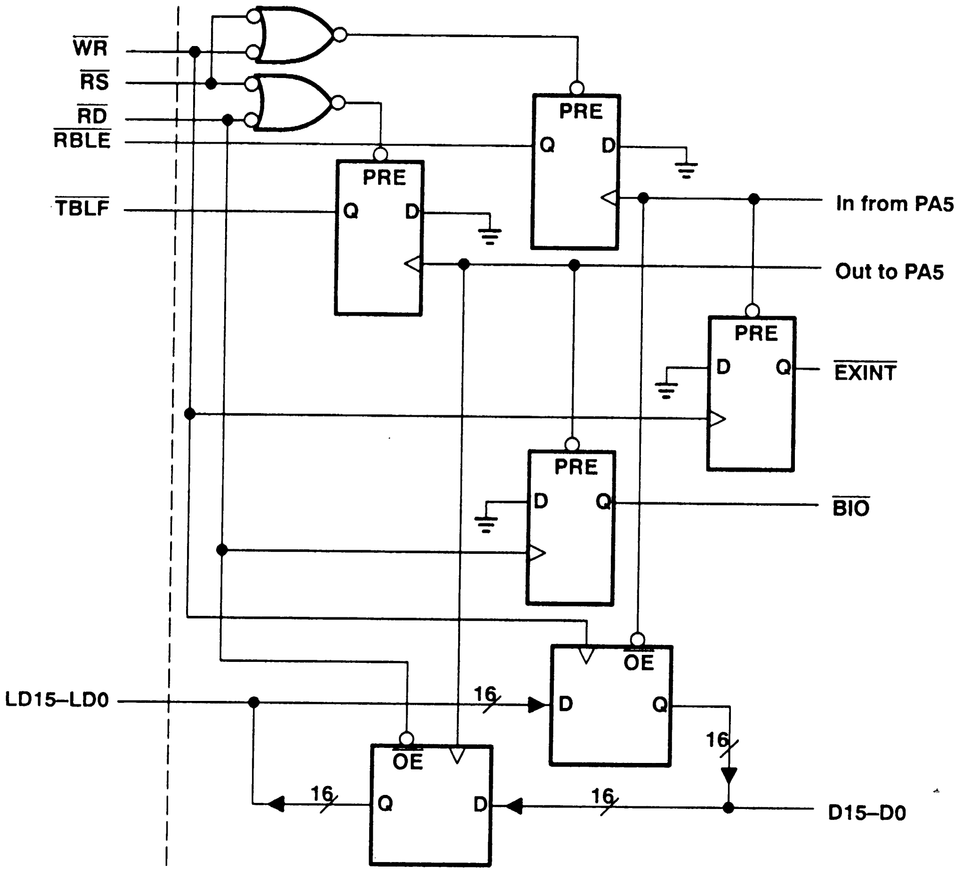


Figure 24. TMS320C17/E17 Simplified Coprocessor Port Logic Diagram

For full details of the coprocessor port refer to the *First Generation TMS320 User's Guide* (Reference [1]).

As an example this section considers an 8-bit interface, as may be required by a TMS7000 8-bit microcomputer.

Coprocessor mode is selected by setting both the MC/$\overline{\text{PM}}$ input (pin 27) and the MC input (pin 3) to low. Bit 30 in the TMS320C17 control register selects either a 16-bit or an 8-bit interface. This should be set to zero for an 8-bit interface. Connections to the TMS320C17 coprocessor port should be as shown in Figure 25.

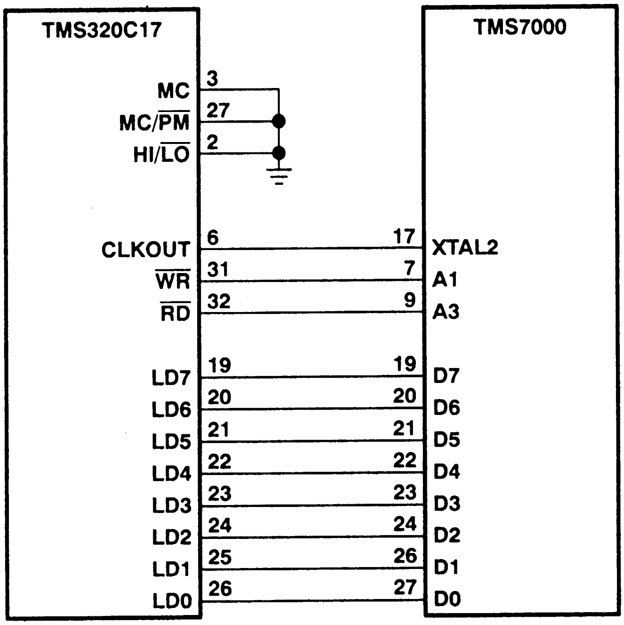


Figure 25. TMS320C17 to 8-Bit Microcomputer (TMS7000) Interface

The coprocessor port is accessed through I/O port 5 in the TMS320C17, and all parallel I/O IN and OUT instructions should be changed to access this port. In the listing file in Appendix A, IN instructions are from port 4 and OUT instructions are to either port 4 or port 6. All of these operations are within the interrupt handler section, INTHDL (see page 510 of the listing).

Data transfers in coprocessor mode operate on the same basis as presented in Host Write Cycle and Host Read Cycle sections, but the host CPU write and read sub-cycles operate differently. Transfers to the TMS320C17 operate as follows:

1. The $\overline{\text{WR}}$ signal is driven low by the microcomputer using a single I/O bit.
2. Data present on the LD7-LD0 bus is written to the receive buffer latch (D7-D0) when the WR signal is driven high by the microcomputer.
3. An internal $\overline{\text{EXINT}}$ signal is generated, causing the interrupt flag to be set in the TMS320C17.

4. The TMS320C17 responds to this interrupt condition in exactly the same way as the present code does, by executing the interrupt handler and executing an IN instruction (from port 5 in this case).

Transfers from the TMS320C17 use the following sequence:

1. The TMS320C17 writes 8 bits of data to the transmit buffer latch (D7-D0) with an OUT instruction to port 5.
2. At some point after this, the $\overline{\text{RD}}$ signal is driven low by the micro-computer using a single I/O bit.
3. Data is driven from the transmit buffer latch (D7-D0) to the LD7-LD0 bus until the RD signal is driven high by the microcomputer.

This interface may be further enhanced by implementing hardware handshaking between the TMS320C17 and the microcomputer, using the RBLE and TBLF signals from the TMS320C17.

Use of DTMF Receiver or Tone Receiver in Isolation

This application report describes an integrated DTMF and tone detection system. Both the DTMF receiver and tone receiver may separately be enabled or disabled (see Mode section), but the code for both is resident at all times. For any application requiring only the DTMF receiver function or only the general-tone function, ROM space can be saved by removing the unwanted code. Due to the complexity of functions such as time-stamping which are shared by both the DTMF receiver and the tone receiver, it is not feasible to describe a complete solution, but some of the major considerations are outlined below.

Note all subsequent page references are to the page number of the listing file given in Appendix A.

The DTMF code section can be removed from the program without significant modification. The DTMF code is very self-contained and is executed as a single block, with few external calls to subroutines within it. The test for the DTMF bit in the mode register should be removed from the end of the routine MAIN (see page 492). Calls to the DTMF reset routine RSDTMF should be removed from the cold reset routine (CRESET on page 514) and the self-test routine (SLFTST on page 516). The DTMF routine may then be removed completely (pages 504 to 510). DTMF constants may be removed, and the data memory locations they were loaded into used for other purposes. Care should be taken to ensure that the correct initialization of locations required by the tone receiver is not disturbed. The section in the warm reset routine (WRESET on page 514) which initializes DTMF data memory locations in page 1 should also be removed.

The tone receiver section is far more complex and cannot be removed as easily. Because the DTMF receiver relies upon certain of the ancillary functions of the tone receiver, these must be left intact. The routines which can be removed are:

FILTER	(pages 496 to 499)
CHNGS	(page 501)
LVLS	(pages 501 and 502)
COMPLT	(page 502)
RSTFIL	(pages 502 and 503)
SQRT	(pages 503 and 504)

Associated data memory locations and initialisation values may also be removed. Care should be taken to check all remaining sections of the code for references to code or memory locations which have been removed. This applies particularly to the following routines:

CRESET	(pages 514)
WRESET	(pages 514 and 515)
SLFTST	(pages 516 and 517)
INTHDL	(pages 510 to 513)
ENVDET	(pages 494 and 495)

It is recommended that these changes are not attempted without an in-circuit emulator for the TMS320C17. This can be used to trace program execution and, with its powerful hardware breakpoint facilities can readily debug the modified source code.

For anyone who wishes to investigate the possibility of customizing the code presented here and does not feel capable of taking on the development work, there is a U.K. company who may be willing to help on a consultancy basis:

Ensigma Ltd.
Archway House
Welsh Street
Chepstow
Gwent
NP6 5LL
Wales

Contacts:
Dr. Mike Carey
Adrian Anderson

Telephone: (44) 291 625422 (International)
0291 625422 (Within U.K.)

Flexibility Through Programmability

Due to the programmability of the tone detector, this solution is not bound by the constraints of a custom hardware solution. Although the DTMF decoder performance is targeted to the CEPT recommendations, the tone receiver is dynamically re-programmable to suit a wide variety of incoming tones across a range of applications.

A simple tone detection system comprising no more than four chips may thus be controlled by a PC or a single chip 8-bit microcontroller to perform any of the tasks described by merely re-programming the on-chip registers of the TMS320C17.

Conclusion

This report has presented a high-functionality DTMF and general tone decoder. The application as described has been fully tested and incorporated into a commercially available telephony peripheral.

Information has been presented which allows a designer to incorporate the tone detector function into a product. A full source listing is included in this report for customization. Performance characteristics for any customized version may vary from those given here.

The objective has been to describe both a particular implementation of the tone detector and provide a level of insight for further development. In order to keep this last part as simple as possible the mathematical detail has been kept to a minimum. If a detailed explanation of this aspect is required Ensigma Ltd. should be approached (see Use of DTMF Receiver or Tone Receiver in Isolation section).

References

[1] *TMS320 First Generation Digital Signal Processors Data Sheet*, Texas Instruments Incorporated. Literature # SPRS009A, January 1987.

[2] *Understanding Telephone Electronics*, W.Sams Inc.

[3] *Technical Reference - Personal Computer AT*, International Business Machines Corporation, September 1985.

[4] *Technical Reference - Personal Computer XT*, revised edition, International Business Machines Corporation, March 1986.

[5] *Telecom Circuits Data Book*, Literature # SCTD001A, Texas Instruments Incorporated, 1987.

[6] *First Generation TMS320 User's Guide*, Literature # SPRT013A, Texas Instruments Incorporated, 1987.

Appendix A
Tone Detector Source Code

```
***************************************************************************
*
*    TMS320C17 SOURCE CODE FOR TONE DETECTOR MODULE
*
*    COPYRIGHT (c) TEXAS INSTRUMENTS June 1986, May 1987
*
*    WRITTEN BY ENSIGMA LTD.
*
*    REVISION 2.08 NOV 1988
*
***************************************************************************
*
*    ASSEMBLER EQUATES FOR TONE DETECTOR
*
***************************************************************************
*
*    PORT DEFINITIONS
*
CTLPRT    .set      0               ; CONTROL PORT
CTLUPR    .set      1               ; UPPER CONTROL PORT
CDCPRT    .set      1               ; CODEC PORT
ATTPRT    .set      2               ; STATUS ATTENTION PORT
PRMPRT    .set      3               ; SIMULATOR INPUT PORT
DATPRT    .set      4               ; DATA PORT
STAPRT    .set      6               ; STATUS READ PORT
SIMPRT    .set      7               ; SIMULATOR FLAG PORT
*
*    FLAG POSITIONS IN FLAGS REGISTER
*
TPRFLG    .set      15              ; TONE PRESENT
ONSFLG    .set      14              ; ONSET TIME VALID
FSTFLG    .set      13              ; FIRST BLOCK OF FILTERING
STAFLG    .set      12              ; INTERRUPT HANDLER STATE BIT
DTMFLG    .set      11              ; DTMF ON FLAG
INTFLG    .set      10              ; INTERRUPT HAS OCCURRED
*
RES1      .set      7               ; RESERVED
RES2      .set      6               ; RESERVED
FILT1     .set      5               ; LEVEL 1 ABOVE CHANGE THR
FILT2     .set      4               ; LEVEL 2 ABOVE CHANGE THR
FILT3     .set      3               ; LEVEL 3 ABOVE CHANGE THR
FILT4     .set      2               ; LEVEL 4 ABOVE CHANGE THR
FILT5     .set      1               ; LEVEL 5 ABOVE CHANGE THR
FILT6     .set      0               ; LEVEL 6 ABOVE CHANGE THR
*
PAGE1     .set      080h            ; ADDRESS OFFSET FOR DATA PAGE 1
*
CPCBIT    .set      8               ; PORT 1 CONTROL BIT
CEFBIT    .set      9               ; EXTERNAL FRAMING BIT
CXFBIT    .set      10              ; XF OUTPUT LATCH BIT
CSPBIT    .set      11              ; SERIAL PORT ENABLE BIT
CEEBIT    .set      12              ; COMPANDER ENCODE ENABLE
CDEBIT    .set      13              ; COMPANDER DECODE ENABLE
CUABIT    .set      14              ; F-LAW, A-LAW SELECT BIT
CSCBIT    .set      15              ; SERIAL CLOCK CONTROL BIT
*
*    MODE REGISTER BITS
*
RC0       .set      0               ; SELF TEST RESULT
RC1       .set      1               ; SELF TEST RESULT
*         .set      2               ; RESERVED
*         .set      3               ; RESERVED
*         .set      4               ; RESERVED
TONEBT    .set      5               ; TONE DETECTOR ON/OFF BIT
DTMFBT    .set      6               ; DTMF DETECTOR ON/OFF BIT
TESTBT    .set      7               ; SELFTEST ONLY ON/OFF BIT
*
*    STATUS REGISTER BITS
*
RDRDY     .set      8               ; NA (interface only)
WRRDY     .set      9               ; NA (interface only)
DPINBT    .set      10              ; Tone Depart interrupt
OSINBT    .set      11              ; Tone Onset interrupt
CHINBT    .set      12              ; Tone Change interrupt
DTINBT    .set      13              ; DTMF Digit interrupt
STINBT    .set      14              ; Short Tone interrupt
*         .set      15              ; Reserved
*
*    EQUATES FOR SELF TESTS
*
ROMFAI    .set      3
RAMFAI    .set      2
PASS      .set      1
CDCFAI    .set      0
ACCHLD    .set      0
TOTAL     .set      1
ROMVAL    .set      2
*
*    (THE REGISTER MAPPING TABLE IS USED TO CONVERT INTERFACE COMMANDS TO
*    TMS320 DATA ADDRESSES.) FLAGS USED IN REGISTER MAPPING TABLE TO INDICATE
*    ACCESSES REQUIRING SPECIAL PROCESSING. THE FUNCTION OF EACH BIT IS
*    DESCRIBED BELOW.
*
L         .set      512
U         .set      1024
T         .set      2048
F         .set      4096
S         .set      8192
M         .set      16384
*
*    SHIFTS FOR TESTING THE REGISTER MAPPING BITS
*
RWBIT     .set      4               ; READ ACCESS OF REGISTER
LBIT      .set      9               ; ACCESS OF ADDRESS 0 OR 1
UBIT      .set      10              ; ACCESS OF AN UPPER BYTE
TBIT      .set      11              ; READ OF CURRENT TIME
FBIT      .set      12              ; WRITE OF FREQUENCY LOWER BYTE
SBIT      .set      13              ; READ OF STATUS REGISTER
```

```
MBIT     .set     14              ; WRITE TO MODE REGISTER
*
*   ALLOW BETWEEN 515 AND 715 IDLE CYCLES BETWEEN CODEC INTERRUPTS
*
INTMAX   .set     077h            ; MAXIMUM AND MINIMUM NUMBER OF VALID
INTMIN   .set     056h            ; TRANSITS OF LOOPB IF CODEC IS INTER-
*                                 ; RUPTING PROPERLY
*
*   LOOP LENGTH IS 6 CYCLES
*
INTLFT   .set     (INTMAX-INTMIN) ; MINIMUM NUMBER OF REMAINING TRANSITS
*
######################################################################
*
*   ASSEMBLER EQUATES FOR DTMF PROGRAM
*
######################################################################
*
SD       .set     06A0h           ; SCALING FACTOR FOR INPUT
Z        .set     0380h           ; SCALING FACTOR FOR THRESHOLDS
SCNT     .set     120             ; MINIMUM SAMPLE COUNT (30MS)
MINTH    .set     0DAh            ; MINIMUM SIGNAL LEVEL
TWSTLO   .set     640             ; 2.5 * 2**8
TWSTHI   .set     2048            ; 8 * 2**8
THRHL1   .set     034h            ; THRESHOLD COUNT FOR 2ND ORDER
THRHL2   .set     02Dh
THRHL3   .set     02Bh
THRHL4   .set     02Dh
THRHH1   .set     029h
THRHH2   .set     027h
THRHH3   .set     024h
THRHH4   .set     022h
LOLIM    .set     14              ; MAX OVERSPILL LO BAND
HILIM    .set     7               ; MAX OVERSPILL HI BAND
MIN      .set     7               ; IDLE LINE DETECT
*
######################################################################
*
*   PAGE 0 DATA DEFINITIONS FOR TONE DETECTOR
*
######################################################################
*
*   UNINITIALIZED VARIABLES FOR TONE DETECTOR
*
######################################################################
         .bss     DUM,033h        ; DUMMY RAM LOCATIONS
         .bss     ONE,1           ; UNITY
         .bss     TEMP,1          ; SCRATCH LOCATION
         .bss     TEMP1,1         ; SCRATCH LOCATION
         .bss     TEMP2,1         ; SCRATCH LOCATION
         .bss     SAMPLE,1        ; CURRENT LINEARIZED INPUT SAMPLE SHARED
*                                 ; WITH DTMF
*
*   THE FOLLOWING LOCATION MUST BE AT AN ADDRESS ENDING IN 8
```

```
*
         .bss     QUEUE,8         ; CIRCULAR QUEUE FOR LINEARIZED INPUT
*                                 ; SAMPLES
         .bss     TEMP3,1         ; SCRATCH LOCATION
         .bss     ITEMP,1         ; INTERRUPT HANDLER SCRATCH LOCATION
         .bss     CMSAVE,1        ; INTERRUPT HANDLER COMMAND BYTE SAVE
         .bss     ARSAVE,1        ; INTERRUPT HANDLER AUXILIARY REGISTER 0
*                                 ; SAVE
         .bss     WINDOW,1        ; WINDOW SAMPLE READ FROM ROM TABLE. 2**15
*                                 ; FORM
         .bss     SNCWIN,1        ; SIN(X)/X . WINDOW PRODUCT. 2**15 FORM
         .bss     FILPOS,1        ; CURRENT BLOCK FILTERING POSITION [COUNTS
*                                 ; UP FROM -16384 TO +16384 IN STEPS OF
*                                 ; (2FL + 32)]
         .bss     ACWNHI,1        ; HIGH WORD OF 32BIT WINDOW ACCUMULATORS
         .bss     ACWNLO,1        ; LOW WORD OF 32BIT WINDOW ACCUMULATORS
         .bss     ACSWHI,1        ; HIGH WORD OF 32BIT SIN(X)/X. WINDOW
*                                 ; PRODUCT ACCUMULATORS
         .bss     ACSWLO,1        ; LOW WORD OF 32BIT SIN(X)/X. WINDOW
*                                 ; PRODUCT ACCUMULATORS
*
         .bss   ACSQHI,1          ; HIGH WORD OF 32BIT SIGNAL SQUARED
*                                 ; ACCUMULATORS
         .bss   ACSQLO,1          ; LOW WORD OF 32BIT SIGNAL SQUARED
*                                 ; ACCUMULATORS
*
######################################################################
*
*   INITIALIZED VARIABLES FOR TONE DETECTOR
*
######################################################################
*
         .bss     IVAR1,0
         .bss     CTL320,1        ; INITIAL VALUE FOR TMS320 CONTROL REGISTER
*                                 ; (LOWER)
         .bss     CTL322,1        ; SECOND VALUE FOR TMS320 CONTROL REGISTER
*                                 ; (LOWER)
         .bss     CTL32U,1        ; VALUE FOR UPPER CONTROL PORT
         .bss     QIN,1           ; POINTS TO NEXT FREE QUEUE INPUT LOCATION
         .bss     QOUT,1          ; POINTS TO NEXT AVAILABLE LINEARIZED
*                                 ; SAMPLE ON QUEUE
         .bss     CORREC,1        ; CORRECTION FACTOR FOR SINE AND COS =
*                                 ; 1/0.9050 * 2**12
         .bss     K1,1            ; CONSTANT USED IN THE SINE COSINE ROUTINE.
         .bss     SCALEF,1        ; SCALE FACTOR FOR SCALING A-LAW LINEARIZED
*                                 ; INPUT SAMPLE INTO AN OPTIMAL NUMBER
*                                 ; RANGE. THE INTERNAL SAMPLE HAS THE VALUE
*                                 ; (SCALEF * 2.25 * LINEARIZED(ALAW) /4).
*                                 ; THE STARTUP VALUE FOR SCALEF IS 4, BUT
*                                 ; TWO OF THE BITS IN THE MODE REGISTER MAY
*                                 ; BE USED TO SELECT SCALEF = 1,4, OR 16.
*                                 ; SCALEF = 4 GIVES TONE DETECTOR DYNAMIC
*                                 ; RANGE SUCH THAT OUTPUT OF 254 == -10dBm0.
*                                 ; THE OTHER SCALEF VALUES MOVE THE DYNAMIC
*                               ; RANGE BY 12dBm0 IN EACH DIRECTION.
*
        .bss     MSOOFF,1       ; BIT MASK VALUE >OOFF
*
################################################################
*
*   REGISTERS FOR COMMUNICATION WITH INTERFACE
*
################################################################
*
        .bss     STMODE,1       ; STATUS REGISTER (UPPER) AND MODE REGISTER
*                               ; (LOWER)
        .bss     FLSTOR,1       ; LAUUE OF FL SAVED TO PREVENT UPDATE
*                               ; DURING FILTERING
        .bss     UPRTHR,1       ; UPPER ENVELOPE DETECTOR THRESHOLD IN
*                               ; LOWER BYTE
        .bss     LWRTHR,1       ; LOWER ENVELOPE DETECTOR THRESHOLD IN
*                               ; LOWER BYTE
        .bss     EDFCT,1        ; ENVELOPE DECAY FACTOR (UPPER) CHANGE
*                               ; THRESHOLD (LOWER)
        .bss     FMSFL,1        ; UPPER BYTE OF DETECTOR CENTER FREQUENCY
*                               ; IN UPPER, AND FL IN LOWER BYTE. FILTER
*                               ; LENGTH IS (4 * FL + 1)
*
        .bss     FSPW,1         ; FILTER SELECT (UPPER) AND PASSBAND WIDTH
*                               ; (LOWER)
*
*                               ; IEND1-IVAR1 IS THE LENGTH OF THE PAGE 0
*                               ; INITIALIZED VARIABLES SECTION
*
################################################################
*
*   UNINITIALIZED VARIABLES FOR TONE DETECTOR
*
################################################################
*
        .bss     IEND1,0
        .bss     OSTIME,1
*
*   MODULO 65536 AND RELATIVE TO END OF LAST RESET
*
        .bss     DPTIME,1       ; TONE DEPART TIME IN MS
        .bss     CRTIME,1       ; CURRENT TIME IN MS
        .bss     OSHOLD,1       ; ONSET TIME LATCH REGISTER
        .bss     CRHOLD,1       ; CURRENT TIME LATCH REGISTER
*
*   DETECTED SIGNAL LEVELS
*
        .bss     LVL12,1        ; FILTER1 (UPPER) FILTER2 (LOWER)
        .bss     LVL34,1        ; FILTER3 (UPPER) FILTER4 (LOWER)
        .bss     LVL56,1        ; FILTER5 (UPPER) FILTER6 (LOWER)
*
        .bss     ENVEL,1        ; SMOOTHED SIGNAL ENVELOPE
        .bss     CTLTSL,1       ; CONTROL REGISTER (UPPER) AND TOTAL SIGNAL
*                               ; LEVEL (LOWER)
        .bss     FLAGS,1        ; MULTIPLE FLAG REGISTER. ALL FLAGS HIGH
*                               ; ASSERTED.
*                               ; SEE ASSEMBLER EQUATES FOR FLAG
*                               ; DEFINITIONS. UNUSED BITS READ AS ZERO.
*
        .bss     SINCPH,1       ; PHASE FOR SIN(X)/X FUNCTION
*
################################################################
*
*   VARIABLES USED IN FILTER ROUTINE, CONTINUE INTO PAGE 1
*   DO NOT INSERT OR DELETE ANY VARIABLES AFTER THIS POINT
*
################################################################
*
        .bss     FREQ1,1        ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss     PHASE1,1       ; PHASE OF FREQN GENERATOR
        .bss     COS1HI,1       ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss     COS1LO,1       ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss     SIN1HI,1       ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss     SIN1LO,1       ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss     FREQ2,1        ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss     PHASE2,1       ; PHASE OF FREQN GENERATOR
        .bss     COS1H2,1       ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss     COS2LO,1       ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss     SIN2HI,1       ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss     SIN2LO,1       ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss     FREQ3,1        ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss     PHASE3,1       ; PHASE OF FREQN GENERATOR
        .bss     COS3HI,1       ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss     COS3LO,1       ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss     SIN3HI,1       ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss     SIN3LO,1       ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss     FREQ4,1        ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss     PHASE4,1       ; PHASE OF FREQN GENERATOR
        .bss    COS4HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS4LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN4HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
*************************************************************************
*
*   PAGE 1 DATA DEFINITIONS FOR TONE DETECTOR.
*
*************************************************************************
*
*   UNINITIALIZED VARIABLES FOR TONE DETECTOR (CONTINUED)
*
*************************************************************************
*
*   THIS LOCATION MUST REPRESENT THE BOUNDARY BETWEEN PAGE 0 AND PAGE 1
*   I.E., DATA MEMORY LOCATION 0080h
*
*************************************************************************
*
*                               ; CONTINUED VARIABLES USED IN FILTER
*                               ; ROUTINE
        .bss    SIN4LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    FREQ5,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE5,1        ; PHASE OF FREQN GENERATOR
        .bss    COS5HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS5LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN5HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN5LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    FREQ6,1         ; FREQN = (FILTER N CENTER FREQUENCY) /
*                               ; 0.12207
        .bss    PHASE6,1        ; PHASE OF FREQN GENERATOR
        .bss    COS6HI,1        ; HIGH WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    COS6LO,1        ; LOW WORD OF 32BIT COSINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN6HI,1        ; HIGH WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
        .bss    SIN6LO,1        ; LOW WORD OF 32BIT SINE FILTER
*                               ; ACCUMULATOR
*
        .bss    SRSAVE,1        ; INTERRUPT HANDLER STATUS REGISTER SAVE
        .bss    ACCUHI,1        ; HIGH WORD OF INTERRUPT HANDLER
*                               ; ACCUMULATOR SAVE
```

```
        .bss    ACCULO,1        ; LOW WORD OF INTERRUPT HANDLER ACCUMULATOR
*                               ; SAVE
*
*************************************************************************
*
*   UNINITIALIZED VARIABLES OF THE DTMF PROGRAM (PAGE 1)
*
*************************************************************************
*
        .bss    UNITY,1
        .bss    DECIM,1         ; DECIMATION FLAG
        .bss    PAUSE,1         ; WE-ARE-IN-THE-GAP FLAG
        .bss    X1,1            ; NEWEST LINEAR DATA SAMPLE
        .bss    GN1,1           ; HIGHPASS/NOTCH-FILTER
        .bss    GN2,1           ; SAMPLE DELAYS
        .bss    Y5,1            ; HIGHPASS/NOTCH-FILTER OUTPUT
        .bss    GAP,1           ; GAP-MUST-FOLLOW FLAG
        .bss    SIGCNT,1        ; SAMPLE COUNTER
        .bss    STOP,1          ; FLAG FOR AGC/RWIST/TONE DECODE
        .bss    SEMA,1          ; FLAG SEMAPHORE FOR THE DECIMATION
        .bss    X,1             ; NEWEST CODEC SAMPLE
*
*   FILTER DELAY SAMPLES
*
        .bss    L1N1,1
        .bss    L1N2,1
        .bss    L2N1,1
        .bss    L2N2,1
*
*   LOW-BAND 8TH ORDER BANDPASS
*
        .bss    L3N1,1
        .bss    L3N2,1
        .bss    L4N1,1
        .bss    L4N2,1
*
*   LOW-BAND 8TH ORDER FILTER OUTPUT
*
        .bss    LY,1
*
*   LOW-BAND 2ND ORDER SUB-FILTER OUTPUTS
*
        .bss    LY1,1
        .bss    LY2,1
        .bss    LY3,1
        .bss    LY4,1
*
        .bss    F1,1            ; 697 HZ
        .bss    F2,1            ; 770 HZ
        .bss    F3,1            ; 852 HZ
        .bss    F4,1            ; 941 HZ
*
*   FILTER DELAY SAMPLES HIGH-BAND 8TH ORDER BANDPASS
*
```

```
        .bss    H1N1,1
        .bss    H1N2,1
        .bss    H2N1,1
        .bss    H2N2,1
        .bss    H3N1,1
        .bss    H3N2,1
        .bss    H4N1,1
        .bss    H4N2,1
*
*   HIGH-BAND 8TH ORDER FILTER OUTPUT
*
        .bss    HY,1
*
*   HIGH-BAND 2ND ORDER SUB-FILTER OUTPUTS
*
        .bss    HY1,1
        .bss    HY2,1
        .bss    HY3,1
        .bss    HY4,1
*
        .bss    F5,1            ; 1209 HZ
        .bss    F6,1            ; 1336 HZ
        .bss    F7,1            ; 1477 HZ
        .bss    F8,1            ; 1633 HZ
        .bss    TEMPD,1         ; SCRATCH-PAD REGISTER
        .bss    ADJL,1          ; HIGH-BAND THRESHOLD ADJUST VALUE
        .bss    ADJH,1          ; LOW-BAND THRESHOLD ADJUST VALUE
        .bss    CNTR1,1         ; SCRATCH COUNTER
        .bss    CNTR2,1         ; SCRATCH COUNTER
        .bss    CNTR,1          ; SAMPLE COUNTER FOR GAP SEARCH
        .bss    TESTG,1         ; GOOD DIGITS TEST COUNT
        .bss    TESTB,1         ; BAD DIGITS TEST COUNT
        .bss    TMP,1           ; TEMP REGISTER FOR CODEC SETTING
*
************************************************************************
*
*   INITIALIZED VARIABLES OF THE DTMF PROGRAM (PAGE 1)
*
************************************************************************
*
*   HIGHPASS/NOTCH-FILTER COEFFICIENTS
*
        .bss    IVAR0,0
        .bss    A1,1
        .bss    A2,1
        .bss    B0,1
        .bss    B1,1
        .bss    B2,1
*
*   FILTER COEFFICIENTS LOW-BAND 8TH ORDER BANDPASS
*
        .bss    L1C,1
        .bss    L1D,1
        .bss    L2C,1
```

```
        .bss    L2D,1
        .bss    L3C,1
        .bss    L3D,1
        .bss    L4C,1
        .bss    L4D,1
*
*   FILTER COEFFICIENTS HIGH-BAND 8TH ORDER BANDPASS
*
        .bss    H1C,1
        .bss    H1D,1
        .bss    H2C,1
        .bss    H2D,1
        .bss    H3C,1
        .bss    H3D,1
        .bss    H4C,1
        .bss    H4D,1
*
        .bss    THRSH8,1        ; 8TH ORDER THRESHOLD VALUE
        .bss    THRHI,1         ; 2ND ORDER THRESHOLD HIGH BAND
        .bss    THRLO,1         ; 2ND ORDER THRESHOLD LOW BAND
        .bss    DVLOW,1         ; STROBE LOW MASK
        .bss    DIGIT,1         ; DIGIT OUTPUT REGISTER
        .bss    MINTHH,1        ; ALTERNATIVE MINTH FOR LOW SIGNAL LEVELS
        .bss    MINTHL,1
*
************************************************************************
*
*   INITIALIZED VARIABLE FOR CHECKSUM ROUTINE
*
************************************************************************
*
        .bss    MARKER,1        ; REGISTER TO HOLD PROGRAM END ADDRESS
*
        .bss    IEND0,0
*
************************************************************************
*
*   VECTORS AND CONSTANTS FOR TONE DETECTOR
*
************************************************************************
*
        .text
*
        B       CRESET          ; COLD RESET VECTOR
0       B       INTHDL          ; INTERRUPT VECTOR
*
************************************************************************
*
*   THIS IS THE TABLE OF CONSTANTS WHICH ARE NOT COPIED INTO RAM
*
************************************************************************
*
CHECKS  .word   02CACh          ; ROM CHECKSUM LOCATION.
SXCENT  .word   23291           ; CENTRAL MAXIMUM OF SIN(X)/X FUNCTION
```

```
*
****************************************************************************
*
*   TABLE OF PAGE 1 DTMF CONSTANTS FOR COPYING TO DATA RAM BY RESET HANDLER
*
****************************************************************************
*
CONST0   .set     $
*
         .word    -27801        ; -30307
         .word    -13617        ; -13952
         .word    +20767        ; +18463
         .word    -28968        ; -25764
         .word    +20767        ; +18427
         .word    03880h        ; CENTER:705 BW:8
         .word    0832Dh
         .word    +11402
         .word    -31755        ; COEFFICIENTS FOR LOW BANDPASS
         .word    +7432
         .word    -31755
         .word    +2985
         .word    -31755
         .word    0D5D6h
         .word    085E4h        ; 1219/15
         .word    0C234h
         .word    087CCh        ; 1332/20
         .word    0AACEh        ; COEFFICIENTS FOR HIGH BANDPASS
         .word    087CCh        ; 1482/20
         .word    09853h
         .word    087CCh        ; 1630/20
         .word    07FFFh        ; THRESHOLD FOR 8TH ORDER OUTPUT
         .word    07FFFh        ; THRESHOLD FOR 2ND ORDER HIGH
         .word    07FFFh        ; THRESHOLD FOR 2ND ORDER LOW
         .word    0FF7Fh        ; MASK FOR DATA VALID STROBE
         .word    070h          ; OUTPUT DIGIT (INIT. INVALID)
         .word    MINTH         ; INITIAL VALUE FOR MINTHH
         .word    0             ; INITIAL VAlUE FOR MINTHL
         .word    PRGEND
*
CONEN0   .set     $
*
****************************************************************************
*
*   TABLE OF PAGE 0 TONE DETECTOR CONSTANTS FOR COPYING TO DATA RAM BY RESET
*   HANDLER
*
****************************************************************************
*
CONST1   .set     $
*
         .word    0FD9Fh        ; CTL320
         .word    07C90h        ; CTL322
         .word    0CFEh         ; CTL32U
         .word    QUEUE         ; QIN
```

```
          .word    QUEUE         ; QOUT
*
          .word    011AEh        ; CORREC 0.9050 * 2**12
          .word    059A8h        ; K1     1.4008687 * 2**14
          .word    04h           ; SCALEF 4 (DEFAULTS TO 254==-10dBm0)
          .word    0FFh          ; MSOOFF
*
****************************************************************************
*
*   REGISTERS FOR COMMUNICATION WITH INTERFACE
*
****************************************************************************
*
          .word    0FF00h        ; STMODE
          .word    032h          ; FLSTOR
          .word    0FFh          ; UPRTHR
          .word    0FFh          ; LWRTHR
          .word    07800h        ; EDFCT
          .word    032h          ; FMSFL
          .word    03F00h        ; FSPW
*
CONEN1    .set     $
*
****************************************************************************
*
*   THIS IS THE TABLE OF MAPPINGS BETWEEN THE 16 LOGICAL READ AND WRITE BYTE
*   ADDRESSES IN THE INTERFACE AND THE PHYSICAL WORD LOCATIONS IN THE
*   TMS320C17
*
****************************************************************************
*
INTTAB    .set     $
*
****************************************************************************
*
*   WRITE REGISTERS
*
****************************************************************************
*
*      ;     Physical locations:  Logical locations:
*
          .word    (L+U+CTLTSL)  ; CONTROL
          .word    (L+M+STMODE)  ; MODE
          .word    (U+EDFCT)     ; ENVELOPE DECAY FACTOR
          .word    (UPRTHR)      ; UPPER THRESHOLD
          .word    (LWRTHR)      ; LOWER THRESHOLD
          .word    (FMSFL)       ; FILTER LENGTH SPECIFIER
          .word    (FSPW)        ; PASSBAND WIDTH SPECIFIER
          .word    (EDFCT)       ; CHANGE THRESHOLD
          .word    (U+FMSFL)     ; FREQUENCY MS BYTE
          .word    (F+FREQ1)     ; FILTER1 CENTER FREQUENCY LS BYTE
          .word    (F+FREQ2)     ; FILTER2 CENTER FREQUENCY LS BYTE
          .word    (F+FREQ3)     ; FILTER3 CENTER FREQUENCY LS BYTE
          .word    (F+FREQ4)     ; FILTER4 CENTER FREQUENCY LS BYTE
```

```
        .word     (F+FREQ5)       ; FILTER5 CENTER FREQUENCY LS BYTE
        .word     (F+FREQ6)       ; FILTER6 CENTER FREQUENCY LS BYTE
        .word     (U+FSPW)        ; FILTER WIDTH SELECT
*
##########################################################################
*
*   READ REGISTERS
*
##########################################################################
*
*  ;      Physical locations:     Logical locations:
*
        .word     (L+S+STMODE)    ; STATUS
        .word     (L+STMODE)      ; MODE
        .word     (DIGIT)         ; DTMF DIGIT
        .word     (U+OSTIME)      ; TONE ARRIVAL (MS)
        .word     (OSTIME)        ; TONE ARRIVAL (LS)
        .word     (U+DPTIME)      ; TONE DEPARTURE (MS)
        .word     (DPTIME)        ; TONE DEPARTURE (LS)
        .word     (T+CRTIME)      ; CURRENT TIME (MS)
        .word     (CRHOLD)        ; CURRENT TIME (LS)
        .word     (U+LVL12)       ; FILTER1 SIGNAL LEVEL
        .word     (LVL12)         ; FILTER2 SIGNAL LEVEL
        .word     (U+LVL34)       ; FILTER3 SIGNAL LEVEL
        .word     (LVL34)         ; FILTER4 SIGNAL LEVEL
        .word     (U+LVL56)       ; FILTER5 SIGNAL LEVEL
        .word     (LVL56)         ; FILTER6 SIGNAL LEVEL
        .word     (CTLTSL)        ; TOTAL SIGNAL LEVEL
*
##########################################################################
*
*   LOOKUP TABLE FOR CONVERTING 2 SC MODE BITS INTO A SCALE FACTOR
*
##########################################################################
*
SCATAB    .set      $
*
        .word     4               ; SCALEF=4 (DEFAULT)
        .word     4               ; SCALEF=4        -10dBm0 RANGE
        .word     1               ; SCALEF=1         +2dBm0 RANGE
        .word     16              ; SCALEF=16       -22dBm0 RANGE
*
##########################################################################
*
*   LOOKUP TABLE FOR CONVERTING 3 IACK MODE BITS INTO THE EQUIVAIENT STATUS
*   BIT TO BE SET BY AN ACKNOWLEDGE.
*
##########################################################################
*
ACKTAB    .set      $
*
        .word     0               ; IACK = 0
        .word     0               ; IACK = 1
        .word     1024            ; IACK = 2
```

```
          .word     2048          ; IACK = 3
          .word     4096          ; IACK = 4
          .word     8192          ; IACK = 5
          .word     16384         ; IACK = 6
          .word     0             ; IACK = 7
*
##########################################################################
*
*   THIS IS THE TABLE OF WINDOW COEFFICIENTS. ONLY HALF OF THE WINDOW IS
*   STORED, STARTING IN THE MIDDLE.
*
##########################################################################
*
WINTAB    .word     32767         ; CENTRAL COEFFICIENT.
          .word     32763
          .word     32749         ; LENGTH OF HALF-WINDOW = 129
          .word     32727
          .word     32696
          .word     32655
          .word     32607
          .word     32549
          .word     32482
          .word     32407
          .word     32323
          .word     32230
          .word     32129
          .word     32019
          .word     31901
          .word     31774
          .word     31639
          .word     31496
          .word     31345
          .word     31186
          .word     31019
          .word     30844
          .word     30662
          .word     30472
          .word     30274
          .word     30069
          .word     29857
          .word     29638
          .word     29412
          .word     29180
          .word     28940
          .word     28694
          .word     28442
          .word     28184
          .word     27920
          .word     27650
          .word     27374
          .word     27093
          .word     26807
          .word     26515
          .word     26218
```

```
        .word   25917
        .word   25611
        .word   25301
        .word   24987
        .word   24668
        .word   24346
        .word   24020
        .word   23691
        .word   23358
        .word   23022
        .word   22684
        .word   22343
        .word   21999
        .word   21654
        .word   21306
        .word   20956
        .word   20605
        .word   20252
        .word   19898
        .word   19543
        .word   19187
        .word   18830
        .word   18473
        .word   18115
        .word   17758
        .word   17400
        .word   17043
        .word   16686
        .word   16330
        .word   15975
        .word   15621
        .word   15268
        .word   14916
        .word   14566
        .word   14218
        .word   13871
        .word   13526
        .word   13184
        .word   12844
        .word   12506
        .word   12172
        .word   11839
        .word   11510
        .word   11184
        .word   10861
        .word   10541
        .word   10225
        .word   9913
        .word   9604
        .word   9299
        .word   8998
        .word   8701
        .word   8408
        .word   8119
        .word   7834
        .word   7554
        .word   7279
        .word   7008
        .word   6741
        .word   6480
        .word   6223
        .word   5971
        .word   5723
        .word   5481
        .word   5244
        .word   5012
        .word   4784
        .word   4562
        .word   4345
        .word   4134
        .word   3927
        .word   3725
        .word   3529
        .word   3338
        .word   3152
        .word   2972
        .word   2796
        .word   2626
        .word   2461
        .word   2301
        .word   2146
        .word   1996
        .word   1851
        .word   1712
        .word   1577
        .word   1448
        .word   1323
        .word   1203
*
ENDWIN  .set    $               ; END OF TABLE OF WINDOW COEFFICIENTS
*
******************************************************************************
*
*  ROUTINE: MAIN
*
*  REFERENCE IN FLOWCHART: READ QUEUE
*                          INCREMENT TIME
*                          SCALE
*                          DTMF
*
*  FUNCTION: READ SAMPLE FROM INPUT QUEUE, AND UPDATE CURRENT TIME. SCALE
*            THE SAMPLE AND CALL DTMF IF IT IS SWITCHED ON.
*
******************************************************************************
*
MAIN    .set    $               ; READ NEXT SAMPLE FROM QUEUE INTO SAMPLE.
*                               ; INCREMENT CURRENT TIME EVERY MS.
*
```

```
READQ   LAC     QIN          ; WAIT FOR SOMETHING ON QUEUE. THE QUEUE IS
        SUB     QOUT         ; EMPTY WHEN THE QUEUE INPUT POINTER EQUALS
        BZ      READQ        ; THE QUEUE OUTPUT POINTER.
*
QNEMPT  LAR     AR0,QOUT     ; LOAD AR0 WITH QUEUE OUTPUT POINTER
*
        LAC     *,0          ; READ SAMPLE FROM QUEUE
*
        BGEZ    POSSMP

*
        ADD     ONE,15       ; CONVERT FROM SIGNED-MAGNITUDE NEGATIVE
        SACL    SAMPLE       ; TO TWOS-COMPLEMENT NEGATIVE
        SUB     SAMPLE,1
POSSMP  SACL    SAMPLE       ; STORE IN SAMPLE
*
        LAC     QOUT
        SUB     ONE          ; DECREMENT OUTPUT POINTER MODULO 8
        SACL    TEMP
*
        LAC     ONE,3
        OR      TEMP         ; POINTER COUNTS 002Fh THROUGH 0028h WHERE
        SACL    QOUT         ; THE QUEUE STARTS AT 0028h
*
        LACK    QUEUE
        SUB     QOUT         ; EVERY TIME THE QUEUE OUTPUT POINTER GETS
        BNZ     SCALE        ; TO 0028h, ONE MS HAS ELAPSED.
*
INCR    ZALS    CRTIME       ; INCREMENT CURRENT TIME BY ONE,
*                            ; MODULO 65536
        ADD     ONE
        SACL    CRTIME
*
****************************************************************************
*
*  SCALE SAMPLE INTO WORKING RANGE. THE WORKING RANGE IS SET SO THAT NONE OF
*  THE ACCUMULATORS IN THE TONE DETECTOR WILL OVERFLOW UNDER ANY SIGNAL
*  CONDITIONS. THE PEAK-TO-PEAK SINUSOIDAL SWITCH WHICH CAUSES A FULL SCALE
*  (254) READING IN THE TOTAL SIGNAL OUTPUT REGISTER, HAS AN INTERNAL
*  AMPLITUDE OF 2030. WHEN THE DEFAULT FACTOR OF 4 IS SELECTED, 2030
*  CORRESPONDS TO AN INPUT SIGNAL LEVEL OF -10 dBm0. THE OTHER POSSIBLE
*  SCALE FACTORS SHIFT THIS VALUE BY 12dB EITHER WAY. SOFTWARE LIMITING OF
*  THE INTERNAL SIGNAL LEVEL OCCURS AT q 8191, WHICH, FOR THE DEFAULT SCALE
*  FACTOR, CORRESPONDS TO A SIGNAL LEVEL OF +2.1 dBm0. THIS LEVEL IS ALSO
*  SHIFTED BY 12 dB EITHER WAY BY SELECTING THE OTHER SCALE FACTORS, HOWEVER
*  THE CODEC WILL CLIP ANY SIGNALS LARGER THAN +3 dBm0.
*
****************************************************************************
*
SCALE   .set    $
*
        ZALH    SAMPLE       ; LOAD SAMPLE INTO HIGH ACCUMULATOR
        ADD     SAMPLE,13
        SACH    SAMPLE,1     ; 2.25 * SAMPLE IN SAMPLE
```

```
*
        LT      SAMPLE
        MPY     SCALEF       ; MULTIPLY SAMPLE BY SCALE FACTOR
                             ; (1,4 OR 16)
*
        PAC
        SACH    TEMP2        ; SAVE SIGN
        SACL    TEMP1
*
        ABS                  ; CHECK FOR OVERFLOW OF THE <4 * 8192
        SUB     ONE,15       ; LIMIT BY SUBTRACTING 32768
        BLZ     SIZOK
*
OVRLOD  LAC     TEMP2
        SACH    TEMP2        ; EXTRACT PURE SIGN OF SAMPLE
*
        LAC     ONE,15       ; LOAD UP 32768
        SUB     ONE          ; 32767
        XOR     TEMP2        ; 32767 * SIGN(SAMPLE)
        SACL    TEMP1
*
SIZOK   LAC     TEMP1,14
        SACH    SAMPLE       ; DIVIDE BY 4, MAX VALUE IS 8191 OR -8192
*
****************************************************************************
*
*  CHECK WHETHER DTMF IS SWITCHED ON. DTMF IS SWITCHED ON WHEN THE
*  APPROPRIATE BIT IN THE FLAGS REGISTER IS SET. THIS BIT IS COPIED FROM THE
*  MODE REGISTER EVERY TIME THE RESET-FILTERING ROUTINE IS CALLED, WHICH MAY
*  BE DETERMINED FROM THE FLOWCHART. THIS ENSURES THAT THE ON/OFF STATUS OF
*  DTMF CANNOT CHANGE IN THE MIDDLE OF A FILTERING BLOCK. (DTMF AFFECTS THE
*  NUMBER OF FILTERS USED)
*
****************************************************************************
*
        LAC     ONE,DTMFLG
        AND     FLAGS
        BZ      FILCHK       ; IF DTMF IS OFF, BRANCH AROUNF THE CALL
*                            ; TO IT
*
        CALL    DTMF         ; CALL THE DTMF ROUTINE.
*
FILCHK  .set    $
*
        LAC     FILPOS
        SUB     ONE,14       ; CHECK FOR END OF FILTERING, FILTER
        BGZ     LEVCAL       ; POSITION GREATER THAN 16384.
*
****************************************************************************
*
*  IF FILTER POSITION HAS INCREMENTED PAST 16384, THEN A BLOCK OF FILTERING
*  HAS BEEN COMPLETED AND PROGRAM FLOW BRANCHES TO LEVEL CALCULATION, OTHER-
*  WISE IT CONTINUES WITH THE ENVELOPE DETECTOR.
*
****************************************************************************
```

```
*
######################################################################
*
*   ROUTINE: ENVDET
*
*   REFERENCE IN FLOWCHART: POWER DETECTOR
*
*   FUNCTION: DETECT CHANGES IN SIGNAL ENVELOPE RELATIVE TO THE USER-
*             PROGRAMMED UPPER AND LOWER THRESHOLDS. ENVELOPE DETECTOR ALWAYS
*             RUNS, REGARDLESS OF WHETHER TONE DETECTION IS ENABLED. THE
*             ENVELOPE DETECTOR IS USED FOR TIMESTAMPING.
*
######################################################################
*
ENVDET   .set     $
*
         LAC      EDFCT,8
         SACH     TEMP
         LAC      TEMP
         AND      MS00FF         ; EXTRACT EDF FROM EDFCT
*
         SACL     TEMP
         LAC      TEMP,5
         SACL     TEMP           ; 32 * EDF IN TEMP
*
*   THIS OPERATION IMPLEMENTS A SMOOTHING FILTER OF THE FORM:
*
*       ENVEL = (2**15 * ENVEL) + ABS(32EDF * SAMPLE) - (32EDF * ENVEL)
*               -------------------------------------------------------
*                                    2**15
*
*   WHICH IS THE SAME AS
*
*       ENVEL = ((1 - K) * ENVEL) + (K * ABS(SAMPLE))
*
*   WHERE EDF IS K * 2**10, K POSITIVE
*
######################################################################
*
         LT       TEMP
         MPY      SAMPLE
         PAC
         ABS
         MPY      ENVEL
         SPAC
         ADD      ENVEL,15
         ADD      ONE,14
         SACH     ENVEL,1
*
######################################################################
*
*   THE NEXT PIECE OF CODE CHECKS THE ENVELOPE LEVEL AGAINST THE UPPER OR
*   LOWER THRESHOLD, ACCORDING TO THE STATE OF THE SIGNAL PRESENT FLAG.
*
```

```
######################################################################
*
         LAC      ONE,TPRFLG     ; LOAD A 1 IN TONE PRESENT FLAG POSITION
         AND      FLAGS          ; AND WITH THE FLAG REGISTER
         BGZ      TPRSNT         ; BRANCH TO TONE PRESENT IF IT IS SET
*
NOTONE   LAC      UPRTHR,2       ; WE WANT TO COMPARE WITH
*                                ; (8 * 2/c * UPRTHR)
         ADD      UPRTHR         ; WHICH IS VERY NEARLY FIVE * UPRTHR
*
         SUB      ENVEL          ; BRANCH TO NOSIG IF ENVEL IS THE UPPER
         BGEZ     NOSIG          ; THRESHOLD.
*
         LAC      LWRTHR,2       ; WE ALSO COMPARE WITH (8 * 2/C * LWRTHR)
*                                ; BECAUSE THE HIGHER OF THE TWO THRESHOLDS
*                                ; IS TAKEN AS THE UPPER ONE.
         ADD      LWRTHR
*
         SUB      ENVEL          ; DROP THROUGH TO NOSIG IF ENVEL IS THE
         BLZ      TONSET         ; LOWER THRESHOLD.
*
NOSIG    .set     $
*
         CALL     RSTFIL         ; RESET FILTERING TO INCORPORATE ANY
*                                ; CHANGED PARAMETERS
*
         B        MAIN           ; RETURN TO START.
*
######################################################################
*
*   TONE PRESENT FLAG WAS SET.
*
######################################################################
*
TPRSNT   .set     $
*
         LAC      LWRTHR,2       ; WE WANT TO COMPARE WITH
*                                ; (8 * 2/c * LWRTHR)
         ADD      LWRTHR         ; WHICH IS VERY NEARLY FIVE * LWRTHR
*
         SUB      ENVEL          ; BRANCH TO TDEPT IF ENVEL LOWER THAN THE
         BGZ      TDEPT          ; THRESHOLD.
*
*                                ; OTHERWISE BRANCH TO TONCHK, AND IF THE
*                                ; TONE DETECTOR IS ON, THEN BRANCH TO THE
*                                ; FILTER ROUTINE.
*
*
         B        TONCHK
*
######################################################################
*
*   ROUTINE: TONSET
*
*   REFERENCE IN FLOWCHART: SET TONE PRESENT FLAG
*                      HOLD ONSET TIME
*
*   FUNCTION: HANDLE OCCURANCE OF TONE ONSET
*
**********************************************************************
*
TONSET   LAC      ONE,TPRFLG     ; LOAD A 1 IN TONE PRESENT FLAG POSITION
         OR       FLAGS
         SACL     FLAGS          ; THIS HAS SET THE TONE PRESENT FLAG
*
         LAC      CRTIME
         SACL     OSHOLD         ; SAVE ONSET TIME OF DETECTED SIGNAL IN
*                                ; ONSET TIME LATCH REGISTER
*
**********************************************************************
*
*   CHECK THAT THE TONE DETECTOR IS SWITCHED ON. IF IT IS NOT, THEN RESET THE
*   FILTER AND RETURN TO THE BEGINNING. IF IT IS, THEN BRANCH TO THE FILTER-
*   ING ROUTINE.
*
**********************************************************************
*
TONCHK   .set     $
*
         LAC      ONE,TONEBT
         AND      STMODE
         BGZ      FILTER         ; IF TONE DETECTOR IS ON, BRANCH TO FILTER
*                                ; ROUTINE
*
**********************************************************************
*
*   IF AT THE END OF THE ROUTINE A BLOCK OF FILTERING HAS BEEN COMPLETED,
*   PROGRAM FLOW BRANCHES TO LEVELS, OTHERWISE IT BRANCHES TO MAIN
*
**********************************************************************
*
TONOFF   B        CONT3          ; RESET THE FILTERS READY FOR WHEN IT IS
*                                ; SWITCHED ON AGAIN.
*
**********************************************************************
*
*   ROUTINE: TDEPT
*
*   REFERENCE IN FLOWCHART: CLEAR TONE PRESENT FLAG
*                           RESET FILTERING
*                           SET ONSET TIME VALID FLAG
*                           CLEAR ONSET FLAG
*                           SAVE DEPART TIME
*                           SET DEPART INTERRUPT
*                           SET SHORT TONE INTERRUPT
*
*   FUNCTION: HANDLE TONE DEPARTURE
*
**********************************************************************
```

```
*
TDEPT    LAC      ONE,TPRFLG     ; PUT A 1 IN THE TONE PRESENT FLAG POSITION
         XOR      FLAGS
         SACL     FLAGS          ; THIS HAS CLEARED THE TONE PRESENT FLAG.
*
         CALL     RSTFIL         ; CALL RESET FILTERING ROUTINE TO CLEAR
*                                ; DOWN ALL ACCUMULATORS AND SET UP FILTER
*                                ; READY FOR THE NEXT BLOCK.
*
         LAC      ONE,ONSFLG     ; LOAD A 1 IN THE ONSET TIME VALID FLAG
*                                ; POSITION
         AND      FLAGS
         BGZ      OSVAL          ; IF THE FLAG IS SET, BRANCH TO OSVAL
*
         LAC      ONE,TONEBT     ; ELSE IF THE TONE DETECTOR IS ON, SET A
         AND      STMODE         ; SHORT TONE INTERRUPT.
         BZ       MAIN           ; IF TONE DETECTOR IS OFF, GO HOME.
*
STINT    LAC      ONE,STINBT
         XOR      STMODE
         AND      STMODE         ; ASSERT SHORT TONE INTERRUPT
         SACL     STMODE
*
         CALL     ATTEN          ; WRITE OUT STATUS
*
         CALL     XFUPD          ; UPDATE XF FLAG
*
         B        MAIN
*
OSVAL    XOR      FLAGS          ; CLEAR ONSET TIME VALID FLAG
         SACL     FLAGS
*
         LAC      CRTIME
         SACL     DPTIME         ; TONE DEPARTURE TIME = CURRENT TIME.
*
         LAC      ONE,TONEBT     ; IF TONE DETECTOR IS ON, THEN SET A DEPART
         AND      STMODE         ; INTERRUPT.
         BZ       MAIN           ; ELSE GO HOME.
*
*                                ; SET A TONE DEPARTURE INTERRUPT
DPINT    LAC      ONE,DPINBT
         XOR      STMODE
         AND      STMODE         ; ASSERT DEPART INTERRUPT
         SACL     STMODE
*
         CALL     ATTEN          ; WRITE OUT STATUS
*
         CALL     XFUPD          ; UPDATE XF FLAG
*
         B        MAIN
*
**********************************************************************
*
*   ROUTINE: FILTER
```

```
*
*   REFERENCE IN FLOWCHART: FILTER
*
*   FUNCTION: ROUTINE FOR FILTERING AND ACCUMULATING THE INPUT SAMPLE
*
**********************************************************************
*
FILTER   .set     $
*
**********************************************************************
*
*   THE FIRST SECTION CALCULATES A SAMPLE OF SIN(X)/X. THE PHASE X IS STORED
*   IN 2**-5 FORM TO AVOID POSSIBLE OVERFLOW, AND THE RESULTING SAMPLE IS IN
*   2**14 FORM, MAXIMUM VALUE C/2 * 2**14. THE DIVISION ASSUMES NUMERATOR IN
*   2**14 FORM, HOWEVER THE SINE SECTION PRODUCES A 2**13 RESULT, AND THE
*   DENOMINATOR IS ACTUALLY IN 2**-5 FORM, THUS THE NUMERATOR IS SHIFTED LEFT
*   BY 10 PLACES INSTEAD OF 14.  (14-5+1 = 10)
*
**********************************************************************
*
        LACK     SXCENT          ; LOAD UP CENTRAL VALUE OF SIN(X)/X
        TBLR     TEMP1           ; FUNCTION INTO TEMP1
        LAC      SINCPH,5        ; SIN(X)/X WHERE THE PHASE X = ZERO IS A
        BZ       PHASE0          ; SPECIAL CASE.
*
**********************************************************************
*
*   ACCUMULATOR NOW CONTAINS FULL 2**0 REPRESENTATION OF SINCPHASE. THIS IS
*   TRUNCATED TO THE NORMAL q c REPRESENTATION.
*
**********************************************************************
*
        ADD      ONE,14          ; GENERATE SINE OF SINCPH
        SACL     TEMP
        LAC      TEMP
        ABS
        SUB      ONE,14
        SACL     TEMP
*
        LT       TEMP
        LAC      K1,15
        MPY      TEMP
        SPAC
        SACH     TEMP,1
*
        MPY      TEMP
        PAC
        SACH     TEMP1,1         ; SINE(X) IN TEMP1 IN 2**13 FORM
*
        LT       TEMP1           ; LOAD T WITH SIN(X)
        MPY      SINCPH          ; DUMMY MULTIPLY IN ORDER TO OBTAIN
        PAC                      ; SIGN OF SIN(X)/X QUOTIENT IN TEMP3
        SACH     TEMP3
*
        LAC      SINCPH
        ABS                      ; OBTAIN POSITIVE DIVISOR (PHASE X) IN TEMP
        SACL     TEMP            ; DIVISOR ANGLE STILL IN 2**-5 FORM
*
        LAC      TEMP1,10        ; SINE(X) IN 2**13 FORM * 2**10 IN
*                                ; ACCUMULATOR
        ABS                      ; READY FOR DIVISION. INITIAL RESULT IN
*                                ; 2**8.
*
DIV0    SUBC     TEMP            ; 8 CYCLE DIVIDE LOOP
        NOP
        SUBC     TEMP
        NOP
        SUBC     TEMP
        NOP
        SUBC     TEMP
        NOP
        SUBC     TEMP
        NOP
        SUBC     TEMP
        NOP
        SUBC     TEMP
        NOP
        SUBC     TEMP
        NOP
*
        AND      MSOOFF
        SACL     TEMP1           ; 8 BIT RESULT IN 2**8 FORM
        LAC      TEMP1,8
        ADD      ONE,7           ; ROUND BY ADDING 128
        SACL     TEMP1           ; FINAL RESULT IN 2**14 FORM
*
        LAC      TEMP3
        BGEZ     PHASE0          ; DONE IF REQUIRED SIGN IS POSITIVE
*
        ZAC
        SUB      TEMP1
        SACL     TEMP1           ; OTHERWISE INVERT SIGN OF RESULT
*
PHASE0  .set     $               ; SIN(X)/X SAMPLE IN TEMP1
*
**********************************************************************
*
*   NOW CALCULATE WINDOW POSITION AND READ WINDOW SAMPLE FROM ROM TABLE
*
**********************************************************************
*
        LAC      FILPOS,9
        ABS
        SACH     TEMP            ; DIVIDE ABS (FILTERPOSITION) BY 128
*                                ; THE RESULT IS THE OFFSET INTO THE HALF-
*                                ; LENGTH WINDOW TABLE.
        LACK     WINTAB
        ADD      TEMP            ; ADD IN WINDOW TABLE OFFSET
```

```
*
        LAC     ONE,DTMFLG
        AND     FLAGS
        BNZ     FLOOP           ; IF DTMF IS ON, ONLY DO THREE FILTERS,
*
        LARK    AR0,5           ; ELSE DO ALL SIX
*
FLOOP   .set    $               ; FILTERING LOOP
*
        LARP    1
*
************************************************************************
*
*   THIS IS THE FILTERING LOOP. FOR EACH OF THE FILTERS, A SINE AND A COSINE
*   SAMPLE IS GENERATED, AND EACH IS MULTIPLIED BY THE PRODUCT OF THE INPUT
*   SAMPLE AND THE APPROPRIATE FILTER COEFFICIENT, WIDE OR NARROW. THE NARROW
*   FILTER COEFFICIENT IS JUST THE WINDOW SAMPLE WHEREAS THE WIDE FILTER
*   COEFFICIENT IS THE WINDOW SAMPLE MULTIPLIED BY A SIN(X)/X SAMPLE.
*
************************************************************************
*
************************************************************************
*
*   GENERATE SINE AND COSINE SAMPLE AT THE SPECIFIED SEARCH FREQUENCY OF THIS
*   FILTER
*
************************************************************************
*
        LAC     *+              ; LOAD UP FREQN WHICH IS THE REQUIRED PHASE
*                               ; INCREMENT
        ADD     *               ; ADD CURRENT PHASE.
        SACL    *               ; CALCULATE NEW PHASE OF SEARCH FREQUENCY
*
************************************************************************
*
*   SINE AND COSINE ROUTINE. REQUIRES ARGUMENT IN TEMP1 IN REPRESENTATION
*   WHERE q 2**15 REPRESENTS q c. RESULT IS -COS(TEMP1) SCALED IN LOCATION
*   TEMP2 AND SINE(TEMP1) SCALED IN LOCATION TEMP1 USES TEMP AS A SCRATCH
*   LOCATION.
*
************************************************************************
*
*   K1 IS A CONSTANT EQUAL TO 1.4008687 * 2**14. THE OCCURRANCES OF ONE,14
*   REPRESENT C/2. THE RESULT IS A SINE AND A -COSINE SCALED BY 0.9050 *
*   2**13. THE ALGORITHM ACCEPTS ANGLES IN THE RANGE q 2 * 2**14, REPRESENT-
*   ING q c, AND CONVERTS THEM INTO 2 EQUIVALENT ANGLES IN THE RANGE q 1 *
*   2**14, ONE SHIFTED BY C/2.  IT THEN PERFORMS A SINE ALGORITHM ON EACH OF
*   THESE TWO VALUES TO YIELD THE DESIRED RESULTS.
*
************************************************************************
*
SINCOS  .set    $
*
        LAC     *               ; MAKE ANGLE MODULO 65536
```

```
*
        TBLR    WINDOW          ; READ WINDOW SAMPLE
*
        LT      WINDOW
        MPY     TEMP1           ; TAKE WINDOW * SIN(X)/X PRODUCT
        PAC
*
        SACH    SNCWIN,1        ; STORE PRODUCT
*
************************************************************************
*
*   ACCUMULATE WINDOW AND SNCWIN INTO THEIR RESPECTIVE ACCUMULATORS; THESE
*   WILL LATER BE USED TO NORMALIZE THE FILTER OUTPUTS TO COMPENSATE FOR THE
*   EFFECT OF THE WINDOWING AND THE MULTIPLICATION BY A SIN(X)/X FUNCTION.
*
************************************************************************
*
        ZALH    ACSWHI
        ADDS    ACSWLO          ; LOAD UP SNCWIN ACCUMULATOR
*
        ADD     SNCWIN
        SACL    ACSWLO
        SACH    ACSWHI          ; STORE ACCUMULATION BACK.
*
        ZALH    ACWNHI
        ADDS    ACWNLO          ; LOAD UP WINDOW ACCUMULATOR
*
        ADD     WINDOW
        SACL    ACWNLO
        SACH    ACWNHI          ; STORE ACCUMULATION BACK.
*
        LAC     SAMPLE,2
        SACL    TEMP            ; MULTIPLY SCALED LINEARIZED SAMPLE BY FOUR
*
        LT      TEMP
        MPY     SNCWIN          ; GENERATE WIDE FILTER SAMPLE
        PAC
*
        SACH    SNCWIN,1        ; SNCWIN NOW CONTAINS WIDE FILTER SAMPLE
*
        MPY     WINDOW          ; GENERATE NARROW FILTER SAMPLE
        PAC
*
        SACH    WINDOW,1        ; WINDOW NOW CONTAINS NARROW FILTER SAMPLE
*
        LAC     MSOOFF,8
        AND     FSPW            ; MASK IN FILTERSELECT BYTE
*
        SACL    TEMP3
        LAC     TEMP3,2         ; THE SIX FILTERSELECT BITS ARE NOW IN THE
        SACL    TEMP3           ; TOP SIX BITS OF TEMP3
*
        LARK    AR1,FREQ1
        LARK    AR0,2           ; SET UP LOOP COUNTER FOR THE FIRST THREE
        ABS
        SUB     ONE,14
        SACL    TEMP
        LT      TEMP
        LAC     K1,15
        MPY     TEMP
        SPAC
        SACH    TEMP,1
        MPY     TEMP
        PAC
        SACH    TEMP2,1
        LAC     *+
        ADD     ONE,14
        SACL    TEMP
        LAC     TEMP
        ABS
        SUB     ONE,14
        SACL    TEMP
        LT      TEMP
        LAC     K1,15
        MPY     TEMP
        SPAC
        SACH    TEMP,1
        MPY     TEMP
        PAC
        SACH    TEMP1,1
*
************************************************************************
*
*   END OF SINE COSINE ROUTINE
*
************************************************************************
*
************************************************************************
*
*   THIS SECTION GETS EITHER WIDE OR NARROW SAMPLE INTO THE T REGISTER.
*
************************************************************************
*
SELECT  .set    $
*
        LT      SNCWIN          ; SNCWIN CONTAINS THE WIDE SAMPLE
*
        LAC     TEMP3,1         ; PICK OFF THE FILTERSELECT BIT FOR THIS
*                               ; FILTER
        SACL    TEMP3
        BLZ     WIDEF           ; IF THE FILTERSELECT BIT WAS SET, THEN
*                               ; THIS FILTER IS WIDE.
*
NARROF  .set    $               ; THIS FILTER IS NARROW
*
        LT      WINDOW          ; WINDOW CONTAINS THE NARROW SAMPLE
*
************************************************************************
```

```
*
*   END OF SELECTION SECTION
*
************************************************************************
*
WIDEF   .set    $               ; THIS FILTER IS WIDE
*
        MPY     TEMP2
        PAC
        ADD     ONE,14
        SACH    TEMP,1          ; SAMPLE*COSINE / 2**15
*
        LAC     TEMP,2          ; LOAD SAMPLE*COSINE PRODUCT, NOW
*                               ; ACCUMULATE
*
        ADDH    *+              ; COSNHI ADDED
        ADDS    *-              ; COSNLO ADDED
        SACH    *+              ; STORE BACK ACCUMULATOR.
        SACL    *+
        MPY     TEMP1
        PAC
        ADD     ONE,14
        SACH    TEMP,1          ; SAMPLE*SINE / 2**15
        LAC     TEMP,2          ; LOAD SAMPLE*SINE PRODUCT, NOW ACCUMULATE
*
        ADDH    *+              ; SINNHI ADDED
        ADDS    *-              ; SINNLO ADDED
        SACH    *+              ; STORE BACK ACCUMULATOR.
        SACL    *+,0,AR0        ; AR0 NOW POINTING AT FREQN FOR NEXT FILTER
*
        BANZ    FLOOP
*
************************************************************************
*
*   THE NEXT SECTION ACCUMULATES (SAMPLE/4)) SO THAT A MEASURE OF THE TOTAL
*   SIGNAL ENERGY IN THE BAND CAN BE CALCULATED.
*
************************************************************************
*
        LAC     SAMPLE,14       ; SAMPLE/4 IN HIGH ACCUMULATOR
        SACH    TEMP
        LT      TEMP
        MPY     TEMP
        PAC                     ; (SAMPLE/4)) IN ACCUMULATOR
        ADDH    ACSQHI          ; ACCUMULATOR
        ADDS    ACSQLO
        SACH    ACSQHI          ; STORE ACCUMULATOR BACK.
        SACL    ACSQLO
*
************************************************************************
*
*   INCREMENT FILTER POSITION
*
************************************************************************
```

```
*
        LAC     FLSTOR,1        ; CALCULATE FILTERPOSITION INCREMENT AS
        ADD     ONE,5           ; (2FL +32)
*
        ADD     FILPOS
        SACL    FILPOS
*
######################################################################
*
*   INCREMENT SIN(X)/X PHASE
*
######################################################################
*
        LAC     MSOOFF
        AND     FSPW            ; MASK IN PW THE PASSBANDWIDTH, WHICH
*                               ; EQUALS PHASE INCREMENT REQUIRED FOR THE
*                               ; Sin(X)/X PHASE.
        ADD     SINCPH
        SACL    SINCPH
*
        B       MAIN            ; RETURN TO BEGINNING OF MAIN LOOP.
*
######################################################################
*
*   ROUTINE: LEVCAL
*
*   REFERENCE IN FLOWCHART: CALCULATE LEVELS
*
*   FUNCTION: CAICULATES THE LEVELS AT THE END OF EACH BLOCK OF FILTERING
*
######################################################################
*
LEVCAL  .set    $
*
######################################################################
*
*   FIRST CHECK THAT THE QUEUE IS EMPTY. IF IT IS NOT, JUMP BACK TO THE
*   BEGINNING OF THE PROGRAM. DON'T DO ANY OF THIS PROCESSING UNTIL THE QUEUE
*   IS EMPTY; THIS WILL RESTORE THE INTERRUPT TIME OVERHEAD PROVIDED BY
*   HAVING AN EMPTY QUEUE.
*
######################################################################
*
        LAC     QIN             ; THE QUEUE IS EMPTY WHEN THE QUEUE INPUT
        SUB     QOUT            ; POINTER EQUALS THE OUTPUT POINTER.
        BNZ     MAIN
*
######################################################################
*
*   CALCULATE THE TOTAL SIGNAL LEVEL IN THE WHOLE BAND. WE HAVE AN ACCUMU-
*   LATION OF (SAMPLE/4)) IN ACSQRD. THIS IS DIVIDED BY (4*FILTERLENGTH),
*   WHERE FILTER LENGTH IS (16384/(FL + 16)) + 1, AND THEN MULTIPLIED BY 2
*   AND SQUARE-ROOTED.
*
```

```
######################################################################
*
        LAC     FLSTOR
        ADD     ONE,4           ; GIVES FL + 16
        SACL    TEMP            ; AS DIVISOR IN TEMP.
*
        LARK    AR0,15          ; SET UP4AR0 AS 16 CYCLE COUNTER
        ZALH    ONE
*
DLOOP   SUBC    TEMP
        BANZ    DLOOP
*
        ADD     ONE,2
        SACL    TEMP            ; 4 * FILTERLENGTH IN TEMP
*
        ZALH    ACSQHI
        ADDS    ACSQLO
*
        SUB     TEMP,15         ; LIMIT DIVIDEND TO LESS THAN (2**15 *
*                               ; DIVISOR) SO THAT RESULT OF DIVISION WILL
*                               ; WILL BE LESS THAN 2**15
        BLZ     SIZEOK
*
TOOBIG  ZAC
        SUB     ONE
*
SIZEOK  .set    $
*
        ADD     TEMP,15         ; DIVIDEND LIMITED.
*
        LARK    AR0,15          ; SET UP AR0 FOR 16 SUBC'S
*
ALOOP   SUBC    TEMP
        BANZ    ALOOP
*
        SACL    TEMP
        LAC     TEMP,3          ; RESULT NOW IN 2**2 FORM AS REQUIRED BY
*                               ; THE SQUARE ROOT ROUTINE.
        SACH    TEMP1
        SACL    TEMP2
        CALL    SQRT            ; SQUARE ROOT. RESULT IN TEMP3
*
        LAC     MSOOFF,8
        AND     CTLTSL          ; MASK IN CONTROL REGISTER BITS ONLY.
        ADD     TEMP3
        SACL    CTLTSL          ; STORE RESULT IN LOWER HALF OF CTLTSL
*
######################################################################
*
*   DIVIDE THE (WINDOW*SIN(X)/X) WIDE ACCUMULATION ANF THE WINDOW NARROW
*   ACCUMULATION BY 2**15.
*
######################################################################
*
```

```
        ZALH    ACSWHI
        ADDS    ACSWLO
        SACH    ACSWLO,1        ; RESULT IN ACSWLO
*
        ZALH    ACWNHI
        ADDS    ACWNLO
        SACH    ACWNLO,1        ; RESULT IN ACWNLO
*
******************************************************************************
*
*   NOW NORMALIZE THE SINE AND COSINE FILTER ACCUMULATORS BY DIVIDING THEM BY
*   EITHER THE (WINDOW * SIN(X)/X) ACCUMULATION OR THE WINDOW ACCUMULATION
*   DEPENDING ON WHETHER THE FILTER IS WIDE OR NARROW.
*
******************************************************************************
*
        LAC     MS00FF,8
        AND     FSPW            ; MASK IN FILTER SELECT BYTE
*
        SACL    FILPOS
        LAC     FILPOS,2        ; THE SIX FILTER SELECT BITS ARE NOT IN THE
        SACL    FILPOS          ; TOP SIX BITS OF FILPOS
*
        LARK    AR1,FREQ1
        LARK    AR0,2           ; SET UP LOOP COUNTER FOR THE FIRST THREE
*                               ; FILTERS ONLY
        LAC     ONE,DTMFLG
        AND     FLAGS
*
        BNZ     NLOOP           ; IF DTMF IS ON, ONLY DO THREE FILTERS,
*
        LARK    AR0,5           ; ELSE DO ALL SIX
*
NLOOP   .set    $
*
        LARP    1
*
******************************************************************************
*
*   THIS IS THE NORMALIZATION LOOP.
*
*   THIS SECTION GETS EITHER THE WIDE OR NARROW ACCUMULATION INTO TEMP
*
******************************************************************************
*
        LAC     ACSWLO          ; ACSWLO CONTAINS THE WIDE NORMALIZATION
        SACL    TEMP
*
        LAC     FILPOS,1        ; PICK OFF THE FILTERSELECT BIT FOR THIS
*                               ; FILTER
        SACL    FILPOS
        BLZ     WIDEFL          ; IF THE FILTERSELECT BIT WAS SET, THEN
*                               ; THIS FILTER IS WIDE.
*
```

```
NARROW  .set    $               ; THIS FILTER IS NARROW
*
        LAC     ACWNLO          ; ACWNLO CONTAINS THE NARROW NORMALIZATION
        SACL    TEMP
*
WIDEFL  .set    $               ; FIRST NORMALIZE THE COSINE ACCUMULATOR
        LAC     *+
        LAC     *+              ; TWO DUMMY READS TO INCREMENT POINTER TO
*                               ; COSNHI
*
        ZALH    *+
        ADDS    *               ; LOAD UP COSINE FILTERN ACCUMULATOR
        ABS
*
        SAR     AR1,TEMP1       ; SAVE AR1 IN TEMP1
        LARK    AR1,15          ; USE AR1 TO CONTROL A 16 CYCLE DIVIDE.
*
CDIV    SUBC    TEMP            ; DIVIDE BY THE SELECTED ACCUMULATOR
        BANZ    CDIV
*
        LAR     AR1,TEMP1       ; RESTORE AR1
*
        SACL    TEMP2
        LT      TEMP2           ; MULTIPLY RESULT OF DIVISION BY SINE
        MPY     CORREC          ; CORRECTION FACTOR.
        PAC
        SACH    *+,1,AR1        ; STORE 2*RESULT IN COSNLO
*
******************************************************************************
*
*   NOW REPEAT FOR THE SINE ACCUMULATOR
*
******************************************************************************
*
        ZALH    *+
        ADDS    *-              ; LOAD UP SINE FILTERN ACCUMULATOR
        ABS
*
        SAR     AR1,TEMP1       ; SAVE AR1 IN TEMP1
        LARK    AR1,15          ; USE AR1 TO CONTROL A 16 CYCLE DIVIDE.
*
SDIV    SUBC    TEMP
        BANZ    SDIV
*
        LAR     AR1,TEMP1       ; RESTORE AR1
*
        SACL    TEMP2
        LT      TEMP2           ; MULTIPLY RESULT OF DIVISION BY SINE
        MPY     CORREC          ; CORRECTION FACTOR
        PAC
        SACH    *,1,AR1         ; STORE 2*RESULT IN SINnHI
*
        LTC     *
        MPY     *-
        PAC                     ; SQUARE SINE RESULT
*
        LT        *
        MPY       *+            ; SQUARE COS RESULT
        APAC                    ; AND ADD IT IN   -> 4* SUM OF SQUARES
*
        SACH      TEMP1
        SACL      TEMP2
        CALL      SQRT          ; NOTE THAT SQRT ALWAYS RETURNS WITH ARP=0
*
        LARP      1
*
        LAC       *+            ; DUMMY READ TO INCREMENT POINTER TO END OF
*                               ; BLOCK.
        LAC       TEMP3         ; LOAD UP RESULT OF SQUARE ROOT
        SACL      *+,0,AR0      ; STORE RESULT (FILTERN OUTPUT LEVEL) IN
*                               ; SINNLO
*
******************************************************************************
*
*    END OF NORMALIZATION SECTION. REPEAT FOR EACH FILTER.
*
******************************************************************************
*
        BANZ      NLOOP
*
******************************************************************************
*
*    ROUTINE: CHNGS
*
*    REFERENCE IN FLOWCHART: CHECK CHANGES
*
*    FUNCTION: CHECK FOR LEVEL CHANGES DURING A TONEBURST
*
******************************************************************************
*
******************************************************************************
*
*    NOW CHECK FOR CHANGES IN ANY OF THE FILTER LEVELS WHICH CROSS THE CHANGE
*    THRESHOLD. COPY THE FLAGS REGISTER INTO TEMP, ANF USE THAT TO CHECK FOR
*    CHANGES WHILE THE REAL FLAGS REGISTER IS MODIFIED TO REFLECT THE FILTER
*    OUTPUTS WHICH ARE CURRENTLY ABOVE THE CHANGE THRESHOLD.
*
******************************************************************************
*
CHNGS   .set      $
*
        LAC       FLAGS
        SACL      TEMP          ; COPY FLAGS INTO TEMP
*
        LAC       MSOOFF,8
        AND       FLAGS
        SACL      FLAGS         ; CLEAR OUT THE SIX FILTER LEVEL BITS.
*
```

```
        LAC       EDFCT
        AND       MSOOFF        ; LOAD UP CHANGE THRESHOLD
        SACL      TEMP1         ; STORE CT IN TEMP1
*
        LARK      AR0,FREQ1     ; START POINTING AT FREQ1
*
        LAC       ONE,FILT1
        SACL      TEMP2         ; INITIAL BIT TO WORK WITH IN FLAGS
*
CLOOP   MAR       *+
        MAR       *+
        MAR       *+
        MAR       *+
        MAR       *+            ; NOW POINTING AT SINNLO
*
        LAC       TEMP1         ; LOAD UP CHANGE THRESHOLD
        SUB       *+            ; AR1 NOW POINTING AT FREQN+1
*
        BGEZ      CEND          ; DROP TO END OF LOOP IF LEVEL IN THIS BAND
*                               ; IS BELOW THE CHANGE THRESHOLD.
        LAC       TEMP2         ; LOAD UP APPROPRIATE FLAG BIT MASK
        XOR       TEMP
        SACL      TEMP          ; FLIP THE BIT IN TEMP
*
        LAC       TEMP2
        OR        FLAGS
        SACL      FLAGS         ; SET THE BIT IN FLAGS
*
CEND    LAC       TEMP2,15
        SACH      TEMP2         ; SHIFT BIT OF INTEREST ALONG TO THE RIGHT
*
        LAC       TEMP2
        BNZ       CLOOP         ; REPEAT CLOOP FOR EACH FILTER
*
******************************************************************************
*
*    ROUTINE: LVLS
*
*    REFERENCE IN FLOWCHART: WRITE OUT LEVELS
*
*    FUNCTION: WRITE LEVELS INTO REGISTERS
*
******************************************************************************
*
******************************************************************************
*
*    COPY FILTER OUTPUT LEVELS INTO THEIR RESPECTIVE REGISTERS IN COMPRESSED
*    FORMAT, 2 LEVELS TO A WORD.
*
******************************************************************************
*
LVLS    .set      $
*
        LAC       SIN1LO,8
        ADD     SIN2LO
        SACL    LVL12
*
        LAC     SIN3LO,8
        LDPK    1
        ADD     SIN4LO
        LDPK    0
        SACL    LVL34
*
        LDPK    1
        LAC     SIN5LO,8
        ADD     SIN6LO
        LDPK    0
        SACL    LVL56
*
########################################################################
*
*   ROUTINE: COMPLT
*
*   REFERENCE IN FLOWCHART: FIRST BLOCK FLAG SET?
*                           CHANGES?
*                           SET CHANGE INTERRUPT
*                           CLEAR FIRST BLOCK FLAG
*                           SAVE HELD ONSET TIME
*                           SET ONSET TIME VALID FLAG
*                           SET ONSET INTERRUPT
*
*   FUNCTION: COMPLETE OPERATIONS READY FOR NEXT FILTERING OPERATION
*
########################################################################
*
COMPLT  .set    $
*
        LAC     ONE,FSTFLG
        AND     FLAGS
        BNZ     FSTTIM          ; IF IT IS THE FIRST BLOCK, SKIP OVER THE
*                               ; NEXT SECTION
        LAC     TEMP
        AND     MSOOFF
        BZ      CONT3           ; IF ALL THE 6 FLAG BITS ARE ZERO, THERE
*                               ; WAS NO CHANGE IN LEVEL ACROSS THE CHANGE
*                               ; THRESHOLD
*
CHINT   LAC     ONE,CHINBT      ; ELSE THERE WAS A CHANGE
        XOR     STMODE
        AND     STMODE          ; ASSERT CHANGE INTERRUPT
        SACL    STMODE
*
        CALL    ATTEN           ; WRITEOUT STATUSs
*
        CALL    XFUPD           ; UPDATE XF FLAG
*
        B       CONT3
*
```

```
FSTTIM  .set    $
*
        LAC     OSHOLD
        SACL    OSTIME          ; COPY ONSET TIME FROM HOLDING REGISTER
*                               ; INTO ONSET TIME REGISTER
        LAC     ONE,ONSFLG
        OR      FLAGS
        SACL    FLAGS           ; SET THE ONSET TIME VALID FLAG
*
*
########################################################################
*
*   SET AN ONSET INTERRUPT
*
########################################################################
*
OSINT   LAC     ONE,OSINBT
        XOR     STMODE
        AND     STMODE          ; ASSERT ONSET INTERRUPT
        SACL    STMODE
*
        CALL    ATTEN           ; WRITE OUT STATUS
        CALL    XFUPD           ; UPDATE XF FLAG
*
CONT3   .set    $
*
        CALL    RSTFIL          ; RESET THE FILTER
*
        LAC     ONE,FSTFLG
        XOR     FLAGS
        AND     FLAGS
        SACL    FLAGS           ; CLEAR THE FIRST BLOCK FLAG
*
        B       MAIN            ; RETURN TO BEGINNING
*
########################################################################
*
*   ROUTINE: RSTFIL
*
*   REFERENCE IN FLOWCHART: RESET FILTERING
*
*   FUNCTION: CLEAR DOWN FILTER ACCUMULATORS AND RESET POINTERS
*             READY FOR ANOTHER FILTERING OPERATION.
*
########################################################################
*
RSTFIL  .set    $               ; RESET FILTERING ROUTINE.
*
        LAC     MSOOFF          ; LOAD UP LOWER BYTE MASK
        AND     FMSFL           ; MASK IN FL
        SACL    FLSTOR          ; FL IN FLSTOR
*
        ZAC
        SUB     ONE,14
        SACL      FILPOS        ; RESET FILTER POSITION TO -16384
*
######################################################################
*
*   CALCULATE INITIAL SIN(X)/X PHASE
*
*   EXPRESSION FOR THIS IS:
*   (INITIAL FILTERPOSITION * SIN(X)/X INCREMENT)/HALF-FILTERLENGTH
*   WHICH IS THE SAME AS:
*   (-16384 * PW) / (2FL +32)
*
######################################################################
*
        LAC       FSPW          ; LOAD UP FILTER SELECT AND PASSBANDWIDTH
        AND       MSOOFF        ; MASK IN PASSBANDWIDTH SPECIFIER PW
        SACL      TEMP
*
        LAC       ONE,14        ; 16384
        SACL      TEMP1
*
        LAC       FLSTOR,1
        ADD       ONE,5         ; HALF FILTERLENGTH = (2FL +32)
        SACL      TEMP2         ; DIVISOR
*
        LT        TEMP1
        MPY       TEMP          ; 16384 * PW
        PAC                     ; DIVIDEND IN ACCUMULATOR
*
        LARK      0,15
*
SLOOP   SUBC      TEMP2
        BANZ      SLOOP
*
        SACL      SINCPH
        SUB       SINCPH,1
        SACL      SINCPH        ; INITIAL SINCPHASE
*
######################################################################
*
*   NOW ZERO ALL THE ACCUMULATORS
*
######################################################################
*
        LARK      AR1,FREQ1
        LARK      AR0,2         ; SET UP LOOP COUNTER FOR THE FIRST THREE
*                               ; FILTERS ONLY.
        LAC       ONE,DTMFLG
        AND       FLAGS
        BNZ       ZLOOP         ; IF DTMF IS ON, ONLY DO THREE FILTERS,
*
        LARK      AR0,5         ; ELSE DO ALL SIX
*
ZLOOP   LARP      AR1           ; FOR EACH FILTER, DO DUMMY INCREMENTS OF
        LAC       *+            ; AR1 OVER THE FREQN AND PHASEN LOCATIONS,
```

```
        LAC       *+
        ZAC                     ; THEN ZERO THE ACCUMULATOR AND PUT ZERO
        SACL      *+            ; INTO THE NEXT FOUR LOCATIONS.
        SACL      *+
        SACL      *+
        SACL      *+,0,AR0
*
        BANZ      ZLOOP         ; THEN ZERO THE REMAINING ACCUMULATORS
*
        SACL      ACSQHI        ; HIGH WORD OF 32BIT SIGNAL SQUARED
*                               ; ACCUMULATOR
        SACL      ACSQLO        ; LOW WORD OF 32BIT SIGNAL SQUARED
*                               ; ACCUMULATOR
        SACL      ACWNHI        ; HIGH WORD OF 32BIT WINDOW ACCUMULATOR
        SACL      ACWNLO        ; LOW WORD OF 32BIT WINDOW ACCUMULATOR
*
        SACL      ACSWHI        ; HIGH WORD OF 32BIT SIN(X)/X. WINDOW
*                               ; PRODUCT ACCUMULATOR
        SACL      ACSWLO        ; LOW WORD OF 32BIT SIN(X)/X. WINDOW
*                               ; PRODUCT ACCUMULATOR
*
######################################################################
*
*   CHECK INTO BIT IN MODE REGISTER AND SET THE DTMF ON/OFF FLAG IN THE FLAGS
*   REGISTER ACCORDINGLY
*
######################################################################
*
        LAC       ONE,DTMFLG
        AND       FLAGS
        SACL      TEMP          ; GET CURRENT DTMF FLAG INTO TEMP
*
        LAC       ONE,DTMFBT
        AND       STMODE
        SACL      TEMP1         ; GET STATE OF DTMF BIT INTO TEMP1
*
        LAC       TEMP1,(DTMFLG-DTMFBT)
        XOR       TEMP          ; RESULT IS A ONE IF FLAGS ARE DIFFERENT
*
        XOR       FLAGS
        SACL      FLAGS         ; DTMFLG NOW EQUALS DTMFBT
*
        LAC       ONE,FSTFLG    ; LOAD A 1 IN THE FIRST BLOCK FLAG POSITION
        OR        FLAGS
        SACL      FLAGS         ; THIS HAS SET THE FIRST BLOCK FLAG
*
        RET
*
######################################################################
*
*   ROUTINE: SQRT
*
*   REFERENCE IN FLOWCHART: NONE
*
```

```
*    FUNCTION: USED IN THE LEVEL CALCULATION ROUTINE. GENERATES THE SQUARE
*    ROOT OF AN INTEGER, WITH AN OUTPUT WHICH SATURATES AT 255.
*
############################################################################
*
SQRT      .set      $
*
############################################################################
*
*    THIS IS THE SQUARE ROOT ROUTINE. THE RESULT RANGE IS ZERO TO 255, AND IS
*    THE NEAREST INTEGER TO THE SQUARE ROOT OF THE INPUT NUMBER. ANY INPUT
*    NUMBER WHICH HAS A SQUARE ROOT r 254.5 WILL GIVE A RESULT IF 255. THE
*    INPUT NUMBER MUST BE STORED IN THE PAIR OF LOCATIONS TEMP1(HIGH) AND
*    TEMP2(LOW) IN 2**2 FORM, AND MUST BE POSITIVE. NEGATIVE NUMBERS WILL GIVE
*    THE RESULT ZERO. TEMP IS USED AS A TEMPORARY LOCATION, AND THE RESULT IS
*    RETURNED IN TEMP3. THE ROUTINE TAKES 111 CYCLES. SQRT ALWAYS RETURNS WITH
*    ARP = 0
*
############################################################################
*
          LAC       ONE,8          ; LOAD UP 128 * 2**1
*                                  ; WORK IN 2**1 FORM THROUGHOUT.
          SACL      TEMP           ; INITIAL INCREMENT IS 128 * 2**1
          SUB       ONE
          SACL      TEMP3          ; INITIAL ROOT GUESS IS 127.5 * 2**1
*
          LARP      AR0
          SAR       AR0,SAMPLE     ; SAVE AR0 IN AN UNUSED LOCATION
*
          LARK      AR0,7          ; SET UP AR0 FOR 8 ITERATIONS
*
LOOP0     LAC       TEMP,15
          SACH      TEMP           ; HALVE THE INCREMENT
*
          LT        TEMP3          ; SQUARE THE ROOT
          MPY       TEMP3
          PAC
*
          SUBH      TEMP1
          SUBS      TEMP2
*
          BLEZ      RTOOSM         ; ROOT TOO SMALL
*
RTOOBG    .set      $              ; ROOT TOO BIG
*
          ZALS      TEMP3
          SUB       TEMP           ; SUBTRACT CURRENT INCREMENT FROM ROOT
          SACL      TEMP3
          BANZ      LOOP0
          B         END
*
RTOOSM    .set      $
*
          ZALS      TEMP3
          ADD       TEMP           ; ADD CURRENT INCREMENT TO ROOT
          SACL      TEMP3
          BANZ      LOOP0
*
END       .set      $
*
          LAC       TEMP3,15
          SACH      TEMP3          ; PUT ROOT INTO 2**0 FORM
*
          LAR       AR0,SAMPLE     ; RETRIEVE AR0 FROM ITS TEMPORARY STORE.
          RET
*
############################################################################
*
*    ROUTINE: DTMF
*
*    REFERENCE IN FLOWCHART: DTMF
*
*    FUNCTION: DETECT DTMF DIGITS
*
############################################################################
*
RSDTMF    .set      $
*
          LDPK      1
          ZAC
          SACLC      SIGCNT        ; ZERO VARIABLES
          SACL      CNTR
          SACL      CNTR1
          SACL      CNTR2
          SACL      STOP
          SACL      GAP
          SACL      PAUSE
          SACL      F1             ; ZERO FREQUENCY ARRAY
          SACL      F2
          SACL      F3
          SACL      F4
          SACL      F5
          SACL      F6
          SACL      F7
          SACL      F8
          SACL      ADJL
          SACL      ADJH
          SACL      SEMA           ; INITIALIZE SEMAPHORE
          SACL      TESTG
          SACL      TESTB
          LACK3
          SACLTMP
AGAIN     LDPK      0              ; RETURN TO PAGE 0
          RET                      ; END OF DTMF PROCESSING
*
DTMF      .set      $
*
          LDPK      1              ; DTMF PROCESSING ON PAGE 1
```

```
        LAC     UNITY           ; DECIMATE THE SAMPLES, DTMF USES
        XOR     DECIM           ; ALTERNATE ONES.
        AND     UNITY
        SACL    DECIM
        BZINP
*
        LDPK    0               ; RETURN TO PAGE 0
        RET
*
INP     .set    $
*
**********************************************************************
*
*   THIS SECTION ADDED TO IMPROVE THE DYNAMIC RANGE BY PROVIDING A DYNAMIC
*   THRESHOLD MINTHH WHICH COMES INTO PLAY DURING LOUD SIGNALS.
*
**********************************************************************
*
        LAC     MINTHH,2
        SUBADJL
        BGZ     MELSE
*
MTHEN   LAC     ADJL,14
        SACH    MINTHH
        ZAC
        SACL    MINTHL
        BMEND
*
MELSE   ZALH    MINTHH
        ADDS    MINTHL
        SUB     MINTHH,6        ; DECAY HALF LIFE OF 700 SAMPLES
        SACH    MINTHH
        SACL    MINTHL
*
MEND    .set    $
*
        LACK    MINTH
        SUB     MINTHH
        BLEZ    MNTHOK
*
        LACK    MINTH
        SACL    MINTHH
*
MNTHOK  .set    $
*
        LDPK0
        LAC     SAMPLE,2        ; SCALE INTO CORRECT RANGE
        LDPK1
        SACLX1
*
**********************************************************************
*
*   DTMF DECODER PROCESSING
*
```

```
**********************************************************************
*
        LT      X1              ; GET LINEAR INPUT SAMPLE
*
**********************************************************************
*
*   SCALE INPUT SAMPLE SO THAT THE 2ND ORDER SUB-FILTERS DO NOT OVERFLOW
*
**********************************************************************
*
        MPYK    SD              ; SCALE IT DOWN
        PAC
        SACH    X               ; AND STORE IT AS 8TH ORDER INPUT
*
**********************************************************************
*
*   8TH ORDER DETECTION WINDOW FOR DTMF LOW BAND
*
**********************************************************************
*
        LAC     X,15            ; NO
        LT      L1N2
        MPY     L1D             ; N2*D
        LTD     L1N1            ; N1-->N2
        MPY     L1C             ; N1*C
        APAC
        SACH    LY1,1           ; NO+N1*C+N2*D-->Y
        LTA     L2N2            ; NO+N1*C+N2*D+N1*C-->(ACC)
        SACH    L1N1,1          ; (ACC)-->N1
*
        LAC     X,15
        MPY     L2D
        LTD     L2N1
        MPY     L2C
        APAC
        SACH    LY2,1
        LTA     L3N2
        SACH    L2N1,1
*
        LAC     X,15
        MPY     L3D
        LTD     L3N1
        MPY     L3C
        APAC
        SACH    LY3,1
        LTA     L4N2
        SACH    L3N1,1
*
        LAC     X,15
        MPY     L4D
        LTD     L4N1
        MPY     L4C
        APAC
        SACH    LY4,1
        APAC
        SACH      L4N1,1
*
****************************************************************
*
*   THE OUTPUT OF THE 8TH ORDER FILTER (SUM OF 2ND ORDER SUB-FILTERS) HAS
*   ONLY + THE MAGNITUDE (BY THEORY). SO THERE IS AN UPSCALE BY TWO
*
****************************************************************
*
        ZALH      LY1            ; PROCESS RESULTS
        SUBH      LY2
        ADDH      LY3
       -SUBH      LY4
        SACH      LY,1           ; UPSCALE BY TWO AND STORE RESULTS
*
****************************************************************
*
*   8TH ORDER DETECTION WINDOW FOR DTMF HIGH BAND
*
****************************************************************
*
        LAC       X,15
        LT        H1N2
        MPY       H1D
        LTD       H1N1
        MPY       H1C
        APAC
        SACH      HY1,1
        LTA       H2N2
        SACH      H1N1,1
*
        LAC       X,15
        MPY       H2D
        LTD       H2N1
        MPY       H2C
        APAC
        SACH      HY2,1
        LTA       H3N2
        SACH      H2N1,1
*
        LAC       X,15
        MPY       H3D
        LTD       H3N1
        MPY       H3C
        APAC
        SACH      HY3,1
        LTA       H4N2
        SACH      H3N1,1
*
        LAC       X,15
        MPY       H4D
        LTD       H4N1
        MPY       H4C
```

```
        APAC
        SACH      HY4,1
        APAC
        SACH      H4N1,1
*
        ZALH      HY1            ; PROCESS RESULTS
        SUBH      HY2
        ADDH      HY3
        SUBH      HY4
        SACH      HY,1           ; UPSCALE BT TWO AND STORE RESULT
*
        ZALS      PAUSE          ; LOOKING AT GAP ?
        BNZ       GAP1
*
        ZALS      GAP            ; AGC DOES NOT RUN DURING
        BNZ       T8             ; THE GAP
*
****************************************************************
*
*   ACCUMULATE PEAKS
*
****************************************************************
*
        LAC       LY             ; SAVE PEAK INFO FOR NOW
        ABS
        SACL      TEMPD
        SUB       ADJL
        BLEZ      PK
*
        LAC       TEMPD
        SACL      ADJL
PK      LAC       HY             ; SAVE PEAK INFO FOR HIGH
        ABS
        SACL      TEMPD
        SUB       ADJH
        BLEZ      PK1
*
        LAC       TEMPD
        SACL      ADJH
PK1     ZALS      CNTR1
        ADD       UNITY
        SACL      CNTR1
        NOP
        LACK      MIN
        SUB       CNTR1          ; SAVE PEAKS OVER MIN SAMPLES
        BGZ       T8
*
        ZAC
        SACL      CNTR1
*
        ZALS      CNTR2          ; SET AGC FLAG AFTER CNTR2 UPDATES
        ADD       UNITY
        SACL      CNTR2
                                 ; LACK7
        LACK    6               ; 6 * 7 * 0.25 MS
        SUB     CNTR2
        BGZ     TSTW
*
        ZAC
        SACL    CNTR2
*
        LACK    1
        SACL    STOP            ; STOP WAITING FOR FILTER TRANSIENT
*
##########################################################################
*
*   TEST FOR TWIST - SET LEVELS WITH TWSTHI (LO>HI) AND TWSTLO (HI>LO)
*
##########################################################################
*
TSTW    LAC     ADJL
        SACL    TEMPD
        LAC     ADJL,8          ; TEST FOR TWIST BEFORE
        LTA     DJH
        MPYK    TWSTHI
        SPAC
        BGEZ    RSDTMF
*
        LAC     ADJH,8
        LTA     DJL
        MPYK    TWSTLO
        SPAC
        BGEZ    RSDTMF
*
ADJST   LAC     MINTHH          ; TEST FOR MINIMUM SIGNAL
        SUB     ADJH
        BGEZ    TCNT
*
        LAC     MINTHH          ; IN HI AND LO BAND
        SUB     ADJL
        BGEZ    TCNT
*
        LT      TEMPD           ; CALCULATE NEW THRESHOLDS
        MPYK    Z
        PAC
        SACH    THRSH8,4
        LT      ADJH
        MPYK    Z
        PAC
        SACH    THRHI,4
        LT      ADJL
        MPYK    Z
        PAC
        SACH    THRLO,4
*
##########################################################################
*
*   THRESHOLD 8TH ORDER RESULT
*
##########################################################################
*
T8      LAC     LY              ; GET LOW RESULT
        ABS                     ; MAKE POSITIVE
        SUB     THRSH8          ; APPLY THRESHOLD
        BLEZ    THR
*
        ZAC
        SACL    CNTR            ; ZERO TEMP CNTR
*
        B       SECND
*
THR     ZALS    CNTR            ; IF INPUT SIGNAL IS
        ADD     UNITY           ; GONE FOR MIN CONSECUTIVE
        SACL    CNTR            ; SAMPLES, RESET SYSTEM
        LACK    MIN
        SUB     CNTR            ; OR LOOK FOR GAP
        BGZ     TCNT
*
        ZALS    GAP             ; IS GAP REQUIRED ?
        BZ      RSDTMF
*
        ZAC
        SACL    GN1
        SACL    GN2
        SACL    Y5
*
        GAPC    ZAC
        SACL    SIGCNT          ; USE THIS FOR GAP COUNT
        SACL    CNTR
        LAC     THRSH8,13       ; GAP THRESHOLD = PREVS-
        SACH    TEMPD           ; 8TH THRESHOLD / 8  (DB)
        LACK1
        SACL    PAUSE           ; SET FLAG FOR GAP TRAP
*
GAP1    LAC     X1,12           ; THRESHOLD INPUT SAMPLE
        LT      GN1             ; HIGHPASS CHANNEL NOIAND
        MPY     A1              ; NOTCH OUT CP TONES
        LTA     GN2
        MPY     A2
        APAC
        SACH    Y5,1
        ZAC
        MPY     B2
        LTD     GN1
        MPY     B1
        LTD     Y5
        MPY     B0
        APAC
        SACH    Y5,1
        LAC     Y5
        ABS
        SUB     TEMPD
        BLEZ    GAP2
*
        ZALS    CNTR          ; ACCUMULATE SAMPLES
        ADD     UNITY         ; ABOVE GAP THRESHOLD
        SACL    CNTR
GAP2    ZALS    SIGCNT        ; INCREMENT GAP COUNT
        ADD     UNITY
        SACL    SIGCNT
*
**********************************************************************
*
*   INTER-DIGIT PAUSE IS DETERMINED BY FOLLOWING INSTRUCTION
*
**********************************************************************
*
CA1C    LACK    01Ch          ; LOAD GAP TIMER (07 MS + )
*
        SUB     SIGCNT        ; IS GAP TIME UP ?
        BGZ     AGAIN
*
        LACK    030h          ; SMOOTH OUT GLITCHES
        SUB     CNTR          ; IF THERE ARE TOO MANY SAMPLES
        BLEZ    GAPC          ; ABOVE THRESHOLD, DO GAP AGAIN
*
        LDPK    0
*
        ZAC
        SACL    GN1
        SACL    GN2
        SACL    Y5
        B       RSDTMF        ; RESET SYSTEM
*
**********************************************************************
*
*   THRESHOLD 2ND ORDER RESULTS
*
**********************************************************************
*
SECND   ZALS    GAP
        BNZ     AGAIN
*
        ZALS    STOP
        BZ      TCNT
*
        LAC     LY1
        ABS
        SUB     THRLO
        BLEZ    THR1
*
        ZALS    F1
        ADD     UNITY
        SACL    F1
THR1    LAC     LY2
        ABS
```

```
        SUB     THRLO
        BLEZ    THR2
*
        ZALS    F2
        ADD     UNITY
        SACL    F2
THR2    LAC     LY3
        ABS
        SUB     THRLO
        BLEZ    THR3
*
        ZALS    F3
        ADD     UNITY
        SACL    F3
THR3    LAC     LY4
        ABS
        SUB     THRLO
        BLEZ    THR4
*
        ZALS    F4
        ADD     UNITY
        SACL    F4
THR4    LAC     HY1
        ABS
        SUB     THRHI
        BLEZ    THR5
*
        ZALS    F5
        ADD     UNITY
        SACL    F5
THR5    LAC     HY2
        ABS
        SUB     THRHI
        BLEZ    THR6
*
        ZALS    F6
        ADD     UNITY
        SACL    F6
THR6    LAC     HY3
        ABS
        SUB     THRHI
        BLEZ    THR7
*
        ZALS    F7
        ADD     UNITY
        SACL    F7
THR7    LAC     HY4
        ABS
        SUB     THRHI
        BLEZ    TCNT
*
        ZALS    F8
        ADD     UNITY
        SACL    F8
```

```
*
TCNT     ZALS     GAP
         BNZ      AGAIN
*
         ZALS     SIGCNT        ; INCREMENT DTMFTONE COUNT
         ADD      UNITY
         SACL     SIGCNT
*
         LACK     SCNT          ; TEST FOR TIME UP
         SUB      SIGCNT
         BGZ      AGAIN
*
**************************************************************************
*
*    DETERMINE TONE DIGIT FROM SECOND ORDER COUNTERS
*
*    THE FREQUENCY IN EACH BAND WITH ENERGY ABOVE THE BIN THRESHOLD IS RECOG-
*    NISED AND THEN ZEROED. IN ORDER TO BE A VALID DTMF COMBINATION, ALL OTHER
*    BINS HAVE TO BE BELOW THRESHOLD.
*
**************************************************************************
*
         LARP     0
         LARK     0,0
         LARK     1,0
         LACK     THRHL1
         SUB      F1
         BGEZ     FI1
*
         SAR      1,F1
         B        FIND1
*
FI1      MAR      *+
         LACK     THRHL2
         SUB      F2
         BGEZ     FI2
*
         SAR      1,F2
         B        FIND1

FI2      MAR      *+
         LACK     THRHL3
         SUB      F3
         BGEZ     FI3
*
         SAR      1,F3
         B        FIND1
*
FI3      MAR      *+
         LACK     THRHL4
         SUB      F4
         BGEZ     NOFIND
*
         SAR      1,F4

FIND1    SAR      0,TEMPD
         LACK     LOLIM
         SUB      F1
         BLZ      NOFIND
*
         LACK     LOLIM
         SUB      F2
         BLZ      NOFIND
*
         LACK     LOLIM
         SUB      F3
         BLZ      NOFIND
*
         LACK     LOLIM
         SUB      F4
         BLZ      NOFIND
*
         LARK     0,0
         LACK     THRHH1
         SUB      F5
         BGEZ     FI5
*
         SAR      1,F5
         B        FIND2
*
FI5      MAR      *+
         LACK     THRHH2
         SUB      F6
         BGEZ     FI6
*
         SAR      1,F6
         B        FIND2
*
FI6      MAR      *+
         LACK     THRHH3
         SUB      F7
         BGEZ     FI7
*
         SAR      1,F7
         B        FIND2
 FI7     MAR      *+
         LACK     THRHH4
         SUB      F8
         BGEZ     NOFIND
*
         SAR      1,F8
 FIND2   .set     $
         LACK     HILIM
         SUB      F5
         BLZ      NOFIND
*
         LACK     HILIM
         SUB      F6
         BLZ      NOFIND
```

```
*
        LACK    HILIM
        SUB     F7
        BLZ     NOFIND
*
        LACK    HILIM
        SUB     F8
        BLZ     NOFIND
*
        LAC     TEMPD,2         ; + LOW-BAND OFFSET * 4
        SAR     0,TEMPD
        ADD     TEMPD           ;+ HIGH-BAND OFFSET
*
******************************************************************************
*
*   OUTPUT ROUTINE
*
******************************************************************************
*
DINT                            ; INTERRUPT PROTECTED BECAUSE THE OVERRUN
        SACL    DIGIT           ; BIT HAS NOT BEEN UPDATED YET
        LACK    070h
        ADD     DIGIT
        SACL    DIGIT
        LDPK    0
        LAC     ONE,DTINBT
        AND     STMODE
        BGZ     NOVRUN
*
        LDPK    1
        LACK    080h
        ADD     DIGIT
        SACL    DIGIT
        LDPK    0
*
NOVRUN  EINT
        LAC     OSHOLD
        SACL    OSTIME
*
        LAC     ONE,TPRFLG
        AND     FLAGS
*
        BZ      DTINT
*
        LAC     ONE,ONSFLG
        OR      FLAGS
        SACL    FLAGS
*
DTINT   LAC     ONE,DTINBT
        XOR     STMODE
        AND     STMODE
        SACL    STMODE
*
DATTEN  CALL    ATTEN
```

```
*
        CALL    XFUPD
*
        B       AGAIN
*
NOFIND  .set    $
        ZALS    TESTB
        ADD     UNITY                   ; INCREMENT BAD DIGITS
        SACL    TESTB
*
        LACK    1                       ; TONE NOT VALID
        SACL    GAP                     ; NOW LOOK FOR GAP
        B       AGAIN
*
******************************************************************************
*
*   ROUTINE: INTHDL
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: INTERRUPT HANDLER
*
******************************************************************************
*
INTHDL  .set    $
*
        SST     SRSAVE                  ; SAVE STATUS REGISTER
        LDPK    1
        SACH    ACCUHI
        SACL    ACCULO                  ; SAVE CONTENTS OF ACCUMULATOR
        LDPK    0
        SAR     AR0,ARSAVE              ; SAVE CURRENT AUXILLIARY
*                                       ; REGISTER IN ARSAVE
        LARP    0                       ; POINT TO AR0
*
******************************************************************************
*
*   CHECK SOURCE OF INTERRUPT, EITHER CODEC OR PARALLEL INTERFACE.
*
******************************************************************************
*
        IN      ITEMP,CTLPRT            ; READ CONTROL REGISTER
        LAC     ONE,3
        AND     ITEMP                   ; CHECK FOR CODEC INTERRUPT BIT
*                                       ; SET
*
        BZ      NOTCDC
*
******************************************************************************
*
*   CODEC INTERRUPT HANDLER
*
******************************************************************************
*
```

```
CODEC   .set    $               ; CODEC INTERRUPT HANDLER
*
        ADDS    CTL320
        SACL    ITEMP
        OUT     ITEMP,CTLPRT    ; CLEAR CODEC INTERRUPT
*
        LAC     ONE,INTFLG
        OR      FLAGS
        SACL    FLAGS           ; SET CODEC INTERRUPT INDICATOR FLAG
*
QREAD   LAR     AR0,QIN         ; LOAD UP THE QIN POINTER
*
        IN      *,CDCPRT        ; READ NEXT LINEARIZED SAMPLE INTO
*                               ; QUEUE IN SIGNED MAGNITUDE FORM.
*
        LAC     QIN
        SUB     ONE             ; DECREMENT THE QIN POINTER.
        SACL    ITEMP
*
        LAC     ONE,3
        OR      ITEMP           ; POINTER COUNTS 003Fh THRU 0038h
        SACL    QIN             ; UPDATE QIN
        B       CINEND
INTEND  .set    $               ; COMMON EXIT PATH FROM INTERRUPT HANDLER
*
        LACK    7
        ADDS    CTL320
        SACL    ITEMP
        OUT     ITEMP,CTLPRT    ; CLEAR ALL LATCHED NON-CODEC INTERRUPTS
*
CINEND  LAR     AR0,ARSAVE      ; RESTORE AR0
        LDPK    1
        ZALH    ACCUHI
        ADDS    ACCULO          ; RESTORE ACCUMULATOR
        LST     SRSAVE          ; RESTORE STATUS REGISTER
        EINT
        RET
*
************************************************************************
*
*   PARALLEL INTERFACE INTERRUPT HANDLER
*
************************************************************************
*
NOTCDC  .set    $               ; PARALLEL INTERFACE INTERRUPT HANDLER
*
        LAC     ONE,STAFLG
        AND     FLAGS           ; CHECK STATE BIT, IF IT IS SET WE ARE HALF
*                               ; WAY THROUGH A WRITE OPERATION AND MUST
*                               ; DO THE SECOND PAIR OF TRANSFERS.
        BGZ     WRITE2
*
************************************************************************
*
```

```
*   THE STATE BIT WAS NOT SET, SO WE ARE AT THE BEGINNING OF A TRANSFER
*   OPERATION, EITHER READ OR WRITE. READ OPERATIONS REQUIRE TWO TRANSFERS
*   ONE EACH WAY, WRITE TRANSFERS REQUIRE TWO TRANSFERS IN EACH DIRECTION AND
*   TWO INTERRUPTS, WHICH IS WHY A STATE BIT IS REQUIRED TO FLAG THE SECOND
*   HALF OF A WRITE OPERATION.
*
************************************************************************
*
        IN      ITEMP,DATPRT    ; READ COMMAND FROM INTERFACE.
*
        LAC     ONE,5           ; MASK INTERFACE COMMAND TO 5 BITS
        SUB     ONE
        AND     ITEMP
        SACL    ITEMP
*
        LAC     ONE,RWBIT
        AND     ITEMP           ; CHECK RW BIT, IF IT IS SET, THIS IS A
*                               ; READ.
*
        BGZ     READOP
*
************************************************************************
*
*   THIS IS THE FIRST PART OF A WRITE TRANSFER
*
************************************************************************
*
WRITE1  .set    $
*
        LAC     ONE,STAFLG
        OR      FLAGS           ; SET THE STATE BIT TO FLAG THAT THE FIRST
        SACL    FLAGS           ; PART OF A WRITE OPERATION HAS BEEN DONE.
*
        LAC     ITEMP
        SACL    CMSAVE          ; SAVE THE COMMAND BYTE FOR THE SECOND PART
*                               ; OF THE WRITE TRANSFER.
ACKNOW  .set    $
*
        LAC     STMODE,8        ; ACKNOWLEDGE WRITE BY WRITING OUT STATUS
        SACH    ITEMP           ; TO STATUS PORT
        LAC     ITEMP
        AND     MSOOFF
        SACL    ITEMP
        OUT     ITEMP,STAPRT    ; ALSO CLEARS HARDWARE INTERRUPT SOURCE.
*
        B       INTEND
*
************************************************************************
*
*   THIS IS THE FIRST PART OF A READ TRANSFER
*
************************************************************************
*
READOP  .set    $
```

```
*
        LACK     INTTAB          ; LOAD UP THE START ADDRESS OF INTTAB
        ADDS     ITEMP           ; ADD ON THE INTERFACE COMMAND, WHICH IS AN
*                                ; OFFSET INTO THE INTTAB TABLE OF REGISTER
*                                ; MAPPINGS.
        TBLR     ITEMP           ; READ THE REGISTER MAPPING.
*
        LAR      AR0,ITEMP       ; MAPPING WORD IN AR0. THE MAP WORD
*                                ; INDICATES WHICH PHYSICAL ALLOCATION IS TO
*                                ; BE ACCESSED IN ITS LOWER 9 BITS, AND ALSO
*                                ; CONTAINS CONTROL FLAGS ON THE UPPER BYTE.
*                                ; THESE CONTROL FLAGS ARE USED BY THE
*                                ; INTERRUPT HANDLER TO INDICATE WHAT TYPE
*                                ; OF TRANSFER IS HAPPENING.
*
        LAC      ONE,TESTBT      ; CHECK FOR TEST BIT SET IN MODE REGISTER
        AND      STMODE          ; WHICH MEANS THAT TEST MODE IS CURRENTLY
*                                ; ON
        BZ       NOTEST
*
        LAC      ONE,LBIT        ; CHECK FOR L BIT SET IN MAP WORD
        AND      ITEMP           ; WHICH MEANS AN ACCESS OF ADDR 0 OR 1
*
        BGZ      NOTEST
*
        OUT      CMSAVE,DATPRT   ; ALL OTHER REGISTERS IN TEST MODE ARE
*                                ; MAPPED ONTO THE SAME REGISTER (CMSAVE).
        B        INTEND
*
NOTEST  LAC      ONE,TBIT        ; CHECK FOR T BIT SET IN MAP WORD WHICH
        AND      ITEMP           ; MEANS A READ OF CURRENT TIME
*
        BGZ      TBTSET
*
        LAC      ONE,SBIT        ; CHECK FOR S BIT SET IN MAP WORD WHICH
        AND      ITEMP           ; MEANS A READ OF THE STATUS REGISTER
*
        BGZ      SBTSET
*
        LAC      ONE,UBIT        ; CHECK FOR U BIT SET IN MAP WORD WHICH
        AND      ITEMP           ; MEANS A READ OF AN UPPER BYTE
*
        BGZ      UBITRD
*
##############################################################################
*
*   THIS IS A READ OF THE LOWER BYTE OF THE LOCATION SPECIFIED IN THE LOWER 9
*   BITS OF THE MAP WORD AND ALSO IN AR0.
*
##############################################################################
*
        OUT      *,DATPRT
*
        B        INTEND
```

```
*
##############################################################################
*
*   THIS IS A READ OF THE CURRENT TIME MS REGISTER.
*
##############################################################################
*
TBTSET  .set     $
*
        LAC      CRTIME
        SACL     CRHOLD          ; SAVEW CURRENT TIME
*
        B        UBITRD
*
##############################################################################
*
*   THIS IS A READ OF THE STATUS REGISTER.
*
##############################################################################
*
SBTSET  .set     $
*
        LAC      STMODE,8
        SACH     ITEMP
        OUT      ITEMP,STAPRT    ; WRITE OUT MS BYTE OF STMODE
*
        B        INTEND
*
##############################################################################
*
*   THIS IS A READ OF THE UPPER BYTE OF THE LOCATION SPECIFIED IN THE LOWER 9
*   BITS OF THE MAP WORD
*
##############################################################################
*
UBITRD  .set     $
*
        LAC      *,8
        SACH     ITEMP
        OUT      ITEMP,DATPRT
*
        B        INTEND
*
##############################################################################
*
*   THIS IS THE SECOND PART OF A WRITE TRANSFER
*
##############################################################################
*
WRITE2  .set     $
*
        XOR      FLAGS           ; CLEAR STATE BIT IN FLAGS REGISTER
        SACL     FLAGS
*
```

```
        LACK    INTTAB
        ADDS    CMSAVE          ; ADD ON THE SAVED INTERFACE COMMAND, WHICH
*                               ; IS AN OFFSET INTO THE INTTAB TABLE OF
*                               ; REGISTER MAPPINGS.
        TBLR    ITEMP           ; READ THE REGISTER MAPPING.
*
        LAR     AR0,ITEMP       ; MAPPING WORD IN AR0.
*
        IN      CMSAVE,DATPRT   ; READ THE DATA IN WHICH IS TO BE WRITTEN
*                               ; TO A REGISTER, USE CMSAVE FOR THIS.
        LAC     CMSAVE
        AND     MSOOFF
        SACL    CMSAVE          ; MASK OUT UNDEFINED BITS
*
        LAC     ONE,MBIT        ; CHECK FOR M BIT SET IN MAP WORD
        AND     ITEMP           ; WHICH MEANS A WRITE TO THE MODE REGISTER
*
        BGZ     MBTSET
*
        LAC     ONE,TESTBT      ; CHECK FOR TEST BIT SET IN MODE REGISTER
        AND     STMODE          ; WHICH MEANS A TEST MODE IS CURRENTLY ON
*
        BZ      CHECKF          ; TEST MODE NOT SET
*
        LAC     ONE,LBIT        ; CHECK FOR L BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A WRITE TO ADDRESS 0 OR 1
*
        BZ      ACKNOW          ; TEST MODE. LEAVE DATA IN CMSAVE
*
CHECKF  LAC     ONE,FBIT        ; CHECK FOR F BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A WRITE TO A FREQUENCY REGISTER
*
        BGZ     FBTSET
*
        LAC     ONE,UBIT        ; CHECK FOR U BIT SET IN MAP WORD WHICH
        AND     ITEMP           ; MEANS A WRITE TO AN UPPER BYTE
*
        BGZ     UBITWR
*
######################################################################
*
*   THIS IS THE SECOND PART OF A WRITE TO THE LOWER HALF OF A REGISTER
*
######################################################################
*
LOWWRT  LAC     MSOOFF,8
        AND     *               ; ADD THE CURRENT CONTENTS OF THE UPPER
LOWWR2  ADDS    CMSAVE          ; REGISTER TO THE NEW LOWER HALF
        SACL    *
*
        B       ACKNOW
*
######################################################################
*
```

```
*   THIS IS A WRITE TO THE MODE REGISTER
*
######################################################################
*
MBTSET  .set    $
*
######################################################################
*
*   OPERATIONS TO BE PERFORMED HERE ARE:
*
*   1. COPY THE BYTE WRITTEN, INTO THE MODE REGISTER, PRESERVING THE STATE
*      OF THE RC BITS WHICH ARE ALREADY IN THE MODE REGISTER IN POSITIONS 0
*      AND 1.
*   2. SET THE SAMPLE SCALE FACTOR ACCORDING TO THE SC BITS.
*   3. IF THE TEST BIT HAS BEEN SET, THEN CLEAR THE DTMF AND TONE BITS.
*   4. IF ANY INTERRUPTS HAVE BEEN ACKNOWLEDGED (IACK BITS SET) THEN SET THE
*      APPROPRIATE STATUS BIT BACK TO A 1 AND UPDATE THE STATUS OF THE XF
*      FLAG.
*   5. IF THE TEST BIT IS SET, THEN DO A SELF TEST, ACKNOWLEDGE THE WRITE,
*      THEN RESTART, ELSE ACKNOWLEDGE THE WRITE AND RETURN FROM INTERRUPT.
*
######################################################################
*
        LAC     MSOOFF,8
        ADD     ONE,1
        ADD     ONE             ; MASK IN STATUS BYTE AND RC BITS IN MODE
*                               ; BYTE.
*
        AND     STMODE          ; ZERO MODE BYTE, EXCEPT FOR THE BOTTOM TWO
        SACL    STMODE          ; BITS, AND STORE BACK IN STMODE
*
        LACK    3
        AND     CMSAVE          ; EXTRACT THE SC BITS FROM THE BYTE BEING
        SACL    ITEMP           ; WORKED
*
        LACK    SCATAB          ; ADD IN TABLE OFFSET .
        ADD     ITEMP
        TBLR    SCALEF          ; READ THE DESIRED SCALE FACTOR INTO SCALEF
*
        LAC     ONE,8
        SUB     ONE,2           ; MASK IN LOWER BYTE, EXCEPT FOR THE BOTTOM
*                               ; TWO (SC) BITS, OF THE BYTE BEING WRITTEN
*                               ; TO MODE
        AND     CMSAVE
        ADDS    STMODE          ; ADD TP STMODE. MODE IS NOW UPDATED, BUT
        SACL    STMODE          ; THE BOTTOM TWO (RC) BITS HAVE BEEN LEFT
*                               ; INTACT IN THE READABLE VERSION OF MODE.
*
        LAC     ONE,TESTBT      ; CHECK FOR TEST BIT SET IN MODE REGISTER
        AND     STMODE          ; WHICH MENAS THAT TEST MODE IS CURRENTLY
*                               ; ON
*
        BZ      CLRACK
*
```

```
TESTW   B       SLFTST          ; BRANCH TO THE SELF TEST ROUTINE
*
CLRACK  LAC     CMSAVE,14
        SACH    CMSAVE
        LACK    7
        AND     CMSAVE          ; MASK IN THE THREE IACKBITS
        SACL    ITEMP
*
        LACK    ACKTAB          ; ADD IN TABLE OFFSET
        ADD     ITEMP
*
        TBLR    ITEMP
        LAC     ITEMP           ; STATUS BIT TO BE SET TO 1, IF ANY
        OR      STMODE
        SACL    STMODE
*
ENDSLF  .set    $               ; SELF TEST BRANCHES BACK TO HERE
*
        CALL    XFUPD           ; UPDATE XF FLAG
*
        B       ACKNOW
*
##########################################################################
*
*   THIS IS THE SECOND PART OF A WRITE TO A FILTER CENTER FREQUENCY
*
##########################################################################
*
FBTSET  .set    $
*
        LAC     MSOOFF,8
        AND     FMSFL           ; ADD THE CURRENT CONTENTS OF FREQUENCY MS
*                               ; BYTE
*
        B       LOWWR2
*
##########################################################################
*
*   THIS IS THE SECOND PART OF A WRITE TO THE UPPER HALF OF A REGISTER
*
##########################################################################
*
UBITWR  .set    $
*
        LAC     MSOOFF
        AND     *               ; ADD THE CURRENT CONTENTS OF THE LOWER
        ADD     CMSAVE,8        ; REGISTER TO THE NEW UPPER HALF
        SACL    *
*
        CALL    XFUPD           ; UPDATE THE XF FLAG (ONLY AFFECTED BY
*                               ; WRITES TO THE CONTROL TEGISTER.)
        B       ACKNOW
*
##########################################################################
```

```
*
*   ROUTINE: CRESET
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: COLD RESET HANDLER
*
##########################################################################
*
*                                       ; INITIALISE PROCESSOR
*
CRESET  .set    $               ; COLD RESET. BRANCH HERE FROM RESET
*                               ; VECTOR.
*
        LDPK    0               ; INITIALIZE DATA PAGE POINTER
*
        BV      CLROVF          ; CLEAR OVERFLOW FLAG
*
CLROVF  SOVM                    ; SET OVERFLOW MODE
*
        LARP    0
*
        CALL    WRESET
*
        CALL    RSTFIL          ; CALL RESET FILTERING ROUTINE TO CLEAR
*                               ; DOWN ALL ACCUMULATORS AND SET UP FILTER
*                               ; READY FOR THE FIRST BLOCK.
*
        CALL    RSDTMF          ; REINITIALIZE THE DTMF CODE
*
        EINT                    ; ENABLE INTERRUPTS
        B       MAIN            ; JUMP TO MAIN (MAINSTREAM CODE)
*
##########################################################################
*
*   ROUTINE: WRESET
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: WARM RESET HANDLER
*
##########################################################################
*
                                        ; RE-INITIALIZE PROCESSOR
*
##########################################################################
*
WRESET  .set    $               ; WARM RESET.  CALLED BY SELF-TEST ROUTINE
*                               ; AND BY COLD RESET HANDLER.
*
##########################################################################
*
*   ZERO ALL RAM LOCATIONS IN PAGES 0 & 1
*
```

```
******************************************************************************
*
        LARK    AR0,0FFh        ; SET UP AR0 TO CONTROL COPYING LOOP
        ZAC
*
ZERO0   LAR     PAR0
        SACL    *
        BANZ    ZERO0
*
        LACK1
        SACL    ONE
        LDPK    1
        SACL    UNITY
        LDPK    0
*
******************************************************************************
*
*   INITIALIZE DTMF MEMORY LOCATIONS IN PAGE 1
*
******************************************************************************
*
        LARK    AR0,(IEND0-IVAR0-1) ; SET UP AR0 TO CONTROL COPYING LOOP
        LARK    AR1,(IEND0-1)   ; SET UP AR1 TO POINT TO DATA SUM
*
        LACK    CONEN0          ; LOAD ACCUMULATOR WITH (1 + END OF TABLE)
*
COPY0   SUB     ONE
        LAR     PAR1
        TBLR    *-,AR0
        BANZ    COPY0
*
******************************************************************************
*
*   INITIALIZE TONE DETECTOR LOCATIONS IN PAGE 0
*
******************************************************************************
*
        LARK    AR0,(IEND1-IVAR1-1) ; SET UP AR0 TO CONTROL COPYING LOOP
        LARK    AR1,IEND1-1     ; SET UP AR1 TO POINT TO DATA RAM.
*
        LACK    CONEN1          ; LOAD ACCUMULATOR WITH (1 + END OF TABLE)
*
COPY1   SUB     ONE
        LAR     PAR1
        TBLR    *-,AR0
        BANZ    COPY1
*
        OUT     CTL320,CTLPRT   ; SET UP LOWER CONTROL REGISTER BITS TO
*                               ; FD9Fh
        OUT     CTL32U,CTLUPR   ; WRITE VALUE 0CFEh TO UPPER CONTROL PORT
*
        LAC     CTL322          ; RESET THE PORT 0 CONTROL BIT TO POINT AT
*                               ; THE LOWER CONTROL PORT, AND SET SCLK TO
*                               ; BE AN OUTPUT. CLEAR THE INTERRUPT ACKNOW-
*                               ; LEDGE BITS FROM CTL320.
        SACL    CTL320          ; KEEP THIS AS THE DEFAULT VALUE OF CTL320
*                               ; IN RAM
        OUT     CTL320,CTLPRT   ; SET UP LOWER CONTROL REGISTER BITS TO
*                               ; 7C90h
*
        RET
*
******************************************************************************
*
*   ROUTINE: ATTEN
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: WRITE OUT STATUS TO DRAW ATTENTION TO A CHANGE IN ONE OR MORE
*             OF THE STATUS BITS.
*
******************************************************************************
*
ATTEN   .set    $
*
        LAC     STMODE,8
        SACH    ITEMP
        OUT     ITEMP,ATTPRT    ; WRITE OUT MS BYTE OF STMODE
*
        RET
*
******************************************************************************
*
*   ROUTINE: XFUPD
*
*   REFERENCE IN FLOWCHART: NONE
*
*   FUNCTION: UPDATE THE XFFLAG. CALLED EVERY TIME A STATUS REGISTER
*             INTERRUPT FLAG IS UPDATED, AND EVERY TIME THE CONTROL REGISTER
*             IS WRITTEN TO.
*
******************************************************************************
*
XFUPD   .set    $
*
        LAC     MSOOFF,8
        XOR     STMODE
        AND     CTLTSL
*
        SACL    ITEMP
        LAC     MSOOFF,8
        AND     ITEMP           ; ANY BITS NOW SET MEAN PENDING ENABLED
*                               ; INTERRUPTS ?
        BZ      SETXF
*
        CLRXF   LACONE,CXFBIT   ; INTERRUPT(S) ASSERTED
        XOR     CTL320
        AND     CTL320          ; CLEAR XF BIT
        SACL    CTL320
*
        RET
*
SETXF   LAC     ONE,CXFBIT      ; NO INTERRUPT
        OR      CTL320          ; SET XF BIT
        SACL    CTL320
*                               ; CTL320 WILL GET WRITTEN TO THE CONTROL
*                               ; REGISTER DURING THE NEXT CODEC INTERRUPT.
        RET
*
##############################################################################
*
*    ROUTINE: SLFTST
*
*    REFERENCE IN FLOWCHART: NONE
*
*    FUNCTION: SELF TEST OF PROCESSOR. PERFORM INTERNAL RAM TEST AND ROM
*              CHECKSUM TEST. SELF TEST USES THE STACK AS A HOLDING REGISTER.
*              AT THE END OF THE SELF TEST, THE PROCESSOR IS REINITIALIZED AND
*              BRANCHES INTO THE MAIN STREAM CODE. THE CONTENTS OF THE STACK
*              ARE DISCARDED. EACH TIME A VALUE IS PUSHED, THE STACK IS FIRST
*              POPPED, TO PREVENT STACK OVERFLOW MESSAGES FROM OCCURRING ON
*              SIMULATORS. THESE REDUNDANT POP INSTRUCTIONS MAY BE REMOVED;
*              THEY ARE MARKED (+) IN THE CODE.
*
##############################################################################
*
SLFTST  .set    $
*
        LACK    ROMFAI
        PUSH                    ; PUT ROM FAIL RESULT ONTO STACK
*
ROMTST  .set    $
*
##############################################################################
*
*   ROM CHECKSUM TEST
*
##############################################################################
*
        ZAC
        SACL    TOTAL           ; SET CHECKSUM TO ZERO
*
        LACK    1
        SACL    ONE             ; RESTORE THE UNITY LOCATION
        LDPK    1
        SACL    UNITY
*
        LAC     MARKER          ; PROGRAM END ADDRESS IN ACCUMULATOR
*
        LDPK    0
ROMLP   SUB     ONE
        TBLR    ROMVAL
*
        SACL    ACCHLD
*
        ZALS    ROMVAL
LABEL2  ADDS    TOTAL
        SACL    TOTAL
*
        LAC     ACCHLD
        BNZ     ROMLP
*
        LAC     TOTAL           ; CHECK FOR ZERO CHECKSUM
        BNZ     RESULT
*
##############################################################################
*
*   RAM TEST
*
##############################################################################
*
        POP                     ; (+) SEE NOTE IN ROUTINE HEADER
        LACK    RAMFAI
        PUSH                    ; PUT RAM FAIL RESULT INTO STACK
        LARK    AR0,0FFh        ; AR0 COUNTS 0100h ITERATIONS OF OUTER
*                               ; LOOP
        LARK    AR1,15          ; AR1 COUNTS 16 ITERATIONS OF INNER LOOP
        LACK    1               ; START WITH ONE
INLP    LARP    AR0
        SACL    *               ; STORE IN RAM LOCATION
        SUBS    *               ; READ IT BACK AND CHECK IT BY SUBTRACTION
        BNZ     RESULT
*
        LAC     *,1,AR1         ; SHIFT OPERAND LEFT BY ONE BIT, REPEATING
        BANZ    INLP            ; FOR EACH BIT POSITION.
*
        BGEZ    RESULT          ; FINALLY CHECK THAT BIT HAS SHIFTED OUT OF
*                               ; LOWER ACCUMULATOR (FFFF0000h IN ACC)
        MAR     *,0             ; FLIP BACK TO AR0, REPEAT WHOLE FOR EACH
        SACL    *               ; LEAVE TESTED LOCATION AT ZERO RAM
        BANZ    OUTLP           ; LOCATION
*
CDCTST  .set    $
*
##############################################################################
*
*   CODEC INTERRUPT CHECK
*
##############################################################################
*
        POP                     ; (+) SEE NOTE IN ROUTINE HEADER.
*
        CALL    WRESET          ; DO A WARM RESET TO RE-INITIALIZE ALL
*                               ; VARIABLES
*
        ZALH    ONE
        SUB     ONE,7           ; CREATE MASK TO ENABLE ONLY CODEC
*                               ; INTERRUPTS
        AND     CTL320
        SACL    CTL320
*
        OUT     CTL320,CTLPRT   ; ENABLE CODEC INTERRUPTS ONLY, IN MASK.
*                               ; ONLY TAKES EFFECT AFTER THE NEXT EINT
        LARP    AR0
        LARK    AR0,INTMAX      ; LOAD AR0 WITH THE MAXIMUM NUMBER OF LOOPS
*                               ; EXPECTED BETWEEN INTERRUPTS. LOOPA IS 6
*                               ; CYCLES LONG.
        EINT                    ; CODEC INTERRUPTS NOW ENABLED.
*
        LAC     ONE,INTFLG
        AND     FLAGS
        XOR     FLAGS
        SACL    FLAGS           ; CLEAR INTERRUPT INDICATOR FLAG. AWAIT
*                               ; NEXT INTERRUPT, COUNTING LOOPS WITH AR0
LOOPA   LAC     ONE,INTFLG
        AND     FLAGS
        BNZ     FSTINT          ; BRANCH WHEN NEXT CODEC INTERRUPT ARRIVES
*
        BANZ    LOOPA           ; LOOP UNTIL AR0 = 0 OR UNTIL AN INTERRUPT
*
        B       CDCERR          ; IF AR0 REACHES 0 THEN CODEC ERROR.
*
FSTINT  XOR     FLAGS
        SACL    FLAGS           ; CLEAR INTERRUPT INDICATOR FLAG
*
        LARK    AR0,INTMAX      ; LOAD AR0 WITH THE MAXIMUM NUMBER OF LOOPS
*                               ; EXPECTED BETWEEN INTERRUPTS. LOOPB IS 6
*                               ; CYCLES LONG. AWAIT NEXT INTERRUPT,
*                               ; COUNTING LOOPS WITH AR0
*
LOOPB   LAC     ONE,INTFLG
LABEL3  AND     FLAGS
        BNZ     GOTINT          ; NEXT INTERRUPT HAS OCCURRED
*
        BANZ    LOOPB           ; AUX REGISTER NOT ZERO YET. LOOP
*                               ; AUX REGISTER ZERO. INTERRUPTS TOO
*                               ; INFREQUENT
*
CDCERR  .set    $
*
        LACK    CDCFAI
        PUSH                    ; PUT CODEC INTERRUPT FAILURE RESULT ON
*                               ; STACK
        B       RESULT
*
GOTINT  SAR     AR0,ITEMP       ; CHECK FOR INTERRUPTS TOO FREQUENT
        LACK    INTLFT
        SUB     ITEMP
        BLZ     CDCERR
*
```

```
#####################################################################
*
*   IF PATH REACHES HERE, ALL THE TESTS HAVE BEEN SUCCESSFUL
*
#####################################################################
*
        LACK    PASS
        PUSH                    ; PUT TEST PASS RESULT ONTO STACK
*
#####################################################################
*
*   END OF CODEC INTERRUPT CHECK
*
#####################################################################
*
RESULT  .set    $
*
        DINT                    ; DISABLE INTERRUPTS.
        CALL    WRESET          ; DO A WARM RESET TO REINITIALIZE THE
*                               ; PROCESSOR
*
        POP                     ; RETRIEVE TEST RESULT FROM STACK
        ADDS    STMODE
        ADD     ONE,7           ; RESTORE TEST MODE BIT.
        SACL    STMODE
*
        CALL    RSTFIL          ; CALL RESET FILTERING ROUTINE TO CLEAR
*                               ; DOWN ALL ACCUMULATORS AND SET UP FILTER
*                               ; READY FOR THE NEXT BLOCK.
*
        CALL    RSDTMF          ; REINITIALIZE THE DTMF CODE
*
        LAC     STMODE,8        ; ACKNOWLEDGE THE WRITE WHICH CAUSED THE
        SACH    ITEMP           ; SELFTEST BY WRITING OUT STATUS
*
        LAC     ITEMP
        AND     MSOOFF
        SACL    ITEMP
        OUT     ITEMP,STAPRT
*
        LAC     ONE,3           ; CLEAR ALL NON-CODEC INTERRUPTS SO THAT
        SUB     ONE             ; ANY SPURIOUS FSX OR FSR INTERRUPTS
        ADDS    CTL320          ; GENERATED BY SUB-STANDARD HARDWARE WON'T
        SACL    ITEMP           ; HANG UP THW SYSTEM.
        OUT     ITEMP,CTLPRT    ; CLEAR ALL LATCHED NON-CODEC INTERRUPTS
*
        EINT
        B       MAIN            ; IGNORE THE STACK AND RESTART.
PRGEND  .word   0               ; END OF PROGRAM MARKER FOR CHECKSUM
*                               ; ROUTINE
*
        .end
```

Appendix B
PC Application Program

```
Program Tone_Detector_Demo (INPUT,OUTPUT);

{ Tone Detection Demonstration Program }

{ This program is written in Turbo Pascal and has been tested using an
  IBM PC-AT(X) with PC-DOS version 3.30 and Turbo Pascal version 3.02A

  The program implements the interface described in section 4 of this
  application report.

  The program is terminated by typing 'Q' at the PC keyboard.

  The program is included for illustrative purposes only. It may be used
  as a whole or in part for the evaluation of the tone detector. Certain
  changes may be necessary in order to ensure correct operation with a
  particular PC, operating system or Pascal Compiler.              }

{ Turbo Pascal is a registered trademark of Borland International, Inc.
  IBM is a registered trademark of International Business Machines Corp }


{ Written by   Craig Marven   May 1988     }

{$C-,U-} {Compiler directive for correct operation of Keypressed function}

Const
   ERRORX  = 1;   {X coordinate for error message     }
   ERRORY  = 24;   {Y coordinate for error message     }
   OFFSET       = $300;  {Change this if board address is not 0300h}
   WRITEBLOCK  = 23;   {Column address for write data     }
   READBLOCK  = 66;   {Column address for read data      }
   DISP : Array[1..50] of Integer = (0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,
                                     2,0,0,0,0,0,0,0,
                                     0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,
                                     1,1,0,0,0,0,0,0);
     {Cursor Y displacement for any given row position}
   FIRSTCOL : Array[5..19] of Integer = (1,1,5,5,5,5,6,5,5,5,5,5,5,5,3);
     {Initial column for each write data row   }

Type
   Eightstring = Array[1..8] of Char;
   Errorstring = Array[1..24] of Char;
   Table       = Array[5..19] of Eightstring;

Var
   SAVEX,SAVEY : Integer;      {Temporary cursor location storage }
   I         : Integer;      {Temporary variable    }
   BLOCK,COL   : Integer;
   ROW,CODE    : Integer;
   GAIN_FACTOR : Real;         {Current gain setting of tone detector}
   CMD         : Char;         {Last character input from keyboard}
   ERRFLAG     : Boolean;       {Error flag}
   TEMP_CELL   : Eightstring;  {Temporary keyboard input string}
   WRITE_TABLE : Table;         {Array containing data values for update of
tone
      detector registers}

Procedure Soft_cursor (COMMAND : Integer);
{ This procedure places, or removes a software curser at the current location
  of the screen curser. This curser remains in place while screen updates go
  on elsewhere }
Var
   CHARACTER : Char;
Begin
  Case COMMAND of
     1 : Begin
           CHARACTER := TEMP_CELL[COL];
           Textcolor(black);
           Textbackground(white);
           Write(CHARACTER);
           Textcolor(14);
           Textbackground(0);
           Gotoxy(Wherex-1,Wherey);
         End;
     0 : Begin
           CHARACTER := TEMP_CELL[COL];
           Textcolor(14);
           Textbackground(0);
           Write(CHARACTER);
           Gotoxy(Wherex-1,Wherey);
         End;
  End; {Case}
End; {Soft_cursor}

Procedure Error_message (MESSAGE : Errorstring);
{ This prcedure places a message in red (blinking) at the normal error message
  location and produces a 'bleep'}
Begin
   Gotoxy(ERRORX,ERRORY);
   Textcolor(12+BLINK);
   Write(MESSAGE);
   Sound(1100);
   Delay(500);
   Nosound;
   Textcolor(7);
End; {Error_message}

Procedure Cell_check (CELL : Eightstring ; LROW : Integer ; Var LCOL : Integer
;
                      Var ERRFLAG : Boolean);
{ This procedure checks the contents of a new input from the keyboard before
  before allowing it to be passed on to the remainder of the program. It tests
  for invalid digits, multiple decimal points, and trailing blanks, none of
  which are allowed }
Var
   I           : Integer;
   DIGIT_FOUND : Boolean;
   POINT_FOUND : Boolean;
```

```
Begin
  ERRFLAG := False;
  DIGIT_FOUND := False;
  POINT_FOUND := False;
  LCOL := 0;
  Repeat
    LCOL := LCOL + 1;
    Case LROW of
      5,6,19 : Case CELL[LCOL] of
                 '0','1' : DIGIT_FOUND := TRUE;
                 Else
                 Begin
                   ERRFLAG := True;
                   Error_message('Invalid binary digit    ')
                 End; {Else}
               End; {Case}
      7..18   : Case CELL[LCOL] of
                 '0'..'9': DIGIT_FOUND := True;
                 '.'     : If POINT_FOUND then
                             Begin
                               Error_message('Multiple decimal points ');
                               ERRFLAG := True;
                             End {If}
                             Else POINT_FOUND := True;
                 ' '     : If DIGIT_FOUND then
                             Begin
                               Error_message('Trailing blanks invalid ');
                               ERRFLAG := True;
                             End; {If}
                 Else
                 Begin
                   ERRFLAG := True;
                   Error_message('Invalid decimal digit   ');
                 End; {Else}
               End; {Case}
    End; {Case}
  Until (ERRFLAG = True) or (LCOL = 8);
  If not DIGIT_FOUND then
  Begin
    Error_message('A number must be input  ');
    ERRFLAG := True;
  End; {If}
  If ERRFLAG = False then
  Begin
    Gotoxy(ERRORX,ERRORY);
    Write('                        ');
  End; {If}
End; {Cell_check}

Function Binary_to_int (Var CELL : Eightstring) : Integer;
{ This function converts an 8 digit binary number to a decimal integer}
Var
  I,TEMP,POWER : Integer;
Begin
  POWER := 1; TEMP := 0;
  For I := 8 downto 1 do
  Begin
    If CELL[I] = '1' then TEMP := TEMP + POWER;
    POWER := POWER + POWER;
  End; {For}
  Binary_to_int := TEMP;
End; {Binary_to_int}

Procedure Outbin (INT,NUMDIG : Integer);
{ This recursive procedure outputs an integer as a binary number of any
  length}
Begin
  If NUMDIG > 1 then outbin(INT div 2,NUMDIG-1);
  Textcolor(14);
  Write(chr(INT mod 2 + 48));
  Textcolor(0);
End; {Outbin}

Procedure Zero_fill (Var CELL : Eightstring);
{ This procedure replaces leading blanks on a keyboard input with zeroes}
Var  X : integer;
Begin
  For X := 1 to 8 do If CELL[X] = ' ' then CELL[X] := '0';
End; {Zero_fill}

Function Getreg (RNUM : Byte) : Byte;
{ This function reads the current value of any tone detector register}
Var
  STATUS : Integer;
Begin
  port[OFFSET] := RNUM + 16;
  Delay(1);
  Getreg := port[OFFSET];
  port[OFFSET] := 16;
  Delay(1);
  STATUS := port[OFFSET];
End; {Getreg}

Procedure Putreg (RNUM : Byte ; VALUE : Byte );
{ This procedure puts a new value into any tone detector register}
Var
  STATUS : Integer;
Begin
  port[OFFSET] := RNUM;
  Delay(1);
  STATUS := port[OFFSET];
  port[OFFSET] := VALUE;
  Delay(1);
  STATUS := port[OFFSET];
End; {Putreg}

Procedure Update_register(LROW : Integer ; Var ERRFLAG : Boolean);
{ Update register is called by the user pressing return after entering some
```

```
  data into the program, as long as the input value is valid. Depending upon
  cursor position update register calls one of its own procedures to convert
  the user input into a format understood by the tone detector, and writes the
  new value to the tone detector}
Var
   VALUE : Integer;

Procedure Change_control(CELL : Eightstring);
{ Writes the new value for the control register into the tone detector}
Begin
  Putreg(0,Binary_to_int(CELL));
End; {Change_control}

Procedure Change_mode(CELL : Eightstring);
{ Writes the new value for the mode register into the tone detector, and up-
  dates program variable GAIN_FACTOR used in other calculations}
Begin
   Putreg(1,Binary_to_int(CELL));
   Case CELL[7] of
      '0' : GAIN_FACTOR := 4;
      '1' : If CELL[8] = '0' then GAIN_FACTOR := 1
            Else GAIN_FACTOR := 16;
   End {Case}
End; {Change_mode}

Procedure Change_env_time_constant (CELL : Eightstring ; Var ERRFLAG :
Boolean);
{ Checks for valid range of new envelope time constant - if valid writes new
  value to tone detector, if not gives error message}
Var
   TEMP : Real;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   If TEMP = 0 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else
   Begin
      VALUE := Round(1024*(1-exp(-1/(8*TEMP))));
      If (VALUE > 255) or (VALUE < 1) then
      Begin
         Error_message('Value out of range      ');
         ERRFLAG := True;
      End {If}
      Else Putreg(2,VALUE);
   End; {Else}
End; {Change_env_time_constant}

Procedure Change_thresholds (CELL : Eightstring ; THRESHOLD_TYPE : Char ; Var
                              ERRFLAG : Boolean);
{ Checks for valid range of new threshold - if valid writes new value to tone
  detector, if not gives error message}
Var
   RNUM,TEMP : Integer;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   VALUE := Round(254*(0.743*GAIN_FACTOR*TEMP)/1000);
   If VALUE > 255 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else Case THRESHOLD_TYPE of
      'U' : RNUM := 3;
      'L' : RNUM := 4;
      'C' : RNUM := 7;
   End; {Case}
   Putreg(RNUM,VALUE);
End; {Change_thresholds}

Procedure Change_filter_length (CELL : Eightstring ; Var ERRFLAG : Boolean);
{ Checks for valid range of new filter length - if valid writes new value to
  tone detector, if not gives error message}
Var
   TEMP : Integer;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   VALUE := Round((16384/(TEMP-1))-16);
   If (VALUE > 255) or (VALUE < 0) then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else Putreg(5,VALUE);
End; {Change_filter_length}

Procedure Change_passband_width (CELL : Eightstring ; Var ERRFLAG : Boolean);
{ Checks for valid range of new passband width - if valid writes new value to
  tone detector, if not gives error message}
Var
   TEMP : Integer;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   VALUE := Round(0.128*TEMP);
   If VALUE > 63 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else Putreg(6,VALUE);
End; {Change_passband_width}
```

```
Procedure Change_frequency (CELL : Eightstring ; BAND : Integer ; Var ERRFLAG
:
                            Boolean);
{ Checks for valid range of new freqency - if valid converts to the values to
  be programmed into the frequncy MS byte and LS byte registers and writes new
  values to tone detector, if not gives error message}
Var
   TEMP : Real;
Begin
   Zero_fill(CELL);
   Val(CELL,TEMP,CODE);
   If TEMP > 3400 then
   Begin
      Error_message('Value out of range      ');
      ERRFLAG := True;
   End {If}
   Else
   Begin
      VALUE := Round(8.192*TEMP) div 256;
      Putreg(8,VALUE);
      VALUE := Round(8.192*TEMP) mod 256;
      Putreg(8+BAND,VALUE);
   End; {Else}
End; {Change_frequency}

Procedure Change_filter_select (CELL : Eightstring);
{ Writes the new value for the fliter select register into the tone detector}
Begin
   VALUE := Binary_to_int(CELL);
   Putreg(15,VALUE);
End; {Change_filter_select}

Begin {Update_register}
   Case LROW of
      5      : Change_control(WRITE_TABLE[5]);
      6      : Change_mode(WRITE_TABLE[6]);
      7      : Change_env_time_constant(WRITE_TABLE[7],ERRFLAG);
      8      : Change_thresholds(WRITE_TABLE[8],'U',ERRFLAG);
      9      : Change_thresholds(WRITE_TABLE[9],'L',ERRFLAG);
      10     : Change_filter_length(WRITE_TABLE[10],ERRFLAG);
      11     : Change_passband_width(WRITE_TABLE[11],ERRFLAG);
      12     : Change_thresholds(WRITE_TABLE[12],'C',ERRFLAG);
      13..18 : Change_frequency(WRITE_TABLE[LROW],LROW-12,ERRFLAG);
      19     : Change_filter_select(WRITE_TABLE[19]);
   End; {Case}
End; {Update_register}

Procedure Update_read_value;
{ Update read value is called continuously as long as no keyboard input
  processing is pending. Update read value calls its own internal procedures
  in turn to convert values read from the tone detector into the display
  format}
Var
   I     : Integer;
```

```
Procedure Read_status;
{ Places cursor at start of status register display and outputs value read
  from tone detector}
Begin
   Gotoxy(BLOCK,5);
   Outbin(Getreg(0),8);
End; {Read_status}

Procedure Read_mode;
{ Places cursor at start of mode register display and outputs value read from
  tone detector}
Begin
   Gotoxy(BLOCK,6);
   Outbin(Getreg(1),8);
End; {Read_mode}

Procedure Read_DTMF;
{ Places cursor at start of DTMF digit display, cnverts and outputs value read
  from tone detector}
Var
   DTMF : Integer;
Begin
   Gotoxy(BLOCK,8);
   DTMF := Getreg(2);
   If DTMF > 127 then
   Begin
      DTMF := DTMF - 240;
      Textcolor(14);
      Write('OVRUN  ');
      Textcolor(0);
   End {If}
   Else
   Begin
      DTMF := DTMF - 112;
      Textcolor(0);
      Write('       ');
   End; {Else}
   Textcolor(14);
   Case DTMF of
      0..2  : Write(DTMF+1);
      3     : Write('A');
      4..6  : Write(DTMF);
      7     : Write('B');
      8..10 : Write(DTMF-1);
      11    : Write('C');
      12    : Write('*');
      13    : Write('0');
      14    : Write('#');
      15    : Write('D');
   End; {Case}
   Textcolor(0);
End; {Read_DTMF}
```

```
Procedure Read_Time (TIME : Char);
{ Places cursor at start of time display, reads and outputs present value of
  current time, departure time and arrival time from tone detector}
Var
   RESULT : Real;
Begin
   Case TIME of
     'A' : Begin
             Gotoxy(BLOCK+3,9);
             Result := Getreg(3) * 256 + getreg(4);
           End;
     'D' : Begin
             Gotoxy(BLOCK+3,10);
             Result := Getreg(5) * 256 + Getreg(6);
           End;
     'C' : Begin
             Gotoxy(BLOCK+3,11);
             Result := Getreg(7) * 256 + Getreg(8);
           End;
   End; {Case}
   If RESULT < 0 then RESULT := RESULT + 65536.0;
   Textcolor(14);
   Write(RESULT:5:0);
   Textcolor(0);
End; {Read_time}

Procedure Read_level (BAND : Integer);
{ Places cursor at start of signal level display for the required freqency
  band and outputs value read from tone detector. This procedure is called six
  times in succession with BAND incrementing from 1 to 6}
Var
   RESULT : Real;
   TEMP   : Real;
Begin
   Gotoxy(BLOCK+4,12+BAND);
   RESULT := Getreg(8+BAND);
   TEMP := (RESULT*5.3)/GAIN_FACTOR;
   If (TEMP < 4.5) then TEMP := 0;
   Textcolor(14);
   Write(TEMP:4:0);
   Textcolor(0);
End; {Read_level}

Begin {Update_read_value}
   BLOCK := READBLOCK;
   Textcolor(0);
   Read_status;
   Read_mode;
   Read_DTMF;
   Read_time('A');
   Read_time('D');
   Read_time('C');
   For I := 1 to 7 do Read_level(I);
   BLOCK := WRITEBLOCK;
   Textcolor(14);
End; {Update_read_value}

Procedure Initialise_tone_detector;
{ Sets up screen display, initial values for all tone detector registers and
  loads those registers}
Begin
   GAIN_FACTOR := 4;
   ROW := 5;
   COL := FIRSTCOL[ROW];
   Textcolor(11);
   Gotoxy(6,3);  Write('Write Registers');
   Textcolor(10);
   Gotoxy(1,5);  Write('CONTROL Register       '); Write('       b');
   Gotoxy(1,6);  Write('MODE   Register        '); Write('       b');
   Gotoxy(1,7);  Write('Envelope time constant '); Write('       mS');
   Gotoxy(1,8);  Write('Upper threshold        '); Write('       mV');
   Gotoxy(1,9);  Write('Lower threshold        '); Write('       mV');
   Gotoxy(1,10); Write('Filter length          '); Write('       samples');
   Gotoxy(1,11); Write('Passband width         '); Write('       Hz');
   Gotoxy(1,12); Write('Change threshold       '); Write('       mV');
   Gotoxy(1,13); Write('Band 1 frequency       '); Write('       Hz');
   Gotoxy(1,14); Write('Band 2 frequency       '); Write('       Hz');
   Gotoxy(1,15); Write('Band 3 frequency       '); Write('       Hz');
   Gotoxy(1,16); Write('Band 4 frequency       '); Write('       Hz');
   Gotoxy(1,17); Write('Band 5 frequency       '); Write('       Hz');
   Gotoxy(1,18); Write('Band 6 frequency       '); Write('       Hz');
   Gotoxy(1,19); Write('Filter select          '); Write('       b');
   Textcolor(11);
   Gotoxy(50,3);  Write('Read Registers');
   Textcolor(10);
   Gotoxy(45,5);  Write('STATUS Register      '); Write('       b');
   Gotoxy(45,6);  Write('MODE   Register      '); Write('       b');
   Gotoxy(45,8);  Write('DTMF Digit is        '); Write(' ');
   Gotoxy(45,9);  Write('Tone arrival time    '); Write('       mS');
   Gotoxy(45,10); Write('Tone departure time  '); Write('       mS');
   Gotoxy(45,11); Write('Current time is      '); Write('       mS');
   Gotoxy(45,13); Write('Band 1 signal level  '); Write('       mV');
   Gotoxy(45,14); Write('Band 2 signal level  '); Write('       mV');
   Gotoxy(45,15); Write('Band 3 signal level  '); Write('       mV');
   Gotoxy(45,16); Write('Band 4 signal level  '); Write('       mV');
   Gotoxy(45,17); Write('Band 5 signal level  '); Write('       mV');
   Gotoxy(45,18); Write('Band 6 signal level  '); Write('       mV');
   Gotoxy(45,19); Write('Total signal level   '); Write('       mV');
   Textcolor(14);
   Update_read_value;
   WRITE_TABLE[5] := '00000000';
   WRITE_TABLE[6] := '00100000';
   WRITE_TABLE[7] := '    1.00';
   WRITE_TABLE[8] := '     100';
   WRITE_TABLE[9] := '      50';
   WRITE_TABLE[10] := '     250';
   WRITE_TABLE[11] := '     400';
   WRITE_TABLE[12] := '     100';
  WRITE_TABLE[13] := '     500';
  WRITE_TABLE[14] := '    1000';
  WRITE_TABLE[15] := '    1500';
  WRITE_TABLE[16] := '    2000';
  WRITE_TABLE[17] := '    2500';
  WRITE_TABLE[18] := '    3000';
  WRITE_TABLE[19] := '00111111';
  For I := 5 to 19 do begin
     Gotoxy(BLOCK+1,I);
     Write(WRITE_TABLE[I]);
     Update_register(I,ERRFLAG);
  End; {For}
  ROW := 5;
  Gotoxy(BLOCK+1,ROW);
  TEMP_CELL := WRITE_TABLE[ROW];
  Soft_curser(1);
End; {Initialise_tone_detector}

Procedure Cursor_up;
{ Moves current cursor location and soft cursor up on screen. Vertical
  displacement depends upon current location, and is given by array DISP}
Begin
   If not ERRFLAG then
   Begin
      Gotoxy(BLOCK+1,ROW);
      Write(WRITE_TABLE[ROW]);
      ROW := ROW - DISP[ROW+24];
      TEMP_CELL := WRITE_TABLE[ROW];
      COL := FIRSTCOL[ROW];
      Gotoxy(BLOCK+COL,ROW);
      Soft_cursor(1);
   End {If}
End; {Cursor_up}

Procedure Cursor_down;
{ As for cursor up, but moves down}
Begin
   If not ERRFLAG then
   Begin
      Gotoxy(BLOCK+1,ROW);
      Write(WRITE_TABLE[ROW]);
      ROW := ROW + DISP[ROW+26];
      COL := FIRSTCOL[ROW];
      TEMP_CELL := WRITE_TABLE[ROW];
      Gotoxy(BLOCK+COL,ROW);
      Soft_curser(1);
   End; {If}
End; {Cursor_down}

Procedure Cursor_left;
{ Moves current curser location and soft cursor to left on screen. Cursor
  remains within value window for each parameter, given by FIRSTCOL array}
Begin
   If COL > FIRSTCOL[ROW] then
   Begin
      Gotoxy(BLOCK+COL,ROW);
      Soft_cursor(0);
      COL := COL - 1;
      Gotoxy(BLOCK+COL,ROW);
      Soft_cursor(1);
   End; {If}
End; {Cursor_left}

Procedure Cursor_right;
{ Moves current cursor location and soft cursor to right on screen. Cursor
  remains wthin value window (column 8)}
Begin
   If COL < 8 then
   Begin
      Gotoxy(BLOCK+COL,ROW);
      Soft_cursor(0);
      COL := COL + 1;
      Gotoxy(BLOCK+COL,ROW);
      Soft_curser(1);
   End; {If}
End; {Cursor_right}

Procedure Data_entry;
{ This procedure takes a keyboard numeric entry into the temporary string, and
  moves the soft cursor to the right if necessary}
Begin
   Gotoxy(BLOCK+COL,ROW);
   Soft_cursor(0);
   Write(CMD);
   TEMP_CELL[COL] := CMD;
   If COL < 8 then COL := COL +1;
   Gotoxy(BLOCK+COL,ROW);
   Soft_cursor(1);
End; {Data_entry}

Procedure Newline;
{ This procedure terminates data entry for a particular value, inputs the
  value to the cell check procedure, moves the soft cursor down, and loads the
  temporary string with the present data of the new parameter}
Begin
   Gotoxy(BLOCK+COL,ROW);
   Soft_cursor(0);
   Cell_check(TEMP_CELL,ROW,COL,ERRFLAG);
   If not ERRFLAG then
   Begin
      WRITE_TABLE[ROW] := TEMP_CELL;
      Update_register(ROW,ERRFLAG);
      If ERRFLAG then COL := FIRSTCOL[ROW]
      Else
      Begin
         ROW := ROW + DISP[ROW+26];
         COL := FIRSTCOL[ROW];
         TEMP_CELL := WRITE_TABLE[ROW];
         End; {Else}
      End; {If}
      Gotoxy(BLOCK+COL,ROW);
      Soft_cursor(1);
   End; {Newline}

   Begin
      Rewrite(OUTPUT);
      Reset(INPUT);
      Clrscr;
      Textcolor(15);
      CMD := ' ';
      Gotoxy(20,1);
      Writeln('TONE DETECTION DEMONSTRATION PROGRAM');
      Initialise_tone_detector;
      Repeat
         If KeyPressed then
         Begin
            Read(kbd,CMD);
            If (CMD = #27) and KeyPressed then
            Begin
               Read(kbd,CMD);
               Case CMD of
                  #72 : Cursor_up;
                  #75 : Cursor_left;
                  #77 : Cursor_right;
                  #80 : Cursor_down;
               End; {Case}
            End; {If}
            Case CMD of
               '0'..'9','A'..'F','a'..'f',' ' : Data_entry;
               '.'     : If ROW = 7 then Data_entry;
               #13     : Newline;
            End; {Case}
         End {If}
      Else Update_read_value;
      Until (CMD = 'Q') or  (CMD = 'q');
      Clrscr;
   End.
```

Appendix C

Power Detector Operational Considerations

C. 1 Arrival and Departure Time Skew

The use of a large time constant τ for the envelope decay factor results in a greater time delay between the appearance or departure of a tone and the envelope detector output crossing a threshold. See the section ''Envelope Decay Factor'' for details of the envelope decay detector.

If the signal level on the line changes to S at time $t = 0$ when the output of the envelope detector is EI, we may determine the time t taken for the envelope detector output to reach a pre-defined threshold level L as follows:

$$t = -\tau \times \ln[(S-L) / (S-EI)]$$

For example, at time $t = 0$, the output of the envelope detector $EI = 0$ and the upper (arrival) threshold level is set to -20 dBm, a signal appears on the line at a level of -10 dBm.

Therefore, when $t = 0,\ EI = 0$

$$S = 10^{(-10/20)} = 0.3162$$

$$L = 10^{(-20/20)} = 0.1$$

This gives a value of $t = 0.38\tau$ for the envelope detector output to cross the arrival threshold.

In the case for tone departure at time $t = 0$, the output of the envelope detector is stable at -10 dBm, and the lower (departure) threshold is set to -20 dBm, the signal on the line departs.

Therefore, when $t = 0,\ S = 0,\ L = 0.1$ as before

$$EI = 10^{(-10/20)} = 0.3162$$

This gives a value of $t = 1.15\tau$ for the envelope detector output to cross the departure threshold.

The skew between the delays for the envelope detector to cross the arrival and departure thresholds is thus 0.77τ in this instance. This skew obviously increases as the envelope factor τ is increased.

C.2 Sampling Frequency Considerations

Due to the envelope detector assessing the level of the signal on the line by rectification and smoothing, anomalies arise in its behavior when the signal being rectified is a pure tone at a frequency which is an integer sub-multiple of the sampling frequency of 8 kHz. This effect is most significant at 2 kHz, where the signal level assessed by the envelope detector may vary over the range of +0.91 dB to −2.1 dB relative to the true signal level. If the output signal differs from 2 kHz by a very small amount, the envelope detector output will vary over the above range as the phase of the signal slides past the sampling instants. Ensuring a hysteresis band between the upper and lower thresholds of at least 3 dB will avoid the possibility of a series of apparent tone arrivals and departures in the presence of a steady pure tone.