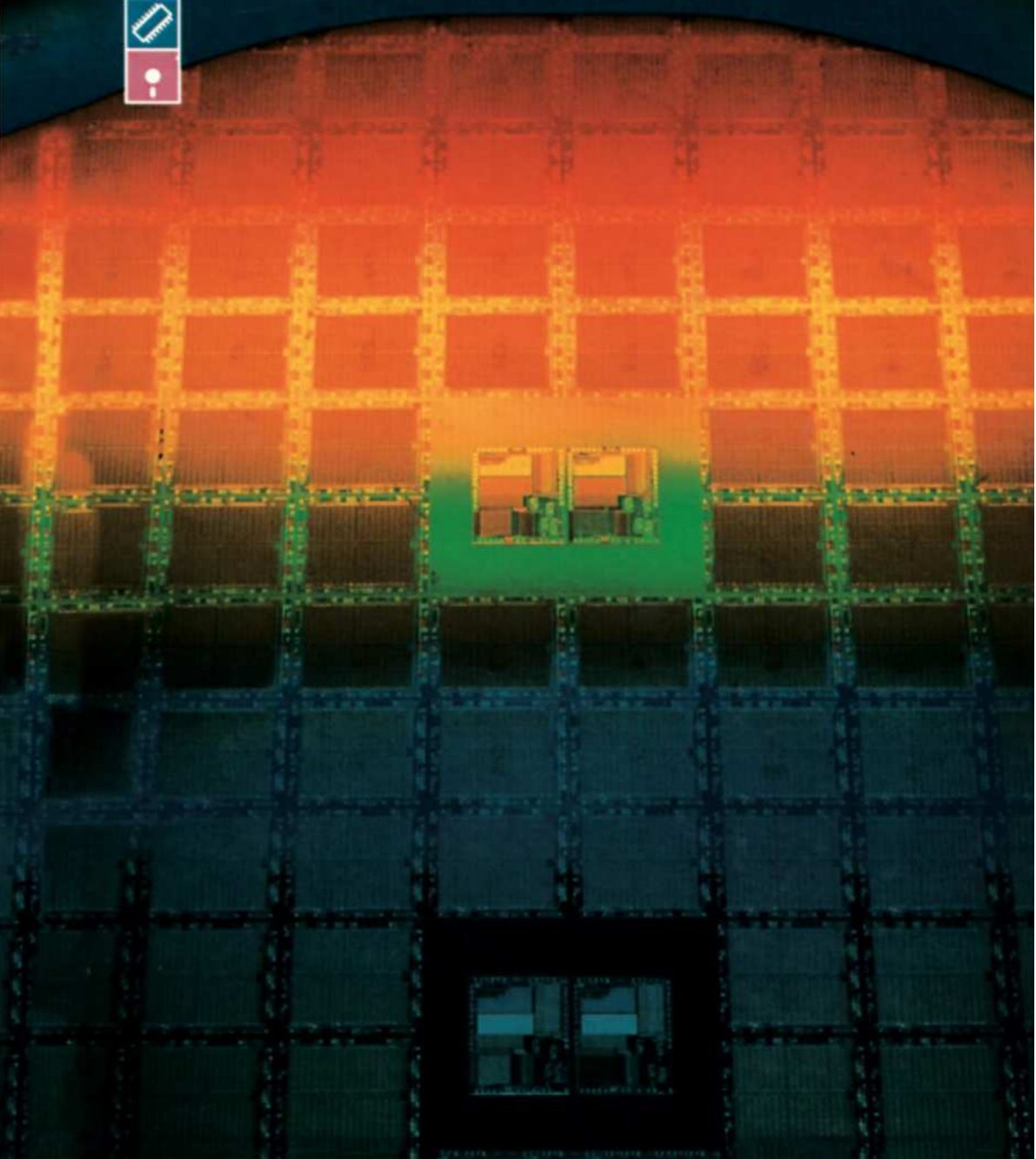




# DESIGNING WITH SPEECH PROCESSING CHIPS

Ricardo Jiménez





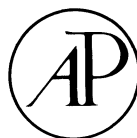
# Designing with Speech Processing Chips

---

Ricardo Jiménez

*BESTNET Telecommunications*

*Calexico, California*



**ACADEMIC PRESS, INC.**

Harcourt Brace Jovanovich, Publishers

San Diego New York Boston London

Sydney Tokyo Toronto

This book is printed on acid-free paper. (∞)

Copyright © 1991 by ACADEMIC PRESS, INC.

All Rights Reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher.

ACADEMIC PRESS, INC.

San Diego, California 92101

*United Kingdom Edition published by*

Academic Press Limited

24–28 Oval Road, London NW1 7DX

Library of Congress Cataloging-in-Publication Data

Jiménez, Ricardo.

Designing with speech processing chips / Ricardo Jiménez.

p. cm.

ISBN 0-12-385348-6

1. Integrated circuits--Design and construction--Data processing.

2. Speech processing systems. 3. Computer-aided design. I. Title.

II. Title: Speech processing chips.

TK7874.J56 1991

621.39'9--dc20

90-49691

CIP

PRINTED IN THE UNITED STATES OF AMERICA

91 92 93 9 8 7 6 5 4 3 2 1

*This book is dedicated  
to my wife Patricia*

# Preface

The speech processing chip is a relatively new and complex device that appeared in the late 1970s. This book provides the theory and the basic design tools needed to utilize speech processing chips more effectively in electronic circuit design. It presents design examples for a wide range of real-world applications and information on interconnection of these components into functional equipment for instrumentation, data processing, inventory display, and control systems. Special emphasis is placed on those circuits with the most potential for future development, LSI and VLSI devices. Popular commercially available products are used throughout as illustrative examples, and the important characteristics of the devices are summarized for each functional category.

The book goes far beyond the presentation of block diagrams, micro-controller architecture, and software programming. It shows a step-by-step development of hardware and software, how to combine them most effectively, and how to interface speech processors with input devices, such as sensors or data sources, and output devices, such as relay actuators, thyristors, or display devices.

In this book, practicing design and system engineers, technicians, engineering students, and other interested readers will find a comprehensive overview of the entire topic of speech processing chips. The book also describes popular, commercially available circuits for each functional category presented and discusses specific applications in sufficient depth to interest the experienced designer. Engineering students will be able to follow the book if they have been exposed to courses on circuit design theory, logical circuits, and integrated circuits.

As with my other publications, I wanted this to be a book that was organized

and used from a practical viewpoint. Throughout the book emphasis is placed on using the popular CMOS and HCMOS ICs of each functional category as illustrative examples.

Speech processing chips for electronic and other applications are available at low cost. The progress made in the manufacture and supply of these chips expands enormously the opportunities to design and build highly effective equipment and systems. This book shows how to make use of these developments. The reader is shown step-by-step how to build both simple and sophisticated projects with all the necessary details. Many proven examples are included throughout the book for industrial, laboratory, health care, and home use.

The information needed to follow the many design examples in this book is given in a simple, direct manner with supporting flowcharts and tables. Once this know-how is acquired, you will be able to build systems with artificial voice with less effort and less time than ever before.

The book is divided into seven chapters. Chapter 1 introduces the different speech processing techniques, describes how the basic speech processor integrated circuit works, and presents IC pin comparisons of the different packages. It also includes the basic datasheet for the device.

Chapter 2 explains how a speech processor can be used in applications for which it was not originally intended—how this basically digital device can be used as a variety of different logic devices. Chapter 3 provides help understanding analog-to-digital converter families and their respective advantages and limitations. After reading these three chapters, you should come away with a fundamental understanding of what a speech processor is and how to use it.

To write information into the specific controller and then announce it, interface devices or systems are required. Chapter 4 shows how such interface devices can be built with minimal effort and at a cost that is often a small fraction of the price of commercial products.

Chapter 5 presents a wide selection of test and measurement circuits that also can be interfaced with a specific application by the user or designer. These circuits are widely used in data acquisition systems. Chapter 6 presents different kinds of burglar alarms varying from simple designs to fault-tolerant systems where failures are critical and not acceptable.

Chapter 7 covers voice recognition techniques as well as devices now available. Some applications for control systems are also considered.

# Acknowledgments

I owe thanks to several people who helped me complete this book.

I would like to thank all the publishers who gave permission to include material that originally appeared elsewhere. Footnotes to selected articles cite these original sources.

I am grateful to those instructors from other institutions who reviewed the manuscript for Academic Press, Inc., and who made many constructive suggestions.

Thanks also to my research students of the courses “Logical Circuits II,” “Sequential Systems,” and “Instrumentation” who suffered through the building and testing process of many of the circuits presented here and in particular to Ruben Rios, Francisco Meza, Jose J. Lara, Ramiro Jacinto, Ruben Avalos, Francisco J. Mendoza, Alvaro E. Salgado, Julio Garcia, Rigoberto Hernandez, and Fabian Muro.

Thanks in particular goes to the Dean of the Technological Institute of Mexicali, Ramon A. Heredia.

I am also grateful to Marcos Silva, Roberto Dewar, and Dr. Fidel Diaz for proofreading the manuscript to make it more understandable. Of course, the patience and love of my wife Patricia was required throughout.

*Ricardo Jiménez, E. E.*

# *Speech Processing Chips*

## **1.1** Introduction to Voice Synthesis Digital ICs

Circuits with artificial voice offer a new dimension of sophistication to almost any electrical or electronic modern system. Traditionally, magnetic tape recording has been used in applications requiring speech announcements, for example, telephone announcement systems; a system of this type is costly because it requires a large number of tapes for different messages. It will not let you create mixed messages for different situations. Consider the case of a public service telephone that tells you the time. This device will need 60 different tapes for each hour, not to mention the number of tapes required for a complete 24-hour day. In contrast, a telephone system that uses artificial voice stored in digital memories can create different messages by pulling up the different words required to create a specific message. This system requires only a few chips that will do the work of a large number of tapes.

Let us now consider a talking voltmeter in the range of 0 to 5 V with a resolution of 0.1 V. Here, we will need 50 different messages corresponding to the 50 possible voltage readings. It would be costly and time-consuming to develop a tape mechanism for this project.

In the past, speech systems were treated as data acquisition circuits, in which a voice waveform was treated like any other fluctuating voltage input: The circuit recorded the waveform by periodically taking a sample of the signal's voltage through an analog-to-digital (A/D) converter and storing it as a binary value. (The number of samples needed per second depends upon the frequency of the input signal.) These digital speech signals were stored as pulse code modulation (PCM) in semiconductor memories. Once the samples



were stored in RAM or ROM, the circuit could recreate the original waveform by sequentially sending the stored values to a D/A converter at the same rate as the original sampling. One second of digital voice required from 4 to 32 Kbytes of memory. If the amount of data stored was reduced (compressed) using a known principle, the restoration of the original sound was called "synthesis."

The synthesis technique provides a dramatic reduction in the amount of memory required for one second of speech. Memory requirements vary from 400 to 2,000 bits per second, depending on the desired speech attributes and overall quality. A bad reproduction will sound unnatural or unintelligible. A speech signal is highly redundant and predictable, and by coding only the slowly varying coefficients of speech or by dramatic compression of digitized speech, significant bandwidth reductions in the digitized signal can be obtained. The synthesizer technique becomes practical when it is developed with VLSI semiconductor technology.

Today, applications for voice synthesis are endless. The following are some of them: telecommunications; consumer appliances; automotive; counters; consumer products; instrumentation; teaching aids; clocks; language translation; annunciators; voice interactive computer terminals; nautical and aeronautical instrumentation annunciators; voice back units for banking, weather, and time announcements; elevators; trains; subway systems; toys and games; warning systems for fire and police emergency.

In the area of instrumentation, a speech synthesizer is a very important tool. When a failure is presented in the system being monitored, the speech synthesizer will immediately start reading the procedures contained in the manual in order to indicate to the operator how to correct the specific failure. Great benefits are obtained if a speech synthesizer is installed in power stations, nuclear plants, or places where the user must monitor a myriad of controls. Here the speech synthesizer augments the operator's ability to respond rapidly and correctly when a process has extended its normal limits.

In industry, a speech synthesizer can be used to augment productivity by giving spoken messages on how to assemble specific products, thereby freeing a user for other tasks. Here, failure to follow precise directions could lead to the destruction of equipment or injury to personnel.

The use of vocal warnings on automobiles has been spreading since the early 1980s to remind the driver of the electrical or mechanical situation of the vehicle. The same features are also applied to airplanes where the synthetic speech guides the pilot with directions such as "slow down," "climb," or other appropriate instructions.

The pace of the speech processing field is so rapid that some systems now under development are excluded from this book. The emphasis of this book is on designing with the systems now available.

## 1.2 Synthesis Techniques

The basic phonological element of speech is the phoneme, which is the name given to a group of similar sounds in language. A phoneme is acoustically different depending upon its position within a word. Each of these positional variants is an allophone of the same phoneme. Phoneme reproduction is a basic element in any speech synthesizer. The method of “allophone speech synthesis” is used to create words or phrases where the user has to think in terms of sounds, not letters. With this technique you can synthesize an unlimited vocabulary by using allophones and silences in the appropriate sequence. Phonemes, together with speaker inflection and volume, are the fundamental building blocks of speech.

The American English language consists of approximately 38 to 40 phonemes: 14 to 16 vowel sounds and 24 consonant sounds. For example, the initial K sound used in words like “comb” sounds slightly different from the Ks in words like “can’t.” These small variations are due to the vowel which follows them, in this case, “o” and “a.” Each phoneme is generated with either a voiced sound, as in “eye” or an unvoiced sound like the “sh” in “shy.” There are also allophones classified as resonants, voiced fricatives, voiceless fricatives, voiced stops, voiceless stops, affricates, and nasals.

Voice synthesis methods are divided into three major types: waveform encoding, parametric synthesis, and synthesis by rule. Each method is explained below.

### *Waveform Coding Methods*

This type of voice synthesis includes differential pulse code modulation (DPCM), adaptive delta modulation (ADM), and adaptive differential PCM (ADPCM). The original sound wave amplitude is sampled at fixed intervals, digitized, and the volume of data is then reduced on the basis of the synthesis principles.

### *Parametric Synthesis Methods*

Characteristic information included in voice waveforms is extracted as parameters for synthesizing purposes. The partial autocorrelation (PARCOR) method is a typical example. In this method, models of the human vocalization mechanism are used. Voiced and voiceless consonant sounds are discriminated, and voiced sound pitch and amplitude data are extracted together with filter characteristics of the vocal tract. Voice synthesis is then obtained by passing these data to hardware consisting of digital filter circuits.

### *Synthesis by Rule Method*

In this synthesis method, groups of phonemes expressed by small quantities of data are skillfully linked together to reproduce any desired words or phrases,

which makes it easy to develop your own set of words or phrases for your specific application. However, this method lacks the flexibility to create intonation, accents, and length of certain sounds in order to get a natural-sounding voice. This method will be more efficient when vowels and diphthongs with different accents are contained in the allophone set.

There are two general approaches used to derive synthetic speech: (1) time domain synthesis and (2) frequency domain synthesis. The first method works with a synthetic speech waveform representation of the original speech. About half of the synthetic waveform is silence and is made up of symmetric segments which range over a very restricted set of amplitude values. In this form, the synthetic waveform can be stored using only 1% of the bits that are necessary to reproduce the original speech waveform.

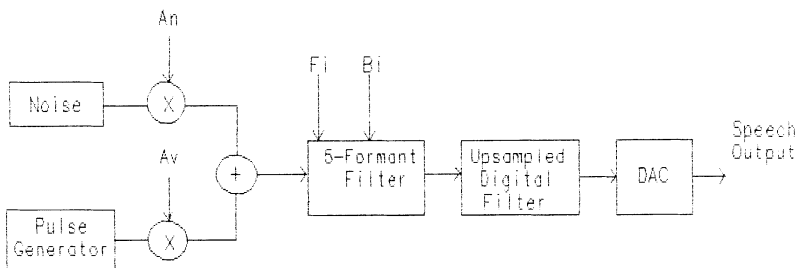
Frequency domain synthesis has two main branches: formant synthesis and linear predictive coding (LPC). Formant synthesis generates speech by reproducing the spectral shape of the waveform using the formant center frequencies, bandwidths, and the pitch periods as inputs. A frequency region where the amplitude of a vowel sound is concentrated (i.e., frequency peaks in the voice spectrum) is known as formant. Figure 1.1 shows an electronic speech model of the human speech production mechanism. This model is used in the Signetics speech synthesizer FPC8200.

LPC is based on a mathematical model of the human vocal tract. Pitch, amplitude, and speech variables are obtained from speech recordings. The speech data are analyzed and encoded to reproduce input data suitable for the digital model.

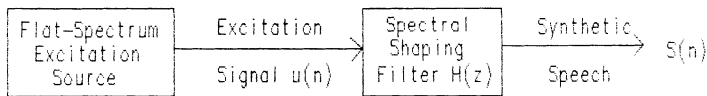
The basic model used in linear predictive analysis is illustrated in Figure 1.2. The two major components are a flat-spectrum excitation source and a spectral shaping filter  $H(z)$ .

The excitation source provides a signal  $u(n)$  containing a flat spectral envelope that is used to drive the filter  $H(z)$ , resulting in the synthetic speech output signal  $S(n)$ . Because the excitation signal has a flat spectrum, the spectral envelope of the output signal  $S(n)$  will have the same shape as the spectrum of the filter  $H(z)$ .

In speech synthesis, the parameters of  $H(z)$  must be set on a time-varying



**Figure 1.1** Electronic speech model of the human speech reproduction mechanism.



**Figure 1.2** Speech synthesis model.

basis such that its short-term spectrum is the same as that of the desired short-term speech spectrum envelope.

### 1.3 SPO256B/SPO264 Narrator Speech Synthesis Processors

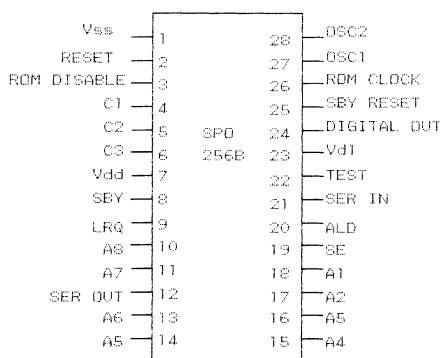
The SPO256B/SPO264 speech processors use the method of synthesis by rule. They also support LPC synthesis, formant synthesis, and allophone synthesis. Phoneme synthesis works by combining basic sound elements (phonemes) to form complete words and sentences. This method is suitable when the type of vocabulary required is not fixed.

Microchip (2355 W. Chandler Blvd., Chandler, AZ 85224-6199), the only manufacturer of SPO256B/SPO264 speech processors, uses the approach just described, combining phoneme synthesis with a digital filter. The speech processors from Microchip employ formant coding, which is a frequency domain synthesis that is similar to LPC. These devices model speech as the output of a series of cascaded resonators.

The speech process is initiated by addressing the ROM address that contains the phoneme desired. A maximum of 256 phonemes can be stored in the 16 Kbits of internal ROM. This device contains a microcontroller and a vocal-tract model. The vocal-tract model is a digital filter. These processors are classified as narrators because they use a preprogrammed custom vocabulary, or phrase set, which is recorded in a serial ROM SPR128 or SPR128B. One of these memories is directly interfaced with the speech processor (SP). If a parallel ROM is desired for recording the phrase set, a parallel-to-serial interface is required. This is made by using the SPR000, as we will see in Section 1.6.

The SPO256B contains an on-chip controller with 16 Kbit internal ROM, while the SPO264 has a 64 Kbit memory. Both devices support a controller for external ROM SPR128A, which is a 128 Kbit ROM. These processors incorporate four basic functions:

- A software programmable digital filter that can be made to model a vocal tract.
- A 16K ROM which stores both data and instructions.
- A microcontroller which controls the data flow from the ROM to the digital filter.
- A pulse width modulator that creates a digital output which is routed to an external low-pass filter in order to get an analog signal.



**Figure 1.3** Pin configuration of the SPO256B/SPO264 speech processors.

When amplified, this analog output signal will drive a loud speaker. The digital output is equivalent to a flat frequency response varying from 0 to 5 kHz, a dynamic range of 42 dB, and a signal-to-noise ratio of approximately 35 dB.

A nice feature of these synthesizers is the natural speech, and the external ROM directly expandable to a total of 480 K. A version of the SPO256B with internal preprogrammed ROM is the SPO256-AL2.

Figure 1.3 shows the pin configuration of the SPO256B, which is identical to the SPO264.

### Operation<sup>1</sup>

The addressing of the SPO256B is controlled by the address pins (A1–A8), address load (ALD), and strobe enable (SE). Strobe enable controls the two modes available for loading an address into the chip.

**Mode 0 ( $SE = 0$ ):** The SP latches an address when any one (or more) address pin makes a low-to-high transition. All address lines must be returned to zero prior to entering a new address. In this mode of operation, address zero (0000 0000) is not a valid address input. This mode of operation is used in applications consisting of no more than eight words or messages such that single address line transitions can be made. These words or messages must be stored using the binary address format 1, 2, 4, 8, 16, . . . , 128.

**Mode 1 ( $SE = 1$ ):** The SPO256B will latch an address specified on the address bus (A1–A8) when the /ALD pin is pulsed low. Any address varying from 0 to 255 can be loaded using this mode. Specific setup and hold times are required using this mode.

In order to interface the SP with microprocessors ( $\mu P$ ) or microcontrollers ( $\mu C$ ), two interface pins are available. They are load request (/LRQ) and

<sup>1</sup>This section and the following section, *Test Modes*, are adapted from Publication #DS5018A-2, p. 3. © 1988 Microchip Technology, Inc.

standby (SBY). /LRQ indicates when the input buffer is full. SBY tells the specific processor that the chip has stopped talking and no new address has been loaded. When /LRQ is low, a new address may be loaded onto the address bus. Pulsing /ALD low will cause the new address to be loaded and /LRQ to go high. When the address bus is available to accept a new address, /LRQ goes low again.

The SPO256B can load a new address while it is speaking the last word or message.

Standby (/SBY) goes low when an address is loaded and stays low while the chip is talking. SBY can be used to determine the time between address load requests, which will be variable depending on the length of the word or message currently being spoken.

Several pins are designated for use with an external ROM. They are ROM Disable, C1, C2, C3, Serial Out (SER OUT), Serial In (SER IN), and Test. ROM Disable is tied to a logic zero to enable the external ROM; when left floating, it disables the external ROM. C1 to C3 are the output control lines which are synchronous to the ROM clock. ROM CLOCK (pin 26) is a 1.56 MHz clock output used to drive an external serial speech ROM. The operating frequency of the SPO256B is set by an external 3.12 MHz crystal connected to OSC1 and OSC2 (pins 27 and 28), respectively. You can also use 3.27 to 3.14 MHz crystals; in this case the voice pitch will be slightly altered.

When C1C2C3 = 000, no operation is executed. When C1C2C3 = 001, the address shift register load (ASRL) serially shifts out data from the SER OUT pin as shown in Figure 1.6. The ASRLD loads 16 bits of the ASR with two 8-bit load sequences followed shortly by a program counter load (PCLD).

When C1C2C3 = 010, the contents of the address shift register are loaded into the program counter (PC) when 16 ASR loads have occurred.

When C1C2C3 = 011, the data shift register loads the 8-bit data shift register with the contents of the ROM pointed to by the current address in the program counter. The data shift register will shift out the LSB of the 8 bits and increment the program counter. With C1C2C3 = 100, data is shifted out of the data shift register starting with the second LSB (the first LSB is shifted out with the occurrence of C1C2C3 = 011). Seven shifts occur after every DSRLD.

When C1C2C3 = 101, the stack is loaded with the current value of the PC. With C1C2C3 = 110, the PC is loaded with the contents of the stack to perform the RETURN operation. Finally, C1C2C3 = 111 will occur when /SBY RESET and /RESET are pulsed low.

### *Test Modes*

By using the TEST logically anded with the address inputs A1, A2, A3, or A5, the SPO256B can be interfaced to an SPRO000 (serial-to-parallel ROM) in order to use an EPROM as the speech data device.

The test modes are controlled by the TEST pin (22). This is achieved by

making the TEST pin high within the appropriate address input. The following is a list of test modes:

$$T0 = T * (A1 = A2 = A3 = A5 = 0)$$

$$T1 = T * A1$$

$$T2 = T * A2$$

$$T3 = T * A3$$

$$T5 = T * A5$$

T0 makes the SPO256B ignore its internal ROM. In this case, instructions are supplied serially through pin 21 (SER IN) by using an SPR000. T0 may be entered by wiring TEST to a permanent high and pulsing /RESET. T0 will be entered and all other test modes permanently locked out. Note that the voltage levels on any address input are irrelevant when this mode is entered.

T1 is used to read out the internal ROM of the SPO256B on pin 12 (SER OUT).

T2 causes the internal serial data bit stream to appear on pin 24 in place of the normal DIGITAL OUT.

T3 is used with T2. It causes the internal coefficient data ROM to be read out if the inputs to pin 21 are appropriate.

T5 causes the internal CE (chip enable) signal to come out on pin 9 (/LRQ)

### *Electrical Characteristics*

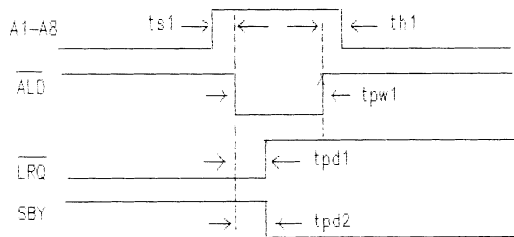
Vdd (pin 7) is the power supply for all portions of the SP except the microprocessor interface logic. VD1 (pin 23) is the power supply for the microprocessor interface logic and controller. When not tied to Vdd, VD1 must remain high. When the SPO256B is not talking, it is better to put it in the low-power mode by disconnecting Vdd (pin 7) via a switching relay or transistor.

The SPO256B consumes a total current of 80 to 95 mA in the normal mode of operation, and a supply current of 25 to 29 mA in the standby mode. Both modes are required to hold high the /RESET and /SBY inputs.

The /ALD input must be pulsed low for a period of 200 to 960 nanoseconds; consequently, 500 ns (or 0.5  $\mu$ s) will be the standard pulse period we'll be using in most of the projects presented here. The address pins (A1–A8) need a minimum period of 160 ns, while /LRQ and SBY use a maximum of 300 ns. These times must be added to the delays caused by the external peripherals that will be controlling the SPO256B in order to avoid pulsing the /ALD input when the SPO256B has not yet received the specific speech address. Glitches in the address bus also must be considered.

The timing diagram in Figure 1.4 presents the four most important functions that must be controlled. They are A1–A8, /ALD, /LRQ, and SBY.

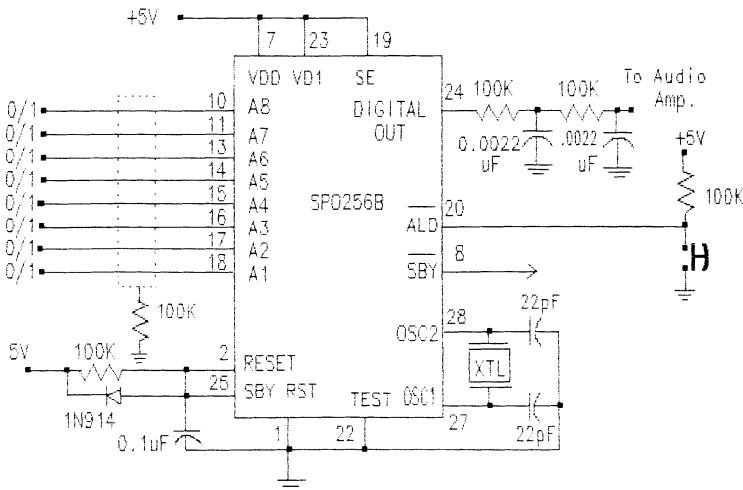
The SPO256B can be tested in a stand-alone configuration prior to starting further developments with microprocessors or data acquisition systems. The



**Figure 1.4** SPO256B timing diagram for the functions /ALD, /LRQ, and /SBY when an address input (A1–A8) is present.

low-pass filter is formed by two 100K resistors and two 0.0022  $\mu$ F capacitors; the output will drive a loudspeaker or headphones when connected to an audio amplifier. Depending on the applications of the user, the power of such an operational amplifier can be variable.

In Figure 1.5, XTL is a 3.12 MHz crystal. The resistor connected from pin 2 to +5 V in series with the 0.1  $\mu$ F capacitor forms an RC timing network that produces a negative transient pulse when power is turned on. Diode 1N914 is necessary only if power is turned off and then on in less than 50 ms. In order to test this circuit, the user must set a specific address by setting the eight switches on or off. Address “9,” for example, is given with 00001001. Thus switches 1 and 4 must be closed in order to give a logic one to the address inputs A1 and A4. The normally open switch connected to /ALD (pin 20)



**Figure 1.5** Stand-alone configuration for the SPO256B.

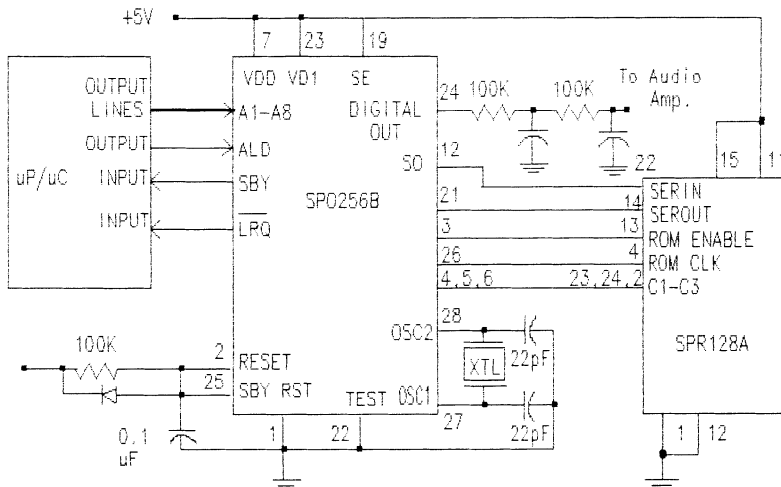


must be momentarily closed. This will cause a negative transient pulse on /ALD input. The SPO256B will speak the first word or message that is recorded in its internal ROM. You can evaluate the complete vocabulary by setting the 256 possible binary combinations at the address inputs. Take into account that the power supply must be capable of giving off a minimum current of 100 mA.

For microcomputer interface, the circuit in Figure 1.6 shows a typical microcontroller (Intel 8748). In this case, the setup and hold times must be complied with by the microcontroller because the speech processor is a relatively low-speed device. Data acquisition systems will be very useful with this configuration in applications where speed is not an important factor. For high-speed circuits it will be necessary to use fast microsequencers, such as Amd29CPL151/152/154. These devices work with one clock cycle per instruction, and the maximum permissible frequency is 30 MHz.

For typical  $\mu P/\mu C$  interface applications and where extensive vocabularies (normally more than 32 words or phrases) are necessary, an external ROM SPR128A is required. In this case, linear predictive coding is supported by the SPO256B with the advantage of having a very natural male/female voice. Because the ROM consumes about 20 mA, and considering the 90 mA consumed by the SPO256B, a 5 V power supply with more than 250 mA is required for the circuit shown in Figure 1.6.

The  $\mu P/\mu C$  programs for controlling the address bus, /ALD, SBY, and /LRQ lines must be capable of monitoring the standby (/SBY) and load request (/LRQ) inputs in order to send the desired recorded binary data from the



**Figure 1.6** SPO256B interfaced with a microprocessor/microcomputer.

$\mu C/\mu P$ . Any 8-bit microprocessor or microcontroller will be suitable for the tasks mentioned. Details about the serial speech ROM SPR128A are described in Section 1.5.

## 1.4 SPO256-AL2 Allophone Speech Synthesis Processor

The SPO256-AL2, manufactured exclusively by Microchip, is a single chip speech processor that is preprogrammed with a ROM pattern containing 64 allophones. An unlimited vocabulary in the English language can be achieved by concatenating user-selected allophones. It is also possible to construct a limited vocabulary in the Spanish language. The only constraints to obtaining an unlimited vocabulary are the Spanish “r” and “ñ” sounds which do not exist in the American English language. This problem can be partially solved by using synonyms of the selected word.

Fifty-nine discrete speech sounds and five pauses are stored at different addresses in the SPO256 internal ROM. Addressing these 64 locations requires six address bits; A1 to A6. Address inputs A7 and A8 will remain grounded for this specific chip.

### *Allophones*

The phoneme is the basic unit of distinctive sound. A phoneme can represent different sounds, depending upon its position within a word. Each of these positional variants is an allophone of the same phoneme. This method is called “allophone speech synthesis.” Certain allophones can vary in duration; for example, in the word “four” the “f” sound is long compared with the “f” sound of the word “fit.”

### *Phonemes of English*

Each language has a set of phonemes which is slightly different from that of other languages. Table 1.1 shows the allophone set contained in the SPO256-AL2 internal ROM.

## 1.5 Serial Speech ROM SPR128A

The SPR128A, manufactured exclusively by Microchip, interfaces directly to the SPO256A speech processor to provide vocabulary expansion. A maximum of four SPR128s can be interfaced to the SPO256B without buffering. The SPR128A is a mask programmable ROM providing 16 Kbytes of memory. The operating voltage is 4.5 to 7 V. Inputs and outputs are TTL compatible and the serial output has tri-state capability. This low power device consumes only 20 mA and can be powered down when the system is inactive.

The SPR128A is addressed by an internal program counter (PC). The serial in/parallel out (SIPO) shift register, denoted as ASR, is used to assemble an

**TABLE 1.1**  
Allophone Address Table for the SPO256-AL2

Decimal address	Hexadecimal address	Allophone	Sample word	Duration (milliseconds)
0	00	PA1	PAUSE	10
1	01	PA2	PAUSE	30
2	02	PA3	PAUSE	50
3	03	PA4	PAUSE	100
4	04	PA5	PAUSE	200
5	05	/OY/	Boy	420
6	06	/AY/	Sky	260
7	07	* /EH/	End	70
8	08	/KK3/	Comb	120
9	09	/PP/	Pow	210
10	0A	/JH/	Dodge	140
11	0B	/NN1/	Thin	140
12	0C	* /IH/	Sit	70
13	0D	/TT2/	To	140
14	0E	* /RR1/	Rural	170
15	0F	/AX/	Succeed	70
16	10	/MM/	Milk	180
17	11	/TT1/	Part	100
18	12	/DH1/	They	290
19	13	/IY/	See	250
20	14	/EY/	Beige	280
21	15	/DD1/	Could	70
22	16	/UW1/	To	100
23	17	* /AO/	Aught	100
24	18	* /AA/	Hot	100
25	19	/YY2/	Year	180
26	1A	* /AE/	Hat	120
27	1B	/HH1/	He	130
28	1C	/BB1/	Business	80
29	1D	/TH/	Thin	180
30	1E	* /UH/	Book	100
31	1F	/UW2/	Food	150
32	20	/AW/	Out	370
33	21	/DD2/	Do	160
34	22	/GG3/	Whig	140
35	23	/VV/	Vest	190
36	24	/GG1/	Got	80
37	25	/SH/	Ship	160
38	26	/ZH/	Azure	190
39	27	/RR2/	Brain	120
40	28	/FF/	Food	150
41	29	/KK2/	Sky	190
42	2A	/KK1/	Can't	160
43	2B	/ZZ/	Zoo	210
44	2C	/NG/	Anchor	220

45	2D	/LL/	Lake	110
46	2E	/WW/	Wool	180
47	2F	/XR/	Repair	360
48	30	/WH/	Whig	200
49	31	/YY1/	Yes	130
50	32	/CH/	Church	190
51	33	/ER1/	Starter	160
52	34	/ER2/	Beer	300
53	35	/OW/	Close	240
54	36	/DH2/	They	240
55	37	/SS/	Vest	90
56	39	/NN2/	No	190
57	39	/HH2/	Hoe	180
58	3A	/OR/	Store	330
59	3B	/AR/	Alarm	290
60	3C	/YR/	Clear	350
61	3D	/YY2/	Guest	40
62	3E	/EL/	Saddle	190
63	3F	/BB2/	Business	50

(Reprinted with permission from Publication #DS5005A-1. © 1984 Microchip Technology, Inc.)

address to be loaded in parallel into the program counter. The serial input (SERIN) is used to synchronously load the ASR register. The contents of the program counter are loaded into a 16-bit return register and can be restored later. This feature allows the device to return when a JUMP to a different address is performed. The program counter points to a specific address in the ROM which gives out parallel data into a shift register (DSR). The DSR shifts out the ROM data to the serial output pin.

The ROM ENABLE input is an active low select that tri-states the serial out when brought high. It is used to avoid bus conflict on the serial out pin during SPR128A power up. Chip select input CS1 is used to tri-state the serial output when low. An internal pull-up resistor permits you to leave the pin unconnected if not being used. /CS2 is an active low select that tri-states the serial output pin when high. It also contains a pull-down resistor that allows the pin to float when unconnected. The ROM clock input receives a frequency of 1.56 MHz from the SPO256B speech processor. The function of the SPR128A to be executed is determined by control pins C1, C2, and C3. A block diagram of SPR128A is shown in Figure 1.7.

The control states of inputs C1, C2, and C3 are now explained. The 16 bit address specified by the program counter uses the two upper bits (A14, A15) to select the action to be performed.

Up to four SPR128s can be interfaced directly to SPO256B without buffering. Figure 1.8 shows how to interface and control two SPR128s using either a  $\mu$ P or a  $\mu$ C. Figure 1.8 also illustrates the interface of two SPR128As to the SPO256 speech processor.



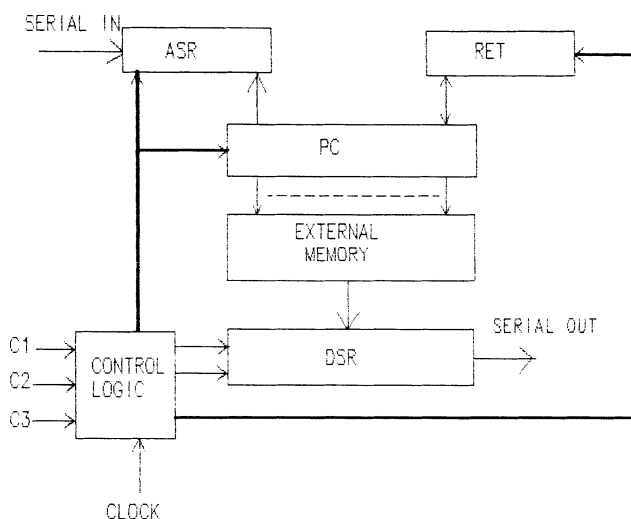
## 1.6 SPR000 Parallel-to-Serial Speech Interface Chip

The SPR000, manufactured by Microchip, contains all necessary logic for data communication between standard ROM, PROM, or EPROM to the SPO256B speech processor. The communication protocols are controlled by the speech processor. The SPR000 is suitable for SPO256B testing and speech ROM emulation. Typical testing consists of evaluating a custom vocabulary by using, for example, 27C16/32/64/ EPROMs. Figure 1.9 shows the SPR000 block diagram.

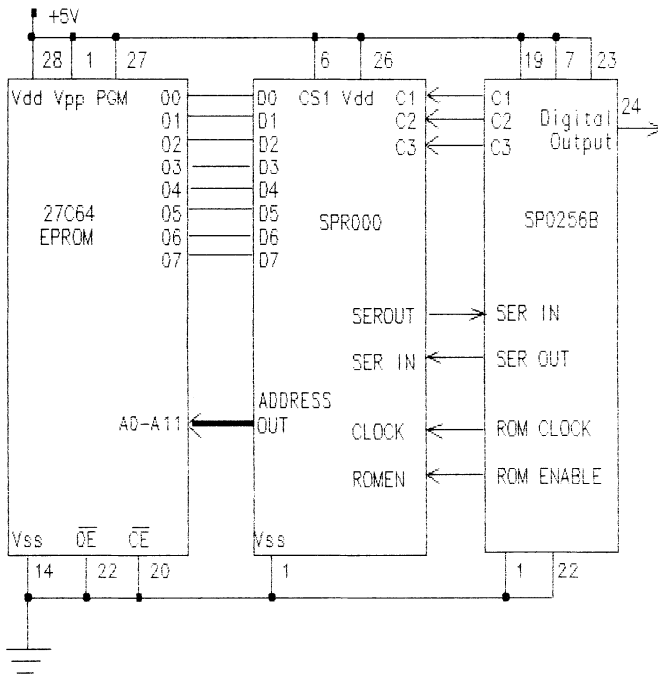
For applications requiring you to bank blocks of memory under external control, such as a  $\mu\text{C}/\mu\text{P}$ , the function pins CS1 and /CS2 must be used. CS1 is an active high-chip select, and will tri-state the serial output when low. /CS2 is an active low-chip select that will tri-state the output when high. To simplify chip selecting, address outputs /A11 to /A15 are used to select external memories. A0 to A15 are the address outputs to external memory. C1, C2, and C3 are the decoded control pins to determine device function, which is controlled by the SPO256B.

The ROM CLOCK input receives a 1.56 MHz frequency from the SPO256B. The serial input loads the 16-bit address into the device while the serial out shifts out the data byte. Eight-bit data outputs from external standard memories are received in SPR000's D0–D7 data inputs.

Figure 1.10 shows the popular 27C64 EPROM interfaced with the SPO256B via the SPR000.



**Figure 1.9** Block diagram of SPR000. Reprinted with permission from Publication #DS5007A-1, p. 1. © 1984 Microchip Technology, Inc.



**Figure 1.10** SPR000 interfacing a parallel 27C64 EPROM with the SPO256B speech processor.

The SPR000 operates from 4.5 to 7 V, consuming a current of 40 mA. An access time of 560 ns is typical to obtain each data byte from the EPROM.

## 1.7 SDS624 Speech Synthesis Development Board

The SDS264 is a development system consisting of a PC board and several diskettes. The system requires an IBM PC, XT, or AT with a hard disk, a math coprocessor, and extended memory. This system is a code generator for the SPO264/SPO256B speech processors and the SP1000LPC synthesis and recognition processor.

The system has the following features:

- Digitizes analog input speech using a 12-bit onboard A/D converter.
- Analyzes and computes speech synthesis parameters using analysis software included with the system.
- Edits speech synthesis parameters by means of a software editing capability.
- Reviews synthesized data output from the SPO216, SPO264, or SP1000 speech synthesis chips located on the development board.

- Compresses and formats the synthesized data into the desired bit rate (1200 to 5000 bps) for downloading to a PROM programmer. This feature allows you to create special words or phrases for your specific memories (EPROMs or EEPROMs).

The system also features a natural-sounding voice, variable sampling rates, and a 500 Hz output bandwidth.

The SDS264 is available from Telinovation, Inc., 447 Salmar Ave., Campbell, CA 95008.

## 1.8 Digitalker Kit DT1050

The Digitalker system consists of three n-MOS integrated circuits. The main IC is referred to as the speech processor chip, or SPC. The Digitalker ROMs store only those speech elements that the ear needs to hear. (The human vocal tract generates sounds that do not convey any intelligible information.) The techniques of digitization and compression are used by National Semiconductor in this system. This is a time-domain synthesis technique that reduces the amount of information needed to store electronic speech by removing the excess or redundant data from the speech signal.

The four main schemes that perform the task are:

1. Removing all redundant pitch periods and portions of certain other pitch periods.
2. Adaptive delta modulation coding, involving storing the arithmetic differences of successive wave amplitudes. This minimizes memory requirements.
3. Phase-angle adjustments, which remove the direction component of the speech waveform.
4. Half-period zeroing, replacing the low-level amplitude portion of a pitch period with silence. This technique reduces by 50% the amount of ROM required to store the speech data.

The result of using multiple compression techniques is a system capable of storing and reconstructing a word or phrase with high quality. The Digitalker is programmed with control information that instructs it how many times to repeat a specific waveform. Recordings of actual speech are sampled for digitization at a rate at least twice that of the highest frequency in the waveform pattern. Inside the SPC (see Figure 1.11) there are a programmable frequency generator and a variable gain D/A converter to add inflection that makes a realistic-sounding speech. The ROM set is programmed with a vocabulary consisting of 136 words, one complete phrase, two tones, and five different silence durations. Each word or phrase is assigned an 8-bit address. Address 129 (81H) is the “ss” sound; it is used after a word to make it plural. The system is more like a digital recorder that digitizes actual voices, stores, and then plays back, while the other methods model the vocal tract. The system is





**TABLE 1.2**  
Master Word List for the Digitalker PROMs

Decimal Address	Word	Decimal Address	Word
0	THIS IS DIGITALKER	58	AGAIN
1	ONE	59	AMPERE
2	TWO	60	AND
3	THREE	61	AT
4	FOUR	62	CANCEL
....	....	63	CASE
....	....	64	CENT
18	EIGHTEEN	65	400 Hz TONE
19	NINETEEN	66	80 Hz TONE
20	TWENTY	67	20 ms SILENCE
21	THIRTY	68	40 ms ''
22	FORTY	69	80 ms ''
23	FIFTY	70	160 ms ''
24	SIXTY	71	320 ms ''
25	SEVENTY	72	CENTI
26	EIGHTY	73	CHECK
27	NINETY	74	COMMA
28	HUNDRED	75	CONTROL
29	THOUSAND	76	DANGER
30	MILLION	77	DEGREE
31	ZERO	78	DOLLAR
32	A	79	DOWN
33	B	80	EQUAL
34	C	81	ERROR
35	D	82	FEET
36	E	83	FLOW
37	F	84	FUEL
38	G	85	GALLON
39	H	86	GO
40	I	87	GRAM
41	J	88	GREAT
42	K	89	GREATER
43	L	90	HAVE
44	M	91	HIGH
45	N	92	HIGHER
46	O	93	HOUR
47	P	94	IN
48	Q	95	INCHES
49	R	96	IS
50	S	97	IT
51	T	98	KILO
52	U	99	LEFT
53	V	100	LESS
54	W	101	LESSER
55	X	102	LIMIT
56	Y	103	LOW
57	Z	104	LOWER

Continued

Continued

105	MARK	124	PULSES
106	METER	125	RATE
107	MILE	126	RE
108	MILLI	127	READY
109	MINUS	128	RIGHT
110	MINUTE	129	SS (Prefix, See Note)
111	NEAR	130	SECOND
112	NUMBER	131	SET
113	OF	132	SPACE
114	OFF	133	SPEED
115	ON	134	STAR
116	OUT	135	START
117	OVER	136	STOP
118	PARENTHESIS	137	THAN
119	PERCENT	138	THE
120	PLEASE	139	TIME
121	PLUS	140	TRY
122	POINT	141	UP
123	POUND	142	VOLT
<i>Continued</i>		143	WEIGHT

Note: "SS" makes any singular word plural.

(Reprinted with permission from Linear Data Book, 1982, 13-17. © 1980 National Semiconductor Corporation)

**Chip Select (/CS):** The speech processor chip (SPC) is selected when /CS is low. /CS must be low during a command to the SPC, for example, when a /WR pulse is issued.

**Data Bus (SW1-8):** 8-bit address which defines any one of 256 speech entry points (see the master word list in Table 1.2). When not all the words listed are used, unused inputs must be connected to Vss.

**Command Select (CMS):** This input specifies the two possible commands to the SPC. When CMS is zero, it works as a reset interrupt and starts the speech sequence. When CMS is high, it works as a reset interrupt only.

**Write Strobe (/WR):** When pulsed low, the address specified in the data bus is latched into a register. On the rising edge of the /WR, it starts execution of the command as specified by CMS. If /WR is pulsed low to start a new speech sequence when the SPC is still executing the last one, the new speech sequence will be started immediately. This permits you to cut words or phrases at any desired point to concatenate a different message or word. (See Figure 1.11.)

**ROM Data (RData 1-8):** This is an 8-bit parallel bus for use with an external parallel speech ROM.

**Interrupt (INTR):** A logic 1 output indicates that the SPC is inactive. When the SPC is executing a speech sequence, INTR goes low. Therefore, /WR can be pulsed low only when INTR is high.

**ROM Address (ADR 0-ADR 13):** 14-bit parallel output bus that issues the address of the speech data to the speech ROM.



increments of one. A logic oscillator increments the counter and causes the write input (/WR) to be pulsed low via the half-monostable formed by N3 (1/3 4093). This low transient pulse causes the MM54104 to start speaking, at first using a female voice: "This is Digitalker." All subsequent messages are spoken using a male voice.

## 1.9 Toshiba CMOS Speech Synthesis LSI Devices

The speech synthesis devices from Toshiba appeared in February 1987. These devices are classified in two branches: ADM (adaptive delta modulation) and PARCOR (partial autocorrelation).

The ADM devices features are for low-cost speech systems, direct record, and to reproduce speech and sounds. Most of these devices are suitable for both low- and high-volume user's applications.

Tables 1.3 and 1.4 show a list of ADM devices. A typical application is shown in the schematic presented in Figure 1.13.

### 1.10 TSP5220C Voice Synthesis Chip from Texas Instruments Inc.

At the present time, Texas Instruments Inc. is offering four voice synthesis processors: the TSP50C40A, TSP50C50, TSP5110A, and TSP5220C. In this section, we will describe these devices briefly.

**TABLE 1.3**  
ADM Devices

Part Number	Function	Bit Rate/ Speech Time	Supply Volt
T6668	Recording/Reproduction DRAM Type	8K-32Kbps	4.5-5.7V
T6831	Recording/Reproduction SRAM Type	5.5K-16Kbps	4.5-5.7V
TC8830F	Recording/Reproduction SRAM Type	8K-32Kbps	4.5-5.5V
T6667	Reproduction Only, Built-in ROM	5.5K-16Kbps	3.5-5.7V
T6658A	Speaker-Dependent Word Recognition	10-40 words	4.5-5.5V

(Reprinted with permission from Microcomputer Product Summary, February 1987, 11. © Toshiba America, Inc.)

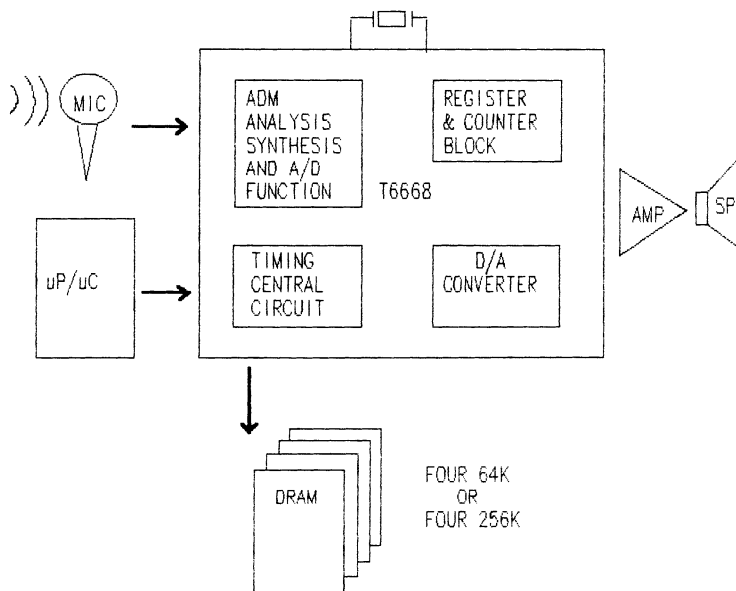
**TABLE 1.4**  
Two Speech Synthesizers and Two Speech ROMs Belong to the PARCOR  
(partial autocorrelation devices)

Part Number	Function	Bit Rate/ Speech Time	Supply Volt
T6803	Speech Synthesis Built-in 64K MROM	2.5-9.8Kbps	3.9-5.7V
T6721	Speech Synthesis	2.4-9.6Kbps	3.5-5.7V
T6772	64K Masked ROM	9-35 Sec	3.5-5.7V
T6884	128K Masked ROM	17-70 Sec	3.5-5.7V

(Reprinted with permission from Microcomputer Product Summary, February 1987, 11. © 1987 Toshiba America, Inc.)

Speech encoding on all TI voice processors is achieved with LPC coding. The inputs of these processors contain the codes for 12 synthesis parameters (pitch, energy, and 10 filter coefficients). These codes are decoded by the voice processor to give out time-varying signals of the LPC model of the original voice.

The digital filter of this voice processor receives periodic and random signals. Periodic inputs are used to reproduce vowels or voiced fricatives (z,



**Figure 1.13** A typical T6668 device application interfaced with an external CPU.  
(Reprinted with permission from Microcomputer Product Summary, February 1987, 11. © 1987 Toshiba America, Inc.)

b, d). On the other hand, random inputs derive unvoiced sounds, such as s, f, t, and sh. Two separated sources generate the voiced and unvoiced excitations. The output of the digital-to-analog converter is filtered before driving a loudspeaker.

Texas Instruments produces the TSP5220C, a speech synthesis device based on an LPC-10 (linear predictive coding with a 10th-order filter). Its architecture allows different storage media for the model of the vocal tract. An external microcontroller can also be interfaced to the TSP5220C to select the digital data.

To operate the TSP5220C in a voice synthesizer system, the following devices are required:

1. Storage device (ROM, RAM or TSP6100) for TSP5220C input data.
2.  $\mu\text{P}/\mu\text{C}$  to direct the TSP5220C modes of operation. Simple digital logic can also be used to work as a host controller.
3. A low-pass filter to remove high-frequency switching noise from the output signal of the TSP5220C.
4. An audio amplifier and a speaker.

The TSP5220C features a low data rate (1000 to 17000 bps), a 4 or 5 kHz voice input bandwidth, an 8-bit digital-to-analog converter, and a pitch-excited LPC-10 synthesis algorithm. Voice data input can be selected through an 8-bit data bus or a serial interface for use with a TSP6100 masked ROM.

### *Theory of Operation*

An external host controller ( $\mu\text{P}/\mu\text{C}$ ) can be used to issue commands and filter parameters to produce synthetic speech. Figure 1.14 shows a block diagram of the TSP5220C voice synthesizer.

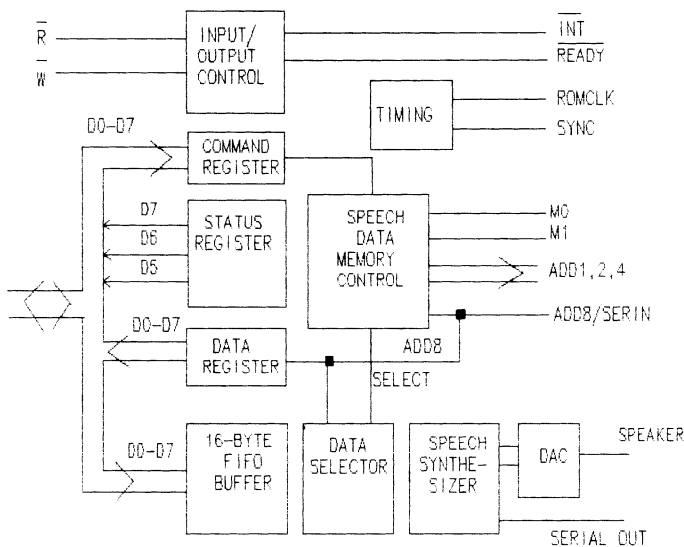
The input and output structure is described as follows:

**D0–D7:** Memory data bus interface for use with the external host controller. This is achieved by controlling the inputs /READ (/R), /WRITE (/W), and monitoring the outputs interrupt (/INT) and ready (/READY).

**ADD1, ADD2, ADD4, and ADD8/SER IN:** These are the address outputs to the external voice synthesis memory. The pin function ADD 8/SER IN can also be used as a serial data input. Note that ADD1 is the least significant bit while ADD8 is the MSB when both ADD1 to ADD8 are used as addressing inputs to a vocabulary ROM (TSP6100) series. Clock and control signals M0, M1 are provided for memory control.

### *Memory Data Bus*

The 8-bit data bus used for interface with an external controller can receive data into the command or FIFO registers by pulsing the write input (/W) low. By pulsing the read command low, data are read from the data or status registers to the external host controller.



**Figure 1.14** TSP5220C block diagram. (Reprinted with permission from Texas Instruments Inc. © 1986)

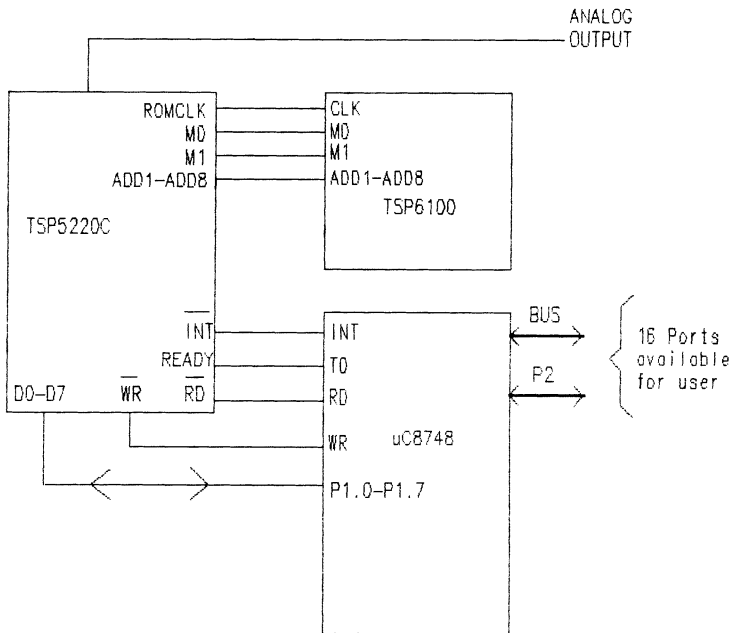
The four registers used to interface to the external memory data are the command register, FIFO register, data register, and status registers (flags).

The command register is formed by an 8-bit latch which is controlled from an external controller. The FIFO register is a 16-byte register that receives, via the memory data bus, speech data from the bus controller. Also, speech data are serially given to the speech synthesizer. The data register is an 8-bit SI/PO latch that receives serial speech data from a TSP6100 ROM. These data are routed to a parallel byte-wide bus for access with the memory data bus for output to the external controller. The 3-bit status register contains data on the status of the TSP5220C. The status word is put onto the memory data bus so that it can be accessed by an external host controller. The status register can be read by taking the Read (/R) input low. When this happens, the TSP5220C sends status data to the memory data bus. When the data are stable, the /READY signal goes low. A 12  $\mu$ s time delay is required before applying another write or read command. The 8-bit memory data bus has internal pull up resistors.

Figure 1.15 illustrates the TSP5220C system using the TSP6100 ROM for speech data interfaced with an Intel microcontroller 8748.

With this ROM (TSP6100), the TSP 5220C can “talk” about 200 words in more than 100 seconds. The TMS5220C can access a maximum of sixteen 128K memories. A complete voice synthesis system can be assembled from three ICs, a speaker, and a microcontroller (see Figure 1.15).





**Figure 1.15** TSP5220C speech synthesizer system controlled by a  $\mu$ C.

### 1.11 CMOS ADPCM Speech Synthesizers and Recorders from Oki Semiconductor

Oki Semiconductor produces different types of speech synthesizers and recorders of the series 52XX and 62XX. In order to develop speech synthesis systems for specific applications, Oki Semiconductor offers on a loan basis the speech analyzer OSA-1 and other support tools. The OSA-1 system allows the user to make straight/compressed adaptive differential PCM (ADPCM) data for the speech synthesizers MSM6243 and MSM6212 by applying an input voice via a microphone or a tape deck player. The user can also change the degree of compression for voice analysis purposes. The ADPCM-analyzed output data can be downloaded into an EPROM via an RS232 interface.

Speech analyzers can be built around the MSM5218RS to generate the ADPCM data on a real-time basis by applying the necessary input voice. These speech data can be stored as ROM data for the simulator MSM5248.

The simulator MSM5248 contains the same functions as the chip MSM5248 and is able to synthesize the same quality voice. There are two more simulators available: the MSM6243 and the MSM6212.

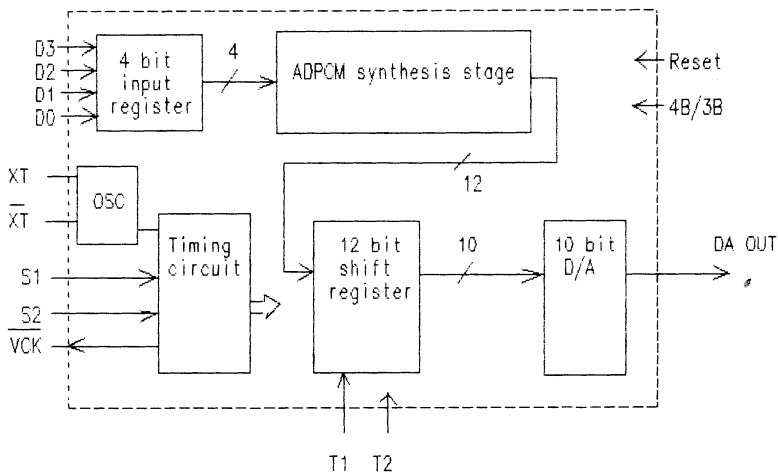
The speech synthesizer MSM5205, for example, is an integrated circuit

which accepts ADPCM data. It contains a synthesis stage that expands the 3- or 4-bit ADPCM data to 12-bit PCM data and a stage that converts the PCM data to analog signals via a 10-bit D/A converter. The sampling frequency can be selected by inputs S1 and S2 in steps of 2 kHz (4, 6, and 8 kHz) when a 384 kHz crystal is used in conjunction with two timing capacitors of 220 pF.

The MSM5205 device operates from a 5 V power supply with an operating temperature range of  $-30^{\circ}\text{C}$  to  $+70^{\circ}\text{C}$ . Figure 1.16 shows the functional block diagram.

The MSM5218 is a speech analysis/synthesis IC that enables the user to develop his own speech analysis and synthesis systems. The data compression is also made by ADPCM. This device also synthesizes PCM data from ADPCM data. The PCM data are accessible directly or in analog form via the internal 10-bit D/A converter. The MSM5218 also features variable sampling frequency (4, 6, and 8 kHz), handshaking signals for synchronous operation with an external A/D converter, and its typical power consumption is 15 mW. It is available in 24-pin plastic DIP and 32-pin plastic flat.

The MSM5248 is an ADPCM voice synthesizer with 48 Kbits of internal ROM for the user's program. Its sampling frequency is 5.46 kHz when controlled by a 32.768 kHz crystal. The chip supports a maximum length of speech of 3 seconds with a limit of seven words selectable by the user. For applications requiring more vocabulary, the speech synthesizer MSM6243, which contains 192 Kbits of ROM, can support a maximum of 124 words. The maximum speaking time of compressed ADPCM data for the MSM6243 is 20 seconds, while the MSM6212 has a maximum speaking time of 40 seconds. The MSM6212 contains 288 Kbits of internal ROM.



**Figure 1.16** Functional block diagram of the MSM5205 speech synthesis IC. (Oki Voice Synthesis LSI Data Book, July 1989, 29. © 1989, Oki Electric Industry Co., Ltd. Reprinted with permission.)



Oki Semiconductor also produces two types of solid state recorders, the MSM6258 and the MSM6258V. Both devices contain an ADPCM speech processor implemented in CMOS technology for low power consumption.

A/D and D/A converters are contained in both chips and can be looped to connect external devices. They also feature a voice detector circuit and a phrase selector accepting analog or PCM data input and processing analog or PCM data output, a static RAM interface accepting a maximum of 128 Kbytes, and a static RAM interface for a maximum of two megabytes. Three sampling frequencies are possible; 4.0, 5.3, and 8.0 kHz (@4.096 MHz) clock. It contains recording and playback outputs and seven phrase channels with individual lengths.

It is available in two versions: for stand-alone operation and for MPU interface (8-bit), as illustrated in Figures 1.17 and 1.18, respectively.

Both devices operate with a single power supply of +5 V (10%) with a current consumption of 4 mA (@4.096 MHz) or 10  $\mu$ A during standby condition for SRAM interface. The stand-alone version comes in a 60-pin plastic Flat or 68-pin PLCC. The MPU interface version comes in a 44-pin plastic flat or 40-pin plastic DIP.

## 1.12 Samsung Voice Synthesizers ICs

The KS59XXX series of speech synthesizers are developed by Samsung Semiconductor and Telecommunications Co., Ltd.(SST). These devices are classified as KS5901A, KS5902XX, KS5911, and KS5912XX, using the encoded reproduction algorithm LPC. Speech is compressed by processing an externally provided variable bit stream of encoded LPC speech data. The result is then converted to an audible output with an on-chip 9-bit D/A converter.

Table 1.5 shows the principal characteristics of the four kinds of speech synthesizers available from SST.

<b>TABLE 1.5</b> Characteristics of the Four Kinds of Speech Synthesizers from SST (Reprinted with permission of Samsung Semiconductor and Telecommunication)				
Features	KS5901A	KS5902XX	KS5911	KS5912XX
Synthesis Method	LPC	LPC	ADM	ADM
Operating Voltage	5V	5V	5V	5V
Oscillation Frequency	640KHz (X-Tal OSC)	2.56 MHz (X-Tal OSC)	640 KHz (RC OSC)	640 KHz (RC OSC)

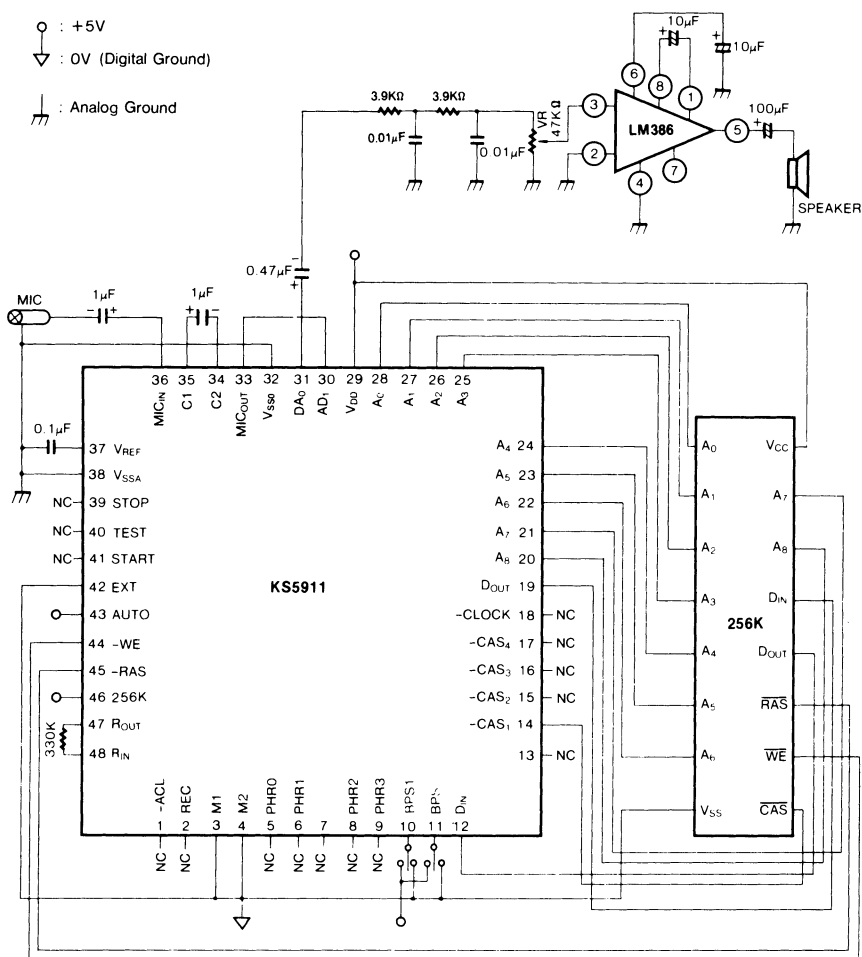
Sampling Frequency	8 KHz	8 KHz	8, 11, 16, 32 KHz	8, 11, 16, 32 KHz
Bit Rate	2.4~9.6 Kbps	2.4~9.6 Kbps	8, 11, 16, 32 Kbps	8, 11, 16, 32 Kbps
Control Mode	CPU/Manual	CPU/Manual /Auto	Talk-Back/Manual	Manual
Data ROM (or RAM)	External ROM Max. 64KBytes	Internal ROM 48KBits	External RAM 64K/256DRAM×4	Internal ROM 64KBits
Speech Times	Max. 4 Min.	Max. 20 Sec.	Max. 2 Min.	Max. 8 Sec.
D/A Converter Bits	9 Bits	9 Bits	10 Bits	10 Bits
PKG	60 FQP	24 DIP	48 FQP	16 DIP
Applications	Sound Information A/M	Toy Simple Sound Information	Talk-Back A/M	Toy Natural Sound Effect

The talk-back speech synthesis system shown in Figure 1.19, for example, is composed of the following three chips.

1. KS5911A: CMOS speech synthesizer
2. PROM: External 8-bit DRAM memory.
3. LM386: Low-power audio amplifier.

## References

- Slater, Neil.: Introduction to Electronic Speech Synthesis. Blacksburg, Howard & Sams & CO. Inc.
- Oki Voice Synthesis LSI Data Book. (First Edition, Aug. 1987)
- TSP5220 Speech Synthesis Manual, 1987. Texas Instruments
- Stout, David F.: Microprocessor Applications Handbook. McGraw-Hill Book Company
- Kevin Leary and David Morgan: Fast and accurate analysis with LPC gives a DSP chip speech-processing power. Electronic Design.
- SPO256B Narrator Speech Processor, (DS50018A-1)
- SPO264 Narrator Speech Processor, (DS50012C-1)
- SPR128A/128B 128K Bit Serial Speech ROM, (DS500006C-1)
- SPO256-AL2 Narrator Speech Processor, (DS50005A-1)
- 1988 Microchip Technology Inc.



**Figure 1.19** KS5911 Block diagram interfaced with an external controller. (Samsung Semiconductor Product Guide, 1990, p. 207. Reprinted with permission.)

## *Experimenting with Speech Processors*

### **2.1** Evaluating the Allophone Addresses for the SPO256-AL2

Before applying the speech processor SPO256-AL2 in a variety of circuits, the user should know how every allophone sounds to have an idea of how to concatenate the right allophone using the basic linguistic rules described in Chapter 1. The allophone set is shown in Table 1.1 of Section 1.4. Please have it at hand when you start this circuit, so you can recognize each allophone.

The speech processor SPO256-AL2 is used here to read the allophones that are contained on its own 64K ROM. The first requirement is to apply a power-up reset pulse to the 8-bit binary counter CD4520 and to the dual BCD counter CD4518, IC1 and IC4 respectively (see Figure 2.1). This procedure will ensure that both counters will start with a count of zero.

Binary counter IC1 provides the addresses sequentially to the speech processor. IC4 works like a mirror of IC1 by counting the same clock pulses, but in BCD code. The positive reset pulse required by IC1 and IC4 is supplied by the R1 C1 network, and the negative reset pulse for the speech processor is supplied by the R2 C2 network.

When the START switch is pressed, the speech processor vocalizes the first allophone “AY” and continues with the next allophones in sequential order.

The standby (SBY) output is normally in the high state and will go low when the /ALD input is pulsed low. That is, the SBY output stays low until the chip stops vocalizing a particular allophone. A high standby output enables the Nand gate IC2a for a new input command. There are no timing problems using the Nand gate since only when the speech processor is ready to accept a new /ALD pulse will the Nand gate be enabled for transmitting a new pulse.





the dual BCD counter 4518, which is used to observe the decimal address of the allophone that is currently being vocalized. The BCD output of CD4518 is fed to a pair of CD4543s (BCD to seven segment decoder-drivers). The pair of CD4543s drive the two-digit liquid crystal display LCD002. Nand gate IC2b and associated components form a 100-Hz logic oscillator whose output frequency controls the LCD's back plane input as well as the phase input (pin 6 of CD4543) of the two decoders.

The period of the reset pulses can be obtained by assuming that the reset inputs have threshold levels of one-half of  $V_{dd}$ ; that is, 2.5 V.

For the network R1 C1:  $T = R1 C1 \ln [5/2.5] = 0.7 R1 C1$

$$T = 0.7 (100K) (0.1\mu F) = 7 \text{ ms}$$

And for the R2 C2 network we get:

$$T = R2 C2 \ln [5/(5 - 2.5)] = 0.7 R3 C3$$

$$T = 7 \text{ ms}$$

The timing diagram in Figure 2.2 shows the most important waveforms of the circuit presented in Figure 2.1. Note that the reset pulse applied to IC1 and IC4 is given by the voltage in resistor R1 which is

$$V(R1) = (V_{dd}/R)e^{-t/RC}$$

The reset pulse applied to the speech processor is given by the voltage in capacitor C2 which is:

$$V(C2) = V_{dd}(1 - e^{-t/RC})$$

The propagation times  $t_{phl}$  and  $t_{plh}$  are both equal to 300 ns, and they correspond to the time that the Schmitt trigger Nand gate takes to respond to a given input.

The next step for this circuit is to create words and phrases that can be vocalized correctly with this processor; this is achieved by adding an EPROM memory between IC1 and IC3 with a previously tested program. This circuit is shown in Figure 2.3.

This circuit can be tested by recording the data shown in Table 2.1 in the EPROM. These data correspond to the words "zero, one, two, three."

A variation of the circuit presented in Figure 2.1 is to set the flip-flop device when the power is turned on to enable the speech processor to give spoken instructions for a specific task routine. When the instructions are over, the O6 output of EPROM 2764 resets the flip-flop (4013) and the binary counter (4040). This is achieved by programming 40H after the last speech data, where 40H corresponds to a binary output (01000000). The timing diagram in Figure 2.4 shows the pulse generated by EPROM 2764 at the end of a complete message.

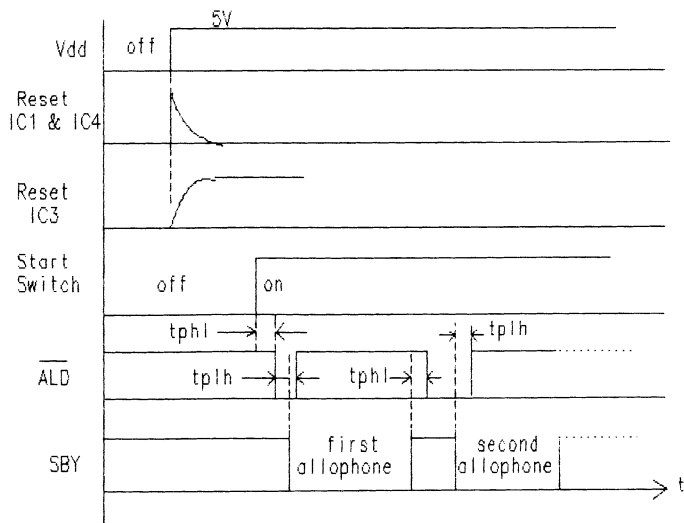


Figure 2.2 Timing diagram for circuit in Figure 2.1.

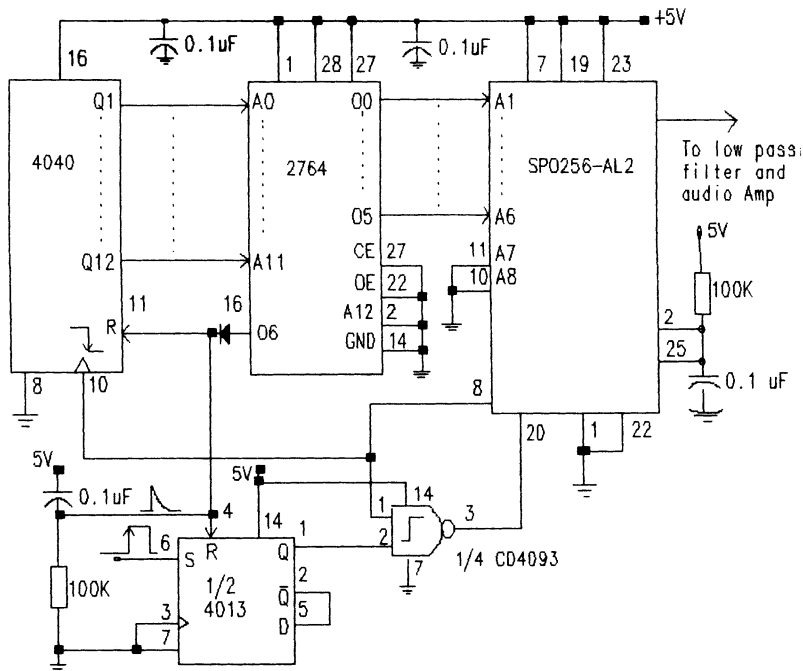
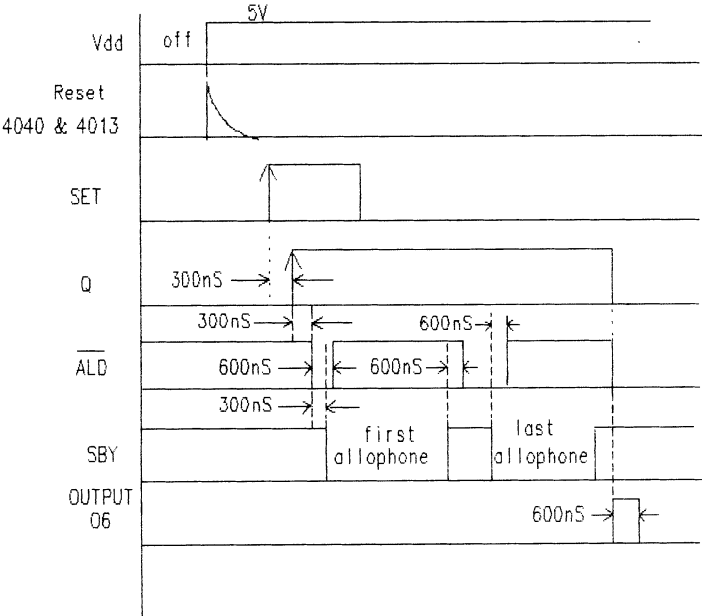


Figure 2.3 Circuit to create custom words or phrases with the speech processor SPO256-AL2.

**TABLE 2.1**  
EPROM Program for the Words "Zero, One, Two, Three."

Hex Address	Hex Data		Hex Address	Hex Data	
0	4		9	4	
1	2B		A	D	
2	3C	zero	B	1F	two
3	35		C	4	
4	4		D	1D	
5	39		E	E	three
6	F	one	F	13	
7	F		10	04	pause
8	B		11	40	reset pulse

*Continued*



**Figure 2.4** Timing diagram for circuit of Figure 2.3.

## 2.2 Interfacing the SPO256-AL2 with a PC/XT

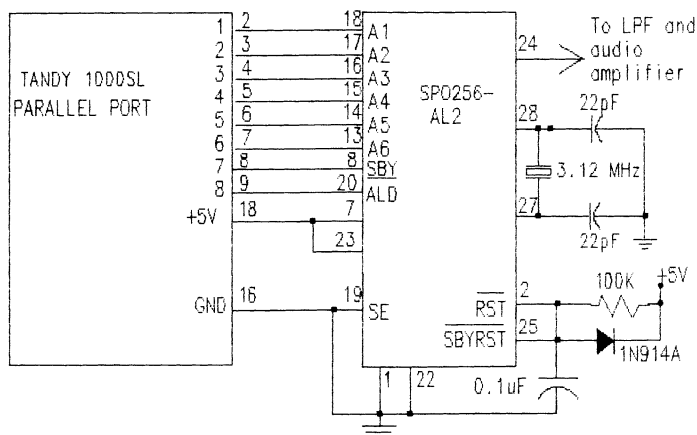
By using the GWBASIC version 3.21, the parallel port of a PC/XT IBM-compatible computer can be accessed to control the most important functions of the speech processor SPO256-AL2. This way, you will be able to practice with the allophone set to create your own custom vocabulary for a specific application.

Figure 2.5 shows the schematic diagram for interfacing the speech processor with the parallel port of the TANDY 1000SL microcomputer. The connector is a 36-pin card edge type. Notice that the speech processor has the strobe enable input (SE) tied to ground, which disables the /ALD input (pin 20) of the SPO256-AL2. The speech processor is used in MODE 0, requiring you to apply zeroes at the address inputs (A1–A6) of the speech processor after a specific address has been issued.

The computer controls the SPO256-AL2 with simple GWBASIC instructions. In this project, a TANDY 1000SL microcomputer is used to control the speech processor via the parallel printer port. The following program allows you to evaluate the 59 allophones available (see Table 2.2).

Another application for your computer is to make it speak a specific message when you instruct it to do so. This will be used in programs written in GWBASIC. By calling the message routine you will hear your prerecorded message or group of messages. Table 2.3 shows a program to make your computer speak the message “try again.”

The previous routines have as many applications as the scope of your



**Figure 2.5** Schematic diagram for the interface of a PC-compatible computer with the speech processor SPO256-AL2.

**TABLE 2.2**

Routine to Evaluate the Set of Allophones and Pauses  
of the Speech Processor SPO256-AL2.

---

```

10 'Evaluating the set of allophones.
20 FOR A = 0 TO 63
30 OUT 32, A           'output port is loaded with the value of A
40 OUT 32, 0           'output port is loaded with zeroes to start the
42                     'speech utterance, SBY = 0
50 PB = INP(127)       'reads the 7-bits
60 F = PB AND 64       'reads bit 7 only (SBY)
70 IF F<>64 THEN 50    'If SBY=0 continue reading SBY, else goto 80
80 NEXT A              'increment value of A
90 OUT 32,1
95 END

```

**TABLE 2.3**

Use this BASIC Routine and You Will Hear the Message "Try Again"

---

```

10 'try again
20 FOR J = 1 TO 9
22 READ A
30 OUT 32, A           'output port is loaded with the value of A
40 OUT 32, 0           'output port is loaded with zeroes to start the
42                     'speech utterance, SBY = 0
50 PB = INP(127)       'reads the 7-bits
60 F = PB AND 64       'reads bit 7 only (SBY)
70 IF F<>64 THEN 50    'If SBY=0 continue reading SBY, else goto 80
80 NEXT J              'increment value of A
90 DATA 13,39,6,4,24,36,7,11,4
95 END

```

imagination can reach. You may try, for example, to create a routine for a talking clock program.

## 2.3 Interfacing a Speech Synthesizer<sup>1</sup> to a Commodore 64 Computer

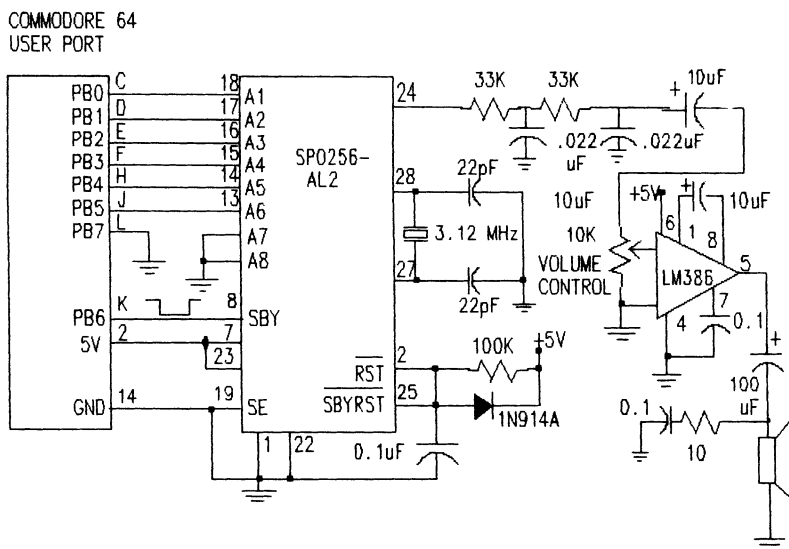
The versatility of a Commodore 64 computer can be enhanced by adding a speech processor. In this section we will interface the allophone-based speech processor SPO256-AL2 to the parallel user port of a Commodore 64 computer

<sup>1</sup>Source: Reprinted and adapted with permission from *Computer Digest*, August 1986. © Copyright Gernsback Publications, Inc., 1986.

in order to create several software programs to control the speech processor efficiently.

Figure 2.6 shows the schematic diagram for interfacing the speech processor SPO256-AL2 to the Commodore 64 and to the audio amplifier LM386. The user port must be configured with lines PBO to PB5 as outputs. Lines PB6 and PB7 will be configured as inputs. This is achieved with the statement "POKE 56579,63," where PBO to PB5 send a specified address to the speech processor IC1 (see Figure 2.6,) and PB6 receives the logic status of the STANDBY output (pin 8 of IC1). In this case, the speech processor is configured in MODE 0; for every speech datum issued by the computer, output lines PBO to PB6 will go to a low state. In this manner, the speech processor will enunciate the assigned allophone. Then the computer will start reading the user port in order to know the STANDBY status (PB6), which tells the computer when the speech processor is ready to be triggered again. The instruction "PEEK(56577)" reads the user port, but we need to read PB6 only, which represents the decimal number 64. This is made by masking the user port with the AND function. If PB6 is a logic one, the computer sends a new address.

We will now proceed to evaluate the set of allophones of the SPO256-AL2 by using the routine shown in Table 2.4. The FOR-NEXT loop (line 30) is used to increment the value of A from 0 to 63. The purpose is to use the variable A in the POKE statement (line 40) in order to address each allophone. The instruction "POKE 56577,0" clears and prepares the user port for the



**Figure 2.6** Schematic diagram of the circuit that interfaces the SPO256-AL2 to a Commodore 64 computer.

**TABLE 2.4**  
Software Program to Evaluate  
the 59 Available Allophones

```

10 REM ALLOPHONES EVALUATION
12 REM BY R. JIMENEZ AND ADRIAN VALLE
20 POKE 56579,63
30 FOR A=0 TO 64:PRINT A
40 POKE 56577,A
50 POKE 56577,0
60 PB = PEEK(56577)
70 F=PBAND64
80 IF F<>64 THEN 60
90 NEXT A
100 POKE 56577,1
120 END

```

next data. The PEEK statement (line 60) reads the user port, storing its value in the variable PB. Then PB is compared with the number 64 on the IF statement (line 80). If F is not equal to 64, the speech synthesizer is not ready to receive new data, and the computer goes back to line 60 automatically; otherwise, new data are sent. The statement "POKE 56577,1" makes the speech

**TABLE 2.5**  
Software Program to Make the Computer  
Say the Words "I Am a Talking Computer"

```

10 REM I'M A TALKING COMPUTER
60 POKE 56579,63
65 FOR J=1 TO 27
70 READ A
80 POKE 56577,A
85 POKE 56577,0
90 PB=PEEK(56577)
94 F=PB AND 64
96 IF F<>64 THEN 90
100 NEXT J
102 DATA 24,6,1:REM      I
104 DATA 7,7,16,2:REM   AM
106 DATA 24,2:REM       A
108 DATA 13,23,23,2,42,12,44,1:REM TALKING
110 DATA 42,15,16,9,49,22,13,51,1,4,:REM COMPUTER
160 RESTORE
170 FOR T=1 TO 500:NEXTT:GOTO 65
200 END

```

processor stop saying the last allophone. If you wish to listen to the allophones slowly, just press the (CTRL) control key.

Table 2.5 shows a program which makes the speech synthesizer say the sentence "I am a talking computer." This program works like the first one (Table 2.4), using the same instruction to write and read by means of the user port. The only difference is that the data (lines 103–110) are going to be sent

**TABLE 2.6**

This Program Will Cause Your Computer  
to Clearly Speak Each Number When  
the Appropriate Number Key Is Pressed

---

```

12 NUMBERS
15 REM BY R. JIMENEZ AND A. VALLE
50 PRINT " (SC) "
60 POKE 56579, 63
65 FOR I=0 TO 9
70 READ A(I)
75 FOR J=1 TO A(I)
78 READ B(I, J)
100 NEXT J, I
110 DATA 4, 43, 60, 53, 4: REM          ZERO
112 DATA 5, 57, 15, 15, 11, 4: REM      ONE
114 DATA 3, 13, 31, 4: REM              TWO
116 DATA 4, 29, 14, 19, 4: REM          THREE
118 DATA 4, 40, 40, 58, 4: REM          FOUR
120 DATA 5, 40, 40, 6, 35, 4: REM        FIVE
122 DATA 8, 55, 55, 12, 12, 2, 41, 55, 4: REM SIX
124 DATA 8, 55, 55, 7, 7, 35, 12, 11, 4: REM SEVEN
126 DATA 4, 20, 2, 13, 4: REM           EIGHT
128 DATA 5, 56, 24, 6, 11, 4, : REM     NINE
150 V = V+1
160 IF V=10 THEN V=0 : PRINT " (SC) "
170 PRINT "  VALUE OF X("; V; ") ";
200 GET C$: IF C$=" " THEN 200
205 C = ASC(C$)
210 IF C<48 OR C>57 THEN 200
215 C = C-48 : PRINT C
217 POKE 56577, 0
220 FOR I=1 TO A(C)
230 POKE 56577, B(C, I)
240 POKE 56577, 0
250 PB=PEEK(56577)
260 F=PB AND 64
270 IF F<>64 THEN 250
280 NEXT I: POKE 56577, 1
290 GOTO 150
300 END

```



with the READ statement (line 70). Note that the data are written in decimal numbers. The words used in this sentence were taken from the dictionary included with the SPO256-AL2 package.

Table 2.6 shows a routine that can be used for data processing. This routine makes the speech synthesizer say numbers from 0 to 9 when you press the respective key number. Lines 65 to 100 are used exclusively to assign the data found in lines 100 to 128 to the dimensioned variables A(I) and B(I,J). Here you can see how A(I) contains the data that the speech synthesizer will use to pronounce a particular word. In this case, 10 numbers can be pronounced. For example, A(I) holds the number 4 when the variable “I” has a value of 0. And the statement READ (line 70) is executed by the computer. This means that the word “zero” will be spoken by the speech synthesizer by just sending the four data contained in the vector B(I,J) for values of “I” equal to zero with “J” varying 1 to 4.

The FOR-NEXT loops are used as auxiliaries to the statement READ. Observe how A(I) holds the first data to be used as a variable of the statement FOR (line 75) to read the exact quantity of data for each spoken number. Data from lines 110 to 128 are the decimal number values that we need to make the speech synthesizer talk. Vector A(I) will store the first data contained in such lines (4, 5, 3, 4, 4, 5, 8, 8, 4, 5) which, as we said before, indicates the data contained in the vector B(I,J), respectively. For example, the data of line 110 form the word “ZZ YR OW PA5” where the respective data are 43, 60, 53, and 4.

Lines 150 to 215 give an example of how to make the speech synthesizer more versatile. The screen will display “VALUE of X(V)?” where “V” varies from 0 to 9. The A(C) works as a variable in line 220 of the statement FOR, which means that “I” varies from 1 to the A(C) value where “C” indicates the number that will be spoken. Line 217 POKes a zero so that the new data can be accepted without problems. Line 240 serves the same purpose, and lines 250 to 270 are used to read, as was explained before, the STANDBY condition.

## 2.4 Generating a Technical Vocabulary for the SPO256-AL2

Because most of the projects in this book require technical words for measuring many kinds of variables, it is a good beginning to have these words ready in hex code. An EPROM or a microcontroller needs the hex code to give out the desired speech entry points to the speech processor. The column containing the allophones will be used in two projects presented in Section 10 of this chapter as well as in most of the circuits used in the following chapters.

Table 2.7 presents the most widely used words as well as the respective hex code and allophones.

**TABLE 2.7**  
Technical Vocabulary in Hex Code with the Respective Allophones

Word	Hex Code	Allophones
Zero	2B, 3C, 35	ZZ, YR, OW
One	39, F, F, B	WW, AX, AX, NN1
Two	D, 1F	TT2, UW2
Three	10, E, 13	TH, RR1, IY
Four	28, 28, 3A	FF, FF, OR
Five	28, 28, 6, 23	FF, FF, AY, VV
Six	37, 37, C, 2, 29, 37	SS, SS, IH, IH, PA3, KK2, SS
Seven	37, 37, 7, 7, 23, 7, B	SS, SS, EH, EH, VV, EH, NN1
Eight	14, 2, D	EY, PA3, TT2
Nine	38, 18, 6, B	NN2, AA, AY, NN1
Ten	D, 07, 07, B	TT2, EH, EH, NN1
Eleven	C, 2D, 7, 7, 23, C, B	IH, LL, EH, EH, VV, EH, NN1
Twelve	D, 30, 7, 7, 2D, 23	TT2, WH, EH, EH, LL, VV
Thirteen	1D, 33, 1, 2, D, 13, B	TH, ER1, PA2, PA3, TT2, IY, NN1
Fourteen	28, 3A, 1, 2, D, 13, B	FF, OR, PA2, PA3, TT2, IY, NN1
Fifteen	28, C, 28, 2, D, 13, B	FF, IH, FF, PA2, TT2, IY, NN1
Sixteen	37, 37, C, 2, 29, 37, 2, D, 13, B	SS, IH, PA2, KK2, SS, PA2, TT2, IY, NN1
Seventeen	37, 37, 7, 23, 1D, B, 2, D, 13, B	SS, EH, VV, TH, NN1, PA2, TT2, IY, NN1
Eighteen	14, 2, D, 13, B	EY, PA2, TT2, IY, NN1
Nineteen	B, 6, B, 2, D, 13, B	NN1, AY, NN1, PA3, TT2, IY, NN1
Twenty	D, 30, 7, 7, B, 2, D, 13	TT2, WH, EH, NN1, PA3, TT2, IY
Thirty	1D, 34, 2D, 13	TH, ER2, PA3, TT2, IY
Forty	28, 3A, 2, D, 13	FF, OR, PA3, TT2, IY
Fifty	28, 28, C, 28, 2, D, 13	FF, FF, IH, FF, PA3, TT2, IY
Sixty	37, 37, C, 2, 29, 37, 1, D, 13	SS, SS, IH, PA3, KK2, SS, PA2, TT2, IY
Seventy	37, 37, 7, 23, C, B, 2, D, 13	SS, SS, EH, NV, IH, NN1, PA3, TT2, IY
Eighty	14, 2, D, 13	EY, PA3, TT2, IY
Ninety	B, 6, 11, 2, D, 13	NN1, AY, NN1, PA3, TT2, IY
Hundred	39, F, F, B, 1, 21, 27, C, C, 1, 15	HH2, AX, AX, NN1, PA2, DD2, RR2, IH, IH, PA1, DD1
Thousand	1D, 18, 2D, 2B, 1, B, 15	TH, AA, ZZ, TH, PA2, NN1, DD1
Million	10, C, C, 2D, 31, F, B	MM, IH, IH, LL, Y1, AX, NN1
And	18, B, 15	AA, NN1, DD1
Ampere	18, 10, 9, 34	AA, MM, PP, ER2
Cent	37, 37, 7, B, 11	SS, SS, EH, NN1, TT1
Centi	37, 37, 7, B, C	SS, SS, EH, NN1, IH
Check	32, 7, 7, 2, 29	CH, EH, EH, PA3, KK2
Danger	21, 7, B, 19, 33	DD2, EH, NN1, YY2, ER1
Degree	21, C, 24, 27, C	DD2, IH, GG1, RR2, IH
Dollar	21, F, 20, 33	DD2, AX, LL, ER1
Equal		IY, PA2, PA3, KK3, WH, AX, EL,

Error	7, 2F, 3A	EH, XR, OR
Feet	28, C, 11	FF, IH, TT1
Farads	28, F, 27, F, 37	FF, AX, RR2, AX, DD1, SS
Fuel	28, 13, 2D	FF, IY, LL
Gallon	24, F, 2D, 35, B	GG1, AX, LL, OW, NN1
Go	24, 35	GG1, OW
Gram	24, 27, 1A, 10	GG1, RR2, AE, MM
High	39, 6	HH2, AY
Higher	39, 6, 33	HH2, AY, ER1
Hour	20, 33	AW, ER1
Inches	13, B, 25, 7, 37	IY, NN1, SH, EH, SS
Is	C, 37, 37	IH, SS, SS
It	C, 11	IH, TT1
Kilo	2A, 13, 2D, 35	KK1, IY, LL, OW
Less	2D, 7, 37, 37	LL, EH, SS, SS
Lesser	2D, 7, 37, 33	LL, EH, DD, ER1
Limit	2D, C, 10, C, 11	LL, IH, MM, C, TT1
Low	2D, 35	LL, OW
Lower	2D, 35, 33	LL, OW, ER1
Milli	10, C, 2D, C	MM, IH, LL, IH
Micro	10, 6, 8, 27, 35	MM, AY, KK3, RR2, OW
Minus	10, 6, B, F, 37	MM, AY, NN1, AX, SS
Minute	10, C, B, C, 2D	MM, IH, NN1, IH, PA3, TT2
Number	38, F, 10, 3F, 33	NN2, AX, MM, BB2, ER1
Off	18, 28, 28	AA, FF, FF
On	18, B,	AA, NN1
Percent	9, 33, 37, 7, 38, 11	PP, ER1, SS, EH, NN2, TT1
Pico	9, C, 8, 35	PP, IH, KK3, OW
Please	9, 2D, 13, 37	PP, LL, IY, SS
Point	9, 5, B, 11	PP, OY, NN1, TT1
Pulses	9, 1E, 2D, 37, 7, 37	PP, UH, LL, SS, EH, SS
Rate	E, 14, 11	RR1, EY, TT1
Ready	E, 7, 7, 1, 21, 2D	RR1, EH, EH, PA1, DD2, IY
Right	E, 6, 11	RR, AY, TT
RPM	3B, 9, 13, 7, 10	AR, PP, IY, EH, MM
Set	37, 7, D	SS, EH, TT2
Speed	37, 2, 9, C, 21	SS, PA1, PP, IH, DD2
Stop	37, 37, 11, 18, 9	SS, SS, TT1, AA, PP
Than	36, 1A, 18	DH2, AE, NN1
The	36, 1A	DH2, AE
Time	D, 18, 6, 10	TT2, AA, AY, MM
Try	D, 27, 6	TT2, RR2, AY
Temperature	D, 7, 10, 9, 33, 25, 34	TT2, EH, MM, PP, ER1, SH, ER2
Volt	23, 35, 2D, 11	VV, OW, LL, TT1



A light beam striking the photocell causes the V1 voltage to stay at a logic high (4.5 V). When a person crossing breaks the light beam momentarily, the photocell changes its resistance to a higher value (megohms); consequently, the voltage V1 drops below the negative threshold level ( $V_{t-} = 1.8 \text{ V}$ ) of Nand gate IC4a. This in turn triggers the real monostable formed by Nand Gates IC4a and IC4b. The monostable sends a positive transient pulse, which triggers flip-flop IC2 to enable Nand gate IC4c. The IC4c Nand gate initiates the speech processing sequence produced by the speech processor (IC6) in conjunction with the EPROM memory (IC3) and the binary counter 74HC4040 (IC1).

As we already know, the /SBY output stays high when the speech processor is in the standby mode. This output is routed to the input of Nand gate IC4c. When IC4c is enabled by the flip-flop output, this Nand gate (IC4c) pulses the /ALD input low, causing the generation of the first allophone. Bear in mind that when an allophone starts, the /SBY output goes to a logic zero, causing the Nand gate output (IC4c) to return to a logic one. This way, the /ALD input is pulsed low until the speech processor accepts such input and starts generating the first speech-command sequence (allophone or pause).

The binary counter (74HC4040) is used here as an 11-bit EPROM scanner. When the first person has broken the light beam, for example, counter (IC1) starts being clocked by Nand gate IC4c via Nand gate IC4d. When the first speech sequence has been completed, EPROM 2716 sends a positive pulse (via output O6) to reset flip-flop IC2, thus disabling Nand gate IC4c and the speech processor. At this point, counter IC1 stops the binary counting sequence on the decimal number “5” because the first word “one” contains four allophones, a 200 ms pause, and the hex number 40H that gives a logic one at the output O6 of IC3 to reset flip-flop 4013 (IC2). Because flip-flop 4013 needs only a positive transient reset pulse, output O6 of the EPROM is first routed to a pair of inverters that form a buffer. The RC network connected at the output of Nand gate IC5d provides the required transient pulse to reset IC2; therefore, flip-flop IC3 is ready to be triggered again.

When a second person breaks the light beam, counter IC1 will start the counting sequence at number six (location six of the EPROM). This process is repeated until the last speech sequence occurs, then we add a pause, and the hex number C0H gives a logic one at the outputs O6 and O7 of the EPROM. In this form, the binary counter 74HC4040 and the flip-flop 4013 receive separately a reset pulse in order to restart the entire system. Table 2.8 shows a part of the speech data table necessary to concatenate the allophones. The complete table depends on the maximum number of persons that you wish to detect. Note that the maximum possible number in this configuration is 200. If you want to increase the range, augment the EPROM capacity by adding an EPROM 2732, 2764, or higher.

When all the speech data are over, outputs O6 and O7 of EPROM 2764 reset the flip-flop (4013) and the binary counter (4040). This is achieved by

**TABLE 2.8**  
 Allophone Sequence for the EPROM IC3 of Figure 2.7

Hex Address	Hex Data	
0	4	
1	2B	
2	3C	"zero"
3	35	
4	4	"pause"
5	40	"reset"
6	39	
7	F	"one"
8	F	
9	B	
A	4	"pause"
B	40	"reset"
C	D	
D	1F	two
E	4	
F	40	
10	1D	
11	E	three
12	13	
13	04	pause
.	:	
.	:	
:	:	
N-1	04	last pause
N	C0	(resets IC1 and IC2; 08-01
= 1100 0000)		

programming C0H after the last speech data where C0H corresponds to a binary output (11000000).

A different use for this circuit is when the operator needs spoken instruction just when the circuit is first turned on. In this case the flip-flop 4013 must be set by a network that is activated by the power-up transition.

## **2.6** Circuit Vocalizes Hex Code for a 4-Bit Input

The circuit shown in Figure 2.8 synthesizes audible words for hexadecimal codes corresponding to the binary input on lines A, B, C, and D. When you press the test switch, the input present on lines ABCD is stored in the quad



input and starts speaking the first allophone that was previously addressed by IC3. At this time the SBY output goes to a logic zero, causing the Nand gate output to go high; it is the positive edge transition that clocks counter IC2a. This means that, while the speech processor is saying the first allophone, the EPROM memory is addressing the second allophone to the speech processor waiting to be loaded by the next /ALD pulse. It is clear that counter IC2a scans those memory locations in sequence by driving the lower address bits A0 to A3. As a result, the EPROM delivers a preprogrammed sequence of instructions to the speech processor.

Since four bits (A0 to A3) are controlling the sequence of data, a maximum of 16 allophones is allowed at each block of memory. In this case the blocks of memory start in the following decimal locations: 16, 32, 48, 64, 80, 96, 128, 144, 160, 176, 192, 208, 224, and 240. This can be seen in Table 2.9, where the speech data for programming the EPROM are ready to use by selecting the hex address and the hex code.

The power-up reset pulse to counters IC2a and IC2b is given by the R1 C1 network, which provides a positive transient pulse to allow a zero state before beginning the operation of the whole circuit.

Following each report, the hex data instructions 4 and 44 reset the speech processor (internally) and gates IC2a and IC2b via output O6. This output can be buffered using the two remaining Schmitt trigger Nand gates (IC1c and IC1d) if counter 4520 is not reset properly. Diode D1 at the output O6 of IC4 prevents the output stage from receiving the power-up reset positive pulse caused by the R1 C1 network. Otherwise, this pulse would destroy the NMOS output transistor contained in the EPROM.

## 2.7 Circuit Vocalizes 8-Bit Binary Input

The method of vocalizing a binary code used here is similar to the one explained in Section 2.6, except for a few enhancements that will be explained in this section.

To read an 8-bit binary input with the speech processor, the circuit shown in Figure 2.9 uses an 8-bit latch 74HC373 (IC4), which stores the binary input on lines D1 to D8.

When power is first turned on, the power-up reset network (R1 C1) gives a positive transient pulse that resets IC3 and the half-monostable formed by Nand gates IC2b and IC2c. IC1 is a D-type flip-flop that is clocked by the half-monostable circuit at the negative edge transition.

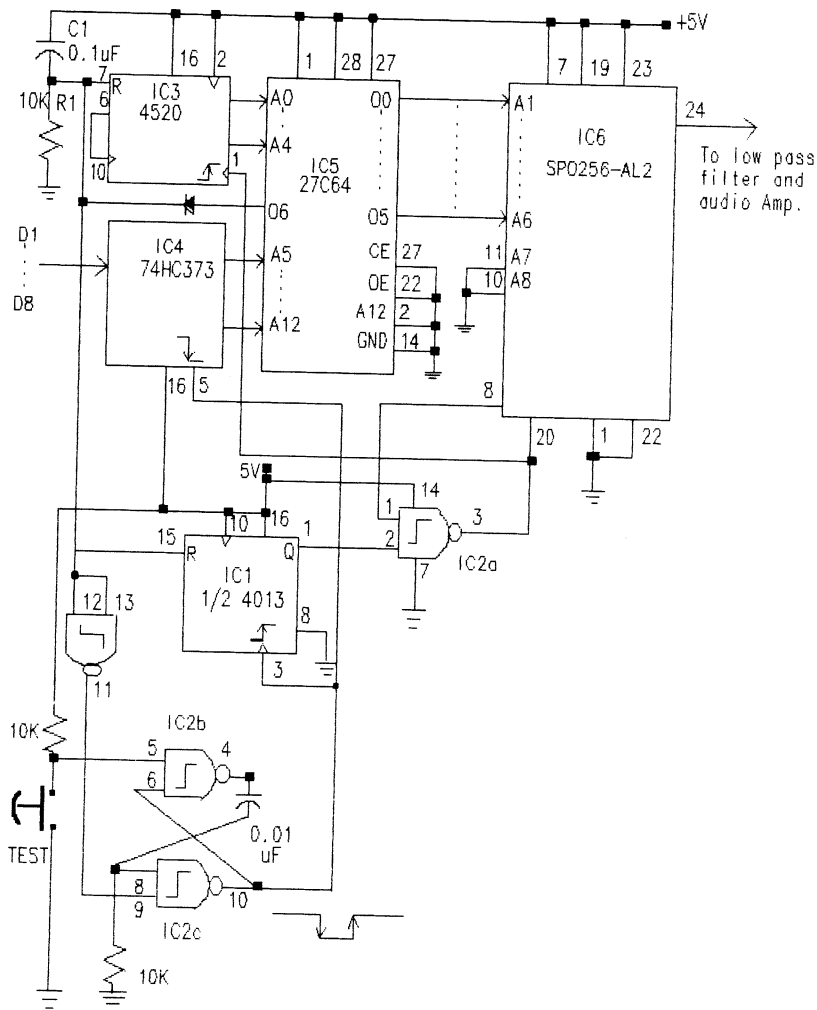
To start the circuit, press the TEST switch to generate a positive transient pulse; this pulse will latch the 8-bit data input. IC4 has a propagation time of 15 ns to present the data to the EPROM address inputs (A5 to A12). The EPROM used here (27C64-3) will take 300 ns to deliver the data output to the speech processor address inputs A1 to A6. By adding the time of each event we get 315 nanoseconds. If we consider the propagation time (tp) of the Nand



**TABLE 2.9**  
EPROM Program to be Loaded in EPROM 27C16

Hex Address	Hex Data	Hex Address	Hex Data
00	4	72	7
01	2B	73	7
02	3C	74	23
03	35	75	C
04	4	76	B
05	44	77	4
10	39	78	44
11	F	80	14
12	F	81	C
13	B	82	D
14	4	83	4
15	44	84	44
20	D	90	38
21	1F	91	18
22	4	92	6
23	44	93	B
30	1D	94	4
31	E	95	44
32	13	A0	14
33	4	A1	4
34	44	A2	44
40	28	B0	3F
41	28	B1	17
42	3A	B2	4
43	4	B3	44
44	44	C0	37
50	28	C1	37
51	28	C2	17
52	6	C3	4
53	23	C4	44
54	4	D0	21
55	44	D1	17
60	37	D2	4
61	37	D3	44
62	C	E0	17
63	C	E1	4
64	2	E2	44
65	29	F0	7
66	37	F1	7
67	4	F2	28
68	44	F3	28
70	37	F4	4
71	37	F5	44
			(RESETS IC5)
			(RESETS IC2a & IC2b)

*Continued*



**Figure 2.9** Circuit for the 8-bit vocalizer.

gate IC2b, we will have a total propagation time of 615 nanoseconds. It is, therefore, a smart choice to latch the 8-bit data input first and then get a small delay time to activate the flip-flop 4013 to give the first allophone time to be loaded; otherwise the first allophone will not be heard when the /ALD input is pulsed low. Sometimes glitches are still in process at the EPROM output, increasing the chances of hearing only garbage. In that case the circuit must be restarted. To solve this problem, the monostable formed by IC2b and IC2c is used. When the TEST switch is pressed, the negative edge transition of IC2c latches the 8-bit data input (D1 to D8). When the transient pulse of IC2c goes

**TABLE 2.10**  
Allophone Table Showing Ten Cases for Elaborating the Entire Table

Hex Address	Hex Data	Word
0	2B, 3C, 35, 4, 44	Zero
20	39, F, F, B, 4, 44	One
40	D, 1F, 4, 44	Two
60	10, E, 13, 4, 44	Three
.		
.		
140	D, 7, 7, B	Ten
280	D, 30, 7, 7, B, 2, D, 13	Twenty
B40	39, F, F, B, 2, 39, F, F, B, 1, 21, 27, C, C, 1, 15	One Hundred
1900	D, 1F, 4, 39, F, F, B, 1, 21, 27, C, C, 1, 15, 4, 44	Two Hundred
.		
.		
1F00	D, 1F, 4, 39, F, F, B, 1, 21, 27, C, C, 1, 15, 18, B, 15, 2, 28, 28, C, 28, 23, D, 13, 2, 28, 28, 6, 23, 4, 44	Two Hundred And Fifty Five.

back to a logic one, the positive edge clocks flip-flop 4013 that in turn enables the speech processor via the Nand gate IC2a.

The main difference between this circuit and the one in Section 2.6 is that there are five bits for scanning a maximum of 32 instructions, including pauses and reset bytes. Therefore, the blocks of memory fall in multiples of 32, 64, 96, and so on, depending upon the value of the input data on lines D1 to D8. The difficult part in implementing this circuit is to develop the hex data for the EPROM. But always bear in mind that here we are using maximum software with minimum hardware, which has the advantage of reducing costs of layout for pc board assembling. To help you understand how these data are formed, Table 2.10 shows a set of 10 different readings for some typical cases.

## 2.8 Improved Technique Vocalizes Binary Code for 8-Bit Input

So far our discussions have been primarily directed toward the use of an EPROM; however, the use of two smaller EPROMs in series reduces memory data and the time required for programming. The schematic is illustrated in Figure 2.10.

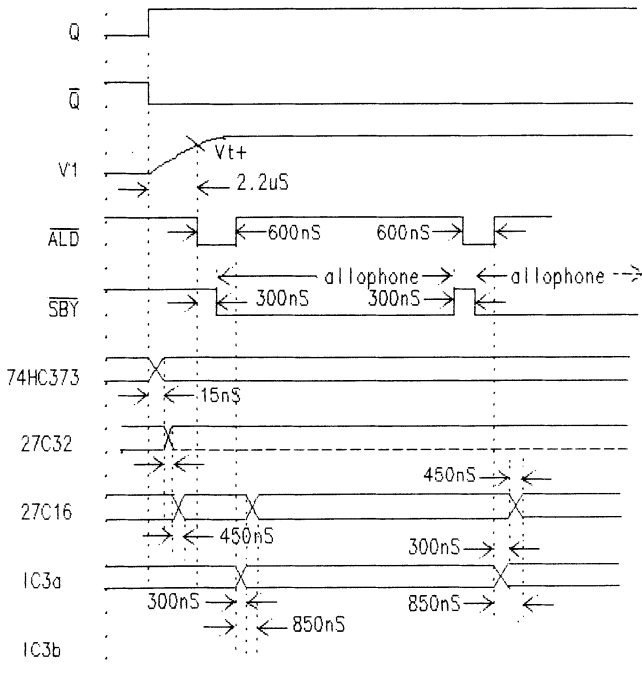


pronounce contain five words (e.g., “two hundred and fifty one”). Thus EPROM IC4 must be able to provide a maximum of five instructions. Even when three bits of counter 4520 (IC3b) are sufficient to scan the five instructions required, four bits are connected in this diagram to have the capacity of addressing a maximum of 16 words per message. Every word can be as long as 13 allophones, as determined by counter 4520 (IC3a). The upper address bits of the first EPROM (IC8) are controlled by an 8-bit transparent latch 74CH373 (IC1). These bits will address the starting point of a block of memory within the EPROM (IC4).

On the other hand, counter 4520 (IC3) will scan the lower address bits of IC5, which contain the data to concatenate the allophones in sequence. Figure 2.11 shows the timing diagram for the complete circuit.

It is necessary to explain how to develop the hexadecimal programs for both EPROMS in order to understand how the circuit works. First, we will see two examples using the timing diagram illustrated in Figure 2.11.

Once the circuit is turned on, press the normally open switch (S1) to set the D-type flip-flop. This causes a logic zero on  $\bar{Q}$  and a logic high on Q output after  $2.2\ \mu\text{s}$ . The negative edge of  $\bar{Q}$  latches the 8-bit data input (D1 to D8)



**Figure 2.11** Timing diagram for the circuit shown in Figure 2.10.

that are controlling the upper address bits of EPROM 27C32 (IC4). For example, if the 8-bit data input contains the binary number  $Q_8-Q_1 = 0000\ 0001$ , EPROM IC4 receives only a logic one at the address input A4; therefore, this EPROM goes to address 16 (10#h) where the data stored must correspond, in this case, to hex number 01#h. Now, the second EPROM (IC5) also receives the address sixteen (10#h). All this takes about 915 ns. Nand gate (IC1a) is still waiting to receive a voltage equal to or greater than 3.3 V ( $V_t + = 3.3\text{ V @ }5\text{ V}$ ). This time delay is given by the following equation:

$$t = RC \ln [V_{dd}/(V_{dd} - V_t +)] = 1.1 RC$$

where  $R = 1K$

$$C = 0.0022\ \mu F$$

With these RC values, Nand gate IC1a will be triggered by the Q output after  $2.2\ \mu s$  (2200 ns). This delay is inserted to give sufficient time to both EPROMs to find the first block of memory. After the  $2.2\ \mu s$  delay, Nand gate IC1a pulses low the /ALD input, and after 300 ns the speech processor begins to announce the first allophone. This 300 ns delay is caused by the standby (SBY) output which is also controlling Nand gate IC1a. Because of the propagation time delay of 300 ns caused by Nand gate IC1a, the /ALD input is held low for an interval of 600 ns. It is here when the output of Nand gate IC1a goes to a logic one which also clocks counter IC3a (1/2 4520), while the speech processor is still speaking the first allophone. Thus counter IC3a is now addressing the second data byte because it has incremented the lower address bits to  $Q_4-Q_1 = 0001$ . Accordingly, when the speech processor completes the first allophone, the SBY output goes back to a logic one, causing Nand gate IC1a to pulse the /ALD input low that will clock counter IC3a again after 600 ns. This process is repeated until the last allophone is heard. At this point, we have to store a pause (pa1 to pa5) to make the speech processor stop talking. In the next byte of EPROM IC5, we have to store the hex code 44H (01000100) to provide a logic one at the EPROM's output 06. This will cause a positive voltage at the output 06 (pin 16 of IC5) that will reset binary counter IC3a and will clock counter IC3b. The time interval of the said pulse will be given by the propagation time of IC3a plus IC5; that is, 850 ns. When this happens, counter IC3b is now incremented to the binary count  $Q_4-Q_1 = 0001$ , which generates the next sequentially address "17." This address is given by  $A_4 = A_1 = 1$  with all other address inputs equal to zero. Therefore, we store the number 80#h = 1000 0000#b in this new address that will give a positive transient voltage of 850 ns at the output 07 (pin 17 of IC4). This pulse resets binary counter IC3b (1/2 4520) and flip-flop 4013. Now the speech processor is disabled and the 8-bit latch (74HC373) is enabled. The circuit is now ready to be triggered again by momentarily pressing the normally open switch S1.

In the case previously explained, the circuit announced the single number

**TABLE 2.11**  
Five Typical Cases for the Words Zero, One, Two, Twenty, Twenty One

27C32																27C16					
Q7	Q6	Q5	Q4	Q3	Q2	Q1	Count	Hex	Hex							Count	Hex	Hex			
A10	A9	A8	A7	A6	A5	A4	A3-A0	Add	Data	A9	A8	A7	A6	A5	A4	A3-A0	Add	Data			
0	0	0	0	0	0	0	0000	00	00	0	0	0	0	0	0	0000	00	00			
0	0	0	0	0	0	1	0000	10	01	0	0	0	0	0	1	0000	10	2B			
0	0	0	0	0	1	0	0000	20	02	0	0	0	0	1	0	0000	20	39			
0	0	1	0	1	0	0	0000	140	14	0	1	0	1	0	0	0000	140	0D			
0	0	1	0	1	0	1	0000	150	14	0	0	0	0	0	0	0000	140	00			
0	0	1	0	1	0	1	0001	151	01	0	1	0	1	0	1	0000	10	00			

“one.” When the circuit has to enunciate a composed number like “twenty one,” the process is a little different. In that case, EPROM 27C32 (IC4) must contain three data bytes necessary to find the two messages “twenty” and “one,” and the data byte that resets IC3b and IC2.

By looking at the program of the controlling EPROM (IC4), you will see how composed numbers are formed by calling the correct messages located at specific addresses in EPROM 27C16 (IC5). Notice that EPROM 27C16 may also contain any specific message you want to add after a digital reading. You can call, for example, the word “volt” after the announced reading to indicate to the user the type of variable he is measuring.

Table 2.11 shows five typical and useful cases of the EPROM programs. As you can see, the 8-bit data input (Q1 to Q8) corresponds to EPROM IC4 inputs (A4 to A11). Note that A11 is not represented for Q8 because of space limitations. The word COUNT represents the 4-bit counter (IC3b) that is interfaced to the lower address inputs of EPROM 27C32 (IC4). In contrast, IC3a is the other 4-bit counter used to scan each allophone.

You can use this circuit as a functional box to the different projects presented in the following chapters. As you can see, Table 2.12 contains only the control program for composed numbers within the range of “zero” to “fifty nine.” But you can increase the range to create the maximum number of “two hundred and fifty five” due to the 8-bit data inputs that we are applying. The user can continue writing the program depending upon the application in mind. It is important to note that you can also change the scale of readings. For example, you may need to announce readings in the range of “zero” to “twenty five point five” (0.0 to 25.5) with a resolution of 0.1, or readings from “zero point zero zero” to “two point fifty five” (0.00 to 2.55) with a

**TABLE 2.12**  
EPROM Program for EPROM 27C16 (IC5)

Hex Address	Hex Data	Word
00	2B, 3C, 35, 4, 4	Zero
10	39, F, F, B, 4, 44	One
20	2B, 3C, 35, 4, 44	Two
30	10, E, 13, 4, 44	Three
60	28, 28, 3A, 4, 44	Four
70	28, 28, 6, 23, 4, 44	Five
80	37, 37, C, 2, 29, 37, 4, 44	Six
90	37, 37, 7, 7, 23, 7, B, 4, 44	Seven
A0	14, 2, D, 4, 44	Eight
B0	38, 18, 6, B, 4, 44	Nine
C0	D, 07, 07, B, 4, 44	Ten
D0	C, 2D, 7, 7, 23, C, B, 4, 44	Eleven
E0	D, 30, 7, 7, 2D, 23, 4, 44	twelve
F0	1D, 33, 1, 2, D, 13, B, 4, 44	thirteen
100	28, 3A, 1, 2, D, 13, B, 4, 44	Fourteen
110	28, C, 28, 2, D, 13, B, 4, 44	Fifteen
120	37, 37, C, 2, 29, 37, 2, D, 13, B, 4, 44	Sixteen
130	37, 37, 7, 23, 1D, B, 2, D, 13, B, 4, 44	Seventeen
140	14, 2, D, 13, B, 4, 44	Eighteen
150	B, 6, B, 2, D, 13, B, 4, 44	Nineteen
160	D, 30, 7, 7, B, 2, D, 13, 4, 44	Twenty
170	1D, 34, 2D, 13, 4, 44	Thirty
180	28, 3A, 2, D, 13, 4, 44	Forty
190	28, 28, C, 28, 2, D, 13, 4, 44	Fifty
1A0	37, 37, C, 2, 29, 37, 1, D, 13, 4, 44	Sixty
1B0	37, 37, 7, 23, C, B, 2, D, 13, 4, 44	Seventy
1C0	14, 2, D, 13, 4, 44	Eighty
1D0	B, 6, 11, 2, D, 13, 4, 44	Ninety
1E0	39, F, F, B, 1, 21, 4, 44	Hundred
1F0	9, 5, B, 11, 4, 44	Point
200	1D, 18, 2D, 2B, 1, B, 15, 4, 44	Thousand
210	10, C, C, 2D, 31, F, B, 4, 44	Million
220	18, B, 15, 4, 44	And
230	7, 2F, 3A, 4, 44	Error
240	28, F, 27, F, 37, 4, 44	Farads
250	10, C, 2D, C, 4, 44	Milli
260	10, 6, 8, 27, 35, 4, 44	Micro
270	10, C, B, C, 2D, 4, 44	Minute
280	9, C, 8, 35, 4, 44	Pico
290	23, 35, 2D, 11, 4, 44	Volt



resolution of 0.01. In both cases, EPROM IC5 remains unaltered; that is, with the same program, but you will have to program the controlling EPROM (IC4) to fit the scale with which you are working. (See Table 2.13.) The time required for programming EPROM IC4 is greatly reduced in comparison to the program employed in Section 2.7.

The circuit presented in this section can be simplified by using a micro-controller or a microsequencer; these topics will be treated in detail in the following section and chapters.

**TABLE 2.13**  
Program in Hex Code for EPROM 27C32 (IC4)

Hex Address	Hex Data		Hex Address	Hex Data
00	00	"zero"	100	10
01	80	"Reset"	101	80
10	01	"one"	110	11
11	80	"Reset"	111	80
20	02	"two"	120	12
21	80	"Reset"	121	80
30	03	"three"	130	13
31	80	.	131	80
40	04	.	140	14
41	80	.	141	80
50	05	.	150	14
51	80	.	151	01
60	06	.	152	80
61	80	.	160	14
70	07	.	161	02
71	80	.	162	80
80	08	.	170	14
81	80	.	171	03
90	09	.	172	80
91	80	.	180	14
A0	0A	"ten"	181	04
A1	80	"Reset"	182	80
B0	0B	.	190	14
B1	80	.	191	05
C0	0C	.	192	80
C1	80	.	1A0	14
D0	0D	.	1A1	06
D1	80	.	1A2	80
E0	0E	.	1B0	14
E1	80	.	1B1	07
F0	0F	.	1B2	80
F1	80	.	1C0	14

*Continued*

*Continued*

1C1	08		2C1	04	
1C2	80		2C2	80	
1D0	14	.	2D0	16	
1D1	09	.	2D1	05	
1D2	80	.	2D2	80	
1E0	15	"thirty"	2E0	16	
1E1	80	.	2E1	06	
1F0	15	.	2E2	80	
1F1	01	.	2F0	16	
1F2	80	.	2F1	07	
200	15		2F2	80	
201	02		300	16	
202	80		301	08	
210	15		302	80	
211	03		310	16	
212	80		311	09	
220	15		312	80	
221	04		320	17	"fifty"
222	80		321	80	"Reset"
230	15		330	17	"fifty-
231	05		331	01	one"
232	80		332	80	
240	15		340	17	
241	06		341	02	
242	80		342	80	
250	15		350	17	
251	07		351	03	
252	80		352	80	
260	15		360	17	
261	08		361	04	
262	80		362	80	
270	15		370	17	
271	09		371	05	
272	80		372	80	
280	16	"forty"	380	17	
281	80	"Reset"	381	06	
290	16	"forty-	382	80	
291	01	one"	390	17	
292	80	"Reset"	391	07	
2A0	16		392	80	
2A1	02		3A0	17	
2A2	80		3A1	08	
2B0	16		3A2	80	
2B1	03		3B0	17	"fifty-
2B2	80		3B1	09	nine"
2C0	16		3B2	80	"Reset"

*Continued*

## 2.9 Microcontroller Routine Handles the Basic Functions of a Speech Processor

In this section we will be headed to the development of the routines necessary to drive the two most commonly used speech processors (SPO256-AL2 from Microchip and DT1050 from National Semiconductor) by using the versatile microcontroller ( $\mu$ C) 8748.

A brief description of the 8748 architecture will be helpful to start developing the program.

### *8748 Architecture*

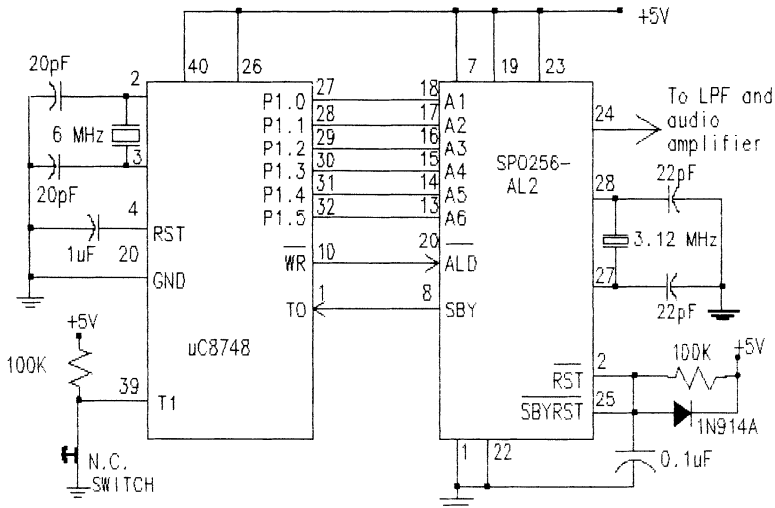
The functional blocks of the 8748 family are now described. The arithmetic section is for basic data manipulation and is divided in the following blocks: arithmetic logic unit (ALU), accumulator, carry flag, and instruction decoder. The program memory is stored in the resident EPROM memory, which is formed by 1024 bytes for the  $\mu$ C 8748 and 2048 bytes for the  $\mu$ C 8749. There are 27 I/O lines grouped as three ports of eight lines each. T0, T1, and INT serve as inputs and are testable with the conditional JUMP instruction without needing to load an input port into the accumulator. Finally, a timer/counter is also contained to aid the user in counting external events or generate accurate time delays without disturbing the processor for these functions. For complete and detailed information about the  $\mu$ C 8748 or  $\mu$ C 8749, consult the reference section at the end of this chapter.

Figure 2.12 shows the  $\mu$ C 8748 interfaced with the SPO256-AL2 to provide full control of the input and output lines of the speech processor. Input T1 will be the external switch that initiates the speech processing sequence; therefore, the  $\mu$ C 8748 will be reading the logical value of input T1. When the user presses the test switch, a logic high will be present momentarily at the input T1, causing the  $\mu$ C to initiate the control program.

At this point, we want the  $\mu$ C program to drive the speech processor by sending a preprogrammed group of allophones. In this case, we will program it to hear the words “one, two, three, and four.” The speech data will be programmed in page three of the  $\mu$ C’s internal EPROM. We know that page three consists of 256 bytes available for speech data programmed by the user. In this manner, a maximum of 256 allophones can be stored for any other specific application you may have in mind. In this example, the four words occupy 17 bytes (11H hexadecimal). The flowchart illustrated in Figure 2.13 shows the steps that must be executed to control the speech processor with the  $\mu$ C 8748.

The following routine (see Table 2.14) corresponds to the flowchart shown in Figure 2.13. It is a good example of how the  $\mu$ C 8748 is instructed to make the speech processor speak four word numbers.

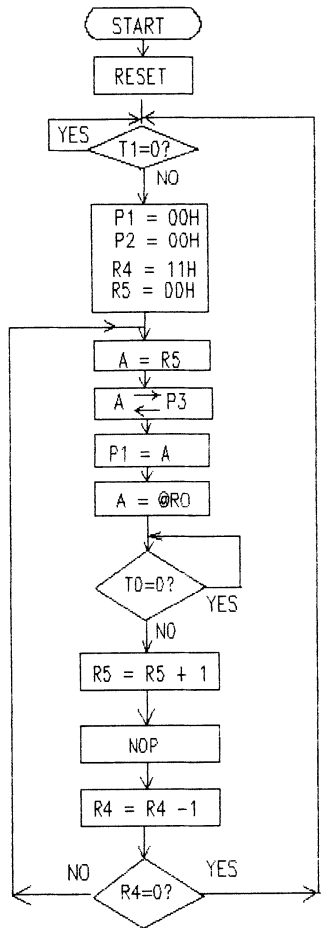
In the software program shown in Table 2.14, the instruction that pulses the



**Figure 2.12** SPO256-AL2 controlled by the  $\mu$ C8748.

write output (/WR) low is MOVX @R0, A. This negative transient pulse stays low for 5  $\mu$ s when a 6 MHz crystal is used. At this point, the speech processor starts the first speech sequence, causing the standby output (/SBY) to go low while the speech processor is speaking the first utterance. Our next instruction must read the STANDBY status; this is made using the JNT0 instruction. When T0 goes to a logic high, the program jumps to the next instruction. In order to hear the next allophone, seven instructions must be executed to bring out the respective speech data. This process causes a total delay of 22.5  $\mu$ s; that is, 0.0225 ms for listening to the next allophone. This short delay will not affect the natural sound because the duration of the allophones varies from 10 to 420 ms. The routine must search for the next speech data if you want to reduce the 22.5  $\mu$ s delay, and have it ready to load it into the speech processor before reading the STANDBY status. This new method is shown in the program in Table 2.15. Register R4 is used as a counter that is decremented every time the speech processor speaks a new allophone. This is performed by the DJNZ instruction on address 21H. There is also register R5 which is incremented in order to find the next allophone on page three of ROM. The instruction to find the next allophone is “MOVP3 A, @A” located on address 19H.

The enhanced routine shown in Table 2.15 reduces the 22.5  $\mu$ s delay to 7.5  $\mu$ s by introducing the instructions: “INC R5,” “MOV A, R5,” and “MOVP3 A, @A,” then proceeding to read the SBY status of the speech processor. (There is plenty of time to do this because the speech processor is a relatively slow device compared with the  $\mu$ C 8748.)



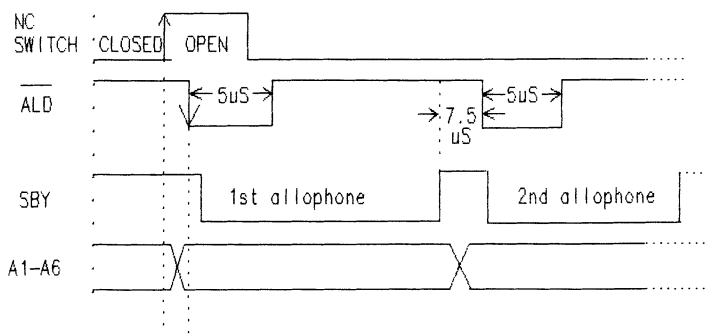
**Figure 2.13** Flowchart for the control routine of the  $\mu\text{C 8748}$ .

**TABLE 2.14**  
Software for the  $\mu\text{C8748}$  Interfaced with the SPO256-AL2

Add	Op	Code	Mnemonic	Comments
00H	04	05	JMP 05H	;jump to address 5
05	46	05	JNT1 05H	;jump to address 5 if T1 is low
07	04	10	JMP 10H	;jump to address 10H
10	99	00	ANL P1, #00H	;clear port 1
12	9A	00	ANL P2, #00H	;clear port 2
14	BC	11	MOV R4, #11H	;load Reg 4 with the # of ;allophones
16	BD	00	MOV R5, #00H	;clear register 5

18	FD	MOV A, R5	;move reg 5 to ACC
19	E3	MOV P3 A, @A	;move page 3 of ROM to ACC
1A	39	OUTL P1, A	;load speech data to port 1
1B	90	MOVX @R0, A	; /WR output is pulsed low
1C	26 1C	JNTO 1CH	;read the SBY status of the SP
1E	1D	INC R5	;inc reg 5 to select new
1F	00	NOP	;allophone
20	EC 18	DJNZ R4, 18H	;decrement the number of
			;allophones
21	04 05	JMP 05H	;jump to address 05H
300	04		;speech data stored in page 3
301	2B		;consisting of the words one,
302	3C		;two, three, four.
303	35		;pause (pa3) is added between
304	04		;each word.
305	39		
306	0F		
307	0F		
308	0B		
309	04		
30A	0D		
30B	1F		
30C	04		
30D	1D		
30E	0E		
30F	13		
310	04		

The following timing diagram (see Figure 2.14) corresponds to the program shown in Table 2.15. The program starts when the normally closed switch is opened momentarily. After 11 instructions ( $27.5 \mu\text{s}$ ), the first allophone will be heard. The /ALD waveform illustrates the  $7.5 \mu\text{s}$  time delay before listening to the next allophone (speech data).



**Figure 2.14** Timing diagram for the program of Table 2.15.

**TABLE 2.15**  
Enhanced Software for the  $\mu$ C8748 Interfaced with the SPO256-AL2

Add	Op Code	Mnemonic	Comments
00	27	CLR A	;clear accumulator
01	04 05	JMP 05H	;go to address 5
05	46 05	JNT1 05H	;jump to 05H if T1 is low
07	04 10	JMP 10H	;go to address 10H
10	99 00	ANL P1, #00H	;clear port 1
12	9A 00	ANL P2, #00H	;clear port 2
14	BC 11	MOV R4, #11H	;reg 4 is loaded for 17 ;allophones
16	BD 00	MOV R5, #00H	;clear reg 5
18	39	OUTL P1, A	;load speech data to port 1
19	90	MOVX $\mu$ R0, A	;pulse low the /WR output
1A	1D	INC R5	;increment reg 5 to select next ;allophone
1B	FD	MOV A, R5	;increment ACC to search next ;speech data
1C	E3	MOV P3 A, $\mu$ A	;load new allophone to ACC
1D	26	JNT0 1DH	;test SBY input
1F	EC 18	DJNZ R4, 18	;decrement R4 and test for zero
21	04 00	JMP 00H	;go to address 00H
300	04		;speech data stored in page 3
301	2B		;consisting of the words one,
302	3C		;two, three, four.
303	35		
304	04		
305	39		
306	0F		
307	0F		
308	0B		
309	04		
30A	0D		
30B	1F		
30C	03		
30D	1D		
30E	0E		
30F	13		
310	04		

In the following chapters, we will be using this routine for more complex applications. The microcontroller 8748 is a powerful device that can be applied in almost any kind of instrumentation system. It is now a cheap and commercially available device that you can find through most mail-order distributors.

## 2.10 Field Programmable Controller Am29CPL100 Drives the Speech Processor SPO256-AL2

The Advanced Micro Devices Am29PL100 is a family of compatible field programmable controller (FPC) devices that permits the designer to implement complex state machines and controllers by programming a specific sequence of instructions in its internal PROM or EPROM. Using these devices results in a savings of board area, power, and critical design time.

The Am29PL100 family members are shown in Table 2.16.

This single-chip CMOS device contains a set of powerful instructions that allow conditional branching, conditional looping, conditional subroutine call, and multiway branch. Engineers can develop optimal solutions with the computing power to control peripherals, instead of using SSI and MSI devices. All Am29CPL100 devices integrate the elements of an intelligent controller into a single chip. These flexible field programmable controllers replace multiple chip alternatives consisting of several programmable and MSI devices, or costly ASIC devices. In this section we will see how FPCs simplify a speech processor's control.

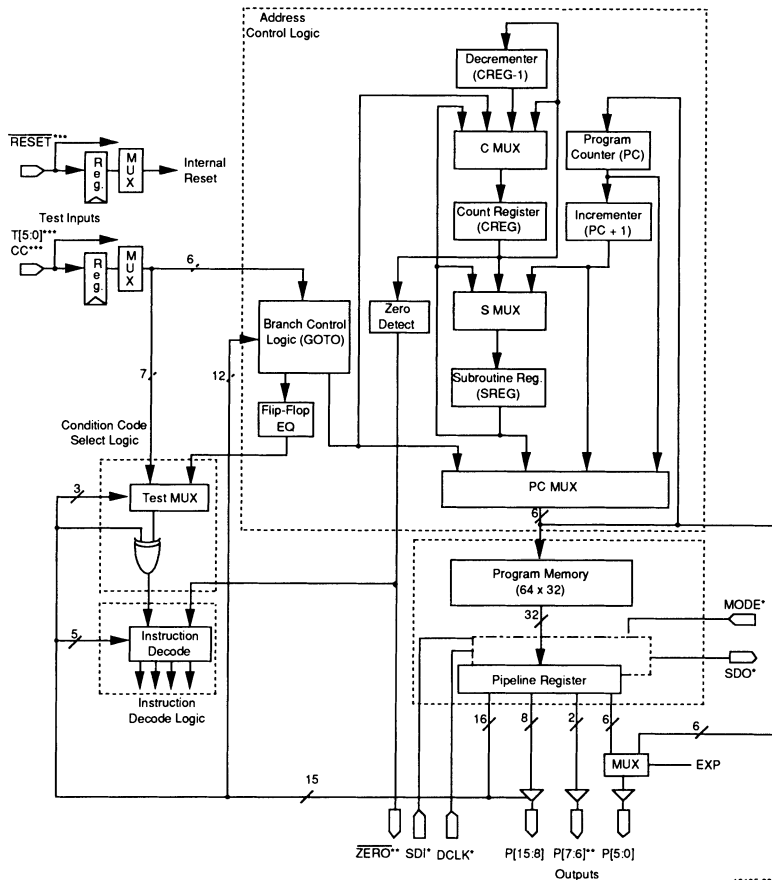
Figure 2.15 shows the simplified block diagram of the FPC Am29CPL151. Table 2.17 shows the instruction set. The Am29CPL100 architecture is composed of three basic functional blocks:

- Address sequencer. Controls the sequence in which instructions are fetched from program memory.
- Program memory. Field programmable controllers feature on-chip program memory for minimum chip count and maximum system speed. Memory size ranges from 64 to 512 words for handling even the most complex controls algorithms. Memory output is fed to the pipeline register.
- Pipeline Register. A portion of the pipeline register's contents provides the outputs that control other devices in the system. The other portion of the register is fed back to the address sequencer.

**TABLE 2.16**  
Am29PL100 Family Members

Part	Memory	Words	Inputs	Outputs	Instructions	Clock Rate
Am29CPL151	EPROM	64	7	16	29	30 MHz
Am29CPL152	EPROM	64	8	16	28	25 MHz
Am29CPL154	EPROM	64	8	16	28	25 MHz





10135-006A

\* These pins available only in SSR mode.

\*\* These pins available only in normal mode.

\*\*\* Each of the T[5:0], RESET, and CC inputs can be individually registered or left unregistered as a programmable option.

**Figure 2.15** Block diagram of the FPC Am29CPL151. (Copyright © Advanced Micro Devices, Inc. June 1988. Publication #10135 p. 7. Reprinted with permission of copyright owner. All rights reserved.)

The Am29CPL100 FPC structure is a cross between a state machine and a microcontroller, giving it a combination of high-speed operation with the convenience of programming using an instruction set. Microcode features a wide instruction word that allows operations to be performed in parallel instead of sequentially, as in microcontrollers. This results in higher performance, but without added design complexity. Using AMD's FPC assembler (ASM14X), code is written in high-level language constructs, using symbolic values for addresses and I/O.

As a first approach, we will use the Am29CPL151 to control the speech

**TABLE 2.17**  
Am29CPL141/CPL151 Microprogram Instruction Set

	opcode	mnemonics	Assembler statement
(1) *	19	GOTOPL	IF (cond) THEN GOTO PL(data)
(2) *	0F	GOTOTM	IF (cond) THEN GOTO TM(data)
(3)	0B	GOTOPLZ	IF (CREG = 0) THEN GOTO PL(data)
or	0B	GOTOPLZ	IF (CREG = 0) THEN GOTO PL(data) AND CLEAR-EQ
(4) *	18	FORK	IF (cond) THEN GOTO PL(data) ELSE GOTO (SREG)
(5) *	1C	CALPL	IF (cond) THEN CALL PL(data)
(6) *	1D	CALPLN	IF (cond) THEN CALL PL(data), NESTED
(7) *	1E	CALTM	IF (cond) THEN CALL TM(data)
(8) *	1F	CALTMN	IF (cond) THEN CALL TM(data), NESTED
(9)	04	LDPL	IF (cond) THEN LOAD PL(data)
(10)	05	LDPLN	IF (cond) THEN LOAD PL(data), NESTED
(11)	06	LDTM	IF (cond) THEN LOAD TM(data)
(12)	07	LDTMN	IF (cond) THEN LOAD TM(data), NESTED
(13)	15	PSH	IF (cond) THEN PUSH
(14)	17	PSHN	IF (cond) THEN PUSH, NESTED
(15)	14	PSHPL	IF (cond) THEN PUSH, LOAD PL(data)
(16)	16	PSHTM	IF (cond) THEN PUSH, LOAD TM(data)
(17)	02	RET	IF (cond) THEN RET
(18)	03	RETN	IF (cond) THEN RET, NESTED
(19)	00	RETPL	IF (cond) THEN RET, LOAD PL(data)
(20)	01	RETPLN	IF (cond) THEN RET NESTED, LOAD PL(data)
(21)	09	DEC	IF (cond) THEN DEC
(22)	0C	DECPL	WHILE (CREG <> 0) WAIT ELSE LOAD PL(data)
(23)	0E	DECTM	WHILE (CREG <> 0) WAIT ELSE LOAD TM(data)
(24) *	1B	DECGOPL	IF (cond) THEN GOTO PL(data) ELSE WHILE (CREG <> 0) WAIT
(25) *	1A	WAIT	IF (cond) THEN GOTO PL(data) ELSE WAIT
(26)	08	LPPL	WHILE (CREG <> 0) LOOP TO PL(data)
or	08	LPPL	WHILE (CREG <> 0) LOOP TO PL(data) AND CLEAR-EQ
(27)	0A	LPPLN	WHILE (CREG <> 0) LOOP TO PL(data) ELSE NEST
or	0A	LPPLN	WHILE (CREG <> 0) LOOP TO PL(data) AND CLEAR-EQ ELSE NEST
(28)	0D	CONT	CONTINUE
(29)	10 - 13	CMP	CMP TM(mask) TO PL(constant)

\* = If test field selects EQ, and if branch is taken, EQ flag is cleared

processor SPO256-AL2 and to make it speak a preprogrammed sequence of allophones. Figure 2.16 shows the circuitry configuration to achieve this specific task.

As shown in Figure 2.16, Nand gates IC1a and IC1b form a monostable that gives a positive transient pulse of 0.1 s each time the normally open test switch is pressed. The timing equation for this monostable is given by

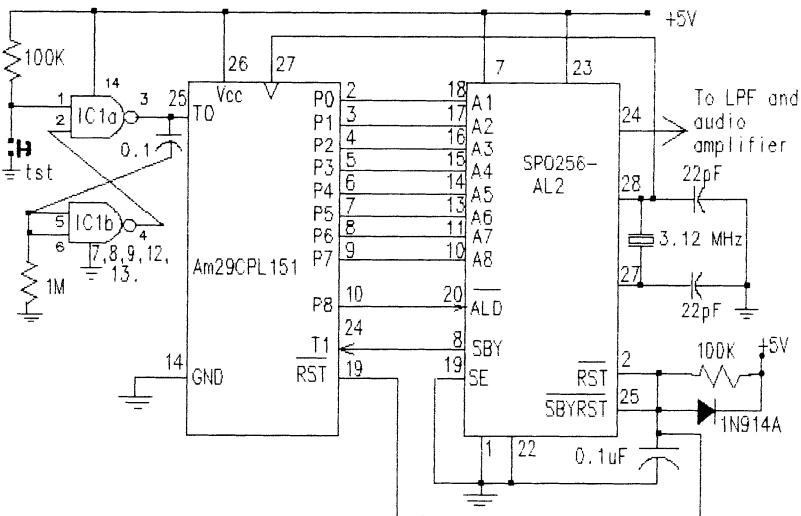
$$t = RC \ln [V_{dd}/V_{t-}]$$

Substituting  $V_{dd}$  and  $V_{t-}$ , we get:

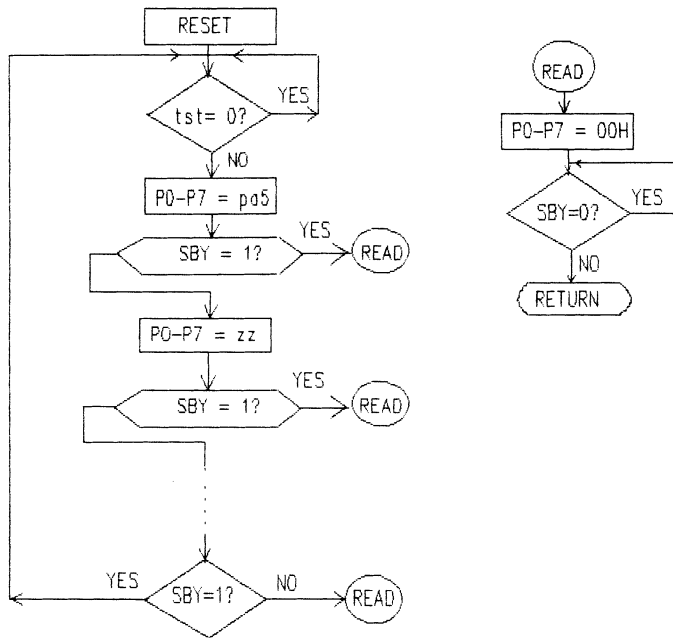
$$t = RC \ln [5V/1.8V]$$

$$t = 1.02 RC$$

In this form, the FPC (IC2) will read the logic high in order to start the microprogram; otherwise, the program keeps reading the low state of T0. To send the data that correspond to the allophone table of the speech processor SPO256-AL2, we will be using 6 of the 16 outputs of the FPC. The fixed clock frequency of 3.12 MHz used to clock the speech processor is also applied to the field programmable controller. This is a good way to reduce the number of components, by not having to build an extra oscillator. You will see by looking at Figure 2.16 that outputs P0 to P7 of IC2 give the desired address to the speech processor. Considering that the speech processor contains only 63 speech locations, P6 and P7 will be providing a logic zero in this circuit all the time. Figure 2.17 illustrates the flowchart required to bring out sequen-



**Figure 2.16** FPC Am29CPL151 interface with the speech processor SPO256-AL2.



**Figure 2.17** Flowchart to develop the control program of the circuit shown in Figure 2.16.

tially a certain number of allophones. The process starts when the user presses the test (tst) switch that causes a logic high at the T0 input of the FPC. When the FPC detects that T0 is high, it will jump automatically to the next instruction; therefore, while the microinstruction asks for the SBY status of the SPO256-AL2, the first allophone will be issued by the FPC.

It is clear that the SBY output will be in a logic high because the inputs A1 to A8 have not yet been pulsed low; therefore, the program now calls conditionally the “read” subroutine which pulses the inputs A1 to A8 low. The /ALD input is disabled because the strobe enable input of the speech processor is tied to a logic low. In this mode of operation there is an extra byte penalty for each allophone issued, and pause pa1 cannot be used. This extra byte, consisting of zeroes at the inputs A1 to A8, is supplied with the output “pa1” in subroutine “pl(read).” It is better to use this mode of operation because we have to define the allophone we desire once only. Otherwise, we would define every allophone twice along with the word “ald” in the first instruction and without the word “ald” in the second instruction, just to indicate that the /ALD input is pulsed low in the second instruction.

When the inputs A1 to A8 receive the byte 00#h, it causes the SBY output to stay low for an interval appropriate to the first speech data, in this case, the

pause pa5. The program now proceeds to read the SBY status. When the SBY output returns to a logic high, the program returns to the next instruction that contains the second speech entry point. This process will be repeated until the last speech data has been issued. Then the program will jump conditionally to the first line with the instruction “if (sby) then goto pl(zero)”; where “zero” is the label of the first instruction. Notice that the particular allophone you wish to listen to must be specified before each instruction and separated by a comma. When you need more outputs, they must be specified, for example, as output1 + output2. This is why the program contains the required allophone or pause to maintain any one or more of the outputs P1 to P8 at a logic high for every instruction, except when the program goes to the byte “zero” (see line 28). Bear in mind that the last speech data must be a pause (pa2 to pa5) in order to stop the processor saying the last allophone.

Table 2.18 shows the ascii file that represents the program for the FPC Am29CPL151 that is suitable for our purposes because it supports a maximum of 64 instructions, while our program spends only 30. To assemble and simulate the program with the ASM14X software package, you have to create a pure ascii file containing the key words DEVICE, DEFAULT, DEFINE, BEGIN, and END as illustrated in Table 2.18.

**TABLE 2.18**  
Program to Control the Speech Processor with the FPC Am29CPL151

```

DEVICE (CPL141)

DEFAULT = 1;

DEFINE "test inputs"
tst = t0 "when the n.o. switch is closed t0 goes high"
sby = t1 "STANDBY goes to zero when A1-A8 are zero"

"output control bits speech data = 59 allophones plus five pauses"
pa1 = 00#h pa2 = 01#h pa3 = 02#h pa4 = 03#h pa5 = 04#h
oy = 05#h ay = 06#h eh = 07#h kk3 = 08#h pp = 09#h
jh = 0A#h nn1 = 0B#h ih = 0C#h tt2 = 0D#h rr1 = 0E#h
ax = 0F#h mm = 10#h tt1 = 11#h dh1 = 12#h iy = 13#h
ey = 14#h dd1 = 15#h uw1 = 16#h ao = 17#h aa = 18#h
yy2 = 19#h ae = 1A#h hh1 = 1B#h bb1 = 1C#h th = 1D#h
uh = 1E#h uw2 = 1F#h aw = 20#h dd2 = 21#h gg3 = 22#h
vv = 23#h gg1 = 24#h sh = 25#h zh = 26#h rr2 = 27#h
ff = 28#h kk2 = 29#h kk1 = 2A#h zz = 2B#h ng = 2C#h
ll = 2D#h ww = 2E#h xr = 2F#h wh = 30#h yy1 = 31#h
ch = 32#h er1 = 33#h er2 = 34#h ow = 35#h dh2 = 36#h
ss = 37#h nn2 = 38#h hh2 = 39#h or = 3A#h ar = 3B#h
yr = 3C#h gg2 = 3D#h el = 3E#h bb2 = 3F#h;

```

```

BEGIN
"the speech processor announces the words: zero, one, two, three, bye"
"wait for test input to go high -- Strobe Enable = 0 (pin 19)"
"1"zero:pa5,    if (not tst) then goto pl(zero);
"2"    pa5,    if (sby) then call pl(read);    "400 ms pause"
"3"    zz,     if (sby) then call pl(read);    "ze"
"4"    yr,     if (sby) then call pl(read);    "r"
"5"    ow,     if (sby) then call pl(read);    "o"
"6"    pa5,    if (sby) then call pl(read);    "400 ms pause"
"7"    ww,     if (sby) then call pl(read);    "etc. . . ."
"8"    ax,     if (sby) then call pl(read);
"9"    ax,     if (sby) then call pl(read);
"10"   nn1,    if (sby) then call pl(read);
"11"   pa5,    if (sby) then call pl(read);
"12"   tt2,    if (sby) then call pl(read);
"13"   uw2,    if (sby) then call pl(read);
"14"   pa5,    if (sby) then call pl(read);
"15"   th,     if (sby) then call pl(read);
"16"   rr1,    if (sby) then call pl(read);
"17"   iy,     if (sby) then call pl(read);
"18"   pa5,    if (sby) then call pl(read);
"19"   ff,     if (sby) then call pl(read);
"20"   ff,     if (sby) then call pl(read);
"21"   or,     if (sby) then call pl(read);
"22"   pa5,    if (sby) then call pl(read);
"23"   pa5,    if (sby) then call pl(read);
"24"   bb1,    if (sby) then call pl(read);
"25"   ay,     if (sby) then call pl(read);
"26"   pa5,    if (sby) then call pl(read);
"27"   pa2,    if (sby) then goto pl(zero);

"routine for reading the standby status of the speech processor"
"28"read:pa1,  continue;"allow SBY pin to go low in 300ns"
"29"    pa1,   if (not sby) then goto pl(stay);  "reading SBY"
"30"    pa1,   if (sby) then ret;

                .org 63#d
"31"            goto pl(zero);

END.

```

The program shown in Table 2.18 specifies the device that we are using, in this case the Am29CPL141, denoted as CPL141. Notice that Figure 2.16 specifies the Am29CPL151 because this is a version with a self-contained EPROM. The DEFAULT section, equal to 1, means that unspecified fuses will remain unblown, thus leaving unspecified microcode words and files at a logic level one. According to the circuit shown in Figure 2.16, the DEFINE section is where we assign names to the test inputs and output control bits; it is here that the 5 pauses, 59 allophones, and the /ALD output are assigned with the hexadecimal value required for addressing the speech processor correctly. In this

manner, we will be applying the allophones or pauses in our main program in an easy-to-follow format.

The assembler program produces three files; JEDEC, PROM bit, and the error file. For more information on how to install and run the assembler (ASM14X) and the simulator (SIM14X), consult the files with the extension “.TXT” contained in the software package ASM14X from Advanced Micro Devices. Assembler statements are written using any familiar word processor. Assembly is done on any personal computer using ASM14X. Error messages are returned for incorrect syntax. Once the error messages have been eliminated, a JEDEC file is produced with the assembler output. Table 2.19 shows the JEDEC file that is generated by the assembler program in order to be downloaded to a standard logic programmer. Table 2.20 illustrates the PROM bit pattern file. By looking at this PROM bit pattern, you will see that the outputs and inputs that we specified for controlling the speech processor are the equivalent outputs but in binary code. Line “28” is responsible for pulsing low the outputs P0–P7 of the FPC to load a speech entry point.

**TABLE 2.19**  
JEDEC File for the FPC Am29CPL151

[speech3.doc]	JEDEC map for device [cp1141]*
L0000	0 00110 0 111 111111 111111111111011 *
L0032	0 00011 1 110 100100 111111111111011 *
L0064	0 00011 1 110 100100 1111111111010100 *
L0096	0 00011 1 110 100100 1111111111000011 *
L0128	0 00011 1 110 100100 1111111111001010 *
L0160	0 00011 1 110 100100 111111111111011 *
L0192	0 00011 1 110 100100 1111111111010001 *
L0224	0 00011 1 110 100100 1111111111110000 *
L0256	0 00011 1 110 100100 1111111111110000 *
L0288	0 00011 1 110 100100 1111111111110100 *
L0320	0 00011 1 110 100100 1111111111111011 *
L0352	0 00011 1 110 100100 1111111111110010 *
L0384	0 00011 1 110 100100 1111111111110000 *
L0416	0 00011 1 110 100100 1111111111111011 *
L0448	0 00011 1 110 100100 11111111111100010 *
L0480	0 00011 1 110 100100 1111111111110001 *
L0512	0 00011 1 110 100100 11111111111101100 *
L0544	0 00011 1 110 100100 1111111111111011 *
L0576	0 00011 1 110 100100 1111111111110111 *
L0608	0 00011 1 110 100100 111111111111010111 *
L0640	0 00011 1 110 100100 111111111111000101 *
L0672	0 00011 1 110 100100 1111111111111011 *
L0704	0 00011 1 110 100100 1111111111111011 *
L0736	0 00011 1 110 100100 11111111111100011 *
L0768	0 00011 1 110 100100 1111111111111001 *

```

L0800 0 00011 1 110 100100 111111111111011 *
L0832 0 00110 1 110 111111 111111111111110 *
L0864 0 00110 0 110 100100 111111111111111 *
L0896 0 11101 1 110 000000 111111111111111 *
L0928 00000000000000000000000000000000*
L0960 00000000000000000000000000000000*
L0992 00000000000000000000000000000000*
L1024 00000000000000000000000000000000*
L1056 00000000000000000000000000000000*
L1088 00000000000000000000000000000000*
L1120 00000000000000000000000000000000*
L1152 00000000000000000000000000000000*
L1184 00000000000000000000000000000000*
L1216 00000000000000000000000000000000*
L1248 00000000000000000000000000000000*
L1280 00000000000000000000000000000000*
L1312 00000000000000000000000000000000*
L1344 00000000000000000000000000000000*
L1376 00000000000000000000000000000000*
L1408 00000000000000000000000000000000*
L1440 00000000000000000000000000000000*
L1472 00000000000000000000000000000000*
L1504 00000000000000000000000000000000*
L1536 00000000000000000000000000000000*
L1568 00000000000000000000000000000000*
L1600 00000000000000000000000000000000*
L1632 00000000000000000000000000000000*
L1664 00000000000000000000000000000000*
L1696 00000000000000000000000000000000*
L1728 00000000000000000000000000000000*
L1760 00000000000000000000000000000000*
L1792 00000000000000000000000000000000*
L1824 00000000000000000000000000000000*
L1856 00000000000000000000000000000000*
L1888 00000000000000000000000000000000*
L1920 00000000000000000000000000000000*
L1952 00000000000000000000000000000000*
L1984 00000000000000000000000000000000*
L2016 00000000000000000000000000000000*
L2048 0 *
L2049 0 *
L2050 0 *
L2051 0 *
L2052 0 *
L2053 0 *
L2054 0 *
L2055 0 *
L2056 0 *
L2057 0 *
C4EB7*
21C8

```



**TABLE 2.20**  
PROM Bit Pattern for the FPC Am29CPL151

PROM Contents:

hex <dec>	OE	OPCODE	POL	TEST	DATA	OUTPUT
000 < 0>	[ 1	11001	1	000	000000	0000000000000100 ]
001 < 1>	[ 1	11100	0	001	011011	0000000000000100 ]
002 < 2>	[ 1	11100	0	001	011011	0000000000010101 ]
003 < 3>	[ 1	11100	0	001	011011	0000000000011110 ]
004 < 4>	[ 1	11100	0	001	011011	0000000000011010 ]
005 < 5>	[ 1	11100	0	001	011011	0000000000000100 ]
006 < 6>	[ 1	11100	0	001	011011	0000000000010110 ]
007 < 7>	[ 1	11100	0	001	011011	0000000000000111 ]
008 < 8>	[ 1	11100	0	001	011011	0000000000000111 ]
009 < 9>	[ 1	11100	0	001	011011	0000000000000101 ]
00A < 10>	[ 1	11100	0	001	011011	0000000000000100 ]
00B < 11>	[ 1	11100	0	001	011011	0000000000001101 ]
00C < 12>	[ 1	11100	0	001	011011	0000000000001111 ]
00D < 13>	[ 1	11100	0	001	011011	0000000000000100 ]
00E < 14>	[ 1	11100	0	001	011011	0000000000001101 ]
00F < 15>	[ 1	11100	0	001	011011	0000000000000110 ]
010 < 16>	[ 1	11100	0	001	011011	0000000000001001 ]
011 < 17>	[ 1	11100	0	001	011011	0000000000000100 ]
012 < 18>	[ 1	11100	0	001	011011	0000000000010100 ]
013 < 19>	[ 1	11100	0	001	011011	0000000000010100 ]
014 < 20>	[ 1	11100	0	001	011011	0000000000011101 ]
015 < 21>	[ 1	11100	0	001	011011	0000000000000100 ]
016 < 22>	[ 1	11100	0	001	011011	0000000000000100 ]
017 < 23>	[ 1	11100	0	001	011011	0000000000001100 ]
018 < 24>	[ 1	11100	0	001	011011	0000000000000110 ]
019 < 25>	[ 1	11100	0	001	011011	0000000000000100 ]
01A < 26>	[ 1	11001	0	001	000000	0000000000000001 ]
01B < 27>	[ 1	11001	1	001	011011	0000000000000000 ]
01C < 28>	[ 1	00010	0	001	111111	0000000000000000 ]

The Am29CPL151 operates at a maximum frequency of 30 MHz, consumes 115 mA, and its operating voltage range is 4.5 to 5.5 V. All the instructions are executed in one clock cycle. You can modify the program to suit your specific needs, but remember that a maximum of 64 instructions are allowed using the Am29CPL151.

Now we will use the field programmable controller Am29CPL152 to make the speech processor vocalize a BCD code input. Figure 2.18 shows the respective circuit where the Am29CPL152 receives the BCD code input on the test input lines (T0 to T3). The test input line t4 is now used here to read the STANDBY status (SBY) of the speech processor. Output lines P0 to P7 are configured as the circuit explained before.



**TABLE 2.21**  
 Program for Controlling the Speech Processor and  
 Make It Speak the Respective BCD Code Input

```
DEVICE (CPL142)
```

```
DEFAULT = 1;
```

```
DEFINE "test inputs"
```

```
b1 = t0 "when the n.o. switch is closed t0 goes high"
```

```
b2 = t1 "STANDBY goes to zero when A1-A8 = 00#h"
```

```
b3 = t2
```

```
b4 = t3
```

```
tst = cc
```

```
equal = eq
```

```
sby = t4
```

```
"Output control bits. Speech data = 59 allophones plus five pauses"
```

```
pa1 = 00#h pa2 = 01#h pa3 = 02#h pa4 = 03#h pa5 = 04#h
```

```
oy = 05#h ay = 06#h eh = 07#h kk3 = 08#h pp = 09#h
```

```
jh = 0A#h nn1 = 0B#h ih = 0C#h tt2 = 0D#h rr1 = 0E#h
```

```
ax = 0F#h mm = 10#h tt1 = 11#h dh1 = 12#h iy = 13#h
```

```
ey = 14#h dd1 = 15#h uw1 = 16#h ao = 17#h aa = 18#h
```

```
yy2 = 19#h ae = 1A#h hh1 = 1B#h bb1 = 1C#h th = 1D#h
```

```
uh = 1E#h uw2 = 1F#h aw = 20#h dd2 = 21#h gg3 = 22#h
```

```
vv = 23#h gg1 = 24#h sh = 25#h zh = 26#h rr2 = 27#h
```

```
ff = 28#h kk2 = 29#h kk1 = 2A#h zz = 2B#h ng = 2C#h
```

```
ll = 2D#h ww = 2E#h xr = 2F#h wh = 30#h yy1 = 31#h
```

```
ch = 32#h er1 = 33#h er2 = 34#h ow = 35#h dh2 = 36#h
```

```
ss = 37#h nn2 = 38#h hh2 = 39#h or = 3A#h ar = 3B#h
```

```
yr = 3C#h gg2 = 3D#h el = 3E#h bb2 = 3F#h;
```

```
BEGIN
```

```
"wait for test input to go high. SE = 0 (pin 19 of SPO256-AL2)"
```

```
"1"stay:pal, if (not tst) then goto pl(stay);
```

```
"2" pal, cmp tm(0F#h) to pl(00#h);
```

```
"3" pal, if (equal) then goto pl(zero);
```

```
"4" pal, cmp tm(0F#h) to pl(01#h);
```

```
"5" pal, if (equal) then goto pl(one);
```

```
"6" pal, cmp tm(0F#h) to pl(02#h);
```

```
"7" pal, if (equal) then goto pl(two);
```

```
"8" pal, cmp tm(0F#h) to pl(03#h);
```

```
"9" pal, if (equal) then goto pl(thre);
```

```
"10" pal, cmp tm(0F#h) to pl(04#h);
```

```
"11" pal, if (equal) then goto pl(four);
```

```
"12" pal, cmp tm(0F#h) to pl(05#h);
```

```
"13" pal, if (equal) then goto pl(five);
```

```
"14" pal, cmp tm(0F#h) to pl(06#h);
```

```
"15" pal, if (equal) then goto pl(six);
```

```
"16" pal, cmp tm(0F#h) to pl(07#h);
```

```
"17" pal, if (equal) then goto pl(svn);
```

```

"18"    pal,    cmp tm(0F#h) to pl(08#h);
"19"    pal,    if (equal) then goto pl(eit);
"20"    pal,    cmp tm(0F#h) to pl(09#h);
"21"    pal,    if (equal) then goto pl(nin);
"22"    pal,    cmp tm(0F#h) to pl(0A#h);
"23"    pal,    if (equal) then goto pl(error);

"routine for the word zero"
"24"zero:zz,    if (sby) then call pl(read);
"25"    yr,     if (sby) then call pl(read);
"26"    ow,     if (sby) then call pl(read);
"27"    pa4,    if (sby) then call pl(read);
"28"    pal,    if (sby) then goto pl(stay);

"routine for the word one"
"29"one:ww,     if (sby) then call pl(read);
"30"    ax,     if (sby) then call pl(read);
"31"    ax,     if (sby) then call pl(read);
"32"    nn1,    if (sby) then call pl(read);
"33"    pa4,    if (sby) then call pl(read);
"34"    pal,    if (sby) then goto pl(stay);

"routine for the word two"
"35"two:tt2,    if (sby) then call pl(read);
"36"    uw2,    if (sby) then call pl(read);
"37"    pa4,    if (sby) then call pl(read);
"38"    pal,    if (sby) then goto pl(stay);

"routine for the word three"
"39"three:th,   if (sby) then call pl(read);
"40"    rr1,    if (sby) then call pl(read);
"41"    iy,     if (sby) then call pl(read);
"42"    pa4,    if (sby) then call pl(read);
"43"    pal,    if (sby) then goto pl(stay);

"routine for the word four"
"44"four:ff,    if (sby) then call pl(read);
"45"    ff,     if (sby) then call pl(read);
"46"    or,     if (sby) then call pl(read);
"47"    pa4,    if (sby) then call pl(read);
"48"    pal,    if (sby) then goto pl(stay);

"routine for the word five"
"49"five:ff,    if (sby) then call pl(read);
"50"    ff,     if (sby) then call pl(read);
"51"    ay,     if (sby) then call pl(read);
"52"    vv,     if (sby) then call pl(read);
"53"    pa4,    if (sby) then call pl(read);
"54"    pal,    if (sby) then goto pl(stay);

"routine for the word six"
"55"six:ss,     if (sby) then call pl(read);
"56"    ss,     if (sby) then call pl(read);
"57"    ih,     if (sby) then call pl(read);
"58"    ih,     if (sby) then call pl(read);
"59"    pa3,    if (sby) then call pl(read);
"60"    kk2,    if (sby) then call pl(read);

```

```

"61"    ss,    if (sby) then call pl(read);
"62"    pa4,   if (sby) then call pl(read);
"63"    pa1,   if (sby) then goto pl(stay);

"routine for the word seven"
"64"svn:ss,   if (sby) then call pl(read);
"65"    ss,    if (sby) then call pl(read);
"66"    eh,    if (sby) then call pl(read);
"67"    eh,    if (sby) then call pl(read);
"68"    vv,    if (sby) then call pl(read);
"69"    eh,    if (sby) then call pl(read);
"70"    nn1,   if (sby) then call pl(read);
"71"    pa4,   if (sby) then call pl(read);
"72"    pa1,   if (sby) then goto pl(stay);

"routine for the word eight"
"73"eit:ey,   if (sby) then call pl(read);
"74"    pa3,   if (sby) then call pl(read);
"75"    tt2,   if (sby) then call pl(read);
"76"    pa4,   if (sby) then call pl(read);
"77"    pa1,   if (sby) then goto pl(stay);

"routine for the word nine"
"78"nin:nn2,  if (sby) then call pl(read);
"79"    aa,    if (sby) then call pl(read);
"80"    ay,    if (sby) then call pl(read);
"81"    nn1,   if (sby) then call pl(read);
"82"    pa4,   if (sby) then call pl(read);
"83"    pa1,   if (sby) then goto pl(stay);

"routine error"
"84"error:eh, if (sby) then call pl(read);
"85"    xr,    if (sby) then call pl(read);
"86"    or,    if (sby) then call pl(read);
"87"    pa4,   if (sby) then call pl(read);
"88"    pa1,   if (sby) then goto pl(stay);

"subroutine for reading the standby status of the speech processor"
"89"read:pa1,  if (not sby) then goto pl(read); "reading SBY"
"90"    pa1,   if (sby) then ret;
           .org 127#d
"91"pa1, if (sby) then goto pl(stay);
END.

```

that contains the next allophone. This process continues until all the allophones have been issued. Note that the last instruction of each routine sends the program back for reading the test (tst) switch. If the number ten (1010) is received at the four inputs (T0 to T3), the program goes to a routine that makes the speech processor announce the message "error." The user can add more instructions if he or she desires to detect the numbers eleven to fifteen. Try doing this and assemble your circuit and verify its operation by building this compact unit. Table 2.22 presents the prom bit pattern for the

Am29CPL152. The JEDEC file is not shown here, but you will get it when you assemble the file speech 4.doc. The JEDEC file is required for the EPROM burner and is directly downloaded from your computer. While the FPCs labeled with the letters PL have a built-in PROM, remember that the field programmable controllers containing the letters CPL have a built-in CMOS EPROM.

If you want to design a 5-bit binary input vocalizer, try to design your program in a form such that the FPC combines the words. For example, the number “twenty one” requires the use of two words: “twenty” and “one.” The

**TABLE 2.22**  
Prom Bit Pattern for the Am29CPL152

PROM Contents:						
hex <dec>	OE	OPCODE	POL	TEST	DATA	OUTPUT
000 < 0>	[ 1 ]	11001	1	0111	00000000	0000000000000000
001 < 1>	[ 1 ]	00110	0	0111	0001111	0000000000000000
		OPCODE		CONSTANT	DATA	
002 < 2>	[ 1 ]	100		0000000	0001111	0000000000000000
003 < 3>	[ 1 ]	11001	0	1000	0011000	0000000000000000
		OPCODE		CONSTANT	DATA	
004 < 4>	[ 1 ]	100		0000001	0001111	0000000000000000
005 < 5>	[ 1 ]	11001	0	1000	0011101	0000000000000000
		OPCODE		CONSTANT	DATA	
006 < 6>	[ 1 ]	100		0000010	0001111	0000000000000000
007 < 7>	[ 1 ]	11001	0	1000	0100011	0000000000000000
		OPCODE		CONSTANT	DATA	
008 < 8>	[ 1 ]	100		0000011	0001111	0000000000000000
009 < 9>	[ 1 ]	11001	0	1000	0100111	0000000000000000
		OPCODE		CONSTANT	DATA	
00A < 10>	[ 1 ]	100		0000100	0001111	0000000000000000
00B < 11>	[ 1 ]	11001	0	1000	0101100	0000000000000000
		OPCODE		CONSTANT	DATA	
00C < 12>	[ 1 ]	100		0000101	0001111	0000000000000000
00D < 13>	[ 1 ]	11001	0	1000	0110001	0000000000000000
		OPCODE		CONSTANT	DATA	
00E < 14>	[ 1 ]	100		0000110	0001111	0000000000000000
00F < 15>	[ 1 ]	11001	0	1000	0110111	0000000000000000
		OPCODE		CONSTANT	DATA	
010 < 16>	[ 1 ]	100		0000111	0001111	0000000000000000
011 < 17>	[ 1 ]	11001	0	1000	1000000	0000000000000000
		OPCODE		CONSTANT	DATA	
012 < 18>	[ 1 ]	100		0001000	0001111	0000000000000000
013 < 19>	[ 1 ]	11001	0	1000	1001001	0000000000000000
		OPCODE		CONSTANT	DATA	
014 < 20>	[ 1 ]	100		0001001	0001111	0000000000000000
015 < 21>	[ 1 ]	11001	0	1000	1001110	0000000000000000

	OPCODE	CONSTANT	DATA	
016 < 22>	[ 1 :	100 :	0001010 :	0001111 : 0000000000000000
017 < 23>	[ 1 :	11001 :	0 : 1000 :	1010100 : 0000000000000000
018 < 24>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000101011
019 < 25>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000111100
01A < 26>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000110101
01B < 27>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
01C < 28>	[ 1 :	11001 :	0 : 0100 :	0000000 : 0000000000000000
01D < 29>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000101110
01E < 30>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001111
01F < 31>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001111
020 < 32>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001011
021 < 33>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
022 < 34>	[ 1 :	11001 :	0 : 0100 :	0000000 : 0000000000000000
023 < 35>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001101
024 < 36>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001111
025 < 37>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
026 < 38>	[ 1 :	11001 :	0 : 0100 :	0000000 : 0000000000000000
027 < 39>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001101
028 < 40>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001110
029 < 41>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000010011
02A < 42>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
02B < 43>	[ 1 :	11001 :	0 : 0100 :	0000000 : 0000000000000000
02C < 44>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000101000
02D < 45>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000101000
02E < 46>	[ 1 :	11100 :	0 : 0100 :	1011001 : 00000000000111010
02F < 47>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
030 < 48>	[ 1 :	11001 :	0 : 0100 :	0000000 : 0000000000000000
031 < 49>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000101000
032 < 50>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000101000
033 < 51>	[ 1 :	11100 :	0 : 0100 :	1011001 : 000000000000110
034 < 52>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000100011
035 < 53>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
036 < 54>	[ 1 :	11001 :	0 : 0100 :	0000000 : 0000000000000000
037 < 55>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000011011
038 < 56>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000011011
039 < 57>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001100
03A < 58>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001100
03B < 59>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000010
03C < 60>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000101001
03D < 61>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000011011
03E < 62>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
03F < 63>	[ 1 :	11001 :	0 : 0100 :	0000000 : 0000000000000000
040 < 64>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000011011
041 < 65>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000011011
042 < 66>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
043 < 67>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
044 < 68>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000100011
045 < 69>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011
046 < 70>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000001011
047 < 71>	[ 1 :	11100 :	0 : 0100 :	1011001 : 0000000000000011

048 < 72>	[ 1	11001	0	0100	0000000	0000000000000000 ]
049 < 73>	[ 1	11100	0	0100	1011001	0000000000010100 ]
04A < 74>	[ 1	11100	0	0100	1011001	0000000000000010 ]
04B < 75>	[ 1	11100	0	0100	1011001	00000000000001101 ]
04C < 76>	[ 1	11100	0	0100	1011001	00000000000000011 ]
04D < 77>	[ 1	11001	0	0100	0000000	0000000000000000 ]
04E < 78>	[ 1	11100	0	0100	1011001	0000000000111000 ]
04F < 79>	[ 1	11100	0	0100	1011001	0000000000011000 ]
050 < 80>	[ 1	11100	0	0100	1011001	0000000000000110 ]
051 < 81>	[ 1	11100	0	0100	1011001	00000000000001011 ]
052 < 82>	[ 1	11100	0	0100	1011001	00000000000000011 ]
053 < 83>	[ 1	11001	0	0100	0000000	0000000000000000 ]
054 < 84>	[ 1	11100	0	0100	1011001	0000000000000111 ]
055 < 85>	[ 1	11100	0	0100	1011001	00000000000101111 ]
056 < 86>	[ 1	11100	0	0100	1011001	00000000000111010 ]
057 < 87>	[ 1	11100	0	0100	1011001	00000000000000011 ]
058 < 88>	[ 1	11001	0	0100	0000000	0000000000000000 ]
059 < 89>	[ 1	11001	1	0100	1011001	0000000000000000 ]
05A < 90>	[ 1	00010	0	0100	1111111	0000000000000000 ]

use of subroutines containing the basic numbers will help to develop a program that compares the magnitude of the digital input in order to determine the way to make the speech processor announce the correct number. In this manner, you will save memory space for other possible tasks of your program.

## 2.11 Multiplexing a Speech Processor with Different Data Sources

To be able to interface a speech processor with several data sources to read and vocalize the information, it is necessary to consider the use of multiplexers (MUX). Multiplexing will permit us to use only one speech processor when we want to measure different types of data sources.

A MUX can be implemented with MSI chips or with programmable logic (PALs or PLDs). It is up to you to decide the technology that better suits your specific needs.

For the interpretation of different data sources using a single speech processor, the circuit shown in Figure 2.19 provides a solution. The 8-bit data input sources may represent any physical or electrical measurement that is in binary code ready to be latched and selected by the 8-bit transparent latches IC1a, IC1b, . . . IC1n (74HC373). It is not necessary to have 8-bit data inputs in all the latches; 7-bit data inputs or less can be used in this specific circuit. If this is the case, data lines must be shifted to the next lower address pin of the EPROM, reducing the size of the EPROM from a 27256, for example, to a 27128, 2764, and so on.

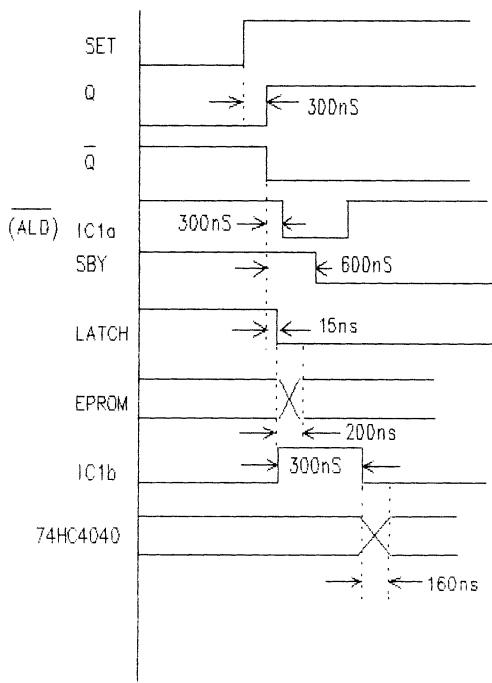




of memory starting at location  $2^{14}$  is selected. Once the flip-flop is high, the SPO256 is enabled to start the timing of the allophone's sequence. IC4 (74HC4040) configured as a 5-bit binary counter is responsible for scanning in sequence a maximum of 32 speech entry points per reading. This counter is triggered at the rising edge of the /ALD input signal. In this form, while the speech processor is speaking the first speech data, counter IC4 is clocked; that is, it is incremented to give time for the EPROM to access the new speech data to the input port (A1 to A6) of the speech processor.

It is very important to avoid glitches in the input port lines (A1 to A6) of the speech processor when you want to pulse the /ALD input low. That consideration is taken into account here because the SBY output gives a logic low at its output, indicating that the speech processor is talking. This causes a logic high at the output of Nand gate IC6a, which in turn clocks counter IC4 via Nand gate IC6b. Counter IC4 specifies the next address for the EPROM memory. The propagation time delay caused by Nand gates IC1a, IC1b, counter IC4, and the EPROM memory (IC5) is about 1300 ns (see Figure 2.20). After that short time delay, the circuit will call out readings.

In the same way, the type of circuit that the user selects to control the inputs (Y0, Y1, and SET) plus the 8-bit data inputs must wait a short time to



**Figure 2.20** Timing diagram for the circuit shown in Figure 2.19.

avoid reading glitches from the 8-bit data inputs. In this case, a “Ready” signal coming from the data source would be very important.

Finally, the new designing tools now available for programmable array logic (PALs) and generic array logic (GALs) will permit you to achieve the kind of multiplexer you are looking for. If you include Nand gates and the flip-flop used in the circuit previously explained, you will save more hardware and space when building this unit.

## References

- CMOS Logic Databook. 400039 Rev. 1. National Semiconductor. 1989.  
SPO256B Narrator Speech Processor, (DS50018A-1)  
SPO256-AL2 Narrator Speech Processor, (DS50005A-1) Microchip Technology Inc. 1989.  
MOS Memory Products. Toshiba Semiconductor, Inc. 1987.  
Am29CPL100 Family of Field Programmable Controllers. Handbook. Advanced Micro Devices. 1988.  
Am29CPL141 FPC Data sheet Advanced Micro Devices. 1988.  
Am29CPL142 FPC Data sheet Advanced Micro Devices. 1989.  
Am29CPL144 FPC Data sheet Advanced Micro Devices. 1989.  
8049/8051 Microcontrollers User's Guide. Signetics. 1989.  
GW Basic User's Manual for TANDY 1000SL. Tandy Corporation. 1989.  
Ricardo Jimenez-G., Circuit Vocalizes Hex Code for a 4-bit Input. EDN, March 18, 1987, pp. 204-206.

## *Analog Circuits*

### **3.1 Basics of the A/D Converters**

Now that digital computers and digital control systems are finding widespread uses, measuring the natural and artificial variables found in the real world is becoming less of an analog function. The variables of temperature, pressure, speed, and so on are natural and artificial variables that occur in quantities that change continuously. In order to change most variables from an analog domain to a digital domain, an interface device is usually required—the analog-to-digital (A/D) converter.

Essentially, an A/D converter is an encoding device that accepts an analog signal ( $V_i$ ) and an analog reference ( $V_r$ ) as inputs and generates a digital output ( $D_o$ ) which is an effect related to the input.

The design and selection of the best converter for a particular system is by no means simple. There is usually no single converter that will meet all specifications; however, to obtain optimum performance at minimum size and cost, the designer must consider factors that are common to all converters: performance, speed, reference supplies, power supplies, buffers, and other factors not considered in this book. Within the framework of these factors, the A/D converters used in this section were selected as the optimum devices for our speech synthesis applications based on availability of product and price, and also because they are the most widely used products by designers.

#### *Analog-to-Digital Converters*

National Semiconductor's low-cost, CMOS 8-bit successive approximation A/D converters, designated as ADC0800, are compatible with most microprocessors/microcontrollers, and in most applications they do not require

added interfacing logic. These devices operate from a single 5-V supply, and the conversion time is 100  $\mu$ s. Five accuracies are available in this family of A/D converters:

- ADC0801 +1/4 LSB full scale adjusted
- ADC0802 +1/2 LSB unadjusted ( $V_{ref}/2 = 2.500$ )
- ADC0803 +1/2 LSB adjusted
- ADC0804 +1 LSB unadjusted
- ADC0805 +1 LSB

These converters contain a new differential analog voltage input, which allows the common-mode rejection to be increased and the analog zero input voltage value to be offset. In addition, National Semiconductor has 10-bit and 12-bit versions of these A/D converters (ADC) to provide a complete family of ADC0800 devices. Figure 3.1 shows the pin configuration for the ADC080X.

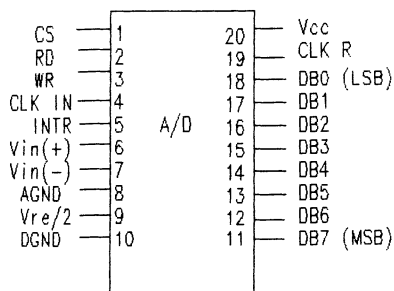
### Functional Description

It is our goal in this section to show you how to use the A/D converters (series ADC080X). Figure 3.2 shows the logic diagram of the A/D converters of the series ADC080X.

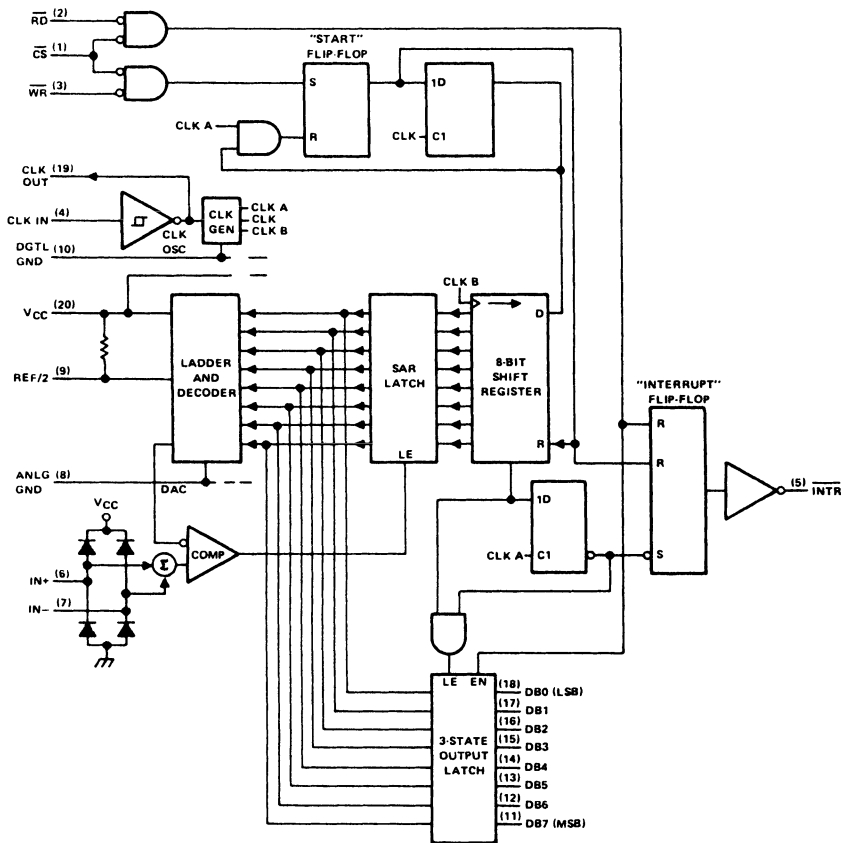
Before you start applying these converters to speech processors, it is recommended that you first test the A/D converter, because you will be able to test the first phase of your system. The simplest test consists of applying a known analog input voltage to the converter and using LEDs to display the resulting digital output code.

Figure 3.3 shows the basic configuration for this test.

The first phase is to test the operation of the circuit illustrated in Figure 3.3 by using a full-scale adjustment. An analog input voltage of 2.55 Vdc should be applied to the  $V_{in}(+)$  pin with the  $V_{in}(-)$  grounded. The value of the  $V_{ref}/2$  input voltage should then be adjusted until the digital output code is just changing from 1111 1110 to 1111 1111. In this case,  $V_{ref}/2$  is set to



**Figure 3.1** Pin configuration for the ADC080X.

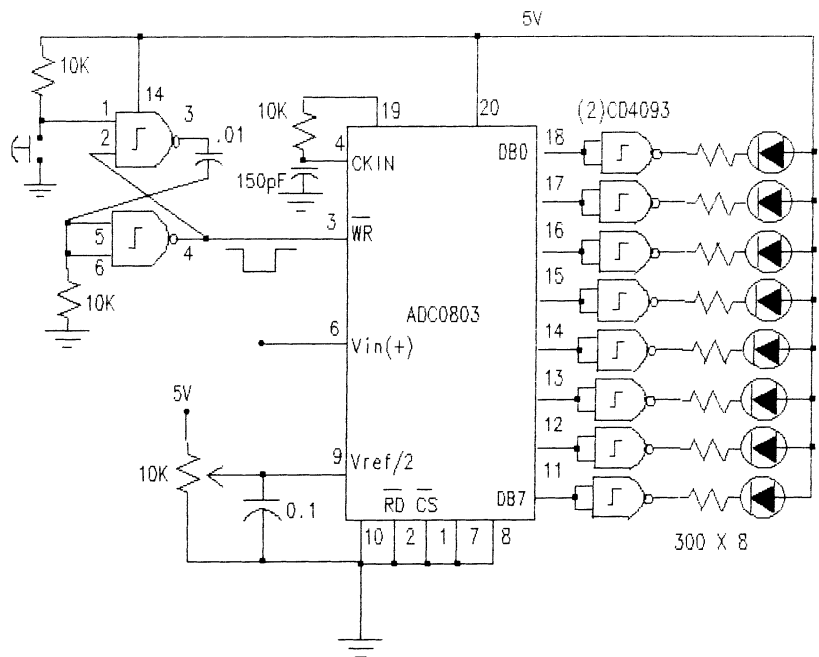


**Figure 3.2** Logic diagram of the ADC080X family of A/D converters. (Reprinted with permission from National Semiconductor Corp. © 1980, Linear Data Book, 1982, pp. 8–41.)

1.30 Vdc. The reference voltage  $V_{ref}/2$  of 1.30 Vdc should then be used for all the tests. This fixed value of  $V_{ref}/2$  provides an LSB value of 0.01 V. The operating input voltage is from 0 to 2.55 V, with a resolution of 0.01 V. Table 3.1 shows the binary equivalent of six different input voltage readings.

When the circuit of Figure 3.3 is first turned on, the monostable circuit formed by IC1a and IC1b pulses low the  $/WR$  input for an interval of 80  $\mu s$  to ensure start-up under all possible conditions during the first power-up cycle. Notice that the monostable lacks power-up reset circuitry which is normally applied to pin 6 of Nand gate IC1b; due to this fact, the monostable is self-triggered when the circuit is first turned on.

Now you can apply, for example, an input voltage of 1.00 Vdc and then press the test switch to pulse the  $/WR$  input low. This action causes the  $/INTR$  output to go high. After 100  $\mu s$  the correct output code will be present on outputs DB0 to DB7. These outputs are buffered with Nand gates to provide



**Figure 3.3** Circuitry for testing the ADC0803 converter.

the sink current to drive the eight LEDs. At this time you can check the binary code with Table 3.1. For interfacing purposes, the output reading in binary code is ready and correct when the interrupt /INTR output of the A/D converter goes to a logic low.

With respect to the A/D conversion process, the most significant bit is tested first and, after eight comparisons (64 cycles) a digital 8-bit binary code

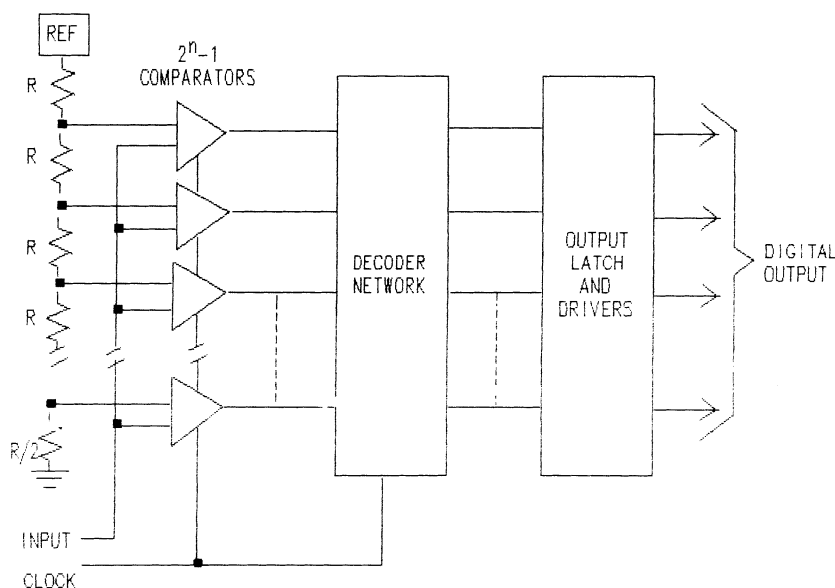
TABLE 3.1									
Input Voltages and Their Equivalent in Binary Code									
	Vin(+)	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1.	0.00 V	0	0	0	0	0	0	0	0
2.	0.05 V	0	0	0	0	0	1	0	1
3.	1.00 V	0	1	1	0	0	1	0	0
4.	2.00 V	1	1	0	0	1	0	0	0
5.	2.50 V	1	1	1	1	1	0	1	0
6.	2.55 V	1	1	1	1	1	1	1	1

is transferred to an output latch; then an interrupt is asserted (/INTR makes a high-to-low transition). The pin functions assigned as write (/WR) and interrupt (/INTR) will be the most widely used for operation with a microcomputer-based system.

In applications where the sensor and the A/D converter are using the same reference, the drift of the reference will not matter, because it will cancel out.

If speed is important, consider the use of ADCs that employ flash conversion techniques. The high speed of this technique lies in the fact that all comparators examine the input voltage simultaneously in parallel mode and make an immediate total conversion. A flash converter has a separate comparator for each of the possible digital output values. All the comparators receive the input voltage. Comparators in which the input is above the reference will generate zeros. The output of all the comparators is fed to an encoder which converts the data to a binary value. This approach is limited by the number of comparators required. The flash A/D converter does not need a sample-hold circuit or a D/A converter. An 8-bit resolution converter requires 255 comparators ( $2^8 - 1$ ). See Figure 3.4 for a block diagram of a typical flash converter.

If you are planning to use a flash converter for a control system containing speech synthesis, bear in mind that the speech processor is a relatively low-speed device compared to the flash converter; consequently, your control



**Figure 3.4** Block diagram of a flash converter.



system must be able to respond rapidly to take control of the situation, and then the speech processor will proceed to give the messages required for a specific situation.

The following sections of this chapter will be using A/D converters and comparators, which will be interfaced to a controller that will take care of driving a speech processor.

### 3.2 Interfacing the ADC0804 to Digitaler DT1050

This application covers all the possible interface combinations for the  $\mu\text{C}$  described below. This interface offers the following advantages:

1. Fast complete cycle times for loading the A/D address, performing conversion, retrieving the conversion, and loading the conversion to a speech processor.
2. Flexible use of A/D converter pins for either A/D conversion or input of digital data.
3. Low cost.

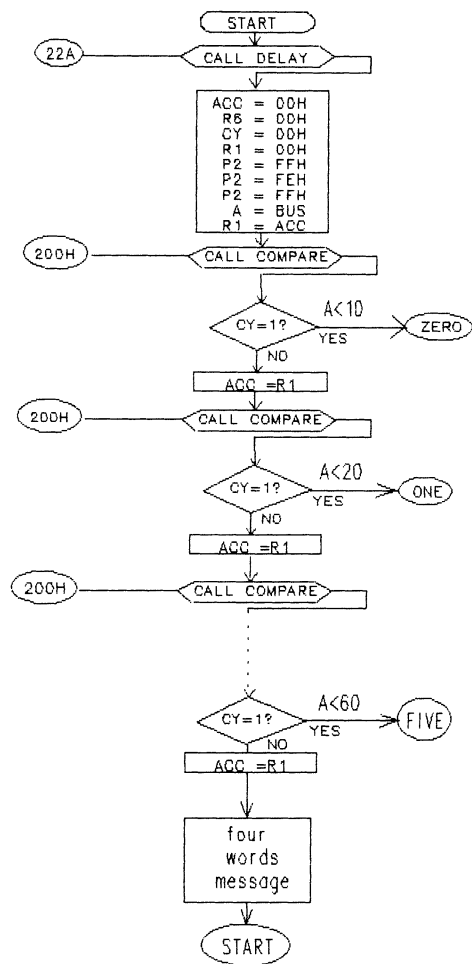
The Intel Corporation microprocessor families for the  $\mu\text{C}$  8048/8049 consist of the following:

8035AHL	8048AH	8748H
8039AHL	8049AH	8749H
8040AHL	8050AH	80C49

Figure 3.5 shows the circuit configuration for the interface. This circuit can be used for the 8048 and 8049 devices interfaced to the ADC0804 device, and the 8051 and 8052 devices interfaced to the ADC0804. The system clock for the A/D converter is obtained from the RC logic oscillator. The RC logic oscillator does not exceed the upper frequency limit of the A/D converter. You can also obtain the system clock for the A/D converter from the microcontroller crystal oscillator. In this case, a high impedance buffer must be used to avoid overloading the oscillator. The buffered crystal oscillator signal must be frequency divided to ensure that the resulting system clock frequency does not exceed the upper limit frequency of the A/D converter. Any convenient divider circuitry may be used to accomplish this task.

The flowchart required to develop the control routine is shown in Figure 3.6. A conversion cycle consists of reading the digital data register and loading the speech data to the Digitaler DT1050. The complete software program can be incorporated into a subroutine, so the designer can easily access the software with a simple subroutine call. Also, the conversion software assumes that the A/D converter address has been placed into the accumulator and then





**Figure 3.6** Flowchart for the software program.

converter. On the other hand, another eight address lines (P1.0–P1.7) and two control signals (T1 and /WR) are utilized to drive the Digitalker DT1050. Two speech ROMs (SSR1 and SSR2) contain in compressed form the data required for the 144 addressable words.

The circuit announces the analog voltage corresponding to the digital input from the microcontroller. With a resolution of 0.1 V, the operating voltage range for this circuit is adjusted for  $0 < V_{in} < 5.9 \text{ Vdc}$ . The heart of the circuit is the software program that is designed to keep the circuit reading and announcing voltage readings every 30 seconds. Figure 3.6 is now our point of reference. As you can see, the flowchart starts calling the subroutine DELAY

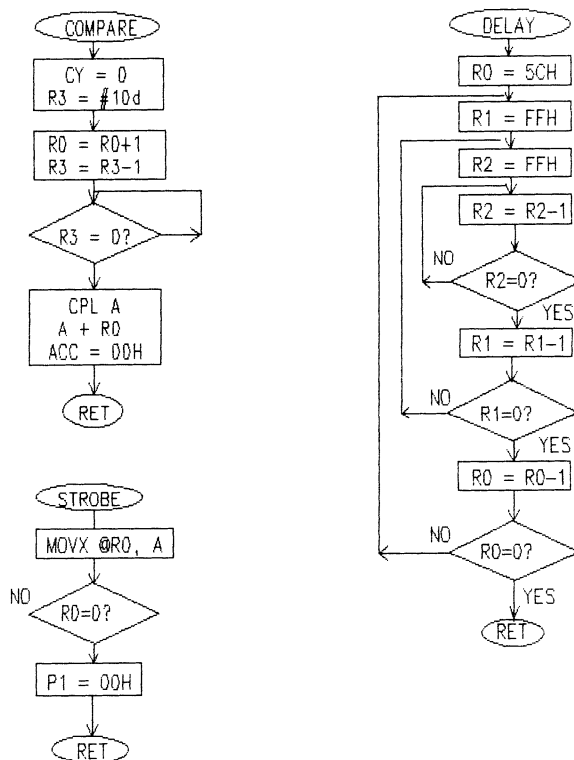
which is illustrated in Figure 3.7. The subroutine DELAY loads the decimal numbers 92, 255, and 255 in registers R0, R1, and R2, respectively. Notice that all subroutines are located in page two of ROM, leaving page zero and page one with some memory space available for adapting or augmenting the program to your needs. The memory space gives you the convenience of augmenting the program without having to modify the current addresses.

The instruction “DJNZ R2, T2” is executed in  $5\ \mu\text{s}$  because it is a two-byte instruction. The delay caused by this instruction is

$$5\ \mu\text{s} \times 255 = 1275\ \mu\text{s}$$

When the register R2 is zero, the program jumps to the next instruction “DJNZ R1, T3” which decrements register R1 by one. The program will keep jumping to address T3 of the current page. This loop takes  $10\ \mu\text{s}$  times 255 plus the time required to decrement register R2 ( $1275 \times 255$ ); therefore, we will have the following elapsed time:

$$(10\ \mu\text{s} \times 255) + (1275\ \mu\text{s} \times 255) = 0.327675\ \text{s}.$$



**Figure 3.7** Flowcharts for the subroutines.

The final step to increase this delay is to make a loop by decrementing register R0 92 times, that is

$$0.3276 \text{ s} \times 92 = 30.14 \text{ s}$$

After this delay the program clears the accumulator, registers R1 and R6, the carry flag (CY), and port one. Output line P2.0 controls the write input (/WR) of the ADC0803 by sending a low transient pulse of 5  $\mu\text{s}$  (see address lines 0EH to 1EH). This negative transient pulse initiates the conversion process that takes 100  $\mu\text{s}$ . When the conversion process ends, the interrupt output (/INTR) goes to a logic zero. The negative edge of this pulse is detected by the input T0 of the microcontroller 8748. When this happens, the microcontroller proceeds to load the digital reading into the accumulator. This digital reading is then transferred to register R1 for future use. Now the program starts comparing the digital reading in order to know in what range the reading is located. This is achieved with the subroutine COMPARE located in address 00H of page two. The COMPARE subroutine determines whether the digital reading loaded in the accumulator is less than the decimal number 10. If that is the case, the carry flag (CY) is set to one. If CY is equal to one, the program jumps to subroutine ZERO. This subroutine loads the word "zero" into the address lines (SW1-SW8) of the Digitalker. The subroutine ZERO now calls the subroutine WRITE, which makes the Digitalker announce the corresponding correct words such as "zero . . . point . . . two" or "zero . . . point . . . nine." The words "zero . . . point" are already contained in the subroutine WRITE. The last word is a number that is obtained from page three of the EPROM memory.

When a digital reading is greater than 10 and less than 20, the program first performs the comparison to issue the words "one . . . point." The program then proceeds immediately to subtract 10 numbers of the digital reading before searching the word in page three of ROM. For example, if the digital reading is 00001111#b, which is the equivalent of 1.5 V, the program first compares the magnitude of the reading by finding that such a reading is less than the constant 20. At this point, the Digitalker will say the words "one . . . point." To find the following correct word "five," the binary reading 00001111#b is decremented 10 times. As a result, the digital reading is now converted to the number 00000101#b. It is this new number that is used by the accumulator to point to address five of page three. By looking at page three of the software program (Table 3.2), you will see that the number 05H is stored at address five; the program takes this new speech datum which is transferred to port 1 in order to have the Digitalker announce the last word "five."

The advantage of this technique is that the words "zero," "one" up to "nine" are stored only one time in ROM in order to save memory space.

The STROBE routine has two functions: to issue a negative transient pulse to the /WR input of the Digitalker and then to proceed to read the interrupt status (/INTR) in order to know when the Digitalker stops saying a word.

**TABLE 3.2**  
Software for  $\mu$ C 8748 to Control the Interface between the ADC0803  
and the Digitalker System

Add	Op	Code	Mnemonics	Comments
00	8A	FF	START: ORL P2, #FFH	; /CS1=1, /WR1=1
02	54	2A	CALL DELAY	;
04	9A	FF	ANL P2, #FFH	; /CS1=0, /WR=
06	99	00	ANL P1, #00H	;
08	27		CLR A	; Acc =00H
09	97		CLR C	; Clear carry flag
0A	A9		MOV R1, A	; Clear registers R0-R7
0B	AA		MOV R2, A	;
0C	AB		MOV R3, A	;
0D	AC		MOV R4, A	;
0E	AD		MOV R5, A	;
0F	AE		MOV R6, A	;
10	AF		MOV R7, A	;
11	9A	FE	ANL P2, #FEH	; A/D starts conversion
13	8A	FF	ORL P2, #FFH	; Set /WR = 1
15	36	15	WAIT: JTO WAIT	; Wait for /INTR to go low
17	00		NOP	; Delay to avoid reading glitches
18	08		INS A, BUS	; Load BUS contents to accumulator
19	A9		MOV R1, A	; Store reading in register R1
1A	54	00	CALL COMPARE	; Call compare to see if A<10
1C	F6	49	JC ZERO	; Go to subroutine ZERO if A<10
1E	F9		MOV A, R1	; Load voltage reading to Acc
1F	54	00	CALL COMPARE	; Call compare to see if A<20
21	F6	4F	JC ONE	; Go to subroutine ONE if A<20
23	F9		MOV A, R1	; Load voltage reading to Acc
24	54	00	CALL COMPARE	; Call compare to see if A<30
26	F6	57	JC TWO	; Go to subroutine TWO if A<30
28	F9		MOV A, R1	; Load voltage reading to Acc
29	54	00	CALL COMPARE	; Call COMPARE to see if A<40
2B	F6	5F	JC THREE	; Go to subroutine THREE if A<40
2D	F9		MOV A, R1	; Load voltage reading to Acc
2E	54	00	CALL COMPARE	; Call COMPARE to see if A<50
30	F6	67	JC FOUR	; Go to subroutine FOUR if A<50
32	F9		MOV A, R1	; Load voltage reading to Acc
33	54	00	CALL COMPARE	; Call COMPARE if A<60
35	F6	6F	JC FIVE	; Go to subroutine FIVE if A<60
37	00		NOP	;
38	00		NOP	;
				; Routine for message "danger"
39	BC	04	MOV R4, #04H	; R4 is loaded with the number of ; words of the message
3B	BD	64	MOV R5, #64H	; R5=#100d to find data in page 3
3D	27		CLR A	;
3E	39		RICK: OUTL P1, A	;
3F	80		MOVX @R0, A	;

```

40 1D          INC R5          ;
41 FD          MOV A, R5       ;
42 E3          MOV P3 A, @A    ;
43 46 43       JNT1, SBY       ;Reading /INTR status of DT1050
45 EC 39       DJNZ R4, RICK    ;
47 04 00       JMP 00H         ;
;-----
;Subroutine ZERO
49 89 1F       ORL P1, #1FH     ;"Zero"
4B 54 20       CALL WRITE      ;
4D 04 00       JMP 00H         ;
;-----
;Subroutine ONE
4F 89 01       ONE: ORL P1, #01H ;"One"
51 FE 0A       MOV R6, #0AH     ;R6 = #10d
53 54 0B       CALL SEARCH     ;Call address IF in page two
55 04 00       JMP 00H         ;
;-----
;Subroutine TWO
57 89 02       TWO: ORL P1, #02H ;"Two"
59 FE 14       MOV R6, #14H     ;R6 = #20d
5B 54 0B       CALL SEARCH     ;
5D 04 00       JMP 00H         ;
;-----
;Subroutine THREE
5F 89 03       THREE: ORL P1, #03H ;"Three"
61 FE 1E       MOV R6, #1EH     ;R6 = #30d
63 54 0B       CALL SEARCH     ;
65 04 00       JMP 00H         ;
;-----
;Subroutine FOUR
67 89 04       FOUR: ORL P1, #04H ;"Four"
69 FE 28       MOV R6, #28H     ;R6 = #40d
6B 54 0B       CALL SEARCH     ;
6D 04 00       JMP 00H         ;
;-----
;Subroutine FIVE
6F 89 05       FIVE: ORL P1, #05H ;"Five"
71 FE 32       MOV R6, #32H     ;R6 = #50d
73 54 0B       CALL SEARCH     ;
75 04 00       JMP 00H         ;
END           ;
;-----
;Subroutine COMPARE
200 97        COMPARE: CLR C    ;
201 BB 0A      MOV R3, #0AH     ;R3 = #10d
203 18        T1: INC R0        ;Increment R0 ten times to do A<R0
205 EB 03      DJNZ R3, T1      ;Decrement R3 ten times
207 37        CPL A            ;Compare A<R0 in order to determine
208 68        ADD A, R0        ;the magnitude of the digital
209 27        CLR A            ;reading. If carry flag is set
to 1
20A 83        RET              ;then A<R0
;-----

```

```

;Subroutine SEARCH
20B 54 1A SEARCH:CALL STROBE ;
20D 89 7A ORL P1, #122d ;"Point"
20F 54 1A CALL STROBE ;
211 F9 MOV A, R1 ;Load Acc with voltage reading
212 07 DEC:DEC A ;
213 EE 12 DJNZ R6, DEC ;Decrement Acc n times, where n=R6
215 E3 MOVP3 A, @A ;Transfer the byte in page three
216 39 OUTL P1, A ;Load speech data to Digitaltalker
217 54 1A CALL STROBE ;
219 83 RET ;
;-----
;Subroutine STROBE
21A 90 MOVX @R0, A ;/
WR of DT1050 is pulsed low for 5uS
21B 46 1B RIC:JNT1, RIC ;Reading /INTR status
21D 99 00 ANL P1, #00H ;Clear port 1 to load a new byte
21F 83 RET ;
;-----
;Subroutine WRITE
220 54 1A WRITE:CALL STROBE ;
222 89 7A ORL P1, #7AH ;"Point"
224 54 1A CALL STROBE ;
226 F9 MOV A, R1 ;Load Acc with voltage reading
227 E3 MOVP3 A, @A ;
228 39 OUTL P1, A ;
229 54 1A CALL STROBE ;
22B 83 RET ;
;-----
;Routine DELAY (30.14 seconds)
22C B8 5C DELAY:MOV R0, #5CH ;Do delay to allow "n" system
22E B9 FF T4:MOV R1, #FFH ;clocks to occur.
230 BA FF T3:MOV R2, #FFH ;
232 EA 32 T2:DJNZ R2, T2 ;
234 E9 30 DJNZ R1, T3 ;
236 E8 2E DJNZ R0, T4 ;
238 83 RET ;
;-----
;Speech data in page 3 for words:
300 IF ;"Zero"
301 01 ;"One"
302 02 ;"Two"
303 03 ;"Three"
304 04 ;"Four"
305 05 ;"Five"
306 06 ;"Six"
307 07 ;"Seven"
308 08 ;"Eight"
309 09 ;"Nine"
364 00 ;"This is Digitaltalker"
365 4C ;"Danger"
366 41 ;"Try"
367 8C ;"Again"

```



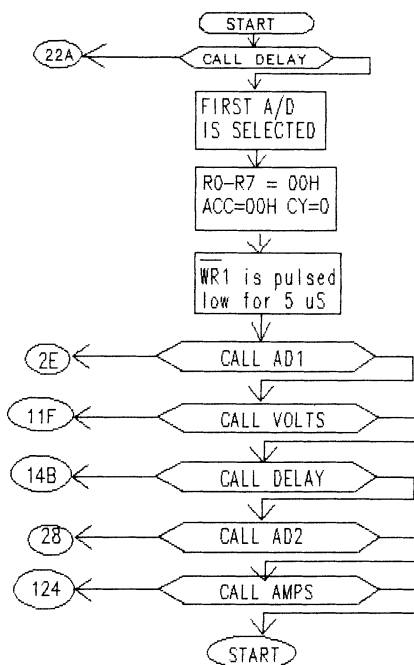


required for interfacing a maximum of eight A/D converters. For explanation and simplifying purposes, only two A/D converters are considered in the circuitry shown in Figure 3.8. A 3-to-8 line decoder (74HC138) is used to select one of the two A/D converters (ADC0804) by controlling their chip select inputs (/CS1 or /CS2) via the Y0 and Y1 outputs. Notice that output control lines Y2 to Y7 are left unconnected and can be used to control six more A/D converters. The three input lines (A, B, and C) of the 74HC138 are controlled by port two of the  $\mu$ C 8748 (P2.4, P2.5, and P2.6). On the other hand, an 8-channel analog multiplexer (74HC4051) configured as a selector is responsible for selecting the interrupt output (/INTR) of each A/D converter. The output of this selector (pin 3 of 74HC4051) is routed to the T0 input of the microcontroller. Three lines of port two (P2.1, P2.2, and P2.3) are used to select the interrupts (/INT1 or /INT2) of each A/D converter. Output line (P2.0) of the microcontroller is normally held at a logic high and is pulsed low for 5  $\mu$ s in order to start the conversion process of the selected A/D converter.

The method used here is similar to the one used in Section 3.2 of this chapter. The main difference of this method is that we will be using allophones instead of words; therefore, the amount of speech data stored in page three is larger because it contains the allophones to construct the words required for measuring different variables. For explanation purposes we will suppose that we are measuring two different variables: one within the range of 0 to 5.9 V and the other within the range of 0 to 2.0 mA. Both digital readings have a resolution of 0.1 V and 0.1 mA, respectively. Accordingly, both A/D converters are adjusted with a reference voltage of 1.30 V, which is applied to the input  $V_{ref}/2$ .

The flowchart illustrated in Figure 3.9 presents the steps that must be executed in order to make the speech processor announce the correct readings. The first instructions enable the A/D converter (ADC0804) located on the left corner of Figure 3.8. Lines 0AH to 12H are required to clear the accumulator and register R0 to R7. The instructions "ANL P2, #00H" and "ANL P2, #01H" located in lines 13H and 15H, respectively, cause a low transient pulse of 5  $\mu$ s, which starts the A/D conversion process of the selected A/D converter. The program now proceeds to call the subroutine AD1.

Figure 3.10 shows the circuitry for handling the multiple converters. Subroutine AD1 first reads the status of the /INT2 output of the A/D converter. When the interrupt signal (/INT) goes low, it indicates that the conversion process is over. Now the digital reading is loaded into the accumulator and stored in register R1 for future use. The next step is to compare the digital reading against the decimal constants 10, 20, 30, 40, 50, and 60. The task of comparing is made by the subroutine COMPARE. Subroutine COMPARE determines if the reading is less than 10. If so, the carry flag (CY) is set to one. When the program returns from the subroutine COMPARE, the instruction



**Figure 3.9** Flowchart subroutines for controlling two A/D converters interfaced to the speech processor SPO256-AL2.

“JNC, TEST 1” in line 34H detects that CY is set to one. Thus, the program does not jump to label “TEST 1” but continues with the next instruction. Here the program loads register R6 with 00H as well as the accumulator. Now the program calls the subroutine FIND which is located in address 00H of page one. Subroutine FIND clears register R5, loads register R5 with the value contained in the accumulator, and proceeds to interchange the value pointed by the accumulator in page three of ROM. The new value in the accumulator is a pointer that indicates the exact location of the speech data that must be issued to the speech processor. Notice that the first speech data correspond to the number of allophones that contain the message to be spoken. This number is stored in register R4. Now the accumulator is cleared, and register R5 is incremented in order to find the first speech data.

The first speech data are located by moving the contents of register R5 into the accumulator and then pointing to page three of ROM. The first speech data are now transferred to the accumulator and then to port one. At this point, the speech processor receives the binary address for finding the allophone. The program now calls subroutine STROBE, which pulses low the /ALD input of the speech processor SPO256-AL2; also, subroutine STROBE keeps reading the standby (SBY) status of the SPO256-AL2 in order to determine when the



When the program returns, it goes to address 3CH of page zero where the instruction “CALL POINT” is executed. The purpose of subroutine POINT is to make the speech processor say the word “point.”

It is important to notice that we are dealing with cases where readings have a magnitude of less than 10. That is, the cases correspond to readings that range from 0000 0000#b to 0000 1010#b. Because we are working with a resolution of 0.1 for readings of voltage or current, the speech processor has to announce the words “zero point..” prior to a digital reading. The final step to get the next word is made by the set of instructions that start at address 3EH of page one. Here the accumulator is loaded with the value of the digital reading that had been previously stored in register R1. The contents of the accumulator are then used to transfer the pointer from page three of ROM that will be used by the subroutine FIND. Finally, subroutine FIND will take care of finding the word that corresponds to the digital reading.

When the program returns from subroutine FIND, the program goes back to address 19H of page one. Here subroutine VOLTS is called in order to make the speech processor say the word “volts.” In this way, the user knows what variable has been measured and announced. Now the program calls subroutine DELAY that makes the system halt for 15 s. You can adjust this time delay according to your needs by changing the value of register R0 (see line 4B in page one). When this delay has occurred, the program now calls subroutine AD2. Subroutine AD2 disables the first A/D converter and enables the second via the 3-to-8 line decoder (74HC138). Once the second A/D converter is enabled, the write input of both converters is pulsed low, but only the second converter initiates the conversion process.

The procedure for having the speech processor enunciate the second reading is similar to the one explained above. The only difference is that, when the speech processor ends saying the second reading, the program makes the speech processor enunciate the word “milliamperes.” This makes the user aware of what type of reading he is listening to. Line 26H of page one is the end of the program. Here, instruction “JUMP START” makes the program start again by loading port two with ones in order to enable the first A/D converter again.

If you want to increase the range of the digital readings, increase the number of comparison routines that take place in the instructions labeled “TEST1” to “TEST5.” Also, bear in mind that the words or numbers that you are planning to use are already stored in page three of ROM.

Figure 3.10 : Circuitry for handling multiple converters using the uC 8748 along with the SPO256-AL2.

The  $\mu$ C software program is shown in Table 3.3.

**Table 3.3**

Software Program for Interface and Control of Multiple A/D Converters to the Speech Processor SPO256-AL2

Add	Op	Code	Mnemonics	Comments
00	8A	FF	START: ORL P2, #FFH	; /CS1=1, CS2=1, /WR1=1
02	34	46	CALL DELAY	;
04	9A	01	ANL P2, #01H	; /CS1=0, CS2=1, /WR=1
06	9A	00	ANL P1, #00H	;
08	00		NOP	;
09	00		NOP	;
0A	27		CLR A	; Acc =00H
0B	97		CLR C	; Clear carry flag
0C	A9		MOV R1, A	; Clear registers R0-R7
0D	AA		MOV R2, A	;
0E	AB		MOV R3, A	;
0F	AC		MOV R4, A	;
10	AD		MOV R5, A	;
11	AE		MOV R6, A	;
12	AF		MOV R7, A	;
13	9A	00	ANL P2, #00H	; /WR1=0, A/D initiates conversion
15	8A	01	ORL P2, #01H	; /WR1=1
17	34	2E	CALL AD1	;
19	34	1F	CALL VOLTS	;
20	34	46	CALL DELAY	;
22	14	28	CALL AD2	;
24	34	24	CALL AMPS	;
26	04	00	JMP START	;
28	8A	1F	AD2: ORL P2, #1FH	; /CS1=1, /CS2=0, /WR2=1
2A	9A	10	ANL P2, #10H	; /CS1=1, /CS2=0, /WR2=0
2C	8A	1F	ORL P2, #1FH	; /CS1=1, /CS2=0, /WR2=1
2E	36	2E	AD1: JTO AD1	; Wait for A/D conversion to occur
2F	00		NOP	; Delay to avoid reading glitches
30	08		INS A, BUS	; Load digital reading Vin1 into Acc.
31	A9		MOV R1, A	; Store digital reading in register R1
32	34	0F	CALL COMPARE	; Vin < #10d?
34	E6	43	JNC TEST 1	;
36	BE	00	MOV R6, #00H	;
38	23	00	MOV A, #00H	; Acc=00H to announce the word "ZERO"
3A	34	00	CALL FIND	;
3C	34	1A	CALL POINT	; Word "Point"
3E	F9		MOV A, R1	; Acc is loaded with reading less than 10.
3F	E3		MOVP3 A, @A	; Load the pointer byte
40	34	00	CALL FIND	;
42	83		RET	;
43	34	0F	TEST1: CALL COMPARE	; Vin < #20d?
45	E6	50	JNC TEST2	;
47	BE	08	MOV R6, #08H	; R6=#10d
49	23	0E	MOV A, #0EH	; Pointer for the word "ONE"
4B	34	00	CALL FIND	;
4D	34	30	CALL SEARCH	;

```

4F 83          RET          ;
50 34 0F TEST2:CALL COMPARE ;Vin <#30d?
52 E6 5D          JNC TEST3 ;
54 BE 14          MOV R6, #14H ;R6=#20d
56 23 13          MOV A, #13H ;Pointer for the word "TWO"
58 34 00          CALL FIND  ;
5A 34 37          CALL SEARCH ;
5C 83          RET          ;
5D 34 0F TEST3:CALL COMPARE ;Vin <#40d?
5F E6 6A          JNC TEST4 ;
61 BE 1E          MOV R6, #1EH ;R6=#30d
63 23 17          MOV A, #17H ;Pointer for the word "THREE"
65 34 00          CALL FIND  ;
67 34 37          CALL SEARCH ;
69 83          RET          ;
6A 34 0F TEST4:CALL COMPARE ;Vin <#50d?
6C E6 77          JNC TEST5 ;
6E BE 28          MOV R6, #28H ;R6=#40d
70 23 1B          MOV A, #1BH ;Pointer for the word "FOUR"
72 34 00          CALL FIND  ;
74 34 30          CALL SEARCH ;
76 83          RET          ;
77 34 0F TEST5:CALL COMPARE ;Vin <#60d?
79 E6 84          JNC ERROR  ;
7B BE 32          MOV R6, #32H ;R6=#50d
7D 23 1F          MOV A, #1FH ;Pointer for the word "FIVE"
7F 34 00          CALL FIND  ;/ALD=0
81 34 30          CALL SEARCH ;
83 83          RET          ;
                        ;Message "ERROR"
84      07 ERROR:MOV A, #3CH ;"EH"
86      34 00      CALL FIND  ;
88      04 00      JMP START  ;
                        ;Subroutine FIND
100 BD 00 FIND:MOV R5, #00H ;
102 AD          MOV R5, A ;Pointer for page three
103 E3          MOVP3 A, @A ;Acc = # of allophones
104 AC          MOV R4, A ;R4 = #n; for n allophones
105 27          CLR A ;
106 1D          PATY2:INC R5 ;Increment R5 to get next allophone
107 FD          MOV A, R5 ;
108 E3          MOVP3 A, @A ;
109 39          OUTL P1, A ;
10A 34 2A      CALL STROBE ;
10C EC 06      DJNZ R4, PATY2 ;
10E 83          RET          ;
                        ;Subroutine COMPARE; A < R0?
10F F9 COMPARE:MOV A, R1 ;Acc = Vin1
110 97          CLR C ;CY = 0
111 BB          MOV R3, #0A ;R3 = #10d
113 18          DEC:INC R0 ;
114 EB 13      DJNZ R3, DEC ;
116 37          CPL A ;
117 68          ADD A, R0 ;

```

```

118 27          CLR A          ;
119 83          RET            ;
                                ;
                                ;Message "POINT"
11A 23 3C POINT:MOV A, #3CH    ;
11C 34 00          CALL FIND    ;
11E 83          RET            ;
                                ;
                                ;Message "VOLTS"
11F 23 46 VOLTS:MOV A, #46H    ;
121 34 00          CALL FIND    ;
123 83          RET            ;
                                ;
                                ;Message "MILLIAMPERES"
124 23 4E AMPS:MOV A, #4EH     ;
126 34 00          CALL FIND    ;
128 04 00          JMP START    ;
                                ;
                                ;Subroutine STROBE
12A 80          STROBE:MOVX @R0, A ;/ALD is pulsed low for five uS
12B 46 2B WAIT:JNT1, WAIT      ;Wait for SBY to go high
12D 99 00          ANL P1, #00H ;
12F 83          RET            ;
                                ;
                                ;Subroutine SEARCH
130 BD 00 SEARCH:MOV R5, #00H  ;
132 34 1A          CALL POINT    ;
134 F9          MOV A, R1        ;Acc = voltage reading
135 07          DEC2:DEC A        ;Decrements n times Acc, where n
= R6
136 EE 35          DJNZ R6, DEC2  ;
138 E3          MOVP3 A, @A      ;
                                ;Load byte that points to the correct
                                ;word
139 AD          MOV R5, A        ;Store it in register R5
13A E3          MOVP3 A, @A      ;
A      ;Load first byte containing # of ;alloph.
13B AF          MOV R7, A        ;Store it in register R7
13C 1D          LUIGI:INC R5      ;
13D FD          MOV A, R5        ;
13F E3          MOVP3 A, @A      ;
140 39          OUTL P1, A        ;
141 34 2A          CALL STROBE    ;
143 EF 3C          DJNZ R7, LUIGI ;
145 83          RET            ;
                                ;
                                ;Routine DELAY (15 seconds)
146 23 04 DELAY:MOV A, #04H     ;
148 39          OUTL P1, A        ;Pause 3
149 34 2A          CALL STROBE    ;/ALD = 0
14B B8 2E          MOV R0, #2EH   ;Do delay to allow "n" system clocks
                                ;to
14D B9 FF          C:MOV R1, #FFH ;occur
14F BA FF          B:MOV R2, #FFH ;
151 EA 51          A:DJNZ R2, A    ;
153 E9 4F          DJNZ R1, B      ;

```



```

155 E8 4D      DJNZ R0, C      ;
157 83         RET            ;

```

```

;Addresses 00 to 09 of page three
;contain the pointers for the
;speech data located from address
;0A to 3A

```

Add	Data	Comments	Add	Data	Comments
300	0A	;Zero	32C	37	;
301	0E	;One	32D	07	;
302	09	;Two	32E	07	;
303	13	;Three	32F	23	;
304	1A	;Four	330	07	;
305	1E	;Five	331	0B	;
306	23	;Six	332	03	;3 allophones
307	2A	;Seven	333	14	; "EIGHT"
308	32	;Eight	334	02	;
309	36	;Nine	335	0D	;
30A	03	;3 allophones	336	04	;4 allophones
30B	2B	; "ZERO"	337	38	; "NINE"
30C	2C	;	338	18	;
30D	35	;	339	06	;
30E	04	;4 allophones	33A	0B	;
30F	39	; "ONE"	33B	04	;4 allophones
310	0F	;	33C	07	; "ERROR"
311	0F	;	33D	2F	;
312	0B	;	33E	3A	;
313	02	;2 allophones	33F	04	;
314	0D	;TWO	340	04	;4 allophones
315	1F	;	341	09	; "POINT"
316	03	;3 allophones	342	05	;
317	10	;THREE	343	0B	;
318	0E	;	344	11	;
319	13	;	345	07	;7 allophones
31A	03	;3 allophones	346	23	; "VOLTS"
31B	28	;FOUR	347	35	;
31C	28	;	348	2D	;
31D	3A	;	349	11	;
31E	04	;4 allophones	34A	09	;
31F	28	;FIVE	34B	37	;
320	28	;	34C	04	;
321	06	;	34D	0A	;10 allophones
322	23	;	34E	10	; "MILLIAMPERES"
323	06	;6 ALLOPHONES	34F	0C	;
324	37	;SIX	350	2D	;
325	37	;	351	0C	;
326	0C	;	352	18	;
327	02	;	353	10	;
328	29	;	354	09	;
329	37	;	355	34	;
32A	07	;7 allophones	356	37	;
32B	37	; "SEVEN"	357	04	;

Continued

### 3.4 Using a Window Comparator to Drive a Speech Processor

Window comparators are frequently employed in the design of test equipment to detect either the presence of a signal within a specified voltage range, or to detect when a signal has stepped outside the specified range.

We will design a window comparator that will be interfaced to a field programmable controller (FPC) Am29CPL151. The FPC will drive the speech processor SPO256-AL2 that will give three different messages about the input voltage. In this case, the window comparator is adjusted for detecting voltage levels in the range of 2 to 4 V; this is achieved by the resistor network R1, R2, and R3 (see Figure 3.11). In order to have a current consumption of 1 mA in this divider network, we must have a total resistor  $R_t$  as follows:

$$R_t = 5V/1 \text{ mA} = 5,000 \text{ ohms}$$

Now, the noninverting comparator “B” must receive a fixed input voltage of 2 V, and the inverting comparator “A” must receive a fixed input voltage of 4 V; consequently, a resistor network divides the power supply voltage of 5 V in two voltages of 4 and 2 V. The resistor values selected for R1, R2, and R3 are 1K, 2K, and 2K respectively. These resistor values give the  $R_t$  value that we desire for a low current consumption in this network.

In this application, the speech processor is programmed for three different cases: an input voltage lower than 2 V; an input voltage between the range of 2 to 4 V; and an input voltage higher than 4 V. These three cases are presented in binary format to the controller. Because the complete circuit designed to do this task requires only three different messages to be announced, the FPC Am29CPL151 is selected.

The three different cases with the respective selected voiced messages are shown in Table 3.4.

The messages “Lower,” “Ok,” and “Higher” indicate how the input voltage is with respect to the fixed window levels of 2 and 4 V. First, the message “Lower” means that the voltage being monitored is lower than 2 V. Second, the message “Ok” indicates that the voltage being monitored is in the pre-

**TABLE 3.4**  
Three Different Cases for the Window Comparator

Input voltage	T1	T0	Message
$0V < V_{in} < 2.00V$	0	1	"Lower"
$2V \leq V_{in} < 4.00V$	1	1	"Ok"
$4V \leq V_{in} \leq 5.00V$	1	0	"Higher"

programmed range; that is, equal to or higher than 2 V and less than 4 V. The last message "Higher" states that the input voltage being monitored is equal to or higher than 4 V.

The words of the messages were selected to permit the designer to change the window levels without having to alter the allophones that are already programmed in the FPC Am29CPL151.

The software program for the FPC Am29CPL151 is shown in Table 3.5. By looking at the circuitry shown in Figure 3.11, you will see that we are using four testable inputs, T0 to T4. Inputs T0 and T1 are directly interfaced to the FPC. The input T2 is normally held at a logic low. When the test switch is pressed, the input T2 goes momentarily to a logic high. This initiates the testing process of the input voltage which is presented in binary format at inputs T0 and T1. The input T3 is employed for monitoring the STANDBY status of the speech processor.

In most applications where the field programmable controllers of the series Am29CPL15X are driving the speech processor SPO256-AL2, the speech processor will be used in mode zero. This mode disables the /ALD input; therefore, the speech processor is triggered by applying the data byte into the input port and then by applying a byte zero (0000 0000) in order to start the

**TABLE 3.5**  
Software Program for the FPC Am29CPL151

```

DEVICE (CPL141)

DEFAULT = 1;
DEFINE    "test inputs"
higher = t0
lower = t1
test = t2
sby = t3
equal = eq
"ouput control bits"
"speech data = 59 allophones plus five pauses"
pa2 = 01#h  pa3 = 02#h  pa4 = 03#h  pa5 = 04#h  oy = 05#h  ay = 06#h
eh = 07#h  kk3 = 08#h  pp = 09#h  jh = 0A#h  nn1 = 0B#h  ih = 0C#h
tt2 = 0D#h  rr1 = 0E#h  ax = 0F#h  mm = 10#h  tt1 = 11#h  dh1 = 12#h
iy = 13#h  ey = 14#h  dd1 = 15#h  uw1 = 16#h  ao = 17#h  aa = 18#h
yy2 = 19#h  ae = 1A#h  hh1 = 1B#h  bb1 = 1C#h  th = 1D#h  uh = 1E#h
uw2 = 1F#h  aw = 20#h  dd2 = 21#h  gg3 = 22#h  vv = 23#h  gg1 = 24#h
sh = 25#h  zh = 26#h  rr2 = 27#h  ff = 28#h  kk2 = 29#h  kk1 = 2A#h
zz = 2B#h  ng = 2C#h  ll = 2D#h  ww = 2E#h  xr = 2F#h  wh = 30#h
yy1 = 31#h  ch = 32#h  er1 = 33#h  er2 = 34#h  ow = 35#h  dh2 = 36#h
ss = 37#h  nn2 = 38#h  hh2 = 39#h  or = 3A#h  ar = 3B#h  yr = 3C#h
gg2 = 3D#h  el = 3E#h  bb2 = 3F#h;

DEFAULT-OUTPUT = 0000#h;
TEST-CONDITION = sby;          "default test condition"

```

```

BEGIN
"wait for test input to go high"
"1"start:      ,if (not test) then goto pl(start);
"2"           ,load tm(03#h);
"3"           ,cmp tm(03#h) to pl (01#h);
"4"           ,if (equal) then goto pl(msg1);
"5"           ,cmp tm(03#h) to pl (02#h);
"6"           ,if (equal) then goto pl(msg2);
"7"           ,cmp tm(03#h) to pl (03#h);
"8"           ,if (equal) then goto pl(msg3);
"9" msg1: ll,   call pl(read);                "LOWER"
"10"          ow, call pl(read);
"11"          er1, call pl(read);
"12"          pa5, call pl(read);
"13"          ,goto pl(start);
"14"msg2: ao,   call pl(read);                "OK"
"15"          kk1, call pl(read);
"16"          eh, call pl(read);
"17"          ih, call pl(read);
"18"          pa5, call pl(read);
"19"          ,goto pl(start);
"20"msg3: hh1,  call pl(read);                "HIGHER"
"21"          ay, call pl(read);
"22"          er1, call pl(read);
"23"          pa5, call pl(read);
"24"          ,goto pl(start);
"25"read:      ,continue;
"26"stay:      ,if (not sby) then goto pl(stay);
"27"          ,ret;
              .org 63#d
"28"          ,goto pl(start);
END.

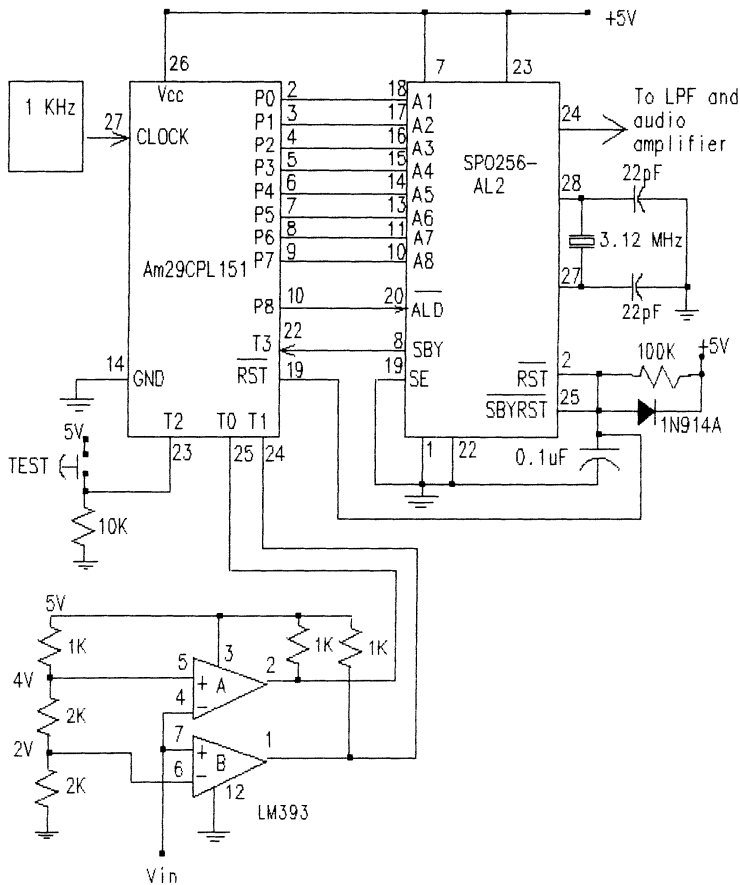
```

speech sequence. This sequence is easily solved by calling a subroutine that contains the byte zero and that also keeps reading the /SBY status of the speech processor.

The software program shown in Figure 3.13 presents the complete routine that reads the status of the window comparator and then proceeds to announce the required message.

Notice that the key word "DEVICE" is specifying the FPC Am29CPL141 even when the circuit is using the FPC Am29CPL151. Both devices are functionally identical. The Am29CPL141 is a 28-pin device, one-time programmable, and 0.6 inches wide. On the other hand, the FPC Am29CPL151 is a space-saving version of 28 pins and 0.3 inches wide. The Am29CPL151 is field programmable and contains an 32-bit per 64 words EPROM.

The key word "DEFINE" specifies the test inputs to be used with name assignments. In this case, the word "test" is used for the input T2 while the



**Figure 3.11** Circuit for driving a window comparator and a speech processor using the FPC Am29CPL151.

word “sby” is employed for the input T3. Also, the flip-flop EQ of the FPC is assigned as equal. If the flip-flop “equal” is set to one, the comparison of two numbers previously defined is equal. Continuing with the program, now the 59 allophones plus the four pauses are defined with hexadecimal numbers. In this way, you do not have to specify an equivalent number for every allophone. All you have to do is write the abbreviation of each allophone in the line in which you desire to produce the desired sound. Notice that pause1 is not defined. That is because the speech processor does not accept pause1 in mode zero.

The key word “DEFAULT\_\_OUTPUT” is used to specify the required 16-bit output at p0–p15. In this case, when a line of the program does not indicate any output, it will be full of zeros in the 16-bit output (p0–p15). The keyword “TEST\_\_CONDITION” is used to specify the STANDBY input

coming from the speech processor as the default test condition. This feature reduces the text of the program because you do not have to ask if the “sby” input is present to continue with the program.

The program starts with the key word “BEGIN.” The instruction in line one is continuously reading the input defined as “test,” that is, T2. When the user presses the “test” switch, the program automatically jumps to line two and loads the value of the test inputs T0 and T1; this is achieved by loading all the test inputs with the immediate mask 03#h (00011#b). Now lines three to eight are used to compare T0 and T1 against constant numbers. If the testable inputs are T1 = 1 and T1 = 0, the program jumps to label “msg1” where the message “Lower” must be announced. Notice that line nine calls subroutine “read” while it is giving the allophone “ll.” The program jumps to subroutine “read” when the following clock cycle arrives. The subroutine “read” (lines 25–27) starts with the instruction “continue,” which by default issues the byte zero to the outputs (p0–p7). At this point, the speech processor starts saying the first allophone and, simultaneously, the speech processor indicates that by pulsing low the /SBY output. According to the data sheet, the /SBY output spends 300 ns to go to the low state when the speech processor is triggered; therefore, the instruction “continue” (line 25) is added to give the /SBY time to go low before the FPC starts reading its logic status. It is the next instruction “If (not sby) then goto p1(stay),” located in line 26, the one that keeps reading the /SBY status. When the /SBY function goes to a logic high, the program jumps to line 27, which contains the return (ret) instruction that makes the program jump automatically to line 10.

As you can see in line 10 of the program, the instruction “call p1(read)” issues the next allophone “ow” to the speech processor and the program jumps again to subroutine “read.” This process is repeated until the program reaches line 13, which makes the internal PC counter of the FPC jump conditionally to the label “start.”

Now, with the FPC controller in line one, the program will be waiting again for the “test” input to go high. The two final instructions located below line 27 are utilized as a software reset when the program is first initialized; consequently, we can ensure that the program will start at line one when the circuitry is first turned on.

The box indicating the 1 kHz oscillator can be any kind of CMOS free-running oscillator. The frequency of 1 kHz is equivalent to pulses with a period of 1 ms; this means that the FPC executes every instruction in 1 ms. According to the software program shown in Table 3.5, the FPC will spend 2 ms between each allophone; 1 ms for the instruction “ret” and another one for the instruction “call p1(read).” It does not mean that there will be pauses of 2 ms between each allophone. Remember that the speech processor SPO256-AL2 keeps saying the last sound of each allophone until one of the pauses is asserted. That is why we are adding a pause at the end of each of the three messages (see lines 12, 18, and 23).

**TABLE 3.6**  
PROM Bit Pattern for the FPC Am29CPL151

PROM Contents:

hex <dec>	OE	OPCODE	POL	TEST	DATA	OUTPUT
000 < 0>	[ 1	11001	1	010	000000	0000000000000000
001 < 1>	[ 1	00110	0	011	000011	0000000000000000
		OPCODE	CONSTANT		DATA	
002 < 2>	[ 1	100		000001	000011	0000000000000000
003 < 3>	[ 1	11001	0	111	001000	0000000000000000
		OPCODE	CONSTANT		DATA	
004 < 4>	[ 1	100		000010	000011	0000000000000000
005 < 5>	[ 1	11001	0	111	001101	0000000000000000
		OPCODE	CONSTANT		DATA	
006 < 6>	[ 1	100		000011	000011	0000000000000000
007 < 7>	[ 1	11001	0	111	010011	0000000000000000
008 < 8>	[ 1	11100	0	011	011000	0000000000101101
009 < 9>	[ 1	11100	0	011	011000	0000000000110101
00A < 10>	[ 1	11100	0	011	011000	0000000000110011
00B < 11>	[ 1	11100	0	011	011000	0000000000000100
00C < 12>	[ 1	11001	0	011	000000	0000000000000000
00D < 13>	[ 1	11100	0	011	011000	000000000010111
00E < 14>	[ 1	11100	0	011	011000	0000000000101010
00F < 15>	[ 1	11100	0	011	011000	0000000000000111
010 < 16>	[ 1	11100	0	011	011000	0000000000001100
011 < 17>	[ 1	11100	0	011	011000	0000000000000100
012 < 18>	[ 1	11001	0	011	000000	0000000000000000
013 < 19>	[ 1	11100	0	011	011000	000000000011011
014 < 20>	[ 1	11100	0	011	011000	0000000000000110
015 < 21>	[ 1	11100	0	011	011000	0000000000110011
016 < 22>	[ 1	11100	0	011	011000	0000000000000100
017 < 23>	[ 1	11001	0	011	000000	0000000000000000
018 < 24>	[ 1	01101	1	111	111111	0000000000000000
019 < 25>	[ 1	11001	1	011	011001	0000000000000000
01A < 26>	[ 1	00010	0	011	111111	0000000000000000
03F < 63>	[ 1	11001	0	011	000000	0000000000000000

The advantage of the circuit presented in this section in relation to the circuits presented in Chapter two is that here we are using an intelligent programmable controller that saves space; therefore, the cost for building this type of circuit is much less than using MSI circuitry.

Table 3.6 shows the PROM bit pattern generated by the software ASM14X from Advanced Micro Devices. A JEDEC file (not shown) is also generated by the assembler ASM14X in order to be loaded to the PROM programmer. The PROM programmer then loads the JEDEC file into the FPC Am29CPL151 which will be used to read the window comparator and to control the speech processor.





**TABLE 3.7**  
Truth Table for PAL20R4

I n p u t s											:	O u t p u t s				:	Comments
/OC	A	B	C	D	E	F	G	H	I	J	:	C3	C2	C1	C0	:	
L	L	H	H	H	H	H	H	H	H	H		L	L	L	L		Level 0
L	L	L	H	H	H	H	H	H	H	H		L	L	L	H		Level 1
L	L	L	L	H	H	H	H	H	H	H		L	L	H	L		Level 2
L	L	L	L	L	H	H	H	H	H	H		L	L	H	H		Level 3
L	L	L	L	L	L	H	H	H	H	H		L	H	L	L		Level 4
L	L	L	L	L	L	L	H	H	H	H		L	H	L	H		Level 5
L	L	L	L	L	L	L	L	H	H	H		L	H	H	L		Level 6
L	L	L	L	L	L	L	L	L	H	H		L	H	H	H		Level 7
L	L	L	L	L	L	L	L	L	L	L	H	H	L	L	L		Level 8
L	L	L	L	L	L	L	L	L	L	L	L	H	L	L	H		Level 9
H	X	X	X	X	X	X	X	X	X	X		Hi Z					

ority encoder. The truth table and the design equations for the PAL20R4 are shown in Tables 3.7 and 3.8, respectively.

The LM3914, manufactured by National Semiconductor, is a dot/bar display driver, a chip that contains 10 independent comparators and a voltage divider network. It has a self-contained decoding network that is capable of driving the output in bar-graph mode or in single dot mode. This feature is controlled by the input MODE-CONTROL (pin 9). The chip incorporates outputs with constant current that allow direct drive of the LEDs. The LM3914 has active low outputs that generate the truth table shown in Table 3.7. In this case, the outputs of the LM3914 have been designated by the letters A to J. Notice that the level zero is indicated by a low-state “L” at output A. In this case, the readout range for the LM3914 is 0.1 to 1.0 V (0.1 V per LED). To adjust this scale, apply 1.00 V to the Vin input (pin 5) and adjust the 1K pot (R1) until LED 10 (output A) glows. Repeat this procedure by applying 0.1 V at Vin input and check that the output one (pin 1) goes low.

In order to make the best possible interface of the LM3914 to the FPC Am29CPL152, we need to encode the 10 bits of the LM3914 by using a 10-to-4 line priority encoder. The encoder must be able to accept low-active inputs and must contain a 4-bit register at the output. Accordingly, the 4-bit register contained in the encoder will be storing the 4-bit readings so that the FPC can take a reading without problems. Also, the encoder must have priority inputs to permit the user to select both modes of control: dot or bar.

**TABLE 3.8**  
Design Equations for PAL20R4

---

```

C0 := /A * /B
      + /A * /B * /C * /D
      + /A * /B * /C * /D * /E * /F
      + /A * /B * /C * /D * /E * /F * /G * /H
      + /A * /B * /C * /D * /E * /F * /G * /H * /I * /J

C1 := /A * /B * /C
      + /A * /B * /C * /D
      + /A * /B * /C * /D * /E * /F * /G
      + /A * /B * /C * /D * /E * /F * /G * /H

C2 := /A * /B * /C * /D * /E
      + /A * /B * /C * /D * /E * /F
      + /A * /B * /C * /D * /E * /F * /G
      + /A * /B * /C * /D * /E * /F * /G * /H

C3 := /A * /B * /C * /D * /E * /F * /G * /H * /I
      + /A * /B * /C * /D * /E * /F * /G * /H * /I * /J

```

---

Table 3.7 shows the 10 inputs named A to J, and the four encoded outputs C3 to C0. From Table 3.7 we obtain the design equations for the PAL20R4. The design equations are obtained using minterms for each of the outputs. The symbol “:=” means that the respective output is registered with a flip-flop that is self-contained in the PAL20R4.

You can use any type of software to assemble and simulate your design equations.

The software program for the Am29CPL152 is shown in Table 3.9. In the circuitry shown in Figure 3.12, the PAL20R4 outputs are routed to the four testable inputs T0, T1, T2, and T3 of the FPC Am29CPL152. Outputs P0 to P7 are dedicated to load the speech entry points to the speech processor. The output P8 of the FPC is used to control the latch function of the PAL20R4. When the latch pin makes a low-to-high transition, the registered outputs C0 to C4 are latched.

The first line of the software program after the keyword “BEGIN” (see Table 3.9) corresponds to the instruction “if (not tst) then goto p1(stay).” This instruction keeps the program continuously reading the “test” switch which is connected to the input T5 of the FPC. When the “test” switch is pressed momentarily, a logic high is present at the input T5, causing the program jump to line two. The instruction “continue” in line two is used only to give a logic high at the latch input of the PAL20R4. For instance, the 4-bit encoded output of the PAL20R4 is latched. Now the program jumps to line three where the

**TABLE 3.9**  
Software Program for the FPC Am29CPL152

```
DEVICE (CPL142)
```

```
DEFAULT = 1;
```

```
DEFINE      "test inputs"
```

```
C0 = t0
```

```
C1 = t1
```

```
C2 = t2
```

```
C3 = t3
```

```
tst = t5
```

```
equal = eq
```

```
sby = t4
```

```
"ouput control bits"
```

```
"speech data = 59 allophones plus five pauses"
```

```
pa2 = 01#h  pa3 = 02#h  pa4 = 03#h  pa5 = 04#h  oy = 05#h  ay = 06#h
```

```
eh = 07#h  kk3 = 08#h  pp = 09#h  jh = 0A#h  nn1 = 0B#h  ih = 0C#h
```

```
tt2 = 0D#h  rr1 = 0E#h  ax = 0F#h  mm = 10#h  tt1 = 11#h  dh1 = 12#h
```

```
iy = 13#h  ey = 14#h  dd1 = 15#h  uw1 = 16#h  ao = 17#h  aa = 18#h
```

```
yy2 = 19#h  ae = 1A#h  hh1 = 1B#h  bb1 = 1C#h  th = 1D#h  uh = 1E#h
```

```
uw2 = 1F#h  aw = 20#h  dd2 = 21#h  gg3 = 22#h  vv = 23#h  gg1 = 24#h
```

```
sh = 25#h  zh = 26#h  rr2 = 27#h  ff = 28#h  kk2 = 29#h  kk1
```

```
= 2A#h  zz = 2B#h  ng = 2C#h  ll = 2D#h  ww = 2E#h  xr = 2F#h  wh
```

```
= 30#h
```

```
yy1 = 31#h  ch = 32#h  er1 = 33#h  er2 = 34#h  ow = 35#h  dh2 = 36#h
```

```
ss = 37#h  nn2 = 38#h  hh2 = 39#h  or = 3A#h  ar = 3B#h  yr = 3C#h
```

```
gg2 = 3D#h  el = 3E#h  bb2 = 3F#h
```

```
latch = 100#h;
```

```
DEFAULT_OUTPUT = 0000#h;
```

```
TEST_CONDITION = SBY;  "default test condition"
```

```
BEGIN
```

```
"wait for test input to go high. SE = 0 (pin 19 of SP0256-AL2)"
```

```
"1"stay:      ,if (not tst) then goto pl(stay);
```

```
"2"  latch, continue; "Registers are loaded on the low-high pulse."
```

```
"3"  latch, load tm(0F#h);
```

```
"4"  ll,      call pl(read);  "LEVEL"
```

```
"5"  eh,      call pl(read);
```

```
"6"  vv,      call pl(read);
```

```
"7"  el,      call pl(read);
```

```
"8"  pa5,     call pl(read);
```

```
"9"  ,cmp tm(0F#h) to pl(00#h);
```

```
"10" ,if (equal) then goto pl(zero);
```

```
"11" ,cmp tm(0F#h) to pl(01#h);
```

```

"12"      ,if (equal) then goto pl(one);
"13"      ,cmp tm(0F#h) to pl(02#h);
"14"      ,if (equal) then goto pl(two);
"15"      ,cmp tm(0F#h) to pl(03#h);
"16"      ,if (equal) then goto pl(thre);
"17"      ,cmp tm(0F#h) to pl(04#h);
"18"      ,if (equal) then goto pl(four);
"19"      ,cmp tm(0F#h) to pl(05#h);
"20"      ,if (equal) then goto pl(five);
"21"      ,cmp tm(0F#h) to pl(06#h);
"22"      ,if (equal) then goto pl(six);
"23"      ,cmp tm(0F#h) to pl(07#h);
"24"      ,if (equal) then goto pl(svn);
"25"      ,cmp tm(0F#h) to pl(08#h);
"26"      ,if (equal) then goto pl(eit);
"27"      ,cmp tm(0F#h) to pl(09#h);
"29"      ,if (equal) then goto pl(nin);
"30"      ,goto pl(stay);

"routine for the word zero"
"31"zero:zz, call pl(read);
"32"  yr, call pl(read);
"33"  ow, call pl(read);
"34"  pa4, call pl(read);
"35"      ,goto pl(stay);

"routine for the word one"
"36"one:ww, call pl(read);
"37"  ax, call pl(read);
"38"  ax, call pl(read);
"39"  nn1, call pl(read);
"40"  pa4, call pl(read);
"41"      ,goto pl(stay);

"routine for the word two"
"42"two:tt2, call pl(read);
"43"  uw2, call pl(read);
"44"  pa4, call pl(read);
"45"      ,goto pl(stay);

"routine for the word three"
"46"thre:th, call pl(read);
"47"  rrl, call pl(read);
"48"  iy, call pl(read);
"49"  pa4, call pl(read);
"50"      ,goto pl(stay);

"routine for the word four"
"51"four:ff, call pl(read);
"52"  ff, call pl(read);
"53"  or, call pl(read);
"54"  pa4, call pl(read);
"55"      ,goto pl(stay);

```

```

"routine for the word five"
"56"five:ff, call pl(read);
"57" ff, call pl(read);
"58" ay, call pl(read);
"59" vv, call pl(read);
"60" pa4, call pl(read);
"61" ,goto pl(stay);

"routine for the word six"
"62"six:ss, call pl(read);
"63" ss, call pl(read);
"64" ih, call pl(read);
"65" ih, call pl(read);
"66" pa3, call pl(read);
"67" kk2, call pl(read);
"68" ss, call pl(read);
"69" pa4, call pl(read);
"70" ,goto pl(stay);

"routine for the word seven"
"71"svn:ss, call pl(read);
"72" ss, call pl(read);
"73" eh, call pl(read);
"74" eh, call pl(read);
"75" vv, call pl(read);
"76" eh, call pl(read);
"77" nn1, call pl(read);
"78" pa4, call pl(read);
"79" ,goto pl(stay);

"routine for the word eight"
"80"eit:ey, call pl(read);
"81" pa3, call pl(read);
"82" tt2, call pl(read);
"83" pa4, call pl(read);
"84" ,goto pl(stay);

"routine for the word nine"
"85"nin:nn2, call pl(read);
"86" aa, call pl(read);
"87" ay, call pl(read);
"88" nn1, call pl(read);
"89" pa4, call pl(read);
"90" ,goto pl(stay);

"subroutine for reading the standby status of the speech processor"
"91"read: ,continue;
"92"styl: ,if (not sby) then goto pl(styl); "reading SBY"
"93" ,ret;

.org 127#d
"94" ,goto pl(stay);

END.

```

instruction “load tm(0F#h)” is used for loading only four of the six testable inputs. Because we only need to read the testable inputs T0 to T3, the immediate mask “0F#h” is used.

Lines four to seven contain four allophones and one pause in order to make the speech processor say the word “level.” Certainly, when the user hears the word “level,” he will be expecting to hear a number that indicates in what level the reading is at that moment. Lines 9 to 29 compare the 4-bit reading against a constant number. If the 4-bit reading is equal to the constant number stored in the p1 field, the internal flag “eq” is set to one. In this case, the flag “eq” was assigned the name “equal”; therefore, the program keeps reading for this flag after every comparison. If the flag is set to one, the program jumps to the routine indicated in the p1 field. Once the program jumps to the routine used for announcing the respective number, the routine issues the required allophones and keeps calling the “read” subroutine. It is the “read” subroutine that starts the speech sequence of the speech processor (see lines 91 to 93). When the respective number has been correctly announced, the program jumps again to the beginning at line one that is labeled “stay.” Here the program waits again for the test switch to be pressed momentarily in order to repeat the process.

You can make certain modifications to the software program presented in Table 3.9. For example, you can add a time delay at the beginning of the program and avoid reading the status of the “test” switch. In this way, your program will be reading and announcing continuously the encoded reading that corresponds to the voltage level presented at the input Vin of the LM3914. In the program presented here, a total of 94 lines of real instructions were spent; consequently, you can augment the size of the program for more specific instructions. For example, you can add warning messages if the voltage level has reached a “dangerous” zone or a message of the steps that must be followed when the voltage level has dropped below a “permitted” level. As you can see, the flexibility of this circuit relies on the capacity of the field programmable controller to be reprogrammed according to the needs of the user.

### 3.6 Interfacing a Speech Processor to a Logic Probe

The multifunction logic probe shown in Figure 3.13 is able to indicate logic states and the presence of pulses. In digital circuits, finding out if the the clock section of a module is functioning properly can be a difficult job, especially when you do not have access to an oscilloscope.

The logic probe presented in Figure 3.13 is divided into two parts: the logic part and the programmable part. The logic part corresponds to the upper portion of the circuit. Here, the circuit will be working as a conventional logic probe that indicates the state of the input by merely glowing two LEDs. When



also triggered for an interval of 0.33 s. Notice that the negative output of the second monostable is utilized for disabling the first monostable for the period of 0.33 s. This feature permits the monitoring of a continuous stream of pulses which will be heard as an intermittent sound caused by the piezotransducer. Otherwise, a continuous sound generated by the buzzer would be indicating the presence of pulses.

The main feature of this logic probe is that it has the capability of indicating the logic state of an input as well as the presence of pulses by means of a speech processor. This enables the user to monitor or test a digital circuit without having to look at the analog or digital display and waste precious time.

The logic probe is interfaced to the speech processor SPO256-AL2 by means of the field programmable controller Am29CPL151. In this case, T0 is designated as the logic input, while T1 is connected to a DPDT switch that will be indicating the mode of operation. When the mode of operation is PULSE, the testable input T1 is held at a logic low. By looking at Table 3.10 you will see that the keyword "DEFINE" assigns the name "probe" to input T0, the name "pulse" for input T1, and the name "sby" for input T2. After all the allophones have been assigned with hexadecimal equivalents, the program starts at line one.

**TABLE 3.10**

Software Program to Control the Logic Probe Using the FPC Am29CPL151

```

DEVICE (CPL141)

DEFAULT = 1;
DEFINE      "test inputs"
prob = t0
puls = t1
sby = t2
"speech data = 59 allophones plus four pauses"
pa2 = 01#h  pa3 = 02#h  pa4 = 03#h  pa5 = 04#h  oy = 05#h  ay = 06#h
eh = 07#h  kk3 = 08#h  pp = 09#h  jh = 0A#h  nn1 = 0B#h  ih = 0C#h
tt2 = 0D#h  rr1 = 0E#h  ax = 0F#h  mm = 10#h  tt1 = 11#h  dh1 = 12#h
iy = 13#h  ey = 14#h  dd1 = 15#h  uw1 = 16#h  ao = 17#h  aa = 18#h
yy2 = 19#h  ae = 1A#h  hh1 = 1B#h  bb1 = 1C#h  th = 1D#h  uh = 1E#h
uw2 = 1F#h  aw = 20#h  dd2 = 21#h  gg3 = 22#h  vv = 23#h  gg1 = 24#h
sh = 25#h  zh = 26#h  rr2 = 27#h  ff = 28#h  kk2 = 29#h  kk1 = 2A#h
zz = 2B#h  ng = 2C#h  ll = 2D#h  ww = 2E#h  xr = 2F#h  wh = 30#h
yy1 = 31#h  ch = 32#h  er1 = 33#h  er2 = 34#h  ow = 35#h  dh2 = 36#h
ss = 37#h  nn2 = 38#h  hh2 = 39#h  or = 3A#h  ar = 3B#h  yr = 3C#h
gg2 = 3D#h  el = 3E#h  bb2 = 3F#h;

DEFAULT_OUTPUT = 0000#h;
TEST_CONDITION = sby;  "default test condition"

```



```

BEGIN
"read t1 for mode pulse or mode logic"
"1"start:  ,if (puls) then goto pl(edge);
"2"        ,if (not prob) then goto pl(one);
"3"  zz,    call pl(read);  "ZERO"
"4"  yr,    call pl(read);
"5"  ow,    call pl(read);
"6"  pa5,   call pl(read);
"7"  pa5,   call pl(read);
"8"        ,goto pl(start);
"9"one: ww,  call pl(read);  "ONE"
"10" ax,    call pl(read);
"11" ax,    call pl(read);
"12" nn1,   call pl(read);
"13" pa5,   call pl(read);
"14" pa5,   call pl(read);
"15"        ,goto pl(start);
"16"edge:   ,if (not prob) then goto pl(edge);
"17" pp,    call pl(read);
"18" uh,    call pl(read);
"19" ll,    call pl(read);
"20" ss,    call pl(read);
"21" pa5,   call pl(read);
"22" pa5,   call pl(read);
"23"        ,goto pl(start);
"24"read:   ,continue;
"25"stay:   ,if (not sby) then goto pl(stay);  "reading SBY"
"26"        ,ret;
          ,org 63#d
"27"        ,goto pl(start);
END.

```

The program first asks for the mode of selection, that is, whether it is LOGIC or PULSE that the user has chosen. In this case the program was designed for monitoring the signal automatically, so the user does not spend time pressing a switch for each test. If the mode LOGIC is selected, the program jumps to line two, which reads the logic status of the input named "probe." If the input pulse is low, the program makes the speech processor say the word "zero"; otherwise, the speech processor will announce the word "one." When the announcing of the word ends, the program goes back again to line one in order to read the new status of the input assigned as "probe."

On the other hand, if the user selects the logic probe in the mode "PULSE," the first instruction makes the program jump to the label "edge" located in line 6. The instruction "if (not probe) then goto pl(edge)" keeps the FPC reading the status of the input "probe." If a low-to-high transition occurs at

the input of the logic probe, the speech processor will say the word “pulse.” When this word ends, the program goes back to line one in order to know if the mode of operation has changed. In fact, after any of the three messages is heard, the program goes back and reads the mode of operation for a possible new selection.

In the program shown in Table 3.10, you can see that only 26 lines were spent for developing the program; therefore, you have lines 27 to 63 free for augmenting or adapting the program with some different features. For example, you can add a low-battery indicator to this logic probe, so that the program monitors when the voltage has dropped below a preprogrammed level. In this case, the speech processor would issue a warning message indicating that the battery is losing power. Many other features can be added to this logic probe, but most of them will depend upon the imagination of the user.

### 3.7 Talking Programmable Gain Amplifier

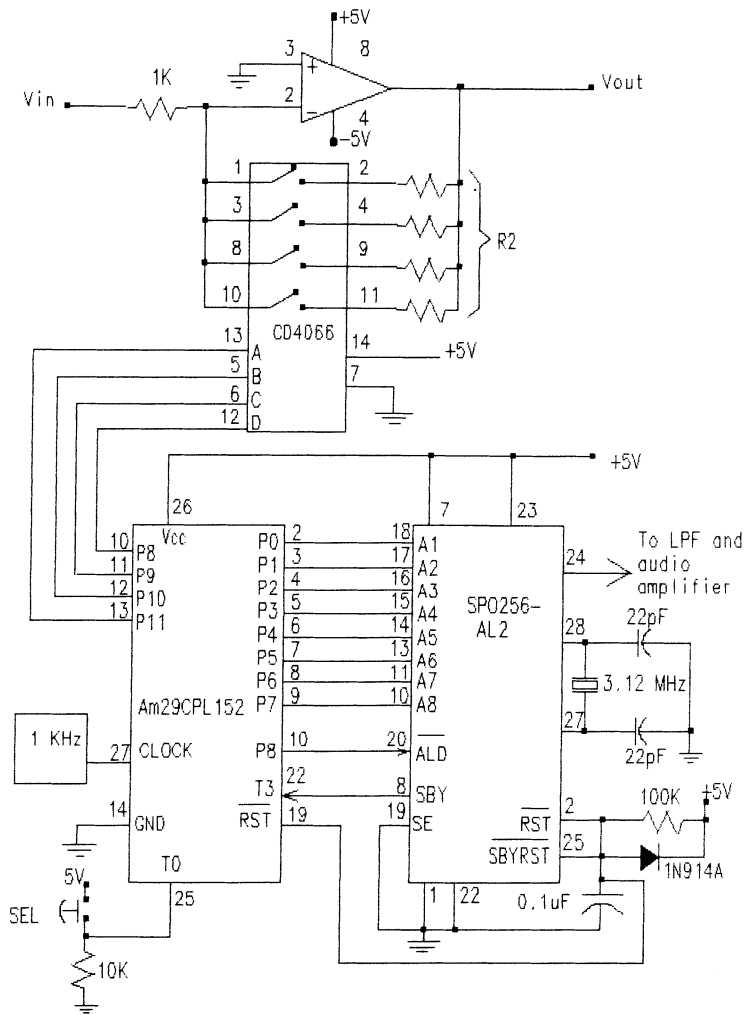
The Figure 3.14 circuit is a programmable gain amplifier that has the capacity of selecting 16 different gains by means of a switching network. The switching network is controlled by the FPC Am29CPL151. In addition, the speech processor SPO256-AL2 announces the gain every time the user selects it.

The op-amp LF353 configured as an inverting amplifier is controlled by the CMOS switching network contained in the IC CD4066. A set of four resistors connected to the CD4066 is used to make 16 possible combinations. In other words, 16 possible resistor values are obtained to control the gain of the op-amp. The gain equation for the inverting amplifier is

$$V_{out} = -V_{in} (R_2/R_1)$$

In the preceding equation, resistor R2 can have 16 different values, starting with the value of zero ohms. To make the selection of resistors, the four control inputs named A, B, C, and D will select the required combination via the control outputs p8, p9, p10, and p11 provided by the FPC Am29CPL151.

After the power-up reset, the circuit starts selecting the value of resistor two as zero ohms. To select the next 15 gain levels, the user only has to press the normally open switch “SEL.” When the user presses this switch for the first time, the first gain level is selected and the resistor Ra is connected. This event causes the speech processor to say the message “level one.” If the user needs a higher gain, he will have to press the switch “SEL” again. In this way, the speech processor will be announcing the selected gain level just after the switch is pressed.



**Figure 3.14** Circuitry for the talking programmable gain amplifier.

The software program for controlling the gain of the op-amp and the speech processor is shown in Table 3.11. The section “DEFINE” of the program assigns the names of “SEL” and “sby” to T0 and T1 respectively. Also, allophones and pauses are defined in the same way as in the previous programs.

Once you have written your program as a pure ascii file, you can assemble it to produce the JEDEC file. The JEDEC file must be loaded into the FPC Am29CPL151 by using an appropriate PLD programmer.

**TABLE 3.11**  
Software Program for the Programmable Gain Amplifier

```

DEVICE (CPL141)

DEFAULT = 1;

DEFINE    "test inputs"
sel = t0
sby = t1

"output control bits are given allophone assignments"
"speech data = 59 allophones plus four pauses"
pa2 = 01#h      "      -----      "
pa3 = 02#h      "/zero->|1          28|- Vcc "
pa4 = 03#h      "    p0<-|2          27|<-clk "
pa5 = 04#h      "    p1<-|3          26|<-cc  "
oy  = 05#h      "    p2<-|4          25|<-t0  "
ay  = 06#h      "    p3<-|5    Am29CPL 24|<-t1  "
eh  = 07#h      "    p4<-|6      151  23|<-t2  "
kk3 = 08#h      "    p5<-|7          22|<-t3  "
pp  = 09#h      "    p6<-|8          21|<-t4  "
jh  = 0A#h      "    p7<-|9          20|<-t5  "
nn1 = 0B#h      "    p8<-|10         19|<-/reset"
ih  = 0C#h      "    p9<-|11         18|<-p15  "
tt2 = 0D#h      "    p10<-|12        17|<-p14  "
rr1 = 0E#h      "    p11<-|13        16|<-p13  "
ax  = 0F#h      "    Gnd-|14         15|<-p12  "
mm  = 10#h      "      -----      "
tt1 = 11#h  dh1 = 12#h  iy  = 13#h  ey  = 14#h  dd1 = 15#h  uw1 = 16#h
ao  = 17#h  aa  = 18#h  yy2 = 19#h  ae  = 1A#h  hh1 = 1B#h  bb1 = 1C#h
th  = 1D#h  uh  = 1E#h  uw2 = 1F#h  aw  = 20#h  dd2 = 21#h  gg3 = 22#h
vv  = 23#h  gg1 = 24#h  sh  = 25#h  zh  = 26#h  rr2 = 27#h  ff  = 28#h
kk2 = 29#h  kk1 = 2A#h  zz  = 2B#h  ng  = 2C#h  ll  = 2D#h  ww  = 2E#h
xr  = 2F#h  wh  = 30#h  yy1 = 31#h  ch  = 32#h  er1 = 33#h  er2 = 34#h
ow  = 35#h  dh2 = 36#h  ss  = 37#h  nn2 = 38#h  hh2 = 39#h  or  = 3A#h
ar  = 3B#h  yr  = 3C#h  gg2 = 3D#h  el  = 3E#h  bb2 = 3F#h

gain1 = 100#h
gain2 = 200#h
gain3 = 300#h
gain4 = 400#h
gain5 = 500#h
gain6 = 600#h
gain7 = 700#h;

DEFAULT-OUTPUT = 0000#h;

TEST-CONDITION = SBY;    "default test condition"

BEGIN
"read t1 for mode pulse or mode logic"
"1"stay;      ,if (not sel) then goto pl(stay); "Gain zero"
"2"  ww,      call pl(read); "ONE"
"3"  ax,      call pl(read);
"4"  ax,      call pl(read);
"5"  nn1,     call pl(read);
"6"  pa2,     call pl(read);

```

```

"7"sty1:gain1, if (not sel) then goto pl(sty1);
"8"    tt2,    call pl(read); "TWO"
"9"    uw2,    call pl(read);
"10"   pa2,    call pl(read);
"11"sty2:gain2, if (not sel) then goto pl(sty2);
"12"    th,    call pl(read); "THREE"
"13"    rr1,    call pl(read);
"14"    iy,    call pl(read);
"15"    pa2,    call pl(read);
"16"sty3:gain3, if (not sel) then goto pl(sty3);
"17"    ff,    call pl(read); "FOUR"
"18"    ff,    call pl(read);
"19"    or,    call pl(read);
"20"    pa2,    call pl(read);
"21"sty4:gain4, if (not sel) then goto pl(sty4);
"22"    ff,    call pl(read);
"23"    ff,    call pl(read);
"24"    ay,    call pl(read);
"25"    vv,    call pl(read);
"26"    pa2,    call pl(read);
"27"sty5:gain5, if (not sel) then goto pl(sty5);
"28"    ss,    call pl(read);
"29"    ss,    call pl(read);
"30"    ih,    call pl(read);
"31"    ih,    call pl(read);
"32"    pa2,    call pl(read);
"33"    kk2,    call pl(read);
"34"    ss,    call pl(read);
"35"    pa2,    call pl(read);
"36"sty6:gain6, if (not sel) then goto pl(sty6);
"37"    ss,    call pl(read);
"38"    ss,    call pl(read);
"39"    eh,    call pl(read);
"40"    eh,    call pl(read);
"41"    vv,    call pl(read);
"42"    eh,    call pl(read);
"43"    nn1,    call pl(read);
"44"    pa2,    call pl(read);
"45"sty7:gain7, if (not sel) then goto pl(sty7);
"46"    ,goto pl(stay);
"47"read:    ,continue;
"48"sty8:    ,if (not sby) then goto pl(sty8);  "reading SBY"
"49"    ,ret;
    ,org 63#d
"50"    ,goto pl(stay);
END.

```

### 3.8 Designing an Electronic Thermometer that Announces Readings

The Figure 3.15 circuit is an LED digital thermometer that has an operating range of  $-5^{\circ}\text{C}$  to  $45^{\circ}\text{C}$  with a resolution of  $0.1^{\circ}\text{C}$ . Thermistor YSI44202 is manufactured by Yellow Springs Instruments and is used as the sensor element that sends the voltage variations to ICL7117's input (pins 31 and 30) via the instrumentation amplifier formed by IC2, IC3, and IC4.

The temperature range used in the thermometer was selected based on biological applications. The main advantage for the LED digital thermometer circuit is that it can measure small changes in temperature accurately and quickly in comparison to other conventional thermometers such as mercury thermometers. The temperature measurements are exhibited in an LED display and announced by a speech processor simultaneously.

The thermoliner sensor contains a linearization network formed by two thermistors and two resistors (see Figure 3.16). This network provides a linear output voltage versus temperature.

The output voltage  $V_o$  of the thermistor is described by the equation:

$$V_o = 0.77 + (0.028875 \text{ V}/^{\circ}\text{C})T$$

By substituting the temperature  $T$  from  $-5^{\circ}\text{C}$  to  $45^{\circ}\text{C}$  in the preceding equation, we get Table 3.12.

The 3-V power supply biasing the thermistor is used to avoid the errors produced by internal heating in the sensor, which would produce wrong readings.

An instrumentation amplifier (see Figure 3.17) is employed in order to get accurate readings at the ICL7117's input. The instrumentation amplifier is a fixed-gain differential amplifier consisting of three op-amps, as indicated in the circuitry of Figure 3.17. The gain expression is formally the same as that for an op-amp, for example,

$$V_o = A(E_1 - E_2)$$

except that the open-loop gain is replaced by the gain with feedback  $A$ . Specifically,

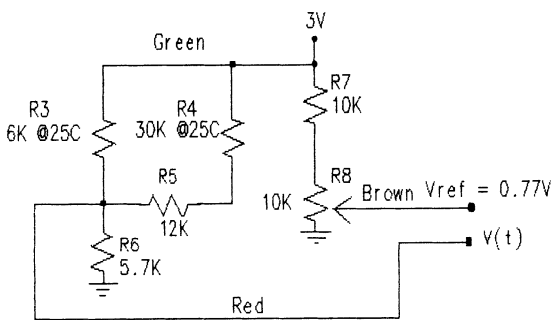
$$A = (2 R_{11} + R_{10}) R_{15}/(R_{10} R_{13})$$

It is basically an improved version of the differential amplifier. The main features are (1) high input impedance, especially with FET op-amps on the input; (2) high CMRR, and (3) precision high gain. High input impedance is achieved by using the noninverting amplifier configuration on the inputs. Precision high gain is obtained by two stages of feedback amplifiers. High common-mode rejection is achieved by the dual noninverting-configuration circuit, which utilizes a common feedback resistor  $R_{10}$ .









**Figure 3.16** Thermistor linearization network.

The gain “A” is given by:

$$A = V_o/(E_1 - E_2) = R_{15}/R_{13}(1 + 2 R_{11}/R_{10}) = 0.3463$$

From Table 3.12 we pick up the voltage values of  $V_o$  to verify the output voltage  $V_o$  of the instrumentation amplifier depending upon the temperature; thus we get Table 3.13. Notice that  $V(t)$  is the output voltage of the thermistor network. In Table 3.13  $V_o$  is now the output voltage that is routed to the ICL7117. The values of  $V_o$  are obtained by substituting the gain A in Eq.  $V_o = A(E_1 - E_2)$ .

In the instrumentation amplifier, resistor R15 is used to adjust the common mode gain, and resistor R10 is used to adjust the gain (see Figure 3.17).

The ICL7117 is a 3-1/2 digit single-chip converter. This A/D converter

**TABLE 3.12**  
Temperature versus Voltage for the Thermistor Network

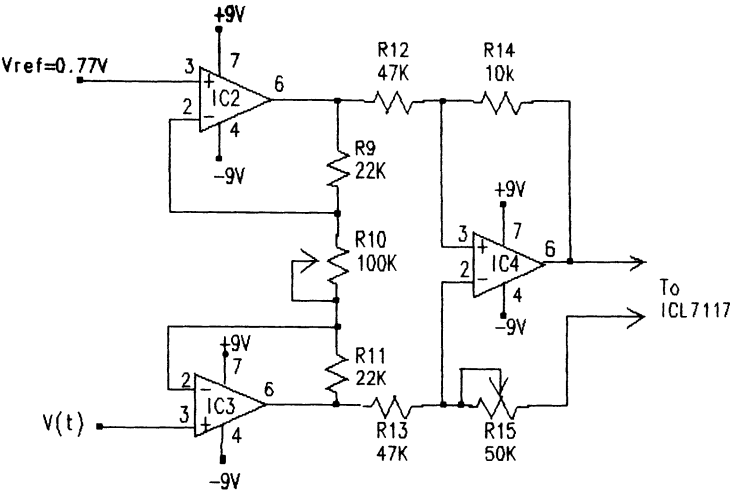
T (°C)	V <sub>o</sub> (Volts)
-5	0.6256
0	0.7700
5	0.9143
10	1.0587
15	1.2031
20	1.3475
25	1.4918
35	1.7806
40	1.9250
45	2.0693

**TABLE 3.13**  
Typical Values of  $V_o$  versus Temperature When  $V_{ref} = 0.77V$

T (C)	$V_{ref}$ (E2)	$V(t)$ (E1)	$V_o = A (E1 - E2)$
-5	0.77	0.6256	-0.05
0	0.77	0.7700	0.00
5	0.77	0.9143	0.05
10	0.77	1.0587	0.10
15	0.77	1.2031	0.15
20	0.77	1.3475	0.20
25	0.77	1.4918	0.25
30	0.77	1.6362	0.30
35	0.77	1.7806	0.35
40	0.77	1.9250	0.40
45	0.77	2.0693	0.45

contains all the necessary active devices in a single CMOS IC. Included are seven segment decoders, display drivers, a voltage reference, and a clock. The ICL7117 is designed to interface with a common anode LED display.

To interface the temperature readings to a speech processor, we have to convert the low-active seven segment outputs of the ICL7117 to a BCD code.



**Figure 3.17** The instrumentation amplifier.

The conversion is achieved by using three seven-segment-to-BCD decoders 74C915. The LSB digit that represents the decimal values of the temperature is sent to P2.0–P2.3 of the microcontroller. Because the microcontroller’s operating voltage is 5 V, the BCD code received by port two as well as by the BUS must be in the logical levels of 0 to 5 V. To reduce the logical levels from 9 to 5 V, 12 voltage followers contained in two CD4050s are used. The output P2.7 of the  $\mu\text{C}$  8748 is selected for handling the hold reading (HLDR) function of the ICL7117. In this way, the  $\mu\text{C}$  8748 holds the temperature reading temporarily and selects the three 74C915s by holding low the input control (/OC) via the output P2.6.

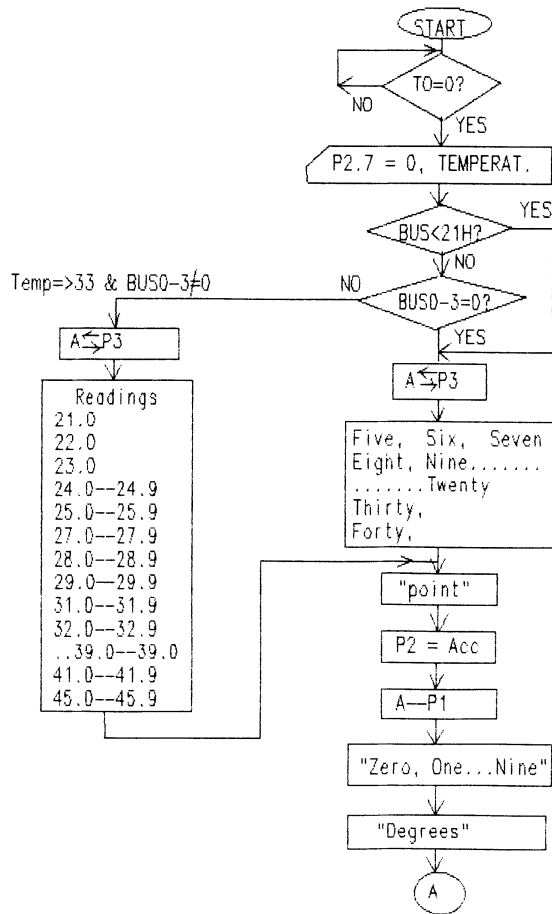
Now, the software program of the microcontroller is the one responsible for making the decisions to drive correctly the speech processor Digitalker. The Digitalker will proceed to announce the temperature readings every time the user presses the TEST switch. The software program used to convert the BCD code temperature readings to vocal messages is described below.

Table 3.14 shows different cases for temperature readings presented in BCD code.

By looking at Table 3.14, you will see that some temperature readings require three words and others require four words. For temperature readings be-

**TABLE 3.14**  
A Temperature Reading Presented in BCD Code at the Inputs Port 2.X and BUS,  
Causes a Spoken Message in the Speech Processor

Message	Temperature	B U S								P o r t 2 . X					
		7	6	5	4	3	2	1	0	3	2	1	0		
five point zero	5.0	:	0	0	0	0	0	1	0	1	:	0	0	0	0
five point one	5.1	:	0	0	0	0	0	1	0	1	:	0	0	0	1
six point zero	6.0	:	0	0	0	0	0	1	1	0	:	0	0	0	0
nine point nine	9.9	:	0	0	0	0	1	0	0	1	:	1	0	0	1
ten point zero	10.0	:	0	0	0	1	0	0	0	0	:	0	0	0	0
nineteen point zero	19.0	:	0	0	0	1	1	0	0	1	:	0	0	0	0
twenty point zero	20.0	:	0	0	1	0	0	0	0	0	:	0	0	0	0
twenty one point zero	21.0	:	0	0	1	0	0	0	0	0	:	0	0	0	0
twenty one point five	21.5	:	0	0	1	0	0	0	0	1	:	0	1	0	1
twenty two point zero	22.0	:	0	0	1	0	0	0	1	0	:	0	0	0	0
thirty point zero	30.0	:	0	0	1	1	0	0	0	0	:	0	0	0	0
thirty point nine	30.9	:	0	0	1	1	0	0	0	0	:	1	0	0	1
thirty one point zero	31.0	:	0	0	1	1	0	0	0	1	:	0	0	0	0
thirty nine point zero	39.0	:	0	0	1	1	1	0	0	1	:	0	0	0	0
forty point zero	40.0	:	0	1	0	0	0	0	0	0	:	0	0	0	0
forty one point zero	41.0	:	0	1	0	0	0	0	0	1	:	0	0	0	0
forty five point zero	45.0	:	0	1	0	0	0	1	0	1	:	0	0	0	0



**Figure 3.18** Flow chart for converting temperature readings in BCD code to vocal messages.

low 20.9° C, the message contains three words. Furthermore, the readings that range from 30.0 to 30.9° C and the readings from 40.0 to 40.9° C are formed with three words; however, in readings varying from 21.0 to 45.0 the messages are formed with four words, with the exception of the two ranges just described above. A flowchart will be helpful to understand the routes that the program must follow in order to announce a temperature reading correctly (see Figure 3.18).

The microcontroller's routine for reading the temperature in BCD code and then translating it to a vocal message is shown in Table 3.15. In this case the Digitalker system is used.

**TABLE 3.15**

Software Program for the  $\mu\text{C}$  8748 Which Makes the Digitalker System to Announce Temperature Readings Which Are Received in BCD Code

Add	Op	Code	Mnemonics	Comments
00	26	00	STAY: JTO STAY	;Wait for T0 to go low
02	8A	FF	ORL P2, #FFH	;HLDR function of ICL7117 is
04	9A	EF	ANL P2, #EFH	;is pulsed low to hold a
06	8A	FF	ORL P2, #FFH	;new reading
08	9A	BF	ANL P2, #BFH	;1011 1111, /OC = 0
0A	08		INS A, BUS	;Load temperature reading
0B	A9		MOV R1, A	;Register R1 contains units
				;and tens of the temperature.
0C	0A		IN A, P2	;Acc gets decimal value
0D	AA		MOV R2, A	;R2 contains the decimal value
0E	8A	FF	ORL P2, #FFH	;
10	97		CLR C	;Carry flag is set to zero.
11	BB	21	MOV R3, #21H	;Register R3 = #33d
13	37		CPL A	;Routine to compare if A<R3
14	6B		ADD A, R3	;If CY is set then A<R3
15	27		CLR A	;
16	F6	23	JC POINTR	;Jump to POINTR if CY is one
18	F9		MOV A, R1	;Load Acc with temperature
19	53	0F	ANL A, #0FH	;Mask the Acc to test if the
1B	96	48	JNZ FIND	;units of temp are no zero.
1D	F9		POINTR: MOV A, R1	;Load Acc with temperature
1E	53	0F	ANL A, #F0H	;Acc contains tens of temp.
20	14	3A	CALL PULSE1	;
22	F9		MOV A, R1	;Load Acc with temperature
23	53	0F	ANL A, #0FH	;Acc contains units of temp.
25	14	3A	CALL PULSE1	;
27	89	7A	ORL P1, #7AH	;Word "point"
29	14	3C	CALL PULSE2	;
2B	27		CLR A	;
2C	FA		DCM: MOV A, R2	;Decimal value of temperature
2D	53	0F	ANL A, #0FH	;is loaded into Acc
2F	C6		JZ ZERO	;Testing if temp = zero.
30	14	3A	CALL PULSE1	;
32	89	72d	CD: ORL P1, #72d	;Word "degree"
34	14	3C	CALL PULSE2	;
36	89	129d	ORL P1, #129d	;Sound "ss"
38	14	3C	CALL PULSE2	;
3A	04	00	JMP START	;
3B	E3		PULSE1: MOV P3 A, @	
A			;Find the speech data in page3	
3C	39		OUTL P1, A	;Load port1 with speech data
3D	80		PULSE2: MOVX @RO, A	; /WR of DT1050 is pulsed low
3E	46	3D	STAY: JNT1, STAY	;Wait for /INTR to go low
40	99	00	ANL P1, #00H	;Clear port1
42	83		RET	;

```

43 23      ZERO:MOV A, #1FH      ;Load Acc with data 1FH
44 39      OUTL P1, A           ;Load port1 with word "zero"
45 14 3C      CALL PULSE2      ;
47 04 32      JMP CD           ;
49 27      FIND:CLR A          ;
4A F9      MOV A, R1           ;Temp<#33d (<21 C)
4B 14 3A      CALL PULSE1      ;
4D 89 7A      ORL P1, #7AH     ;Word "point"
4F 14 3C      CALL PULSE2      ;
51 04 2C      JMP DCM          ;
;
;Page three of ROM
;
300 1F      ;"Zero"
301 01      ;"One"
302 02      ;"Two"
303 03      ;"Three"
304 04      ;"Four"
305 05      ;"Five"
306 06      ;"Six"
307 07      ;"Seven"
308 08      ;"Eight"
309 09      ;"Nine"
310 0A      ;"Ten"
311 0B      ;"Eleven"
312 0C      ;"Twelve"
313 0D      ;"Thirteen"
314 0E      ;"Fourteen"
315 0F      ;"Fifteen"
316 10      ;"Sixteen"
317 11      ;"Seventeen"
318 12      ;"Eighteen"
319 13      ;"Nineteen"
320 14      ;"Twenty"
330 15      ;"Thirty"
340 16      ;"Forty"

```

### 3.9 Interfacing Displacement Transducers to Speech Processors

Speech processors can be interfaced to almost any analog or digital system to indicate the presence or the magnitude of an input signal. Displacement transducers are not the exception. A displacement transducer is a device capable of sensing the change in position or displacement of an object. A displacement transducer must be sensitive enough to avoid affecting the event being measured. Using the right circuitry, transducers can be interfaced to a speech processor. Certainly, there are many applications where a speech processor will

result in a better system for monitoring displacements instead of using the conventional analog or digital indicators.

Strain gages are typical examples of displacement transducers. Most gages consist of a metal foil or wire bonded to an insulating base, as illustrated in Figure 3.19. The base is cemented to the surface of the object under test, and the gage thus experiences the same strain as the surface. The strain “E” is defined as

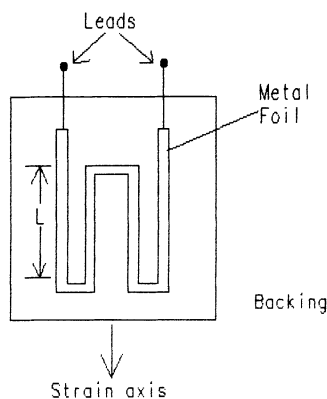
$$E = \Delta L/L$$

where L is the distance between two reference points fixed in the object.

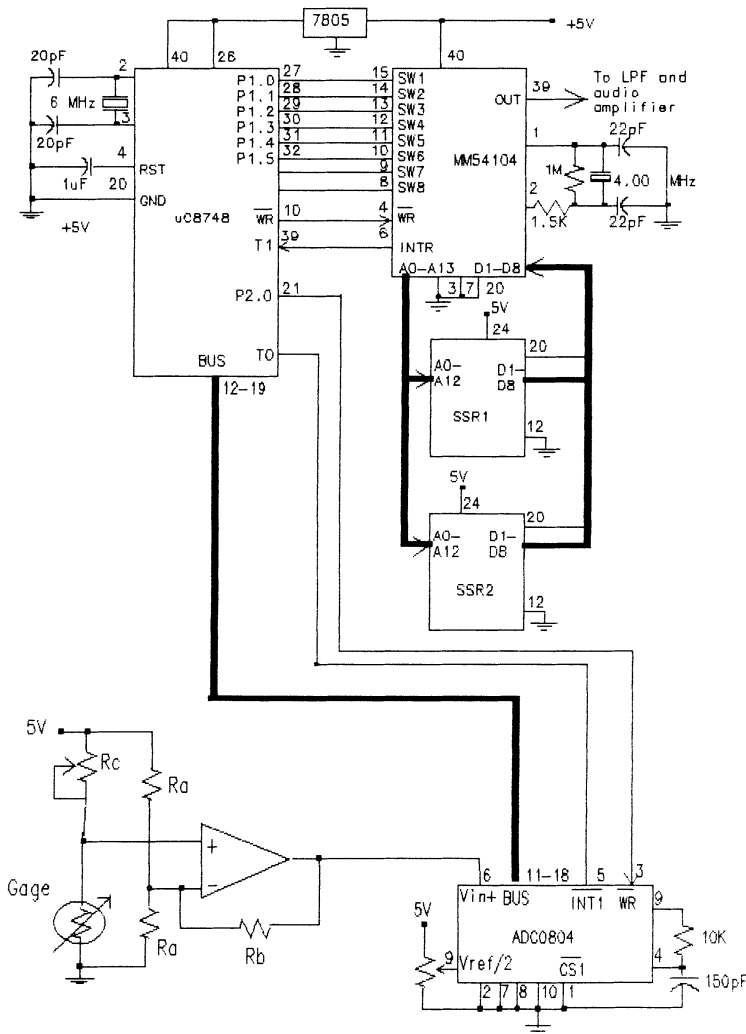
For applications involving small dc signals, a strain gage can be interfaced to a differential op-amp. Figure 3.19 shows this type of circuitry. The output voltage  $V_o$  must be interfaced to an A/D converter if a high resolution is needed. You can even use a set of level comparators to detect several steps of the displacement. Figure 3.20 shows how to interface a strain gage to a microcontroller via an A/D converter. The microcontroller drives the speech processor SPO256-AL2. As you can see in Figure 3.20, the circuit for A/D conversion with the microcontroller and the speech processor is similar to the one presented in Section two of this chapter. You will have to make a few modifications to the software program presented in Section two. For example, you can change the message “volt” to the word “millimeters,” representing a displacement. Now we will see how an optical displacement transducer can be adapted to a speech processor.

### *Digital Displacement Transducer*

A digital displacement transducer is used when relatively large displacements are to be determined. There are two types of transducers that are easy to implement and use: incremental and absolute. Figure 3.21 shows a basic incre-



**Figure 3.19** Strain gage diagram.

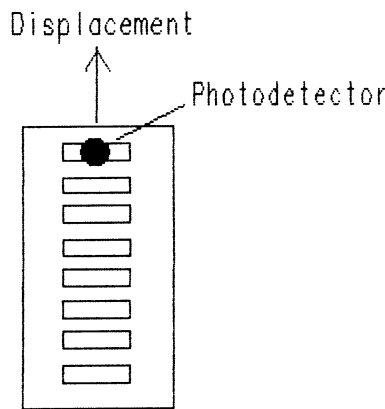


**Figure 3.20** Strain gage interfaced to a speech processor.

mental digital displacement transducer. In this type of transducer, the photo-detector picks up the pulses generated by the slide, which can be perforated (see Figure 3.21)

On the opposite side of the slide, a light source must be emitting a light beam that will be interrupted by the movement of the slide. Each interruption which might occur will generate a pulse that will be routed to a binary counter. This type of displacement transducer has the disadvantage of not being capable of detecting the direction of the displacement, but it has the

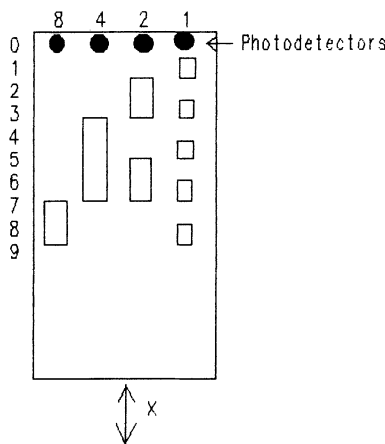




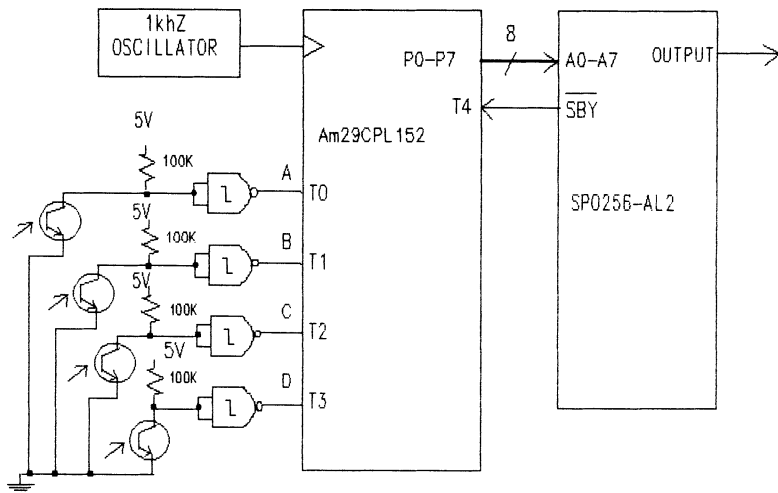
**Figure 3.21** Basic incremental digital displacement transducer.

advantage that the velocity of the movement can be detected; such velocity can be accomplished by detecting the number of pulses that occur in a certain period of time. The magnitude of this measurement will be proportional to the velocity of the displacement.

Figure 3.22 shows a system that contains an absolute digital displacement transducer. In this case the slide that is attached to the object under movement is perforated with a BCD code. Four photodetectors have to be used in this particular form. The advantage of the obtained BCD code is that the code can be sent directly to a BCD-to-seven-segment decoder or to a controller. Obviously, we will prefer the controller option using the FPC Am29CPL151 interfaced directly to a speech processor. Figure 3.23 shows a block diagram



**Figure 3.22** Absolute digital displacement transducer.



**Figure 3.23** Block diagram for a talking displacement system.

of the system that solves the problem completely. The software program for the FPC shown in Figure 3.23 is similar to the one presented in Chapter two, Section 10. In other words, the FPC and the speech processor must be programmed to work as a talking BCD-code meter, although you can make the modifications to the software program according to your real needs.

### 3.10 BCD A/D Converter Interface to SPO256

The advantage of interfacing a BCD A/D converter to a speech processor is that you have two ways of representing a digital reading: a digital display and a voiced reading. In addition, the vocalized reading can be programmed to indicate to the user the procedure to be followed in order to correct a possible malfunction. This feature avoids the time the user would spend trying to find the manual that contains the procedures.

Generally, the use of an A/D converter can be applied to measure analog variables such as pressure, temperature, resistance, capacitance, inductance, and others. Certainly, most of these applications will require a circuit capable of converting these variables to voltage. The input voltage will have to be applied in the correct scale that the A/D converter is requiring in order to obtain the proper output reading.

The circuit shown in Figure 3.24 uses the BCD A/D converter TSC8750 manufactured by Teledyne Semiconductor. The TSC8750 is a 3-1/2 digit A/D converter with parallel BCD output. This converter has a conversion time of 10 ms and contains an input, which can be used to control the data output



used for readings in the range of 5°C to 45°C. In this case you will have to work, for example, within a range of 00.0 to 10.0 V. A resolution of 0.1 V will keep the format of the program in the same way it was developed for the thermometer. If your requirement is for a resolution of 0.01 V, the program will have to be modified so that the microcontroller is able to identify the decimal and centesimal digits. Please take into account that the routine required for this conversion is greatly simplified, because we are working in BCD code. In this way, most of the comparisons required to detect the magnitude of the input voltage are made directly by merely masking the digital reading. Masks will permit you to have the unit, the decimal, or the centesimal value of the input voltage.

If you decide to use the Digitalker system for vocalizing the voltage readings, the software program becomes smaller because the words are already constructed in the speech ROMs and are easier to find, specially when you try to find numbers.

## *Digital Circuits*

### **4.1 4-Bit Magnitude Comparator Calls Out the Results**

Digital comparators use only exclusive OR circuits to compare the respective pairs of bits in each of the two words presented. The outputs of the comparators are routed to a gate that gives a logic one at its output when all the bit pairs are equal and a logic zero when one or more are not. Additional logic can indicate which input is larger than the other.

A basic single-bit magnitude comparator is designed by a truth table that indicates the logic states, as shown in Table 4.1. Here we are comparing the magnitude of two input words A and B.

There are three possible outputs for the magnitude comparator shown in Table 4.1, which we can define as  $f(A < B)$ ,  $f(A = B)$ , and  $f(A > B)$ . By applying minterms to these three outputs, we get the following three equations:

$$\begin{aligned}f(A < B) &= \neg A * B \\f(A = B) &= (\neg A * \neg B) + (A * B) \\f(A > B) &= A * \neg B\end{aligned}$$

For comparison of two words that are larger than one bit, for example, four bits, several of the magnitude comparators available in TTL, CMOS, or HCMOS technology can be used. A good approach is to use the IC 74HC85, a 4-bit magnitude comparator. The 74HC85 provides three fully decoded outputs that indicate which of two 4-bit inputs is larger than the other or if both inputs are equal. It also includes three cascade inputs that permit two or more

**TABLE 4.1**  
Truth Table for a Basic Single-bit  
Magnitude Comparator

A	B	A < B	A = B	A > B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

74HC85s to compare words having eight or more bits. In this case, if two 74HC85s are connected in cascade, the speed of the total comparator is cut to one-half.

It is important to cite that if the reader has access to a PLD programmer, an n-bit magnitude comparator can be designed using a programmable array logic (PAL) chip. This approach is justified when speed is an important factor in the design. A low-cost universal programmer that I strongly recommend is the PLD1100, which is manufactured and distributed by BP Microsystems.

The total number of product terms required for an n-bit comparator is  $2^{n-1}$ . Comparators require a large number of product terms; a PAL that offers 16 product terms is the PAL16L4/R4. A magnitude comparator can also be designed using generic array logic (GAL) devices. The GAL22V10 from Lattice Semiconductor, for example, can be programmed to emulate a 4-bit magnitude comparator. For additional information regarding GAL devices, see the GAL Data Book from Lattice Semiconductor.

We will now examine how to interface the 4-bit magnitude comparator 74HC85 to the speech processor SPO256-AL2 by using the FPC Am29CPL151 (see Figure 4.1). The 74HC85 has a maximum propagation delay from a data input to any output of 21 ns when the ambient temperature is 25°C.

In the circuit presented in Figure 4.1, the three outputs ( $P < Q$ ,  $P = Q$ ,  $P > Q$ ) of the 74HC85 provide the answer of the digital comparison that are sent to inputs T0, T1, and T2 of the FPC Am29CPL151. T4 is used to detect the transient negative pulse caused when the user presses the TEST switch. This way, the circuit will be waiting for T4 to go low. When this happens, the FPC will start reading the logic status of the testable inputs T0, T1, and T2. When the FPC detects the first input in a logic high, it jumps to the routine containing the message that corresponds to the status of the magnitude comparator. The FPC is programmed to make the speech processor SPO256-AL2 announce the messages “P is less than Q,” “P is equal to Q,” and “P is greater than Q.”



SPO256-AL2. Figure 4.2 shows the EPROM-based circuitry required for building a talking 4-bit magnitude comparator. Table 4.3 shows the hex code for the EPROM 2716 that contains the speech data.

The configuration shown in the circuitry of Figure 4.2 is a small modification of the circuits illustrated in Chapter 2 where such circuits were explained

**TABLE 4.2**  
Software for Interfacing a Magnitude Comparator  
to the Speech Processor SPO256-AL2  
by Using the FCP Am29CPL151

```

DEVICE (CPL151)

DEFAULT = 1;

DEFINE      "test inputs"
less = t0
equal = t1
greater = t2
sby = t3
test = t4

"output control bits are given allophone assignments"
"speech data = 59 allophones plus four pauses"
pa2 = 01#h      "      -----      "
pa3 = 02#h      "/zero->|1          28|<- Vcc "
pa4 = 03#h      "  p0<-|2          27|<-clk  "
pa5 = 04#h      "  p1<-|3          26|<-cc   "
oy = 05#h      "  p2<-|4          25|<-t0   "
ay = 06#h      "  p3<-|5   Am29CPL 24|<-t1   "
eh = 07#h      "  p4<-|6       151 23|<-t2   "
kk3 = 08#h      "  p5<-|7          22|<-t3   "
pp = 09#h      "  p6<-|8          21|<-t4   "
jh = 0A#h      "  p7<-|9          20|<-t5   "
nn1 = 0B#h      "  p8<-|10         19|<- /reset"
ih = 0C#h      "  p9<-|11         18|<->p15  "
tt2 = 0D#h      "  p10<-|12        17|<->p14  "
rr1 = 0E#h      "  p11<-|13        16|<->p13  "
ax = 0F#h      "    Gnd-|14        15|<->p12  "
mm = 10#h      "      -----      "
tt1 = 11#h
dh1 = 12#h      iy = 13#h      ey = 14#h      dd1 = 15#h      uw1 = 16#h
ao = 17#h      aa = 18#h      yy2 = 19#h      ae = 1A#h      hh1 = 1B#h
bb1 = 1C#h      th = 1D#h      uh = 1E#h      uw2 = 1F#h      aw = 20#h
dd2 = 21#h      gg3 = 22#h      vv = 23#h      gg1 = 24#h      sh = 25#h
zh = 26#h      rr2 = 27#h      ff = 28#h      kk2 = 29#h      kk1 = 2A#h
zz = 2B#h      ng = 2C#h      ll = 2D#h      ww = 2E#h      xr = 2F#h
wh = 30#h      yy1 = 31#h      ch = 32#h      er1 = 33#h      er2 = 34#h
ow = 35#h      dh2 = 36#h      ss = 37#h      nn2 = 38#h      hh2 = 39#h
or = 3A#h      ar = 3B#h      yr = 3C#h      gg2 = 3D#h      el = 3E#h
bb2 = 3F#h;

DEFAULT-OUTPUT = 0000#h;

TEST-CONDITION = SBY;      "default test condition"

```



```

BEGIN
"wait for test input to go low"
"1"stay:      ,if (test) then goto pl(stay);
"2"      pp,    call pl(read);                "p is ...."
"3"      iy,    call pl(read);
"4"      pa5,   call pl(read);
"5"      iy,    call pl(read);
"6"      ss,    call pl(read);
"7"      ss,    call pl(read);
"8"      pa2,   call pl(read);
"9"                ,if (less) then goto pl(msg1);
"10"                ,if (equal) then goto pl(msg2);
"11"                ,if (greater) then goto pl(msg3);
"12"msg1:ll,    call pl(read);
"13"      eh,    call pl(read);                "LESS..."
"14"      ss,    call pl(read);
"15"      ss,    call pl(read);
"16"      pa5,   call pl(read);
"17"                ,goto pl(msg4);
"18"msg2:ih,    call pl(read);
"19"      kk1,   call pl(read);                "EQUAL TO..."
"20"      uh,    call pl(read);
"21"      ax,    call pl(read);
"22"      ll,    call pl(read);
"23"      pa5,   call pl(read);
"24"      tt1,   call pl(read);
"25"      ow,    call pl(read);
"26"      pa4,   call pl(read);
"27"                ,goto pl(msg5);
"28"msg3:gg1,   call pl(read);                "GREATER"
"29"      rr2,   call pl(read);
"30"      ey,    call pl(read);
"31"      pa2,   call pl(read);
"32"      er1,   call pl(read);
"33"      pa5,   call pl(read);
"34"msg4:dh1,   call pl(read);                "THAN"
"35"      ae,    call pl(read);
"36"      nn1,   call pl(read);
"37"      pa4,   call pl(read);
"38"msg5:kk1,   call pl(read);                "Q"
"39"      iy,    call pl(read);
"40"      uh,    call pl(read);
"41"      pa2,   call pl(read);
"42"                ,goto pl(stay);

"43"read:      ,continue;
"44"styl:      ,if (not sby) then goto pl(styl);
"45"                ,ret;
                .org 63#d
"46"                ,goto pl(stay);
END.

```



**TABLE 4.3**  
EPROM Program for the Circuit of Figure 4.2

Hex Add	Hex Data	Hex Add	Hex Data	Hex Add	Hex Data
20	09	40	09	80	09
21	13	41	13	81	13
22	04	42	04	82	04
23	13	43	13	84	13
24	37	44	37	85	37
25	37	45	37	86	37
26	02	46	02	87	02
27	2D	47	0C	88	24
28	07	48	2A	89	27
29	37	49	1E	8A	14
2A	37	4A	0F	8B	02
2B	04	4B	2D	8C	33
2C	12	4C	04	8D	04
2D	1A	4D	11	8E	12
2E	0B	4E	35	8F	1A
2F	03	4F	03	90	0B
30	2A	50	2A	91	03
31	13	51	13	92	2A
32	1E	52	1E	93	13
33	04	53	04	94	1E
34	40	54	40	95	04
<i>Continued</i>		<i>Continued</i>		96	40

scanning the lower address bits of the EPROM in order to issue all the speech data that the speech processor need to announce a message. The end of a message is also indicated by the EPROM, which asserts a logic high at output O6 in order to clear the flip-flop (1/2 CD4013).

Bear in mind that using the circuit of Figure 4.2, in contrast with the circuit of Figure 4.1, occupies more board space and increases the cost of this system.

**4.2 A Talking Hexadecimal Keyboard Encoder**

A talking keyboard encoder can be an important feature that can be added to most digital control systems. The project presented here will introduce relevant applications in burglar alarms, telephone dialers, motor-speed controllers, and stand-alone PROM programmers. In applications requiring precision (e.g., control of a set of huge ac motors where a small failure or mistake can



**TABLE 4.4**

Software Program for the FPC Am29CPL152 to Make the  
Speech Processor SPO256-AL2 Announce a Key Pressed

```

DEVICE (CPL152)

DEFAULT = 1;

DEFINE "test inputs"
data = t4
sby = t5

"allophones and pauses are given name assignments"

pa2 = 01#h      "      -----"
pa3 = 02#h      "      t6->:1      28;- Vcc "
pa4 = 03#h      "      p0<-:2      27;<-clk "
pa5 = 04#h      "      p1<-:3      26;<-cc  "
oy = 05#h      "      p2<-:4      25;<-t0  "
ay = 06#h      "      p3<-:5      Am29CPL 24;<-t1  "
eh = 07#h      "      p4<-:6      152    23;<-t2  "
kk3 = 08#h      "      p5<-:7      22;<-t3  "
pp = 09#h      "      p6<-:8      21;<-t4  "
jh = 0A#h      "      p7<-:9      20;<-t5  "
nn1 = 0B#h      "      p8<-:10     19;<-/reset"
ih = 0C#h      "      p9<-:11     18;->p15  "
tt2 = 0D#h      "      p10<-:12    17;->p14  "
rr1 = 0E#h      "      p11<-:13    16;->p13  "
ax = 0F#h      "      Gnd->:14     15;->p12  "
mm = 10#h      "      -----"
tt1 = 11#h
dh1 = 12#h      iy = 13#h      ey = 14#h      dd1 = 15#h      uw1 = 16#h
ao = 17#h      aa = 18#h      yy2 = 19#h      ae = 1A#h      hh1 = 1B#h
bb1 = 1C#h      th = 1D#h      uh = 1E#h      uw2 = 1F#h      aw = 20#h
dd2 = 21#h      gg3 = 22#h      vv = 23#h      gg1 = 24#h      sh = 25#h
zh = 26#h      rr2 = 27#h      ff = 28#h      kk2 = 29#h      kk1 = 2A#h
zz = 2B#h      ng = 2C#h      ll = 2D#h      ww = 2E#h      xr = 2F#h
wh = 30#h      yy1 = 31#h      ch = 32#h      er1 = 33#h      er2 = 34#h
ow = 35#h      dh2 = 36#h      ss = 37#h      nn2 = 38#h      hh2 = 39#h
or = 3A#h      ar = 3B#h      yr = 3C#h      gg2 = 3D#h      el = 3E#h
bb2 = 3F#h;

DEFAULT-OUTPUT = 0000#h;

TEST-CONDITION = SBY; "/STANDBY is the default test condition"

BEGIN
"0"      ,goto pl(zero);
"1"      ,goto pl(one);
"2"      ,goto pl(two);
"3"      ,goto pl(thre);
"4"      ,goto pl(four);
"5"      ,goto pl(five);
"6"      ,goto pl(six);
"7"      ,goto pl(svn);
"8"      ,goto pl(eit);

```

```

"9"          ,goto pl(nin);
"10"         ,goto pl(ltra);
"11"         ,goto pl(ltrb);
"12"         ,goto pl(ltrc);
"13"         ,goto pl(ltrd);
"14"         ,goto pl(ltre);
"15"         ,goto pl(ltrf);
"16"zero:zz,  call pl(read); "ZERO"
"17"        yr,  call pl(read);
"18"        ow,  call pl(read);
"19"        pa2, call pl(read);
"20"         ,goto pl(stay);
"21"one:ww,   call pl(read); "ONE"
"22"        ax,  call pl(read);
"23"        ax,  call pl(read);
"24"        nn1, call pl(read);
"25"        pa2, call pl(read);
"26"         ,goto pl(stay);
"27"two:tt2,  call pl(read); "TWO"
"28"        uw2, call pl(read);
"29"        pa2, call pl(read);
"30"         ,goto pl(stay);
"31"thre:th,  call pl(read); "THREE"
"32"        rr1, call pl(read);
"33"        iy,  call pl(read);
"34"        pa2, call pl(read);
"35"         ,goto pl(stay);
"36"four:ff,  call pl(read); "FOUR"
"37"        ff,  call pl(read);
"38"        or,  call pl(read);
"39"        pa2, call pl(read);
"40"         ,goto pl(stay);
"41"five:ff,  call pl(read); "FIVE"
"42"        ff,  call pl(read);
"43"        ay,  call pl(read);
"44"        vv,  call pl(read);
"45"        pa2, call pl(read);
"46"         ,goto pl(stay);
"47"six:ss,   call pl(read); "SIX"
"48"        ss,  call pl(read);
"49"        ih,  call pl(read);
"50"        ih,  call pl(read);
"51"        pa3, call pl(read);
"52"        kk2, call pl(read);
"53"        ss,  call pl(read);
"54"        pa2, call pl(read);
"55"         ,goto pl(stay);
"56"svn:ss,   call pl(read); "SEVEN"
"57"        ss,  call pl(read);
"58"        eh,  call pl(read);
"59"        eh,  call pl(read);
"60"        vv,  call pl(read);
"61"        eh,  call pl(read);
"62"        nn1, call pl(read);

```

```

"63"    pa2,    call pl(read);
"64"    ,goto pl(stay);
"65"eit:ey,    call pl(read);    "EIGHT"
"66"    pa3,    call pl(read);
"67"    tt2,    call pl(read);
"68"    pa2,    call pl(read);
"69"    ,goto pl(stay);
"70"nin:nn2,    call pl(read);    "NINE"
"71"    aa,    call pl(read);
"72"    ay,    call pl(read);
"73"    nn1,    call pl(read);
"74"    pa2,    call pl(read);
"75"    ,goto pl(stay);
"76"ltra:ey,    call pl(read);    "A"
"77"    pa2,    call pl(read);
"78"    ,goto pl(stay);
"79" ltrb:bb2,    call pl(read);    "B"
"80"    iy,    call pl(read);
"81"    pa2,    call pl(read);
"82"    ,goto pl(stay);
"83" ltrc:ss,    call pl(read);    "C"
"84"    ss,    call pl(read);
"85"    iy,    call pl(read);
"86"    pa2,    call pl(read);
"87" ltrd:dd2,    goto pl(stay);    "D"
"88"    iy,    call pl(read);
"89"    pa2,    call pl(read);
"90"    ,goto pl(stay);
"91" ltre:iy,    call pl(read);    "E"
"92"    pa2,    call pl(read);
"93"    ,goto pl(stay);
"94" ltrf:eh,    call pl(read);    "F"
"95"    eh,    call pl(read);
"96"    ff,    call pl(read);
"97"    ff,    call pl(read);
"98"    pa2,    call pl(read);
"99"    ,goto pl(stay);
"----- wait for data input to go high"
"100"stay:    ,if (not data) then goto pl(stay);
"101"    ,goto tm(001111#b);    "Go to address: T3,T2,T1,T0"
"subroutine for reading the standby status of the speech processor"
"102"read:    ,continue;
"103"sty1:    ,if (not sby) then goto pl(sty1);    "reading SBY"
"104"    ,ret;
    .org 127#d
"105"    ,goto pl(stay);
END.

```

entry has been made. When the entered key is released, the DATA AVAILABLE output returns to a low level.

The software program that reads the 16-key pad and makes the speech processor say the pressed key is shown in Table 4.4. The routine for this program must be able to detect when a key has been pressed and then to proceed driving the speech processor. When the circuit is first turned on, a power-up reset pulse is applied to the reset input of the FPC (see Figure 4.3); therefore, we have to apply the same reset pulse in the software program. This pulse is achieved with the instruction “org 127#d.” Once this reset pulse has been applied, the program jumps to line 105, which contains the instruction “goto pl(stay).” This instruction makes the program jump to the label “stay” that is located in line 100.

The instruction located in line 100 “if (not data) then goto pl (stay)” keeps the program reading the logic status of the input presented in T4. As Figure 4.3 shows, the output DATA AVAILABLE coming from the encoder (74C922) is routed to the testable input T4. The instruction in line 100 will be waiting for the data input (T4) to go high; consequently, if the input T4 is high, the program jumps to the next instruction (line 101). Instruction “goto tm(001111#b);” located in line 101, makes the program jump to the address indicated by the four testable inputs T0, T1, T2, and T3; it means that the hex code presented in these inputs will make the program jump to the addresses within the range of zero to fifteen. The instructions located within this range are used to indicate to the program the location of the set of allophones containing the word that must be vocalized. When the message corresponding to the pressed key has been issued, the program jumps directly to line 100 in order to continue monitoring the status of the data input (DA).

To make sure the program of Table 4.4 is performing the instructions correctly, we will write a short file called “SIMKBD.” This file (see Table 4.5) contains only seven vectors that must be executed by the program SIM14X, version 5.0 from Advanced Micro Devices. The program SIM14X executes the simulation by reading the JEDEC file that corresponds to the program presented in Table 4.4.

Table 4.6 is the output file generated by the program SIM14X. This file presents all the logical status of the inputs, outputs, and registers. In this manner, you can check the outputs related to the inputs that you have already programmed. If a mismatch occurs, it will be indicated by the symbol “?” In this case, this file shows correctly all the outputs, which means that our program is working perfectly. You can also augment the size of the test file presented in Table 4.5 in order to test different points or steps of the program.

With the project presented here, we will be able to add a key pad to the designs that require a digital input from the user. We will use the program presented here as a simple subroutine whenever we need it. Remember that this routine can be added to programs working with the FPC Am29CPL152 or Am29CPL154.





**TABLE 4.6**  
Output File Generated by the Simulator SIM14X

V0000	INPUT	OUTPUT (Expansion disabled)
/		
r		
Pin c e		
Name: l s t t t t t t		
p p p p p p p p		
k e 5 4 3 2 1 0		
7 6 5 4 3 2 1 0		
R/NR: R R R R R R R		
Pin#: 27 19 20 21 22 23 24 25		
9 8 7 6 5 4 3 2		
Vect: C 0 X X X X X X		
X X X X X X X X		
Comp:		
X X X X X X X X		
CREG = ****, PC = ****, EQ = *		
STK [ 0] **** [ 1] ****		
Pipeline : ***		
Mnemonics: ***		
Current PL contents are undefined		
-----		
V0001	INPUT	OUTPUT (Expansion disabled)
/		
r		
Pin c e		
Name: l s t t t t t t		
p p p p p p p p		
k e 5 4 3 2 1 0		
7 6 5 4 3 2 1 0		
R/NR: R R R R R R R		
Pin#: 27 19 20 21 22 23 24 25		
9 8 7 6 5 4 3 2		
Vect: C 1 1 0 0 0 0 0		
X X X X X X X X		
Comp:		
X X X X X X X X		
CREG = ****, PC = ****, EQ = *		
STK [ 0] **** [ 1] ****		

```
Pipeline :    ***
Mnemonics:    ***
Current PL contents are undefined
-----
V0002          INPUT          |          OUTPUT  (Expansion disabled)
      /                      |
      r                      |
Pin   c   e                      |
Name: l s t t t t t t      |
      p p p p p p p p      |
      k e 5 4 3 2 1 0      |
      7 6 5 4 3 2 1 0      |
R/NR:   R R R R R R R      |
Pin#: 27 19 20 21 22 23 24 25 |
      9 8 7 6 5 4 3 2      |
Vect: C 1 1 0 0 0 0 0      |
      L L L L L L L L      |
Comp:                      |
      L L L L L L L L      |
CREG = ****,  PC = 7F#H,  EQ = 0
STK [ 0] ****  [ 1] ****
Pipeline :    OE   OPCODE   POL   TEST   DATA   OUTPUTS
              1    19#H     0     05#H   64#H     0000000000000000#B
(0000#H)
Mnemonics:    GOTOPL, IF (cond) THEN GOTO PL(data)
Condition PASS, TEST t5 = 1, VALUE = 1
Current PL contents loaded from ROM address 127 (07F#H)
-----
V0003          INPUT          |          OUTPUT  (Expansion disabled)
      /                      |
      r                      |
Pin   c   e                      |
Name: l s t t t t t t      |
      p p p p p p p p      |
      k e 5 4 3 2 1 0      |
      7 6 5 4 3 2 1 0      |
R/NR:   R R R R R R R      |
Pin#: 27 19 20 21 22 23 24 25 |
      9 8 7 6 5 4 3 2      |
Vect: C 1 1 1 0 1 1 1      |
      L L L L L L L L      |
```

```

Comp:                                     ;
  L L L L L L L L
CREG = ****,  PC = 64#H,  EQ = 0
STK [ 0] ****  [ 1] ****

Pipeline :   OE      OPCODE      POL      TEST      DATA      OUTPUTS
            1      19#H      1      04#H      64#H      0000000000000000#B

(0000#H)
Mnemonics:   GOTOPL, IF (cond) THEN GOTO PL(data)
Condition PASS, TEST t4 = 0, VALUE = 0
Current PL contents loaded from ROM address 100 (064#H)

-----
V0004          INPUT                      ;          OUTPUT  (Expansion disabled)
      /                      ;
      r                      ;

Pin   c   e                      ;

Name: l   s   t   t   t   t   t   t      ;
      p   p   p   p   p   p   p   p
      k   e   5   4   3   2   1   0      ; 7 6 5 4 3 2 1 0
R/NR:   R   R   R   R   R   R   R   R      ;

Pin#: 27 19 20 21 22 23 24 25            ;
      9 8 7 6 5 4 3 2
Vect: C   1   1   0   0   1   1   1      ;
      L   L   L   L   L   L   L   L
Comp:                                     ;
      L   L   L   L   L   L   L   L
CREG = ****,  PC = 64#H,  EQ = 0
STK [ 0] ****  [ 1] ****

Pipeline :   OE      OPCODE      POL      TEST      DATA      OUTPUTS
            1      19#H      1      04#H      64#H      0000000000000000#B

(0000#H)
Mnemonics:   GOTOPL, IF (cond) THEN GOTO PL(data)
Condition FAIL, TEST t4 = 0, VALUE = 1
Current PL contents loaded from ROM address 100 (064#H)

-----
V0005          INPUT                      ;          OUTPUT  (Expansion disabled)
      /                      ;
      r                      ;

Pin   c   e                      ;

Name: l   s   t   t   t   t   t   t      ;
      p   p   p   p   p   p   p   p
      k   e   5   4   3   2   1   0      ; 7 6 5 4 3 2 1 0
R/NR:   R   R   R   R   R   R   R   R      ;

```

```
Pin#: 27 19 20 21 22 23 24 25      |
  9 8 7 6 5 4 3 2
Vect: C 1 1 0 0 0 0 0      |
  L L L L L L L L
Comp:                                | L L L L L L L L
CREG = ****, PC = 65#H, EQ = 0
STK [ 0] **** [ 1] ****
Pipeline :   OE   OPCODE   POL   TEST   DATA   OUTPUTS
            1   1F#H     0    05#H   0F#H   0000000000000000#B
(0000#H)
Mnemonics:  GOTOTM, IF (cond) THEN GOTO TM(data)
Condition PASS, TEST t5 = 1, VALUE = 1, T[6:0] = 27#H, T*M = 07#H
Current PL contents loaded from ROM address 101 (065#H)
-----
V0006          INPUT      |      OUTPUT (Expansion disabled)
      /              |
      r              |
Pin   c   e          |
Name: l   s   t   t   t   t   t   t      |
      p   p   p   p   p   p   p   p      |
          k   e   5   4   3   2   1   0      |
          7   6   5   4   3   2   1   0      |
R/NR:   R   R   R   R   R   R   R   R      |
Pin#: 27 19 20 21 22 23 24 25      |
  9 8 7 6 5 4 3 2
Vect: C 1 1 0 0 0 0 0      |
  L L L L L L L L
Comp:                                |
  L L L L L L L L
CREG = ****, PC = 07#H, EQ = 0
STK [ 0] **** [ 1] ****
Pipeline :   OE   OPCODE   POL   TEST   DATA   OUTPUTS
            1   19#H     0    05#H   38#H   0000000000000000#B
(0000#H)
Mnemonics:  GOTOPL, IF (cond) THEN GOTO PL(data)
Condition PASS, TEST t5 = 1, VALUE = 1
Current PL contents loaded from ROM address 7 (007#H)
-----
V0007          INPUT      |      OUTPUT (Expansion disabled)
      /              |
      r              |
Pin   c   e          |
```

```

Name: l s t t t t t t      ;
      p p p p p p p p
          k e 5 4 3 2 1 0    ;
      7 6 5 4 3 2 1 0
R/NR:   R R R R R R R      ;

Pin#: 27 19 20 21 22 23 24 25 ;
      9 8 7 6 5 4 3 2
Vect: C 1 1 0 0 0 0 0      ;
      L L H H L H H H
Comp:
      L L H H L H H H
CREG = ****, PC = 38#H, EQ = 0
STK [ 0] **** [ 1] ****
Pipeline : OE  OPCODE  POL  TEST  DATA  OUTPUTS
           1   1C#H    0    05#H  66#H  0000000000110111#B
(0037#H)
Mnemonics: CALPL, IF (cond) THEN CALL PL(data)
Condition PASS, TEST t5 = 1, VALUE = 1
Current PL contents loaded from ROM address 56 (038#H)
-----
Simulation completed    0 simulation error(s) found

```

### 4.3 Designing a Talking Semaphore

A traffic controller can be enhanced by integrating a vocal warning system which can protect pedestrians from accidents caused by an elapsed time. Some pedestrians perceive the WALK signal but they do not notice when the WALK signal starts flashing so that they are not in the middle of the street when the oncoming cars have the green light.

The design presented here will be applied to a simple traffic controller where two one-way streets are considered for illustrative purposes. Figure 4.4 shows the traffic intersection with two one-way streets: one in direction 1 and the other in direction 2. Each direction has a set of five light signals consisting of green, yellow, red, pass, and don't pass. The lights assigned for direction 1 are named GREEN1, YELLOW1, RED1, PASS1, and DON'T PASS1. In the same manner, the set of lights assigned for direction 2 are named GREEN2, YELLOW2, RED2, PASS2, and DON'T PASS2. The signals DON'T PASS1 and DON'T PASS2 are generated by using two external inverters, as shown in Figure 4.5. Two sensor switches, SW1 and SW2, will be used for detecting a request made by a pedestrian who desires to cross the street. The selection of the street that the pedestrian wants to cross is made by pressing only the respective switch SW1 or SW2. Also, each direction has a sensor that provides an active high signal (SEN1, SEN2) that indicates the presence of a vehicle.



The traffic controller signals will be generated by the FPC Am29CPL152. Thanks to the 16 output lines of the FPC, it will also be controlling the speech processor that will give the messages to the pedestrians. In this case, we will give names to the streets so that the vocal message will be able to indicate the street that the pedestrian has to cross. For this purpose, the street with direction 1 will be named "MAIN Street," and the street with direction 2 will be named "FIRST Street."

The truth table indicating all the functions of this traffic controller, including the type of vocal warning messages, is shown in Table 4.7. In normal operation the traffic controller will give equal periods of green signals. At the end of each period of green signal, the controller will ask for the status of the input switches SW1 and SW2, and the vehicle sensors SEN1 and SEN2 in order to decide the length of time for the next cycle.

The operation of the software program shown in Table 4.8 is as follows. When the circuit is first turned on, a software reset pulse is applied. Then the program performs the next instruction "go to pl(dir1)," which makes the program to go to the address labeled as "dir1" that is located in line 16. Lines 16 to 22 contain a routine that activates the output lines named "grn1" and "red2." These names, separated by a comma from the instruction, generate a

**TABLE 4.7**  
Table for Traffic Flow Direction

SW1	SW2	SEN1	SEN2	Output
L	L	L	L	Allow traffic in direction 1 (MAIN St)
L	L	L	H	Allow traffic in direction 2 (FIRST St)
L	L	H	L	Allow traffic in direction 1 (MAIN St)
L	L	H	H	Cycle with equal durations in both directions
L	H	L	L	Allow traffic in direction 2 (FIRST St)
L	H	L	H	Allow traffic in direction 2 (FIRST St)
L	H	H	L	Allow traffic in direction 1 (MAIN St)
L	H	H	H	Cycle with equal durations in both directions
H	L	L	L	Allow traffic in direction 1 (MAIN St)
H	L	L	H	Cycle with equal durations in both directions
H	L	H	L	Allow traffic in direction 1 (MAIN St)
H	L	H	H	Cycle with equal durations in both directions
H	H	L	L	Cycle with equal durations in both directions
H	H	L	H	Cycle with equal durations in both directions
H	H	H	L	Cycle with equal durations in both directions
H	H	H	H	Cycle with equal durations in both directions



**TABLE 4.8**

Software Program for the FPC Am29CPL152 to Make the Traffic Controller

```

DEVICE (CPL152)

```

```

DEFAULT = 1;

```

```

DEFINE      "test inputs"

```

```

sw1 = t0

```

```

sw2 = t1

```

```

sen1 = t2

```

```

sen2 = t3

```

```

sby = t5

```

```

equal = eq

```

```

      "allophones and pauses are given name assignments"

```

```

pa2 = 01#h      "      -----"
pa3 = 02#h      "      t6->:1          28!:- Vcc "
pa4 = 03#h      "      p0<-:2          27!<-:clk "
pa5 = 04#h      "      p1<-:3          26!<-:cc  "
oy  = 05#h      "      p2<-:4          25!<-:t0  "
ay  = 06#h      "      p3<-:5      Am29CPL 24!<-:t1  "
eh  = 07#h      "      p4<-:6      152    23!<-:t2  "
kk3 = 08#h      "      p5<-:7          22!<-:t3  "
pp  = 09#h      "      p6<-:8          21!<-:t4  "
jh  = 0A#h      "      p7<-:9          20!<-:t5  "
nn1 = 0B#h      "      p8<-:10         19!<-:/reset"
ih  = 0C#h      "      p9<-:11         18!<->p15  "
tt2 = 0D#h      "      p10<-:12        17!<->p14  "
rr1 = 0E#h      "      p11<-:13        16!<->p13  "
ax  = 0F#h      "      Gnd->:14        15!<->p12  "
mm  = 10#h      "      -----"

```

```

tt1 = 11#h
dh1 = 12#h      iy = 13#h      ey = 14#h      dd1 = 15#h      uw1 = 16#h
ao = 17#h      aa = 18#h      yy2 = 19#h      ae = 1A#h      hh1 = 1B#h
bb1 = 1C#h      th = 1D#h      uh = 1E#h      uw2 = 1F#h      aw = 20#h
dd2 = 21#h      gg3 = 22#h      vv = 23#h      gg1 = 24#h      sh = 25#h
zh = 26#h      rr2 = 27#h      ff = 28#h      kk2 = 29#h      kk1 = 2A#h
zz = 2B#h      ng = 2C#h      ll = 2D#h      ww = 2E#h      xr = 2F#h
wh = 30#h      yy1 = 31#h      ch = 32#h      er1 = 33#h      er2 = 34#h
ow = 35#h      dh2 = 36#h      ss = 37#h      nn2 = 38#h      hh2 = 39#h
or = 3A#h      ar = 3B#h      yr = 3C#h      gg2 = 3D#h      el = 3E#h
bb2 = 3F#h

```

```

grn1 = 40#h      yell1 = 80#h      red1 = 100#h

```

```

grn2 = 200#h      yell2 = 400#h      red2 = 800#h

```

```

pass1 = 1000#h      pass2 = 2000#h      latch= 4000#h      rst = 8000#h;

```

```

DEFAULT-OUTPUT = 0000#h;

```

```

TEST-CONDITION = SBY; "/STANDBY is the default test condition"

```

```

BEGIN

```

```

"0"      ,goto pl(dir1);

```

```

"1"      ,goto pl(dir2);

```

```

"2"      ,goto pl(dir1);

```

```

"3"          ,goto pl(both);
"4"          ,goto pl(dir2);
"5"          ,goto pl(dir2);
"6"          ,goto pl(dir1);
"7"          ,goto pl(both);
"8"          ,goto pl(dir1);
"9"          ,goto pl(both);
"10"         ,goto pl(dir1);
"11"         ,goto pl(both);
"12"         ,goto pl(both);
"13"         ,goto pl(both);
"14"         ,goto pl(both);
"15"         ,goto pl(both);
"-----"
"16"dir1:grn1+red2,      load pl(100);      "20 seconds"
"17"rick:grn1+red2+pa5,  call pl(delay);
"18"      grn1+red2,      while (creg <> 0) loop to pl(rick);
"19"      grn1+red2,      load tm(0F#h);
"20"      grn1+red2,      cmp tm(0F#h) to pl(00#h);
"21"      grn1+red2,      if (equal) then goto pl(dir1);
"22"      grn1+red2,      goto tm(0F#h);
"-----"
"23"dir2:yell1+red2+rst, load pl(20);      "4 seconds"
"24"styl:yell1+red2+pa5, call pl(delay1);
"25"      yell1+red2,      while (creg <> 0) loop to pl(styl);
"26"      grn2+red1+pass2, call pl(msg1);
"27"      grn2+red1+pass2, load pl(100);      "20 seconds"
"28"styy:grn2+red1+pass2+pa5, call pl(delay2);
"29"      grn2+red1+pass2, while (creg <> 0) loop to pl(styy);
"30"      grn2+red1+pass2+ss, call pl(read1);  "STOP"
"31"      grn2+red1+pass2+ss, call pl(read1);
"32"      grn2+red1+pass2+tt2, call pl(read1);
"33"      grn2+red1+pass2+ax, call pl(read1);
"34"      grn2+red1+pass2+pp, call pl(read1);
"35"      grn2+red1+pass2+pa4, call pl(read1);
"36"      grn2+red1+pass2,  load pl(25);      "5 seconds"
"37"sty2:grn2+red1+pass2+pa5, call pl(delay2);
"38"      grn2+red1+pass2,  while (creg <> 0) loop to pl(sty2);
"39"      yel2+red1,        load pl(20);      "4 seconds of Yellow2"
"40"sty3:yel2+red1+pa5,    call pl(delay3);
"41"      yel2+red1,        while (creg <> 0) loop to pl(sty3);
"42"      yel2+red1,        goto pl(dir1);  "goto verify sensors"
"-----"
"43"both:yell1+red2+rst,  load pl(20);      "4 seconds of Yellow1"
"44"sty4:yell1+red2+pa5,  call pl(delay1);
"45"      yell1+red2,      while (creg <> 0) loop to pl(sty4);
"46"      grn2+red1+pass2, call pl(msg1);
"47"      grn2+red1+pass2, load pl(125);      "25 seconds of Green2"
"48"sty5:grn2+red1+pass2+pa5, call pl(delay2);
"49"      grn2+red1+pass2, while (creg <> 0) loop to pl(sty5);
"50"      grn2+red1+ss,    call pl(delay5);  "STOP"
"51"      grn2+red1+ss,    call pl(delay5);
"52"      grn2+red1+tt2,   call pl(delay5);
"53"      grn2+red1+ax,    call pl(delay5);

```

```

"54"      grn2+red1+pp,      call pl(delay5);
"55"      grn2+red1+pa4,     call pl(delay5);
"56"      yel2+red1,         load pl(20);      "4 seconds"
"57"sty6: yel2+red1+pa5,     call pl(delay3);
"58"      yel2+red1,         while (creg <> 0) loop to pl(sty6);
"59"      grn1+red2+pass1,    call pl(msg2);
"60"      grn1+red2+pass1,    load pl(125);      "25 seconds"
"61"sty7: grn1+red2+pass1+pa5, call pl(delay4);
"62"      grn1+red2+pass1,    while (creg <> 0) loop to pl(sty7);
"63"      grn1+red2,         goto tm(0F#h);
"-----"
"64"delay: grn1+red2,        continue;
"65"styz:  grn1+red2,        if (not sby) then goto pl(styz);
"66"      grn1+red2,         ret;
"-----"
"subroutine for reading the standby status of the speech processor"
"67"read1: grn2+red1+pass2,   continue;
"68"paty:  grn2+red1+pass2,   if (not sby) then goto pl(paty);
"69"      grn2+red1+pass2,    ret;
"-----"
"subroutine for reading the standby status of the speech processor"
"70"read2: grn1+red2+pass1,   continue;
"71"styb:  grn1+red2+pass1,   if (not sby) then goto pl(styb);
"72"      grn1+red2+pass1,    ret;
"-----"
"73"delay1: yel1+red2,        continue;
"74"styc:  yel1+red2,         if (not sby) then goto pl(styd);
"75"      yel1+red2,         ret;
"-----"
"76"delay2: grn2+red1+pass2,   continue;
"77"styd:  grn2+red1+pass2,   if (not sby) then goto pl(styd);
"78"      grn2+red1+pass2,    ret;
"-----"
"79"delay3: yel2+red1,        continue;
"80"stye:  yel2+red1,         if (not sby) then goto pl(stye);
"81"      yel2+red1,         ret;
"-----"
"82"delay4: grn1+red2+pass1,   continue;
"83"styf:  grn1+red2+pass1,   if (not sby) then goto pl(styf);
"84"      grn1+red2+pass1,    ret;
"-----"
"85"delay5: grn2+red1+ss,      continue;
"86"styg   :grn2+red1+ss,      if (not sby) then goto pl(styg);
"87"      grn2+red1+ss,      ret;
"-----"
"***** SUBROUTINE FOR THE MESSAGE: PASS THE MAIN STREET"
"88"msg1:  grn2+red1+pass2+pp, call pl(read1);
"89"      grn2+red1+pass2+ax,   call pl(read1);
"90"      grn2+red1+pass2+ss,   call pl(read1);
"91"      grn2+red1+pass2+ss,   call pl(read1);
"92"      grn2+red1+pass2+pa4,  call pl(read1);
"93"      grn2+red1+pass2+dh1,  call pl(read1);

```

```

"94"      grn2+red1+pass2+ax,    call pl(read1);
"95"      grn2+red1+pass2+pa4,   call pl(read1);
"96"      grn2+red1+pass2+mm,    call pl(read1);
"97"      grn2+red1+pass2+ey,    call pl(read1);
"98"      grn2+red1+pass2+nn1,   call pl(read1);
"99"      grn2+red1+pass2+pa4,   call pl(read1);
"100"     grn2+red1+pass2+ss,     call pl(read1);
"101"     grn2+red1+pass2+tt1,   call pl(read1);
"102"     grn2+red1+pass2+rr1,   call pl(read1);
"103"     grn2+red1+pass2+ih,    call pl(read1);
"104"     grn2+red1+pass2+tt2,   call pl(read1);
"105"     grn2+red1+pass2+pa4,   call pl(read1);
"106"     grn2+red1+pass2,       ret;
"-----"
"***** SUBROUTINE FOR THE MESSAGE: PASS THE FIRST STREET"
"107"msg2: grn1+red2+pass1+pp,   call pl(read2);
"108"      grn1+red2+pass1+ax,   call pl(read2);
"109"      grn1+red2+pass1+ss,   call pl(read2);
"110"      grn1+red2+pass1+ss,   call pl(read2);
"111"      grn1+red2+pass1+pa4,   call pl(read2);
"112"      grn1+red2+pass1+dh1,   call pl(read2);
"113"      grn1+red2+pass1+ax,   call pl(read2);
"110"      grn1+red2+pass1+pa4,   call pl(read2);
"111"      grn1+red2+pass1+ff,   call pl(read2);
"112"      grn1+red2+pass1+er2,   call pl(read2);
"113"      grn1+red2+pass1+ss,   call pl(read2);
"114"      grn1+red2+pass1+tt2,   call pl(read2);
"115"      grn1+red2+pass1+pa4,   call pl(read2);
"116"      grn1+red2+pass1+ss,   call pl(read2);
"117"      grn1+red2+pass1+tt1,   call pl(read2);
"118"      grn1+red2+pass1+rr1,   call pl(read2);
"119"      grn1+red2+pass1+ih,   call pl(read2);
"120"      grn1+red2+pass1+tt2,   call pl(read2);
"121"      grn1+red2+pass1+pa4,   call pl(read2);
"122"      grn1+red2+pass1,       ret;
"-----"

.org 127#d
"122"      ,goto pl(dir1);
END.

```

logic high to the outputs that activate the signals GREEN1 and RED2. The instruction “load pl(100)” in line 16 loads the counter register (CREG) with the decimal value “100.” The next instruction “call pl(delay)” then proceeds to call subroutine “delay” in order to load the speech processor with pause pa5. Pause pa5 is used to perform 200 ms silence pauses. Also, subroutine “delay” waits for the speech processor until it finishes executing the 200 ms pause. When the 200 ms pause ends, the program performs a return to line 18 where the instruction “while (creg <> 0) loop to pl(sty1)” is executed. The

instruction in line 18 decrements the counter (CREG) and tests its contents against zero. If the contents differ from zero, the program loops to the instruction labeled “rick,” and the process of calling the subroutine “delay” and decrementing the counter CREG is repeated until the CREG is equal to zero. Consequently, the three instructions in lines 16 to 18 cause a 20 s delay because the loop of 200 ms is executed 100 times. Notice that by using the longest pause contained in the speech processor, a long timing interval can be generated. But consider that the highest possible value that the counter CREG can handle is 127.

Lines 19 to 22 are used to detect if any of the four input sensors has been activated. These sensors are activated by comparing the value of the four input lines T0–T3 against zero. If any of these lines differs from zero, the program jumps to one of the instructions located in lines 0 to 15. It is the value of the four inputs T0 to T3 that indicates the exact location of the jump. In this manner, lines 0 to 15 contain the label where the program has to jump in order to execute the timing routines that will control the output signals. In fact, lines 0 to 16 are used as a selector for the different types of inputs presented when the sensors are activated. According to Table 4.6, there are two timing routines that the program must execute: “dir2” and “both.”

For example, if sensor2 is activated, the program will detect its activation when the instructions 19 to 22 are executed. This will cause a jump to line 1, which in turn makes the program jump to the routine named “dir2.” Now, routine “dir2” will start turning on the lamps named YELLOW1 and RED2 and also will cause a reset pulse to the four external flip-flops. The timing interval for the lamps YELLOW1 and RED2 will be 4 s. The next step is to perform the instruction in line 26, which calls subroutine “msg1” in order to issue the first vocal message to the pedestrian(s). The first message issued is “Pass Main Street.” This message is used to indicate to the pedestrian(s) that now he(they) is(are) allowed to pass Main Street. Also, output signal “PASS2” will be activated for 20 s. Thus, the pedestrian(s) will have two modes of announcement: visual and audible. Five seconds before the PASS2 signal goes off, the speech processor will announce the following message: “STOP.” After the message “STOP” has been issued, signal GREEN2 will remain activated five more seconds to give the pedestrian(s) time to reach the other side of Main Street. There is still another interval of security given by lamp YELLOW2, because this lamp will be activated for 4 s. The last instruction of routine “dir2” makes the program jump to routine “dir1” which is the normal state of this traffic controller; that is, lamps GREEN1 and RED2 are turned on.

Routine “both” makes essentially the same process of routine “dir2.” The only difference is that it allows traffic to flow in both directions. When routine “both” ends, the program will jump to the routine indicated by the four testable inputs T0–T3.

As can be seen, the timing intervals can be changed by augmenting or repeating the set of instructions that perform the respective delay. In the same manner, if more outputs are needed for controlling a larger number of lamps, a second FPC can be added in parallel, which will share the same clock input as the first FPC.

## 4.4 How to Design a Talking Clock

A talking clock can offer different applications, depending upon the area of installation. For example, in a waiting room, it can be programmed to announce the time every 30 minutes. In a factory, a talking clock can be used to report entrance and exit times of the workers. Talking clocks are useful for people with visual handicaps. The talking clock presented here is programmed to show the time in hours and minutes in an LED display and to tell the time every hour. To report the time every hour, Table 4.9 presents the complete set of required messages. Notice that all the hours within the range of 12:00 PM to 12:00 AM are reported.

The digital clock will be built around the chip MM5387 manufactured by National Semiconductor. This chip is capable of driving four LED digits: M1, M10, H1, and H10. The seven-segment outputs are compatible with common-anode LED displays. Figure 4.6 shows the complete circuit for the talking

**TABLE 4.9**  
Messages for Programming the Talking Clock

Hour	Message
12:00	It is twelve o'clock AM/PM
1:00	It is one o'clock AM/PM
2:00	It is two o'clock AM/PM
3:00	It is three o'clock AM/PM
4:00	It is four o'clock AM/PM
5:00	It is five o'clock AM/PM
6:00	It is six o'clock AM/PM
7:00	It is seven o'clock AM/PM
8:00	It is eight o'clock AM/PM
9:00	It is nine o'clock AM/PM
10:00	It is ten o'clock AM/PM
11:00	It is eleven o'clock AM/PM
12:00	It is twelve o'clock AM/PM

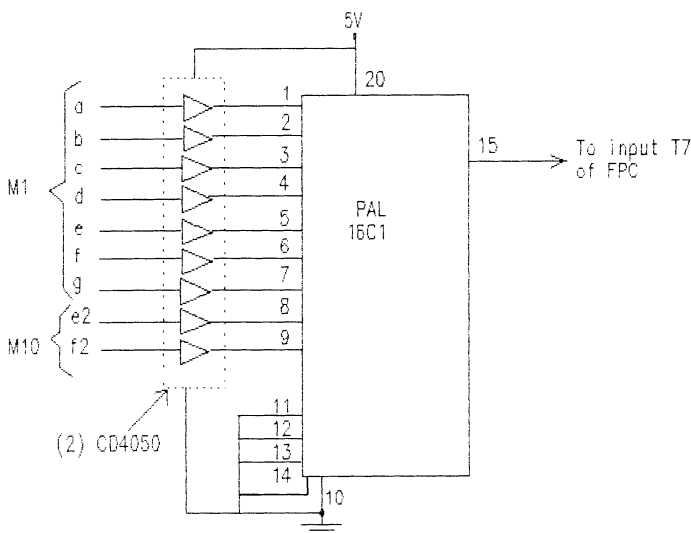


74C915 (A, B, C, and D) are converted from a 10 V logic level to a 5 V logic level with four voltage followers contained in the IC CD4050. The testable inputs T0 to T3 receive the BCD data that correspond to the hour units H1. The FPC also needs to know three more factors: (1) the digit that represents the tens of hours (H10), (2) the bit that indicates if the hour is AM or PM, and (3) the bit that states when the hour has just changed in order to start the vocal message announcing a different hour. To send even more information to the FPC about the three last factors, the testable input T4 is used for detecting line H10. When line H10 goes to a logic one, the tens of hour digit is activated. Also, the testable input T5 receives the bit AM. If the bit AM is in a logic one, the hour reading is within the range of 12:00 AM to 11:59 AM. Notice that T4 and T5 receive their input signals in a 5 V logic, thanks to the two voltage followers contained in the IC CD4050. The FPC also needs to know when the hour reading contains exactly zero minutes. For example, when the hour changes from 7:59 to 8:00, the FPC will detect this change by reading its testable input T7; therefore, we need a circuit that detects when the digits representing the minutes (M10 and M1) change from the number 59 to 00.

Figure 4.7 shows the circuit that detects when digits M10 and M1 change from the number 59 to 00. To detect when digit M1 changes from nine to zero, we will need a seven-input AND gate performing the product

$$M1 = a * b * c * d * e * f * /g$$

Notice that input “g” is inverted because the digital number “0” is formed



**Figure 4.7** Circuit for detecting when digits M10 and M1 are zero.



with all the segments activated, except “g.” On the other hand, to detect when digit M10 changes from 5 to zero, a two-input AND gate is needed. The following truth table (Table 4.10) shows the six logical states of digit M10.

As Table 4.10 shows, we are using only six logical states for digit M10. Thus, to detect when digit M10 is zero, we only require the following equation:

$$M10 = e2 * f2$$

The product  $e * f$  is sufficient to detect the zero state in digit M10 because no other logical state presents the same combination. Now, we need to join the two equations to detect when both digits M10 and M1 are zero. In this manner, we get the following equation that detects when M10 and M1 are in state 00.

$$ZERO = a * b * c * d * e * f * /g * e2 * f2$$

To perform the above logic equation we need a PAL that accepts a minimum of nine inputs and nine product terms. The PAL16C1 is suitable to perform the equation ZERO. Figure 4.7 shows the PAL16C1 which must be programmed with the preceding equation ZERO.

The software program required to perform the complete task of reading the hour and giving the vocalized messages is shown in Table 4.11. The main routine is more or less similar to the one presented previously. That is, the program first needs to detect if the time presented is exactly a determined hour with zero minutes. The output bit of the PAL16C1 is the one that indicates to the FPC if digits M10 and M1 are in the numbers 00. In this way, the FPC must stay in a loop waiting for the input T7 to go to a logic high. When the input T7 is high, the program will have to determine the type of message to be announced by reading the time presented in the inputs T0 to T5. The inputs T0 to T5 will indicate to the program the location where the required message has been stored.

TABLE 4.10								
Truth Table for Digit M10								
Number	a2	b2	c2	d2	e2	f2	g2	
0	1	1	1	1	1	1	0	
1	0	1	1	0	0	0	0	
2	1	1	0	1	1	0	1	
3	1	1	1	1	0	0	1	
4	0	1	1	0	0	1	1	
5	1	0	1	1	0	1	1	

**TABLE 4.11**  
Software Program for the FPC Am29CPL154 to Perform  
the Control of the Talking Clock

```

DEVICE (CPL154)

DEFAULT = 1;
DEFINE      "test inputs"
h10 = t4
am  = t5
sby = t6
mints = t7
equal = eq

      "allophones and pauses are given name assignments"
pa2 = 01#h      "      -----"
pa3 = 02#h      "      t6->|1      28|'- Vcc "
pa4 = 03#h      "      p0<-|2      27|<-clk "
pa5 = 04#h      "      p1<-|3      26|<-t7  "
oy  = 05#h      "      p2<-|4      25|<-t0  "
ay  = 06#h      "      p3<-|5      Am29CPL 24|<-t1  "
eh  = 07#h      "      p4<-|6      154    23|<-t2  "
kk3 = 08#h      "      p5<-|7      22|<-t3  "
pp  = 09#h      "      p6<-|8      21|<-t4  "
jh  = 0A#h      "      p7<-|9      20|<-t5  "
nn1 = 0B#h      "      p8<-|10     19|<-/reset"
ih  = 0C#h      "      p9<-|11     18|<->p15  "
tt2 = 0D#h      "      p10<-|12    17|<->p14  "
rr1 = 0E#h      "      p11<-|13    16|<->p13  "
ax  = 0F#h      "      Gnd->|14    15|<->p12  "
mm  = 10#h      "      -----"
tt1 = 11#h
dh1 = 12#h      iy = 13#h      ey = 14#h      dd1 = 15#h      uw1 = 16#h
ao  = 17#h      aa = 18#h      yy2 = 19#h      ae = 1A#h      hh1 = 1B#h
bb1 = 1C#h      th = 1D#h      uh = 1E#h      uw2 = 1F#h      aw = 20#h
dd2 = 21#h      gg3 = 22#h      vv = 23#h      gg1 = 24#h      sh = 25#h
zh  = 26#h      rr2 = 27#h      ff = 28#h      kk2 = 29#h      kk1 = 2A#h
zz  = 2B#h      ng  = 2C#h      ll  = 2D#h      ww  = 2E#h      xr  = 2F#h
wh  = 30#h      yy1 = 31#h      ch  = 32#h      er1 = 33#h      er2 = 34#h
ow  = 35#h      dh2 = 36#h      ss  = 37#h      nn2 = 38#h      hh2 = 39#h
or  = 3A#h      ar  = 3B#h      yr  = 3C#h      gg2 = 3D#h      el  = 3E#h
bb2 = 3F#h;

DEFAULT-OUTPUT = 0000#h;

TEST-CONDITION = SBY; "/STANDBY is the default test condition"

BEGIN
"1"      ,goto pl(one);
"2"      ,goto pl(two);
"3"      ,goto pl(three);
"4"      ,goto pl(four);
"5"      ,goto pl(five);
"6"      ,goto pl(six);
"7"      ,goto pl(seven);

```

```

"8"      .goto pl(eight);
"9"      .goto pl(nine);
"10"     .goto pl(ten);
"11"     .goto pl(eleven);
"12"     .goto pl(twelve);
"13"one: ww,    call pl(read); "ONE"
"14"     ax,    call pl(read);
"15"     ax,    call pl(read);
"16"     mn1,   call pl(read);
"17"     pa2,   call pl(read);
"18"     .goto pl(msg2);
"19"two: tt2,   call pl(read); "TWO"
"20"     uw2,   call pl(read);
"21"     pa2,   call pl(read);
"22"     .goto pl(msg2);
"23"three: th,  call pl(read); "THREE"
"24"     rr1,   call pl(read);
"25"     iy,    call pl(read);
"26"     pa2,   call pl(read);
"27"     .goto pl(msg2);
"28"four: ff,   call pl(read); "FOUR"
"29"     ff,    call pl(read);
"30"     or,    call pl(read);
"31"     pa2,   call pl(read);
"32"     .goto pl(msg2);
"33"five: ff,   call pl(read); "FIVE"
"34"     ff,    call pl(read);
"35"     ay,    call pl(read);
"36"     vv,    call pl(read);
"37"     pa2,   call pl(read);
"38"     .goto pl(msg2);
"39"six: ss,    call pl(read); "SIX"
"40"     ss,    call pl(read);
"41"     ih,    call pl(read);
"42"     ih,    call pl(read);
"43"     pa3,   call pl(read);
"44"     kk2,   call pl(read);
"45"     ss,    call pl(read);
"46"     pa2,   call pl(read);
"47"     .goto pl(msg2);
"48"seven: ss,  call pl(read); "SEVEN"
"49"     ss,    call pl(read);
"50"     eh,    call pl(read);
"51"     eh,    call pl(read);
"52"     vv,    call pl(read);
"53"     eh,    call pl(read);
"54"     mn1,   call pl(read);
"55"     pa2,   call pl(read);
"56"     .goto pl(msg2);
"57"eight: ey,  call pl(read); "EIGHT"
"58"     pa3,   call pl(read);
"59"     tt2,   call pl(read);

```

```

"60"    pa2,    call pl(read);
"61"    ,goto pl(msg2);
"62"nine:nn2, call pl(read);    "NINE"
"63"    aa,     call pl(read);
"64"    ay,     call pl(read);
"65"    nn1,    call pl(read);
"66"    pa2,    call pl(read);
"67"    ,goto pl(msg2);
"68"ten:tt2,   call pl(read);    "TEN"
"69"    eh,     call pl(read);
"70"    eh,     call pl(read);
"71"    nn1,    call pl(read);
"72"    pa5,    call pl(read);
"73"    ,goto pl(msg2);
"74"eleven:ih,call pl(read);    "ELEVEN"
"75"    ll,     call pl(read);
"76"    eh,     call pl(read);
"77"    eh,     call pl(read);
"78"    vv,     call pl(read);
"79"    eh,     call pl(read);
"80"    nn1,    call pl(read);
"81"    pa4,    call pl(read);
"82"    ,goto pl(msg2);
"83"twelve:tt2,call pl(read);    "TWELVE"
"84"    wh,     call pl(read);
"85"    eh,     call pl(read);
"86"    eh,     call pl(read);
"87"    ll,     goto pl(stay);
"88"    vv,     call pl(read);
"89"    pa4,    call pl(read);
"90"    ,goto pl(msg2);
"91"msg2: ow,   call pl(read);    "O' CLOCK"
"92"    pa2,    call pl(read);
"93"    kk1,    call pl(read);
"94"    ll,     call pl(read);
"95"    aa,     call pl(read);
"96"    aa,     call pl(read);
"97"    pa3,    call pl(read);
"98"    kk2,    call pl(read);
"99"    pa5,    call pl(read);
"100"   ey,     if (not am) then goto pl(pm);
"101"   eh,     call pl(read);    "am"
"102"   eh,     call pl(read);
"103"   mm,     call pl(read);
"104"   pa5,    call pl(read);
"105"   ,goto pl(delay);
"106"pm:pp,    call pl(read);    "pm"
"107"   iy,     call pl(read);
"108"   pa2,    call pl(read);
"109"   eh,     call pl(read);
"110"   eh,     call pl(read);
"111"   mm,     call pl(read);

```

```

"112" pa5, call pl(read);
"113"delay: ,load pl(255); "70 seconds delay to avoid reading.."
"114" pa5, call pl(read); " the mints input again"
"115" ,while (creg <> 0) loop to pl(delay);
"116" ,load pl(50);
"117"del2:pa5, call pl(read);
"118" ,while (creg <> 0) loop to pl(del2);

***** wait for data input to go high *****
"119"stay: ,if (not mints) then goto pl(stay); "Wait for M10=M1=00"
"120"msg1: ih, call pl(read); " IT IS..."
"121" tt1, call pl(read);
"122" pa2, call pl(read);
"123" ih, call pl(read);
"124" ss, call pl(read);
"125" ss, call pl(read);
"126" pa5, call pl(read);
"127" ,goto tm(001111#b); "Go to address specified by mask"
"subroutine for reading the standby status of the speech processor"
"128"read: ,continue;
"129"styl: ,if (not sby) then goto pl(styl); "reading SBY"
"130" ,ret;

.org 255#d
"131" ,goto pl(stay);
END.

```

## 4.5 Designing a Speaking Coin Detector

A coin detector is a widely used commercial device for most vending machines now in use in the market. Vending machines have mechanisms that are usually difficult to operate because the process for obtaining a product is different in each type of device. Moreover, most vending machines do not have a display that indicates the sum of money that has been deposited. With these drawbacks in mind, we will design a general-purpose coin detector that gives spoken instructions to users. This coin detector can be installed in conventional machines to enhance their versatility. Even a child who cannot read printed instructions will easily understand the spoken directions for obtaining the chosen product.

Figure 4.8 presents the circuit for a talking coin detector that uses the microcontroller  $\mu\text{C}$  8748 as the main processing unit. The  $\mu\text{C}$  8748 receives three input signals, indicating the type of coin that has been deposited. The three input signals are assigned for each type of coin, that is, nickels, dimes, and quarters. Three D-type flip-flops detect the kind of coin by receiving a transient pulse in their inputs "SET" coming from the coin-detecting mechanism. The three outputs Q1, Q2, and Q3 of the flip-flops are routed to the inputs P2.0 to P2.3 of the  $\mu\text{C}$  8748. The output P2.3 of the  $\mu\text{C}$  8748 is used



detects the selection that the user has made, it will activate the output signal that corresponds to the desired product.

From the microcontroller, eight address lines (P1.0–P1.7) and two control signals (T1 and /WR) are utilized to drive the Digitalker DT1050. Two speech ROMs (SSR1 and SSR2) contain, in compressed form, the data required for the 144 addressable words. The 16-key encoder 74C922 is used to let the operator select the product he wants when he has deposited a total amount of fifty cents. In this case, the Digitalker will issue the message “Please, mark the number.” When the user presses the number in the keypad, the  $\mu\text{C}$  8748 sends the respective data to the four most significant bits in port two. Notice that a 4-to-16 decoder can be added in order to control the external mechanism that will bring the products out. The software program for the  $\mu\text{C}$  is shown in Table 4.12.

**TABLE 4.12**  
Software for  $\mu\text{C}$  8748 to Control a Talking Coin Detector

Add	Op	Code	Mnemonics	Comments
00	05		EN I	;Enables Interrupt
01	04	05	JMP START	;
03	04	F2	JMP CANCEL	;Cancel operation and return money.
05	8A	00	START: ANL P2, #00H	;
07	54	08	ORL P2, #08H	;Reset D-type flip flops
09	9A	00	ANL P2, #00H	; /CS1=0, /WR=
0B	99	00	ANL P1, #00H	;
0C	27		CLR A	;Acc =00H
0E	97		CLR C	;Clear carry flag
0F	A9		MOV R1, A	;Clear registers R0-R7
1B	AA		MOV R2, A	;
11	AB		MOV R3, A	;
12	AC		MOV R4, A	;
13	AD		MOV R5, A	;
14	AE		MOV R6, A	;
15	AF		MOV R7, A	;
16	0A		READ: INA, P2	;Load P2 contents to accumulator
17	A9		MOV R1, A	;Store reading in register R1
18	53	01	ANL A, #01H	;Mask the Acc to test the 5c bit
1A	96	26	JNZ FIVEC	;Go to add five cents
1C	F9	49	MOV A, R1	;Move reading to Acc
1D	53	02	ANL A, #02H	;Mask the Acc to test the 10c bit
1F	96	2A	JNZ TENC	;Go to add ten cents
21	F9		MOV A, R1	;Move reading to Acc
22	53	04	ANL A, #04H	;Mak the Acc to test the 25c bit
24	96	2E	JNZ TFIVE	;Go to add twentyfive cents
26	6A	05	FIVEC: ADD R2, #05H	;Add five cents to register R2
28	04	30	JMP COMPARE	;

```

2A 6A 0A TENC :ADD R2, #0AH ;Add ten cents to register R2
2C 04 30      JMP COMPARE ;
2E 6A 19 TFIVEC :ADD R3, #19H ;Add 25 cents to register R2
;
30 23 32 COMPARE:MOV A, #32H ;Load Acc with number 50d
32 D9      XRL A, R1 ;Compare the amount against 50d
33 C6      JZ MSG50 ;If amount=50c goto MSG50
;
35 23 05 AMOUNT:MOV A, #05H ;Load Acc with constant 5c
37 DA      XRL A, R2 ;Compare
38 C6 64      JZ MSG5 ;If amount= 5 cents goto MSG5
3A 23 0A      MOV A, #0AH ;Load Acc with constant 10c
3C DA      XRL A, R2 ;Compare
3D C6 6C      JZ MSG10 ;If amount= 10 cents goto MSG10
3F 23 0F      MOV A, #0FH ;Load Acc with constant 15c
41 DA      XRL A, R2 ;Compare
42 C6 74      JZ MSG15 ;If amount= 15 cents goto MSG15
44 23 14      MOV A, #14H ;Load Acc with constant 20c
46 DA      XRL A, R2 ;Compare
47 C6 7C      JZ MSG20 ;If amount= 20 cents goto MSG20
49 23 19      MOV A, #19H ;Load Acc with constant 25c
4B DA      XRL A, R2 ;Compare
4C C6 84      JZ MSG25 ;If amount= 25 cents goto MSG25
4E 23 1E      MOV A, #1EH ;Load Acc with constant 30c
50 DA      XRL A, R2 ;Compare
51 C6 90      JZ MSG30 ;If amount= 30 cents goto MSG30
53 23 23      MOV A, #23H ;Load Acc with constant 35c
55 DA      XRL A, R2 ;Compare
56 C6 98      JZ MSG35 ;If amount= 35 cents goto MSG35
58 23 28      MOV A, #28H ;Load Acc with constant 40c
5A DA      XRL A, R2 ;Compare
5B C6 A4      JZ MSG40 ;If amount= 40 cents goto MSG40
5D 23 2D      MOV A, #2DH ;Load Acc with constant 45c
5F DA      XRL A, R2 ;Compare
60 C6 AC      JZ MSG45 ;If amount= 45 cents goto MSG45
62 04 B8      JMP MSG50 ;amount = 50 cents
-----
64 BD 05      MSG5:MOV R5, #05 ;"Five"
66 14 EA      CALL VOCAL ;
68 14 E0      CALL CENTS ;
6A 04 16      JMP READ ;
6C BD 0A      MSG10:MOV R5, #0A ;"Ten"
6E 14 EA      CALL VOCAL ;
70 14 E0      CALL CENTS ;
72 04 16      JMP READ ;
74 BD 0F      MSG15:MOV R5, #0FH ;"Fifteen"
76 14 EA      CALL VOCAL ;
78 14 E0      CALL CENTS ;
7A 04 16      JMP READ ;
7C BD 14      MSG20:MOV R5, #14H ;"Twenty"
7E 14 EA      CALL READ ;
80 14 E0      CALL CENTS ;
82 04 16      JMP READ ;

```



```

84 BD 14 MSG25:MOV R5, #14H ;"Twenty five"
86 14 EA CALL VOCAL ;
88 BD 05 MOV R5, #05H ;
8A 14 EA CALL VOCAL ;
8C 14 E0 CALL CENTS ;
8E 04 16 JMP READ ;
90 BD 15 MSG30:MOV R5, #15H ;"Thirty"
92 14 EA CALL VOCAL ;
94 14 E0 CALL CENTS ;
96 04 16 JMP READ ;
98 BD 15 MSG35:MOV R5, #15H ;"Thirty five"
9A 14 EA CALL VOCAL ;
9C BD 05 MOV R5, #05H ;
9E 14 EA CALL VOCAL ;
A0 14 E0 CALL CENTS ;
A2 04 16 JMP READ ;
A4 BD 16 MSG40:MOV R5, #16H ;"Forty"
A6 14 EA CALL VOCAL ;
A8 14 E0 CALL CENTS ;
AA 04 16 JMP READ ;
AC BD 16 MSG45:MOV R5, #16H ;"Forty five"
AE 14 EA CALL VOCAL ;
B0 BD 05 MOV R5, #05H ;
B2 14 EA CALL VOCAL ;
B4 14 E0 CALL CENTS ;
B6 04 16 JMP READ ;
B8 BD 17 MSG50:MOV R5, #17H ;"Fifty"
BA 14 EA CALL VOCAL ;
BC 14 E0 CALL CENTS ;
BE BD 78 MOV R5, #78H ;"PLease"
C0 14 EA CALL VOCAL ;
C2 BD 69 MOV R5, #69H ;"Mark"
C4 14 EA CALL VOCAL ;
C6 BD 8A MOV R5, #8A ;"The"
C8 14 EA CALL VOCAL ;
CA BD 70 MOV R5, #70H ;"Number"
CC 14 EA CALL VOCAL ;
CE 26 CE DATA:JNT0 DATA ;Wait for DA to go high
D0 08 INS A, BUS ;Read the pressed number
D1 3A OUTL P2, A ;Turn on the product mechanism for
D2 B8 0F MOV R0, #0FH ;five seconds
D4 B9 FF T4:MOV R1, #FFH ;
D6 BA FF T3:MOV R2, #FFH ;
D8 EA D8 T2:DJNZ R2, T2 ;
DA E9 D6 DJNZ R1, T3 ;
DC E8 D4 DJNZ R0, T4 ;
DE 04 05 JMP START ;
;Routine for the word "Cents"
-----
E0 BD 40 CENTS:MOV R5, #40H ;"Cent.."
E2 14 EA CALL VOCAL ;
E4 BD 81 MOV R5, #81H ;"ss"

```

E6	14	EA	CALL VOCAL	;
E8	83		RET	;
				;Routine to control the DT1050
-----				
EA	FD		VOCAL: MOV A, R5	;
EC	39		OUTL P1, A	;
ED	90		MOVX A, @R0	;
EE	00		NOP	;
EF	46	EF	WAIT: JNT1 WAIT	;
F1	83		RET	;
				;Routine to cancel the operation
-----				
F2	23	80	CANCEL: MOV A, # 80H	;
F4	3A		OUTL P2, A	;
F6	B8		T5: MOV R0, #FFH	;
F8	EB		DJNZ R0, T5	;
FA	04	05	JMP START	;

## 4.6 Designing a Talking Coffee Machine Controller

The coffee machine controller presented here will use part of the software program for the coin detector in Section 4.5. The process of designing a controller for a coffee vending machine now will be applied to the allophone-based speech processor SPO256-AL2. Certainly, with this speech processor we will be allowed to create any type of word for indicating to the user the choices he might have.

The coffee machine controller will be built around the microcontroller  $\mu\text{C}$  8748. The following possible varieties of coffee will be available for the consumer:

1. Coffee black
2. Coffee with sugar
3. Coffee with cream
4. Coffee with cream and sugar
5. Extra sugar
6. Extra cream

The variations of coffee are indicated in numbers one to four. Numbers five and six are available in order to let the user add extra sugar or extra cream to the type of coffee he has already selected. In this way, a keypad will be installed with options one to six. Also, number seven in the keypad will be labeled "coin return" to permit the user to cancel the operation and to make the machine return the coins he has deposited. The system will be programmed to accept the "coin return" request when the consumer is still inserting coins or before he has selected a type of coffee. This feature stops the vending machine

from serving a coffee and then returning the coins to the user if he presses the “coin return” key. In this manner, the interrupt input of the  $\mu\text{C}$  8748 will be used for “coin return” and will be disabled when the machine starts preparing the coffee.

The program of the  $\mu\text{C}$  8748 will first detect if the total amount of 50 cents has been deposited before starting to prepare the coffee. To detect if the amount of 50 cents has been deposited, the program will take part of the software presented in Section 4.5. Some minor changes to that software program will take place now because our  $\mu\text{C}$  8748 will be controlling the speech processor SPO256-AL2.

The output control signals that need to be generated from the  $\mu\text{C}$  8748 are:

1. Cup drop
2. Water on
3. Coffee on
4. Cream on
5. Sugar on
6. Coin return

The routine that the software program has to perform is as follows:

1. The program will execute nothing until a coin is detected.
2. Upon coin detection the speech processor will announce the amount the operator has deposited until he reaches a total amount of 50 cents.
3. The speech processor will indicate that the user should select his options with the message “Please select your options.”
4. If coin return is detected, the machine returns coins, gives the message “Pick up your coins please,” and waits for the next coin insertion to be deposited.
5. The cup has 2.0 s to get into place.
6. Depending on selection, powders will be released for different intervals as follows:
 

coffee .....	2.0 s
coffee with sugar .....	coffee 2.0 s, sugar 2.0 s
coffee with cream .....	coffee 2.0 s, cream 2.0 s
coffee with cream and sugar .....	coffee 2.0 s, sugar 2.0 s, cream 2.0 s
7. Check to see if extra sugar and/or extra cream are selected. If yes, extra cream 2.0 s, extra sugar 2.0 s.
8. Water on for 10.0 s.

As can be seen, there are four possible types of coffee and two extra options for extra sugar and/or extra cream. The software program will make the speech processor speak the options the user is selecting. In this manner, the user will be quite sure that the options he is selecting are being accepted by



**TABLE 4.13**Software for  $\mu$ C 8748 to Control a Talking Coffee Vending Machine

Add	Op	Code	Mnemonics	Comments
00	05		EN I	; Enables Interrupt
01	04	05	JMP START	;
03	04	F2	JMP CANCEL	; Cancel operation and return money.
05	8A	00	START: ANL P2, #00H	;
07	54	08	ORL P2, #80H	; Reset D-type flip flops
09	9A	00	ANL P2, #00H	;
0B	99	00	ANL P1, #00H	;
0D	97		CLR C	; Clear carry flag
0E	A9		MOV R1, A	; Clear registers R0-R7
0E	AA		MOV R2, A	;
0F	AB		MOV R3, A	;
10	AC		MOV R4, A	;
11	AD		MOV R5, A	;
12	AE		MOV R6, A	;
13	AF		MOV R7, A	;
14	27		READ: CLR A	; Acc =00H
15	54	80	ORL P2, #80H	; Clear D-type flip flops
17	08		INS A, BUS	; Load BUS contents to accumulator
19	A9		MOV R1, A	; Store reading in register R1
1A	53	01	ANL A, #01H	; Mask the Acc to test the 5c bit
1C	96	28	JNZ FIVEC	; Go to add five cents
1E	F9	49	MOV A, R1	; Move reading to Acc
1F	53	02	ANL A, #02H	; Mask the Acc to test the 10c bit
21	96	2C	JNZ TENC	; Go to add ten cents
23	F9		MOV A, R1	; Move reading to Acc
24	53	04	ANL A, #04H	; Mak the Acc to test the 25c bit
26	96	30	JNZ TFIVE	; Go to add twentyfive cents
28	6A	05	FIVEC: ADD R2, #05H	; Add five cents to register R2
2A	04	32	JMP COMPARE	;
2C	6A	0A	TENC : ADD R2, #0AH	; Add ten cents to register R2
2E	04	32	JMP COMPARE	;
30	6A	19	TFIVEC : ADD R2, #19H	; Add 25 cents to register R2
32	23	05	COMPARE: MOV A, #05H	; Load Acc with constant 5c
34	DA		XRL A, R2	; Compare
35	C6	61	JZ MSG5	; If amount= 5 cents goto MSG5
37	23	0A	MOV A, #0AH	; Load Acc with constant 10c
39	DA		XRL A, R2	; Compare
3A	C6	67	JZ MSG10	; If amount= 10 cents goto MSG10
3C	23	0F	MOV A, #0FH	; Load Acc with constant 15c
3E	DA		XRL A, R2	; Compare
3F	C6	6D	JZ MSG15	; If amount= 15 cents goto MSG15
41	23	14	MOV A, #14H	; Load Acc with constant 20c
43	DA		XRL A, R2	; Compare
44	C6	73	JZ MSG20	; If amount= 20 cents goto MSG20
46	23	19	MOV A, #19H	; Load Acc with constant 25c
48	DA		XRL A, R2	; Compare

```

49 C6 79      JZ MSG25      ;If amount= 25 cents goto MSG25
4B 23 1E      MOV A, #1EH   ;Load Acc with constant 30c
4D DA        XRL A, R2      ;Compare
4E C6 82      JZ MSG30      ;If amount= 30 cents goto MSG30
50 23 23      MOV A, #23H   ;Load Acc with constant 35c
52 DA        XRL A, R2      ;Compare
53 C6 88      JZ MSG35      ;If amount= 35 cents goto MSG35
55 23 28      MOV A, #28H   ;Load Acc with constant 40c
57 DA        XRL A, R2      ;Compare
58 C6 92      JZ MSG40      ;If amount= 40 cents goto MSG40
5A 23 2D      MOV A, #2DH   ;Load Acc with constant 45c
5C DA        XRL A, R2      ;Compare
5D C6 98      JZ MSG45      ;If amount= 45 cents goto MSG45
5F 04 A2      JMP MSG50      ;amount = 50 cents
-----
61 BD 05      MSG5:MOV A, #1EH ;"Five"
63 34 65      CALL FIND     ;
65 04 14      JMP READ      ;
67 BD 0A      MSG10:MOV A, #23H ;"Ten"
69 34 65      CALL FIND     ;
6B 04 14      JMP READ      ;
6D BD 0F      MSG15:MOV A, #28H ;"Fifteen"
6F 34 65      CALL FIND     ;
71 04 14      JMP READ      ;
73 BD 14      MSG20:MOV R5, #2FH ;"Twenty"
75 34 65      CALL FIND     ;
77 04 14      JMP READ      ;
79 23 2F      MSG25:MOV A, #2FH ;"Twenty five"
7A 34 65      CALL FIND     ;
7C BD 1E      MOV A, #1EH   ;
7E 34 65      CALL FIND     ;
80 04 14      JMP READ      ;
82 23 38      MSG30:MOV A, #38H ;"Thirty"
84 34 65      CALL FIND     ;
86 04 14      JMP READ      ;
88 23 38      MSG35:MOV A, #38H ;"Thirty five"
8A 34 65      CALL FIND     ;
8C 23 1E      MOV A, #1EH   ;
8E 34 65      CALL FIND     ;
90 04 14      JMP READ      ;
92 23 3D      MSG40:MOV A, #3DH ;"Forty"
94 34 65      CALL FIND     ;
96 04 14      JMP READ      ;
98 23 3D      MSG45:MOV A, #3DH ;"Forty five"
9A 34 65      CALL FIND     ;
9C 23 1E      MOV A, #1EH   ;
9E 34 65      CALL FIND     ;
A0 04 14      JMP READ      ;
A2 23 43      MSG50:MOV A, #43H ;"Fifty"
A4 34 65      CALL FIND     ;
A6 23 93      MOV A, #93H   ;
A8 34 65      CALL FIND     ;"Cents"

```

```

AA 23 77      MOV A, #77H      ; "Select your options"
AC 34 65      CALL FIND       ;
AE 23 8D      MOV A, #8DH      ; "Please"
B0 34 65      CALL FIND       ;
B2 26 B2      DATA: JNT0 DATA ; Wait for DA to go high
B4 15         DIS I           ; Disable the "return coin" key
B5 08         INS A, BUS      ; Read the pressed number
B6 AD         MOV R5, A        ; Store selection in register R5
B7 9A 00      ANL P2, #00H     ;
B9 8A 01      ORL P2, #01H     ; Cup drop
BB 34 38      CALL DEL2        ; Wait cup drop for 2 seconds
BD 23 50      MOV A, #50H      ;
BF DD         XRL A, R5        ; Extra cream?
C0 96 CE      JNZ XTSGR        ; Jump if not extra cream to XTSGR
C2 8A 08      ORL P2, #08H     ; Add extra cream
C4 34 38      CALL DEL2        ;
C6 23 63      MOV A, #63H      ; "Extra Cream"
C8 34 65      CALL FIND       ;
CA 23 5D      MOV A, #5DH      ;
CC 34 65      CALL FIND       ;
CE 23 40      XTSGR: MOV A, #40H ;
D0 DD         XRL A, R5        ; Extra sugar?
D1 96 DF      JNZ CFWS         ; Jump if not extra sugar to CFWS
D3 8A 10      ORL P2, #10H     ; Add extra sugar
D5 34 38      CALL DEL2        ;
D7 23 63      MOV A, #63H      ; "Extra sugar"
D9 34 65      CALL FIND       ;
DB 23 57      MOV A, #57       ;
DD 34 65      CALL FIND       ;
DF 23 10      CFWS: MOV A, #10H ;
E1 DD         XRL A, R5        ; Coffee with sugar?
E2 96 F6      JNZ CFWC         ; Jump to Coffee/Cream
E4 8A 14      ORL P2, #14H     ; Add coffee with sugar
46 34 38      CALL DEL2        ;
E8 23 4B      MOV A, #4BH      ; "Coffee"
EA 34 65      CALL FIND       ;
EC 23 52      MOV A, #52H      ; "With"
EE 34 65      CALL FIND       ;
F0 23 57      MOV A, #57H      ; "Sugar"
F2 34 65      CALL FIND       ;
F4 24 32      JMP WATER        ;
F6 23 20      CFWC: MOV A, #20H ;
F8 DD         XRL A, R5        ; Coffee with cream?
F9 24 0C      JNZ CWCS         ; Jump to Coffee/Cream/Sugar
FB 8A 0C      ORL P2, #0CH     ; Add coffee with cream
FD 34 38      CALL DEL2        ;
FF 23 4B      MOV A, #4B       ; "Coffee"
100 34 65     CALL FIND       ;
102 23 52     MOV A, #52H      ; "With"
104 34 65     CALL FIND       ;
106 23 5D     MOV A, #5DH      ; "Cream"
108 34 65     CALL FIND       ;

```

```

10A 24 32      JMP WATER      ;
10C 23 30      CWCS:MOV A, #30H ;
10E DD        XRL A, R5       ;Coffee with cream and sugar?
10F 96 2A      JNZ CBLACK     ;Go to prepare coffee black
111 8A 1C      ORL P2, #1CH   ;Add coffee + cream + sugar
112 34 38      CALL DEL2      ;
114 23 4B      MOV A, #4BH    ;"Coffee"
116 34 65      CALL FIND      ;
118 23 52      MOV A, #52H    ;"With"
11A 34 65      CALL FIND      ;
11C 23 5D      MOV A, #5DH    ;"Cream"
11E 34 65      CALL FIND      ;
120 23 52      MOV A, #52H    ;"With"
122 34 65      CALL FIND      ;
124 23 57      MOV A, #57H    ;"Sugar"
126 34 65      CALL FIND      ;
128 24 32      JMP WATER      ;
12A 23 4B      CBLACK:MOV A, #4BH ;"Coffee"
12C 34 65      CALL FIND      ;
12E 8A 1C      ORL P2, #04H   ;Release coffee only
130 34 38      CALL DEL2      ;
132 8A 20      WATER:ORL P2, #20H ;Release water for 10 seconds
134 34 47      CALL DEL10     ;
136 04 05      JMP START      ;
                                   ;Routine for 2 seconds delay
-----
138 B8 05      DEL2:MOV R0, #05H ;2 SECONDS
13A B9 FF      T4:MOV R1, #FFH  ;
13C BA FF      T3:MOV R2, #FFH  ;
13E EA 3E      T2:DJNZ R2, T2   ;
140 E9 3C      DJNZ R1, T3      ;
142 E8 3A      DJNZ R0, T4      ;
144 9A 00      ANL P2, #00H     ;
146 83         RET             ;
                                   ;Routine for 10 seconds delay
-----
147 B8 1E      DEL10:MOV R0, #1EH ;10 seconds
149 B9 FF      T4:MOV R1, #FFH  ;
14B BA FF      T3:MOV R2, #FFH  ;
14D EA 4D      T2:DJNZ R2, T2   ;
14F E9 4B      DJNZ R1, T3      ;
151 E8 49      DJNZ R0, T4      ;
153 9A 00      ANL P2, #00H     ;
155 83         RET             ;
                                   ;Routine for coin return
-----
156 23 20      CANCEL:MOV A, #20H ;
158 3A         OUTL P2, A       ;Output "return coin" is activated
159 34 38      CALL DEL2      ;for two seconds
15A 23 6A      MOV A, #6AH     ;
15C 3A         OUTL P2, A       ;"Pick up your coins"
15D 23 65      CALL FIND      ;

```



```

15F 23 91      MOV A, #91H      ; "Please"
161 23 65      CALL FIND        ;
163 04 05      JMP START        ;
-----
165 BD 00      FIND: MOV R5, #00H ;
167 AD         MOV R5, A         ; Pointer for page three
168 E3         MOV P3 A, @A      ; Acc = # of allophones
169 AC         MOV R4, A         ; R4 = #n; for n allophones
16A 27         CLR A             ;
16B 1D         RICK: INC R5       ; Increment R5 to get next allophone
16C FD         MOV A, R5         ;
16D E3         MOV P3 A, @A      ;
16E 39         OUTL P1, A        ;
16F 80         MOVX A, @R0       ; /WR is pulsed low for 5 uS
170 46 70      SBY: JNT1 SBY      ; Wait for /SBY input to go high
172 EC 6B      DJNZ R4, RICK      ;
174 83         RET              ;
-----

```

Add	Data	Comments
300	0A	;Five
301	0E	;Ten
302	09	;Fifteen
303	13	;Twenty
304	1A	;Thirty
305	1E	;Forty
306	23	;Fifty
307	2A	;Coffee
308	32	;With
309	36	;Sugar
30A	03	;Cream
30B	2B	;Extra
30C	2C	;Pick up your coins
30D	35	;Select your options
30E	04	;Please
30F	39	;Cents
310	04	;4 allophones
311	28	;
312	28	;
313	06	;2 allophones
314	23	;TWO
315	1F	;
316	03	;3 allophones
317	10	;THREE
318	0E	;
319	13	;
31A	03	;3 allophones
31B	28	;FOUR
31C	28	;
31D	3A	;
31E	04	;4 allophones

```

31F  28      ;FIVE
320  28      ;
321  06      ;
322  23      ;
323  04      ;6 ALLOPHONES
324  0D      ;TEN
325  07      ;
326  07      ;
327  0B      ;
328  07      ;7 allophones
329  0C      ;FIFTEEN
32A  28      ;
32B  02      ;
32C  0D      ;
32D  13      ;
32E  0B      ;
32F  08      ;8 allophones
330  0D      ;TWENTY
331  30      ;
332  07      ;
333  07      ;
334  0B      ;
335  02      ;
336  0D      ;
337  13      ;
338  04      ;4 allophones
339  1D      ;THIRTY
33A  34      ;
33B  2D      ;
33C  13      ;
33D  05      ;5 allophones
33E  28      ;FORTY
33F  3A      ;
340  02      ;
341  0D      ;
342  13      ;
343  07      ;7 allophones
344  28      ;FIFTY
345  28      ;
346  0C      ;
347  28      ;
348  02      ;
349  0D      ;
34A  13      ;-----
34B  06      ;5 allophones plus one pause
34C  08      ;COFFEE
34D  AA      ;
34E  28      ;
34F  28      ;
350  13      ;
351  04      ;

```

352	03	;3 allophones plus one pause
353	2E	;WITH
354	13	;
355	11	;
356	04	;
357	04	;4 allophones plus one pause
358	32	;SUGAR
359	1E	;
35A	24	;
35B	3B	;
35C	04	;
35D	04	;4 allophones plus one pause
35E	2A	;CREAM
35F	27	;
360	0C	;
361	10	;
362	04	;
363	05	;5 allophones
364	07	;EXTRA
365	2A	;
366	37	;
367	0D	;
368	27	;
369	18	;
36A	11	;16 allophones
36B	09	;PICK UP YOUR COINS
36C	0C	;
36D	0C	;
36E	29	;
36F	03	;
370	0F	;
371	09	;
372	03	;
373	19	;
374	35	;
375	33	;
376	03	;
377	2A	;
378	05	;
379	0B	;
37A	37	;
37B	15	;21 allophones
37C	37	;SELECT YOUR OPTIONS
37D	37	;
37E	07	;
37F	07	;
380	2D	;
381	07	;
382	02	;
383	29	;
384	11	;
385	04	;

```

386  19      ;
387  35      ;
388  33      ;
389  04      ;
38A  17      ;
38B  09      ;
38C  25      ;
38D  35      ;
38E  0B      ;
38F  37      ;
390  37      ;
391  06      ; 6 allophones
392  09      ; PLEASE
393  2D      ;
394  0C      ;
395  37      ;
396  37      ;
397  04      ;
398  08      ; 7 allophones and 1 pause.
399  37      ; CENTS
39A  07      ;
39B  07      ;
39C  0B      ;
39D  11      ;
39E  37      ;
39F  37      ;
340  04      ;

```

## 4.7 Designing a Talking Random Number Generator

Random numbers are used to compute statistical problems and in games such as spinners. Special software has been created to process random numbers for many games and statistical estimations. A technique to produce random numbers electronically is to apply a burst of high-frequency clock pulses to a counter as shown in Figure 4.10. This figure shows a complete working version of a pseudorandom-number generator that tells the selected number by means of the speech processor SPO256-AL2.

Figure 4.10 shows a timer 7555 configured as a free-running oscillator. The timer 7555 oscillates at a frequency of approximately 1000 Hz, given by the equation:

$$f_o = \frac{1.44}{(R1 + 2R2)C}$$

The 47  $\mu$ F capacitor, located in the RC timing network of the timer, permits the addition of a gradual slowdown feature for the frequency ( $f_o$ ) generated. When switch S1 is closed momentarily by the operator, the timer will be



**TABLE 4.14**  
Software Program for the Pseudo-random Number Generator

```

DEVICE (CPL152)

DEFAULT = 1;
DEFINE    "test inputs"
sby = t5
s1 = t4
equal = eq
        "allophones and pauses are given name assignments"

pa2 = 01#h      "      -----"
pa3 = 02#h      "      t6->|1      28|- Vcc "
pa4 = 03#h      "      p0<-|2      27|<-clk "
pa5 = 04#h      "      p1<-|3      26|<-  "
oy = 05#h      "      p2<-|4      25|<-t0 "
ay = 06#h      "      p3<-|5      Am29CPL 24|<-t1 "
eh = 07#h      "      p4<-|6      152  23|<-t2 "
kk3 = 08#h      "      p5<-|7      22|<-t3 "
pp = 09#h      "      p6<-|8      21|<-t4 "
jh = 0A#h      "      p7<-|9      20|<-t5 "
nn1 = 0B#h      "      p8<-|10     19|<-/reset"
ih = 0C#h      "      p9<-|11     18|<-p15 "
tt2 = 0D#h      "      p10<-|12    17|<-p14 "
rr1 = 0E#h      "      p11<-|13    16|<-p13 "
ax = 0F#h      "      Gnd-|14     15|<-p12 "
mm = 10#h      "      -----"
tt1 = 11#h
dh1 = 12#h      iy = 13#h      ey = 14#h      dd1 = 15#h      uw1 = 16#h
ao = 17#h      aa = 18#h      yy2 = 19#h      ae = 1A#h      hh1 = 1B#h
bb1 = 1C#h      th = 1D#h      uh = 1E#h      uw2 = 1F#h      aw = 20#h
dd2 = 21#h      gg3 = 22#h      vv = 23#h      gg1 = 24#h      sh = 25#h
zh = 26#h      rr2 = 27#h      ff = 28#h      kk2 = 29#h      kk1 = 2A#h
zz = 2B#h      ng = 2C#h      ll = 2D#h      ww = 2E#h      xr = 2F#h
wh = 30#h      yy1 = 31#h      ch = 32#h      er1 = 33#h      er2 = 34#h
ow = 35#h      dh2 = 36#h      ss = 37#h      nn2 = 38#h      hh2 = 39#h
or = 3A#h      ar = 3B#h      yr = 3C#h      gg2 = 3D#h      el = 3E#h
bb2 = 3F#h;

DEFAULT-OUTPUT = 0000#h;
TEST-CONDITION = SBY; "/STANDBY is the default test condition"
BEGIN
"0"      ,goto pl(zero);
"1"      ,goto pl(one);
"2"      ,goto pl(two);
"3"      ,goto pl(three);
"4"      ,goto pl(four);
"5"      ,goto pl(five);
"6"      ,goto pl(six);
"7"      ,goto pl(seven);
"8"      ,goto pl(eight);
"9"      ,goto pl(nine);

```

```

"10"zero:zz, call pl(read); "ZERO"
"11"   yr,   call pl(read);
"12"   ow,   call pl(read);
"13"   pa2,  call pl(read);
"14"           ,goto pl(stay);
"15"one:ww,  call pl(read); "ONE"
"16"   ax,   call pl(read);
"17"   ax,   call pl(read);
"18"   nn1,  call pl(read);
"19"   pa2,  call pl(read);
"20"           ,goto pl(stay);
"21"two:tt2, call pl(read); "TWO"
"22"   uw2,  call pl(read);
"23"   pa2,  call pl(read);
"24"           ,goto pl(stay);
"25"three:th, call pl(read); "THREE"
"26"   rr1,  call pl(read);
"27"   iy,   call pl(read);
"28"   pa2,  call pl(read);
"29"           ,goto pl(stay);
"30"four:ff,  call pl(read); "FOUR"
"31"   ff,   call pl(read);
"32"   or,   call pl(read);
"33"   pa2,  call pl(read);
"34"           ,goto pl(stay);
"35"five:ff,  call pl(read); "FIVE"
"36"   ff,   call pl(read);
"37"   ay,   call pl(read);
"38"   vv,   call pl(read);
"39"   pa2,  call pl(read);
"40"           ,goto pl(stay);
"41"six:ss,   call pl(read); "SIX"
"42"   ss,   call pl(read);
"43"   ih,   call pl(read);
"44"   ih,   call pl(read);
"45"   pa3,  call pl(read);
"46"   kk2,  call pl(read);
"47"   ss,   call pl(read);
"48"   pa2,  call pl(read);
"49"           ,goto pl(stay);
"50"seven:ss, call pl(read); "SEVEN"
"51"   ss,   call pl(read);
"52"   eh,   call pl(read);
"53"   eh,   call pl(read);
"54"   vv,   call pl(read);
"55"   eh,   call pl(read);
"56"   nn1,  call pl(read);
"57"   pa2,  call pl(read);
"58"           ,goto pl(stay);
"59"eight:ey, call pl(read); "EIGHT"
"60"   pa3,  call pl(read);

```

```

"61"    tt2,    call pl(read);
"62"    pa2,    call pl(read);
"63"                ,goto pl(stay);
"64"nine:nn2,    call pl(read);    "NINE"
"65"    aa,     call pl(read);
"66"    ay,     call pl(read);
"67"    nn1,    call pl(read);
"68"    pa2,    call pl(read);
"69"                ,goto pl(stay);
"70"ten:tt2,    call pl(read);    "TEN"
"71"    eh,     call pl(read);
"72"    eh,     call pl(read);
"73"    nn1,    call pl(read);
"74"    pa2,    call pl(read);
"75"stay:    ,if (not s1) then goto pl(stay); "Wait for S1 to go high"
"76"msg1: ih,   call pl(read);    " IT IS..."
"77"    tt1,    call pl(read);
"78"    pa2,    call pl(read);
"79"    ih,     call pl(read);
"80"    ss,     call pl(read);
"81"    ss,     call pl(read);
"82"    pa5,    call pl(read);
"83"                ,load pl(80);    "15 seconds delay to avoid reading.."
"84"delay:pa5,call pl(read); "a wrong pseudorandom number"
"85"                ,while (creg <> 0) loop to pl(delay);
"86"                ,goto tm(0001111#b); "Go to address specified by mask"
"subroutine for reading the standby status of the speech processor"
"87"read:    ,continue;
"88"sty1:    ,if (not sby) then goto pl(sty1); "reading SBY"
"89"                ,ret;

.org 127#d
"90"                ,goto pl(stay);
END.

```

program jump to lines zero to nine. Lines zero to nine of the program make the program jump to the exact location where the message is recorded. Then the recorded message is announced by the speech processor SPO256-AL2. Once the speech processor ends saying the decimal number, the program jumps to the address named "stay." The instruction in the address "stay" will keep the program waiting for switch S1 to be pressed again by the operator.

After the read-out, the random number generator can again be started by giving a 0 starting signal to the input T0 of the Am29CPL152.

If a sequence of random numbers with "2" decimal digits has to be generated, another BCD counter is required. This second BCD counter can be obtained from the same IC CD4518 and must be connected in cascade with the first BCD counter.





and R/8. This resistive network controls the magnitude of the current that passes through diac D30. The LEDs that are self-contained in the optocouplers are activated by the 4-bit latch CD4042. The four inputs (D1–D4) and the clock signal of the latch CD4042 are directly controlled by the FPC Am29CPL152, which also controls a speech processor.

When the circuit is first turned on, the R1C1 network resets the FPC Am29CPL152 and the speech processor SPO256-AL2. Under these conditions, the motor will not run because the LEDs inside the four optocouplers are not turned on. This state is achieved by programming the FPC Am29CPL152 with the software program shown in Table 4.15. The software program for the FPC Am29CPL152 makes the program jump to line 1 (labeled as stay) after the power-up reset pulse has occurred. The instruction in line 1 “If (s1) then go to pl(stay)” is used to maintain the FPC reading the status of the normally open switch S1. When the operator wishes to start running the motor, he will momentarily press switch S1 and the program will jump automatically to line 2. Line 2 of the program “call pl(del5)” then will proceed to call subroutine “del5” that causes a 5 s delay. This delay is inserted to allow the operator to quit the program before it starts running the motor. If the operator decides to continue the program, he will have to wait only 5 s after he has pressed switch S1 for the first time. When the 5 s delay has elapsed, the program goes to line 3, where the instruction “call pl(msg1)” is executed; therefore, this instruction makes the speech processor announce the word “speed.” When the program returns from subroutine “msg1,” the program will start performing sequentially the set of instructions located within lines 4 to 8 which make the speech processor say the word “one.” In this manner, the operator will hear the complete message “speed one.”

When the SPO256-AL2 ends saying the words “speed one,” the program jumps to line 9, where the instruction “if (S1) then goto pl(stay)” is located. Notice that this instruction contains the outputs named “speed1 + latch,” which starts running the motor M in the first speed. This first speed is achieved because the FPC issues the logical output 100#h, which is equivalent to having the outputs P11–P8 with the binary output 0001#b. This binary output causes a logic one at the output P8 of the FPC which turns on the LED contained inside the optocoupler that selects resistor R. Also, the output P12 of the FPC goes to a logic one, causing the CD4042 to latch the 4-bit output coming from P8 to P11. Notice that the output speed1 is also specified one line prior to the output “latch.” That is because latch CD4042 must first receive the data input and then the clock pulse in order to latch the 4-bit input.

Because the program stays in the same loop while executing the instruction in line 9, the FPC will keep giving speed1 for the motor M; therefore, resistor R is enabled to transmit the voltage across capacitor C6, which increases as the source Vac passes through zero on each alternation. When Vc reaches 30 V, the diac breakover voltage, the diac turns on and discharges C6 across the TIC216D and G leads of the triac, thus triggering it. The triac, therefore,

**TABLE 4.15**  
Software Program for the Motor-speed Controller

```

DEVICE (CPL152)

DEFAULT = 1;
DEFINE    "test inputs"
sby = t1
s1 = t0
cancel = t2
        "allophones and pauses are given name assignments"

pa2 = 01#h      "      -----"
pa3 = 02#h      "      t6->|1          28|- Vcc  "
pa4 = 03#h      "      p0<-|2          27|<-clk  "
pa5 = 04#h      "      p1<-|3          26|<-cc/sdi"
oy = 05#h      "      p2<-|4          25|<-t0   "
ay = 06#h      "      p3<-|5  Am29CPL 24|<-t1   "
eh = 07#h      "      p4<-|6      152  23|<-t2   "
kk3 = 08#h     "      p5<-|7          22|<-t3   "
pp = 09#h     "      p6<-|8          21|<-t4   "
jh = 0A#h     "      p7<-|9          20|<-t5   "
nn1 = 0B#h    "      p8<-|10         19|<-/reset"
ih = 0C#h     "      p9<-|11         18|<-p15   "
tt2 = 0D#h    "      p10<-|12        17|<-p14   "
rr1 = 0E#h    "      p11<-|13        16|<-p13   "
ax = 0F#h     "      Gnd-|14         15|<-p12   "
mm = 10#h     "      -----"
tt1 = 11#h
dh1 = 12#h    iy = 13#h    ey = 14#h    dd1 = 15#h    uw1 = 16#h
ao = 17#h    aa = 18#h    yy2 = 19#h    ae = 1A#h    hh1 = 1B#h
bb1 = 1C#h    th = 1D#h    uh = 1E#h    uw2 = 1F#h    aw = 20#h
dd2 = 21#h    gg3 = 22#h    vv = 23#h    gg1 = 24#h    sh = 25#h
zh = 26#h    rr2 = 27#h    ff = 28#h    kk2 = 29#h    kk1 = 2A#h
zz = 2B#h    ng = 2C#h    ll = 2D#h    ww = 2E#h    xr = 2F#h
wh = 30#h    yy1 = 31#h    ch = 32#h    er1 = 33#h    er2 = 34#h
ow = 35#h    dh2 = 36#h    ss = 37#h    nn2 = 38#h    hh2 = 39#h
or = 3A#h    ar = 3B#h    yr = 3C#h    gg2 = 3D#h    el = 3E#h
bb2 = 3F#h    latch = 1000#h

speed0 = 000#h speed1 = 100#h speed2 = 200#h speed3 = 300#h
speed4 = 400#h speed5 = 500#h speed6 = 600#h speed7 = 700#h
speed8 = 800#h speed9 = 900#h speed10 = A00#h speed11 = B00#h
speed12 = C00#h speed13 = D00#h speed14 = E00#h speed15 = F00#h;

DEFAULT-OUTPUT = 0000#h;

TEST-CONDITION = SBY; "/STANDBY is the default test condition"

BEGIN
"1"stay:      ,if (s1) then goto pl(stay);
"2"          ,call pl(del5);  "delay to allow user to stop the motor"
"3"          ,call pl(msg1);  "message 1 is --speed--"
"4"          ww,  call pl(read); "ONE"
"5"          ax,  call pl(read);
"6"          ax,  call pl(read);

```

```

"7"          nn1,  call pl(read);
"8"          pa2,  call pl(read);
"9"          speed1, continue;
"10"stya:speed1+latch, if (s1) then goto pl(stya);
"11"                                     ,call pl(del5);
"12"                                     ,call pl(msg1);
"13"          tt2,  call pl(read); "TWO"
"14"          uw2,  call pl(read);
"15"          pa2,  call pl(read);
"16"          speed2, continue;
"15"styb:speed2+latch, if (s1) then goto pl(styb);
"16"                                     ,call pl(del5);
"17"                                     ,call pl(msg1);
"18"          th,   call pl(read); "THREE"
"19"          rr1,  call pl(read);
"20"          iy,   call pl(read);
"21"          pa2,  call pl(read);
"22"          speed3, continue;
"23"styc:speed3+latch, if (s1) then goto pl(styc);
"24"                                     ,call pl(del5);
"25"                                     ,call pl(msg1);
"26"          ff,   call pl(read); "FOUR"
"27"          ff,   call pl(read);
"28"          or,   call pl(read);
"29"          pa2,  call pl(read);
"30"          speed4, continue;
"31"styd:speed4+latch, if (s1) then goto pl(styd);
"32"                                     ,call pl(del5);
"33"                                     ,call pl(msg1);
"34"          ff,   call pl(read); "FIVE"
"35"          ff,   call pl(read);
"36"          ay,   call pl(read);
"37"          vv,   call pl(read);
"38"          pa2,  call pl(read);
"39"          speed5, continue;
"40"stye:speed5+latch, if (s1) then goto pl(stye);
"41"                                     ,call pl(del5);
"42"                                     ,call pl(msg1);
"43"          ss,   call pl(read); "SIX"
"44"          ss,   call pl(read);
"45"          ih,   call pl(read);
"46"          ih,   call pl(read);
"47"          pa3,  call pl(read);
"48"          kk2,  call pl(read);
"49"          ss,   call pl(read);
"50"          pa2,  call pl(read);
"51"          speed6, continue;
"52"styf:speed6+latch, if (s1) then goto pl(styf);
"53"                                     ,call pl(del5);
"54"                                     ,call pl(msg1);
"55"          ss,   call pl(read); "SEVEN"
"56"          ss,   call pl(read);
"57"          eh,   call pl(read);

```

```

"58"          eh,      call pl(read);
"59"          vv,      call pl(read);
"60"          eh,      call pl(read);
"61"          nn1,     call pl(read);
"62"          pa2,     call pl(read);
"63"          speed7,  continue;
"64"styg:speed7+latch, if (s1) then goto pl(styg);
"65"          ,call pl(del5);
"66"          ,call pl(msg1);
"67"          ey,      call pl(read);    "EIGHT"
"68"          pa3,     call pl(read);
"69"          tt2,     call pl(read);
"70"          pa2,     call pl(read);
"71"          speed8,  continue;
"72"styh:speed8+latch, if (s1) then goto pl(styh);
"73"          ,call pl(del5);
"74"          ,call pl(msg1);
"75"          nn2,     call pl(read);    "NINE"
"76"          aa,      call pl(read);
"77"          ay,      call pl(read);
"78"          nn1,     call pl(read);
"79"          pa2,     call pl(read);
"80"          speed9,  continue;
"81"          speed9+latch, if (not s1) then goto pl(stopm) else wait;
"82"msg1: ss,      call pl(read);    " SPEED.."
"83"          pa2,     call pl(read);
"84"          pp,      call pl(read);
"85"          ih,      call pl(read);
"86"          dd2,     call pl(read);
"87"          pa5,     call pl(read);
"88"          ,ret;
"89"del5:         ,load pl(24);    "5 S delay to let the user..."
"90"delay:pa5,    call pl(read); "stop the motor immediatly"
"91"          ,if (cancel) goto pl(stopm);
"92"          ,while (creg <> 0) loop to pl(delay);
"93"          ,ret;
"subroutine for reading the standby status of the speech processor"
"94"read:         ,continue;
"95"sty1:         ,if (not sby) then goto pl(sty1); "reading SBY"
"96"          ,if (not cancel) then ret;
"97"stopm:        ss,      call pl(read2);    "STOP"
"98"          latch+ss,   call pl(read2);
"99"          tt1,       call pl(read2);
"100"          aa,       call pl(read2);
"101"          pp,       call pl(read2);
"102"          pa5,      call pl(read2);
"103"          ,goto pl(stay);
"104"read2:        ,continue;
"105"sty2:         ,if (not sby) then goto pl(sty2); "reading SBY"
"106"          ,ret;

          .org 127#d
"107"          ,goto pl(stay);
END.

```

becomes conductive after the source  $V_{ac}$  reaches 30 V. The result is that the motor will run at the lowest speed (speed1).

If the operator wants to increase the speed of the motor from speed1 to speed2, he will have to press switch S1 again. After the operator presses switch S1, he will again have to wait for 5 s. During this time, the operator may decide to stop the motor. If so, he only has to press the “cancel” switch for at least 1 s. This action will stop the motor and the program will proceed to jump to line 1 again in order to wait for a new starting. The 4-bit latch CD4042 serves to maintain the selected speed of the motor while the FPC is running the routines that make the SPO256-AL2 tell the speed selected by the operator. In this case, a maximum of 9 speeds were programmed in the FPC Am29CPL152 even when the set of four resistors controlling the diac D30 can give a maximum of 15 speeds. The number of speeds to the motor can be augmented by inserting more optocouplers controlling more resistors. For example, if 31 speeds are needed for the motor, just add another optocoupler below the MOC3010 controlling the resistor R/8. In this case, the new optocoupler will be controlling a new resistor named R/16. Bear in mind that if the number of speeds for the motor is augmented, the speech routines for the SPO256-AL2 must also be included. Schmitt-trigger inverters E and F make up a 10 kHz oscillator that drives the FPC Am29CPL152.

The choice of the triac will depend upon the amount of current required by the motor. In this case, a triac TIC216D was used because the motor demand was only 1.5 A. A good starting value for the parallel resistor network is  $R = 10K$ , while R11 is adjusted to 250K.

As can be seen, the versatility of the speech processor SPO256-AL2 is immense, considering all the areas of control where it can be applied. It will be shown in Chapter 5 that a speech processor in conjunction with a programmable controller allows the designer to implement a speech-based system capable of handling most tasks assigned to a system designer in the field of test and measurement.

## *Test and Measurement Circuits*

### **5.1 Designing a Talking Autorange Frequency Counter**

A very interesting application for speech processors is a talking frequency counter that voices the input frequency under measurement. If the frequency counter features autorange, the operator will be able to take frequency measurements without looking at a digital display or having to adjust the scale of the frequency counter. The frequency meter presented in this section monitors an input frequency automatically and enunciates the result, freeing an operator for other tasks.

The circuit shown in Figure 5.1 measures frequencies from dc to 10 MHz with an automatic floating point. Notice that the talking autorange frequency counter does not contain a digital display, because the speech processor will be enunciating the frequency readings and the scales in hertz, kilohertz, and megahertz.

The input frequency is selected by IC4 (74HC4051) configured as a four-to-one channel selector. IC1, IC2, and IC3 are three high-speed Johnson counters used as dividers by 10 each, connected in cascade to obtain three different input frequencies:  $f_i/10$ ,  $f_i/100$ , and  $f_i/1000$ , respectively. When the circuit is first turned on, for example, a reset pulse is applied to counter IC5. IC5 selects the direct input frequency “fi.” Consequently, BCD counters IC8a to IC8d will count from 0 to 9999 Hz. IC8a to IC8d are connected in cascade to form a 16-bit BCD counter. The BCD format of these counters was chosen because it reduces the size and development of the software program used by the FPC Am29CPL154, designated as IC10 in Figure 5.1.

The time base section is formed by IC7. The oscillator/divider MM5369EST





FPC proceeds again to determine if a new overflow has occurred. If no overflow occurs in the new selected scale, the FPC reads the measured frequency through the four testable inputs T0 to T3. This way, the FPC starts the routine to drive the speech processor Digtalker (MM54104). The Digtalker kit was selected for this application because it contains a vocabulary with numbers and words that are used for frequency measurements. Notice that the Digtalker and the two PROM memories are not included in the schematic because the FPC needs to control only three functions of the Digtalker:

1. Address bus (input lines SW1 to SW8)
2. Write input (/WR)
3. Interrupt output (/INTR)

We will now examine the operation of the complete circuit of Figure 5.1 in more detail. When the circuit is first turned on, the FPC Am29CPL154 resets the BCD counter IC5 via output P9. IC5 is the counter that registers the over range that may occur when the selected input frequency causes an overflow in the dual BCD counters IC8a to IC8d. Then the FPC clears the dual BCD counters (IC8a to IC8d) by sending a positive transient pulse via output P15. In this manner, the FPC resets counters IC8a to IC8d prior to starting a new frequency reading. Now, the FPC enables the input frequency “fi” to counters IC8a to IC8d for 1.00 s in order to determine its magnitude. When counters IC8a to IC8d finish counting the input frequency for 1.00 s, the FPC checks to see if an overflow has occurred by reading the outputs A and B of counter IC5. If A or B are in a logic high, the FPC clears counters IC8a to IC8d, and the FPC starts a new counting cycle by enabling the input “enable” of the top counter IC8a. Now counters IC8a to IC8d will be receiving the input frequency “fi/10.” If no overflow occurs, the FPC will proceed to read the measured frequency through the four testable inputs T0 to T3. Here the FPC will perform the routine for vocalizing the frequency reading.

Table 5.1 shows the cases that might occur when the input signal under measurement contains a frequency that ranges from 0 to 10 MHz.

**TABLE 5.1**  
Input Frequencies for the Autorange Frequency Counter

Frequency Range	Input Frequency	Node Selected	Display Reading	IC5 B A	Units
0-10 KHz	9999 Hz	fi	9 9 9 9	0 0	Hz
10-100 KHz	10,000 Hz	fi/10	1 0 0 0	0 1	KHz
100 KHz-1 MHz	100,000 Hz	fi/100	1 0 0 0	1 0	KHz
1- 10 MHz	1,000,000 Hz	fi/1000	1 0 0 0	1 1	MHz

Table 5.1 shows four cases representing all possible ranges for the input frequency under measurement. In the first row, the frequency counter accepts up to 9999 hertz via the node named “fi.” Bear in mind that counter IC5 makes IC4 select “fi” with outputs A and B equal to a logic zero. If the first overflow occurs, counter IC5 will then select the input frequency “fi/10.” With “fi/10” selected, the FPC will start a new counting cycle to get the new reading in the scale of kilohertz. The FPC will detect the change of scale by reading the outputs A and B of IC6 using the testable inputs T4 and T5. If a second overflow occurs, IC5 will change the input node from “fi/10” to “fi/100.” With “fi/100” selected, the frequency counter will be capable of performing frequency readings within the range of 100 to 999.9 kHz. If the input frequency under measurement, for instance, is equal to or higher than 1 MHz, counter IC5 will select the node “fi/1000”; the FPC will then start a new frequency reading in order to obtain the correct value. The logical values of outputs A and B of IC5 will serve to indicate the scale of the reading to the FPC (IC10). This way, the speech processor will enunciate a frequency reading in hertz, kilohertz or megahertz.

Table 5.2 shows the microcode program for the FPC Am29CPL154. As shown in Figure 5.1, the FPC is clocked by a 100 Hz frequency that is generated by IC7 and associated components. This means that the FPC will perform each instruction in 10 ms (10,000  $\mu$ s); therefore, the FPC will issue output pulses with 10 ms of duration. The output pulses are named “reset1,” “reset2,” “enable,” and “GL.” Thanks to the stable, low frequency of 100 Hz, the output “enable” (P10 of IC10) is generated by merely loading the CREG counter of the FPC with the number 99. Because the FPC Am29CPL154 contains an 8-bit CREG counter, the number 99 is loaded in one step, and then counter CREG is decremented and tested against zero. Notice that while the CREG counter is being decremented, the output named “enable” permits counters IC8a to IC8d to count the incoming pulses received at the input CLK (pin 1 of IC8a).

Table 5.2 presents the software program for the FPC Am29CPL154. The section for “comments” in the software program explains in detail the steps that are being followed by the FPC in order to drive the speech processor correctly. In this case, the FPC Am29CPL154 is capable of reading the input frequency of four BCD counters (IC8a, IC8b, IC8c, and IC8d) by driving a 3-to-8 decoder 74HC137 (IC6). IC6 is an active low decoder that selects the BCD output of each counter by applying a logic zero at the input /OC of the selected 4-bit latch (74HC173). Because the IC’s 74HC173 (IC9a to IC9d) contain tri-state outputs, bus conflicts are avoided; therefore, the four 74HC173s share the same bus that provides the frequency measurement to the FPC Am29CPL154. The FPC drives the 74HC137 by asserting a 3-bit data via the outputs P12, P13, and P14.

The next step consists of driving the GL input of IC6 to a logic high while the outputs P12 to P14 maintain the data. It is necessary to keep the input GL

**TABLE 5.2**  
Software Program for the FPC Am29CPL154

DEVICE (CPL154)

DEFAULT = 1;

DEFINE "test inputs"

intr = t6 equal = eq

"Output control bits are given name assignments"

zero = 1F#h one = 01#h two = 02#h three = 03#h

four = 04#h five = 05#h six = 06#h seven = 07#h

eight = 08#h nine = 09#h ten = 0A#h eleven = 0B#h

twelve = 0C#h thirteen = 0D#h fourteen = 0E#h fifteen = 0F#h

sixteen = 10#h seventeen = 11#h eighteen = 12#h nineteen = 13#h

twenty = 14#h thirty = 15#h forty = 16#h fifty = 17#h sixty

= 18#h

seventy = 19#h eighty = 1A#h ninety = 1B#h hundred = 1C#h

thousand = 1D#h million = 1E#h pulses = 9C#h kilo = 62#h

point = 9A#h dig1 = 1000#h wr = 5000#h and = 3C#h

reset1 = 100#h dig2 = 2000#h

reset2 = 200#h dig3 = 3000#h

cken = 400#h dig4 = 4000#h GL = 0800#h;

DEFAULT\_OUTPUT = 0000#h;

OUT\_POLARITY = F7FF#h;

TEST\_CONDITION = INTR; "Default test condition"

BEGIN

"0" zero, goto pl(n0);

"1" one, goto pl(n1);

"2" two, goto pl(n2);

"3" three, goto pl(n3);

"4" four, goto pl(n4);

"5" five, goto pl(n5);

"6" six, goto pl(n6);

"7" seven, goto pl(n7);

"8" eight, goto pl(n8);

"9" nine, goto pl(n9);

```

"  ~~~~~~"
"                                     "
"                                     "
"  ~~~~~~"

```

"10"start:reset2, call pl(count);

"11" ,cmp tm (30#h) to pl (00#h); "testing overflow"

"12" ,if (equal) then goto pl(SPHZ);

"13" reset2, call pl(count);

"14" ,cmp tm (30#h) to pl (10#h); "testing overflow"

"15" ,if (equal) then goto pl(SPKHZ);

"16" reset2, call pl(count);

"17" ,cmp tm (30#h) to pl (20#h); "testing overflow"

"18" ,if (equal) then goto pl(SPKHZ);

"19" reset2, call pl(count);

"20" ,goto pl(SPMHZ);

```

"  ~~~~~~"

```

```

"          DISPLAY FORMAT: 0000 Scale: Hz  D4 D3 D2 D1  "
"21"SPHZ: dig4, continue;
"22" dig4+GL, continue; " Digit 4 is selected by IC 74HC137"
"23" ,cmp tm(0F#h) to pl(00#h); "D4 = 0?"
"24" dig3, if (not equal) then goto pl(spKD4);
"25" dig3+GL, continue;
"26" ,cmp tm(0F#h) to pl(00#h); "D3 = 0?"
"27" dig2, if (not equal) then goto pl(spKD3);
"28" dig2+GL, continue;
"29" ,cmp tm(0F#h) to pl(00#h); "D2 = 0?"
"30" ,if (not equal) then goto pl(ptyc);
"31"spkd1: dig1, continue;
"32" dig1+GL, call pl(announ); "Announce D1 "
"33"hrtz: ,call pl(HZ); "Hertz"
"34" ,goto pl(start);
"35"spkd4: dig4, continue;
"36" dig4+GL, call pl(announ); "Announce D4"
"37" thousand, continue;
"38"thousand+wr, continue; "Thousand..."
"39"same: dig3, if (not intr) then goto pl(same);
"40" dig3+GL, continue;
"41" ,cmp tm(0F#h) to pl(00#h); "D3 = 0?"
"42" ,if (not equal) then goto pl(spKD3);
"43" dig2, continue;
"44" dig2+GL, continue;
"45" ,cmp tm(0F#h) to pl(00#h); "D2 = 0?"
"46" ,if (not equal) then goto pl(ptyand);
"47"spkand: and, continue;
"48" and+wr, continue; "And..."
"49"same2: ,if (not intr) then goto pl(same2);
"50" ,goto pl(spkd1);
"51"spkd3: dig3, continue;
"52" dig3+GL, call pl(announ); "Announce D3"
"53" hundred, continue;
"54" hundred+wr, continue; "Hundred..."
"55"same3: dig2, if (not intr) then goto pl(same3);
"56" dig2+GL, continue;
"57" ,cmp tm(0F#h) to pl(00#h); "D2 = 0?"
"58" ,if (not equal) then goto pl(ptyand);
"59" dig1, continue;
"60" dig1+GL, continue;
"61" ,cmp tm(0F#h) to pl(00#h); "D1 = 0?"
"62" ,if (not equal) then goto pl(spkanD);
"63" ,goto pl(hrtz);
"64"ptyand: and, continue;
"65" and+wr, continue;
"66"same4: dig2, if (not intr) then goto pl(same4);
"67"ptyc: dig2+GL, continue;
"68" ,cmp tm(0F#h) to pl(01#h); "D2 = 1?"
"69" ,if (not equal) then goto pl(ptyb);
"70" dig1, continue;
"71" dig1+GL, call pl(BCD4); "Announce D1"
"72" ,goto pl(hrtz);
"73"ptyb: dig2, continue; "Announce D2"

```

```

"74"  dig2+GL,    call pl(BCD3b);
"75"      dig1,    continue;
"76"  dig1+GL,    continue;
"77"      ,cmp tm(0F#h) to pl(00#h);          "D1 = 0?"
"78"      ,if (equal) then goto pl(hrtz);
"79"      ,call pl(announ);                  "Announce D1"
"80"      ,goto pl(hrtz);
"
-----"
      " DISPLAY FORMAT: 10.00   Scale: KHz   D4 D3.D2 D1"
"81"SPKHZ: dig4,    continue; "D4 is necesarily not zero"
"82"      dig4+GL,    continue; "D4 is latched"
"83"      ,cmp tm(0F#h) to pl(00#h);          "D4=0? "
"84"      ,if (not equal) then goto pl(rick5);
"85"said3: dig3,    continue;          "say D3 because D4=0"
"86"      dig3+GL,    call pl(announ); "D3 is latched"
"87"pnt:  point,    continue;          "Point.."
"88"      point+wr,    continue;
"89"idle:      ,if (not intr) then goto pl(idle);
"90"      dig2,    continue;
"91"      dig2+GL,    continue;
"92"      ,cmp tm(0F#h) to pl(00#h); "D2=0? "
"93"      ,if (not equal) then goto pl(rick2);
"94"      ,continue;
"95"      ,call pl(announ); "announce D2"
"96"pat:  dig1,    continue;
"97"      dig1+GL,    call pl(announ); "announce D1"
"98"paty:      ,call pl(KHZ); "announce KiloHertz"
"99"      ,goto pl(start);
"100"rick2: dig1,    continue;
"101"      dig1+GL,    continue;
"102"      ,cmp tm(0F#h) to pl(00#h);
"103"      ,if (not equal) then goto pl(rick3);
"104"      dig2,    continue;
"105"      dig2+GL,    call pl(BCD3);
"106"      ,goto pl(paty);
"107"rick3: dig2,    continue;
"108"      dig2+GL,    continue;
"109"      ,cmp tm(0F#h) to pl(01#h);
"110"      ,if (not equal) then goto pl(rick4);
"111"      ,call pl(BCD4);
"112"      ,goto pl(paty);
"113"rick4: dig2,    continue;
"114"      dig2+GL,    call pl(BCD3b);
"115"      ,goto pl(pat);
"116"rick5: dig4,    continue;
"117"      dig4+GL,    continue;
"118"      ,cmp tm(0F#h) to pl(01#h); "D4=1? "
"119"      ,if (not equal) then goto pl(rick6);
"120"      dig3,    continue;
"121"      dig3+GL,    call pl(BCD4); "Announce D3"
"122"      ,goto pl(pnt);
"123"rick6: dig4,    continue;

```

```

"124"      dig4+GL,      call pl(BCD3b);      "Announce D4"
"125"      dig3,        continue;
"126"      dig3+GL,     continue;
"127"      ,cmp tm(0F#h) to pl(00#h); "D3 = 0? "
"128"      ,if (not equal) then goto pl(sayd3);
"129"      ,goto pl(pnt);

"  -----"
"          DISPLAY FORMAT: 100.0   Scale: KHz      D4 D3 D2.D1      "
"130"SPKH2: dig4,        continue; "D4 is not zero"
"131"      dig4+GL,      call pl(announ); "D4 is latched"
"132"      hundred,      continue;      "Hundred..."
"133"      hundred+wr,    continue;
"134"sty5:  ,if (not intr) then goto pl(sty5);
"135"      and,          continue;
"136"      and+wr,        continue;      "And..."
"137"sty6:  ,if (not intr) then goto pl(sty6);
"138"      dig3,          continue;
"139"      dig3+GL,       continue;
"140"      ,cmp tm(0F#h) to pl(00#h); "D3=0? "
"141"      ,if (not equal) then goto pl(lug3);
"142"lug4:  dig2,          continue;
"143"      dig2+GL,       call pl(announ); "D2 is announced"
"144"lug5:  point,         continue;
"145"      point+wr,      continue;
"146"sty20: ,if(not intr) then goto pl(sty20);
"147"      dig1,          continue;
"148"      dig1+GL,       call pl(announ); "D1 is announced"
"149"      ,call pl(KHZ);
"150"      ,goto pl(start);
"151"lug3:  dig3,          continue;
"152"      dig3+GL,       continue;
"153"      ,cmp tm(0f#h) to pl(01#h);      "D3=1?"
"154"      ,if (equal) then goto pl(lug6);
"155"      ,call pl(BCD3b);
"156"      dig2,          continue;
"157"      dig2+GL,       continue;
"158"      ,cmp tm(0F#h) to pl(00#h); "D2=0?"
"159"      ,if (not equal) then goto pl(lug4);
"160"      ,goto pl(lug5);
"161"lug6:  dig2,          continue;
"162"      dig2+GL,       call pl(BCD4);
"163"      ,goto pl(lug5);

"  -----"
"          DISPLAY FORMAT: 1.000   Scale: MHz      "
"164"SPMHZ: dig4,        continue;
"165"      dig4+GL,      call pl(announ); "D4 is latched"
"166"      point,        continue;
"167"      point+wr,      continue;      "Point"
"168"sty8:  ,if (not intr) then goto pl(sty8);
"169"      dig3,          continue;
"170"      dig3+GL,       call pl(announ); "D3 is latched"
"171"      dig2,          continue;

```

```

"172"      dig2+GL,      call pl(announ); "D2 is selected"
"173"      dig1,        continue;
"174"      dig1+GL,     call pl(announ); "D1 is selected"
"175"                                     ,call pl(MHZ);
"176"                                     ,goto pl(start);

" ~~~~~"
" ***** Routine for enabling counters IC8-IC9 for 1 second ***** "
"177"count:reset1,      load pl(99);
"178"stay:cken,         while (creg <> 0) loop to pl(stay);
"179"                                     ,ret;

" ~~~~~"
"          ROUTINES BCD3 AND BCD3b          "
" ~~~~~"

"180"BCD3:      ,cmp tm(0F#h) to pl(01#h);
"181"  ten+GL,   if (equal) then goto pl(n10);
"182"BCD3b:     ,cmp tm(0F#h) to pl(02#h);
"183" twenty+GL, if (equal) then goto pl(n20);
"184"          ,cmp tm(0F#h) to pl(03#h);
"185" thirty+GL, if (equal) then goto pl(n30);
"186"          ,cmp tm(0F#h) to pl(04#h);
"187" forty+GL,  if (equal) then goto pl(n40);
"188"          ,cmp tm(0F#h) to pl(05#h);
"189" fifty+GL,  if (equal) then goto pl(n50);
"190"          ,cmp tm(0F#h) to pl(06#h);
"191" sixty+GL,  if (equal) then goto pl(n60);
"192"          ,cmp tm(0F#h) to pl(07#h);
"193" seventy+GL, if (equal) then goto pl(n70);
"194"          ,cmp tm(0F#h) to pl(08#h);
"195" eighty+GL, if (equal) then goto pl(n80);
"196" ninety+GL, goto pl(n90);
"197"n10:ten+wr,  goto pl(xb);
"198"n20:twenty+wr, goto pl(xb);
"199"n30:thirty+wr, goto pl(xb);
"200"n40:forty+wr,  goto pl(xb);
"201"n50:fifty+wr,  goto pl(xb);
"202"n60:sixty+wr,  goto pl(xb);
"203"n70:seventy+wr, goto pl(xb);
"204"n80:eighty+wr, goto pl(xb);
"205"n90:ninety+wr, goto pl(xb);
"206"xb:         ,if (not intr) then goto pl(xb);
"207"          ,ret;

" ~~~~~"
"          ROUTINE BCD4          "
" ~~~~~"

"208" BCD4:      ,cmp tm(0F#h) to pl(00#h);
"209"  ten+GL,   if (equal) then goto pl(n10);
"210"          ,cmp tm(0F#h) to pl(01#h);
"211" eleven+GL, if (equal) then goto pl(n11);
"212"          ,cmp tm(0F#h) to pl(02#h);
"213" twelve+GL, if (equal) then goto pl(n12);
"214"          ,cmp tm(0F#h) to pl(03#h);

```

```

"215"thirteen+GL, if (equal) then goto pl(n13);
"216"                ,cmp tm(0F#h) to pl(04#h);
"217"fourteen+GL, if (equal) then goto pl(n14);
"218"                ,cmp tm(0F#h) to pl(05#h);
"219"fifteen+GL, if (equal) then goto pl(n15);
"220"                ,cmp tm(0F#h) to pl(06#h);
"221"sixteen+GL, if (equal) then goto pl(n16);
"222"                ,cmp tm(0F#h) to pl(07#h);
"223"seventeen+GL, if (equal) then goto pl(n17);
"224"                ,cmp tm(0F#h) to pl(08#h);
"225"eighteen+GL, if (equal) then goto pl(n18);
"226"nineteen+GL, goto pl(n19);
"227"n11: eleven+wr,      goto pl(finish);
"228"n12: twelve+wr,      goto pl(finish);
"229"n13: thirteen+wr,    goto pl(finish);
"230"n14: fourteen+wr,    goto pl(finish);
"231"n15: fifteen+wr,     goto pl(finish);
"232"n16: sixteen+wr,     goto pl(finish);
"233"n17: seventeen+wr,   goto pl(finish);
"234"n18: eighteen+wr,    goto pl(finish);
"235"n19: nineteen+wr,    goto pl(finish);

"236"announ:              ,goto tm(0F#h);
"237"n0: zero+wr,         goto pl(finish);
"238"n1: one+wr,          goto pl(finish);
"239"n2: two+wr,          goto pl(finish);
"240"n3: three+wr,        goto pl(finish);
"241"n4: four+wr,         goto pl(finish);
"242"n5: five+wr,         goto pl(finish);
"243"n6: six+wr,          goto pl(finish);
"244"n7: seven+wr,        goto pl(finish);
"245"n8: eight+wr,        goto pl(finish);
"246"n9: nine+wr,         goto pl(finish);

"247"finish:              ,if (not intr) then goto pl(finish);
"248"                    ,ret;

" ~~~~~ "

"249"HZ: pulses,          continue;
"250"  pulses+wr,         continue;
"251"                    ,if (intr) then goto pl(stop) else wait;
"252"KHZ: kilo,           continue;
"253"  kilo+wr,           continue;
"254"                    ,if (intr) then goto pl(HZ) else wait;
"255"MHZ: million,        continue;
"256"  million+wr,        continue;
"257"                    ,if (intr) then goto pl(HZ) else wait;
"258"stop: wr,            ret;

" ~~~~~ "

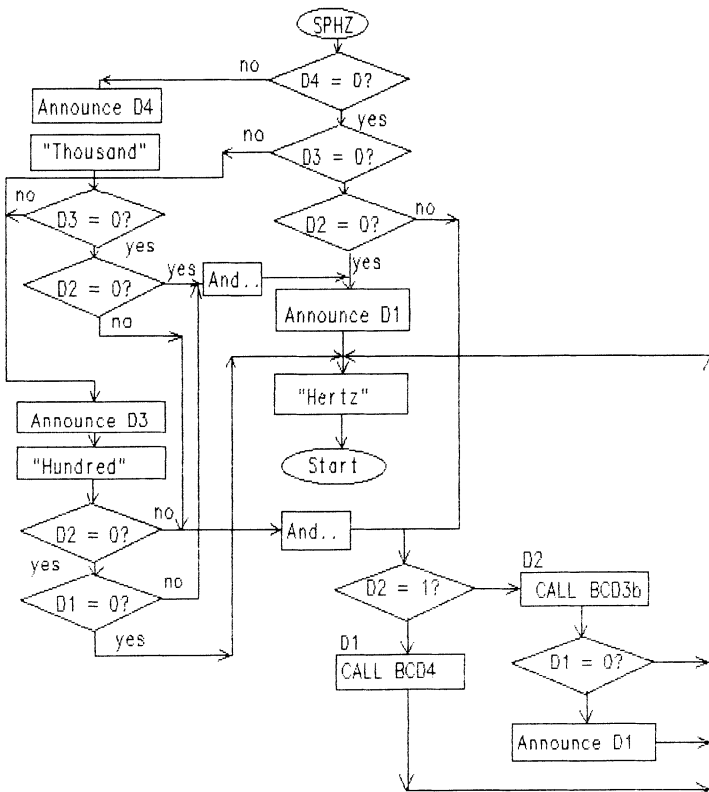
                .org 511#d
"259"                ,goto pl(start);
END.

```

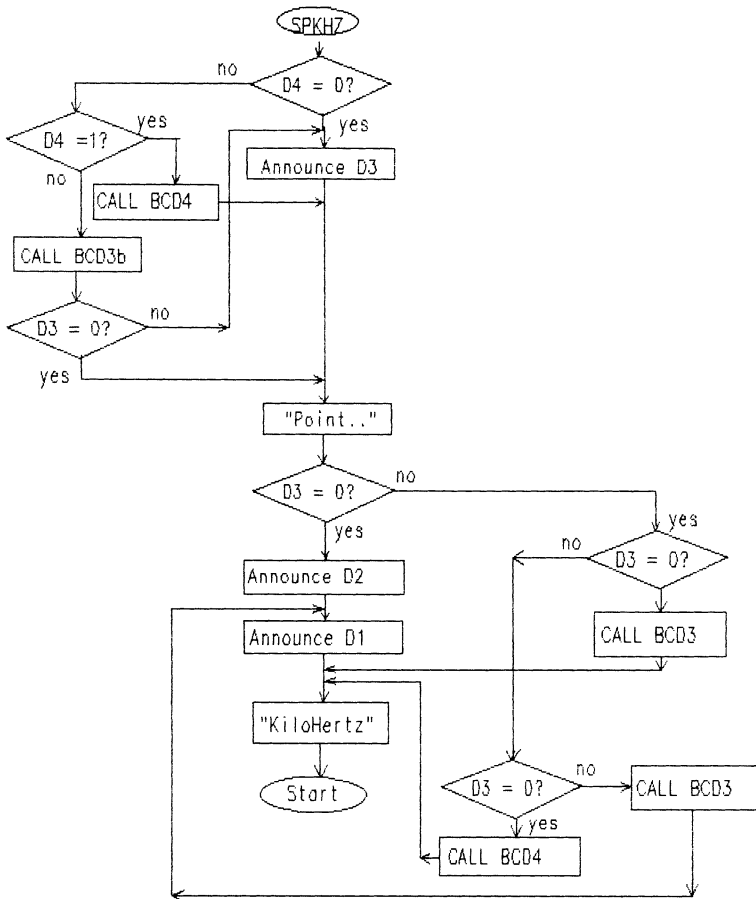


of the 74HC137 in a logic high in order to maintain the selection of one of the four 74HC173 latches. This feature permits the FPC to perform several tasks of comparison with the selected BCD digit. In this manner, the FPC selects one of the four BCD digits to execute the subroutine of comparisons in order to determine the decimal value of the selected digit. The 74HC137 (IC6) is also used to assert a logic low to the /WR input of the Digitalker via output Y5. When the 74HC137 is not working, the output Y0 (not shown) stays in a logic low while the rest of the outputs remain in a logic high.

Table 5.2 shows that the main process of the program is in lines 10 to 20 of the microcode program. The routines "SPHZ," "SPKHZ," and "SPKHZ2" required by the FPC Am29CPL154 to determine the magnitude and the scale of a frequency reading are shown in Figures 5.2, 5.3, and 5.4, respectively. The routine "SPMHz" is used for input frequencies within the range of 1.000 to 9.999 MHz. Figures 5.5 and 5.6 show the comparison subroutines that very often are used to determine the decimal value of the digit under measurement.

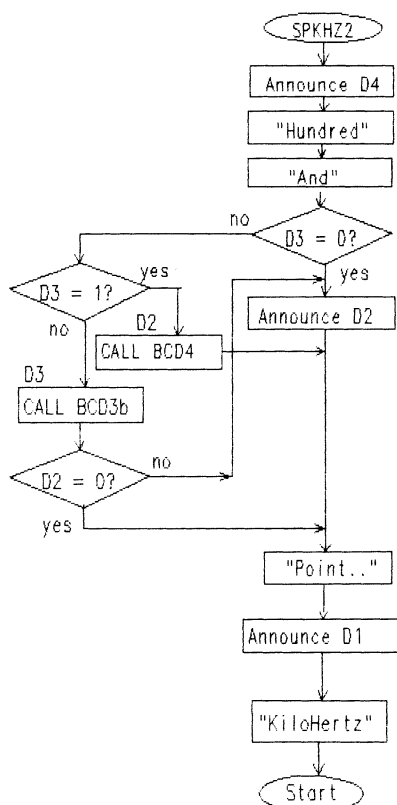


**Figure 5.2** Flowchart for subroutine "SPHZ."



**Figure 5.3** Flowchart for subroutine "SPKHZ."

To perform a complete reading with the circuitry of Figure 5.1, the process shown in Table 5.2 must take place. When the circuit is first turned on, a software reset pulse is applied with the instruction ".org 511#d" that makes the program jump to the next instruction, located in line 257. Because the default test condition is the output /INTR of the Digitalker, the instruction in line 257 makes the program jump to the instruction labeled "start," which is located in line 10. As we have seen in previous chapters, the /INTR output of the Digitalker gives a logic one when it is ready to be triggered to start saying a determined word. Output lines P0 to P7 of the FPC are used for loading the 8-bit binary address that the Digitalker requires to announce a word (see Figure 5.1.) The instruction "call pl(count)" in line 10 of Table 5.2 calls the routine named "count" located in line 177 and resets counter IC5.



**Figure 5.4** Flowchart for subroutine "SPKHZ2."

The first step that the routine "count" performs is loading the CREG counter contained in the FPC with the decimal number 99. The instruction "load pl(99)" in line 177 also clears the dual BCD counters (IC8a to IC8d) prior to starting the counting sequence. Line 178 contains the instruction "while (creg <> 0)," which keeps decrementing counter CREG of the FPC while output P10 named "enable" remains at a logic high. Output P10 enables counters IC8a to IC8d to start counting the input frequency "fi" for a period of one second. When counter CREG is zero, the program jumps to the instruction "ret" in line 179, which performs a return to line 11. The instruction "cmp tm(30#h) to pl(00#h)" is used to test if one or more overflows have occurred. The bits "A" and "B" of IC5 are routed to the testable inputs T4 and T5 of the FPC; therefore, the test mask 30#h must be used to read the binary value of T4 and T5. Inputs T4 and T5 are compared against zero in line 12. If T4 and T5 are equal to zero, the program jumps to the routine named

“SPHZ.” If T4 and T5 are not zero, the program jumps to the next instruction in line 13. The set of instructions located in lines 13 to 20 are also used to determine if more overflows occur, so the program can jump to the routine that corresponds to the scale of the frequency under measurement.

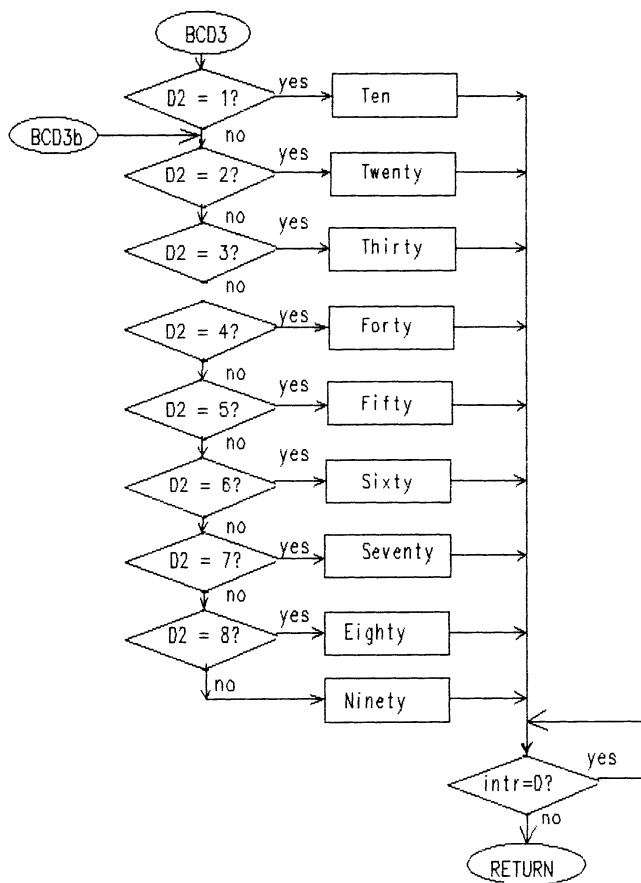
The routine “SPHZ” is used for measuring frequencies within the range of 0 to 9999 hertz. The flowchart used for developing the microcode is shown in Figure 5.2. The following two examples may be helpful in understanding the flowchart.

Suppose we are measuring a low input frequency of 8 hertz; that is, 0008 Hz. In this case, the least significant digit D1 is the only one that contains the magnitude of the reading. Routine “SPHz” starts by comparing digit D4 against zero. Since digit D4 is equal to zero, it then compares digit D3 and then digit D2. Because D3 and D2 are both equal to zero, the program reaches the box named “Announce D1.” Announcing the value of digit D1 is performed by calling the routine “announ” in line 234. The instruction “goto tm(0F#h)” in line 234 uses the inputs T0 to T3 to jump to the lines within the range of zero to nine, depending upon the value of the digit D1. As can be seen in lines zero to nine of Table 5.2, the instructions in that range contain a “goto pl(nX)” command that makes the program jump to the exact location where the speech word for the decimal number is located. In this case, digit D1 makes the program branch to label “n8” located in line 243 where the word “eight” is stored. Notice that line 8 of the program also contains the word “eight” in order to issue the speech data to the address bus of the Digtalker. In addition, when the program jumps to line 243, the word “eight” is still present while the /WR input of the Digtalker is pulsed low for 10 ms. This causes the Digtalker to issue the word “eight,” and the program will jump to line 245 in order to wait for the /INTR input to go high. When the Digtalker has finished saying the word “eight,” its /INTR output goes to a logic high, making the FPC return to line 33. The instruction in line 33 now calls subroutine “HZ,” which is used to deliver the word “hertz” to the Digtalker. Because the word “hertz” is not contained in the ROM vocabulary of the Digtalker, the word “pulses” was selected. When the program returns from routine “HZ,” it then goes to label “start” in order to initiate a new frequency measurement.

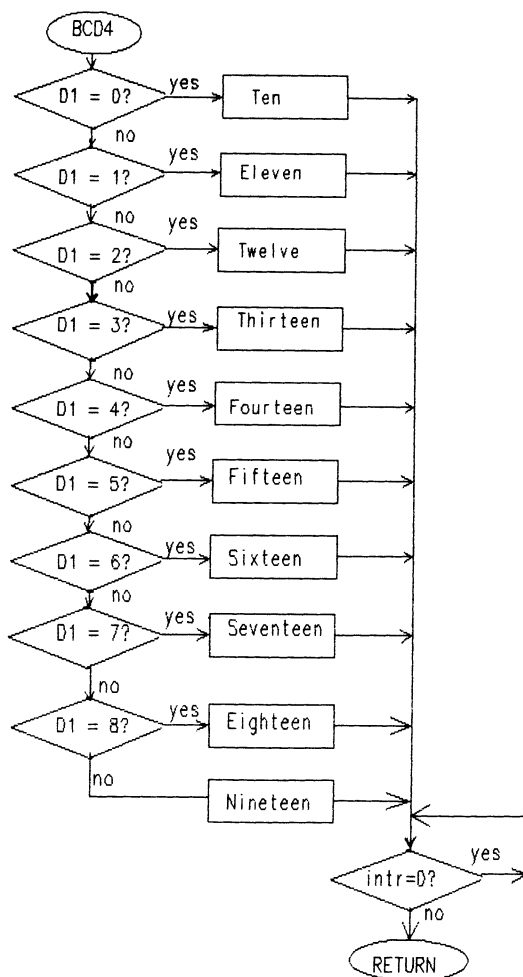
As a second example, assume that our frequency under measurement is 1500 hertz. The flowchart of Figure 5.2 will determine that D4 is not zero and that digit D4 must be announced; consequently, digit D4 is enunciated by calling subroutine “announ.” In this case, the operator will hear the word “one.” The next step is to issue the word “thousand” since digit D4 represents the thousands of hertz under measurement. The left upper corner of the flowchart (see Figure 5.2) shows that the program now has to determine if digit D3 is equal to zero. In this case, digit D3 is not zero, so the program proceeds to announce the value of digit D3, which is “five” followed by the word

“hundred.” The flowchart now indicates that the next step is to determine if D2 and D1 are equal to zero. Since D2 and D1 are both equal to zero, the program will issue the word “pulses.” Thus the operator will hear the phrase “one thousand five hundred pulses” that corresponds to the input frequency under measurement. The reader can try several values within the range of 0 to 9999 Hz with the routine “SPHZ”; it performs all possible measurements correctly.

In the same way, routine “SPKHZ” performs all possible frequency measurements within the range of 10.00 to 99.99 kHz. Notice that the decimal point between D3 and D2 is also considered in the flowchart presented in Figure 5.3. Routine “SPKHZ2” shown in Figure 5.4 gives the measurements for input frequencies within the range of 100.0 to 999.0 kHz. The routine “SPMHz” located in line 164 gives the frequency readings within the range of 1.000 to 9.999 MHz. It is a small routine because it only scans and issues the



**Figure 5.5** Flowchart for subroutine “BCD3.”



**Figure 5.6** Flowchart for subroutine BCD4.

decimal value of the four digits from left to right and adds the words “point” and “megahertz” (see lines 167 and 175, respectively).

Subroutines BCD3 and BCD4, shown in Figures 5.5 and 5.6, respectively, are frequently used by the program to solve the cases of decimal numbers formed by two numbers. For example, “twenty four” or numbers less than 20, as in the number “seventeen.”

The routine presented in this section is extremely useful for measuring variables that have to be represented in digital format. With a few variations to the section “DEFINE,” the routine shown in Table 5.2 can be easily adapted for controlling another type of speech processor for any specific application.



S1 is depressed, the monostable generates a 1.1 ms pulse that sets the Q output of flip-flop IC7 high. The resulting negative transition at the speech processor chip's /ALD input (pin 20) loads the current EPROM output and causes the processor to assert a low logic level at the /SBY output (pin 8). This action changes the IC8b output to a logic one, causing the processor to hold /SBY low for an interval appropriate to that particular allophone. Note that you must connect an audio amplifier and speaker or headphone to the output as indicated.

The processor initiates the next allophone cycle by driving /SBY high. Each audible report requires 3 to 25 allophones, which you can get from the dictionary located in Section 2 of Chapter 2. In essence, you must program the EPROM in 250 blocks of 3 to 25 bytes each.

An input of A12–A5=00000001 (corresponding to 0.1 V input), for example, produces the word “zero point one” from the audio amplifier. The allophones representing these words are stored in the EPROM as shown in Table 5.3. Table 5.3 shows that, after each report, the hex data instructions 4 and 44 internally reset the speech processor and, via the EPROM's O6 output, they reset the counter and the flip-flop as well. Table 5.3 represents only a part of the whole set of data within the ranges of 0.0 to 3.0 V and 24.9 to 25.0 V.

### 5.3 Designing a Direct Current Voltmeter<sup>1</sup> with the Digitalker Kit DT1050

The circuit presented in Figure 5.8 converts inputs of 0 to 25.5 V into a plain-English output with a resolution of 0.1 V. The voltmeter uses an MM54104 Digitalker chip from National Semiconductor as the speech synthesizer (IC5). Two ROMs (IC6 and IC7) contain in compressed form the frequency and amplitude data required for spoken expressions at 144 addressable locations. Figure 5.8 shows the external filter, audio amplifier, and the external speaker that the system requires to enunciate the voltage measurements.

Resistors R1 and R2 divide the input voltage to be measured by 10. Pressing the test switch to take a reading sends a 4 ms negative pulse to the A/D converter from the Nand gate IC1c, configured as a half-monostable. The ADC0804 A/D converter generates an 8-bit binary-coded output word, DB0 to DB7, the digital equivalent of the voltage input. These eight bits serve as the address input to pins A4 to A11 of the 27C64 EPROM (IC4). Half of a 4520 dual counter (IC2a) scans those memory locations in sequence by driving the lower address bits A0 to A3 of the EPROM. As a result, the EPROM delivers a preprogrammed sequence of five instructions to the Digitalker. After each report, Nand gates IC1a and IC1b reset both IC1a and IC2b binary counters.

<sup>1</sup>Reprinted and adapted with permission from *Electronic Design*, (Vol. 36 No. 24) 10/27/88. Copyright 1987 Penton Publishing.



**TABLE 5.3**

EPROM Program that Contains All the Speech Data for the dc Talking Voltmeter

Input voltage	Hex data	Hex data
0.0 volts	00	2B, 3C, 35, 2, 23, 35, 2D, 11, 37, 4, 44
0.1	20	2B, 3C, 35, 2, 9, 5, B, 11, 2, 2E, F, F, B, 4, 44
0.2	40	2B, 3C, 35, 2, 9, 5, B, 11, 2, D, 1F, 4, 44
0.3	60	2B, 3C, 35, 2, 9, 5, B, 11, 2, 10, E, 13, 4, 44
0.4	80	2B, 3C, 35, 2, 9, 5, B, 11, 2, 28, 28, 3A, 4, 44
0.5	A0	2B, 3C, 35, 2, 9, 5, B, 11, 2, 28, 28, 6, 23, 4, 44
0.6	C0	2B, 3C, 35, 2, 9, 5, B, 11, 2, 37, 37, C, 2, 27, 37, 4, 44
0.7	E0	2B, 3C, 35, 2, 9, 5, B, 11, 2, 37, 37, 7, 7, 23, C, B, 4, 44
0.8	100	2B, 3C, 35, 2, 9, 5, B, 11, 2, 14, 2, D, 4, 44
0.9	120	2B, 3C, 35, 2, 9, 5, B, 11, 2, 38, 18, 6, B, 4, 44
1.0	140	2E, F, F, B, 2, 23, 35, 2D, 11, 4, 44
1.1	160	2E, F, F, B, 2, 9, 5, B, 11, 2, 39, F, F, B, 4, 44
1.2	180	2E, F, F, B, 2, 9, 5, B, 11, 2, D, 1F, 4, 44
1.3	1A0	2E, F, F, B, 2, 9, 5, B, 11, 2, 10, E, 13, 4, 44
1.4	1C0	2E, F, F, B, 2, 9, 5, B, 11, 2, 28, 28, 3A, 4, 44
1.5	1E0	2E, F, F, B, 2, 9, 5, B, 11, 2, 28, 28, 6, 23, 4, 44
1.6	200	2E, F, F, B, 2, 9, 5, B, 11, 2, 37, 37, C, 2, 29, 37, 4, 44
1.7	220	2E, F, F, B, 2, 9, 5, B, 11, 2, 37, 37, 7, 7, 23, C, B, 4, 44
1.8	240	2E, F, F, B, 2, 9, 5, B, 11, 2, 14, 2D, 4, 44
1.9	260	2E, F, F, B, 2, 9, 5, B, 11, 2, 38, 18, 6, B, 4, 44
2.0	280	D, 1F, 2, 23, 35, 2D, 11, 37, 4, 44
2.1	2A0	D, 1F, 2, 9, 5, B, 11, 2, 39, F, F, B, 4, 44
2.2	2C0	D, 1F, 2, 9, 5, B, 11, 2, D, 1F, 4, 44
2.3	2E0	D, 1F, 2, 9, 5, B, 11, 2, 10, E, 13, 4, 44
2.4	300	D, 1F, 2, 9, 5, B, 11, 2, 28, 28, 3A, 4, 44
2.5	320	D, 1F, 2, 9, 5, B, 11, 2, 28, 28, 6, 23, 4, 44
2.6	340	D, 1F, 2, 9, 5, B, 11, 2, 37, 37, C, 2, 29, 37, 4, 44
2.7	360	D, 1F, 2, 9, 5, B, 11, 2, 37, 37, 7, 7, 23, C, B, 4, 44
2.8	380	D, 1F, 2, 9, 5, B, 11, 2, 14, 2D, 4, 44
2.9	3A0	D, 1F, 2, 9, 5, B, 11, 2, 38, 18, 6, B, 4, 44
3.0	3C0	10, E, 13, 2, 23, 35, 2D, 11, 37, 4, 44
.....	.....	.....
24.9	1F20	D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 3A, 2, 9, 5, B, 11, 38, 18, 6, B, 4, 44
25.0		D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 6, 23, 2, 23, 35, 2D, 11, 37, 4, 44
25.1		D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 6, 23, 2, 9, 5, B, 11, 2E, F, F, B, 4, 44
25.2		D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 6, 23, 2, 9, 5, B, 11, D, 1F, 4, 44
25.3		D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 6, 23, 2, 9, 5, B, 11, 10, E, 13, 4, 44
25.4		D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 6, 23, 2, 9, 5, B, 11, 28, 28, 3A, 4, 44
25.5		D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 6, 23, 2, 9, 5, B, 11, 28, 28, 6, 23, 4, 44



**TABLE 5.4**  
EPROM Program for the 255 Blocks of IC4

Hex add	Hex data	Message
0	1F, 7A, 1F, 8E, 81	zero point zero volts
10	1F, 7A, 1, 8E, 81	zero point one volts
20	1F, 7A, 2, 8E, 81	zero point two volts
30	1F, 7A, 3, 8E, 81	zero point three volts
40	1F, 7A, 4, 8E, 81	zero point four volts
50	1F, 7A, 5, 8E, 81	zero point five volts
60	1F, 7A, 6, 8E, 81	zero point six volts
70	1F, 7A, 7, 8E, 81	zero point seven volts
80	1F, 7A, 8, 8E, 81	zero point eight volts
90	1F, 7A, 9, 8E, 81	zero point nine volts
A0	1, 7A, 1F, 8E, 81	one point zero volts
B0	1, 7A, 1, 8E, 81	one point one volts
C0	1, 7A, 2, 8E, 81	one point two volts
D0	1, 7A, 3, 8E, 81	one point three volts
E0	1, 7A, 4, 8E, 81	one point four volts
F0	1, 7A, 5, 8E, 81	one point five volts
100	1, 7A, 6, 8E, 81	one point six volts
110	1, 7A, 7, 8E, 81	one point seven volts
120	1, 7A, 8, 8E, 81	one point eight volts
130	1, 7A, 9, 8E, 81	one point nine volts
140	2, 7A, 1F, 8E, 81	two point zero volts
.....		
FA0	19, 5, 7A, 1F, 8E	twenty five point zero volts
FBO	19, 5, 7A, 1, 8E	twenty five point one volts
FC0	19, 5, 7A, 2, 8E	twenty five point two volts
FDO	19, 5, 7A, 3, 8E	twenty five point three volts
FEO	19, 5, 7A, 4, 8E	twenty five point four volts
FF0	19, 5, 7A, 5, 8E	twenty five point five volts

also enables Nand gate IC1d to load that word into the Digitalker. This cycling continues until IC1a and IC1b reset both IC2 counter halves, preparing the circuit for another reading.

**5.4** Using a Field Programmable  
Controller to Design a Compact  
Aurorange Direct Current Voltmeter

The circuit shown in Figure 5.9 measures a dc input voltage within the range of 0 to 1.999 V. An automatic floating point can be added by controlling a switching network for the input voltage that also indicates the scale to the FPC Am29CPL154. Notice that the talking dc voltmeter does not contain a digital



The time base section is formed by IC7. The oscillator/divider MM5369EST (IC7) uses a 3.57 MHz crystal in order to give a stable 100 Hz output frequency, which feeds the FPC Am29CPL154 (IC10). The FPC (IC10) is programmed to control the “initiate conversion” input of A/D TSC8750. The FPC is also used to enable the input voltage under measurement received by the TSC8750 (see Figure 5.9). To perform a voltage reading, the FPC has to know if an overflow has occurred in order to select the next higher scale. If a new scale is selected, the FPC announces the voltage in the scale of 0 to 20 V. If no overflow occurs in the new selected scale, the FPC reads the measured voltage through the testable inputs T0 to T3. This way, the FPC starts the routine for driving the speech processor Digitalker (MM54104). The Digi-talker kit was selected for this application because it contains a vocabulary with numbers and words that are used for voltage measurements.

We will now examine the operation of the entire circuit of Figure 5.9 in more detail. When the circuit is first turned on, the FPC Am29CPL154 resets PAL20R4, which is the chip that registers the overflows that may occur when the selected input voltage causes an overflow. The FPC then proceeds to clear the PAL20R4 by sending a positive transient pulse via output P15. In this manner, the FPC resets the PAL prior to starting a voltage reading. Now, the FPC enables the input Vin to the ADC TSC8750 in order to determine its magnitude. When the ADC TSC8750 ends the conversion process in 10 ms, the FPC checks if an overflow has occurred by reading output Z of the PAL16R4.

If the testable input T4 is in a logic high, the FPC selects the next higher scale and starts a new voltage measurement by enabling the input that contains the 200K resistor. Now the TSC8750 will be measuring voltages within the range of 0 to 19.99 V. This scale is selected by the FPC by sending a logic high to the analog selector CD4051. If no overflow occurs in this scale, the FPC will proceed to read the measured voltage through the inputs T0 to T3. Then the FPC will perform the routine for vocalizing the voltage reading.

Table 5.5 shows the three cases that might occur when the input signal under measurement contains a magnitude that ranges from 0 to 20 V.

Table 5.5 shows three cases that represent all possible ranges for the input

**TABLE 5.5**  
Input Voltages for the Autorange DC Voltmeter

Voltage range	Input voltage	Resistor selected	Display reading	P8	P9	P10
0-199.9 mV	100.1 mV	20 K	1 0 0.1	1	0	0
0-1.999 V	1.125 V	200K	1.1 2 5	0	1	0
0-19.99 V	5.150 V	2 M	5.1 5 0	0	0	1

voltage under measurement. In the first row, the dc voltmeter accepts up to 199.9 mV via the 20K resistor. Notice that the FPC selects the input voltage “Vin” with outputs P8, P9, and P10 equal to 100, respectively. If the first overflow occurs, the FPC will select the second resistor of 200K. With the 200K resistor selected, the FPC will start a new conversion process in order to get the new reading in the voltage scale. The FPC will detect the change of scale by reading the input T4. If a second overflow occurs, the FPC will change the input node from 200K to 2M. Once the 2M resistor is selected, the ADC TSC8750 will be capable of performing voltage readings within the range of 0 to 19.99 V. If the input voltage under measurement, for example, is equal to or higher than 19.99 V, the FPC will make the speech processor announce the word “over range.” After that message, the FPC will start a new voltage reading. The logical values of the output “Z” of the PAL will serve to indicate to the FPC (IC10) the scale of the reading. This way, the speech processor can speak a voltage reading in millivolts or volts.

Table 5.6 shows the microcode program for the FPC Am29CPL154. As shown in Figure 5.9, the FPC is clocked by a 100 Hz frequency that is generated by IC7 and associated components. This means that the FPC will perform each instruction in 10 ms (10,000  $\mu$ s); therefore, the FPC will issue output pulses with 10 ms of duration. The output pulses are named “convers” and “GL.”

Table 5.6 presents the software program for the FPC Am29CPL154. The “comments” section in the software program explains in detail the steps followed by the FPC in order to drive the speech processor correctly. In this case, the FPC Am29CPL154 is capable of reading the input voltage of the 3.5 digits parallel BCD converter by driving a 3-to-8 decoder 74HC137 (IC6). IC6 is an active low decoder that selects the BCD output of each counter by applying a logic zero at the input /OC of the selected 4-bit latch (74HC173). Because the IC’s 74HC173 contain tri-state outputs, bus conflicts are avoided; therefore, the four 74HC173s share the same bus that provides the digital voltage reading to the FPC Am29CPL154. The FPC drives the 74HC137 by asserting a 3-bit data via outputs P12, P13, and P14. The next step consists of driving the GL input to a logic high while the outputs P12 to P14 maintain the data. It is necessary to keep the input GL of the 74HC137 in a logic high in order to maintain the selection of one of the four latches 74HC173. This feature permits the FPC to perform several comparisons with the selected BCD digit. In this manner, the FPC selects one of the four BCD digits to execute the subroutine of comparisons to determine the decimal value of the selected digit. The 74HC137 (IC6) is also used to assert a logic low to the /WR input of the Digtalker via output Y5. When the 74HC137 is not working, the output Y0 stays in a logic low while the rest of the outputs remain in a logic high.

Table 5.6 shows that the main process of the program is in lines 10 to 20 of the microcode program. The routines “SPMV,” “SPVOLT,” and

**TABLE 5.6**  
Software Program for the FPC Am29CPL154

DEVICE (CPL154)

DEFAULT = 1;

DEFINE "test inputs"

hrange = t4 busy = t7 intr = t6 equal = eq

"Output control bits are given name assignments"

zero = 1F#h one = 01#h two = 02#h three = 03#h

four = 04#h five = 05#h six = 06#h seven = 07#h

eight = 08#h nine = 09#h ten = 0A#h eleven = 0B#h

twelve = 0C#h thirteen = 0D#h fourteen = 0E#h fifteen = 0F#h

sixteen = 10#h seventeen = 11#h eighteen = 12#h nineteen = 13#h

twenty = 14#h thirty = 15#h forty = 16#h fifty = 17#h sixty  
= 18#h

seventy = 19#h eighty = 1A#h ninety = 1B#h hundred = 1C#h

thousand = 1D#h milli = 6C#h volt = 8E#h ss = 81#h kilo = 62#h

over = 75#h point = 9A#h wr = 5000#h and = 3C#h

rstPAL = 8000#h "P15" dig1 = 1000#h "P12" convrs = 400#h

scale1 = 0000#h dig2 = 2000#h "P13"

scale2 = 0100#h "P8" dig3 = 3000#h "P12+P13" GL = 0800#h

scale3 = 0300#h dig4 = 4000#h; "P14"

DEFAULT\_OUTPUT = 0000#h;

OUT\_POLARITY = 1111011111111111#b; "GL is high when not specified"

TEST\_CONDITION = INTR; "Default test condition"

BEGIN

"0" zero, goto pl(n0);

"1" one, goto pl(n1);

"2" two, goto pl(n2);

"3" three, goto pl(n3);

"4" four, goto pl(n4);

"5" five, goto pl(n5);

"6" six, goto pl(n6);

"7" seven, goto pl(n7);

"8" eight, goto pl(n8);

"9" nine, goto pl(n9);

" ~~~~~ "

" MAIN PROCESS "

" ~~~~~ "

"10"start:rstPAL+scale1, call pl(voltage);

"11" ,if (not hrange) then goto pl(SPmV); "over range?"

"12" rstPAL, continue;

"13" ,call pl(voltage2);

"14" ,if (not hrange) then goto pl(SPVOLT1); "over range?"

"15" rstPAL, continue;

"16" ,call pl(voltage3);

"17" rstPAL, continue;

"18" ,if (not hrange) then goto pl(SPVOLT2); "over range?"

"19" ,call pl(msgerr);

```

"20"                ,goto pl(start);
"
" ----- "
"  DISPLAY FORMAT: 10.00  Scale: Volts  (0-20V)  D4 D3.D2 D1"
"21"SPVOLT2: dig4,      continue; "D4 is necessarily not zero"
"22"      dig4+GL,      continue; "D4 is latched"
"23"                ,cmp tm(0F#h) to pl(00#h);      "D4=0? "
"24"                ,if (not equal) then goto pl(rick5);
"25"sayd3: dig3,        continue;      "say D3 because D4=0"
"26"      dig3+GL,      call pl(announ); "D3 is latched"
"27"pnt:   point,       continue;      "Point.."
"28"      point+wr,     continue;
"29"idle:                ,if (not intr) then goto pl(idle);
"30"      dig2,         continue;
"31"      dig2+GL,      continue;
"32"                ,cmp tm(0F#h) to pl(00#h);      "D2=0? "
"33"                ,if (not equal) then goto pl(rick2);
"34"                ,continue;
"35"                ,call pl(announ); "announce D2"
"36"pat:   dig1,        continue;
"37"      dig1+GL,      call pl(announ); "announce D1"
"38"paty:                ,call pl(HZ);      "Volts"
"39"                ,goto pl(start);
"40"rick2: dig1,        continue;
"41"      dig1+GL,      continue;
"42"                ,cmp tm(0F#h) to pl(00#h);
"43"                ,if (not equal) then goto pl(rick3);
"44"      dig2,         continue;
"45"      dig2+GL,      call pl(BCD3);
"46"                ,goto pl(paty);
"47"rick3: dig2,        continue;
"48"      dig2+GL,      continue;
"49"                ,cmp tm(0F#h) to pl(01#h);
"50"                ,if (not equal) then goto pl(rick4);
"51"                ,call pl(BCD4);
"52"                ,goto pl(paty);
"53"rick4: dig2,        continue;
"54"      dig2+GL,      call pl(BCD3b);
"55"                ,goto pl(pat);
"56"rick5: dig4,        continue;
"57"      dig4+GL,      continue;
"58"                ,cmp tm(0F#h) to pl(01#h);      "D4=1? "
"59"                ,if (not equal) then goto pl(rick6);
"60"      dig3,         continue;
"61"      dig3+GL,      call pl(BCD4);      "Announce D3"
"62"                ,goto pl(pnt);
"63"rick6: dig4,        continue;
"64"      dig4+GL,      call pl(BCD3b);      "Announce D4"
"65"      dig3,         continue;
"66"      dig3+GL,      continue;
"67"                ,cmp tm(0F#h) to pl(00#h);      "D3 = 0? "
"68"                ,if (not equal) then goto pl(sayd3);
"69"                ,goto pl(pnt);
" ----- "

```



```

" DISPLAY FORMAT: 000.0 - 199.9   Scale: mV (0-200 mV) D4 D3 D2.D1  "
"70"SPmV: dig4,                continue; "D4 is not zero"
"71"    dig4+GL,                call pl(announ); "D4 is latched"
"72"    hundred,                continue;      "Hundred..."
"73"    hundred+wr,             continue;
"74"sty5:                      ,if (not intr) then goto pl(sty5);
"75"    and,                    continue;
"76"    and+wr,                 continue;      "And..."
"77"sty6:                      ,if (not intr) then goto pl(sty6);
"78"    dig3,                   continue;
"79"    dig3+GL,                continue;
"80"                                ,cmp tm(0F#h) to pl(00#h); "D3=0?"
"81"                                ,if (not equal) then goto pl(lug3);
"82"lug4: dig2,                 continue;
"83"    dig2+GL,                call pl(announ); "D2 is announced"
"84"lug5: point,                continue;
"85"    point+wr,               continue;
"86"sty20:                     ,if(not intr) then goto pl(sty20);
"87"    dig1,                   continue;
"88"    dig1+GL,                call pl(announ); "D1 is announced"
"89"                                ,call pl(KHZ);      "Millivolts"
"90"                                ,goto pl(start);
"91"lug3: dig3,                 continue;
"92"    dig3+GL,                continue;
"93"                                ,cmp tm(0f#h) to pl(01#h);      "D3=1?"
"94"                                ,if (equal) then goto pl(lug6);
"95"                                ,call pl(BCD3b);
"96"    dig2,                   continue;
"97"    dig2+GL,                continue;
"98"                                ,cmp tm(0F#h) to pl(00#h); "D2=0?"
"99"                                ,if (not equal) then goto pl(lug4);
"100"                               ,goto pl(lug5);
"101"lug6: dig2,                 continue;
"102"    dig2+GL,                call pl(BCD4);
"103"                               ,goto pl(lug5);
"
-----"
" DISPLAY FORMAT: 1.000 - 1.999   Scale: Volts (0-2V) D4.D3D2D1"
"104"SPVOLT1: dig4,              continue;
"105"    dig4+GL,                call pl(announ); "D4 is latched"
"106"    point,                  continue;
"107"    point+wr,               continue;      "Point"
"108"sty8:                      ,if (not intr) then goto pl(sty8);
"109"    dig3,                   continue;
"110"    dig3+GL,                call pl(announ); "D3 is latched"
"111"    dig2,                   continue;
"112"    dig2+GL,                call pl(announ); "D2 is selected"
"113"    dig1,                   continue;
"114"    dig1+GL,                call pl(announ); "D1 is selected"
"115"                                ,call pl(HZ);
"116"                               ,goto pl(start);

```

```

" -----"
" ***** Routine to initiate conversion process in scale1  "
"117"voltage: convrs+scale1,      continue;
"118"stay: scale1,                if (busy) then goto pl(stay);
"119"                             ,ret;
" -----"
"          ROUTINES BCD3 AND BCD3b          "
" -----"

"120"BCD3:      ,cmp tm(0F#h) to pl(01#h);
"121"  ten+GL,   if (equal) then goto pl(n10);
"122"BCD3b:     ,cmp tm(0F#h) to pl(02#h);
"123"  twenty+GL, if (equal) then goto pl(n20);
"124"           ,cmp tm(0F#h) to pl(03#h);
"125"  thirty+GL, if (equal) then goto pl(n30);
"126"           ,cmp tm(0F#h) to pl(04#h);
"127"  forty+GL,  if (equal) then goto pl(n40);
"128"           ,cmp tm(0F#h) to pl(05#h);
"129"  fifty+GL,  if (equal) then goto pl(n50);
"130"           ,cmp tm(0F#h) to pl(06#h);
"131"  sixty+GL,  if (equal) then goto pl(n60);
"132"           ,cmp tm(0F#h) to pl(07#h);
"133"seventy+GL,  if (equal) then goto pl(n70);
"134"           ,cmp tm(0F#h) to pl(08#h);
"135"  eighty+GL, if (equal) then goto pl(n80);
"136"  ninety+GL, goto pl(n90);
"137"n10: ten+wr,  goto pl(xb);
"138"n20: twenty+wr, goto pl(xb);
"139"n30: thirty+wr, goto pl(xb);
"140"n40: forty+wr,  goto pl(xb);
"141"n50: fifty+wr,  goto pl(xb);
"142"n60: sixty+wr,  goto pl(xb);
"143"n70: seventy+wr, goto pl(xb);
"144"n80: eighty+wr, goto pl(xb);
"145"n90: ninety+wr, goto pl(xb);
"146"xb:           ,if (not intr) then goto pl(xb);
"147"              ,ret;
" -----"
"          ROUTINE BCD4          "
" -----"

"148" BCD4:      ,cmp tm(0F#h) to pl(00#h);
"149"  ten+GL,    if (equal) then goto pl(n10);
"150"           ,cmp tm(0F#h) to pl(01#h);
"151"  eleven+GL, if (equal) then goto pl(n11);
"152"           ,cmp tm(0F#h) to pl(02#h);
"153"  twelve+GL, if (equal) then goto pl(n12);
"154"           ,cmp tm(0F#h) to pl(03#h);
"155"thirteen+GL, if (equal) then goto pl(n13);
"156"           ,cmp tm(0F#h) to pl(04#h);
"157"fourteen+GL, if (equal) then goto pl(n14);
"158"           ,cmp tm(0F#h) to pl(05#h);

```

```

"159"fifteen+GL, if (equal) then goto pl(n15);
"160"           ,cmp tm(0F#h) to pl(06#h);
"161"sixteen+GL, if (equal) then goto pl(n16);
"162"           ,cmp tm(0F#h) to pl(07#h);
"163"seventeen+GL,if (equal) then goto pl(n17);
"164"           ,cmp tm(0F#h) to pl(08#h);
"165"eighteen+GL, if (equal) then goto pl(n18);
"166"nineteen+GL, goto pl(n19);
"167"n11:eleven+wr,      goto pl(finish);
"168"n12:twelve+wr,      goto pl(finish);
"169"n13:thirteen+wr,   goto pl(finish);
"170"n14:fourteen+wr,    goto pl(finish);
"171"n15:fifteen+wr,     goto pl(finish);
"172"n16:sixteen+wr,     goto pl(finish);
"173"n17:seventeen+wr,   goto pl(finish);
"174"n18:eighteen+wr,    goto pl(finish);
"175"n19:nineteen+wr,    goto pl(finish);

"176"announ:             ,goto tm(0F#h);
"177"n0:zero+wr,         goto pl(finish);
"178"n1:one+wr,          goto pl(finish);
"179"n2:two+wr,          goto pl(finish);
"180"n3:three+wr,        goto pl(finish);
"181"n4:four+wr,         goto pl(finish);
"182"n5:five+wr,         goto pl(finish);
"183"n6:six+wr,          goto pl(finish);
"184"n7:seven+wr,        goto pl(finish);
"185"n8:eight+wr,        goto pl(finish);
"186"n9:nine+wr,         goto pl(finish);

"187"finish:             ,if (not intr) then goto pl(finish);
"188"                     ,ret;

" ~~~~~~ "

"189"HZ:volt,            continue;
"190"  volt+wr,          continue;          "VOLT..."
"191"                   ,if (intr) then goto pl(ssa) else wait;
"192"ssa:ss,             continue;          "S..."
"193"  ss+wr,            continue;
"194"                   ,if (intr) then goto pl(stop) else wait;
"195"KHZ:milli,          continue;
"196"  milli+wr,         continue;
"197"                   ,if (intr) then goto pl(HZ) else wait;
"198"stop:wr,            ret;
"199"msgerr:over,        continue;          "OVER..."
"200"  over+wr,          continue;
"201"                   ,if (intr) then goto pl(HZ) else wait;

" ~~~~~~ "

" ***** Routine to initiate conversion process in scale2 "
"202"voltage2:convrs+scale2, continue;
"203"acctnt: scale2,      if (busy) then goto pl(accnt);
"204"                   ,ret;

" ----- "

```

```

" ***** Routine to initiate conversion process in scale3 "
"205"voltage3:convrs+scale3,    continue;
"206"acnt3: scale3,              if (busy) then goto pl(acnt3);
"207"                            ,ret;
" ----- "

                                .org 511#d
"208"                            ,goto pl(start);
END.

```

“SPVOLTS2” are used by the FPC Am29CPL154 to determine the magnitude and the scale of a frequency reading. Routine “SPMV” is used for an input voltage within the range of 0.000 to 199.9 mV.

To perform a complete voltage reading with the circuitry of Figure 5.9, the process shown in Table 5.6 must take place. When the circuit is first turned on, a software reset pulse is applied with the instruction “.org 511#d,” which makes the program jump to the next instruction, located in line 257. Because the default test condition is the output /INTR of the Digitalter, the instruction in line 257 makes the program jump to the instruction labeled “start,” located in line 10. As we have seen in previous chapters, the /INTR output of the Digitalter gives a logic one when it is ready to be triggered in order to start saying a determined word. Output lines P0 to P7 of the FPC are used for loading the 8-bit binary address that the Digitalter requires to announce a word (see Figure 5.9). The instruction “call pl(voltage)” in line 10 of Table 5.6 calls the routine “voltage” located in line 177 and resets the PAL16R4. The first step that this routine performs is asserting a logic high pulse via the output P15 to the “initiate conversion” input of the ADC TSC8750. The instruction “if (busy) then goto pl(stay)” in line 177 waits for the output “BUSY” (line 22 of IC1) to go high prior to starting a new voltage reading.

The instruction “ret” in line 179 performs a return to line 11. The instruction “cmp tm(30#h) to pl(00#h)” is used to test if one or more overflows have occurred. The bit named “A” of IC2 is routed to the testable input T4 of the FPC; therefore, the test mask 10#h must be used to read the binary value of T4. Input T4 is compared against one in line 12; if T4 is equal to zero, the program jumps to the routine “SPHZ.” On the other hand, if T4 is different from zero the program jumps to the next instruction, located in line 13. The set of instructions located in lines 13 to 20 are also used to determine if more overflows occur in order to send the program to the routine that corresponds to the scale of the input voltage under measurement.

In the same way, the routine “SPmV” solves all the possible voltage measurements within the range of 0 to 199.9 mV. Notice that the decimal point between D3 and D2 is considered in that routine. Routine “SPVOLT1” shown

in Table 5.6 gives the measurements for input voltages within the range 0 to 1.999 V. Routine “SPVOLT2” located in line 164 gives the digital voltage readings within the range of 0 to 19.99 V. It is a small routine because it only scans and issues the decimal value of the four digits from left to right and adds the words “point” and “volts” (see lines 167 and 175, respectively).

Subroutines BCD3 and BCD4 are frequently used by the complete program for decimal numbers that are formed by two numbers—for example “twenty four”—or that are less than 20, as in the number “seventeen.”

## 5.5 Designing a Circuit to Announce<sup>2</sup> Alternating Current Line Voltage

With a few passive components for monitoring ac voltages and a speech processing chip, a circuit can announce the measured voltage of ac lines. The range of the ac-voltage monitor presented in this section is 100 to 140 Vac, with a resolution of 1 V. The speech processor (SPO256-AL2) interprets an 8-bit binary input code from an A/D converter. The processor’s pulse-code modulated output then passes through a filter and amplifier before driving the circuit’s speaker to vocalize the corresponding number (see Figure 5.10).

In this application, the allophone-based speech processor requires 20 to 31 allophones for each audible report. Each time switch S1 is pressed, the speech processor program enunciates the monitored voltage readings from 100 to 140 Vac, depending on the code at the input of a 27C64 EPROM (see Table 5.7). For an input voltage of 120 Vac, for example, the speaker announces “one hundred and twenty volts.”

The voltage-monitoring circuit consists of a bridge rectifier, filter capacitors, and a 10K load resistor. A divider,  $R_a$  and  $R_b$ , limits the input voltage to a maximum 2.55 V. The A/D converter, IC4, then sends the voltage reading to the 27C64 EPROM (IC5).

Pressing S1 sends a negative transient pulse to the (/WR) input of the A/D converter, IC4, which has a 100  $\mu$ s conversion process and generates an interrupt (/INTR) high output during the process. The resulting binary-coded reading then, latched in the A/D converter, supplies the EPROM’s current upper-address inputs A5 through A12, which select a block of memory within the EPROM. Next, a CD4520 counter (IC3) scans those memory locations in sequence by driving the lower address bits A0 through A4.

As a result, the EPROM delivers a preprogrammed sequence of instructions to the speech processor. Hexadecimal data instructions, 4H and 44H, at the end of each program line supply a reset signal from the EPROM output O6

<sup>2</sup>Reprinted and adapted with permission from *Electronic Design*, (Vol. 37 No. 2) January 26, 1989. Copyright 1989 Penton Publishing.



DA0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 14, 2, D, 23, 35, 2D, D, 37, 4, 44
DC0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 14, 2, D, 23, 35, 2D, D, 37, 4, 44,
DE0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 38, 18, 6, 23, 35, 2D, D, 37, 4, 44
E00	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 7, 7, B, 23, 35, 2D, D, 37, 4, 44
E20	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, C, 2D, 7, 7, 23, C, B, 23, 35, 2D, D, 37, 4, 44
E40	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, 2D, 23, 23, 35, 2D, D, 37, 4, 44
E60	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, 2D, 23, 23, 35, 2D, D, 37, 4, 44
E80	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 13, 47, D, 13, B, 23, 35, 2D, D, 37, 4, 44
EA0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 28, 28, 3A, D, 13, B, 23, 35, 2D, D, 37, 4, 44
EC0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 28, C, 28, D, 13, B, 23, 35, 2D, D, 37, 4, 44
EE0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 37, 37, C, 29, 37, D, 13, B, 23, 35, 2D, D, 37, 4, 44
F00	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 37, 37, C, 29, 37, D, 13, B, 23, 35, 2D, D, 37, 4, 44
F20	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 37, 37, 7, 23, C, B, D, 13, B, 23, 35, 2D, D, 37, 4, 44
F40	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 14, 2, D, 13, B, 23, 35, 2D, D, 37, 4, 44
F60	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 14, 2, D, 13, B, 23, 35, 2D, D, 37, 4, 44
F80	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, B, 6, B, D, 13, B, 23, 35, 2D, D, 37, 4, 44
FA0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 23, 35, 2D, D, 37, 4, 44
FC0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 2E, F, F, B, 23, 35, 2D, D, 37, 4, 44
FE0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, D, 1F, 23, 35, 2D, D, 37, 4, 44
1000	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, D, 1F, 23, 35, 2D, D, 37, 4, 44
1020	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 1D, E, 13, 23, 35, 2D, D, 37, 4, 44
1040	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 1D, E, 13, 23, 35, 2D, D, 37, 4, 44
1060	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 28, 28, 3A, 23, 35, 2D, D, 37, 4, 44
1080	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 28, 28, 6, 23, 4, 44
10A0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 37, 37, C, 29, 37, 23, 35, 2D, D, 37, 4, 44
10C0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 37, 37, C, 29, 37, 23, 35, 21, D, 37, 4, 44
10E0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 37, 37, 7, 7, 23, C, B, 4, 44
1100	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 14, 2, D, 23, 35, 2D, D, 37, 4, 44
1120	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, D, 30, 7, 7, B, D, 13, 38, 18, 6, B, 23, 35, 2D, D, 37, 4, 44

1140	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 23, 35, 2D, D, 37, 4, 44
1160	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 2E, F, F, B, 23, 35, 2D, D, 37, 4, 44
1180	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 2E, F, F, B, 23, 35, 2D, D, 37, 4, 44
11A0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 2E, F, F, B, 23, 35, 2D, D, 37, 4, 44
11C0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 1D, E, 13, 23, 35, 2D, D, 37, 4, 44
11E0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 28, 28, 3A, 23, 35, 2D, D, 37, 4, 44
1200	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 28, 28, 3A, 23, 35, 2D, D, 37, 4, 44
1220	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 28, 28, 6, 23, 23, 35, 2D, D, 37, 4, 44
1240	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 28, 28, 6, 23, 23, 35, 2D, D, 37, 4, 44
1260	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 37, 37, C, 29, 37, 23, 35, 2D, D, 37, 4, 44
1280	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 37, 37, 7, 7, C, B, 23, 35, 2D, D, 37, 4, 44
12A0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 14, 2, D, 23, 35, 2D, 37, 4, 44
12C0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 14, 2, D, 23, 35, 2D, 37, 4, 44
12E0	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 1D, 34, 2, D, 13, 38, 18, 6, B, 23, 35, 2D, D, 37, 4, 44
1300	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 28, 3A, 2, D, 13, 23, 35, 2D, D, 37, 4, 44
1320	2E, F, F, B, 39, F, F, B, 21, 27, C, C, 15, 28, 3A, 2, D, 13, 23, 35, 2D, D, 37, 4, 44

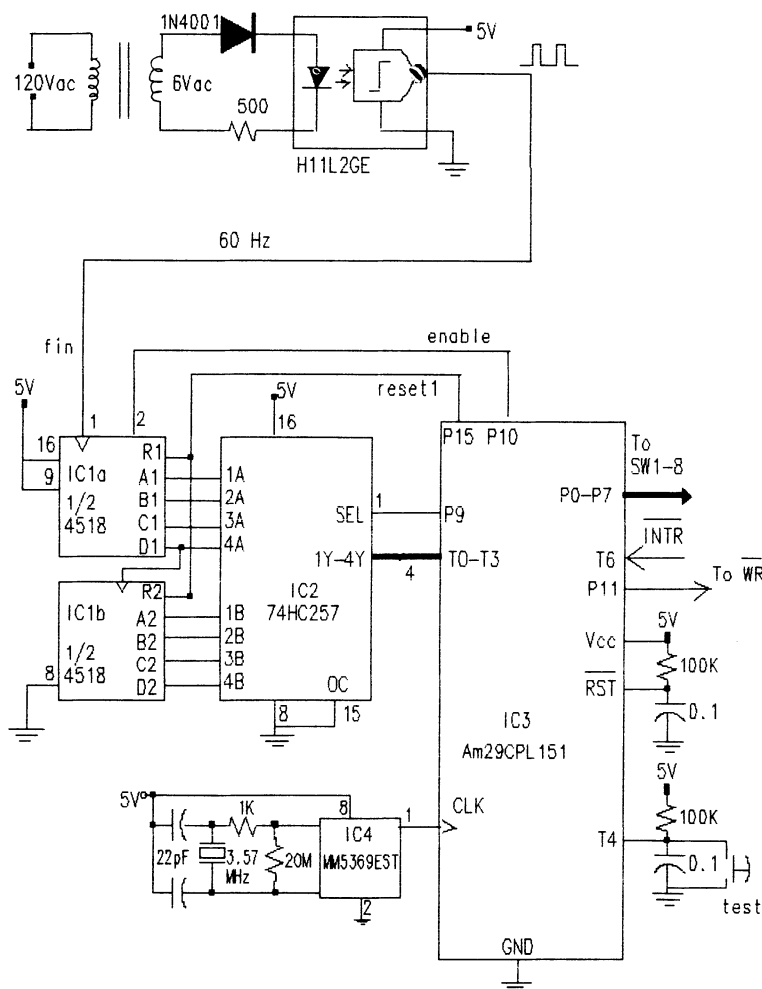
for the counter and flip-flop IC7. At the completion of the conversion, the /INTR output returns to logic zero, and the half-monostable formed by IC2c sends a positive transient pulse that latches IC7's Q output high. The IC7 output enables the IC2d Nand gate.

If the processor is not working, its standby output (/SBY) is high. Consequently, IC2d's Nand output drives the speech processor's address-load input (/ALD) low to start an allophone cycle. During this cycle, the processing chip holds its /SBY output low for an interval appropriate to that particular allophone. Each logic-low /SBY transition advances counter IC3 one count through the Nand gate (starting from zero, following closure of S1). When the speech processor finishes its report and resets, /SBY returns high again, and the Nand gate output goes low, causing the /ALD to load the next EPROM output into the speech processor and initiate the next allophone cycle.



### 5.6 Designing a Circuit to Announce Alternating Current Line Frequency Cycles

The circuit shown in Figure 5.11 measures the frequency of the ac line to which it is connected. A 120 Vac to 6 Vac transformer with a maximum output of 500 mA is utilized to send the positive cycle of the ac line frequency. The negative cycle is clamped by a 1N4001 switching diode. The 500 ohm resistor limits the current through the LED contained in the logic optocoupler with a



**Figure 5.11** Schematic for the speech synthesized alternating current line frequency meter.

TABLE 5.8			
Input Frequencies for AC-line Frequency Counter			
Frequency range	Input frequency	Display reading	Units
50- 69 Hz	60	60	Hz

Schmitt trigger output (H11L2GE) manufactured by Harris Semiconductor. The optocoupler converts the sine wave input cycles to square wave pulses which feed the input frequency to the dual BCD counters (IC1a and IC1b).

The two BCD digits (D2 and D1) that measure the units and tens of units of the ac line frequency are routed to the FPC Am29CPL151 by using a quad 2-to-1 multiplexer (74HC257). Table 5.8 shows the range of the frequency meter which varies from 50 to 69 Hz.

Table 5.9 contains the microcode program for the FPC Am29CPL151. As shown in Figure 5.11, the FPC is clocked by a 100 Hz frequency generated by IC4 and associated components. In this form, the FPC will perform each instruction in 10 ms (10,000  $\mu$ s); issuing output pulses with 10 ms of duration. The output pulses are named “reset,” “cken,” and “SELD2.” Thanks to the stable, low frequency of 100 Hz, the output “enable” (P10 of IC10) is generated by merely loading the CREG counter of the FPC with the number 99. Because the FPC Am29CPL151 contains a 6 CREG counter of six bits, the number 99 is loaded in two steps and then counter CREG is decremented and tested against zero. Notice that while the CREG counter is being decremented, the output “enable” permits counters IC1a and IC1b to count of the incoming pulses received at the input CLK (pin 1 of IC1a).

Table 5.9 presents the software program for the FPC Am29CPL151. The “comments” section in the software program explains in detail the steps being followed by the FPC to drive the speech processor correctly. In this case, the FPC Am29CPL151 is capable of reading the input frequency of two BCD counters (IC1a, and IC1b) by using a multiplexer 74HC257 (IC2). IC2 is a dual quad-channel multiplexer that selects the BCD output of each counter by changing the logic at the input SEL of IC2. The FPC drives the 74HC257 by asserting a logic high via output P9 in order to read the MSD digit. It is necessary to keep the input SEL of the 74HC257 in a logic high in order to maintain the selection of the second digit. Digit D1 is always selected by default when not specified. This feature permits the FPC to perform several tasks of comparison with the selected BCD digit. In this manner, the FPC selects one of the two BCD digits to execute the subroutine of comparisons in order to determine the decimal value of the selected digit. Output P11 of the FPC is used to assert a logic low to the /WR input of the Digitalker.

**TABLE 5.9**  
Software Program for the FPC Am29CPL151

```

DEVICE (CPL151)

DEFAULT = 1;
DEFINE "test inputs"
intr = t6    equal = eq    test = t4

        "Output control bits are given name assignments"
zero = 1F#h    one = 01#h    two = 02#h    three = 03#h
four = 04#h    five = 05#h    six = 06#h    seven = 07#h
eight = 08#h    nine = 09#h    fifty = 17#h    sixty = 18#h
pulses = 9C#h    seld2 = 100#h    wr = 800#h    error = xx#h
reset = 8000#h    cken = 400#h;

DEFAULT_OUTPUT = 0800#h;
TEST_CONDITION = INTR; "Default test condition"

BEGIN
"0"    zero+wr,    goto pl(n0);
"1"    one+wr,    goto pl(n1);
"2"    two+wr,    goto pl(n2);
"3"    three+wr,    goto pl(n3);
"4"    four+wr,    goto pl(n4);
"5"    five+wr,    goto pl(n5);
"6"    six+wr,    goto pl(n6);
"7"    seven+wr,    goto pl(n7);
"8"    eight+wr,    goto pl(n8);
"9"    nine+wr,    goto pl(n9);

"    -----"
"                                "
"                                "
"    -----"
"10"start:    ,if (test) then goto pl(start);
"11"RESET+wr,    load pl(59);
"12"stay:cken+wr,    while (creg <> 0) wait else load pl(39);
"13"stay1:cken+wr,    while (creg <> 0) loop to pl(stay1);

"    DISPLAY FORMAT: 00 Scale: Hz D2 D1    "
"14"SELD2+wr,    continue;
"15"SELD2+wr,    cmp tm(0F#h) to pl(05#h);    "D2 = 5?"
"16" fifty+wr,    if (equal) then goto pl(readD1);
"17"SELD2+wr,    continue;
"18"SELD2+wr,    cmp tm(0F#h) to pl(06#h);    "D2 = 6?"
"19" sixty+wr,    if (equal) then goto pl(rdD1);
"20" error+wr,    call pl(msgerr);
"21"readD1:fifty,    continue;    "/WR is pulsed low"
"22"stay2:    ,if (not intr) then goto pl(stay2);
"23"    ,cmp tm(0F#h) to pl(00#h);    "D1 = 0?"
"24"    ,if (equal) then goto pl(start);
"25"    ,goto tm(0F#h);
"26"rdD1:sixty,    if(test) then goto pl(stay2);"/WR = 0 "
"27"msgerr:error,    continue;    "/WR is pulsed low"
"28"stay3:    ,if (not intr) then goto pl(stay3);

```

```

"29"                ,goto pl(start);
"30"n0: zero,        goto pl(finish);    "/WR is pulsed low"
"31"n1: one,         goto pl(finish);
"32"n2: two,         goto pl(finish);
"33"n3: three,       goto pl(finish);
"34"n4: four,        goto pl(finish);
"35"n5: five,        goto pl(finish);
"36"n6: six,         goto pl(finish);
"37"n7: seven,       goto pl(finish);
"38"n8: eight,       goto pl(finish);
"39"n9: nine,        continue;
"40"finish:         ,if (not intr) then goto pl(finish);
"41"HZ: pulses+wr,   continue;            "Pulses"
"42"   pulses,       continue;
"43"                ,if (intr) then goto pl(start) else wait;
                   ,org 63#d
"44"                ,goto pl(start);
END.

```

When the operator presses the “test” switch, the program jumps to line 2, which resets the dual BCD counters while the internal CREG counter of the FPC is loaded with the number 59. The dual BCD counters (IC1a and IC1b) perform a complete digital reading because the input “enable” is held at a logic high for an interval of 1.000 s. In this case, the FPC makes the Digitalker announce digital readings within the range of 50 to 69. In addition, the Digitalker enunciates the word “pulses” after each digital ac line frequency reading is vocalized.

## 5.7 Using a Speech Processor to Monitor Respiratory Rate

To a seriously injured patient or one undergoing surgery, respiratory rate is critical to immediate survival. With a respiratory sensor placed under an oxygen mask, medical workers can monitor respirations per minute on a liquid crystal display. In addition, a speech synthesizer will announce the readings and warn of possible respiratory failures.

The sensor is a circuit that detects air pressure. Because the pressure of expired air is higher than that of inhaled air, the sensor, placed in the airway at the bottom of an oxygen mask, can monitor the patient’s respiratory flow. The circuit then converts these air-pressure signals to respirations per minute readings.

Edmund Scientific sells the sensing device, an ultrasensitive 0.004 psi air-pressure switch (Catalog No. E36,839). Single-pole, normally open switch



the sensor switch closes. This positive pulse is detected by the FPC Am29CPL154 using the testable input T7.

The FPC (IC3) enables timer IC2, a 5 kHz oscillator that drives a piezoelectric buzzer, to produce an audible tone for every pulse received. The positive pulse also causes the FPC to pass bursts of crystal-controlled 100 Hz signals from the MM5369EST oscillator chip IC5 by making the input COUNT DOWN high.

Once the period reading is stored in the binary counters 74HC193, the FPC enables the latch 74HC373 in order to send the binary reading to the six testable inputs T0 to T5. The binary reading is used by the FPC to jump directly to the location indicated by the same digital reading. When the program jumps to the location specified by the digital input reading, the new specified location will contain the equivalent in respirations per minute (RPM). This new equivalent reading in RPM is fed back and is loaded into the binary counters 74HC193 to be used for the speech processor's program. Also, outputs P0 to P7 of the FPC are used for loading the speech data into the speech synthesizer. Thanks to the 74HC257, the FPC can make use of several subroutines to determine the magnitude of the respiratory rate, so the Digitalker can announce it correctly.

The routine for measuring the period is in lines 3 to 6. The routine used for converting the period measured in binary format to an equivalent in respirations per minute is within lines 7 to 94. The "speech" routine for controlling the Digitalker starts in line 95.

To understand how the software program works, we will assume that we are measuring a respiratory rate with a period of 3.0 s. To measure the period within two respirations, the program will stay in line 3 (see Table 5.10) waiting for the first respiratory pulse. When the first pulse is received at input T7, the program jumps to line 4, which clears counters 74HC193 to a zero state in order to prepare them for a new digital period reading. The instruction in line 4 loads the internal CREG counter with the number 49. Then the instruction "while (creg <> 0) loop to pl(stay)" in line 5 decrements the internal CREG counter until it reaches a value of zero. Because the FPC is being clocked at 100 Hz, the internal counter CREG will be decremented in 0.5 s; the piezo-buzzer will sound for 0.5 s while the external counters 74HC193 keep measuring the period. The program then jumps to line 6 in order to wait for the next respiratory pulse. When the second respiratory pulse is received, the program jumps to line 7 where the instruction "goto tm(00111111#b)" sends the program to an address specified by the binary value of the respiratory period. This binary value is selected by the 74HC373 under control of the FPC.

Lines 8 to 63 are used to specify the equivalent constant in respirations per minute defined as "K." In this case, a period of 3.0 s will make the program jump to line 30. As shown in Table 5.10, line 30 contains the constant K20, equal to number 20 in hexadecimal. At the same time, the instruction in line

**TABLE 5.10**  
Software Program for the Respiratory Rate Meter

DEVICE (CPL154)

DEFAULT=1;

DEFINE "test inputs"

intr=t6 equal=eq resp=t7

" Output control bits and words from the Digitalker are assigned"

HIGH=5B#h "address of word high from DT1050 vocabulary"

zero=1F#h one=01#h two=02#h three=03#h four=04#h five=05#h  
six=06#h seven=07#h eight=08#h nine=09#h eleven=0B#H twelve=0C#H  
thirteen=0D#H fourteen=0E#H fifteen=0F#H sixteen=10#H seventeen  
=11#H eighteen=12#H nineteen=13#H twenty = 14#h thirty=15#H forty  
=15#H fifty = 17#h sixty=18#h

pulses=9C#h selD2=100#h wr=800#h clear=400#h Ena=100#h

buzzer=8000#h ld=1000#h dig1=8000#h dig2=200#h

R=31#h P=2F#h M=2A#h high = 5B#h

danger=4C#h low=67#h rate=7D#h HC373=4000#h HC257=2000#h

K60=60#h K55=55#h K50=50#h K46=46#h K42=42#h K40=40#h

K38=38#h K35=35#h K33=33#h K32=32#h K30=30#h K29=29#h

K27=27#h k26=26#h K25=25#h K24=24#h K23=23#h K22=22#h

K21=21#h K20=20#h K19=19#h K18=18#h K17=17#h K16=16#h

K15=15#h K14=14#h K13=13#h K12=12#h K11=11#h K10 = 10#h

K9 = 09#H;

DEFAULT\_OUTPUT=0800#h; "/WR is high when not specified"

TEST\_CONDITION=INTR; "Default test condition"

BEGIN

" 1 second delay to restart and initialize system"

"1"start: wr+ld+HC373+clear, load pl(99);

"2"same: wr+buzzer+ld+HC373, while (creg <> 0) loop to pl(same);

"3" stay: wr+ld+HC373, if (not resp) then goto pl(stay);

"4" wr+clear+ld+HC373, load pl(49);

"5" sty: wr+ena+ld+HC373+buzzer, while (creg <> 0) loop to pl(sty);

"6"stay2: wr+ena+ld+HC373, if (not resp) then goto pl(stay2);

"7"wr+ld+HC373+buzzer, goto tm(00111111#b); "Mask T0 to T5"

"-----"

" Routine for conversion to Respirations Per Minute "

"-----"

"8"wr+ld+HC257, goto pl(highrate);

"9"wr+ld+HC257, goto pl(highrate);

"10"wr+K60+ld+HC257, goto pl(rpm60);

"11"wr+K55+ld+HC257, goto pl(rpm55);

"12"wr+K50+ld+HC257, goto pl(rpm50);

"13"wr+K46+ld+HC257, goto pl(rpm46);

"14"wr+K42+ld+HC257, goto pl(rpm42);

"15"wr+K40+ld+HC257, goto pl(rpm40);

"16"wr+K38+ld+HC257, goto pl(rpm38);

"17"wr+K35+ld+HC257, goto pl(rpm35);

"18"wr+K33+ld+HC257, goto pl(rpm33);

```

"19"wr+K32+ld+HC257,      goto pl(rpm32);
"20"wr+K30+ld+HC257,      goto pl(rpm30);
"21"wr+K29+ld+HC257,      goto pl(rpm29);
"22"wr+K27+ld+HC257,      goto pl(rpm27);
"23"wr+K27+ld+HC257,      goto pl(rpm27);
"24"wr+K25+ld+HC257,      goto pl(rpm25);
"25"wr+K24+ld+HC257,      goto pl(rpm24);
"26"wr+K23+ld+HC257,      goto pl(rpm23);
"27"wr+K22+ld+HC257,      goto pl(rpm22);
"28"wr+K21+ld+HC257,      goto pl(rpm21);
"29"wr+K21+ld+HC257,      goto pl(rpm21);
"30"wr+K20+ld+HC257,      goto pl(rpm20);
"31"wr+K19+ld+HC257,      goto pl(rpm19);
"32"wr+K19+ld+HC257,      goto pl(rpm19);
"33"wr+K18+ld+HC257,      goto pl(rpm18);
"34"wr+K18+ld+HC257,      goto pl(rpm18);
"35"wr+K17+ld+HC257,      goto pl(rpm17);
"36"wr+K17+ld+HC257,      goto pl(rpm17);
"37"wr+K16+ld+HC257,      goto pl(rpm16);
"38"wr+K16+ld+HC257,      goto pl(rpm16);
"39"wr+K15+ld+HC257,      goto pl(rpm15);
"40"wr+K15+ld+HC257,      goto pl(rpm15);
"41"wr+K15+ld+HC257,      goto pl(rpm15);
"42"wr+K14+ld+HC257,      goto pl(rpm14);
"43"wr+K14+ld+HC257,      goto pl(rpm14);
"44"wr+K14+ld+HC257,      goto pl(rpm14);
"45"wr+K13+ld+HC257,      goto pl(rpm13);
"46"wr+K13+ld+HC257,      goto pl(rpm13);
"47"wr+K13+ld+HC257,      goto pl(rpm13);
"48"wr+K13+ld+HC257,      goto pl(rpm13);
"49"wr+K12+ld+HC257,      goto pl(rpm12);
"50"wr+K12+ld+HC257,      goto pl(rpm12);
"51"wr+K12+ld+HC257,      goto pl(rpm12);
"52"wr+K12+ld+HC257,      goto pl(rpm12);
"53"wr+K11+ld+HC257,      goto pl(rpm11);
"54"wr+K11+ld+HC257,      goto pl(rpm11);
"55"wr+K11+ld+HC257,      goto pl(rpm11);
"56"wr+K11+ld+HC257,      goto pl(rpm11);
"57"wr+K11+ld+HC257,      goto pl(rpm11);
"58"wr+K10+ld+HC257,      goto pl(rpm11);
"59"wr+K10+ld+HC257,      goto pl(rpm10);
"60"wr+K10+ld+HC257,      goto pl(rpm10);
"61"wr+K10+ld+HC257,      goto pl(rpm10);
"62"wr+K9+ld+HC257,       goto pl(rpm9);
"63"wr+ld+HC257,          goto pl(lowrate);

" ----- Load input of counters 74HC193 is pulsed low -----"
"64"rpm60: wr+K60+HC257,   goto pl(speech);
"65"rpm55: wr+K55+HC257,   goto pl(speech);
"66"rpm50: wr+K50+HC257,   goto pl(speech);
"67"rpm46: wr+K46+HC257,   goto pl(speech);
"68"rpm42: wr+K42+HC257,   goto pl(speech);
"69"rpm40: wr+K40+HC257,   goto pl(speech);
"70"rpm38: wr+K38+HC257,   goto pl(speech);
"71"rpm35: wr+K35+HC257,   goto pl(speech);

```



```

"72"rpm33: wr+K33+HC257,      goto pl(speech);
"73"rpm32: wr+K32+HC257,      goto pl(speech);
"74"rpm30: wr+K30+HC257,      goto pl(speech);
"75"rpm29: wr+K29+HC257,      goto pl(speech);
"76"rpm27: wr+K27+HC257,      goto pl(speech);
"77"rpm26: wr+K26+HC257,      goto pl(speech);
"78"rpm25: wr+K25+HC257,      goto pl(speech);
"79"rpm24: wr+K24+HC257,      goto pl(speech);
"80"rpm23: wr+K23+HC257,      goto pl(speech);
"81"rpm22: wr+K22+HC257,      goto pl(speech);
"82"rpm21: wr+K21+HC257,      goto pl(speech);
"83"rpm20: wr+K20+HC257,      goto pl(speech);
"84"rpm19: wr+K19+HC257,      goto pl(speech);
"85"rpm18: wr+K18+HC257,      goto pl(speech);
"86"rpm17: wr+K17+HC257,      goto pl(speech);
"87"rpm16: wr+K16+HC257,      goto pl(speech);
"88"rpm15: wr+K15+HC257,      goto pl(speech);
"89"rpm14: wr+K14+HC257,      goto pl(speech);
"90"rpm13: wr+K13+HC257,      goto pl(speech);
"91"rpm12: wr+K12+HC257,      goto pl(speech);
"92"rpm11: wr+K11+HC257,      goto pl(speech);
"93"rpm10: wr+K10+HC257,      goto pl(speech);
"94"rpm9: wr+K9+HC257,        goto pl(speech);

" -----"
" LD is pulsed high and the 4543 decoders display the RPM "
" -----"

"95"speech: wr+ld+HC257+dig2, continue;
"96"wr+ld+HC257+dig2,          cmp tm(0F#h) to pl(00#h); "D2=0? "
"97"wr+ld+HC257+dig2,          if (equal) then goto pl(BCD4);
"98"wr+ld+HC257+dig2,          cmp tm(0F#h) to pl(02#h); "D2=2? "
"99"wr+ld+HC257+dig2+twenty,   if (equal) then goto pl(n20);
"100"wr+ld+HC257+dig2,         cmp tm(0F#h) to pl(03#h); "D2=3? "
"101"wr+ld+HC257+dig2+thirty,  if (equal) then goto pl(n30);
"102"wr+ld+HC257+dig2,         cmp tm(0F#h) to pl(03#h); "D2=4? "
"103"wr+ld+HC257+dig2+forty,    if (equal) then goto pl(n40);
"104"wr+ld+HC257+dig2,         cmp tm(0F#h) to pl(03#h); "D2=5? "
"105"wr+ld+HC257+dig2+fifty,    if (equal) then goto pl(n50);
"106"wr+ld+HC257+dig2,         cmp tm(0F#h) to pl(03#h); "D2=6? "
"107"wr+ld+HC257+dig2+sixty,    if (equal) then goto pl(n60);
"108"wr+ld+HC257+dig1,         continue;
" ----- By default D2 = 1, and D1 is compared -----"
"109"BCD4: wr+ld+HC257,         cmp tm(0F#h) to pl(00#h);
"110"wr+ld+HC257,              if (equal) then goto pl(n10);
"111"wr+ld+HC257,              cmp tm(0F#h) to pl(01#h);
"112"wr+ld+HC257+eleven,       if (equal) then goto pl(n11);
"113"wr+ld+HC257,              cmp tm(0F#h) to pl(02#h);
"114"wr+ld+HC257+twelve,       if (equal) then goto pl(n12);
"115"wr+ld+HC257,              cmp tm(0F#h) to pl(03#h);
"116"wr+ld+HC257+thirteen,     if (equal) then goto pl(n13);
"117"wr+ld+HC257,              cmp tm(0F#h) to pl(04#h);
"118"wr+ld+HC257+fourteen,     if (equal) then goto pl(n14);
"119"wr+ld+HC257,              cmp tm(0F#h) to pl(05#h);

```

```

"120"wr+ld+HC257+fifteen,    if (equal) then goto pl(n15);
"121"wr+ld+HC257,            cmp tm(0F#h) to pl(06#h);
"122"wr+ld+HC257+sixteen,    if (equal) then goto pl(n16);
"123"wr+ld+HC257,            cmp tm(0F#h) to pl(07#h);
"123"wr+ld+HC257+seventeen,  if (equal) then goto pl(n17);
"124"wr+ld+HC257,            cmp tm(0F#h) to pl(08#h);
"125"wr+ld+HC257+eighteen,   if (equal) then goto pl(n18);
"126"wr+ld+HC257+nineteen,   goto pl(n19);
"  -----/WR is pulsed low in the instructions below  -----"
"127"n11: eleven+ld+HC257,    goto pl(finish);
"128"n12: twelve+ld+HC257,    goto pl(finish);
"129"n13: thirteen+ld+HC257,  goto pl(finish);
"130"n14: fourteen+ld+HC257,   goto pl(finish);
"131"n15: fifteen+ld+HC257,   goto pl(finish);
"132"n16: sixteen+ld+HC257,   goto pl(finish);
"133"n17: seventeen+ld+HC257, goto pl(finish);
"134"n18: eighteen+ld+HC257,  goto pl(finish);
"135"n19: nineteen+ld+HC257,  goto pl(finish);
"136"finish: wr+HC257+ld,     if (not intr) then goto pl(finish);
"137"wr+HC257+ld,             cmp tm(0F#h) to pl(00#h); "D1 = 0?"
"138"wr+HC257+ld,             if (equal) then goto pl(rpm2);
"139"wr+HC257+ld,             cmp tm(0F#h) to pl(01#h); "D1 = 1?"
"140"wr+HC257+ld,             if (equal) then goto pl(n1);
"141"wr+HC257+ld,             cmp tm(0F#h) to pl(02#h); "D1 = 2?"
"142"wr+HC257+ld,             if (equal) then goto pl(n2);
"143"wr+HC257+ld,             cmp tm(0F#h) to pl(03#h); "D1 = 3?"
"144"wr+HC257+ld,             if (equal) then goto pl(n3);
"145"wr+HC257+ld,             cmp tm(0F#h) to pl(04#h); "D1 = 4?"
"146"wr+HC257+ld,             if (equal) then goto pl(n4);
"147"wr+HC257+ld,             cmp tm(0F#h) to pl(05#h); "D1 = 5?"
"148"wr+HC257+ld,             if (equal) then goto pl(n5);
"149"wr+HC257+ld,             cmp tm(0F#h) to pl(06#h); "D1 = 6?"
"150"wr+HC257+ld,             if (equal) then goto pl(n6);
"151"wr+HC257+ld,             cmp tm(0F#h) to pl(07#h); "D1 = 7?"
"152"wr+HC257+ld,             if (equal) then goto pl(n7);
"153"wr+HC257+ld,             cmp tm(0F#h) to pl(08#h); "D1 = 8?"
"154"wr+HC257+ld,             if (equal) then goto pl(n8);
"155"wr+HC257+ld,             goto pl(n9);
"156"n0: zero+HC257+ld,       goto pl(fin); "/WR is pulsed low"
"156"n1: one+HC257+ld,        goto pl(fin);
"157"n2: two+HC257+ld,        goto pl(fin);
"158"n3: three+HC257+ld,      goto pl(fin);
"159"n4: four+HC257+ld,       goto pl(fin);
"160"n5: five+HC257+ld,       goto pl(fin);
"161"n6: six+HC257+ld,        goto pl(fin);
"162"n7: seven+HC257+ld,      goto pl(fin);
"163"n8: eight+HC257+ld,      goto pl(fin);
"164"n9: nine+HC257+ld,       goto pl(fin);
"165"fin:  wr+HC257+ld,        if (not intr) then goto pl(fin);
"166"      wr+HC257+ld,        ,call pl(rpm);
"167"      wr+HC257+ld+clear,   goto pl(start);
"168"n20:  twenty+HC257,       goto pl(fin);

```

```

"169"n30:  thirty+HC257,      goto pl(fin);
"170"n40:  forty+HC257,       goto pl(fin);
"171"n50:  fifty+HC257,       goto pl(fin);
"172"n60:  sixty+HC257,       goto pl(fin);
" ~~~~~~ "
"173"rpm:  wr+ld+HC257+R,      continue;
"174"      ld+HC257+R,         call pl(stand);
"175"      wr+ld+HC257+P,      continue;
"176"      ld+HC257+P,         call pl(stand);
"177"      wr+ld+HC257+M,      continue;
"178"      ld+HC257+M,         call pl(stand);
"179"      wr+ld+HC257,        ret;
"180"      stand: wr+ld+HC257,  if (not intr) then goto pl(stand);
"181"      wr+ld+HC257,        ret;
" ~~~~~~ "
"182"rpm2:  wr+ld+HC257,       call pl(rpm);
"183"      wr+ld+HC257,       goto pl(start);
"184"lowrate: wr+ld+HC257+DANGER, continue;
"185"      ld+HC257+DANGER,    call pl(stand);
"186"      wr+ld+HC257+LOW,     continue;
"187"      ld+HC257+LOW,       call pl(stand);
"188"RATE1:  wr+ld+HC257+RATE, continue;
"189"      ld+HC257+RATE,      call pl(stand);
"190"      wr+ld+HC257,       goto pl(start);
"191"highrate: wr+ld+HC257+DANGER, continue;
"192"      ld+HC257+DANGER,    call pl(stand);
"193"      wr+ld+HC257+HIGH,    continue;
"194"      ld+HC257+HIGH,      call pl(stand);
"195"      wr+ld+HC257+RATE,    goto pl(rate1);
      .org 511#d
"196"      .goto pl(start);
END.

```

20 “goto pl(rpm20)” makes the program jump to the label “rpm20” located in line 83. Notice that the instruction in line 83 does not specify the output “ld” in order to generate a low transition at the input LOAD of counters 74HC193. This low transition will load the number 20 into counters 74HC193. Note that counters 74HC193 now contain the number 20 at its outputs. In fact, the FPC solves the equation

$$F = \frac{60}{T}$$

where the constant 60 represents sixty seconds in a minute, and “T” is the period in seconds.

The instruction “goto pl(speech)” in line 83 now makes the program jump to the routine “speech” in order to start the procedure for the Digitalker. The

routine “speech” compares the two BCD digits selected by the 74HC257 under control of the FPC Am29CPL154. When this routine finds the correct value of the first digit, it loads the address where the number is located in the Digitalker. Then the program gives a logic low pulse to the /WR input of the Digitalker to start a speech sequence. The FPC will keep monitoring the status of the Digitalker by reading its /INTR output at the testable input T6 (see Figure 5.12). In this case the word “twenty . . .” is announced by the Digitalker, after which the program executes the instruction “call pl(rpm)” in line 166. Subroutine “rpm” announces the message “RPM” after the word “twenty,” where the word “RPM” stands for respirations per minute. Once the message “twenty RPM” is announced, the program jumps to the label “start” located in line 1.

Labels “lowrate” and “highrate” are used to make the Digitalker speak the respective word when a period between two respirations is out of range, in this case higher than 6.3 s and lower than 1.0 s, respectively.

## 5.8 Designing a Fault-Tolerant Respiratory Rate Meter

A respiratory rate meter with fault tolerance can be built using the same techniques and circuitry presented in Section 5.7, but in this case we will use redundancy techniques to avoid errors that could affect the output reading. We will use the method of triplicated modular redundancy (TMR) in order to isolate a single-bit fault and avoid a failure of the complete device.

Notice that if the respiratory rate meter announces an incorrect reading, medical workers can respond erroneously because they trust the reliability of the measurement performed by this device. The technique of using voters to isolate a single fault is now explained briefly as an introduction to the topic of redundancy.

Because it is impossible to have components which are totally reliable, redundancy techniques must be employed to increase the probability of system survival during the time  $t$ .

For a system  $M$ ,  $R(t)$  is defined as the probability that the system will not have failed up until time  $t$ . The principal goal of a fault design is to prolong the average life of the system; this measure is referred to as the mean time before failure (MTBF) defined as:

$$MTBF = \int_0^{\infty} -t dR$$

Assuming that the MTBF is constant ( $1/\lambda$ ), and defining  $\lambda$  as the failure rate, then  $R(t)$  is:

$$R(t) = e^{-\lambda t}$$

By using the circuitry of the respiratory rate meter presented in the previous section in triplicated form at the subsystem level to keep components to a minimum, a high-reliability respiratory meter can be achieved. The correction logic is made with a majority logic voter configured for three inputs. The voter realizes the function

$$v(x,y,z) = xy + yz + xz.$$

The scheme for the TMR voter is shown in Figure 5.13. Input errors and faults presented in V will be corrected, and any single fault of each module is permitted. Note that input sensors are equal but independent.

The reliability of the basic TMR configuration is

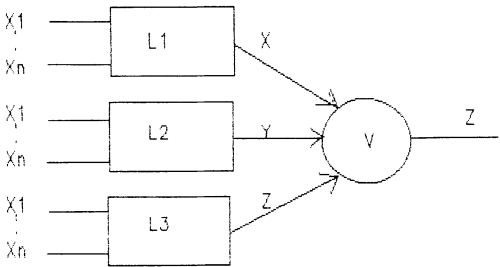
$$R = R_v * (R_m^3 + 3R_m^2 (1 - R_m))$$
$$R = R_v * (3R_m^2 - 2R_m^3)$$

where  $R_v$  and  $R_m$  are the reliabilities of the voter and a single copy of the triplicated module, respectively. The TMR configuration has a single point of failure: the voter. In the circuit of Figure 5.13 the only solution is to make the voter more reliable through a fault-avoidance and/or fault-tolerance technique.

A TMR system will fail only if two or more subsystems  $M_i$  fail; that is, some failure in two or more copies may occur in such a way that an error is not avoided. Assuming that each of the subsystems has probability  $R(t)$  of surviving to time  $t$ , the triplicated system probability of survival is  $R'(t)$  defined by

$$R'(t) = \frac{e^{-\lambda t} e^{-\lambda t} e^{-\lambda t}}{\text{Probability of all three of the subsystems surviving}} + \frac{3e^{-2\lambda t} (1 - e^{-\lambda t})}{\text{Probability of any two of the subsystems surviving}} = e^{-2\lambda t} - 2e^{-3\lambda t}$$

The concept of triplicated modular redundancy can also be applied on the subsystem level, as illustrated in Figure 5.14. In this case a single error in any subsystem will be corrected at the system output. In addition, multiple errors



**Figure 5.13** Basic TMR voter.



**TABLE 5.11**  
Program Utilized to Design TMR Voters with the Chip PAL16L8

```

TITLE          TRIPLE TMR VOTERS
PATTERN
REVISION A
AUTHOR         R. JIMENEZ
COMPANY        SHUGART CORPORATION
DATE           11/10/89
CHIP           VOTER PAL16L8

;pins 1 2 3 4 5 6 7 8 9 10
      X Y Z A B C D E F GND
      NC /P /M /N /O /Q /R /L NC VCC

;pins 11 12 13 14 15 16 17 18 19 20
;
;          VOTER
;          PAL16L8
;          -----
;          |          \ /          |
;          X -{ 1          20}- VCC
;          Y -{ 2          19}-> NC
;          Z -{ 3          18}-> /L
;          A -{ 4          17}-> /R
;          B -{ 5          16}-> /Q
;          C -{ 6          15}-> /O
;          D -{ 7          14}-> /N
;          E -{ 8          13}-> /M
;          F -{ 9          12}-> /P
;          GND -{10          11}- NC
;          |-----|

EQUATIONS

M = X*Y          ;FIRST TMR VOTER
  + X*Z
  + Y*Z

N = A*B          ;SECOND TMR VOTER
  + A*C
  + B*C

O = D*E          ;THIRD TMR VOTER
  + D*F
  + E*F

P = M*N          ;VOTING THE VOTERS
  + M*O
  + N*O

Q = /M           ;MONITORING OUTPUTS
R = /N
L = /O

SIMULATION

TRACE_ON X Y Z M A B C N D E F O P Q R L

```

```

; LOOK ALL TMR VOTERS                                ; LOOK THE VOTING
; VOTER

SETF   X   Y   Z           A   B   /C           D   /E   F
CHECK      M               N               O               P

SETF   X   Y   /Z          A   /B   C           D   /E   /F
CHECK      M               N               /O               P

SETF   X   /Y   Z          A   /B   /C           /D   E   F
CHECK      M               /N               O               P

SETF   X   /Y   /Z         /A   B   C           /D   E   /F
CHECK      /M               N               /O               /P

SETF   /X   Y   Z          /A   B   /C           /D   /E   F
CHECK      M               /N               /O               /P

SETF   /X   Y   /Z         /A   /B   C           /D   /E   /F
CHECK      /M               /N               /O               /P

SETF   /X   /Y   Z         /A   /B   /C           D   E   F
CHECK      /M               /N               O               /P

SETF   /X   /Y   /Z        A   B   C           D   E   /F
CHECK      /M               N               O               P

; LOOK AT THE INDICATOR OUTPUT

SETF   X   Y   Z           A   B   C           D   E   F
CHECK      /Q               /R               /L

TRACE_OFF

```

Using the TMR voters contained in the PAL16L8, Figure 5.14 shows the fault-tolerant version of the respiratory rate meter. Here three 7555 timers use the same RC components to detect respiration using the sensor network. If any one of the three timers, for example, is stuck at zero or stuck at one, the voter V1 will isolate the fault and will give the correct output generated by the rest of the timers operating in good conditions. The voter will also accept one of the timers stuck at zero and a second timer stuck at one while the third timer keeps operating correctly. Such failures are called compensating failures.

On the other hand, the period measured by counters 74HC193 is voted before reaching the inputs of the triplicated latches 74HC373 and the triplicated tri-state selectors 74HC257. Three FPCs sharing a TMR clock execute the program in triplicated form. The result of the respiratory measurement is voted before driving the speech processor. The program stored in the internal EPROM of each FPC is the same used by the FPC in the design presented in Table 5.10. Voting the FPCs has the advantage of being able to withstand failures in two different locations in two different EPROMs that are contained in each FPC. For example, consider what happens when a failure is present in memory location 65 on one memory of the FPC and a failure in memory location 67 on another. Since these failures are on two different FPCs, they do not act together in the voting process to cause an error.



**TABLE 5.12**  
JEDEC File for the PAL16L8

```

PAL16L8
VOTER*
QV512*
QP20*
QF2048*
G0*F0*
L0256 11111111111111111111111111111111*
L0288 11111111111111111111101111111111*
L0512 11111111111111111111111111111111*
L0544 11111111111111111111111011111111*
L0768 11111111111111111111111111111111*
L0800 11111111111111111111111111101111*
L1024 11111111111111111111111111111111*
L1056 11111111111111111111111011101111*
L1088 11111111111111111111101111111011*
L1120 11111111111111111111111110111011*
L1280 11111111111111111111111111111111*
L1312 11111111011101111111111111111111*
L1344 11111111011111110111111111111111*
L1376 11111111111101110111111111111111*
L1536 11111111111111111111111111111111*
L1568 01011111111111111111111111111111*
L1600 11010111111111111111111111111111*
L1632 01110111111111111111111111111111*
L1792 11111111111111111111111111111111*
L1824 11111111111111111111111110111011*
L1856 11111111111111111111110111111011*
L1888 11111111111111111111101110111111*
V0001 111110101NXLLLLHHHXN*
V0002 110101100NXLLLHHHLXN*
V0003 101100011NXLLHLHLHXN*
V0004 100011010NXHHLHLHLXN*
V0005 011010001NXHLHHHLXN*
V0006 010001000NXHHHLLXN*
V0007 001000111NXHHLLXN*
V0008 000111110NXLHLLHHXN*
V0009 111111111NXLLLLHHHXN*
C55B0*
EDBB

```

PALASM XPLOT, V2.23 - MARKET RELEASE (2-1-88)

(C) - COPYRIGHT MONOLITHIC MEMORIES INC, 1988

Title : TRIPLE TMR VOTERS Author : RICARDO JIMENEZ

Pattern : Company :

Revision : A Date : 11/10/89



**TABLE 5.13**  
EPROM Program for the Moisture Meter

Reading	Hex Add	Hex Data
0 %	00	2B, 3C, 35, 4, 44
1 %	40	2E, F, F, B, 4, 44
2 %	80	D, 1F, 4, 44
3 %	C0	1D, E, 13, 4, 44
4 %	100	28, 28, 3A, 4, 44
5 %	140	28, 28, 6, 23, 4, 44
6 %	180	37, 37, C, C, 2, 29, 37, 4, 44
7 %	1C0	37, 37, 7, 7, 23, C, B, 4, 44
8 %	200	14, 2, D, 4, 44
9 %	240	B, 18, 6, B, 4, 44
10 %	280	D, 7, 7, B, 4, 44
11 %	2C0	C, 2D, 7, 7, 23, C, 23, 4, 44
12 %	300	D, 30, 7, 7, 2D, 23, 4, 44
13 %	340	1D, 33, 2, D, 13, B, 4, 44
14 %	380	28, 3A, 1, 2, D, 13, B, 4, 44
15 %	3C0	28, C, 28, 1, 2, D, 13, B, 4, 44
16 %	400	37, 37, C, 2, 29, 37, 2, D, 13, B, 4, 44
17 %	440	37, 37, 7, 23, 1D, B, 2, D, 13, B, 4, 44
18 %	480	14, 2, D, 13, B, 4, 44
19 %	4C0	B, 6, B, 2, D, 13, B, 4, 44
20 %	500	D, 30, 7, 7, B, 2, D, 13, 2, 4, 44
21 %	540	D, 30, 7, 7, B, 2, D, 13, 2, 2E, F, F, B, 4, 44
22 %	580	D, 30, 7, 7, B, 2, D, 13, 2, D, 1F, 4, 44
23 %	5C0	D, 30, 7, 7, B, 2, D, 13, 2, 1D, E, 13, 4, 44
24 %	600	D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 3A, 4, 44
25 %	640	D, 30, 7, 7, B, 2, D, 13, 2, 28, 28, 6, 23, 4, 44
26 %	680	D, 30, 7, 7, B, 2, D, 13, 2, 37, 37, C, C, 2, 29, 37, 4, 44
27 %	6C0	D, 30, 7, 7, B, 2, D, 13, 2, 37, 37, 7, 7, 23, C, B, 4, 44
28 %	700	D, 30, 7, 7, B, 2, D, 13, 2, 14, 2D, 4, 44
29 %	740	D, 30, 7, 7, B, 2, D, 13, 2, B, 18, 6, B, 4, 44
30 %	780	1D, 34, 1, 2, D, 13, 4, 44
.....		
95 %	17C0	B, 6, B, 2, D, 13, 2, 28, 28, 6, 23, 4, 44
96 %	1800	B, 6, B, 2, D, 13, 2, 37, 37, 7, 7, 23, C, B, 4, 44
97 %	1840	B, 6, B, 2, D, 13, 2, 37, 37, 7, 7, 23, C, B, 4, 44
98 %	1880	B, 6, B, 2, D, 13, 2, 14, 2, D, 4, 44
99 %	18C0	B, 6, B, 2, D, 13, 2, B, 18, 6, B, 4, 44
100 %	19C0	2B, 3C, 35, 2, 39, F, F, B, 1, 21, 27, C, C, 1, 15, 4, 44

needed. This is achieved by turning the 1 megaohm pot while the 20K pot is adjusted at approximately 10K. When the sensor probe is out of the water, a reading of “zero” will be announced by the speech processor SPO256-AL2. Now you can measure moisture levels within the range of 0 to 100. For example, a rose plant needs moisture levels within the range of 70 and 80. If the measured level for this plant is within this range, the plant does not need any water. For more information about moisture levels for all kind of plants, see the manual included with the moisture sensor. The EPROM program is given in Table 5.13.

## 5.10 Designing a CMOS MSI<sup>3</sup> Capacitance Meter

The problem with many digital capacitance-measuring circuits lies in TTL or LS technology, which draws high current and mandates a high part count. Also, liquid-crystal displays (LCDs) use far less current than LED displays. A four-digit LED display draws about 250 mA, compared with about 20  $\mu$ A for LCDs.

A speech-synthesized CMOS MSI circuit with an LCD solves both problems (see Figure 5.16). The circuit, a four-digit LCD autoranging capacitance meter, requires only the insertion of the unknown capacitor ( $C_x$ ). The circuit automatically selects the scale and the display's decimal point position. The circuit's range is 1 pF to 1000  $\mu$ F. It consumes only 250 mA at 5 V.

Timer T1, operating in a linear monostable mode, charges  $C_x$  through a constant-current source formed by transistor Q1 and resistors R5 to R8 in accordance with the scale. IC1, a 74HC4066 quad bilateral switch, automatically selects one of these resistors. The time between pulse outputs from T1 then depends on the value of  $C_x$ .

Whenever the output (pin 3) of timer T1 is high, timer T2's reset input (pin 4) oscillates at the fixed frequency  $f_1$  of about 45,000 Hz. IC3, a 74HC4017 decade counter/divider, divides this frequency by 10 to  $f_2$ . The higher frequency serves the picofarad (pF) range; the lower frequency, the microfarad ( $\mu$ F) range. IC4, a 74HC4051 analog multiplexer/demultiplexer, selects  $f_1$  or  $f_2$  and routes it to the counter input of the BCD counters CD4518 (pin 1 of IC6a).

When the  $C_x$  capacitor charges to the threshold value of T1 (two-thirds of  $V_{dd}$ ), T1's output pin 3 goes low; then this pin is read by the FPC Am29CPL154. The input of T1 is triggered by the FPC when the user presses the switch “test,” causing that pin 3 output to go high. At this point, IC5

<sup>3</sup>Reprinted and adapted with permission from *Electronic Design*, Vol. 36, no. 20, September 8, 1988. Copyright 1988, Penton Publishing.





A potentiometer (Ra) nulls the stray collector-base capacitance of transistor Q1. Diodes D1 and D2 protect the T1 timer from possible overcharges caused by inserting a charged capacitor. When first turned on, the circuit starts in the picofarad range, because IC2, another 74HC4017, delivers a high from its Q0 output, which controls IC1, IC4, and the range LEDs through an IC7 XOR gate (one-fourth of CD4071). Accordingly, IC1, which controls the constant charging current of Cx, selects the 10 megaohm resistor, R8. Also, IC4 selects the higher frequency f1 because R8 is for small capacitors that need a high frequency to properly measure the charging time. Finally, the XOR gate output stays high to make the LED D4 glow, indicating that the LCD's digital reading is in picofarads. The circuit enters the picofarad range automatically after announcing the value of the capacitor under measurement. To start a measurement, press the S1 switch to reset IC2 to start with Q0 high. The maximum pF scale reading is 9999 pF.

A Cx with a higher value, for example 0.020  $\mu\text{F}$ , causes an overflow pulse from the /carry output (pin 28) of the IC6 counter. This output activates the clock input of the IC2 decade-counter, which makes IC2's Q1 output go high and selects the 100 kilohm resistor R7. Cx then charges more rapidly than it would with the 10 megaohm resistor and generates no overflow pulse. Also the decimal point DP1 now glows because the XOR gate driving it also depends on Q1 of IC2. The reading display then is 0.020  $\mu\text{F}$ . On this scale, the maximum reading is 9.999  $\mu\text{F}$ .

To interface the autoranging capacitance meter with the speech processor Digitalker, we use the FPC Am29CPL154. The right side of Figure 5.16 shows the FPC controlling a 2-to-7 decoder to select each of the four digits. As can be seen in Figure 5.16, outputs P0 to P7 send the speech data to the SW1–SW8 address inputs of the Digitalker. Output Y5 of IC11 is used to pulse the /WR input of the Digitalker low. Input T7 of the FPC is used for monitoring when the speech processor has completed saying a word. Outputs P10 and P15 of the FPC are used to trigger timer T1 and reset Johnson counter IC2, respectively. Testable inputs T4 and T5 of the FPC are used to determine the position of the decimal point when a reading has been performed. This way, the FPC knows the scale of the digital reading in order to process it and make the Digitalker articulate the message corresponding to the capacitor under measurement. To start a reading, the user has to insert the unknown capacitor Cx near timer T1, and then press the switch S1 in order to cause a logic low at the testable input T6 of the FPC (IC10).

The routine used by the FPC is similar to the one presented in the first section of Chapter 5. In this case, the routine is adapted in the main process section to control timer T1 and counter IC2. Table 5.14 shows the software program required to generate the JEDEC file and program the FPC Am29CPL154 for this particular function.

**TABLE 5.14**

Software Program for the Speech Synthesized Autoranging Capacitance Meter

```

DEVICE (CPL154)

DEFAULT = 1;

DEFINE    "test inputs"
intr = t7    equal = eq
test = t6    "T6 reads switch S1 or output timer T1"

        "Output control bits are given name assignments"
zero = 1F#h    one = 01#h    two = 02#h    three = 03#h
four = 04#h    five = 05#h    six = 06#h    seven = 07#h
eight = 08#h    nine = 09#h    ten = 0A#h    eleven = 0B#h
twelve = 0C#h    thirteen = 0D#h    fourteen = 0E#h    fifteen = 0F#h
sixteen = 10#h    seventeen = 11#h    eighteen = 12#h    nineteen = 13#h
twenty = 14#h    thirty = 15#h    forty = 16#h    fifty = 17#h    sixty
= 18#h
seventy = 19#h    eighty = 1A#h    ninety = 1B#h    hundred = 1C#h
thousand = 1D#h    U = 34#h    P = 2F#h    F = 25#h
point = 9A#h    dig1 = 1000#h    wr = 5000#h    and = 3C#h
reset1 = 8000#h    dig2 = 2000#h    selP8 = 100#h
in1 = 400#h    dig3 = 3000#h
GL = 800#h    dig4 = 4000#h;

DEFAULT_OUTPUT = 0000#h;
OUT_POLARITY = F7FF#h;
TEST_CONDITION = INTR;    "Default test condition"

BEGIN
"0"    zero,    goto pl(n0);
"1"    one,    goto pl(n1);
"2"    two,    goto pl(n2);
"3"    three,    goto pl(n3);
"4"    four,    goto pl(n4);
"5"    five,    goto pl(n5);
"6"    six,    goto pl(n6);
"7"    seven,    goto pl(n7);
"8"    eight,    goto pl(n8);
"9"    nine,    goto pl(n9);

"
-----"
"
                                MAIN PROCESS
"
-----"

"10"start:reset1, if (test) then goto pl(start);
"11"                ,cmp tm (30#h) to pl (00#h);    "testing overflow"
"12"                ,if (equal) then goto pl(pF);    "pF scale "
"13"    reset1, call pl(count);
"14"                ,cmp tm (30#h) to pl (10#h);    "testing overflow"
"15"                ,if (equal) then goto pl(uF1);    "uF scale "
"16"    reset1, call pl(count);
"17"                ,cmp tm (30#h) to pl (20#h);    "testing overflow"
"18"                ,if (equal) then goto pl(uF2);    "uF scale"

```



```

"19"      reset1, call pl(count);
"20"      ,goto pl(uF3);
"
"-----"
"
"21"pF: dig4,      continue;
"22"  dig4+GL,      continue;  " Digit 4 is selected by IC 74HC137"
"23"      ,cmp tm(0F#h) to pl(00#h);      "D4 = 0?"
"24"  dig3,      if (not equal) then goto pl(spkD4);
"25"  dig3+GL,      continue;
"26"      ,cmp tm(0F#h) to pl(00#h);      "D3 = 0?"
"27"  dig2,      if (not equal) then goto pl(spkD3);
"28"  dig2+GL,      continue;
"29"      ,cmp tm(0F#h) to pl(00#h);      "D2 = 0?"
"30"      ,if (not equal) then goto pl(ptyc);
"31"spkd1: dig1,      continue;
"32"  dig1+GL,      call pl(announ);      "Announce D1 "
"33"hrtz:      ,call pl(HZ);      "Hertz"
"34"      ,goto pl(start);
"35"spkd4: dig4,      continue;
"36"  dig4+GL,      call pl(announ);      "Announce D4"
"37"  thousand,      continue;
"38"thousand+wr,      continue;      "Thousand..."
"39"same: dig3,      if (not intr) then goto pl(same);
"40"  dig3+GL,      continue;
"41"      ,cmp tm(0F#h) to pl(00#h);      "D3 = 0?"
"42"      ,if (not equal) then goto pl(spkD3);
"43"  dig2,      continue;
"44"  dig2+GL,      continue;
"45"      ,cmp tm(0F#h) to pl(00#h);      "D2 = 0?"
"46"      ,if (not equal) then goto pl(ptyand);
"47"spkand: and,      continue;
"48"  and+wr,      continue;      "And..."
"49"same2:      ,if (not intr) then goto pl(same2);
"50"      ,goto pl(spkd1);
"51"spkd3: dig3,      continue;
"52"  dig3+GL,      call pl(announ);      "Announce D3"
"53"  hundred,      continue;
"54"  hundred+wr,      continue;      "Hundred..."
"55"same3: dig2,      if (not intr) then goto pl(same3);
"56"  dig2+GL,      continue;
"57"      ,cmp tm(0F#h) to pl(00#h);      "D2 = 0?"
"58"      ,if (not equal) then goto pl(ptyand);
"59"  dig1,      continue;
"60"  dig1+GL,      continue;
"61"      ,cmp tm(0F#h) to pl(00#h);      "D1 = 0?"
"62"      ,if (not equal) then goto pl(spkand);
"63"      ,goto pl(hrtz);
"64"ptyand: and,      continue;
"65"  and+wr,      continue;
"66"same4: dig2,      if (not intr) then goto pl(same4);
"67"ptyc: dig2+GL,      continue;
"68"      ,cmp tm(0F#h) to pl(01#h);      "D2 = 1?"
"69"      ,if (not equal) then goto pl(ptyb);

```

```

"70"  dig1,      continue;
"71"  dig1+GL,   call pl(BCD4);          "Announce D1"
"72"                ,goto pl(hrtz);
"73"ptyb: dig2,   continue;          "Announce D2"
"74"  dig2+GL,   call pl(BCD3b);
"75"    dig1,     continue;
"76"  dig1+GL,   continue;
"77"                ,cmp tm(0F#h) to pl(00#h);      "D1 = 0? "
"78"                ,if (equal) then goto pl(hrtz);
"79"                ,call pl(announ);          "Announce D1"
"80"                ,goto pl(hrtz);

"  -----  "
      "  DISPLAY FORMAT: 10.00  Scale: uF2  D4 D3.D2 D1"
"81"uF2: dig4,      continue; "D4 is necessarily not zero"
"82"    dig4+GL,     continue; "D4 is latched"
"83"                ,cmp tm(0F#h) to pl(00#h);      "D4=0? "
"84"                ,if (not equal) then goto pl(rick5);
"85"said3: dig3,     continue;      "say D3 because D4=0"
"86"    dig3+GL,     call pl(announ); "D3 is latched"
"87"pnt:  point,     continue;      "Point.."
"88"    point+wr,     continue;
"89"idle:            ,if (not intr) then goto pl(idle);
"90"    dig2,         continue;
"91"    dig2+GL,      continue;
"92"                ,cmp tm(0F#h) to pl(00#h); "D2=0? "
"93"                ,if (not equal) then goto pl(rick2);
"94"                ,continue;
"95"                ,call pl(announ); "announce D2"
"96"pat:  dig1,       continue;
"97"    dig1+GL,      call pl(announ); "announce D1"
"98"paty:            ,call pl(KHZ);  "announce KiloHertz"
"99"                ,goto pl(start);
"100"rick2: dig1,      continue;
"101"    dig1+GL,      continue;
"102"                ,cmp tm(0F#h) to pl(00#h);
"103"                ,if (not equal) then goto pl(rick3);
"104"    dig2,         continue;
"105"    dig2+GL,      call pl(BCD3);
"106"                ,goto pl(paty);
"107"rick3: dig2,      continue;
"108"    dig2+GL,      continue;
"109"                ,cmp tm(0F#h) to pl(01#h);
"110"                ,if (not equal) then goto pl(rick4);
"111"                ,call pl(BCD4);
"112"                ,goto pl(paty);
"113"rick4: dig2,      continue;
"114"    dig2+GL,      call pl(BCD3b);
"115"                ,goto pl(pat);
"116"rick5: dig4,      continue;
"117"    dig4+GL,      continue;
"118"                ,cmp tm(0F#h) to pl(01#h); "D4=1? "
"119"                ,if (not equal) then goto pl(rick6);

```

```

"120"      dig3,      continue;
"121"      dig3+GL,   call pl(BCD4);      "Announce D3"
"122"      ,goto pl(pnt);
"123"rick6: dig4,      continue;
"124"      dig4+GL,   call pl(BCD3b);      "Announce D4"
"125"      dig3,      continue;
"126"      dig3+GL,   continue;
"127"      ,cmp tm(0F#h) to pl(00#h); "D3 = 0? "
"128"      ,if (not equal) then goto pl(sayd3);
"129"      ,goto pl(pnt);

"  -----"
"          DISPLAY FORMAT: 100.0   Scale: uF3      D4 D3 D2.D1      "
"130"uF3: dig4,      continue; "D4 is not zero"
"131"      dig4+GL,   call pl(announ); "D4 is latched"
"132"      hundred,   continue;      "Hundred... "
"133"      hundred+wr, continue;
"134"sty5:      ,if (not intr) then goto pl(sty5);
"135"      and,      continue;
"136"      and+wr,    continue;      "And.... "
"137"sty6:      ,if (not intr) then goto pl(sty6);
"138"      dig3,      continue;
"139"      dig3+GL,   continue;
"140"      ,cmp tm(0F#h) to pl(00#h); "D3=0? "
"141"      ,if (not equal) then goto pl(lug3);
"142"lug4: dig2,      continue;
"143"      dig2+GL,   call pl(announ); "D2 is announced"
"144"lug5: point,    continue;
"145"      point+wr,  continue;
"146"sty20:      ,if(not intr) then goto pl(sty20);
"147"      dig1,      continue;
"148"      dig1+GL,   call pl(announ); "D1 is announced"
"149"      ,call pl(KHZ);
"150"      ,goto pl(start);
"151"lug3: dig3,      continue;
"152"      dig3+GL,   continue;
"153"      ,cmp tm(0f#h) to pl(01#h);      "D3=1?"
"154"      ,if (equal) then goto pl(lug6);
"155"      ,call pl(BCD3b);
"156"      dig2,      continue;
"157"      dig2+GL,   continue;
"158"      ,cmp tm(0F#h) to pl(00#h); "D2=0?"
"159"      ,if (not equal) then goto pl(lug4);
"160"      ,goto pl(lug5);
"161"lug6: dig2,      continue;
"162"      dig2+GL,   call pl(BCD4);
"163"      ,goto pl(lug5);

"  -----"
"          DISPLAY FORMAT: 1.000   Scale: uF1      "
"164"uF1: dig4,      continue;
"165"      dig4+GL,   call pl(announ); "D4 is latched"
"166"      point,    continue;
"167"      point+wr,  continue;      "Point"

```

```

"168"sty8:                                ,if (not intr) then goto pl(sty8);
"169"      dig3,                            continue;
"170"      dig3+GL,                        call pl(announ); "D3 is latched"
"171"      dig2,                            continue;
"172"      dig2+GL,                        call pl(announ); "D2 is selected"
"173"      dig1,                            continue;
"174"      dig1+GL,                        call pl(announ); "D1 is selected"
"175"                                          ,call pl(MHZ);
"176"                                          ,goto pl(start);

"
" ***** Routine to trigger timer T1 ***** "
"177"count: selP8, continue; "pin 3 of T1 is selected"
"178"  selP8+in1, continue; "timer T1 is triggered"
"179"stay: selP8,  if (test) then goto pl(stay); "wait for T1
to go low"
"180"                                ,ret;

"
"          ROUTINES BCD3 AND BCD3b          "
"

"180"BCD3:                                ,cmp tm(0F#h) to pl(01#h);
"181"  ten+GL,                            if (equal) then goto pl(n10);
"182"BCD3b:                                ,cmp tm(0F#h) to pl(02#h);
"183"  twenty+GL,                         if (equal) then goto pl(n20);
"184"                                ,cmp tm(0F#h) to pl(03#h);
"185"  thirty+GL,                         if (equal) then goto pl(n30);
"186"                                ,cmp tm(0F#h) to pl(04#h);
"187"  forty+GL,                         if (equal) then goto pl(n40);
"188"                                ,cmp tm(0F#h) to pl(05#h);
"189"  fifty+GL,                         if (equal) then goto pl(n50);
"190"                                ,cmp tm(0F#h) to pl(06#h);
"191"  sixty+GL,                         if (equal) then goto pl(n60);
"192"                                ,cmp tm(0F#h) to pl(07#h);
"193"seventy+GL,                         if (equal) then goto pl(n70);
"194"                                ,cmp tm(0F#h) to pl(08#h);
"195"  eighty+GL,                        if (equal) then goto pl(n80);
"196"  ninety+GL,                        goto pl(n90);
"197"n10: ten+wr,                          goto pl(xb);
"198"n20: twenty+wr,                      goto pl(xb);
"199"n30: thirty+wr,                      goto pl(xb);
"200"n40: forty+wr,                      goto pl(xb);
"201"n50: fifty+wr,                      goto pl(xb);
"202"n60: sixty+wr,                      goto pl(xb);
"203"n70: seventy+wr,                    goto pl(xb);
"204"n80: eighty+wr,                     goto pl(xb);
"205"n90: ninety+wr,                     goto pl(xb);
"206"xb:                                ,if (not intr) then goto pl(xb);
"207"                                ,ret;

```

```

" ~~~~~"
"      ROUTINE BCD4      "
" ~~~~~"

"208" BCD4:      ,cmp tm(0F#h) to pl(00#h);
"209" ten+GL,    if (equal) then goto pl(n10);
"210"           ,cmp tm(0F#h) to pl(01#h);
"211" eleven+GL, if (equal) then goto pl(n11);
"212"           ,cmp tm(0F#h) to pl(02#h);
"213" twelve+GL, if (equal) then goto pl(n12);
"214"           ,cmp tm(0F#h) to pl(03#h);
"215" thirteen+GL, if (equal) then goto pl(n13);
"216"           ,cmp tm(0F#h) to pl(04#h);
"217" fourteen+GL, if (equal) then goto pl(n14);
"218"           ,cmp tm(0F#h) to pl(05#h);
"219" fifteen+GL, if (equal) then goto pl(n15);
"220"           ,cmp tm(0F#h) to pl(06#h);
"221" sixteen+GL, if (equal) then goto pl(n16);
"222"           ,cmp tm(0F#h) to pl(07#h);
"223" seventeen+GL, if (equal) then goto pl(n17);
"224"           ,cmp tm(0F#h) to pl(08#h);
"225" eighteen+GL, if (equal) then goto pl(n18);
"226" nineteen+GL, goto pl(n19);
"227" n11: eleven+wr,      goto pl(finish);
"228" n12: twelve+wr,      goto pl(finish);
"229" n13: thirteen+wr,    goto pl(finish);
"230" n14: fourteen+wr,    goto pl(finish);
"231" n15: fifteen+wr,     goto pl(finish);
"232" n16: sixteen+wr,     goto pl(finish);
"233" n17: seventeen+wr,   goto pl(finish);
"234" n18: eighteen+wr,    goto pl(finish);
"235" n19: nineteen+wr,    goto pl(finish);

"236" announ:      ,goto tm(0F#h);
"237" n0: zero+wr,  goto pl(finish);
"238" n1: one+wr,   goto pl(finish);
"239" n2: two+wr,   goto pl(finish);
"240" n3: three+wr, goto pl(finish);
"241" n4: four+wr,  goto pl(finish);
"242" n5: five+wr,  goto pl(finish);
"243" n6: six+wr,   goto pl(finish);
"244" n7: seven+wr, goto pl(finish);
"245" n8: eight+wr, goto pl(finish);
"246" n9: nine+wr,  goto pl(finish);

"247" finish:      ,if (not intr) then goto pl(finish);
"248"              ,ret;

" ~~~~~"

"249" HZ: P,        continue;
"250"   P+wr,       continue;
"251"              ,if (intr) then goto pl(stop) else wait;
"252" FF: F,        continue;
"253"   F+wr,       continue;
"254"              ,if (intr) then goto pl(stop) else wait;
"

```

```

"255"KHZ:U,          continue;
"256"      U+wr,      continue;
"257"                                ,if (intr) then goto pl(FF) else wait;
"258"MHZ:million      ,goto pl(KHZ);
"259"stop:wr,         ret;
"  -----
                                .org 511#d
"260"                                ,goto pl(start);
END.

```

## 5.11 Designing a Talking Solid State Barometer

The circuit shown in Figure 5.17 is a 0 to 200 mmHg pressure meter that vocalizes readings each time the user presses a switch. The 3-1/2 digit A/D converter will give 199.9 mmHg full-scale. This meter provides a resolution of 0.1 mmHg. The same circuit can also be used for other pressure ranges simply by changing the sensor and gain.

The resistor divider network composed of two 100K resistors and pot R1 provides the offset adjustment for the circuit. The sensor used in this project is the BPO1 manufactured by Sensym (1255 Reamwood Avenue, Sunnyvale, CA 94089). The BPO1 consists of a highly linear, low noise semiconductor pressure sensor in combination with a precision thick film ceramic, housed in a compact nylon case. This package offers small size and excellent isolation to external package stresses. It also provides convenient mounting holes and pressure ports for ease of use with standard plastic tubing.

Since the BPO1 provides an output which is ratiometric to its supply voltage, using the electrical parameters of the data sheet, the expected output voltage will be 10 mV at 200 mmHg when operating from a 5 V supply. For the components values shown, a full-scale input voltage of 200 mV ( $V_o$ ) is required for the TSC8750 in order to display a full-scale output of 1999. This way, the gain required for the instrumentation amplifier is 20. The output voltage equation and the gain equation derived are given below and are now used to solve for the unknown resistance,  $R_t$ :

$$V_{out} = V_{in} [2(1 + R/R_t)] + V_o$$

or, rewriting

$$V_{out} = V_{in} A_v + V_o$$

where  $A_v = 2(1 + R/R_t)$ .

$V_o$  is the initial output  $V_o$  for zero pressure applied. From the last equation, with  $R_1 = 10K$ ,  $R_t$  is found to be 1.1K. The full-scale span adjustment is fine tuned by using pot R3, which sets the reference voltage of the A/D converter.







**TABLE 5.15**  
Software Program for the Talking Barometer

DEVICE (CPL154)

DEFAULT = 1;

DEFINE "test inputs"

busy = t7 intr = t6 equal = eq

"Output control bits are given name assignments"

zero = 1F#h one = 01#h two = 02#h three = 03#h

four = 04#h five = 05#h six = 06#h seven = 07#h

eight = 08#h nine = 09#h ten = 0A#h eleven = 0B#h

twelve = 0C#h thirteen = 0D#h fourteen = 0E#h fifteen = 0F#h

sixteen = 10#h seventeen = 11#h eighteen = 12#h nineteen = 13#h

twenty = 14#h thirty = 15#h forty = 16#h fifty = 17#h sixty  
= 18#h

seventy = 19#h eighty = 1A#h ninety = 1B#h hundred = 1C#h

thousand = 1D#h milli = 6C#h volt = 8E#h ss = 81#h kilo = 62#h

over = 75#h point = 9A#h dig1 = 1000#h wr = 5000#h and = 3C#h

1000#h convrs = 4000#h

dig1 = 1000#h

dig2 = 2000#h

dig3 = 3000#h

dig4 = 4000#h

GL = 0800#h;

DEFAULT\_OUTPUT = 0000#h;

OUT\_POLARITY = F7FF#h;

TEST\_CONDITION = INTR; "Default test condition"

BEGIN

"0" zero, goto pl(n0);

"1" one, goto pl(n1);

"2" two, goto pl(n2);

"3" three, goto pl(n3);

"4" four, goto pl(n4);

"5" five, goto pl(n5);

"6" six, goto pl(n6);

"7" seven, goto pl(n7);

"8" eight, goto pl(n8);

"9" nine, goto pl(n9);

```
"  ~~~~~"
"                MAIN PROCESS                "
```

"10"start:scale1, continue;

"11"voltage:convrs+scale1, continue;"pin 21 is pulsed high"

"12"stay: scale1, if (busy) then goto pl(stay);

"-----"

" DISPLAY FORMAT: 000.0 - 199.9 Scale: mV (0-200 mV) D4 D3 D2.D1"

"13"SPmV:dig4, continue; "D4 is not zero"

```

"14"      dig4+GL,      call pl(announ); "D4 is latched"
"15"      hundred,      continue;      "Hundred..."
"16"      hundred+wr,    continue;
"17"sty5:      ,if (not intr) then goto pl(sty5);
"18"      and,          continue;
"19"      and+wr,        continue;      "And..."
"20"sty6:      ,if (not intr) then goto pl(sty6);
"21"      dig3,          continue;
"22"      dig3+GL,        continue;
"23"      ,cmp tm(0F#h) to pl(00#h); "D3=0?"
"24"      ,if (not equal) then goto pl(lug3);
"25"lug4: dig2,          continue;
"26"      dig2+GL,        call pl(announ); "D2 is announced"
"27"lug5: point,          continue;
"28"      point+wr,        continue;
"29"sty20:      ,if(not intr) then goto pl(sty20);
"30"      dig1,          continue;
"31"      dig1+GL,        call pl(announ); "D1 is announced"
"32"      ,call pl(KHZ);   "Millivolts"
"33"      ,goto pl(start);
"34"lug3: dig3,          continue;
"35"      dig3+GL,        continue;
"36"      ,cmp tm(0f#h) to pl(01#h);      "D3=1?"
"37"      ,if (equal) then goto pl(lug6);
"38"      ,call pl(BCD3b);
"39"      dig2,          continue;
"40"      dig2+GL,        continue;
"41"      ,cmp tm(0F#h) to pl(00#h); "D2=0?"
"42"      ,if (not equal) then goto pl(lug4);
"43"      ,goto pl(lug5);
"44"lug6: dig2,          continue;
"45"      dig2+GL,        call pl(BCD4);
"46"      ,goto pl(lug5);

"      ~~~~~~
"      ROUTINES BCD3 AND BCD3b      "
"      ~~~~~~

"47"BCD3:      ,cmp tm(0F#h) to pl(01#h);
"48"      ten+GL,      if (equal) then goto pl(n10);
"49"BCD3b:      ,cmp tm(0F#h) to pl(02#h);
"50"      twenty+GL,   if (equal) then goto pl(n20);
"51"      ,cmp tm(0F#h) to pl(03#h);
"52"      thirty+GL,   if (equal) then goto pl(n30);
"126"      ,cmp tm(0F#h) to pl(04#h);
"127"      forty+GL,   if (equal) then goto pl(n40);
"128"      ,cmp tm(0F#h) to pl(05#h);
"129"      fifty+GL,   if (equal) then goto pl(n50);
"130"      ,cmp tm(0F#h) to pl(06#h);
"131"      sixty+GL,   if (equal) then goto pl(n60);
"132"      ,cmp tm(0F#h) to pl(07#h);
"133"seventy+GL,   if (equal) then goto pl(n70);
"134"      ,cmp tm(0F#h) to pl(08#h);
"135"      eighty+GL,  if (equal) then goto pl(n80);
"136"      ninety+GL,  goto pl(n90);

```

```

"137"n10: ten+wr,      goto pl(xb);
"138"n20: twenty+wr,   goto pl(xb);
"139"n30: thirty+wr,   goto pl(xb);
"140"n40: forty+wr,    goto pl(xb);
"141"n50: fifty+wr,    goto pl(xb);
"142"n60: sixty+wr,    goto pl(xb);
"143"n70: seventy+wr,  goto pl(xb);
"144"n80: eighty+wr,   goto pl(xb);
"145"n90: ninety+wr,   goto pl(xb);
"146"xb:               ,if (not intr) then goto pl(xb);
"147"                  ,ret;
" ~~~~~~ "
"      ROUTINE BCD4      "
" ~~~~~~ "
"148" BCD4:            ,cmp tm(0F#h) to pl(00#h);
"149"  ten+GL,         if (equal) then goto pl(n10);
"150"                  ,cmp tm(0F#h) to pl(01#h);
"151"  eleven+GL,      if (equal) then goto pl(n11);
"152"                  ,cmp tm(0F#h) to pl(02#h);
"153"  twelve+GL,      if (equal) then goto pl(n12);
"154"                  ,cmp tm(0F#h) to pl(03#h);
"155"  thirteen+GL,    if (equal) then goto pl(n13);
"156"                  ,cmp tm(0F#h) to pl(04#h);
"157"  fourteen+GL,    if (equal) then goto pl(n14);
"158"                  ,cmp tm(0F#h) to pl(05#h);
"159"  fifteen+GL,     if (equal) then goto pl(n15);
"160"                  ,cmp tm(0F#h) to pl(06#h);
"161"  sixteen+GL,     if (equal) then goto pl(n16);
"162"                  ,cmp tm(0F#h) to pl(07#h);
"163"  seventeen+GL,   if (equal) then goto pl(n17);
"164"                  ,cmp tm(0F#h) to pl(08#h);
"165"  eighteen+GL,    if (equal) then goto pl(n18);
"166"  nineteen+GL,    goto pl(n19);
"167"n11: eleven+wr,   goto pl(finish);
"168"n12: twelve+wr,   goto pl(finish);
"169"n13: thirteen+wr, goto pl(finish);
"170"n14: fourteen+wr,  goto pl(finish);
"171"n15: fifteen+wr,   goto pl(finish);
"172"n16: sixteen+wr,   goto pl(finish);
"173"n17: seventeen+wr, goto pl(finish);
"174"n18: eighteen+wr,  goto pl(finish);
"175"n19: nineteen+wr,  goto pl(finish);
"176"announ:           ,goto tm(0F#h);
"177"n0: zero+wr,      goto pl(finish);
"178"n1: one+wr,       goto pl(finish);
"179"n2: two+wr,       goto pl(finish);
"180"n3: three+wr,     goto pl(finish);
"181"n4: four+wr,      goto pl(finish);
"182"n5: five+wr,      goto pl(finish);
"183"n6: six+wr,       goto pl(finish);
"184"n7: seven+wr,     goto pl(finish);

```

```

"185"n8: eight+wr,      goto pl(finish);
"186"n9: nine+wr,       goto pl(finish);
"187"finish:            ,if (not intr) then goto pl(finish);
"188"                    ,ret;
" ~~~~~ "
"189"HZ: volt,          continue;
"190"  volt+wr,         continue;      "mm Hg..."
"191"                    ,if (intr) then goto pl(ssa) else wait;
"192"ssa: ss,           continue;      "S..."
"193"  ss+wr,           continue;
"194"                    ,if (intr) then goto pl(stop) else wait;
"195"KHZ: milli,        continue;
"196"  milli+wr,        continue;
"197"                    ,if (intr) then goto pl(HZ) else wait;
"198"stop: wr,          ret;
"199"msgerr: over,       continue;      "OVER..."
"200"  over+wr,         continue;
"201"                    ,if (intr) then goto pl(HZ) else wait;
                        .org 511#d
"208"                    ,goto pl(start);
END.

```

the testable inputs T0 and T1 of the FPC Am29CPL152. Output P12 of the FPC is used to control Nand gate IC1c, which in turn drives the internal LED contained in the optocoupler MOC3010. The optocoupler MOC3010 is used to isolate the ac power line from the circuit. Outputs P8 to P11 of the FPC drive the BCD-to-seven segment decoder/driver CD4543. Accordingly, output P13 is used to blank the LED display when the subroutine "light" is being performed in order to avoid a reading of "0" when P8 to P11 are not specified in the microcode program.

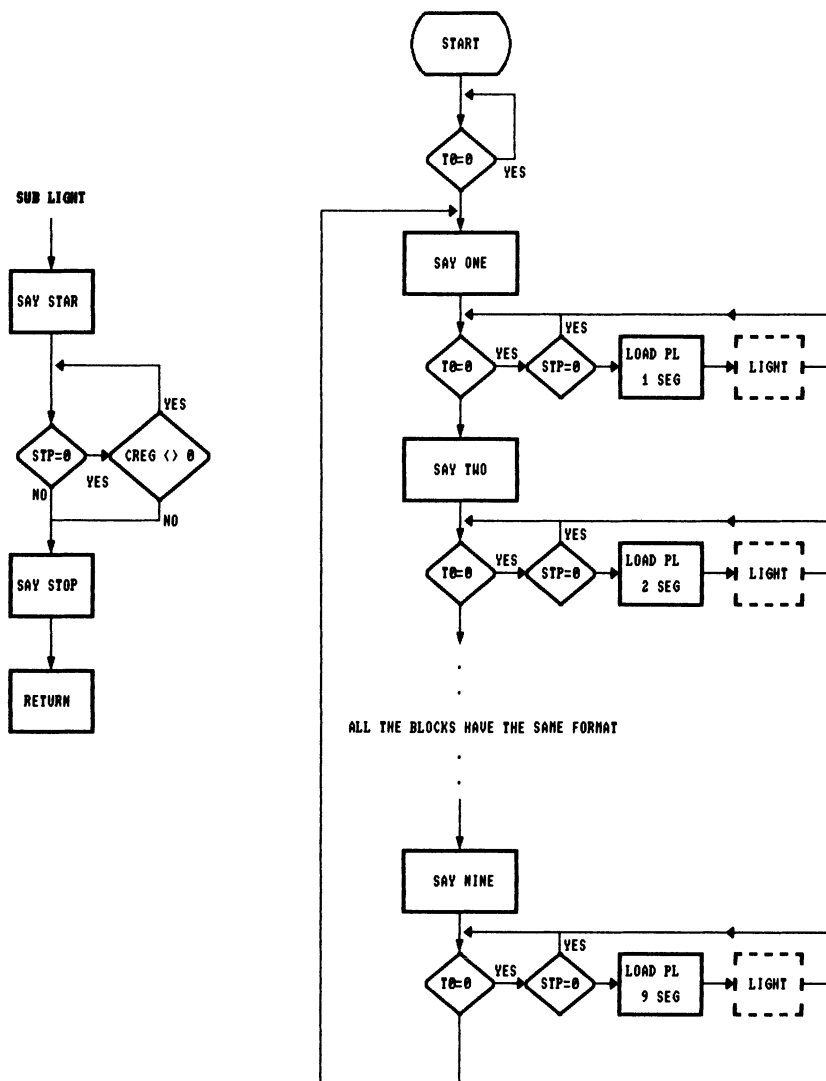
When pushbutton "stp" is closed momentarily, the user will hear the word "start" then the ac lamp will be turned on for the previously selected time. If the user needs to produce the same interval in the ac lamp, he just has to press the "stp" switch again.

Every time you push the "time" switch, the FPC begins its upward count. The current BCD number present at its outputs (P8 to P11) is decoded by CD4543 and is displayed on a common-cathode, seven-segment LED readout. The FPC's program is designed to increment its position every time the "time" switch is pressed; this causes the program to increment the number displayed in the readout and make the speech processor speak the new number of seconds that the light should be on.

When the number of seconds is correct, it is time to press the STP button; this button starts the light. If the user presses "stp" again when the light is on, the light will be turned off immediately.



and then will turn on the ac lamp for an interval of 9 s. When this period ends, the speech processor will speak the word “stop” to indicate that the process is complete. The program then returns to the routine that gives nine seconds of light. If the operator wants to expose several films to periods of nine seconds of light, he will keep pressing the “stp” switch. If not, he will have to press the “time” switch to select a new timing interval of light.



**Figure 5.19** Flowchart for the darkroom timer.

**TABLE 5.16**  
Software Program for the Speech Synthesized Darkroom Timer

```

DEVICE (CPL152)

DEFAULT = 1;
DEFINE "test inputs"
ti = t0
stp = t1
sby = t2

"allophones and pauses are given name assignments"

pa2 = 01#h      "      -----"
pa3 = 02#h      "      t6->:1      28|- Vcc "
pa4 = 03#h      "      p0<-:2      27|<-clk "
pa5 = 04#h      "      p1<-:3      26|<-cc  "
oy = 05#h      "      p2<-:4      25|<-t0  "
ay = 06#h      "      p3<-:5      Am29CPL 24|<-t1 "
eh = 07#h      "      p4<-:6      152  23|<-t2 "
kk3 = 08#h      "      p5<-:7      22|<-t3  "
pp = 09#h      "      p6<-:8      21|<-t4  "
jh = 0A#h      "      p7<-:9      20|<-t5  "
nn1 = 0B#h      "      p8<-:10     19|<-/reset"
ih = 0C#h      "      p9<-:11     18|<->p15  "
tt2 = 0D#h      "      p10<-:12    17|<->p14  "
rr1 = 0E#h      "      p11<-:13    16|<->p13  "
ax = 0F#h      "      Gnd-:14     15|<->p12  "
mm = 10#h      "      -----"
tt1 = 11#h

dh1 = 12#h      iy = 13#h      ey = 14#h      dd1 = 15#h      uw1 = 16#h
ao = 17#h      aa = 18#h      yy2 = 19#h      ae = 1A#h      hh1 = 1B#h
bb1 = 1C#h      th = 1D#h      uh = 1E#h      uw2 = 1F#h      aw = 20#h
dd2 = 21#h      gg3 = 22#h      vv = 23#h      gg1 = 24#h      sh = 25#h
zh = 26#h      rr2 = 27#h      ff = 28#h      kk2 = 29#h      kk1 = 2A#h
zz = 2B#h      ng = 2C#h      ll = 2D#h      ww = 2E#h      xr = 2F#h
wh = 30#h      yy1 = 31#h      ch = 32#h      er1 = 33#h      er2 = 34#h
ow = 35#h      dh2 = 36#h      ss = 37#h      nn2 = 38#h      hh2 = 39#h
or = 3A#h      ar = 3B#h      yr = 3C#h      gg2 = 3D#h      el = 3E#h
bb2 = 3F#h;

"control the light activator and led display readout "
AClight = 1000#h      BI = 2000#h "Blanking Display"

k1 = 100#h      k2 = 200#h      k3 = 300#h      k4 = 400#h
k5 = 500#h      k6 = 600#h      k7 = 700#h      k8 = 800#h      k9 = 900#h;

"the position of the display readout is p8 to p11 ( A thru D )"
DEFAULT_OUTPUT = 0000#h;
TEST_CONDITION = SBY; "/STANDBY is the default test condition"
BEGIN
"Wait for ti to go high"
"1"start:      , if (not ti) then goto pl(start);
"2"st1:      ww+k1, call pl(read); " say ONE "

```

```

"3"          ax+k1, call pl(read);
"4"          ax+k1, call pl(read);
"5"          nn1+k1, call pl(read);
"6"          pa3+k1, call pl(read);
"7"st11:     k1, if (ti) then goto pl(st2);
"8"          k1, if (not stp) then goto pl(st11);
"9"          k1, load pl(4);
"10"         k1, call pl(light);
"11"         k1, goto pl(st11);

"12"st2:     tt2+k2, call pl(read); " say TWO "
"13"         uw2+k2, call pl(read);
"14"         pa3+k2, call pl(read);
"15"st21:    k2, if (ti) then goto pl(st3);
"16"         k2, if (not stp) then goto pl(st21);
"17"         k2, load pl(9);
"18"         k2, call pl(light);
"19"         k2, goto pl(st21);

"20"st3:     th+k3, call pl(read); " say THREE "
"21"         rr1+k3, call pl(read);
"22"         iy+k3, call pl(read);
"23"         pa3+k3, call pl(read);
"24"st31:    k3, if (ti) then goto pl(st4);
"25"         k3, if (not stp) then goto pl(st31);
"26"         k3, load pl(14);
"27"         k3, call pl(light);
"28"         k3, goto pl(st31);

"29"st4:     ff+k4, call pl(read); " say FOUR "
"30"         ff+k4, call pl(read);
"31"         or+k4, call pl(read);
"32"         pa3+k4, call pl(read);
"33"st41:    k4, if (ti) then goto pl(st5);
"34"         k4, if (not stp) then goto pl(st41);
"35"         k4, load pl(19);
"36"         k4, call pl(light);
"37"         k4, goto pl(st41);

"38"st5:     ff+k5, call pl(read); " say FIVE "
"39"         ff+k5, call pl(read);
"40"         ay+k5, call pl(read);
"41"         vv+k5, call pl(read);
"42"         pa3+k5, call pl(read);
"43"st51:    k5, if (ti) then goto pl(st6);
"44"         k5, if (not stp) then goto pl(st51);
"45"         k5, load pl(24);
"46"         k5, call pl(light);
"47"         k5, goto pl(st51);

"48"st6:     ss+k6, call pl(read); " say SIX "
"49"         ss+k6, call pl(read);
"50"         ih+k6, call pl(read);
"51"         ih+k6, call pl(read);
"52"         pa3+k6, call pl(read);
"53"         kk2+k6, call pl(read);
"54"         ss+k6, call pl(read);
"55"         pa3+k6, call pl(read);

```



```

"56"st61:          k6, if (ti) then goto pl(st7);
"57"              k6, if (not stp) then goto pl(st61);
"58"              k6, load pl(29);
"59"              k6, call pl(light);
"60"              k6, goto pl(st61);
"61"st7:           ss+k7, call pl(read); " say SEVEN "
"62"              ss+k7, call pl(read);
"63"              eh+k7, call pl(read);
"64"              eh+k7, call pl(read);
"65"              vv+k7, call pl(read);
"66"              eh+k7, call pl(read);
"67"              nn1+k7, call pl(read);
"68"              pa3+k7, call pl(read);
"69"st71:          k7, if (ti) then goto pl(st8);
"70"              k7, if (not stp) then goto pl(st71);
"71"              k7, load pl(34);
"72"              k7, call pl(light);
"73"              k7, goto pl(st71);
"74"st8:           ey+k8, call pl(read); " say EIGHT "
"75"              pa3+k8, call pl(read);
"76"              tt2+k8, call pl(read);
"77"              pa3+k8, call pl(read);
"78"st81:          k8, if (ti) then goto pl(st9);
"79"              k8, if (not stp) then goto pl(st81);
"80"              k8, load pl(39);
"81"              k8, call pl(light);
"82"              k8, goto pl(st81);
"83"st9:           nn2+k9, call pl(read); " say NINE "
"84"              aa+k9, call pl(read);
"85"              ay+k9, call pl(read);
"86"              nn1+k9, call pl(read);
"87"              pa3+k9, call pl(read);
"88"st91:          k9, if (ti) then goto pl(st1);
"89"              k9, if (not stp) then goto pl(st91);
"90"              k9, load pl(44);
"91"              k9, call pl(light);
"92"              k9, goto pl(st91);

"subroutine READ"
"93"read:          BI, continue;      "allows sby to go low in 300 ns"
"94"styal:         BI, if (not sby) then goto pl(styal); "reading
SBY"
"95"              BI, ret;

"subroutine light"
"96"light:         BI+ss, call pl(read);
"97"              BI+ss, call pl(read);      " say START "
"98"              BI+pa3, call pl(read);
"99"              BI+tt2, call pl(read);
"100"             BI+ar, call pl(read);
"101"             BI+pa3, call pl(read);
"102"             BI+tt2, call pl(read);

```

```

"103"light2:  AClight+pa5, continue;
"104"         AClight+pa1, continue;
"105"stay2:   AClight+pa1, if (not sby) then goto pl(stay2);
"106"         AClight+pa1, if (stp) then goto pl(stop);
"107"         AClight+pa1, while (creg<>0) loop to pl(light2);
"108"stop:    ss, call pl(read);
"109"         ss, call pl(read);      " say STOP "
"110"         pa3, call pl(read);
"111"         tt1, call pl(read);
"112"         aa, call pl(read);
"113"         aa, call pl(read);
"114"         pa3, call pl(read);
"115"         pp, call pl(read);
"116"         pa3, call pl(read);
"117"         , ret;

                org 127#d
"118"         , goto pl(start);
END.

```

## 5.13 Talking Current Meter

The circuit shown in Figure 5.20 is a talking current meter that vocalizes readings within the range of 0 to 200 mA. Its resolution is 1 mA. The conversion of current to voltage is performed by using an A/D converter (ADC) with parallel BCD outputs (TSC8750). The resulting digital conversion is read and controlled by an FPC which also drives the speech processor Digitalker.

The use of a shunt resistor converts the current to a voltage. In this case, a shunt resistor of 1 ohm (1/4 W) is used at the input pin of the ADC TSC8750. When measuring current, the 199 mV scale is used. This limits the voltage drop to 1 mV per count. The relationships for finding the values of resistors  $R_{in}$  and  $R_{ref}$  are:

$$R_{in} = \frac{V_{in \text{ Full-Scale}}}{10 \mu A} = \frac{0.2 \text{ V}}{10 \mu A} = 20K$$

$$R_{ref} = \frac{V_{ref}}{-20 \mu A} = \frac{-5 \text{ V}}{-20 \mu A} = 250K$$

The process for controlling the ADC TSC8750 is similar to the one shown in Section 10 of this chapter. In this case, the TSC8750 is controlled by output P14 of the FPC. When the user connects the terminal cables for measuring a current, he presses the "test" switch to cause a negative transient pulse at the input T4 of the FPC. The testable input T4 is being monitored by the instruction "start: if (test) then goto pl(start)," as shown in Table 5.17; this instruc-



the conversion process of the ADC TSC8750 is complete, line 13 of the program reads the logic status of the output “busy” of the ADC. The instruction “hold: ,if (busy) then goto pl(hold)” will keep reading the function pin “busy” in order to wait for the end of the conversion process. When the output “busy” goes to a logic low, the FPC jumps by default to the next instruction where the routine for reading the digital value of digits D1, D2, and D3, is performed. As soon as the FPC detects a digit different from zero, the FPC causes the Digitalker to announce the digital number according to its position. D3 is the most significant digit and D1 is the less significant digit. (In this application Digit D4 is left unconnected because we are measuring only currents within the range of 1 to 199 mA with a resolution of 1 mA.) Notice that the routine (from line 21 to 160) also monitors the value of digit D4; this will not affect the function of the FPC and the Digitalker. This allows you to change the scale of the readings without having to alter the software for your specific need. Bear in mind that the software program presented in Table 5.17 will be able to handle current measurements when the digits D1 to D4 are not using a decimal point. If you want to use a decimal point, consider using the entire routine presented in Table 5.1.

**TABLE 5.17**  
Software Program for the Talking Current Meter

```

DEVICE (CPL154)

DEFAULT = 1;

DEFINE "test inputs"
intr = t7    equal = eq
test = t4    busy = t5

"Output control bits are given name assignments"
zero = 1F#h    one = 01#h    two = 02#h    three = 03#h
four = 04#h    five = 05#h    six = 06#h    seven = 07#h
eight = 08#h    nine = 09#h    ten = 0A#h    eleven = 0B#h
twelve = 0C#h    thirteen = 0D#h    fourteen = 0E#h    fifteen = 0F#h
sixteen = 10#h    seventeen = 11#h    eighteen = 12#h    nineteen = 13#h
twenty = 14#h    thirty = 15#h    forty = 16#h    fifty = 17#h    sixty
= 18#h
seventy = 19#h    eighty = 1A#h    ninety = 1B#h    hundred = 1C#h
thousand = 1D#h    U = 34#h    P = 2F#h    F = 25#h
point = 9A#h    dig1 = 1000#h    wr = 5000#h    and = 3C#h
reset1 = 8000#h    dig2 = 2000#h    selP8 = 100#h
in1 = 400#h    dig3 = 3000#h
GL = 800#h    dig4 = 4000#h;

DEFAULT_OUTPUT = 0000#h;
OUT_POLARITY = F7FF#h;
TEST_CONDITION = INTR; "Default test condition"

```

```

BEGIN
"0"    zero,      goto pl(n0);
"1"    one,       goto pl(n1);
"2"    two,       goto pl(n2);
"3"    three,     goto pl(n3);
"4"    four,      goto pl(n4);
"5"    five,      goto pl(n5);
"6"    six,       goto pl(n6);
"7"    seven,     goto pl(n7);
"8"    eight,     goto pl(n8);
"9"    nine,      goto pl(n9);

"
-----"
"  PROCESS FOR CONTROLLING THE ADC TSC8750  "
"-----"

"10"start:      ,if (test) then goto pl(start);
"11"    in1,     continue; "ADC initiates conversion"
"12"          ,continue;
"13"hold:      ,if (busy) then goto pl(hold);
-----"

"          DISPLAY FORMAT: 0000 Scale: mA  D4 D3 D2 D1  "
"21" dig4,      continue;
"22" dig4+GL,   continue; " Digit 4 is selected by IC 74HC137"
"23"          ,cmp tm(0F#h) to pl(00#h); "D4 = 0?"
"24" dig3,      if (not equal) then goto pl(spkd4);
"25" dig3+GL,   continue;
"26"          ,cmp tm(0F#h) to pl(00#h); "D3 = 0?"
"27" dig2,      if (not equal) then goto pl(spkd3);
"28" dig2+GL,   continue;
"29"          ,cmp tm(0F#h) to pl(00#h); "D2 = 0?"
"30"          ,if (not equal) then goto pl(ptyc);
"31"spkd1: dig1, continue;
"32" dig1+GL,   call pl(announ); "Announce D1 "
"33"hrtz:      ,call pl(HZ); "mA"
"34"          ,goto pl(start);
"35"spkd4: dig4, continue;
"36" dig4+GL,   call pl(announ); "Announce D4"
"37" thousand, continue;
"38"thousand+wr, continue; "Thousand..."
"39"same: dig3, if (not intr) then goto pl(same);
"40" dig3+GL,   continue;
"41"          ,cmp tm(0F#h) to pl(00#h); "D3 = 0?"
"42"          ,if (not equal) then goto pl(spkd3);
"43" dig2,      continue;
"44" dig2+GL,   continue;
"45"          ,cmp tm(0F#h) to pl(00#h); "D2 = 0?"
"46"          ,if (not equal) then goto pl(ptyand);
"47"spkand: and, continue;
"48" and+wr,    continue; "And..."
"49"same2:      ,if (not intr) then goto pl(same2);
"50"          ,goto pl(spkd1);

```

```

"51"spkd3:dig3,  continue;
"52"  dig3+GL,   call pl(announ);           "Announce D3"
"53"  hundred,   continue;
"54"  hundred+wr,continue;                 "Hundred..."
"55"same3:dig2,  if (not intr) then goto pl(same3);
"56"  dig2+GL,   continue;
"57"                ,cmp tm(0F#h) to pl(00#h);           "D2 = 0?"
"58"                ,if (not equal) then goto pl(ptyand);
"59"  dig1,       continue;
"60"  dig1+GL,    continue;
"61"                ,cmp tm(0F#h) to pl(00#h);           "D1 = 0?"
"62"                ,if (not equal) then goto pl(spkan);
"63"                ,goto pl(hrtz);
"64"ptyand:and,   continue;
"65"  and+wr,     continue;
"66"same4:dig2,  if (not intr) then goto pl(same4);
"67"ptyc:dig2+GL,continue;
"68"                ,cmp tm(0F#h) to pl(01#h);           "D2 = 1?"
"69"                ,if (not equal) then goto pl(ptyb);
"70"  dig1,       continue;
"71"  dig1+GL,    call pl(BCD4);               "Announce D1"
"72"                ,goto pl(hrtz);
"73"ptyb:dig2,   continue;                   "Announce D2"
"74"  dig2+GL,    call pl(BCD3b);
"75"  dig1,       continue;
"76"  dig1+GL,    continue;
"77"                ,cmp tm(0F#h) to pl(00#h);           "D1 = 0?"
"78"                ,if (equal) then goto pl(hrtz);
"79"                ,call pl(announ);           "Announce D1"
"80"                ,goto pl(hrtz);

~~~~~"
"          ROUTINES BCD3 AND BCD3b          "
"~~~~~"

"80"BCD3:        ,cmp tm(0F#h) to pl(01#h);
"81"  ten+GL,     if (equal) then goto pl(n10);
"82"BCD3b:       ,cmp tm(0F#h) to pl(02#h);
"83"  twenty+GL,  if (equal) then goto pl(n20);
"84"                ,cmp tm(0F#h) to pl(03#h);
"85"  thirty+GL,  if (equal) then goto pl(n30);
"86"                ,cmp tm(0F#h) to pl(04#h);
"87"  forty+GL,   if (equal) then goto pl(n40);
"88"                ,cmp tm(0F#h) to pl(05#h);
"89"  fifty+GL,   if (equal) then goto pl(n50);
"90"                ,cmp tm(0F#h) to pl(06#h);
"91"  sixty+GL,   if (equal) then goto pl(n60);
"92"                ,cmp tm(0F#h) to pl(07#h);
"93"seventy+GL,  if (equal) then goto pl(n70);
"94"                ,cmp tm(0F#h) to pl(08#h);
"95"  eighty+GL,  if (equal) then goto pl(n80);
"96"  ninety+GL,  goto pl(n90);

```

```

"97"n10: ten+wr,      goto pl(xb);
"98"n20: twenty+wr,   goto pl(xb);
"99"n30: thirty+wr,    goto pl(xb);
"100"n40: forty+wr,    goto pl(xb);
"101"n50: fifty+wr,    goto pl(xb);
"102"n60: sixty+wr,    goto pl(xb);
"103"n70: seventy+wr,  goto pl(xb);
"104"n80: eighty+wr,   goto pl(xb);
"105"n90: ninety+wr,   goto pl(xb);
"106"xb:               ,if (not intr) then goto pl(xb);
"107"                 ,ret;
" ~~~~~~ "
"      ROUTINE BCD4      "
" ~~~~~~ "
"108" BCD4:             ,cmp tm(0F#h) to pl(00#h);
"109"  ten+GL,          if (equal) then goto pl(n10);
"110"                 ,cmp tm(0F#h) to pl(01#h);
"111"  eleven+GL,       if (equal) then goto pl(n11);
"112"                 ,cmp tm(0F#h) to pl(02#h);
"113"  twelve+GL,       if (equal) then goto pl(n12);
"114"                 ,cmp tm(0F#h) to pl(03#h);
"115"  thirteen+GL,     if (equal) then goto pl(n13);
"116"                 ,cmp tm(0F#h) to pl(04#h);
"117"  fourteen+GL,     if (equal) then goto pl(n14);
"118"                 ,cmp tm(0F#h) to pl(05#h);
"119"  fifteen+GL,      if (equal) then goto pl(n15);
"120"                 ,cmp tm(0F#h) to pl(06#h);
"121"  sixteen+GL,      if (equal) then goto pl(n16);
"122"                 ,cmp tm(0F#h) to pl(07#h);
"123"  seventeen+GL,    if (equal) then goto pl(n17);
"124"                 ,cmp tm(0F#h) to pl(08#h);
"125"  eighteen+GL,     if (equal) then goto pl(n18);
"126"  nineteen+GL,     goto pl(n19);
"127"n11: eleven+wr,    goto pl(finish);
"128"n12: twelve+wr,    goto pl(finish);
"129"n13: thirteen+wr,  goto pl(finish);
"130"n14: fourteen+wr,   goto pl(finish);
"131"n15: fifteen+wr,   goto pl(finish);
"132"n16: sixteen+wr,   goto pl(finish);
"133"n17: seventeen+wr, goto pl(finish);
"134"n18: eighteen+wr,  goto pl(finish);
"135"n19: nineteen+wr,  goto pl(finish);
"136"announ:           ,goto tm(0F#h);
"137"n0: zero+wr,       goto pl(finish);
"138"n1: one+wr,        goto pl(finish);
"139"n2: two+wr,        goto pl(finish);
"140"n3: three+wr,      goto pl(finish);
"141"n4: four+wr,       goto pl(finish);
"142"n5: five+wr,       goto pl(finish);
"143"n6: six+wr,        goto pl(finish);
"144"n7: seven+wr,      goto pl(finish);

```

```

"145"n8: eight+wr,      goto pl(finish);
"146"n9: nine+wr,       goto pl(finish);
"147"finish:            ,if (not intr) then goto pl(finish);
"148"                   ,ret;

" ~~~~~ "
"149"HZ: m,             continue;    "Milli amperes"
"150"  P+wr,            continue;
"151"                   ,if (intr) then goto pl(stop) else wait;
"152"FF: F,             continue;
"153"  F+wr,            continue;
"154"                   ,if (intr) then goto pl(stop) else wait;
"155"KHZ: U,            continue;
"156"  U+wr,            continue;
"157"                   ,if (intr) then goto pl(FF) else wait;
"158"MHZ: million       ,goto pl(KHZ);
"159"stop: wr,          ret;

" ~~~~~ "

      .org 511#d
"160"                   ,goto pl(start);
END.

```

## 5.14 Designing a Liquid-Level<sup>4</sup> Annunciator

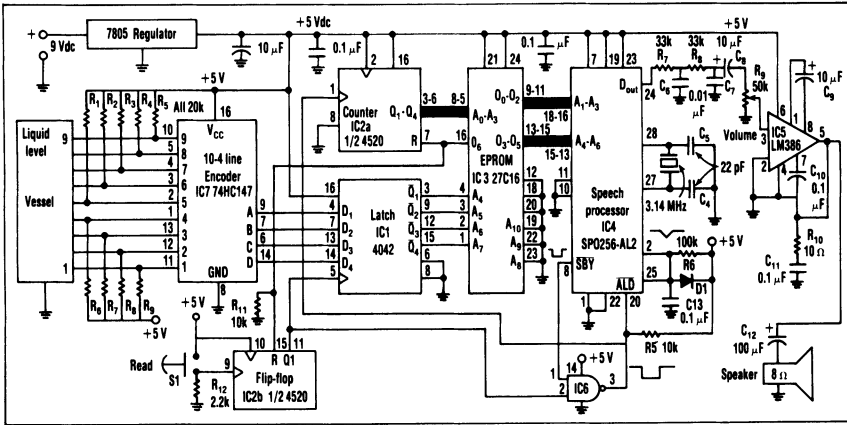
A chemist might find that a liquid-level annunciator circuit comes in handy to detect the liquid level of chemical-reactor vessels in his lab. The annunciator circuit contains a speech processor and loudspeaker that gives an audible report of the level of liquid in a vessel each time the user presses the read switch, S1. For example, for a liquid-level input of nine, the loudspeaker announces "nine."

Measuring liquid level requires only one chip, IC7, a 74HC147 10-to-4-line priority encoder (see Figure 5.21). Each of its nine inputs is connected to a level-sensor electrode in the vessel, and each input has a 20K pull-up resistor (R1 to R9) to provide a logic one when its level-sensor electrode is dry. All the electrodes deliver logic one inputs when the vessel is empty; logic zeros when it is full.

When the user pushes switch S1, a quad clocked-D latch CD4042 (IC1) latches the chip's encoded outputs and triggers IC2b—half of a CMOS dual CD4520 up-counter configured as a flip-flop. The inverted outputs from the latch (/Q1 to /Q4) deliver positive-acting logic, liquid-level values to the upper-address inputs of the 27C16 EPROM (A4 to A7). The other half of the

<sup>4</sup>Reprinted with permission from *Electronic Design*, Vol. 37, no. 4, February 23, 1989. Copyright 1989, Penton Publishing.





**Figure 5.21** The annunciator's circuit loudspeaker gives an audible report of a vessel's liquid when S1 is pressed.

CD4520 counter (IC2a) scans the lower-address memory locations (A1 to A3) in sequence.

After the user closes S1, setting the Q1 output of IC2b high, each negative-going output pulse from Nand gate IC6 to the speech processor's address-load input (/ALD) loads the processor with the currently addressed EPROM data. This block of EPROM memory data delivers a preprogrammed sequence of instructions to the speech processor IC4, an SPO256-AL2. For example, when the liquid level reaches five (ABCD = 0101 from IC7), the loudspeaker announces the number "five."

The processor, while delivering speech, holds its standby output (/SBY) low for an interval appropriate to that particular allophone. The low /SBY resets the Nand gate output to deliver a positive-going output. Starting with a zero count, the positive-going output of the Nand gate advances counter IC2 one step, following the closure of S1.

Each audible report requires one to seven allophones. Following each report, the last two hex-data instructions in the program, 4 and 44, reset the speech processor internally. Also, by its output 06, the EPROM resets counter IC2 and flip-flop IC2b, making the circuit ready for the next reading.

## *Speech-Synthesized Burglar Alarms*

### **6.1** Designing a Burglar Alarm with Artificial Voice

The features found in most alarm systems for home or car usage can be built around one field programmable controller, a speech synthesizer, and several optocouplers. The vocal warning alarm described in this section takes advantage of the allophone-based speech processor SPO256-AL2 to warn the trespasser.

Figure 6.1 shows the entire circuit for the burglar alarm where a hidden switch (S1) inside the car or the house turns on the entire system. As can be seen, the performance of the alarm is based on the program stored in the FPC Am29CPL152. Figure 6.2 shows the flowchart indicating the steps that must be followed to detect a possible intrusion in the protected area.

We will design the alarm circuit to detect when a normally closed switch (S2) is open momentarily; therefore, the first step for the alarm is to check if the sensor is in the closed position. If so, the FPC will make the speech processor announce the message, “Check sensors...” repetitively until the problem has been fixed. When this happens, the operator will turn off the alarm before starting to adjust the input sensor. On the other hand, if the sensors are properly adjusted, when the alarm is turned on, the circuit inhibits it for 12 s before becoming active to let the user exit and close the door without problems. This delay is indicated in the flowchart of Figure 6.2 with the label “rick.” The 12 s delay is controlled by the routine located in lines 14 to 16 in Table 6.1.

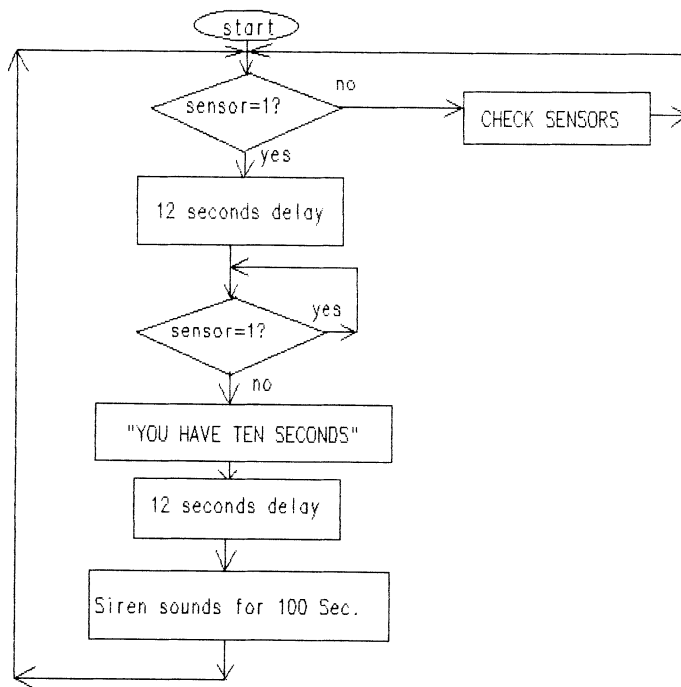
Now the alarm can detect, for example, the opening of a door or window



SPO256-AL2 by default. This set of instructions contain the allophones that have to be issued. Subroutine “read” sends the data byte zero to make the SPO256-AL2 start speaking the programmed allophone (see line 49). The instruction “if (not sby) then goto pl (sty6)” is used to make the FPC wait for the SPO256-AL2 to finish speaking each allophone.

When the FPC has completed sending sequentially the allophones for the message “You have ten seconds,” the FPC continues with the program in line 38 (see Table 6.1). Lines 38 to 40 contain the instructions to generate a 12 s delay. This delay is made by using the largest pause “pa5” available in the speech processor. Pause “pa5” makes the speech processor generate a 200 ms silence. In this form, the instruction “load pl(59)” in line 38 loads the CREG counter of the FPC with the constant 59 to generate the pause “pa5” 60 times; that is, a 12 s delay. The instruction “while (creg <> 0) loop to pl(stay)” is responsible for decrementing and testing CREG against zero until the 60 loops have taken place.

After the 12 s delay, the program activates the siren for 100 s, as can be seen in lines 41 to 47 of Table 6.1. This long delay is generated by loading the



**Figure 6.2** Flowchart for developing the microcode program for the burglar alarm with artificial voice.

**TABLE 6.1**  
Software Program for the Speech Synthesized Burglar Alarm

DEVICE (CPL152)

DEFAULT = 1;

DEFINE "test inputs"

sby = t1

sensor = t0

"output control bits"

"speech data = 59 allophones plus five pauses"

pa2 = 01#h pa3 = 02#h pa4 = 03#h pa5 = 04#h oy = 05#h ay = 06#h

eh = 07#h kk3 = 08#h pp = 09#h jh = 0A#h nn1 = 0B#h ih = 0C#h

tt2 = 0D#h rr1 = 0E#h ax = 0F#h mm = 10#h tt1 = 11#h dh1 = 12#h

iy = 13#h ey = 14#h dd1 = 15#h uw1 = 16#h ao = 17#h aa = 18#h

yy2 = 19#h ae = 1A#h hh1 = 1B#h bb1 = 1C#h th = 1D#h uh

= 1E#h uw2 = 1F#h aw = 20#h dd2 = 21#h gg3 = 22#h vv

= 23#h gg1 = 24#h

sh = 25#h zh = 26#h rr2 = 27#h ff = 28#h kk2 = 29#h kk1 = 2A#h

zz = 2B#h ng = 2C#h ll = 2D#h ww = 2E#h xr = 2F#h wh = 30#h

yy1 = 31#h ch = 32#h er1 = 33#h er2 = 34#h ow = 35#h dh2 = 36#h

ss = 37#h nn2 = 38#h hh2 = 39#h or = 3A#h ar = 3B#h yr = 3C#h

gg2 = 3D#h el = 3E#h bb2 = 3F#h ald = 100#h

siren = 200#h "P9"

led1 = 400#h; "P10"

TEST\_CONDITION = SBY;

DEFAULT\_OUTPUT = 0000#h;

BEGIN

"check for open loops"

"1"start: ,if (not sensor) then goto pl(rick);

"2" ch+led1, call pl(read); "CHECK SENSORS..."

"3" eh+led1, call pl(read);

"4" kk1+led1, call pl(read);

"5" pa5+led1, call pl(read);

"6" ss+led1, call pl(read);

"7" eh+led1, call pl(read);

"8" nn1+led1, call pl(read);

"9" ss+led1, call pl(read);

"10" or+led1, call pl(read);

"11" ss+led1, call pl(read);

"12" pa5+led1, call pl(read);

"13" led1, goto pl(start);

"14"rick: ,load pl(59); "12 sec delay to leave"

"15"stay1:pa5, call pl(read); "the house or car"

"16" ,while (creg <> 0) loop to pl(stay1);

"wait for input sensor S1 to go low"

"17"wait1: ,if (sensor) then goto pl(wait1);

"18" yy1, call pl(read); "YOU HAVE TEN SECONDS"

"19" ih, call pl(read);

"20" uh, call pl(read);

```

"21"      pa5,      call pl(read);
"22"      hh2,      call pl(read);
"23"      ae,       call pl(read);
"24"      vv,       call pl(read);
"25"      pa5,      call pl(read);
"26"      tt2,      call pl(read);
"27"      eh,       call pl(read);
"28"      eh,       call pl(read);
"29"      nn1,      call pl(read);
"30"      pa5,      call pl(read);
"31"      ss,       call pl(read);
"32"      ss,       call pl(read);
"33"      eh,       call pl(read);
"34"      kk3,      call pl(read);
"35"      nn1,      call pl(read);
"36"      zz,       call pl(read);
"37"      pa5,      call pl(read);

"38"              ,load pl(59);      "12 sec delay to turn alarm off"
"39" stay:pa5, call pl(read);      "pa5 = 200 mS"
"40"              ,while (creg <> 0) loop to pl(stay);
"
-----"
"          Routine to activate the siren for 100 seconds
"-----"
"41"      siren,      load pl(9);
"42"time: siren,      push (CREG);      "Push 9#d to TOS"
"43"      siren,      load pl(49);      "Delay = 200ms X 50 = 10 sec."
"44"stay2:siren+pa5, call pl(read2);
"45"      siren,      while (creg <> 0) loop to pl(stay2);
"46"      siren,      if (sby) then pop to(CREG);
"47"      siren,      while (creg <> 0) loop to pl(time);
"48"              ,goto pl(start);
"-----"
"subroutine READ"
"49"read:          ,continue;          "allows sby to go low in 300 ns"
"50"sty6:          ,if (not sby) then goto pl(sty6); "reading SBY"
"51"              ,ret;

"subroutine READ2"
"52"read2:      siren, continue;      "allows sby to go low in 300 ns"
"53"stay3:      siren, if (not sby) then goto pl(stay3); "reading
SBY"
"54"              siren, ret;

.org 127#d

"55"              ,goto pl(start);
END.

```

stack with the constant 9. This is achieved with the instructions “load pl(9)” and “push (creg),” respectively. Instructions in lines 43 to 45 are used to make the speech processor generate 50 times the pause “pa5”; that is, a 10 s delay; for instance, a total delay of 100 s is obtained. While the 100 s delay is being performed, the output control bit (P9) assigned “siren,” located in the left side of the instructions in lines 41 to 47, is utilized to activate the siren. The optocoupler 74OL6000 from Microchip is used to isolate the 5 V power supply from the 12 V line that is used to activate the siren. Transistor TIP120 is the power-switching device that drives the siren which normally consumes about 0.5 A at 12 Vdc.

Line 48 contains the instruction “goto pl(start)” in order to make the FPC jump to the instruction with the label “start” located in line one of Table 6.1. In this form, the FPC is ready to start the process again by monitoring the normally closed input sensor S1.

In this project, the FPC is clocked by a 1 kHz logic oscillator; that is, each instruction is performed in 1 ms. You can augment the 1 kHz frequency up to 30 MHz without altering the functions or delays of the alarm. That is possible because all the delays are generated by means of the speech processor SPO256-AL2.

By making use of the unused testable inputs from T2 to T5, the alarm presented here can be easily configured to also detect normally open (NO) sensors. In addition, you can activate more output devices by using the available outputs P9 to P15 of the FPC Am29CPL152 by using the appropriate interface circuitry.

Table 6.2 shows the PROM bit pattern generated by the assembler ASM14X. As you can see, outputs P0 to P7 are dedicated exclusively to store the speech data that the SPO256-AL2 requires to produce the messages. Output P9 in Table 6.2 contains a logic 1 in lines 40 to 46 to activate the siren. You can check this by looking at lines 41 to 47 in the program shown in Table 6.1. The same situation occurs to activate the LED that is connected to output P10 in Figure 6.1.

**TABLE 6.2**  
PROM Bit Contents for the FPC Am29CPL152

PROM Contents:							
hex	<dec>	OE	OPCODE	POL	TEST	DATA	OUTPUT
000	< 0>	[ 1	11001	1	0000	0001101	0000000000000000
001	< 1>	[ 1	11100	0	0001	0110000	0000010000110010
002	< 2>	[ 1	11100	0	0001	0110000	0000010000000111
003	< 3>	[ 1	11100	0	0001	0110000	0000010000101010
004	< 4>	[ 1	11100	0	0001	0110000	0000010000000100

```

005 < 5> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000010000110111 ]
006 < 6> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000010000000111 ]
007 < 7> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000010000001011 ]
008 < 8> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000010000110111 ]
009 < 9> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000010000111010 ]
00A < 10> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000010000110111 ]
00B < 11> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000010000000100 ]
00C < 12> [ 1 : 11001 : 0 : 0001 : 0000000 : 0000010000000000 ]
00D < 13> [ 1 : 00100 : 0 : 0001 : 0111011 : 0000000000000000 ]
00E < 14> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000100 ]
00F < 15> [ 1 : 01000 : 1 : 0111 : 0001110 : 0000000000000000 ]
010 < 16> [ 1 : 11001 : 0 : 0000 : 0010000 : 0000000000000000 ]
011 < 17> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000110001 ]
012 < 18> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000001100 ]
013 < 19> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000011110 ]
014 < 20> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000100 ]
015 < 21> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000111001 ]
016 < 22> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000011010 ]
017 < 23> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000100011 ]
018 < 24> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000100 ]
019 < 25> [ 1 : 11100 : 0 : 0001 : 0110000 : 00000000000001101 ]
01A < 26> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000111 ]
01B < 27> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000111 ]
01C < 28> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000001011 ]
01D < 29> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000100 ]
01E < 30> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000110111 ]
01F < 31> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000110111 ]
020 < 32> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000111 ]
021 < 33> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000100 ]
022 < 34> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000001011 ]
023 < 35> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000101011 ]
024 < 36> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000100 ]
025 < 37> [ 1 : 00100 : 0 : 0001 : 0111011 : 0000000000000000 ]
026 < 38> [ 1 : 11100 : 0 : 0001 : 0110000 : 0000000000000100 ]
027 < 39> [ 1 : 01000 : 1 : 0111 : 0100110 : 0000000000000000 ]
028 < 40> [ 1 : 00100 : 0 : 0001 : 0001001 : 0000001000000000 ]
029 < 41> [ 1 : 00101 : 0 : 0001 : 1111111 : 0000001000000000 ]
02A < 42> [ 1 : 00100 : 0 : 0001 : 0110001 : 0000001000000000 ]
02B < 43> [ 1 : 11100 : 0 : 0001 : 0110011 : 0000001000000100 ]
02C < 44> [ 1 : 01000 : 1 : 0111 : 0101011 : 0000001000000000 ]
02D < 45> [ 1 : 10111 : 0 : 0001 : 1111111 : 0000001000000000 ]
02E < 46> [ 1 : 01000 : 1 : 0111 : 0101001 : 0000001000000000 ]
02F < 47> [ 1 : 11001 : 0 : 0001 : 0000000 : 0000000000000000 ]
030 < 48> [ 1 : 01101 : 1 : 1111 : 1111111 : 0000000000000000 ]
031 < 49> [ 1 : 11001 : 1 : 0001 : 0110001 : 0000000000000000 ]
032 < 50> [ 1 : 00010 : 0 : 0001 : 1111111 : 0000000000000000 ]
033 < 51> [ 1 : 01101 : 1 : 1111 : 1111111 : 0000001000000000 ]
034 < 52> [ 1 : 11001 : 1 : 0001 : 0110100 : 0000001000000000 ]
035 < 53> [ 1 : 00010 : 0 : 0001 : 1111111 : 0000001000000000 ]
07F <127> [ 1 : 11001 : 0 : 0001 : 0000000 : 0000000000000000 ]

```



## 6.2 Designing a Simple Fault-Tolerant Burglar Alarm

When a burglar alarm is going to be used to protect expensive equipment, jewelry, premises, and so on, the probability of failure must be reduced to a minimum. One of the most annoying features of intrusion alarms is that they can be tripped by things other than an intrusion, such as failure of a component or an integrated circuit. The result is false alarms that are annoying and sometimes even dangerous. Because it is impossible to have components which are totally reliable, redundancy techniques must be employed to increase the probability of system survival or life of the system during the time  $t$ .

By using the burglar alarm circuitry of Figure 6.1 in triplicated form at the subsystem level in order to keep components to a minimum, a high reliability burglar alarm can be achieved (see Figure 6.3). The correction logic is made with four majority logic voters V1, V2, V3, and V4 configured for three inputs. The voters realize the function  $v(xyz) = xy + yz + xz$  by using a PAL programmed to contain a maximum of three voters per chip. The PAL selected for this function is the PAL16L4. The architecture and program to realize this voter were taken from Table 5.11. The scheme for the fault-tolerant burglar alarm is presented in Figure 6.3. Input errors and faults presented in each voter will be corrected; any single fault of each triplicated module is permitted. For example, if any one of the three normally closed (NC) sensors (S1 to S3) fails to open, voter V1 will mask the fault. The voter will respond only when any two or more sensors are triggered; that is, only when two or more Nand gates contained in the three optocouplers are presenting a logic zero at its outputs. Consider the case when a Nand gate contained in the optocoupler H11L1GE on the top is stuck at zero; such a fault is masked by voter V1 to avoid a failure in the system. In this case, the voter V1 will be waiting for sensors S2 and/or S3 to be triggered to generate a logic low at the respective output of the other two optocouplers.

Figure 6.3 also shows a TMR clock used for clocking the FPC Am29CPL152. The TMR clock uses three Schmitt-trigger Nand gates sharing the same RC network to build a 1 kHz logic oscillator. The output frequency obtained at the output of voter V2 is routed to a second set of three Nand gates that function as buffers. The output of this second set of Nand gates is voted by V3 and then connected to the clock input of the FPC Am29CPL152.

The FPC Am29CPL152 is used to monitor the logical status of voter V1. If voter V1 presents a positive transient pulse at its output, the FPC proceeds to execute the routine presented in Table 6.1. A small change has been included in the schematic of Figure 6.3 where the outputs for actuating the siren are now triplicated. Now, outputs P8, P9, and P10 are used to activate separately an optocoupler 74OL6000. These optocouplers already contain the constant current source required for driving the internal LED. The output of these op-



In this form the burglar alarm also tolerates any single fault in each power transistor (Q1 to Q4) to avoid sounding the siren for a simple bad transistor as happens in nonfault-tolerant burglar alarms.

### 6.3 Designing a Vocal Warning Alarm Using CMOS MSI Chips

By using readily available components, you can build a burglar alarm employing traditional design techniques with CMOS MSI chips. By programming the appropriate data into an EPROM memory, you can make your alarm vocalize the message you want. The circuit presented here is based on the principle used in Section 2.9 of the book that makes the speech processor speak a sequence of numbers that are stored in an EPROM memory.

The burglar alarm presented in Figure 6.4 is programmed to announce two different messages, each depending on the condition of the alarm. The timing section of the alarm is made around two Nand gates (1/2 4093) and two timers (T1 and T2) configured as monostables.

Action begins when switch S2 in Figure 6.4 is set to the ARM position. Closing this switch turns on dc power to all circuits and causes power-light emitting diode LED1 to turn on. Once power has been applied to the circuit, capacitor C1 begins to charge exponentially at a rate determined by the formula

$$T1 = R1C1 \ln [V_{dd}/(V_{dd} - V_{t+})]$$

substituting  $V_{dd} = 5 \text{ V}$ , and  $V_{t+} = 3.3 \text{ V}$ , we get

$$T1 = 1.08 R1C1$$

From this formula, it would take approximately 50 s to reach a threshold level of 3.3 V with the component values specified. When the threshold level is reached, the output at pin 4 of N2 goes high and enables the reset at pin 4 of timers T1 and T2. Consequently, you have approximately 30 s to vacate your premises or vehicle before the alarm sounds.

When the alarm is armed and in standby mode, if any attempt is made to gain entry by opening a door or window (or move your protected vehicle), it will cause timer T1 to trigger. When this occurs, timer T1 triggers on and its output at pin 3 signals T1 to turn on for almost 11 s (given by  $T2 = 1.1 R2C2$ ). The T1 period is the time you have to disarm the alarm before the siren sounds. During the 11 s period in which the timer is on, the speech processor will be vocalizing the countdown, starting at “nine . . .” and counting down to “. . . zero.” If the alarm has not been disabled before the count reaches zero, the output of timer T2 will turn on transistor Q1 via optocoupler 4N32 for a period of approximately 220 s (given by  $T3 = 1.1 R3C3$ ). This is how long the siren will sound before the alarm shuts off and resets itself. When monostable timer T2 shuts off, it goes into standby until the next pulse



the output 06 of EPROM 27C16, which resets counter CD4520 every time the speech processing chip's output at pin 24 ceases sending its message to be vocalized to the audio-amplifying section of the circuit (not shown). Chapter 2 shows the schematic for the audio amplifier that is currently used by the SPO256-AL2.

In this project, we use addresses 128 through 172 (80H to ACH) in EPROM 27C16 to store the data for the first message to be vocalized and addresses 256 through 306 to store the data for the second message to be vocalized. By connecting seven address outputs of counter CD4520 to the binary address inputs of EPROM 27C16 as shown, the desired sequential data programmed into the EPROM can be sent via the output lines (O0 to O5) to the address inputs of speech processor SPO256-AL2.

Table 6.3 is an example of the data program that can be fed into the EPROM to make the speech processor vocalize the first message. Most of the words used in this data program were taken from the technical dictionary presented in Chapter 2, Section 3. The addresses were calculated according to the

**TABLE 6.3**  
EPROM Program for First Message

Hex address	Hex data	Hex address	Hex data
80	4	96	2
81	31	97	37
82	1F	98	3
83	2	99	D
84	39	9A	16
85	7	9B	3
86	23	9C	3E
87	3	9D	13
88	1D	9E	23
89	34	9F	4
8A	2	A0	4
8B	D	A1	4
8C	13	A2	9
8D	2	A3	2D
8E	3	A4	13
8F	37	A5	37
90	37	A6	2
91	7	A7	1B
92	2	A8	18
93	2A	A9	3B
94	C	AA	13
95	B	AB	4
<i>Continued</i>		AC	40

**TABLE 6.4**  
EPROM Program for Second Message

Hex address	Hex data	Hex address	Hex data
100	38	119	28
101	18	11A	6
102	6	11B	23
103	B	11C	3
104	2	11D	28
105	14	11E	28
106	2	11F	3A
107	D	120	3
108	2	121	10
109	37	122	E
10A	37	123	13
10B	7	124	4
10C	7	125	D
10D	23	126	D
10E	C	127	4
10F	B	128	39
110	2	129	F
111	37	12A	F
112	37	12B	B
113	C	12C	4
114	2	12D	2B
115	29	12E	3C
116	37	12F	35
117	3	130	4
118	28	131	2
<i>Continued</i>		132	40

binary weight of the two lines A7 and A8 address inputs of the EPROM 27C16. For example, when pin 1 of EPROM 27C16 is at a logic one, the binary address is 128. The first message is programmed so that it is stored in address locations 128 through 201 (see Table 6.3) where the numbers 1 through 4 and 40 reset the speech processing chip and the binary counter, respectively. If address input A8 of EPROM 27C16 is at a logic one, the second message can be heard starting at address 256 (see Table 6.4).

When the alarm circuit is first turned on, the R4C4 network sends a positive transient pulse that sets high output Q1 of flip-flop FF1. Because this high is fed to an OR gate configuration made up of two diodes 1N914 (D1 and D2), the output of these OR-wired diodes enables Nand gate N3 to send the pulse that appears to the /ALD input of the speech processor and to the clock input of counter CD4520.

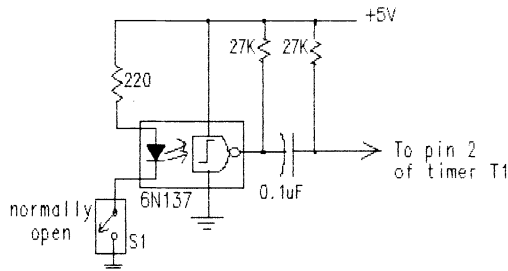
Since the Q output of flip-flop FF1 is at a logic one, the same high logic

level is delivered to address input A7 at pin 1 of EPROM 27C16; therefore, the EPROM sends the data stored in it starting at address 128 of EPROM. At this point, you will hear the message: "You have thirty seconds to leave. Please hurry." When this message ends, the EPROM sends a logic one via its output O6 to the reset inputs R1 and R2 of both flip-flops (FF1 and FF2) and causes a logic zero to appear at the Q outputs.

Now let us suppose that someone attempts to open the door of the protected area. The normally closed (NC) sensor S1 would now trigger timer T1 by causing a logic zero at the output of the logic optocoupler H11L1GE, which generates a negative transient pulse at the trigger input of timer T1. Timer T1 then sends, through the R3C3 network, a positive transient pulse to the pin 8 set input of the second flip-flop (FF2). This causes the output Q2 of FF2 to go to logic one, enabling the Nand gate N3. Because the standby output is normally at logic one, the output of Nand gate N3 will go to a logic zero to make the speech processor announce the first allophone. When the speech processor is speaking the first allophone, the standby (/SBY) output goes to a logic zero, causing a logic one at the Nand gate's output. When this happens, counter CD4520 is incremented by one to address the next sequential allophone that the EPROM loads into the speech processor. When the speech processor ends saying the first allophone, the standby output goes back to a logic one and causes the /ALD input to go low again; therefore, the speech processor starts speaking the allophone previously loaded by the EPROM 27C16, starting at address 256. Hence, you will now hear the second message: "Nine, eight, seven, six, five, four, three, two, one, zero."

From the time this second message starts, you have 10 s to disarm the alarm by setting S2 to DISARM; otherwise, the siren will sound. Notice that the siren is driven by a power pair-Darlington transistor TIP120. The optocoupler 4N32 is used to isolate the siren circuit from the timing section to prevent feedback noise to trigger timers T1 and T2.

As can be seen in Figure 6.4, this project can accommodate normally closed sensors (NC) only. To install normally open sensors, use the input network shown in Figure 6.5.



**Figure 6.5** Network for normally open sensors.





**TABLE 6.5**  
Software Program for the Microcontroller 8748

Add	Op	Code	Mnemonic	Comments
				;Software program for the vocal
				;warning burglar alarm.
00H	99	00	ANL P1, #00H	;clear port 1
02	9A	00	ANL P2, #00H	;clear port 2
04	46	12	STRT: JNT1 MS1	;jump to address MS1 if T1 is low
06	BC	2B	MOV R4, #2BH	;44 allophones are contained in 2nd
				;message.
08	BD	10	MOV R5, #10H	;Pointer for ROM in page 3.
0A	14	35	CALL MSG	;
0C	14	27	CALL DELAY	;
				-----
0D	56	0D	NC:JT1 0DH	;wait for T1 to go low
0F	BC	30	MOV R4, #30H	;load Reg 4 with # of allophones for
				;message "Nine, eight,.... zero."
11	BD	3D	MOV R5, #3DH	;Pointer for ROM in page 3
13	14	35	CALL MSG	;
15	8A	01	ORL P2, #01H	;Siren is activated for 60 seconds.
17	14	27	CALL DELAY	;
19	14	27	CALL DELAY	;
1B	9A	00	ANL P2, #00H	;
1D	04	0D	JMP NC	;
				-----
1F	BC	30	MS1:MOV R4, #0EH	;load reg R4 with # of allophones
for 21	BD	00	MOV R5, #00H	;message "Check sensors"
23	14	35	CALL MSG	;
25	04	04	JMP STRT	;
				-----
27	B8	5C	DELAY:MOV R0, #5CH	;30 second delay
29	B9	FF	T4:MOV R1, #FFH	;
2B	BA	FF	T3:MOV R2, #FFH	;
2D	EA	2D	T2:DJNZ R2, T2	;
2F	E9	2B	DJNZ R1, T3	;
31	E8	29	DJNZ R0, T4	;
34	83		RET	;
				-----
35	FD		MSG:MOV A, R5	;Subroutine for sending alloph.
37	E3		MOV P3 A, @A	;move page 3 of ROM to ACC
39	39		OUTL P1, A	;load speech data to port 1
3B	90		MOVX @R0, A	;/WR output is pulsed low
3C	26	3C	JNTO 3CH	;read the SBY status of the SP
3E	1D		INC R5	;inc reg 5 to select new
3F	00		NOP	;allophone
40	EC	35	DJNZ R4, 35H	;decrement the number of allophones
42	83		RET	;
				-----

```

;Page 3 of ROM
;1st message: "Check sensors"
300 04
301 32
302 7
303 7
304 2
305 29
306 3
307 37
308 37
309 7
30A 7
30B B
30C 37
30D 3A
30E 37
30F 4
310 4
311 31
312 1F
313 2
314 39
315 7
316 23
317 3
318 1D
319 34
31A 2
31B D
31C 13
31D 2
31F 3
320 37
321 37
322 7
323 2
324 2A
325 C
326 B
327 2
328 37
329 3
32A D
32B 16
32C 3
32D 3E
32E 13
32F 23
330 4
331 4
;2nd message: "You have thirty
; seconds to leave, please hurry"

```

332	4	;
333	9	;
334	2D	;
335	13	;
336	37	;
337	2	;
338	1B	;
339	18	;
33A	3B	;
33B	13	;
33C	4	;
33D	38	;3rd message: "Nine, eight, seven,"
33E	18	;six, five, four, three, two, one,
33F	6	;zero."
340	B	
341	2	
342	14	
343	2	
344	D	
345	2	
346	37	
347	37	
348	7	
349	7	
34A	23	
34B	C	
34C	D	
34D	2	
34E	37	
34F	37	
350	C	
351	2	
352	29	
353	37	
354	3	
355	28	
356	28	
357	6	
358	23	
359	3	
35A	28	
35B	28	
35C	3A	
35D	3	
35E	10	
35F	E	
360	13	
361	4	
362	D	
363	D	
364	4	

365	39
366	F
367	F
368	B
369	4
36A	2B
36B	3C
36C	35
36D	4

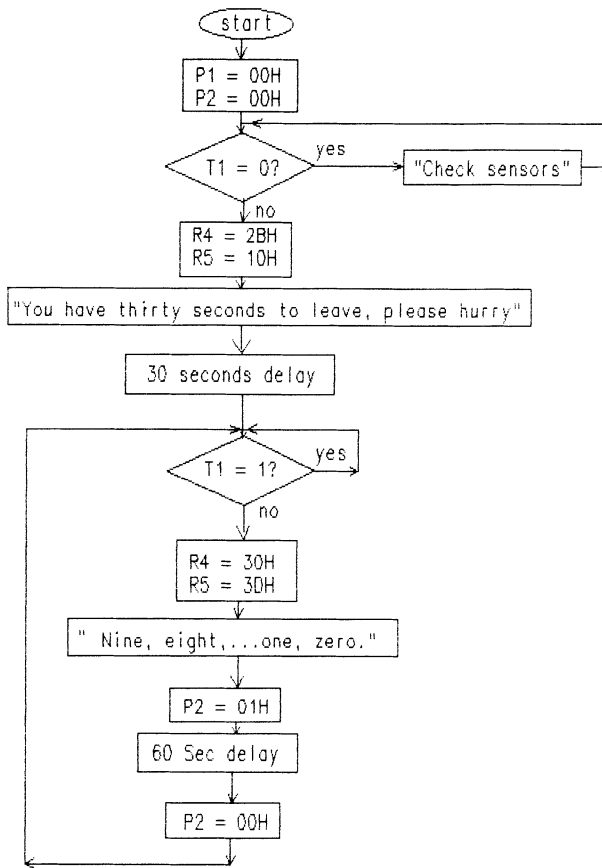
When the user sets the DPDT switch (S2) to the ARM position, the dc voltage will be present in the entire system. For instance, the  $\mu$ C 8748 will automatically initiate executing the control program.

At this point, we want the  $\mu$ C program to drive the speech processor by sending a preprogrammed group of allophones. In this case, we will program it to speak three messages:

1. "Please check sensors"
2. "You have thirty seconds to leave. Please hurry"
3. "Nine, eight, seven, six, five, four, three, two, one, zero"

The speech data will be programmed in page three of the  $\mu$ C's internal EPROM. We know that page three consists of 256 bytes available for speech data programmed by the user. In this manner, a maximum of 256 allophones can be stored for any other set of specific messages. In this case, the three messages spend a total of 109 bytes.

The following routine corresponds to the flowchart shown in Figure 6.7. It is a good example of how the  $\mu$ C 8748 is instructed to make the speech processor speak four-word numbers. In Figure 6.7, the first upper box clears output ports P1 and P2 of the  $\mu$ C 8748. The flowchart then starts asking for the logic status of the input sensor that is connected to testable input T1 (see Figure 6.7). If the sensor is well adjusted in normally closed position, the testable input T1 is held at a logic one. On the other hand, if T1 is low, the program jumps to label "MS1" to make the speech processor announce the message, "check sensors." If the program finds that the input sensor is normally closed, it will load registers R4 and R5 with the number of allophones and the location of the second message, respectively. With registers R4 and R5 loaded, the program calls out the routine that contains the second message, "You have thirty seconds to leave. Please hurry." Once the second message has been enunciated, the flowchart calls the routine "DELAY" to generate a 30 s delay to give the user time to leave and close the door where the sensor is attached. After the 30 s delay, the program keeps monitoring the logic status of the normally closed sensor via the testable input T1. Now, when someone attempting



**Figure 6.7** Flowchart utilized to develop the routine that controls the micro-controller-operated burglar alarm.

to break in opens the (NC) sensor momentarily, the routine immediately loads registers R4 and R5 to start generating the third message, “Zero, nine, eight, seven, six, five, four, three, two, one, zero.” The countdown sequence is to alert the user that he has 10 s to deactivate the alarm by turning off the hidden DPDT switch S2. When this message ends, the siren is activated via the output P2.0 for a period of 60 s. The 60 s period is achieved by calling two times the “DELAY” subroutine.

When the siren is turned off after the 60 s period, the program jumps back again to keep monitoring the input sensor. In this form, the alarm is ready to be triggered again in case someone attempts to break in. If your application requires several (NC) sensors, just connect them in series forming a loop. You can also monitor several (NC) loops by using the input pins available in port P2 and BUS. In this manner, you can program the alarm to announce the loop



be installed in the front door. (NC) sensor S2 will be installed in the back door. Depending on what sensor is triggered, the alarm will enunciate two messages:

1. "Front door sensor activated."
2. "Back door sensor activated."

When the alarm is first turned on by setting the switch SA to the ARM position, it will check for open sensors first. If sensor S1 is found to be open, the alarm will issue the message, "Check front door sensor." Accordingly, the message "Check back door sensor" will be announced when sensor S2 is open.

If both sensors S1 and S2 (designated as SENS1 and SENS2 in Table 6.6) are in normal position, the alarm will keep monitoring the status of both sensors. If sensors S1 or S2 are opened momentarily by an intrusion, a logic low will be detected at the testable inputs T0 or T1 of the FPC Am29CPL152. The program of the FPC then detects which sensors were opened and starts announcing via the speech processor the corresponding message: "Front door sensor activated" or "Back door sensor activated." When this message ends, the alarm gives a 12 s delay to allow the owner to disarm the alarm by turning

**TABLE 6.6**  
Software Program for the Speech Synthesized Burglar Alarm

```

DEVICE (CPL152)

DEFAULT = 1;
DEFINE "test inputs"
sby = t6
sens1 = t0
sens2 = t1

                "output control bits"
                "speech data = 59 allophones plus five pauses"
pa2 = 01#h pa3 = 02#h pa4 = 03#h pa5 = 04#h oy = 05#h ay = 06#h
eh = 07#h kk3 = 08#h pp = 09#h jh = 0A#h nn1 = 0B#h ih = 0C#h
tt2 = 0D#h rr1 = 0E#h ax = 0F#h mm = 10#h tt1 = 11#h dh1 = 12#h
iy = 13#h ey = 14#h dd1 = 15#h uw1 = 16#h ao = 17#h aa = 18#h
yy2 = 19#h ae = 1A#h hh1 = 1B#h bb1 = 1C#h th = 1D#h uh
= 1E#h uw2 = 1F#h aw = 20#h dd2 = 21#h gg3 = 22#h vv
= 23#h gg1 = 24#h
sh = 25#h zh = 26#h rr2 = 27#h ff = 28#h kk2 = 29#h kk1 = 2A#h
zz = 2B#h ng = 2C#h ll = 2D#h ww = 2E#h xr = 2F#h wh = 30#h
yy1 = 31#h ch = 32#h er1 = 33#h er2 = 34#h ow = 35#h dh2 = 36#h
ss = 37#h nn2 = 38#h hh2 = 39#h or = 3A#h ar = 3B#h yr = 3C#h
gg2 = 3D#h el = 3E#h bb2 = 3F#h ald = 100#h
siren = 200#h;

TEST_CONDITION = SBY;
DEFAULT_OUTPUT = 0000#h;

```

```

BEGIN
"check for open loops"

"1"start:      ,cmp tm(000011#b) to pl(03#h);
"2"           ,if (equal) then goto pl(sty);
"3"           ,if (sens1) then call pl(front);
"4"           ,if (sens2) then call pl(back);
"5"           ,goto pl(start);

"6"rick:      ,load pl(59); "12 sec delay to leave the house"
"7"stay1:pa5,  call pl(read);
"8"           ,while (creg <> 0) loop to pl(stay1);

"wait for input sensors S1 or S2 to go low"
"9"sty:       ,cmp tm(000011#b) to pl(03#h);
"10"          ,if (equal) then goto pl(sty);
"11"          ,if (sens1) then goto pl(msga);
"12"          ,if (sens2) then goto pl(msgb);
"13"          ,load pl(59); "12 sec delay to turn alarm off"
"14" stay:pa5, call pl(read); "pa5 = 200 mS"
"15"          ,while (creg <> 0) loop to pl(stay);

"-----"
"          Routine to activate the siren for 100 seconds          "
"-----"
"16"      siren,      load pl(9);      "time to keep the"
"17"time: siren,      push (creg);      "Push 40#d to top of stack"
"18"      siren,      load pl(49);      "Delay = 200ms X 50 = 10 sec."
"19"stay1:siren+pa5,  call pl(read2);
"20"      siren,      while (creg <> 0) loop to pl(stay1);
"21"      siren,      if (sby) then pop to(CREG);
"22"      siren,      while (creg <> 0) loop to pl(time);
"23"          ,goto pl(start);
"-----"

"subroutine READ"
"24"read:      ,continue;      "allows sby to go low in 300 ns"
"25"sty6:      ,if (not sby) then goto pl(sty6); "reading SBY"
"26"          ,ret;

"subroutine READ2"
"27"read2:    siren,  continue;      "allows sby to go low in 300 ns"
"28"stay3:    siren,  if (not sby) then goto pl(stay3); "reading
SBY"
"29"          siren,  ret;

"30"msga:      ,call pl(front);      "FRONT..."
"31"msgab:     ,call pl(door);      "DOOR"
"32"          ,call pl(sensor);      "SENSOR..."
"33"          ,call pl(activ);      "ACTIVATED"
"34"          ,goto pl(sty);
"35"msgb:      ,call pl(back);      "BACK..."
"36"          ,goto pl(msgab);

"37" check:    ch,      call pl(read); "CHECK..."
"38"          eh,      call pl(read);
"39"          kk1,     call pl(read);
"40"          pa5,     call pl(read);
"41"          ,ret;

```



```

"42" sensor:  ss,      call pl(read); "SENSORS. ."
"43"          eh,      call pl(read);
"44"          nn1,     call pl(read);
"45"          ss,      call pl(read);
"46"          or,      call pl(read);
"47"          ss,      call pl(read);
"48"          pa5,     call pl(read);
"49"                                     ,ret;
"50" front:   ff,      call pl(read); "FRONT. ."
"51"          rr2,     call pl(read);
"52"          ow,      call pl(read);
"53"          nn1,     call pl(read);
"54"          tt1,     call pl(read);
"55"          pa5,     call pl(read);
"56"                                     ,ret;
"57" back:    bb1,     call pl(read); "BACK. ."
"58"          ae,      call pl(read);
"59"          kk2,     call pl(read);
"60"          pa5,     call pl(read);
"61"                                     ,ret;
"62" activated: ax,     call pl(read);
"63"          kk3,     call pl(read);
"64"          tt2,     call pl(read);
"65"          ih,      call pl(read);
"66"          vv,      call pl(read);
"67"          ey,      call pl(read);
"68"          rr2,     call pl(read);
"69"          dd1,     call pl(read);
"70"                                     ,ret;
"71" door:    dd2,     call pl(read); "DOOR. ."
"72"          or,      call pl(read);
"73"          pa5,     call pl(read);
"74"                                     ,ret;

                                .org 63#d
"71"                                     ,goto pl(start);
END.

```

the hidden switch SA to DISARM position. If the switch SA is not set to DISARM position, the siren will be actuated for a period of 100 s. After this long interval, the alarm will go back to monitor the status of the sensors in the front and back door. If these sensors are still open, it will announce the respective message, and after a 12 s delay, it will activate the siren again. This process is repeated until sensors S1 and/or S2 are found in the closed position.

Because we are using the FPC Am29CPL152, there are five more testable inputs (T2 to T6) available for monitoring more sensors. These available inputs can be used, for example, for detecting the entrance in five different windows. In this case, the user can augment or modify the length of the messages

that have to be announced when someone tries to break in. There is sufficient memory space in the FPC Am29CPL152 to augment or adapt the program to your particular application. Table 6.6 shows the entire microcode program that controls the vocal warning burglar alarm already described in this section.

## *Voice Recognition Chips*

### **7.1 Introduction to Voice-Recognition Techniques**

Speech-recognition systems, the subject of research for more than 20 years, are becoming more commonplace. Today, many companies offer recognition systems that allow you to enter data or commands into a computer using the human voice. Background noise, misunderstood words, or words the unit cannot identify become an important factor when selecting a speech-recognition system, for example, when the manufacturer offers only a limited number of words that can be recognized by the system. The problems associated with speech recognition make this technology more expensive than its counterpart (speech synthesis) because of the engineering effort and the technology employed. Microchip and National Semiconductor offer speech recognition chips, as do many other companies.

Speech recognition can be applied in the following areas: test stations in factories, data entry, office automation, and as an aid to the disabled. For example, consider the case of a voice-controlled wheelchair where the user gives spoken commands through a headset microphone. This type of controller will be explained in detail in Section 4 of this chapter.

Speech-recognition systems are divided into two categories: speaker dependent and speaker independent. In speaker-dependent systems, the user has to train the device so it can identify words, sounds, or phrases. On the other hand, speaker-independent products can recognize speakers with different pitch, accent, or both without any operator training. Speaker-independent products have the disadvantage of handling a smaller vocabulary for recogni-

tion. Ideally, a truly general voice-recognition system would be capable of recognizing a large vocabulary of words, independently of the speaker.

An effective speech-recognition system must be capable of ignoring the physical factors that cause variations in the speech waveform from speaker to speaker or even in the same person. The changes contained in a speech waveform are related to frequency, relative amplitude, and time duration.

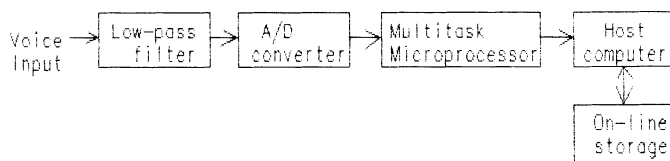
Figure 7.1 shows a block diagram containing the steps commonly used in most voice-recognition systems. A powerful microprocessor can realize the functions contained in the block diagram, that is:

- Conversion of the voiced input analog signal into a digital form by sampling.
- Compression or selection of relevant data for subsequent processing.
- Computation of the boundaries of the word.
- Detection of patterns within the word.
- Pattern classification.
- Association of pattern sequences with words in the vocabulary.

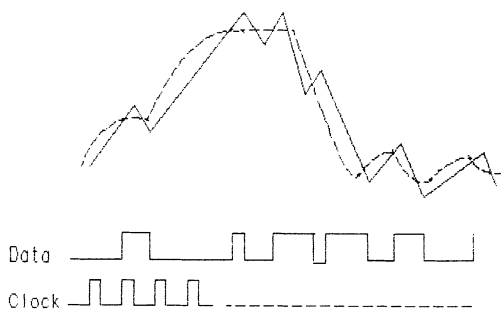
The input stage must be formed by a microphone preamplifier in series with a low-pass filter used to attenuate frequencies above 8 kHz. The A/D converter transforms the voice input to a digital representation with respect to a period of time. The processor will now detect maximum and minimum signal peaks and stores their amplitude and the time interval between peaks. The selected data are compressed in order to decide whether or not they constitute a pattern in order to generate parameters based upon the type of pattern. Finally, the host computer uses the sequence of parameters to determine which word has been spoken. Depending on the application of the speech-recognition system, the real system will be controlled once the spoken word has been recognized.

The sampling rate and the desired time resolution will determine the speed requirement and the number of bits of the A/D converter. The flash converter technique is fast enough for sampling speech. Because a minimum sampling rate of 6 kHz is required, the conversion time must be equal to or less than  $166.7 \mu\text{s}$ .

Delta modulation (DM) must be used to reduce the amount of data to avoid overloading a processor. This approach encodes the differential change in the



**Figure 7.1** Block diagram of a common speech-recognition system.

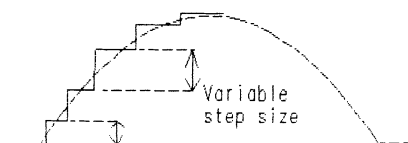


**Figure 7.2** Waveform sampling by delta modulation.

signal magnitude between sampling intervals. If the differential is less than a preset value, no data are recorded; therefore, this technique reduces the amount of data for slowly varying signals. Because voice waveforms contain much redundant data, long periods of silence are interspersed with sounds that vary in pitch slowly. The DM process assumes that the input voice waveform has a fairly uniform and predictable slope. Rather than storing 8- or 12-bit data for each sample, a DM stores only a single bit. When the processor samples the input signal from the A/D converter, it compares the current reading to the preceding sample. If the amplitude of the new sample is greater, the processor stores a bit value of 1. Conversely, if the new sample is less, a 0 will be stored. Figure 7.2 shows how this works.

In practice, a DM can be implemented with an integrator having a constant input with reversible polarity. The precision of this method depends upon the clock rate and the magnitude of the integrated voltage. This way, the amount of data can be reduced significantly, as low as 2 kbyte/s for speech signals. One disadvantage of this technique is that only a single bit changes between samples. The sampling rate must be fast enough that no significant information is lost from the input signal.

The technique called differential pulse-code modulation (DPCM) permits more than a single bit of difference between stored samples; therefore, it permits more variation in the input waveform before severe distortion sets in. (See Figure 7.3)



**Figure 7.3** Waveform sampling by differential pulse-code modulation (DPCM).

Today, some of the fastest flash A/D converters have sample rates between 100 and 500 MHz.

Neural networks are particularly suited to applications in pattern recognition such as speech processing, image recognition, and robot control. A neural network can be used to detect the presence of speech in noise. Reference 2 refers to an article that presents neural networks as an alternative approach to determine the feature sets (collections of signals attributes) that correspond to particular speech characteristics. These feature sets are used in recognizing, compressing, or otherwise processing speech signals.

Linear predictive coding (LPC) is an important factor for synthesizing, recognizing, and coding speech. The LPC technique reduces the algorithms and filter structures needed to generate synthesized speech. In speech recognition, the LPC parameters contain reliable and repeatable features with which to identify sounds and words.

For the reader interested in the techniques generally employed in speech recognition, References 1 to 3 at the end of this chapter are an excellent source.

## 7.2 The Word-Recognizer VCP200

In this section we will see the word-recognizer VCP200 that is distributed by Radio Shack stores. The purpose is to understand its principle and modes of operation in order to interface it to different control systems.

The VCP200 is a speaker-independent IC that recognizes a limited number of words spoken by an operator. It recognizes the spoken words by breaking them into broad phoneme classes and then identifying predetermined strings of these classes. The VCP200 realizes the following functions:

- Performs spectral analysis of the incoming voice signal over the range of 300 to 5500 Hz.
- Determines the membership of phoneme classes by evaluation on spectral shape.
- Forms strings of these classes.
- Compares the strings to the stored strings of selected words.

The VCP200 recognizes the selected spoken commands in real time.

There are two recognition modes for the VCP200: command mode and yes-no/on-off mode. In the command mode, it recognizes five words:

1. Go
2. Stop
3. Left turn
4. Turn right
5. Reverse

In the yes-no/on-off mode it recognizes two words:

1. Yes
2. No

or

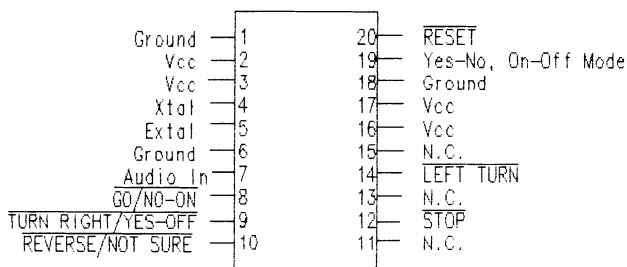
1. On
2. Off

Figure 7.4 shows the pin configuration of the VCP200. When a spoken word is recognized by the VCP200, the respective output for that word provides a latched logic low. In the yes-no/on-off mode, pin 10 of the VCP200 (REVERSE/NOT SURE) indicates when a word has not been recognized.

Figure 7.5 shows the typical circuit used for interfacing a microphone to the VCP200. The audio amplifier is designed to deliver extremely square-wave pulses representing the analog voice input of the microphone. The gain and frequency characteristics of the amplifier can be adjusted according to the following situation:

1. If the ambient noise level is low, augment the gain of the amplifier by selecting the following resistor values:  $R_f = 30K$ ,  $R_b = 10K$ ,  $R_c = 10K$ .
2. If background noise such as caused by fans, motors, or air conditioning is present, adjust the input amplifier's gain, so that the output signal at the second amplifier does not exceed 1 V peak-to-peak with no speech input.

The performance of the microphone depends on its location relative to the person speaking. Notice that high-frequency loss above 4 kHz occurs when the distance of the microphone to the operator increases. To avoid high-frequency loss, a headset microphone is recommended. The VCP200 is designed to ignore some words not contained in its vocabulary by discriminating among



**Figure 7.4** Pin configuration of the word-recognizer VCP200.







**TABLE 7.1**  
Outputs Generated by the Word-Recognizer VCP200  
Needed to Control a Direct Current Motor

Command	GO	REVERSE	STOP	Q1	Q2	Q3	Q4
STOP	H	H	L	OFF	OFF	OFF	OFF
GO	L	H	H	ON	ON	OFF	OFF
REVERSE	H	L	H	OFF	OFF	ON	ON

output for indicating the STOP position. The output GO is utilized to actuate the motor in the forward direction. The output REVERSE is used to power the motor in the reverse direction. In this form, to actuate or stop the motor the operator needs to pronounce only three commands: GO, REVERSE, and STOP. Table 7.1 shows the logic level outputs that are generated by the VCP200 when a spoken command is pronounced by an operator. Table 7.1 shows that transistors Q1 and Q2 actuate the motor in the forward direction, while transistors Q3 and Q4 control the motor in the reverse direction. Notice that not all four transistors can be turned on at the same time because a short would occur in the 12 V dc power supply. Two pair-Darlington optocouplers are required to actuate each pair of transistors. The optocoupler helps to keep the control circuit isolated from the 12 V power supply which will operate the motor in the presence of the noise generated by the motor M.

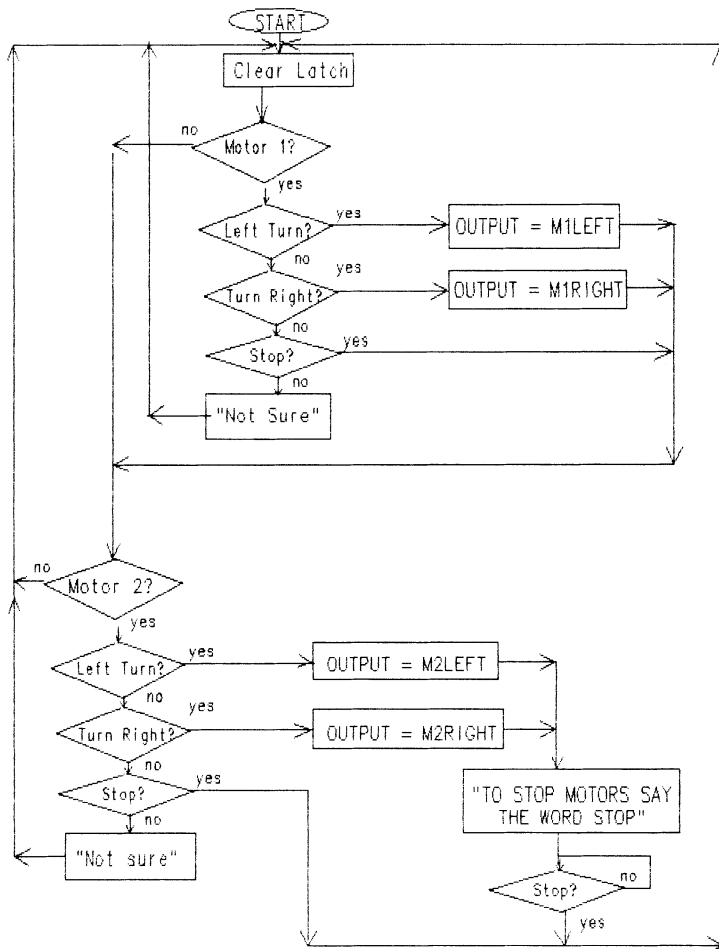
To start the motor, the operator will pronounce the direction that he needs; that is, GO or REVERSE. Once the motor is running in the forward direction, for example, he will have to stop the motor by pronouncing the STOP command before attempting to pronounce the word REVERSE to change the motion of the motor.

If you need to control several dc motors by voice commands, the circuit presented in the following section contains all the necessary hardware and software to achieve that specific task.

## 7.4 How to Control Direct Current Motors with a Word Recognizer and a Speech Synthesizer

An application which involves voice recognition and speech synthesis is now to be described. The circuit shown in Figure 7.7 controls two dc motors, M1 and M2, using the word-recognizer VCP200 interfaced to an FPC. The FPC controls the direction of both dc motors and the speech processor SPO256-AL2.





**Figure 7.8** Flowchart employed by the FPC Am29CPL154 to control two dc motors by voice commands.

The flowchart in Figure 7.8 starts by applying a reset pulse to the quad tri-state R/S latch CD4043. This reset pulse avoids turning a motor when the entire circuit is first turned on. Then the SPO256-AL2 will issue the message “Motor one, yes or no . . .” to indicate to the operator if he wants to actuate motor M1. If so, he has to speak the word “Yes.” Now the program will be waiting for one of the three possible answers of the operator: Left Turn, Turn Right, and Stop. Depending on the word pronounced by the operator, motor M1 will be actuated. If the voice recognizer VCP200 is not sure of the word pronounced by the operator, the SPO256-AL2 will say the message “Not

**TABLE 7.2**

Software Program for the FPC Am29CPL154 to Control Two Direct Current Motors

DEVICE (CPL154)

DEFAULT = 1;

DEFINE "test inputs"

SBY = T3

GO = T0

LEFTTURN = T1

TURNRIGHT = T2

STOP = T4

"output control bits"

"speech data = 59 allophones plus five pauses"

pa2 = 01#h pa3 = 02#h pa4 = 03#h pa5 = 04#h oy = 05#h ay = 06#h

eh = 07#h kk3 = 08#h pp = 09#h jh = 0A#h nn1 = 0B#h ih = 0C#h

tt2 = 0D#h rr1 = 0E#h ax = 0F#h mm = 10#h tt1 = 11#h dh1 = 12#h

iy = 13#h ey = 14#h dd1 = 15#h uw1 = 16#h ao = 17#h aa = 18#h

yy2 = 19#h ae = 1A#h hh1 = 1B#h bb1 = 1C#h th = 1D#h uh

= 1E#h uw2 = 1F#h aw = 20#h dd2 = 21#h gg3 = 22#h vv

= 23#h gg1 = 24#h

sh = 25#h zh = 26#h rr2 = 27#h ff = 28#h kk2 = 29#h kk1 = 2A#h

zz = 2B#h ng = 2C#h ll = 2D#h ww = 2E#h xr = 2F#h wh = 30#h

yy1 = 31#h ch = 32#h er1 = 33#h er2 = 34#h ow = 35#h dh2 = 36#h

ss = 37#h nn2 = 38#h hh2 = 39#h or = 3A#h ar = 3B#h yr = 3C#h

gg2 = 3D#h el = 3E#h bb2 = 3F#h ald = 100#h

"-----"

"Output control bits for motors M1 and M2"

M1RIGHT = 200#h "P9"

M1LEFT = 100#h "P8"

M2RIGHT = 800#h "P11"

M2LEFT = 400#h "P10"

MODE1 = 8000#h "Yes - No are recognized by the VCP200"

LATCH= 1000#h; "P12 loads data into Latch 4042"

TEST-CONDITION = SBY;

DEFAULT-OUTPUT = 0000#h;

BEGIN

"The SP asks for the motor to be actuated"

"2"start:LATCH, continue; "Clears Latch 4042"

"3" ,call pl(msgmtr);

"4" ww, call pl(read); "One...? "

"5" ax, call pl(read);

"6" ax, call pl(read);

"7" nn1, call pl(read);

"8" pa5, call pl(read);

"9" pa5, call pl(read);

"10" ,call pl(msgyn); "Yes or No...?"

"-----"

" This routine is executed for 6 seconds "

"11" ,load pl(24); "5-sec delay"

```

"12"stay:pa5,      call pl(read);
"13"      model,   continue;
"14"      model,   if (GO) then goto pl(MOT2);"If no then..."
"15"                                ,while (creg <> 0) loop to pl(stay);
"-----"
"16"                                ,if (LEFTTURN) then goto pl(MOT1L);
"17"                                ,if (TURNRIGHT) then goto pl(MOT1R);
"18"                                ,if (STOP) then goto pl(MOT2);
"19"                                ,call pl(msgnts);                  "Not Sure"
"20"                                ,goto pl(start);
"-----"
"21"                                ,load pl(24); "5-sec delay"
"22"stay1:pa5,     call pl(read);
"23"MOT2:model,    continue;
"24"      model,   if (GO) then goto pl(start);"If not then..."
"25"                                ,while (creg <> 0) loop to pl(stay1);
"-----"
"26"                                ,if (LEFTTURN) then goto pl(MOT2L);
"27"                                ,if (TURNRIGHT) then goto pl(MOT2R);
"28"                                ,if (STOP) then goto pl(start);
"29"                                ,call pl(msgnts);
"30"                                ,goto pl(start);

"subroutine READ"
"31"read:          ,continue;          "allows sby to go low in 300 ns"
"32"sty6:          ,if (not sby) then goto pl(sty6); "reading SBY"
"33"              ,ret;

"subroutine READ2"
"34"read2:         ,continue;          "allows sby to go low in 300 ns"
"35"stay3:         ,if (not sby) then goto pl(stay3); "reading SBY"
"36"              ,ret;

"37"MOT1L:         m1left,             continue;
"38"              ,goto pl(MOT2);
"39"MOT1R:         m1right,            continue;
"40"              ,goto pl(MOT2);
"41"MOT2L:         m2left,             continue;
"42"              ,goto pl(mstop);
"43"MOT2R:         m2right,            continue;
"44"              ,goto pl(mstop);
"45"MSTOP:         tt2,                call pl(read);      "To stop motors say"
"46"              uw2,                call pl(read);      "the word STOP "
"47"              pa3,                call pl(read);
"48"              ss,                call pl(read);
"49"              ss,                call pl(read);
"50"              tt1,                call pl(read);
"51"              aa,                call pl(read);
"52"              pp,                call pl(read);
"53"              pa3,                call pl(read);
"54"              mm,                call pl(read);
"55"              ax,                call pl(read);
"56"              tt2,                call pl(read);
"57"              ow,                call pl(read);
"58"              er1,                call pl(read);
"59"              pa3,                call pl(read);

```

```

"60"      ss,          call pl(read);
"61"      ey,          call pl(read);
"62"      pa2,         call pl(read);
"63"      dh2,         call pl(read);
"64"      ae,          call pl(read);
"65"      pa2,         call pl(read);
"66"      pa3,         call pl(read);
"67"      wh,          call pl(read);
"68"      or,          call pl(read);
"69"      dd2,         call pl(read);
"70"      pa3,         call pl(read);
"71"      ss,          call pl(read);
"72"      tt1,         call pl(read);
"73"      aa,          call pl(read);
"74"      pp,          call pl(read);
"75"      pa3,         call pl(read);
"76"      , if (STOP) then goto pl(start) else wait;
"77"msgmtr: mm,       call pl(read); " MOTOR ... "
"78"      ow,          call pl(read);
"79"      er1,         call pl(read);
"80"      or,          call pl(read);
"81"      pa5,         call pl(read);
"82"      ,ret;

"83"msggyn: yy2,      call pl(read); "YES... "
"84"      eh,          call pl(read);
"85"      eh,          call pl(read);
"86"      ss,          call pl(read);
"87"      pa4,         call pl(read);
"88"      ow,          call pl(read); "OR... "
"89"      or,          call pl(read);
"90"      pa4,         call pl(read);
"91"      nn2,         call pl(read); "NO... "
"92"      ax,          call pl(read);
"93"      ow,          call pl(read);
"94"      pa5,         call pl(read);
"95"      ,ret;

"96"msgnts: nn2,      call pl(read); "NOT SURE... "
"97"      ax,          call pl(read);
"98"      ow,          call pl(read);
"99"      tt1,         call pl(read);
"99"      pa4,         call pl(read);
"100"     sh,          call pl(read);
"101"     uh,          call pl(read);
"102"     rr2,         call pl(read);
"103"     pa5,         call pl(read);
"100"     ,ret;

.org 511#d

"96"      ,goto pl(start);
END.

```

Sure.” This way, the operator knows he has to again pronounce the word to actuate or stop motor M1. Because the outputs P8 to P11 are used to control both motors by means of a quad latch CD4043, each control output has to be actuated for only one clock cycle.

When the VPC200 recognizes a word to actuate or stop motor M1, it jumps to the same type of questions but now with respect to motor M2. Once motor M2 has been actuated, the SPO256-AL2 says the message “To stop motors say the word stop.” At this point, the FPC will be waiting for the word “Stop” in order to know when to stop both motors M1 and M2.

The program can also be applied to actuate another couple of motors by means of the four unused FPC outputs (P12 to P15). Alternating current motors can also be controlled by using the appropriate optical interface.

## 7.5 Voice-Recognition Synthesis Device SP1000

The single-chip voice recognition and synthesis device SP1000 recognizes speech in real time and synthesizes it. This chip manages both recognition and synthesis with just one reconfigurable filter.

Figure 7.9 shows the block diagram of the SP1000. The SP1000 from Microchip, an n-channel MOS integrated circuit that operates on a 5-V supply, performs LPC feature extraction on the incoming audio signal. It digitizes the spoken word and feeds it to an LPC lattice analysis filter at a data rate of 50 to 100 kbits/s. Switched into the analysis mode, the filter relies on a hard-wired feedback-control loop contained in its arithmetic and logic unit to send it the speech signal's feature data. It uses these data to adapt itself and represent the changing features of the incoming word samples. With its ALU, it calculates LPC filter coefficients, performing data compression in real time with no need to resort to memory. The compressed data in the form of LPC features travels at a slowed rate of 2 to 3 kbits/s from the SP1000 to the microprocessor, which compares the extracted features with templates stored in either random access or mask ROM. The addition of more memory is up to the user.

The SP1000 contains an adaptive filter that has a feedback control scheme and does not need extensive memory. Standard LPC schemes use covariance or autocorrelation, either of which needs at least 3 kilobits of memory to recognize words. On the other hand, the SP1000 requires only 300 bits of on-board memory.

For recognition, the SP1000 calculates the abstract mathematical interval of the test word's features from the template's features. That means detecting end points, providing a parametric representation of the speech signal in the form of LPC features, measuring frame-to-frame distance, and performing dynamic time warping (a technique to identify variables such as the same word spoken differently by the same person). The LPC analyzer is basically a





lattice of filters that approximate a series of resonant cavities, thus simulating the vocal tract.

Synthesis is a simpler operation and proceeds without the feedback-control mechanism. Standard LPC code stored in memory is read by a microprocessor, which sends the appropriate feature data to the SP1000. Routed through a standard synthesis circuit and D/A converter on board, the resulting synthetic speech samples are voiced by a standard speaker.

The SP1000 can be driven by 8-bit standard microprocessors or a microcontroller's bus, with data lines, address lines, chip select line, and read/write line. Eight bits of data can be read from, or written to the chip by the processor, following standard peripheral protocol. The SP1000 board can be used in speaker-dependent or speaker-independent systems with connected or unconnected speech. The designer is not locked into a specific recognition algorithm that may not be suitable for a particular application. Instead, the recognition algorithm is contained in software resident in the host  $\mu P$  or  $\mu C$  and can be easily upgraded to take advantage of advances in recognition techniques. Figure 7.10 shows a block diagram to obtain a voice-recognition synthesis system with the SP1000.

## References

1. Bristow Geoff, ed, *Electronic Speech Recognition*, McGraw-Hill, New York, 1986.
2. William C. Newman, Neural Networks Detect Speech, *Electronic Design*, March 22, 1990, pp. 79–90. Vol 38. No 6.
3. John A. Gallant. Speech Recognition Products, *EDN*, January 19, 1989. pp. 112–122.
4. David Quarmby, *Signal Processor Chips*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632.

- Absolute digital displacement transducer, 138
- Allophone addresses for the SPO256, 32
  - evaluation circuit with EPROM, 34
  - evaluation circuit with MSI, 33
    - EPROM program, 36
    - timing diagram, 36
- Allophone speech synthesis, 3
- AC line frequency cycles meter
  - design of, 234
  - input range, 235
  - software program, 236–237
- AC motor-speed controller
  - artificial voice design, 194
  - operation, 195
  - schematic, 194
  - software program, 196–198
- AC talking voltmeter
  - design of, 230
  - EPROM program, 231–233
  - schematic, 231
- American English language, 3
- Amplifier with programmable gain
  - inverting amplifier, 123
  - schematic, 124
  - software program, 125–126
- Am29CPL100
  - block diagram, 66
  - family members, 65
  - instruction set, 67
  - interface to the SPO256, 68
    - flow chart, 69
    - prom bit pattern, 74
    - software program, 70–71
- Analog-to-digital converters
  - ADC080X series, 85–87
  - basics of, 85
  - block diagram, 87
  - flash converter, 89
  - functional description, 86
  - interface digitalker, 90
    - interface to digitalker, 90
    - flow charts, 91–93
    - software program, 95–97
  - interface to a  $\mu$ C, 98
    - flow chart, 100–101
    - software program, 103–106
  - testing of, 87–88
- Barometer
  - design of, 263
  - pressure sensor, 263
  - schematic, 264–265
  - software program, 265–269
- BCD A/D converter
  - interface to a SPO256, 139
  - microcontroller-based design, 140–141
- BCD code vocalizer with FPC
  - software program, 76–78
  - PROM bit pattern, 78–81
- Burglar alarm
  - CMOS MSI design, 292

- Burglar alarm (*continued*)
  - EPROM programs, 294–295
  - network for N.O. sensors, 296
  - schematic diagram, 294
  - timing equations, 293
  - detects entrance devices, 303
    - schematic diagram, 303
    - software program, 304–306
  - fault-tolerant design, 290
    - schematic diagram, 291
  - FPC-based design, 283
    - flowchart, 285
    - PROM bit contents, 287–288
    - schematic diagram, 284
    - software program, 286–287
  - $\mu$ C-based design, 297
    - flow chart, 302
    - schematic diagram, 297
    - software program, 298–301
  - schematic diagram, 270
  - software program, 272–275
- DC voltmeter
  - FPC-based design, 220
  - input range, 222
  - schematic diagram, 221
  - software program, 225–229
- Delta modulation, 309
  - DPCM, 310
- Digitaler DT1050, 17
  - evaluation circuit, 21
  - functional description, 18–21
  - vocabulary, 19
- Displacement transducers, 135–136
  - digital, 136, 138–139
  - interface to a speech processor, 139
  - strain gage, 137
- Event counter
  - design of a talking, 45
  - operation, 46
  - speech data, 47
- Fault-tolerant
  - burglar alarm, 290
  - respiratory rate meter, 237
- Flash A/D converter, 89
- FPC Am29CPL100 (*see also* Am29CPL100)
  - interface to the SPO256-AL2, 65
- Frequency counter
  - design of a talking, 200
  - flow charts, 210–215
  - input frequencies, 202
  - schematic diagram, 201
  - software program, 204–209
- Hexadecimal keyboard encoder, 148–149
  - simulation program, 154–159
  - software program, 150–152
- Instrumentation amplifier, 127, 131
- Interfacing A/D to digitaler, 90
  - flow charts, 91
  - software program, 95–97
- Interfacing multiple A/Ds, 98
  - flow chart, 100–101
  - software program, 103–106
- Inverting amplifier, 123
- Capacitance meter
  - design of a CMOS MSI, 253
  - schematic 254–255
  - software program, 257–263
- Clock
  - design of a talking, 167
  - schematic diagram, 168–169
  - software program, 171–174
  - table of messages, 167
- Coffee machine controller
  - design of a talking, 179
  - operation, 180
  - schematic diagram, 181
  - software program, 182–189
- Coin detector
  - design of a speaking, 174
  - schematic diagram, 175
  - software program, 176–179
- Converters
  - A/D (*see* analog-to-digital), 85–88
  - BCD A/D interface to SPO256, 139
  - flash, 88
  - microcontroller-based design, 140–141
- Current meter
  - design of a DC, 275
  - schematic diagram, 276
  - software program, 277–281
- Darkroom timer
  - flow chart, 271

- Keyboard encoder, design of a talking, 148–152
- Linear predictive coding, 311
- Liquid level annunciator, 281
  - schematic diagram, 282
- Logic probe
  - applications, 123
  - interface to a speech processor, 119
  - software program, 121–122
- Magnitude comparator, 142
  - design of a 4-bit, 142
    - EPROM-based circuit, 147
    - EPROM program, 148
    - FPC-based design, 145–146
    - interface to a speech processor, 144
    - truth table, 143
- Mean time before failure (MTBF), 245
- Microcontroller handles speech processor, 60–62
  - enhanced program, 64
  - flow chart, 62
  - software program, 62–63
  - timing diagram, 63
- Moisture meter
  - design of a, 251
  - EPROM program, 252
  - schematic, 251
- Multiplexing a speech processor, 81
  - timing diagram, 83
- Neural networks, 311
  - for speech recognition, 311
- Oki semiconductor, speech synthesizers, 26–29
- Optocouplers, in burglar alarms, 296
- Parallel-to-serial speech
  - interface chip, 15
  - interface to SPO256, 16
- Phonemes, 3
- Pressure sensor, 263
- Random number generator
  - design of a talking, 189
  - schematic diagram, 190
  - software program, 191–193
- Redundancy, triplicated modular, 246
- Reference voltage for A/D, 99
- Respiratory rate meter
  - design of a fault-tolerant, 245
  - JEDEC file, 250
  - schematic, 247
  - TMR voter, 248–249
  - triplicated modular redundancy, 246
- Respiratory rate monitor, 237
  - schematic, 238
  - software program, 240–244
- Samsung voice synthesizers, 29–31
  - functional characteristics, 29
  - typical application, 30
- Schmitt trigger, timing circuits, 68, 292
- Semaphore
  - flow direction, 161
  - interface to a SP, 159
  - schematic, 160
  - software program, 162–165
  - traffic intersection, 160
- Speech ROM SPR128A, 11
  - block diagram, 14
  - interface to SPO256, 14
- Speech signals, 2
- Speech synthesis, development board, 16–17
- Speech synthesis processors
  - SPO256/SPO254, 5
    - interface to a FPC, 10
    - interface to a  $\mu$ P, 10
    - interface to a ROM, 10
    - operation, 6
    - stand-alone configuration, 9
    - test modes, 7–8
    - timing diagram, 9
- Speech synthesizer
  - control of two dc motors, 315
  - flowchart, 317
  - schematic, 316
  - software program, 318–320
- Speech synthesizers
  - in airplanes, 2
  - in automobiles, 2
  - in industry, 2
  - instrumentation, 2
- Speech systems, of the past, 1
- SPO256-AL2
  - allophones, 12–13

**SPO256-AL2** (*continued*)

- English phonemes, 11
- interface to a C64, 38
  - software program, 38
- interface to a PC/XT, 37
  - software program, 40–41

Strain gage, 136

Synthesis techniques, 3

Technical vocabulary, SPO256-AL2, 42–44

Texas Instruments TSP5220

- theory of operation, 23–24
- block diagram, 25
- voice synthesis chips, 22

Thermistor linearization, 130

Thermometer

- design of a talking, 127
- flow chart, 133
- instrumentation amplifier, 127, 131
- linearization network, 130
- schematic diagram, 128, 129
- software program, 134–135
- temperature readings, 132
- thermistor equations, 127

Toshiba speech synthesis, LSI devices, 22

Triplicated modular redundancy, 246

TMR voter, 249–249

Vocalizer

- for BCD code, 76–81
- for 4-bit input, 47–49
- for 8-bit input, 49–52
- improved technique, 52–59

Voice recognition

- applications, 308
- delta modulation, 309
- differential PCM, 310
  - LPC, 311
- waveform sampling, 310
- neural networks, 311
- synthesis
  - applications, 322–323
  - block diagram, 322

SP1000, 321

techniques, 308

Voice synthesis ICs, 1

applications, 1

Voice synthesis methods

- frequency domain, 4
- LPC, 4
- parametric synthesis, 3
- synthesis by rule, 3
- time domain, 4
- waveform encoding, 3

Voice-recognizer controller, 314

schematic diagram, 314

outputs, 315

Voltage comparator

- 10-step, 113
- design equations, 115
- going further, 119
- interface to a speech, 113
- PAL truth table, 114
- software program, 116–118

Voltmeter with DT1050, 217

EPROM program, 220

schematic diagram, 219

Voltmeter with SPO56-AL2, 216

EPROM program, 218

operation, 216–217

Waveform sampling, using DPCM, 310

Window comparator

- circuit, 110
- design of a speaking, 107
- PROM bit pattern, 112
- software program, 108–109

Word recognizer

- application circuit, 315
- flow chart, 317
- schematic, 316
- software program, 318–320

Word recognizer VCP200, 311

application circuit, 313

pin configuration, 312

theory of operation, 311–313