TEXAS INSTRUMENTS

MSP50x3x

Mixed-Signal Processor

User's Guide

2000 *Mixed-Signal Products*

SPSU006C

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

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

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Preface

Read This First

About This Manual

This manual describes the MSP50x3x family of speech synthesizing devices. When necessary, the differences between the family members are shown in separate and consecutive sections. The object of this user's guide is to provide the information needed to implement a speech synthesizer design using a MSP50x3x devices.

How to Use This Manual

This document contains the following chapters:

Chapter 1 **Introduction to the MSP50x3x Family**
This chapter describes the MSP50x3x family features, D/A options, pin assignments and descriptions, and gives a brief introduction to linear predictive coding.

Chapter 2 **MSP50x3x Family Architecture**
This chapter describes the architecture of the MSP50x3x family with a separate sections for speech synthesis, interrupts, power control, initialization, and clocks. For the MSP50x37, there are separate sections for the analog-to-digital converter and the power amplifier.

Chapter 3 **MSP50x3x Assembler**
This chapter contains a detailed description of the MSP50x3x assembler.

Chapter 4 **MSP50x3x Instruction Set**
This chapter provides the instruction set for the MSP50x3x.

Chapter 5 **MSP50x3x Applications**
This chapter describes various hints and useful advice for designing applications for the MSP50x3x.

Notational Conventions

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's.

  Here is a sample program listing:

```
0349 0059   6B  SPEAK2      LUAA              –Get word
0350 005A   60              ANEC   StopWord   –End phase?
     005B   FF
```

- In syntax descriptions the following notational conventions are used in this guide:
  - A reserved keyword (an instruction, command or directive) is shown in **bold** capital letters and should be entered as shown.
  - An optional field is indicated by brackets and italics and describes the type of information that should be entered:
    [*label*]
  - User-supplied contents are indicated by angle brackets and italics and describe the type of information that should be entered:

    <*num*>
  - A required blank is indicated by a caret (^).

  The following syntax example demonstrates the notational conventions used in this guide.

  [<*label*>]^**ABAAC**^...[<*comment*>]

- A lower case **h** at the end of a numeric value indicates that the value is hexadecimal (e.g., 01FAh, 032Bh, and 0FFh).
- All addresses in this manual are in hexadecimal format unless otherwise noted. All other are numbers are in decimal format unless otherwise noted.
- Abbreviations:
  - **'04:** MSP50C04
  - **'06:** MSP50C06
  - **'10:** MSP50C10
  - **'11:** MSP50C11
  - **'12:** MSP50C12
  - **'13:** MSP50C13
  - **'14:** MSP50C14
  - **'19:** MSP50C19
  - **LSB, MSB:** Least significant and most significant *bits*
  - **LSbyte, MSbyte:** Least significant and most significant *bytes*

- **Port A** refers to pins PA0 — PA7 operating together.
- **Port B** refers to pins PB0 and PB1 operating together.
- Individual bits of a register are indicated with the register abbreviation followed by a decimal point and the bit number (e.g., bit 5 of the A register is A.5 or bit 2 of the mode register is MR.2).
- *X is the contents of the location pointed to by the address stored in X register.
- A' indicates the old contents of the A register

Information About Cautions

This book may contain cautions.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

The information in a caution is provided for your protection. Please read each caution carefully.

If You Need Assistance . . .

If you want to . . .	Contact Texas Instruments at . . .	
Visit TI online	World Wide Web:	http://www.ti.com
Receive general information or assistance	World Wide Web:	http://www.ti.com/sc/docs/pic/home.htm
	North America, South America:	(972) 644–5580
	Europe, Middle East, Africa	
	Dutch:	33–1–3070–1166
	English:	33–1–3070–1165
	French:	33–1–3070–1164
	Italian:	33–1–3070–1167
	German:	33–1–3070–1168
	Japan (Japanese or English)	
	Domestic toll-free:	0120–81–0026
	International:	81–3–3457–0972 or 81–3–3457–0976
	Korea (Korean or English):	82–2–551–2804
	Taiwan (Chinese or English):	886–2–3771450
Request more information about Texas Instruments Speech Synthesizer products	World Wide Web:	http://www.ti.com/sc/speech
Order Texas Instruments documentation (see Note 1)	Literature Response Center:	(800) 477–8924
Make suggestions about or report errors in documentation (see Note 2)	Email:	comments@books.sc.ti.com
	Mail:	Texas Instruments Incorporated Technical Publications Manager, MS 8711 P.O. Box 655303 Dallas, Texas 75265–5303

Notes: 1) The literature number for the book is required; see the lower-right corner on the back cover.

2) Please mention the full title of the book, the literature number from the lower-right corner of the back cover, and the publication date from the spine or front cover.

Trademarks

Contents

Figures

Tables

Examples

Notes and Cautions

Chapter 1

Introduction to The MSP50x3x Family

The MSP50x3x family uses a revolutionary architecture to combine an 8-bit microprocessor, two speech synthesizers, ROM, RAM, and I/O in a low-cost single-chip system. The architecture uses the same arithmetic logic unit (ALU) for the two synthesizers and the microprocessor, thus reducing chip area and cost and enabling the microprocessor to do a multiply operation in 0.8 μs. The MSP50x3x family features two semi-independent channels of linear predictive coding (LPC), which synthesize high-quality speech at a low data rate. Pulse-code modulation (PCM) can produce music or sound effects. LPC and PCM can be added together to produce a composite result.

Topic **Page**

1.1 MSP50x3x Device Family

The MSP50x3x family of speech synthesizers consists of six family members differentiated by the size and type of hard storage incorporated on the device. Table 1–1 gives a list of members with the amount of ROM or PROM on each device.

Table 1–1. MSP50x3x Device Family

Device	Amount of ROM/PROM	Features
MSP50C32	16K bytes mask ROM	9/10 I/O lines
MSP50C33	32K bytes mask ROM	9/10 I/O lines
MSP50C34	64K bytes mask ROM	9/10 I/O lines in package, 24 I/O lines in die form
MSP50P34	64K bytes PROM	9/10 I/O lines
MSP50C37	16K bytes mask ROM	18 I/O lines, A/D converter/analog amplifier
MSP50P37	16K bytes PROM	18 I/O lines, A/D converter/analog amplifier

1.2 Applications

The MSP50x3x is highly flexible and programmable, making it suitable for a wide variety of applications. Its low system cost opens up new applications for solid-state speech. These include:

- Talking clocks
- Toys
- Games
- Telephone answering machines
- Home monitors
- Navigation aids
- Laboratory instruments
- Personal computers
- Inspection controls
- Inventory controls
- Machine controls
- Warehouse systems
- Warning systems
- Appliances
- Voice mailboxes
- Equipment for the handicapped
- Learning aids
- Computer-aided instruction
- Magazine and direct-mail advertisements
- Point-of-sale displays
- Talking books

The MSP50x37 is basically designed for warning systems such as the gas warning system (GWS), smoke detector systems, and other similar applications. The device is flexible enough to be used in other applications that require A/D converters such as motor control for toys, etc.

1.3 Description

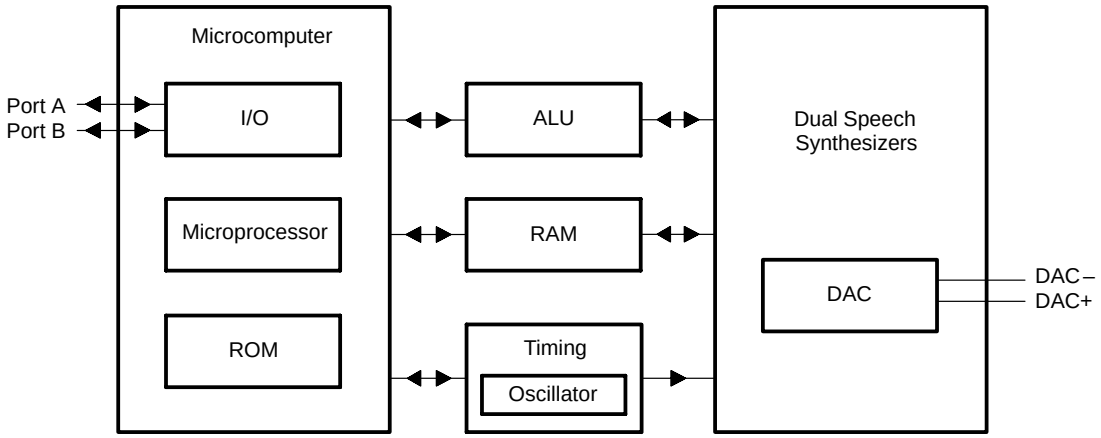
The MSP50x3x can be divided into several functional blocks (see Figure 1–1 and Figure 1–2). The ALU and RAM are shared by the two speech synthesizers and the microcomputer.

The MSP50x3x implements an LPC-12 speech-synthesis algorithm using two 12-pole lattice filters. The internal microprocessor fetches speech data from the internal ROM, decodes the speech data, and sends the decoded data to the synthesizer. The microprocessor also interpolates (smoothes) the speech data between fetches. The microprocessor can calculate a PCM waveform, which can be added to the output of one of the two lattice filters to create composite PCM + LPC waveforms.

The general purpose microprocessor in the MSP50x3x, which is capable of a variety of logical, arithmetic, and control functions, can be used for the non-synthesis tasks of the application as well.

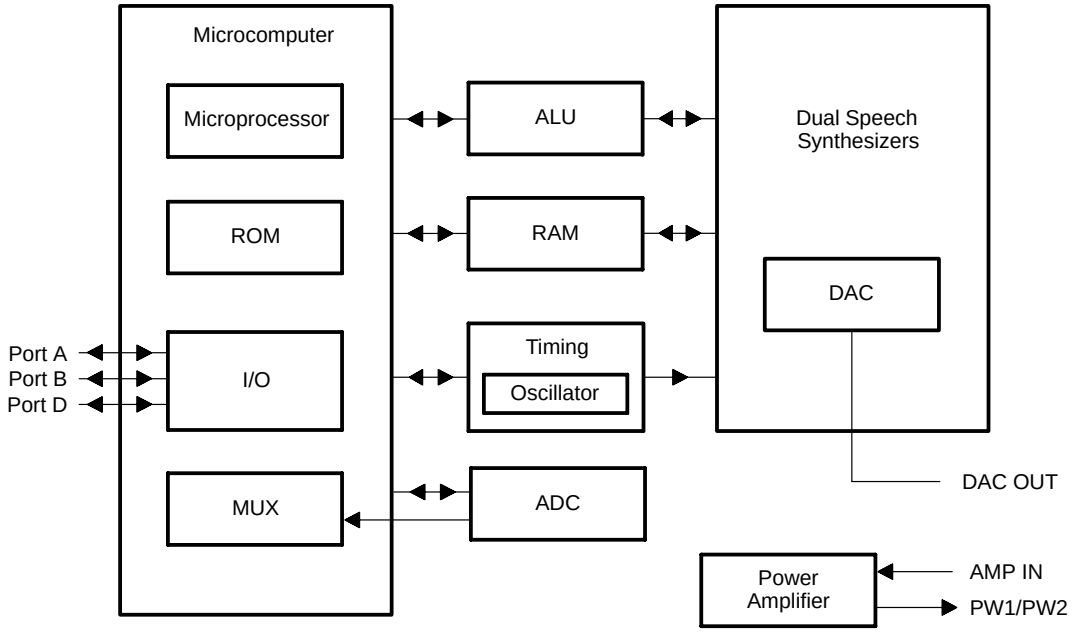
The MSP50x3x family of parts incorporates a built-in oscillator and has the capability of directly driving a 32-Ω speaker.

Figure 1–1. MSP50x3x Functional Block Diagram



The MSP50x37 contains features not contained in the other members of the MSP50x3x family such as an integrated 8-bit successive approximation (SAR) analog-to-digital converter (ADC) with 8 to 1 analog multiplexer ports, a power amplifier that has the capability to directly drive an 8-Ω speaker,a long interval timer and four I/O ports with sufficient drive capability to directly drive an LED.

Figure 1–2. MSP50x37 Functional Block Diagram



1.4 Features

The key features of the entire MSP50x3x family are in the following list.

- ❑ Dual programmable LPC-12 speech synthesizers
- ❑ Simultaneous LPC and PCM
- ❑ 8-bit microprocessor with 61 instructions
- ❑ Thirty-two 12-bit words and 224 bytes of RAM
- ❑ 3.3V to 6.5V CMOS technology for low power dissipation
- ❑ Direct speaker drive capability
- ❑ Mask-selectable internal or external Clock
- ❑ Internal clock generator that requires no external components
- ❑ Two software-selectable clock speeds
- ❑ 10-kHz or 8-kHz speech sample rate
- ❑ Seven levels of stack
- ❑ Internal timer
- ❑ Externally-controlled interrupt
- ❑ Single-cycle multiply instruction
- ❑ Executes 1,200,000 instructions per second
- ❑ Built-in slave mode to act as a microprocessor peripheral
- ❑ Software-configurable wakeup function from any A Port I/O line

1.4.1 MSP50x32/33/34 Additional Features

MSP50x32/33/34 features not present in other family members include:

- ❑ Ten software configurable I/O terminals (nine I/O terminals with external clock selected)
- ❑ Two digital-to-analog (D/A) configurations—mask selectable
- ❑ Several ROM/PROM configurations
  - ■ 16K bytes for MSP50C32
  - ■ 32K bytes for MSP50C33
  - ■ 64K bytes for MSP50C34 and MSP50P34

1.4.2 MSP50x37 Additional Features

MSP50x37 features not present in other family members include:

- ❑ Incorporated 8-bit analog-to-digital converter multiplexed to one of 8 I/O ports (D Ports)
- ❑ 18 software configurable I/O terminals
- ❑ Power amplifier for direct speaker drive (8 Ω/500 mW)
- ❑ 20-mA sink current for LED direct drive (A4 – A7)
- ❑ Additional long interval timer/counter
- ❑ Single-pin double-ended D/A output
- ❑ 16K-bytes of ROM/PROM

1.5 D/A Options

The MSP50x3x family offers two D/A (digital-to-analog) output options to match different applications. Option 1 can directly drive a 32-Ω speaker.

Note:

The MSP50x37 incorporates an analog power amplifier that can drive an 8-Ω speaker.

Table 1–2 gives a list of the devices and the options available for each.

Table 1–2. MSP50x3x D/A Options

Device	D/A Options Available	Speaker Drive
MSP50C32	Option 1	32-Ω direct drive
	Option 2	Drives an operational amplifier
MSP50C33	Option 1	32-Ω direct drive
	Option 2	Drives an operational amplifier
MSP50C34	Option 1	32-Ω direct drive
	Option 2	Drives an operational amplifier
MSP50P34	Option 1	32-Ω direct drive
	Option 2	Drives an operational amplifier
MSP50C37	Option 2	8-Ω direct drive
MSP50P37	Option 2	8-Ω direct drive

1.5.1 Two-Pin Push Pull (Option 1) — Accurate to 1 part in 1024

Option 1 is a two-pin push pull. If direct speaker drive is not desired, it works well with a very efficient and inexpensive four-transistor amplifier. When the DAC is idle or the output value is 0, both pins are low. When the output value is positive, DAC+ pulses high with a pulse density proportional to the output value, while DAC– stays low. When the output value is negative, DAC– goes high with a pulse density proportional to the output value, while DAC+ stays low. This option can respond to values ranging from –512 to +512.

Figure 1–3 shows examples of D/A output waveforms with different output values. Each sample period of the DAC is divided into 512 segments. For a positive output value, x = 0 to 512, DAC+ goes high for x segments while DAC– stays low. When the DAC is idle or the output value is 0, both DAC+ and DAC– are low. For a negative value x = 0 to –512, DAC– goes high for |x| segments while DAC+ stays low.

Figure 1–3. D/A Output Waveforms for Two-Pin Push Pull (Option 1)

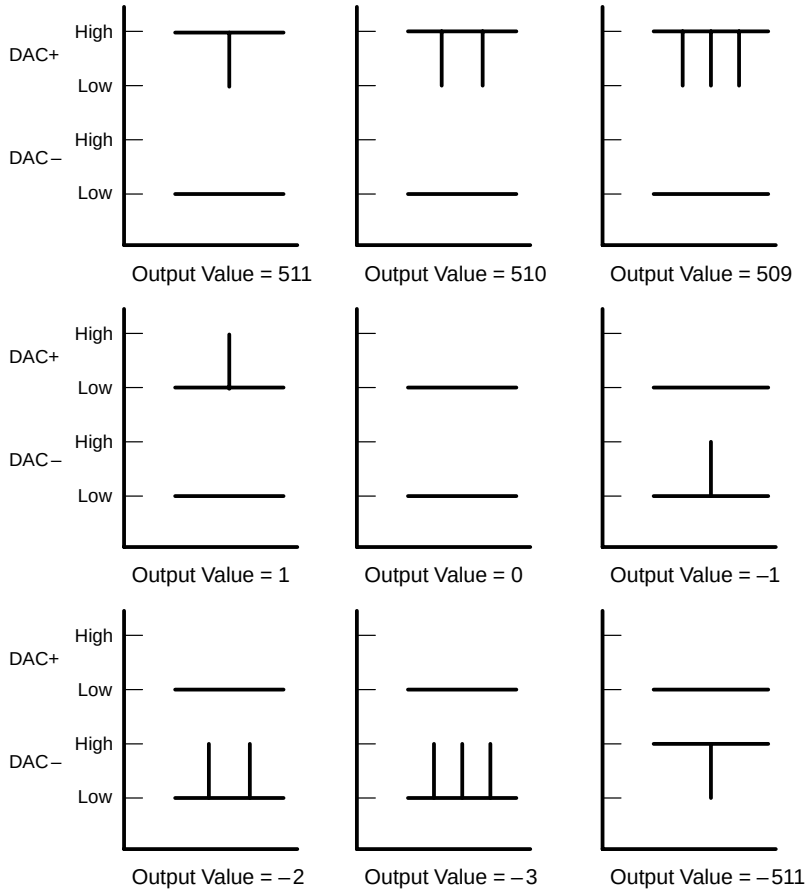


Figure 1–4, Figure 1–5, and Figure 1–6 show examples of circuits that can be used with this option.

Figure 1–4. Four-Transistor Amplifier Circuit

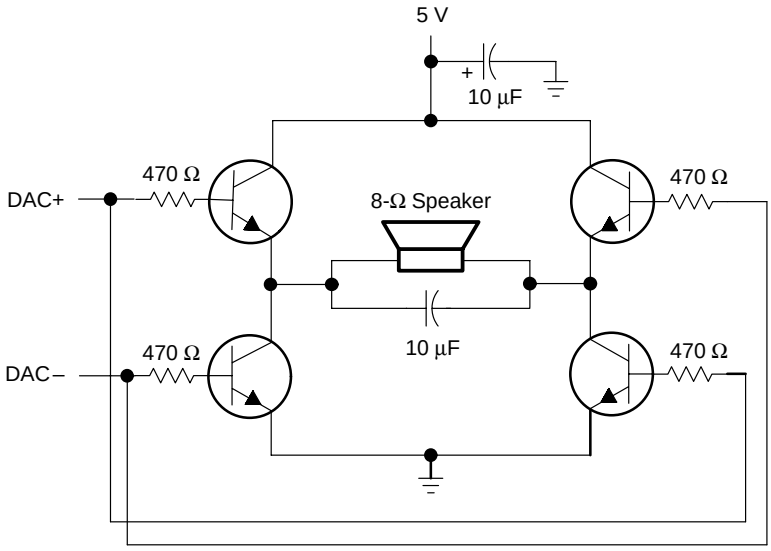


Figure 1–5. Operational Amplifier Interface Circuit

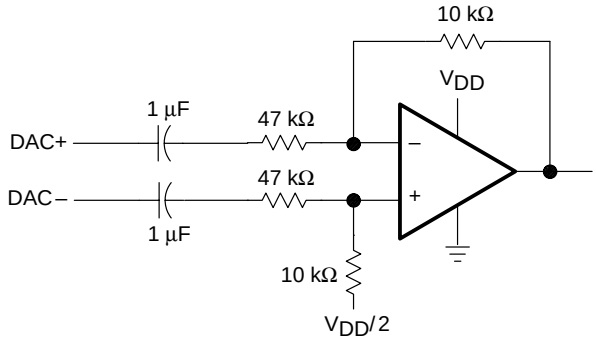
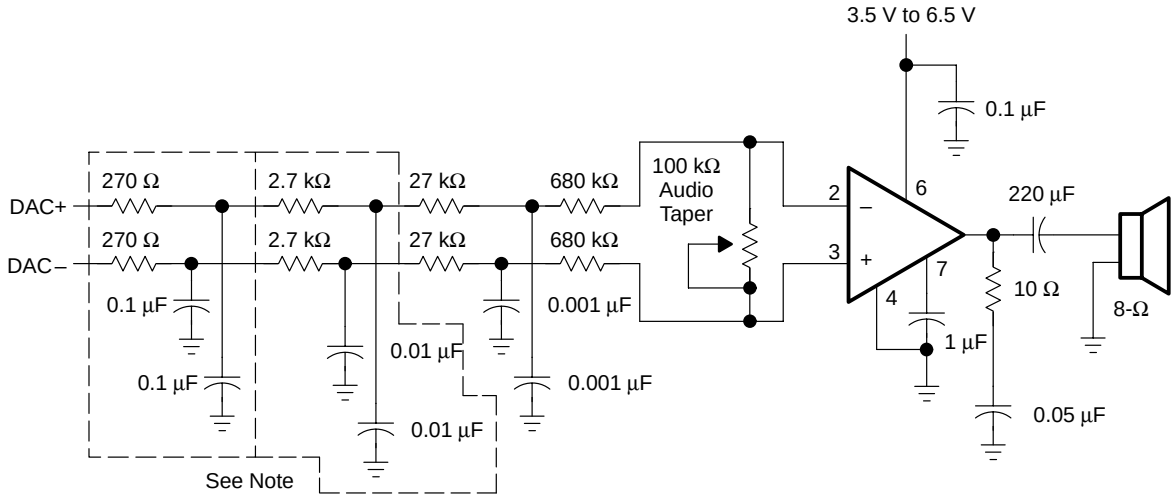


Figure 1–6. Power Amplifier Interface Circuit



Note: The components in each dotted box are optional. They provide stages of low-pass filters. The remainder of the circuit is the minimum configuration recommended. If the application does not need a volume control, the 100-kΩ potentiometer can be replaced with a fixed-value resistor.

1.5.2 Single-Pin Double-Ended (Option 2) — Accurate to 1 part in 1024

Option 2 is for use with operational and power amplifiers. When the output value is zero, the D/A output is biased at approximately 1/2 V_{DD}. When the output value is positive, the D/A output pulses to approximately 3/4 V_{DD} with a pulse density proportional to the output value. When the output value is negative, the D/A output pulses to 1/4 V_{DD} with a pulse density proportional to the output value.

Figure 1–7 shows examples of D/A output waveforms with different output values. Each pulse of the DAC is divided into 512 segments per sample period. For a positive output value x = 0 to 512, DAC+ goes high to 3/4V_{DD} for x segments. When the DAC is idle or the output value is 0, DAC+ stays at 1/2V_{DD}. For a negative value x = 0 to –512, DAC+ goes low to 1/4V_{DD} for |x| segments.

Figure 1–7. Figure 1–7. D/A Output Waveforms for Single-Pin Double-Ended (Option 2)

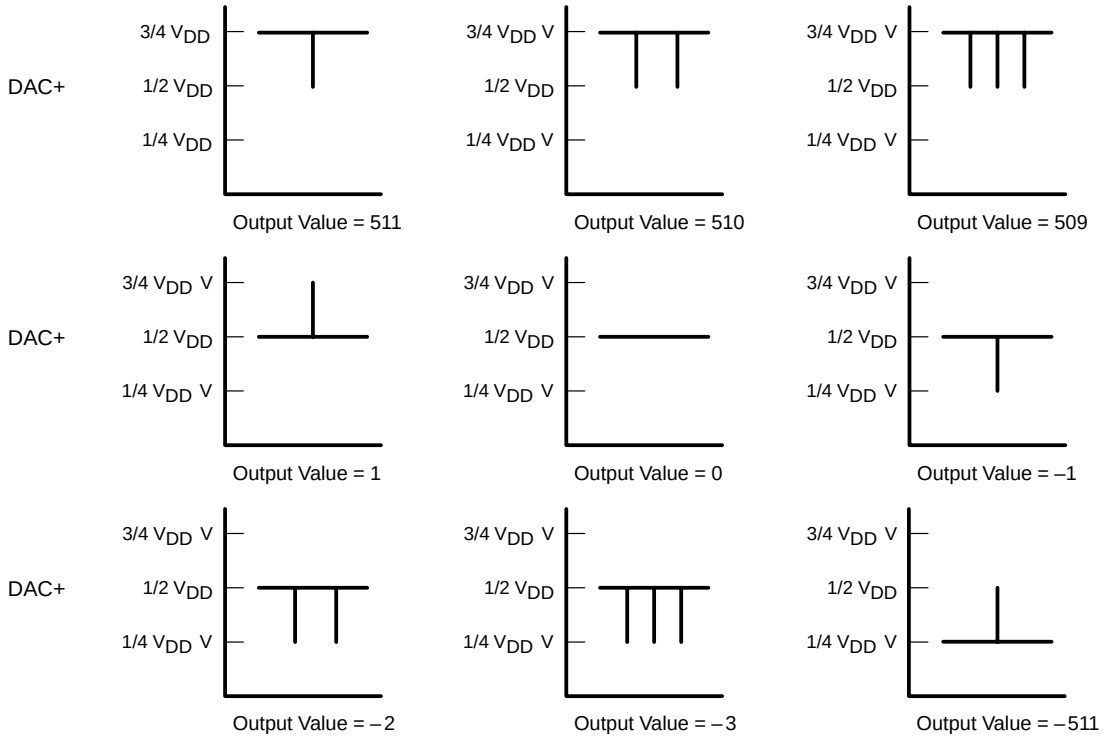
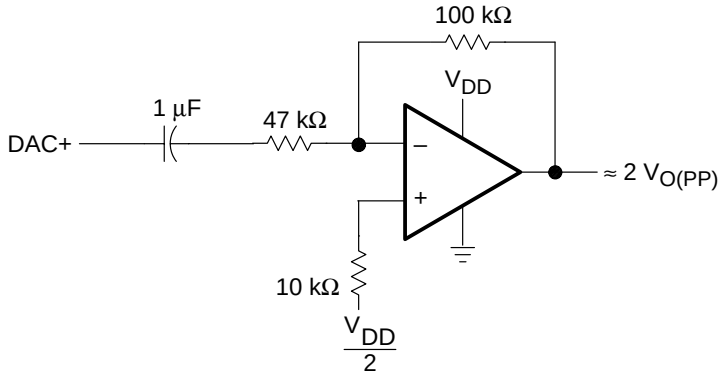


Figure 1–8 gives an example circuit that demonstrates how to interface with this output DAC option.

Figure 1–8. Operational Amplifier Interface Circuit



The MSP50x37 incorporates a power amplifier that consists of one AMP IN terminal and either a two-pin differential output or one-pin analog output. This amplifier has the capability to drive an 8-Ω speaker with 0.5-W typical power. The one-pin analog output typically has one-half the power of the two-pin differential output. The amplifier is set to have ten times (10×) voltage gain. Therefore, the input voltage range should be limited to within 1/2 V_{DD} ±0.28V to prevent the output power from exceeding 0.5 W.

Figure 1–9 shows the two-pin push pull power amplifier output option with an external filter circuit. Figure 1–10 shows the one-pin analog power amplifier output option with an external filter circuit. It is recommended that an analog signal that is filtered at 3.5 kHz externally be applied. When applying a signal not centered to 1/2 V_{DD}, a coupling condenser is required between the filter output and AMP IN.

Figure 1–9. Two-Pin Push Pull Power Amplifier Output Circuit with External LPF

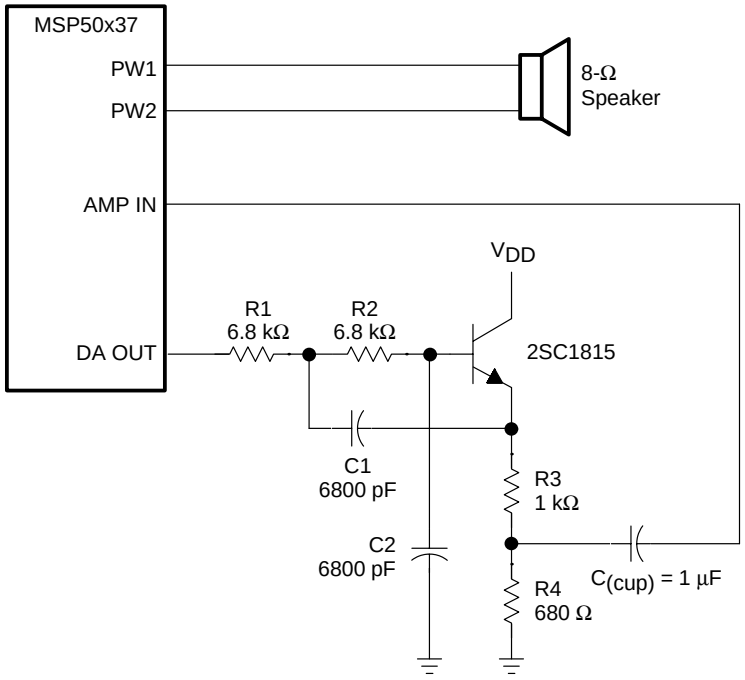
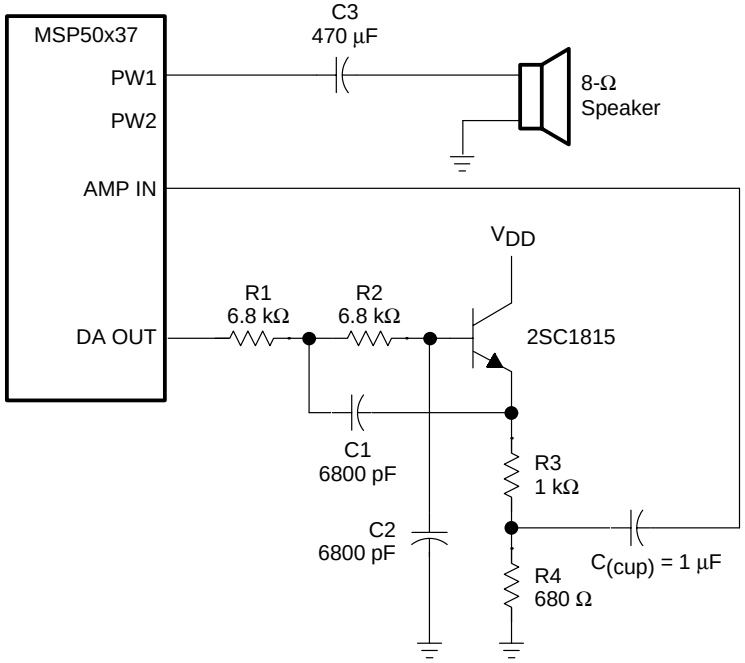


Figure 1–10. One-Pin Analog Power Amplifier Output Circuit with External LPF



1.6 Terminal Assignments and Signal Descriptions

The following sections give terminal assignment and signal description information for the MSP50x3x family. The MSP50x32/33/34 are all available in a 16-pin DW or N package and in die form. The MSP50P34 is also available in a 40-pin NW package, which is used for engineering evaluation. The MSP50x37 is available in a 28-pin DW package.

1.6.1 MSP50x32/33/34 Terminal Assignments and Signal Descriptions (16-Terminal N Package)

The MSP50x32/33/34 are all available in a 16-pin DW or N package and in die form. Figure 1–11 shows the terminal assignments for the MSP50x32/33/34 for the 16-pin package. Table 1–3 provides terminal function descriptions. Table 1–4 gives the pad locations on the MSP50C32/C33 in die form.

Note: MSP50C34I/O Lines

In die form, more I/O lines are available on the MSP50C34 than on the packaged version.

Figure 1–12 illustrates the recommended power-up initialization circuit. Refer to Chapter 6 for more information on I/O configuration. Refer to section 6.3, *Mechanical Information*, for detailed package dimensions.

Figure 1–11. MSP50x32/33/34 16-Pin Package Terminal Assignments

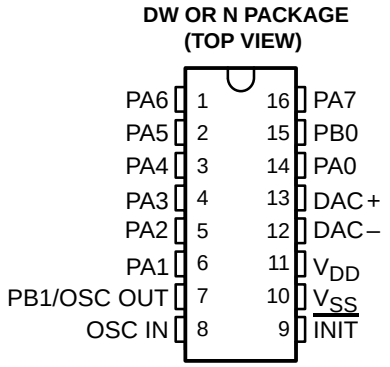


Table 1–3. MSP50C32/33/34 16-Pin Package Terminal Functions

Terminal Name	Terminal Number	I/O	Signal Description
DAC+	13	O	D/A output. When D/A Option 1 is selected, DAC+ pulses high for positive output values. It remains low when negative values are output.
DAC–	12†	O	D/A output. When D/A Options 1 is selected, DAC– pulses high for negative output values. It remains low when positive values are output. When D/A Option 2 is selected, this terminal is driven low.
$\overline{\text{INIT}}$	9	I	Initialize input. When $\overline{\text{INIT}}$ goes low, the clock stops, the MSP50x3x goes into low-power mode, the program counter is set to 0, and the contents of the RAM are retained. An $\overline{\text{INIT}}$ pulse of 1 μs is sufficient to reset the processor.
OSC IN	8	I	Clock input. When not in use, OSC IN should be tied to V_{SS}.
OSC OUT	7	–	Clock return. When the internal clock option is selected, This terminal is the B1 I/O terminal.
PA0–PA7	14,6–1,16	I/O	8-bit bidirectional I/O port
PB0–PB1	15, 7	I/O	2-bit bidirectional I/O port. When the external clock option is selected, B1 is not available, since terminal 7 is used for the OSC OUT function.
V_{DD}	11	–	5-V supply voltage
V_{SS}	10	–	Ground

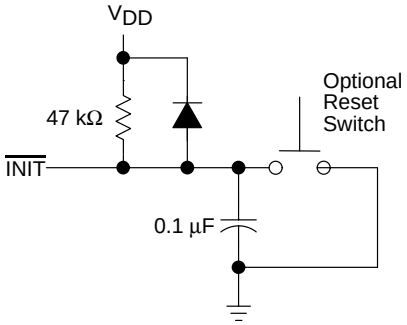
† If D/A Option 2 is selected, the DAC– terminal is asserted low.

Table 1–4. Pad Location on the MSP50C32/C33 Die Form

	Lower Left†		Upper Right†		
Pad Name	**X (microns)**	**Y (microns)**	**X (microns)**	**Y (microns)**	**Description**
PA6	902.25	4321.00	1053.75	4772.50	A Port I/O, terminal 6
PA5	571.50	4321.75	721.50	4471.75	A Port I/O, terminal 5
PA4	185.00	4321.75	335.00	4471.75	A Port I/O, terminal 4
PA3	28.75	1719.75	178.75	1869.75	A Port I/O, terminal 3
PA2	28.75	1333.25	178.75	1483.25	A Port I/O, terminal 2
PA1	261.00	31.00	411.00	181.00	A Port I/O, terminal 1
PB1/ OSC OUT	647.50	31.00	797.50	181.00	B Port I/O, terminal 1
OSC IN	979.00	31.00	1129.00	181.00	Clock input (see Table 1–3)
$\overline{\text{INIT}}$	3153.50	36.50	3303.50	186.50	Initialize input (see Table 1–3)
V_{SS}	3450.50	36.25	3600.50	186.25	Ground
V_{DD}	3783.75	36.25	3933.75	186.25	5-V supply voltage
DAC–	4007.75	1749.75	4157.75	1899.75	D/A output (see Table 1–3)
DAC+	4007.75	2759.25	4157.75	2909.25	D/A output (see Table 1–3)
PA0	4031.00	4043.25	4181.00	4193.25	A Port I/O, terminal 0
PB0	3721.00	4321.75	3871.00	4471.75	B Port I/O, terminal 0
PA7	3334.50	4321.75	3484.50	4471.75	A Port I/O, terminal 7

† Coordinates are in respect to a specific corner of the device.

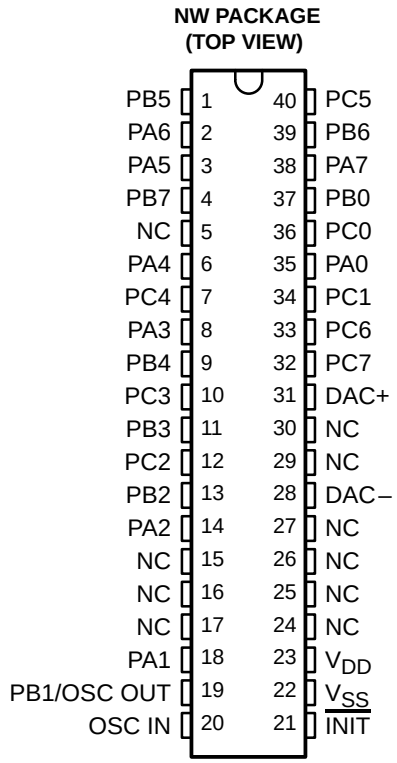
Figure 1–12. Power-Up Initialization Circuit



1.6.2 MSP50C34/P34 Terminal Assignments and Signal Descriptions

The MSP50C34 and MSP50P34 are available in die form allowing additional I/O capability. The MSP50P34 is also available in a 40-pin NW package for engineering evaluation purposes. Figure 1–13 shows the terminal assignments for the MSP50P34 in the 40-pin package. Table 1–5 provides terminal function descriptions. Table 1–6 gives the pad location on the MSP50C34 in die form. Figure 1–12 illustrates the recommended power up initialization circuit. Refer to Chapter 6 for more information on I/O configuration. Refer to section 6.3, *Mechanical Information*, for detailed package dimensions.

Figure 1–13. MSP50P34 40-Pin Package Terminal Assignments



NC – No internal connection

Table 1–5. MSP50P34 40-Pin Package Terminal Functions

Terminal Name	Terminal Number	I/O	Signal Description
DAC+	31	O	D/A output. When D/A Option 1 is selected, DAC+ pulses high for positive output values. It remains low when negative values are output.
DAC–	28†	O	D/A output. When D/A Options 1 is selected, DAC– pulses high for negative output values. It remains low when positive values are output. When D/A Option 2 is selected, this terminal is driven low.
$\overline{\text{INIT}}$	21	I	Initialize input. When $\overline{\text{INIT}}$ goes low, the clock stops, the MSP50x3x goes into low-power mode, the program counter is set to 0, and the contents of the RAM are retained. An $\overline{\text{INIT}}$ pulse of 1 µs is sufficient to reset the processor.
OSC IN	20	I	Clock input. When not in use, OSC IN should be tied to V_{SS}.
OSC OUT	19	–	Clock return. When the internal clock option is selected, this terminal is the B1 I/O terminal.
PA0–PA7	35, 18, 14 8, 6, 3, 2, 38	I/O	8-bit bidirectional I/O port
PB0–PB7	37, 19, 13 11, 9, 1, 39, 4	I/O	8-bit bidirectional I/O port. When the external clock option is selected, B1 is not available, since terminal 19 is used for the OSC OUT function.
PC0–PC7	36, 34, 12, 10 7, 40, 33, 32	I/O	8-bit bidirectional I/O port.
V_{DD}	23	–	5-V supply voltage
V_{SS}	22	–	Ground

† If D/A Option 2 is selected, the DAC– terminal is asserted low.

Table 1–6. Pad Location on the MSP50C34/P34 Die Form

	Lower Left†		Upper Right†		
Pad Name	**X (microns)**	**Y (microns)**	**X (microns)**	**Y (microns)**	**Description**
PA6	982.50	6375.25	1132.50	6525.25	A Port I/O, terminal 6
PA5	538.00	6376.25	688.00	6526.25	A Port I/O, terminal 5
PB7	245.50	6376.25	395.50	6526.25	B Port I/O, terminal 7
PA4	25.50	5903.25	175.50	6053.25	A Port I/O, terminal 4
PC4	25.50	5488.75	175.50	5638.25	C Port I/O, terminal 4
PA3	25.50	4963.25	175.50	5113.25	A Port I/O, terminal 3
PB4	25.50	4232.75	175.50	4832.75	B Port I/O, terminal 4
PC3	25.50	3799.25	175.50	3949.25	C Port I/O, terminal 3
PB3	25.50	3124.25	175.50	3274.25	B Port I/O, terminal 3
PC2	25.50	2709.25	175.50	2859.25	C Port I/O, terminal 2
PB2	25.50	2034.25	175.50	2184.25	B Port I/O, terminal 2
PA2	25.50	1283.75	175.50	1433.75	A Port I/O, terminal 2
PA1	257.00	22.50	407.00	172.50	A Port I/O, terminal 1
PB1/ OSC OUT	614.50	22.50	764.50	172.50	B Port I/O, terminal 1
OSC IN	926.25	22.50	1076.25	172.50	Clock input (see Table 1–5)
$\overline{\text{INIT}}$	3200.50	30.00	3350.50	180.00	Initialize input (see Table 1–5)
V_{SS}	3497.50	29.75	3647.50	179.75	Ground
V_{DD}	3830.75	29.75	3980.75	179.75	5-V supply voltage
DAC–	4053.25	1744.50	4203.25	1894.50	D/A output (see Table 1–5)
DAC+	4053.25	4018.25	4203.25	4168.25	D/A output (see Table 1–5)
PC7	4079.50	4461.50	4229.50	4611.50	C Port I/O, terminal 7
PC6	4079.50	5136.50	4229.50	5286.50	C Port I/O, terminal 6
PC1	4079.50	5551.50	4229.50	5701.50	C Port I/O, terminal 1
PA0	4079.50	6198.50	4229.50	6348.50	A Port I/O, terminal 0
PC0	3751.25	6376.25	3901.25	6526.25	C Port I/O, terminal 0
PB0	3440.25	6376.25	3590.25	6526.25	B Port I/O, terminal 0
PA7	2941.50	6376.25	3091.50	6526.25	A Port I/O, terminal 7
PB6	2565.25	6376.25	2715.25	6526.25	B Port I/O, terminal 6
PC5	1944.75	6376.25	2094.75	6526.25	C Port I/O, terminal 5
PB5	1502.00	6376.25	1652.00	6526.25	B Port I/O, terminal 5

† Coordinates are in respect to a specific corner of the device.

1.6.3 MSP50x37 28-Pin Package Terminal Assignments and Signal Descriptions

Figure 1–14 shows the terminal assignments for MSP50C37 28-pin package. Table 1–7 provides terminal functional descriptions. The device is packaged in 28-pin DW or N Package. Refer to Section 6.3, *Mechanical Information*, for detailed package dimensions.

Figure 1–14. MSP50x37 28-Pin Package Terminal Assignments

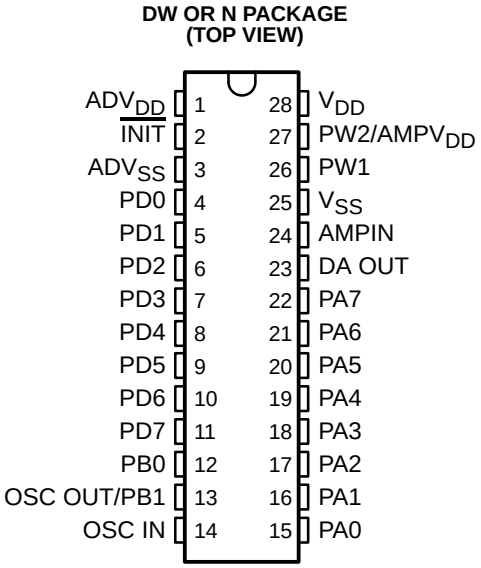


Table 1–7. MSP50x37 28-Pin Package Terminal Functions

Terminal Name	Terminal Number	I/O	Signal Description
ADV_{DD}	1		Power supply for ADC. The reference voltage + is connected to ADV_{DD} internally.
ADV_{SS}	3		Ground level for ADC. The reference voltage – is connected to ADV_{SS} internally.
AMPIN	24	I	Power amplifier input.
$AMPV_{DD}$	27	I	Amplifier input power. When the one-pin analog power amplifier output option is selected, $AMPV_{DD}$ acts as the amplifier power supply. See the signal description for PW2 when the two-pin push pull power amplifier output option is selected.
DA OUT	23	O	Digital-to-analog output. Single-pin double-ended mode.
$\overline{INIT}$	2	I	Initialize input. When $\overline{INIT}$ goes low, the clock stops, the MSP50x3x goes into low-power mode, the program counter is set to 0, and the contents of the RAM are retained. An $\overline{INIT}$ pulse of 1 μs is sufficient to reset the processor.
OSC IN	14	I	Clock input. When not in use, OSC should be tied to V_{SS}.
OSC OUT	13	O	Clock return. When the internal clock option is selected, this terminal is the B1 I/O terminal.
PA0–PA7	15 – 22	I/O	8-bit bidirectional I/O port. PA4 – PA7 have large sink current (20ma)
PB0–PB1	12, 13	I/O	2-bit bidirectional I/O port. When the external clock option is selected, B1 is not available, since this terminal is used for the OSC OUT function.
PD0–PD7	4 – 11	I/O	8-bit bidirectional I/O port or analog signal input terminal for ADC. The terminal configuration can be set by software.
PW1	26	O	Power amplifier output.
PW2	27	O	Power amplifier output. When the two-pin push pull power amplifier output option is selected, PW2 is one of two power amplifier outputs. See the $AMPV_{DD}$ signal description when the one-pin analog power amplifier output option is selected.
V_{DD}	28	–	5-V supply voltage
V_{SS}	25	–	Ground

1.7 Introduction to Linear Predictive Coding (LPC)

The LPC-12 system uses a mathematical model of the human vocal tract to enable efficient digital storage and recreation of realistic speech. To understand LPC, it is essential to understand how the vocal tract works. This introduction, therefore, begins with a short description of the vocal tract, after which the LPC model and data compression techniques are addressed.

1.7.1 The Vocal Tract

Speech is the result of the interaction among three elements in the vocal tract; air from the lungs, a restriction that converts the airflow to sound, and the vocal cavities that are positioned to resonate properly.

Air from the lungs is expelled through the vocal tract when the muscles of the chest and diaphragm are compressed. Pressure is used as a volume control, with higher pressure for louder speech.

As air flows through the vocal tract, it makes little sound if there is no restriction. The vocal cords are one type of restriction. They can be tightened across the vocal tract to stop the flow of air. Pressure builds up behind them and forces them open. This happens over and over, generating a series of pulses. The tension on the vocal cords can be varied to change the frequency of the pulses. Many speech sounds, such as the “A” sound, are produced by this type of restriction, which is called “voiced” speech.

A different type of restriction in the mouth causes a hissing sound called white noise. The “S” sound is a good example. White noise occurs when the tongue and some part of the mouth are in close contact or when the lips are pursed. This restriction causes high flow velocities that cause turbulence producing white noise, which is called “unvoiced” speech.

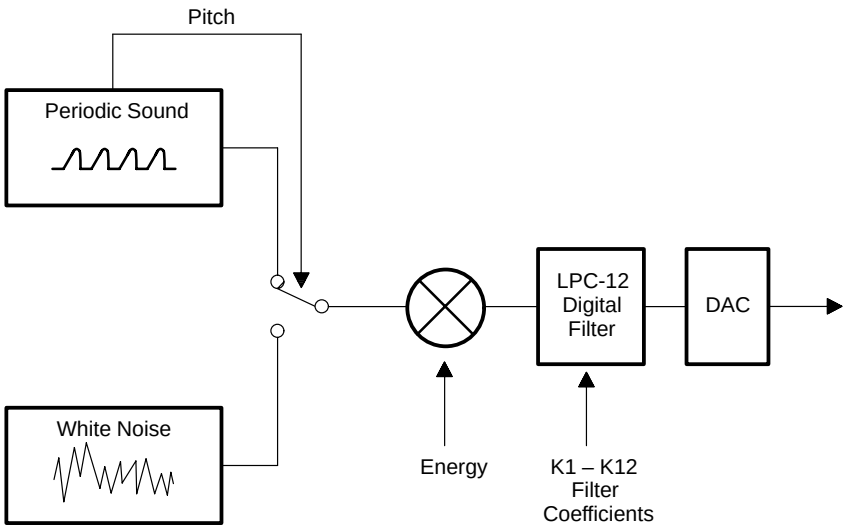
The pulses from the vocal cords and the noise from the turbulence have fairly broad, flat spectral characteristics. In other words, they are noise, not speech. The shape of the oral cavity changes noise into recognizable speech. The positions of the tongue, the lips, and the jaws change the resonance of the vocal tract, shaping the raw noise of restricted airflow into understandable sounds.

1.7.2 The LPC Model

The LPC model incorporates elements analogous to each of the elements of the vocal tract described above. It has an excitation function generator that models both types of restriction, a gain multiplication stage to model the possible levels of pressure from the lungs, and a digital filter to model the resonance in the oral and nasal cavities.

Figure 1–15 shows the LPC-12 vocal tract model in schematic form. The excitation function generator accepts coded pitch information as an input and can generate a series of pulses similar to vocal cord pulses. It can also generate white noise. The waveform is then multiplied by an energy factor that corresponds to the pressure from the lungs. Finally, the signal is passed through a digital filter that models the shape of the oral cavity. In the MSP50x3x, this filter has 12 poles, so the synthesis is referred to as LPC-12.

Figure 1–15. LPC-12 Vocal Tract Model



1.7.3 LPC Data Compression

The data compression for LPC-12 takes advantage of other characteristics of speech. Speech changes fairly slowly, and the oral and nasal cavities tend to fall into certain areas of resonance more that others. The speech is analyzed into frames generally from 10 ms to 25 ms long. The inputs to the model are calculated as an average for the entire frame. The synthesizer smooths or interpolates the data during the frame so that there is no abrupt transition at the end of each frame. Often speech changes even more slowly than the frame.

The Texas Instruments LPC model allows for a repeat frame in which the only values changed are the pitch and the energy. The filter coefficients are kept constant from the previous frame. To take advantage of the recurrent nature of resonance in the oral cavity, all the coefficients are encoded, with anywhere from 3 bits to 7 bits encoding each coefficient. The coding table is designed so that more coverage is given to the coefficient values that occur more frequently.

1.8 MSP50x3x Mask Options

The MSP50x3x can be configured to suit different applications with a variety of mask options.

1.8.1 Clock Select Option

The MSP50x3x family has three mask-selectable clock options: an internal oscillator; an external oscillator; or an input driven by an external clock.

The internal oscillator is recommended when the lowest-cost solution is required and the absolute accuracy of the oscillator is a secondary consideration. The internal clock is trimmed at probe to standard frequencies of 15.36 MHz and 19.2 MHz. The frequency of the internal clock can be switched between these two values in the software by setting or clearing the SPEED bit in mode register 2. When using the internal-clock option or the external-clock option, terminal 7 is available as Port B1.

The external oscillator mask option is recommended when an accurate frequency standard is important. When the external oscillator mask option is selected, Port B1 is not available because terminal 7 is used as the oscillator return line. Either a ceramic resonator or quartz crystal can be connected between the OSC IN and OSC OUT lines with appropriate capacitors to provide the desired frequency clock. Alternatively, the OSC IN terminal may be driven with an externally derived clock signal. When the external oscillator option is used, the SPEED bit in the mode register 2 has no function. The SETOFF instruction and INIT terminal disable the operation of the clock circuit. See Figure 1–16 for a suggested oscillator circuit.

The external clock mask option is recommended when an externally derived clock is available to drive the device. A 15.36-MHz or 19.2-MHz clock should be fed to OSC IN. See Figure 1–17 for a recommended circuit for this mask option. When the external clock option is used, the SPEED bit in the mode register 2 has no function. When using the external clock option, terminal 7 is available as Port B1.

Figure 1–16. Oscillator Circuit

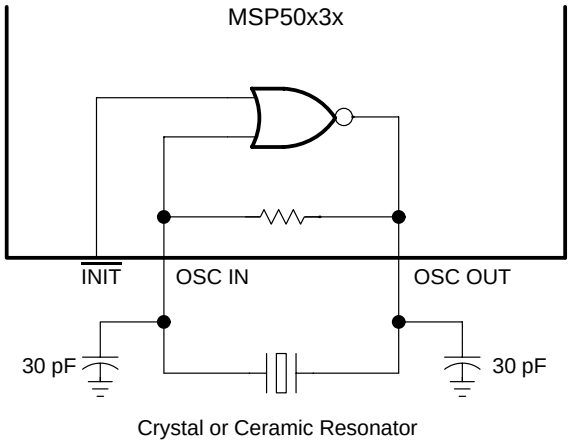
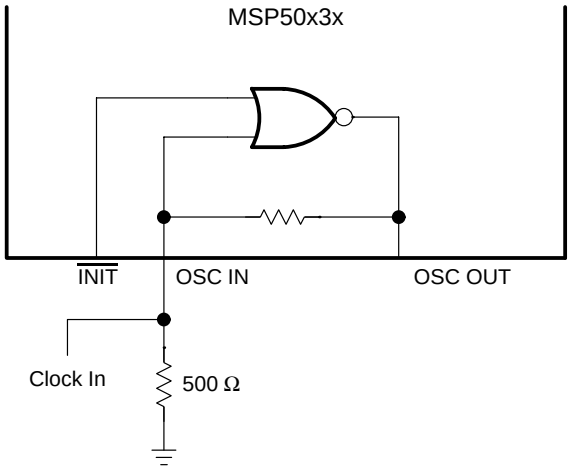


Figure 1–17. External Clock Interface



1.8.2 DAC Option

The DAC for the MSP50C32/33/34 can be selected as either a two-pin push pull or a one-pin analog. The MSP50C37 has only one output option, the one-pin analog power amplifier output option. See Section 1.5, *D/A Options*, for more information.

1.8.3 Power Amplifier Options (MSP50C37 Only)

The MSP50C37 has two mask-selectable power amplifier output options: a one-pin analog and a two-pin push pull differential. When the one-pin analog option is selected, the PW1 terminal is the power amplifier output and the PW2/AMPV_{DD} is the power amplifier current supply. When the two-pin push-pull option is selected, PW1 and PW2/AMPV_{DD} are the power amplifier differential outputs. For both options, the amplifier voltage gain is 10×.

Chapter 2

MSP50x3x Family Architecture

This chapter describes the architecture and function of the MSP50x3x family of speech synthesizers including RAM, ROM, registers, flags, and the DAC.

Topic **Page**

2.1 MSP50C3x Family Architecture

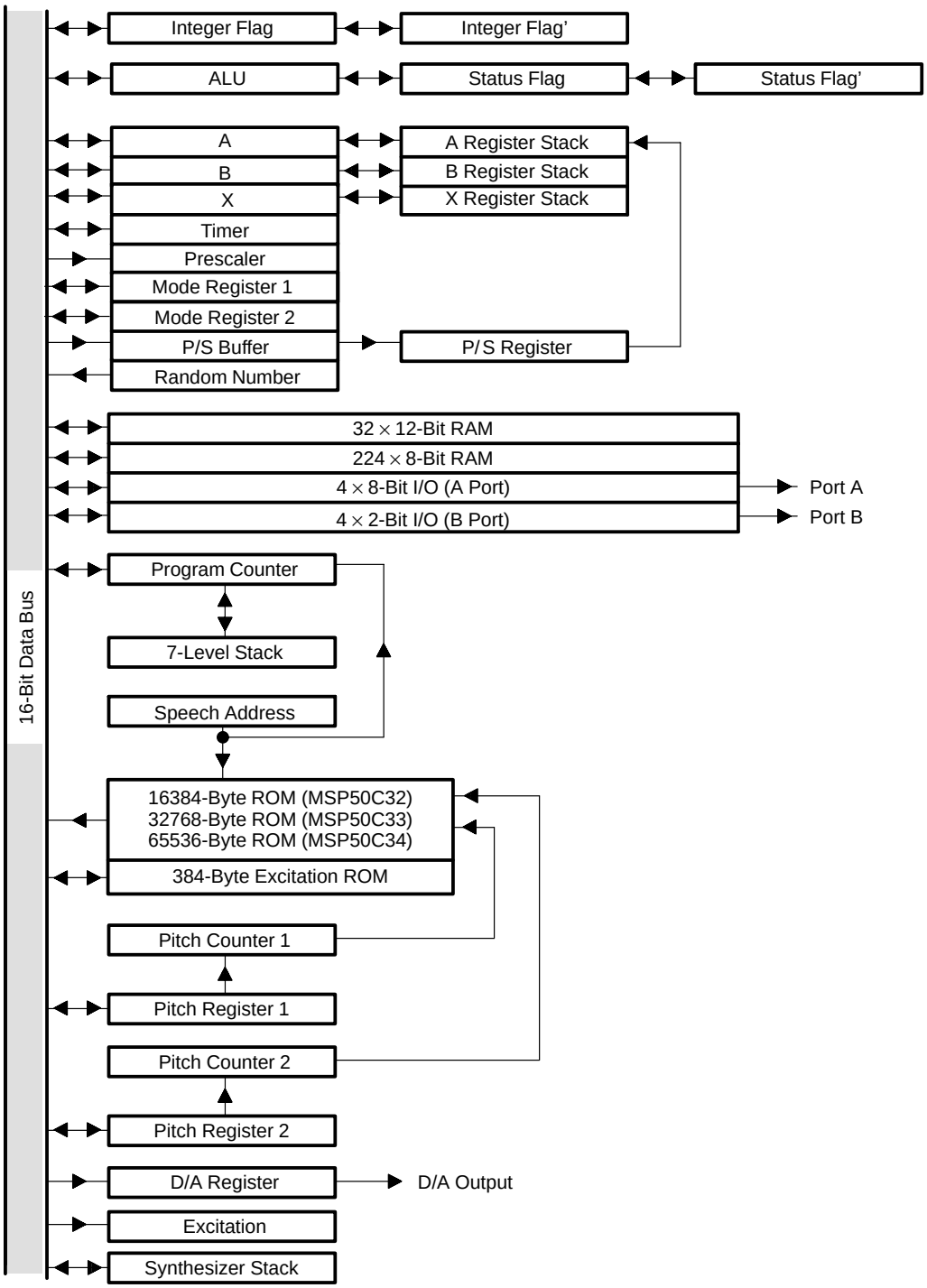
As shown in the block diagram in Figure 2–1, the major components of the MSP50C3x are a speech synthesizer; an 8-bit microprocessor; an internal 16K byte (MSP50C32), 32K byte (MSP50C33), or 64K byte (MSP50C34) ROM; and input/output ports.

The system clock can be internally generated or be driven externally. When the internal clock is used, the clock operates under software control and can be programmed to one of two frequencies, 19.2 MHz (used when LPC is operating at 10,000 samples per second) or 15.36 MHz (for 8,000 samples per second). If the internally generated frequencies are not sufficiently accurate, the device can be configured with a mask option to provide custom frequency operation using an external clock signal, an external ceramic resonator, or a quartz crystal (with appropriate capacitors).

When LPC synthesis is disabled, instructions are fetched by the microprocessor at 1/16 of the clock frequency. These instructions control the actions of the MSP50C3x. By placing different instruction patterns in the ROM, the MSP50C3x can be programmed to accomplish a wide variety of tasks. To generate speech, the processor accesses speech data from the internal ROM internal RAM, or some external source. Once the data has been read, the processor must unpack and decode the individual speech parameters and store the results in a dedicated section of the RAM.

The synthesizer shares access to the RAM and addresses the individual parameter locations as needed when generating speech. Each LPC synthesizer uses approximately one quarter of the available instruction cycles when enabled. The instruction execution rate slows to 3/4 when one LPC synthesizer is enabled and to 1/2 when both LPC synthesizers are enabled.

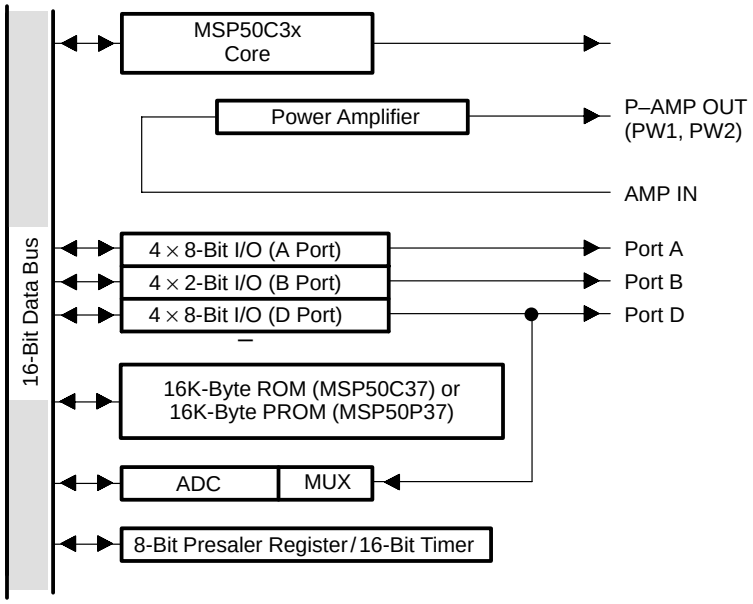
Figure 2–1. MSP50C3x System Block Diagram



The block diagram in Figure 2–2 shows that the MSP50x37 consists of the MSP50C3x core, several I/O ports (A, B, and D), 16K-byte ROM (MSP50C37) or PROM (MSP50P37), an 8-bit A/D converter with an analog multiplexer that is connected to the D Port and power amplifier.

Since the device is structured based on the MSP50C3x core, most of the features associated with performance of the core, such as clock speed, instruction cycle, RAM structure, speech synthesizer operations, etc., are the same as those of the MSP50x32/33/34.

Figure 2–2. MSP50x37 System Block Diagram



2.1.1 Read-Only Memory (ROM)

The MSP50C32 has 16K-bytes of ROM. The MSP50C33 has a 32K-byte ROM. The MSP50C34 has a 64K-byte ROM and the MSP50P34 has 64K-bytes of PROM. The MSP50C37 has 16K-bytes of ROM, and the MSP50P37 has 16K-bytes of PROM. The MSP50C3x can program instructions and speech data as required by the application. Certain locations in the ROM, described in Table 2–1, are reserved for specific purposes.

Table 2–1. Reserved ROM Locations

Address	Function
0000h	Execution start location after $\overline{\text{INIT}}$ rising edge
0002h	Execution start location after I/O wakeup falling edge
0010h – 001Fh	Interrupt start locations (see Section 2.3, *Interrupts*)
3FDBh – 3FFFh	Texas Instruments test code (MSP50C32/37)
7FDBh – 7FFFh	Texas Instruments test code (MSP50C33)
FFDBh – FFFFh	Texas Instruments test code (MSP50C34)

The ROM can be accessed in the following four ways:

- The program counter addresses processor instructions (see Chapter 4 for instruction definitions).
- The GET instruction transfers 1 to 8 bits from the ROM to the A register. The GET counter is initialized by the LUAPS instruction. The SAR (speech address register) points to the ROM location to be used.
- The LUAA instruction transfers a byte from the ROM into the A register. The value in the A register when LUAA is executed points to the ROM address to be used.
- The LUAB instruction transfers a byte from the ROM into the B register. The value in the A register when LUAB is executed points to the ROM address to be used.

2.1.2 Program Counter

The MSP50C3x has a 16-bit program counter that points to the next instruction to be executed. After the instruction is executed, the program counter is normally incremented to point to the next instruction.

When an interrupt occurs, the program counter is loaded with the interrupt vector address; where execution resumes. See Section 2.3, *Interrupts*, for more information. The following instructions modify the program counter:

- BR Branch
- BRA Branch to address in A register
- SBR Short branch
- RETN Return from subroutine
- RETI Return from interrupt
- CALL Subroutine Call

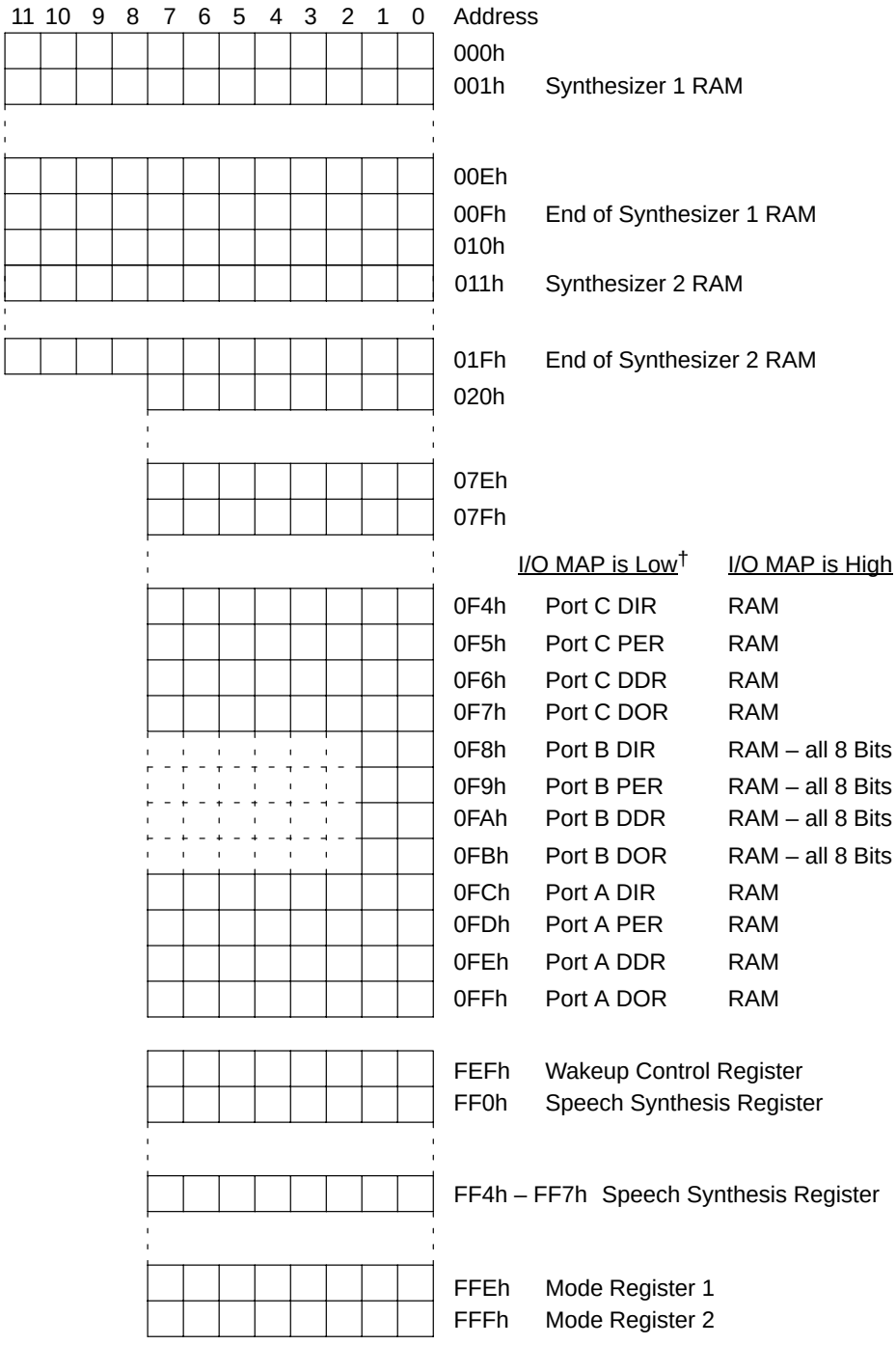
2.1.3 Program Counter Stack

The program counter stack has seven levels. When a subroutine is called or an interrupt occurs, the contents of the program counter are pushed onto the stack. When a RETN (return from subroutine) or an RETI (return from interrupt) is executed, the contents of the top stack location are popped into the program counter.

2.1.4 MSP50C3x Random Access Memory (RAM)

The MSP50C3x has 256 locations of general purpose RAM (Figure 2–3 or Figure 2–4). The first 16 RAM locations, which are 12 bits wide, are used by the first synthesizer when it is enabled. The next 16 RAM locations, which are also 12 bits wide, are used by the second synthesizer when it is enabled. The block of RAM associated with a synthesizer is released to general use when the synthesizer is not enabled. The remaining 224 RAM locations are 8 bits wide. When not synthesizing, the entire RAM can be used for algorithm data storage. The I/O control registers are mapped into the RAM address space from 0F0h to 0FFh (depending on which part is used) when the I/O MAP bit in mode register 2 is low. When the I/O MAP bit is high, RAM is mapped into 0F0h through 0FFh. For more information, see subsection 2.1.17, *Input/Output Ports*. The two mode registers are also mapped into the RAM address space at FFEh and FFFh. For more information, see subsection 2.1.18, *Mode Registers*.

Figure 2–3. MSP50x32/33/34 RAM Map



† Port C is only included on the MSP50C34 die form and on the MSP50P34 NW package.

Figure 2–4. MSP50x37 RAM Map

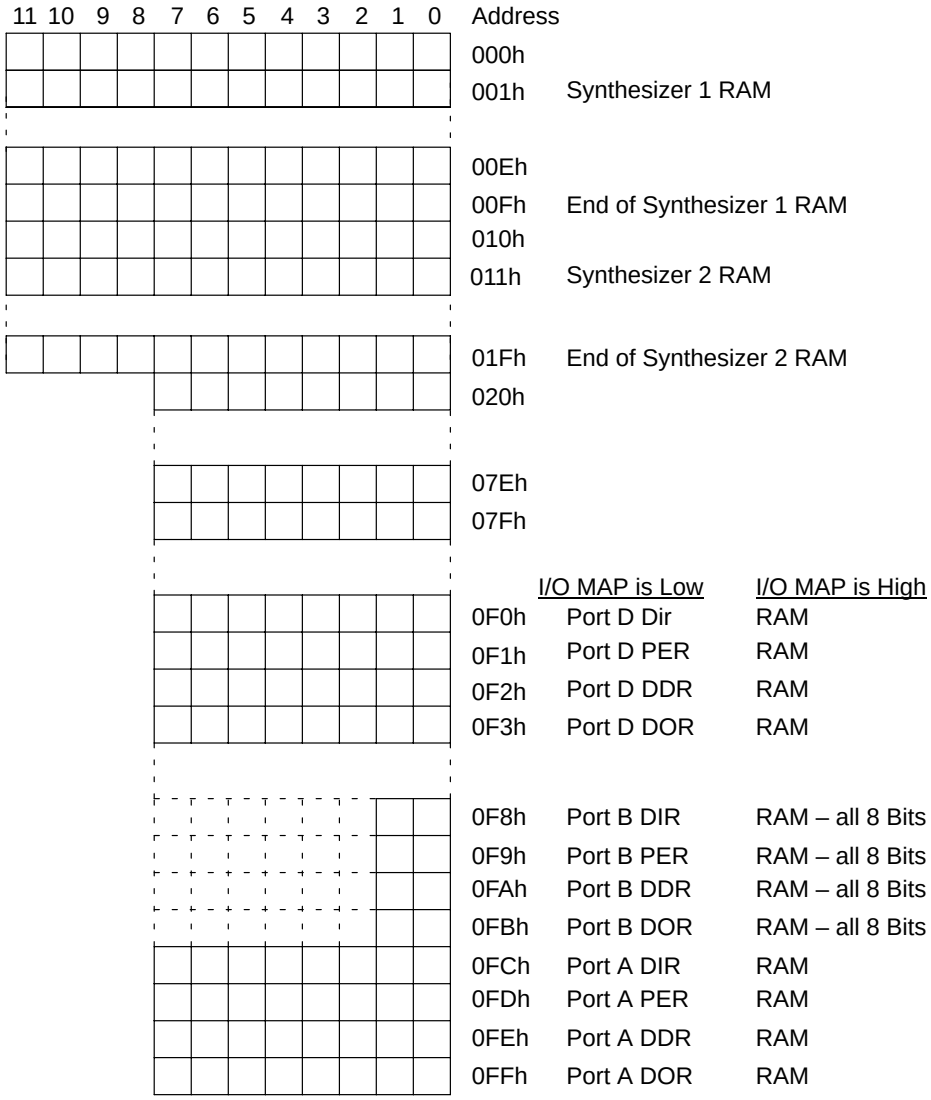
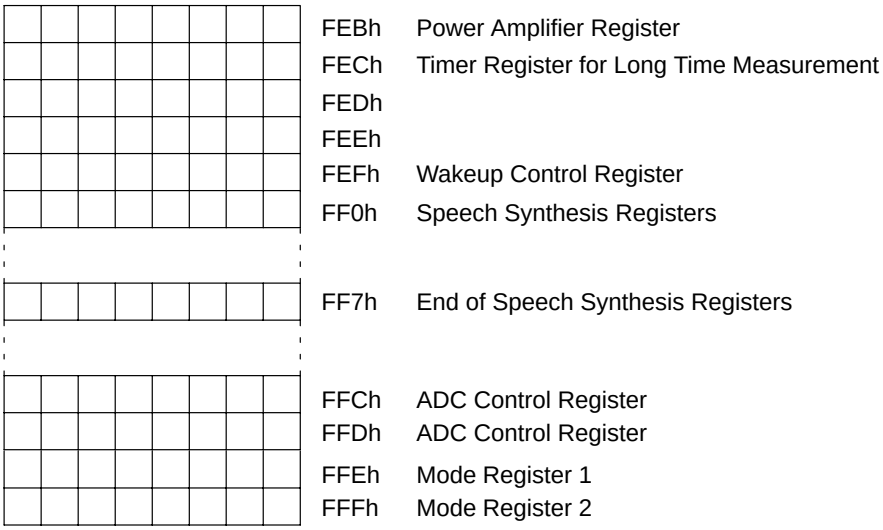


Figure 2–4. MSP50x37 RAM Map (Continued)



2.1.5 MSP50C3x Memory-Mapped Registers

Several internal registers are mapped into the RAM address space. Table 2–2 shows the memory-mapped register allocations for all devices combined.

Table 2–2. Memory-Mapped Registers

Address	Function	
F0h	Port D Input Register	(see subsection 2.1.17)
F1h	Port D Pullup Enable Register	(see subsection 2.1.17)
F2h	Port D Data Direction Register	(see subsection 2.1.17)
F3h	Port D Data Output Register	(see subsection 2.1.17)
F4h	Port C Input Register	(see subsection 2.1.17)
F5h	Port C Pullup Enable Register	(see subsection 2.1.17)
F6h	Port C Data Direction Register	(see subsection 2.1.17)
F7h	Port C Data Output Register	(see subsection 2.1.17)
F8h	Port B Input Register	(see subsection 2.1.17)
F9h	Port B Pullup Enable Register	(see subsection 2.1.17)
FAh	Port B Data Direction Register	(see subsection 2.1.17)
FBh	Port B Data Output Register	(see subsection 2.1.17)
FCh	Port A Input Register	(see subsection 2.1.17)
FDh	Port A Pullup Enable Register	(see subsection 2.1.17)
FEh	Port A Data Direction Register	(see subsection 2.1.17)
FFh	Port A Data Output Register	(see subsection 2.1.17)
FEBh	Power Amplifier Register	(MSP50x37 only)
FECh	Timer Register	(MSP50x37 only)
FEDh	Timer Register	(MSP50x37 only)
FEEh	Timer Register	(MSP50x37 only)
FEFh	Wakeup Select Register	
FF0h	Y1S (16 bit)	(synthesizer control register)
FF1h	PITCH2 (16 bits)	(synthesizer control register)
FF2h	Y1 (16 bits)	(synthesizer control register)
FF3h	PITCH (16 bits)	(synthesizer control register)
FF4h	TEMP (16 bits)	(synthesizer control register)
FF5h	TEMP (16 bits)	(synthesizer control register)
FF6h	TEMP (16 bits)	(synthesizer control register)
FF7h	TEMP (16 bits)	(synthesizer control register)
FFCh	A/D Register	(MSP50x37 only)
FFDh	A/D Register	(MSP50x37 only)
FFEh	Mode Register 1	(see subsection 2.1.18)
FFFh	Mode Register 2	(see subsection 2.1.18)

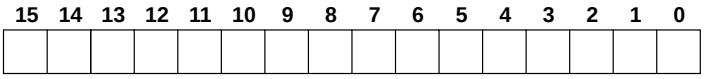
2.1.6 Arithmetic Logic Unit (ALU)

The ALU performs arithmetic and logic functions for the microprocessor and the synthesizer. The ALU is 16 bits wide, providing the resolution needed for speech synthesis. When 8-bit or 12-bit data are transferred to the ALU, they are right justified. The input to the upper 4 or 8 bits can be either zeros (integer mode) or equal to the MSB of the 8-bit data or 12-bit (extended sign mode) depending on the arithmetic mode selected using the EXTSG and INTGR instructions. See the description of each instruction for specific information in Chapter 4. All bit and comparison operations are performed on the lower 8 bits. The ALU is capable of doing an 8-bit by 16-bit multiply with a 16-bit scaled result in a single instruction cycle.

2.1.7 A Register

The A register, or accumulator, is the primary 16-bit register and is used for arithmetic and logical operations. It can be loaded with the contents of ROM, RAM, and most of the other registers. The A register contents can be written to RAM and most registers. The contents are saved in a dedicated storage register during level-1 interrupts and restored by the RETI instruction. When leaving a level-1 interrupt routine using the RETI instruction, the contents of the A register prior to the execution of the RETI instruction are lost. The BRA (Branch to address contained in the A register) can branch between 8K pages.

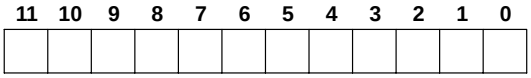
A Register



2.1.8 X Register

The X register is a 12-bit register used as a RAM index register. All RAM access instructions (except for the direct addressing instructions TAMD, TMAD, and TMXD) use the X register to point to a specific RAM location. The X register can also be used as a general purpose counter. The contents of the X register are saved during level-1 interrupts and restored by the RETI instruction. When leaving a level-1 interrupt routine using the RETI instruction, the contents of the X register prior to the execution of the RETI instruction are lost. Sign extension does not affect data transferred to the X register.

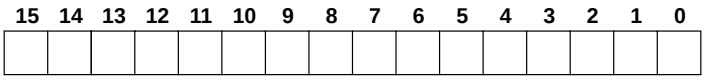
X Register



2.1.9 B Register

The 16-bit B register is used for temporary storage. It is helpful for storing a RAM address because it can be exchanged with the X register using the XBX instruction. The B register can be added to, subtracted from, or exchanged with the A register, making it useful for data storage after calculations. The contents of the B register are saved during level-1 interrupts and restored by the RETI instruction. When leaving a level-1 interrupt routine using the RETI instruction, the contents of the B register prior to the execution of the RETI instruction are lost.

B Register



2.1.10 Status Flag

The 1-bit status flag is set or cleared by various instructions depending on the result of the instruction. Refer to the individual description of instructions in Chapter 4 to determine the effect an instruction has on the value of the status flag. The BR, SBR, and CALL instructions are conditional, modifying the program counter only when the status flag is set. The value of the status flag is initialized to 1 upon powerup, initialization, or wakeup. The status flag is set to 1 by any interrupt.

Status Flag

2.1.11 Integer Mode Flag

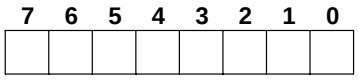
The 1-bit integer mode flag is set by the INTGR instruction and cleared by the EXTSG instruction. When the integer mode flag is set (integer mode), the upper bits of the data less than 16 bits in length are zero filled when being transferred to, added to, or subtracted from the A and B registers. The upper bits of data less than 12 bits in length are zero filled when being transferred to the X register. When the integer mode flag is cleared (extended sign mode), the upper bits of data less than 16 bits in length are sign extended when being transferred to, added to, or subtracted from the A and B registers. The upper bits of data less than 12 bits in length are sign extended when being transferred to the X register. The value of the integer mode flag is saved during interrupts and restored by the RETI instruction.

Integer Mode Flag

2.1.12 Timer Register

The 8-bit timer register generates interrupts and also counts events. It decrements once each time the timer prescale register goes from 00h to FFh. It can be loaded using the TATM instruction and examined with the TTMA instruction. When it decrements from 00h to FFh, a level-2 interrupt request is generated. When interrupts are enabled and no interrupt has been processed already, an immediate interrupt occurs; if not, the interrupt remains pending until interrupts are enabled. The timer continues to count whether or not it is reloaded. The timer does not decrement before it is loaded with an initial value using the TATM instruction. However, on the evaluation module unit (EMU), the timer decrements after a STOP/RUN.

Timer Register



2.1.13 Timer Prescale Register

The 8-bit timer prescale register is a programmable divider between the processor clock and the timer register. When it decrements from 00h to FFh, the timer register is also decremented. The timer prescale register is then reloaded with the value in its preset latch, and the counting starts again. When the value N is loaded to the preset latch, the prescale register counts through N+2 states.

The timer prescale register clock comes from the internal clock. The internal clock runs at 1/32 the clock frequency of the chip; thus, the timer prescale register decrements once every two instruction cycles when not in LPC mode. The TAPSC instruction loads the timer prescale register preset latch. If the timer has not yet been initialized with the TATM instruction, the TAPSC instruction also loads the timer prescale register.

Timer Prescale Register

7	6	5	4	3	2	1	0

2.1.14 Pitch Register and Pitch Period Counter (PPC)

The MSP50C3X family contains two pitch registers and two pitch period counters (PPC). One set is used for each synthesizer. The following discussion is presented for one synthesizer.

Although the 16-bit pitch register and pitch period counter are part of the synthesizer, they affect the microprocessor in many ways. The pitch period counter controls the timing of the periodic impulse (excitation function) that simulates the vocal cords. On the MSP50C3x, the pitch period counter also controls the interpolation of all synthesis parameters during each frame. This pitch-synchronous interpolation helps to minimize the inevitable noise from interpolation by making it occur at the lowest energy part of synthesis and by making it a harmonic of the fundamental frequency.

The pitch register is used when LPC speech is being synthesized. The following discussion presumes that the LPC mode is active. The pitch register is loaded with the TASYN instruction. The channel bit in mode register 2 controls which pitch register is loaded with the TASYN instruction. The pitch period counter is decremented by 20h for each sample, with synthesis samples occurring at an 8-kHz or 10-kHz rate. When the pitch period counter decrements past zero, the pitch register is added to it. When the pitch period counter goes below 200h or when a pitch register is added to it with a result less than 200h, the PPC bit for that synthesis channel is set high. This bit can be polled by the

microprocessor to determine when interpolation should be performed. The excitation function is loaded to the input of the LPC filter while the pitch period counter is between 140h and 000h. For further information, see Chapter 5.

Pitch Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

For voiced or unvoiced frames, the LSB and MSB of the A register must both be set to 0 when data is transferred from the A register to the pitch register with the TASYN instruction (see the following illustration). If this is not done, problems with the MSP50C3x chip can occur. Also, these problems may not be apparent when using the MSE50C3x chip.

A Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															0

Pitch Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															0

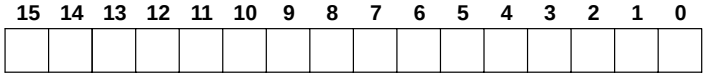
For voiced frames, the pitch register must be loaded with a value no higher than 7FFEh. In addition, there are three recommendations for the minimum pitch register value for voiced frames. First, it is required that the pitch register value be 42h or higher. Second, it is strongly recommended that the pitch register be loaded with a value of 142h or higher. This permits the complete excitation pulse to be used in the LPC synthesis. Third, for best results with the recommended software algorithms, a pitch register value of 202h or higher is recommended. The requirement that the pitch register value be less than or equal to 7FFEh and the recommendation of a value greater than or equal to 142h results in a pitch range of 9 Hz to 994 Hz when operating with a 10-kHz sample rate.

For unvoiced frames, the pitch register is required to be loaded with a value between 42h and 3FEh. If this is not done, problems with the MSP50C3x chip can occur that are not apparent on the MSE50C3x.

2.1.15 Speech Address Register (SAR)

The speech address register (SAR) is a 16-bit register that points to data in the internal ROM. The LUAPS instruction transfers the value in the A register to the speech address register and loads the parallel-to-serial register (see subsection 2.1.16, *Parallel-to-Serial Register*) with the internal ROM value pointed to by the SAR. The GET instruction can then bring 1 to 8 bits at a time from the parallel-to-serial register into the accumulator. Whenever the parallel-to-serial register becomes empty, it is loaded with the internal ROM value pointed to by the SAR, and the SAR is incremented.

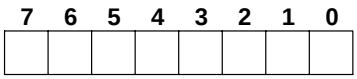
Speech Address Register



2.1.16 Parallel-to-Serial Register

The 8-bit parallel-to-serial register is used primarily to unpack speech data. It can be loaded with 8 bits of data from the internal ROM pointed to by the speech address register or the internal RAM pointed to by the X register. The LUAPS instruction initializes the parallel-to-serial register and zeroes its bit counter. GET instructions can then transfer 1 to 8 bits from the parallel-to-serial register to the accumulator. When the parallel-to-serial register is empty, it is automatically reloaded. When the GET is from the RAM, however, the X register is not automatically incremented. The RAMROM bit in mode register 1 controls the source of the parallel-to-serial register. See the speech address register description in subsection 2.1.15, *Speech Address Register (SAR)*, for more information.

Parallel-to-Serial Register



2.1.17 Input/Output Ports

In the MSP50x32/33/34 devices, ten bidirectional lines – 8-bit Port A and 2-bit Port B – are available for interfacing with external devices. The MSP50C34 in die form has 24 bidirectional lines (an 8-bit Port A, an 8-bit Port B, and an 8-bit Port C). The MSP50x37 has 18 bidirectional I/O lines – 8-bit A Port, 2-bit B Port, and 8-bit D Port, for external interface. Each bit is individually programmable as an input or an output under the control of the respective data direction register. In addition, each output bit can be individually programmed using the pullup enable register for one of two output modes – push-pull or open-drain (no pullup). Each input bit can be programmed by the same register for resistive pullup or high impedance. The four registers associated with each of the two I/O ports are memory mapped. Only 2 bits of Port B are available on the outside of the chip, and the states of the upper 6 bits of its registers are undetermined. On the MSP50C34 all eight bits are available when it is in die form. Transfers from any of the I/O port registers to the A register leave the upper 8 bits (bits 15 – 8) undetermined. On the MSP50x37, the D Port can be an input for the A/D converter. One of the 8-bit ports can be selected by the control register and the analog input signal is multiplexed to the 8-bit SAR A/D converter. Details of the I/O registers are shown in Table 2–3 and Table 2–4.

Table 2–3. I/O Registers

		Location			
Register	**Type**	**Port A**	**Port B**	**Port C ('50C34 Only)**	**Port D ('50x37 Only)**
Data Input Register (DIR)	Read Only	FCh	F8h	F4h	F0h
Pullup Enable Register (PER)	Read/Write	FDh	F9h	F5h	F1h
Data Direction Register (DDR)	Read/Write	FEh	FAh	F6h	F2h
Data Output Register (DOR)	Read/Write	FFh	FBh	F7h	F3h
A/D terminal select and conversion start (MSP50x37 only)	Read Only	—	—	—	FFCh or FFDh

Table 2–4. I/O Terminal Functions

I/O Terminal Function	DOR	DDR	PER	Terminal State
Input, high impedance	X	0	0	High impedance
Input, internal pullup	X	0	1	Passive pullup
Output, active pullup	0	1	0	0
Output, active pullup	1	1	0	1
Output, open drain	0	1	1	0
Output, open drain	1	1	1	High impedance

A read of the DDR, PER, and DOR registers indicates the last value written to them.

A read of the DIR always indicates the actual signal level on the I/O terminal, which is true even when the DDR is set to output. This allows true bidirectional data flow without having to switch the port between input and output. To avoid high current conditions, this should only be attempted on terminals set for open drain with a 1 written to the data register.

Unused I/O Terminals

Any unused I/O terminals should be tied high or low. Floating I/O terminals cause leakage current.

24 I/O Pins

The MSP50P34 has 24 I/O pins. Only 10 of them are wired to pins when the 16-pin package is used. The remaining I/O bits (Port C and bits 2–7 of Port B) should be programmed either as totem-pole output pins or as input pins with passive pullups to avoid these pins being left floating. If this is not done, leakage current may result.

Leaving a high-impedance I/O terminal unconnected could cause power consumption to rise while the processor is in run mode. The power consumption is between V_{DD} and V_{SS} with no increase in current through the input. This should cause no problem with device functionality.

When the part is in sleep mode, unconnected high-impedance terminals have no effect on either power consumption or device functionality.

The I/O terminal can also be put in slave mode, making the MSP50C3x usable as a peripheral to a host microprocessor. Port A can be connected to an 8-bit data bus and controlled by R/W_ (Port B1) and chip enable (Port B0). A read (R/W_ high and chip enable low) puts the Port A output latch values out on Port A. A write (R/W_ low and chip enable low) latches the value on the data bus into the Port A input latch. In addition, bit 7 of the A output latch is cleared. This makes it possible to use A7 as a write handshake line. Any lines used on the data bus in this mode must be configured as inputs.

When the external clock option is selected, B1 is not available because it is used for the OSC OUT function.

If the PCM1 and LPC1 mode register bits are both cleared, a high-to-low transition on B1 causes a level-1 interrupt. The B1 pulse must have a minimum width of 1 μs. This can generate an interrupt with an external event.

2.1.18 Mode Registers

There are two 8-bit mode registers that are memory mapped. The contents of both mode register 1 is cleared to 0 upon power-up reset, wakeup, or when the $\overline{\text{INIT}}$ line pulses low. The contents of the mode register 2 is initialized to 09h (PPC bits are both high) upon INIT. The contents of the two mode registers are not saved during a subroutine call or interrupt.

Mode register 1 is memory mapped to RAM address FFEh and mode register 2 is memory mapped to RAM address FFFh. The contents of either mode register can be copied to the A register using the TMA instruction. Individual bits of either mode register can be tested using the TSTCM instruction or modified using the ANDCM and ORCM instructions. The TAMODE instruction transfers the bottom 8 bits of the A register to mode register 1.

2.1.18.1 Mode Register 1

The usage of the mode register 1 bits are given in Table 2–5. Mode register 1 controls the first synthesizer channel, the interrupt mode, the slave mode, and the GET mode.

The ENA1 and ENA2 bits in the mode register enable or disable the level-1 and level-2 interrupts respectively. If an interrupt condition occurs while the corresponding ENA1 or ENA2 bit in the mode register is cleared to 0, an interrupt pending latch is set, and the execution of the interrupt is delayed until the interrupt is enabled.

The LPC1, PCM1, and UNVOICE1 bits control the activity of the first synthesis channel. See Section 2.2, *Speech Synthesis*, for more information. The LPC1 and PCM1 bits control the interrupt vectors.

Note: Using the ORCM Instruction

When using the ORCM instruction on the UNVOICE1 bit, the LPC1 bit should also be reset to avoid a glitch in the LPC (i.e., ORCM UNV + LPC1).

The SLAVE bit enables a special I/O mode designed to enable the MSP50C3x to operate as a slave processor under the control of an external master microprocessor. See Chapter 5, *Applications*, for more information.

The RAMROM bit selects the data source for GET instructions. When RAMROM is low, the GET instruction accesses the internal ROM. When the RAMROM bit is set high, the GET instruction accesses the RAM location selected by the X register.

Table 2–5. Mode Register 1

Bit	Name	Bit Low	Bit High
0	ENA1	Disables the level-1 interrupt	Enables the level-1 interrupt
1	LPC1	Disables the channel 1 LPC processor – all instruction cycles used by the microprocessor	Enables the channel 1 LPC processor – 25% of instruction cycles dedicated to service this channel. RAM locations 00h – 0Fh are dedicated to the LPC synthesis.
2	PCM1	Disables the PCM mode	Enables the PCM mode on channel 1
3	ENA2	Disables the level-2 interrupt	Enables level-2 interrupt
4	Reserved	Leave this bit at 0	Do not set this bit
5	RAMROM	Enables data source for GET instructions to be either internal ROM	Enables the data source for the GET instructions to be internal RAM
6	SLAVE	Enables I/O master operation. All available I/O terminals are controlled by the internal microprocessor	Enables the I/O slave operation. Terminal B0 becomes hardware chip enable strobe, and B1 becomes R/W_, Port A is controlled by B0 and B1
7	UNVOICE1	Enables the pitch-controlled excitation sequence when in LPC mode (PCM1 low, voiced) on channel 1	Enables the random excitation sequence when in LPC mode (PCM1 low, voiced) on channel 1

2.1.18.2 Mode Register 2

The usage of the mode register 2 bits are given in Table 2–6. Mode register 2 controls the second synthesizer channel, the I/O map, the channel selected, the internal oscillator speed, and reports the status of the two pitch period counters.

The LPC2, PCM2, and UNVOICE2 bits control the activity of the second synthesis channel. See Section 2.2, *Speech Synthesis*, for more information.

The PPC1 and PPC2 bits report the status of the two pitch period counters used for LPC synthesis. They are both initialized to 1 upon power up, $\overline{\text{INIT}}$, or wakeup. PPC1 is set to 1 when the pitch period counter for channel 1 decrements below 200h. PPC2 is set high when the pitch period counter for channel 2 decrements below 200h. They only get cleared to 0 by an explicit write to the register using either the TAM instruction or the ANDCM instruction. Both bits are set to 1 upon $\overline{\text{INIT}}$ and subsequently need to be set in software before starting LPC synthesis.

The CHANNEL bit selects which channel the TASYN addresses. When CHANNEL is 0, the TASYN loads the pitch register for channel 1. When CHANNEL is set to 1, the TASYN instruction loads the pitch register for channel 2.

The SPEED bit controls the speed of the internal oscillator. When this bit is cleared to 0, the internal oscillator generates a 15.36-MHz clock and synthesis operates at 8000 samples/second sample rate. When this bit is set to 1, the internal oscillator generates a 19.2-MHz clock and synthesis operates at a 10 000 samples/second rate. When the external oscillator mask option is in effect, SPEED has no affect.

The I/O_MAP bit controls the RAM address spaces located at 0xF8h through 0xFFh. When this bit is cleared to 0 (the default condition), the I/O ports (port A and port B) are mapped into these locations. When this bit is set to 1, the I/O ports are hidden and RAM is mapped into these addresses. The RAM and the ports maintain separate storage locations so that different data can be maintained in the RAM location and the port latch while using the same address.

Table 2–6. Mode Register 2

Bit	Name	Bit Low	Bit High
0	PPC1	Cleared low using software	Set high when PPC decrements below 200h on channel 1 or by program init or wakeup
1	LPC2	Disables LPC mode	Enables LPC mode on channel 2 — 25% of the instruction cycles are dedicated to service this channel. RAM locations 10h – 1Fh are dedicated to LPC synthesis.
2	PCM2	Disables PCM mode	Enables PCM mode on channel 2
3	PPC2	Cleared low using software	Set high when PPC decrements below 200h on channel 2 or by program init or wakeup
4	CHANNEL	Synthesizer channel 1 selected. The TASYN instruction addresses first channel.	Synthesizer channel 2 selected. the TASYN instruction addresses the second channel.
5	SPEED	Selects internal clock speed of 15.36 MHz (8,000 samples/second sample rate)	Selects internal clock speed of 19.2 MHz (10,000 samples/second sample rate)
6	I/O MAP	I/O ports (ports A, B, C, and D) are mapped into 0xF0h through 0xFFh.	I/O ports are hidden and RAM is mapped into 0xF0h through 0xFFh.
7	UNVOICE2	Enables the pitch-controlled excitation sequence when in LPC mode (PCM1 low, voiced) on channel 2	Enables the random excitation sequence when in LPC mode (PCM1 low, voiced) on channel 2

2.2 Speech Synthesis

The MSP50C3x family incorporates two synthesis channels. Each synthesis channel can be controlled separately by setting or clearing bits in the two mode registers, loading one or both of the pitch registers using the TASYN instruction, and by properly loading values to the synthesis RAM. The output of each channel is added together at the input of the oversampling output filter. Each synthesis channel can operate in one of four modes based upon the setting of the LPCx and PCMx bits in the mode register. In the following discussion, LPCx refers to either the LPC1 or the LPC2 bit in the mode register depending upon which synthesis channel is active. In a similar manner, PCMx , UNVOICEx, and PPCx refer to PCM1, PCM2, UNVOICE1, UNVOICE2, PPC1, and PPC2 depending upon context.

2.2.1 Synthesizer Mode 0 – Off

When the PCMx and LPCx bits are both cleared, the synthesizer channel is disabled. No microprocessor cycles are needed to service a channel in this mode. The TASYN instruction transfers the contents of the A register to the pitch register selected by the CHANNEL bit in mode register 2, making it easy to preload the pitch register prior to starting the LPC synthesizer. If the LPC1 and PCM1 bits in mode register 1 are both cleared to 0, the level-1 interrupt is triggered by a high-to-low transition on terminal B1 and the DAC is disabled.

2.2.2 Synthesizer Mode 1 – LPC

Synthesizer mode 1 is the normal speaking mode. It is selected by setting the LPCx bit to 1 and clearing the PCMx bit to 0. In this mode, the TASYN instruction transfers the contents of the A register to the pitch register selected by the CHANNEL bit in mode register 2. If synthesis channel 1 is in this mode, the level-1 interrupt occurs synchronously with the filter speech sample rate. Approximately 25 percent of the instruction cycles are used by each synthesis channel using this mode.

The PPCx bit in mode register 2 is set to 1 when the pitch period counter for that channel decrements below 200h or when the pitch register is added to the pitch period counter with a result less than 200h. The PPCx bit in mode register 2 remains at 1 until reset to 0 by the software.

The microprocessor controls speech synthesis by unpacking and decoding parameters, setting the update interval (frame rate), and interpolating the parameters during the frame. The speech synthesizer acts as a 12-pole digital lattice filter, a pitch-controlled or white-noise excitation generator, a 2-pole digital low-pass filter, and a digital-to-analog converter (DAC). Speech parameter

input is received from dedicated space in the RAM, and speech samples are generated at 8 kHz or 10 kHz. Communication between the microprocessor and the speech synthesizer takes place by way of a shared memory space in the microprocessor. Refer to Chapter 5, *Applications*, for more information.

When the UNVOICEx bit is cleared to 0, the excitation to the LPC filter is a periodic pulse with its pulse period determined by the value loaded to the pitch register using the TASYN instruction. When the UNVOICEx bit is set to 1, the excitation to the LPC filter is a pseudo-random white noise sequence.

When synthesis channel 1 is operating in synthesizer mode 1, RAM locations 01h – 0Fh are dedicated to LPC synthesis. When synthesis channel 2 is operating in this mode, RAM locations 11h – 1Fh are dedicated to LPC synthesis.

2.2.3 Synthesizer Mode 2 – PCM

When the LPCx bit is zero and the PCMx bit is set to 1, the synthesizer is placed in PCM mode. In synthesizer mode 2, the synthesizer filter for the channel is disabled and does not use any microprocessor cycles, and TASYN transfers the contents of the A register directly to the input of the output filter. When synthesizer channel 1 is in this mode, the level-1 interrupt occurs at the speech sample rate, which gives access to the unfiltered D/A output.

2.2.4 Synthesizer Mode 3 – PCM excited LPC

When both the PCMx and the LPCx bits are set to 1, the LPC synthesizer runs normally with its excitation function being provided by software. The level-1 interrupt occurs at the speech sample rate, and the TASYN instruction transfers the A register to the excitation function input of the synthesizer selected by the CHANNEL bit in mode register 2. Each channel in this synthesizer mode 3 uses approximately 25% of the microprocessor instruction cycles.

2.2.5 Use of RAM by the Synthesizer

The synthesizer uses locations 01h to 1Eh in the RAM. When synthesis is taking place, the parameters for the synthesizer come directly from RAM locations 01h to 1Eh. The addresses are shown in Figure 2–5.

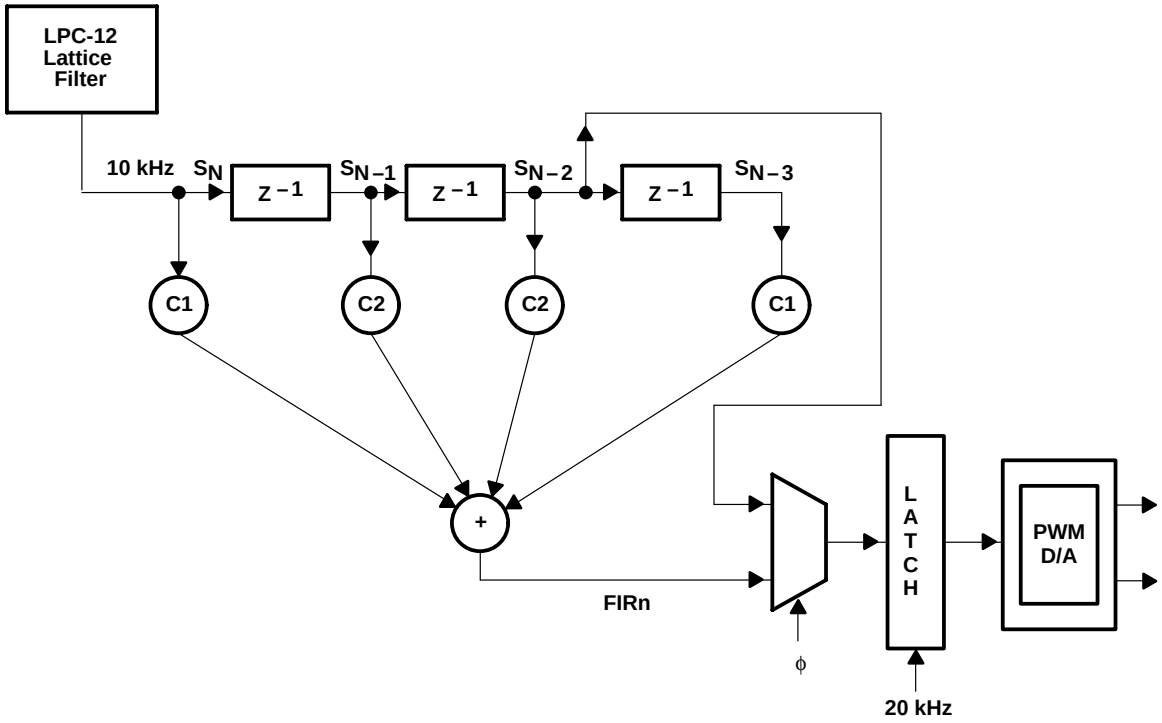
Figure 2–5. RAM Map During Speech Generation

Address	11	10	9	8	7	6	5	4	3	2	1	0	Description
000h													
001h													Energy (Channel 1)
002h													K12 (Channel1)
003h													K11 (Channel1)
004h													K10 (Channel1)
005h													K9 (Channel1)
006h													K8 (Channel1)
007h													K7 (Channel1)
008h													K6 (Channel1)
009h													K5 (Channel1)
00Ah													K4 (Channel1)
00Bh													K3 (Channel1)
00Ch													K2 (Channel1)
00Dh													K1 (Channel1)
00Eh													C1 (Low-Pass Filter)
00Fh													C2
010h													
011h													Energy (Channel2)
012h													K12 (Channel2)
013h													K11 (Channel2)
014h													K10 (Channel2)
015h													K9 (Channel2)
016h													K8 (Channel2)
017h													K7 (Channel2)
018h													K6 (Channel2)
019h													K5 (Channel2)
01Ah													K4 (Channel2)
01Bh													K3 (Channel2)
01Ch													K2 (Channel2)
01Dh													K1 (Channel2)
01Eh													C3 (Channel Scaling)

2.2.6 Low Pass Filter

The MSP50C3x synthesizer incorporates an oversampling output filter. Its flow diagram is shown in Figure 2–6. The output of the LPC filter (appropriately delayed) is alternately switched with the output of a FIR filter as shown at a rate double the speech sample rate. With the appropriate choice of parameters (C1 and C2), this filter interpolates data points and improves sound quality. Recommended values are C1 = F61h and C2 = B67h or C1 = E00h and C2 = A00h.

Figure 2–6. Oversampling Output Filter



2.2.7 Channel Scaling (C3)

When both LPC channels are operating, the channel 2 output is scaled relative to the channel 1 output by the value loaded to RAM location 1Eh. When it is desired that both channels operate with the same scaling, load RAM location 1Eh with the value 7FFh. Loading values smaller than this linearly scale the output of channel 2 with respect to channel 1.

2.2.8 Frame Length Control

The frame length is controlled by the value put into the prescale register and the range over which the timer is allowed to vary. Typical synthesis and interpolation routines let the timer decrement through a range of fixed size. Consequently, the prescale value should be selected to give the proper frame duration based on the timer range.

2.2.9 Digital-to-Analog Converter

The MSP50C3x contains an internal digital-to-analog converter (DAC) connected to the output of the synthesizer. The DAC is available in two pulse-density-modulated forms. See Section 1.5, *D/A Options*, for more information. The DAC output produces samples at a rate given by $f_{osc/1920}$. For a 19.2-MHz clock frequency, this formula results in an output sample rate of 20 KHz. For a 15.36-MHz clock frequency, this results in an output sample rate of 16 KHz. The DAC output rate is twice the speech sample rate with a digital low-pass filter in all modes except PCM mode. When the synthesizer is off (mode 0), the DAC goes to an off state. The off state is the same as a zero state.

2.3 Interrupts

The MSP50C3x has two interrupts: the level-1 interrupt and the level-2 interrupt. Both are enabled and disabled by bits in mode register 1. The level-1 interrupt has higher priority and has more hardware support. When a level-1 interrupt occurs, the program counter is placed on the program counter stack, and the status flag, integer mode flag, A register, B register, and X register are all saved in dedicated storage registers. Then the program counter is loaded with the interrupt start location and execution of the interrupt routine begins. When the interrupt routine returns, all these registers are restored, and the program counter is popped from the stack.

The level-1 interrupt is caused by one of four conditions depending on the state of the two mode-register 1 bits PCM1 and LPC1. These conditions, as well as the interrupt routine start address for each case, are shown in Table 2–7.

Table 2–7. Level-1 Interrupt Vectors

Address	PCM1	LPC1	Interrupt Trigger
0018h	0	0	Terminal B1 goes from high to low (see Section 2.1.17)
001Ah	0	1	f_{clock}/1920 (see subsection 2.1.14)
001Ch	1	0	f_{clock}/1920 (see subsection 2.2.3)
001Eh	1	1	f_{clock}/1920 (see subsection 2.2.4)

The level-2 interrupt has a lower priority and cannot interrupt the level-1 interrupt routine. Although, a level-2 interrupt can be interrupted by the level-1 interrupt. During a level-2 interrupt, the program counter, status bit, and integer mode flag are the only registers saved. The A register, X register, and B register must be saved by the program if they are used by both it and the routine being interrupted. The mode register is not saved. The level-2 interrupt is always caused by a timer underflow—the timer going from 00h to FFh—but it starts at different addresses depending on the state of two mode-register bits. Table 2–8 shows the level-2 interrupt vectors.

Table 2–8. Level-2 Interrupt Vectors

Address	PCM1	LPC1	Interrupt Trigger
0010h	0	0	All level-2 interrupts are caused by timer underflow.
0012h	0	1	
0014h	1	0	
0016h	1	1	

The interrupting conditions for the level-1 and level-2 interrupt set interrupt-pending latches. When an interrupt is enabled (and in the level-2 interrupt case, not overridden by a level-1 interrupt pending condition), the interrupt is taken immediately. If, however, the interrupt is not enabled, the interrupt pending latch causes an interrupt to occur as soon as the respective interrupt is enabled in mode register 1.

Interrupts are not taken in the middle of double-byte instructions, during branch or call instructions, or during the subroutine or interrupt returns (RETN or RETI). A single instruction software loop (instruction of BR, BRA, CALL, or SBR to itself) should be avoided since an interrupt is never taken. Consecutively executed branches or calls delay interrupts until after the execution of the instruction at the eventual destination of the string of branches (or calls).

If consecutive branches (or calls) are avoided, the worst-case interrupt delay in the main level is four instruction cycles. The worst-case delay occurs when the interrupt occurs during the first execution cycle of a branch and the first instruction at the branch destination address is a double-cycle instruction.

When the interrupt occurs, execution begins at the interrupt address. The state of the status bit is not known when the interrupt occurs, so a BR or CALL instruction should not be used for the first instruction. Two SBRs can be used, since one of them is always taken, or it may be possible to use some other instruction that sets the status bit, followed by an SBR.

The mode register is not saved and restored during interrupts. Any changes made to the mode register during interrupts remain in effect after the return, including the enabling and disabling of interrupts.

2.4 MSP50C3x Power Control and Initialization

Upon initial powerup, the status bit is set to 1, the sign extension is set to integer mode, the RAM contents are unknown, and the I/O RAM is cleared to all 0s, forcing the I/O terminals to a high-impedance input state. Mode Register 1 is initialized to 0, and Mode Register 2 is initialized to 09h. The software program begins operating at ROM location 0000. The device may subsequently be placed in one of several power saving modes:

- STANDBY

  Executing a SETOFF instruction places the device in a standby mode. When this occurs, the internal clock is shut down, the program stops executing, and the device is placed in a low-power state. The I/O lines retain their last programmed state.

  This mode can be terminated either by taking $\overline{\text{INIT}}$ low or by waking the device with a negative transition on a designated wakeup line. Any A Port line can be software selected as a wakeup line. This is done by setting the corresponding bit in the Wakeup Control Register (FEFh). When the device is restarted using a wakeup line, the program begins executing at ROM location 0002.

  When the device is restarted, the RAM is retained, the mode registers are cleared to 0, the I/O lines are unchanged, the status bit is set to 1, and the random number counter is unchanged. The device is otherwise completely initialized.

TSP50C1x Device Family

The TSP50C1x device family set the I/O to a high impedance input state after a SETOFF command. This is not true in the MSP50x3x family. The I/O is left in its last programmed state.

Additional Precautions

The MSP50P34 requires additional precautions when placing the device into the standby condition which are not required on the other MSP50C3x device family. If B0 is programmed as an input, than it should be driven either high or low. In addition, the I/O ports B2..B7 and C0..C7 exist on the MSP50P34 and will not be bonded out in the 16-pin package. Software should program these I/O ports to an output state. If these precautions are not taken, some leakage current may result.

- SLEEP

  Taking the $\overline{\text{INIT}}$ line low reinitializes the MSP50C3x and places the device in a sleep mode. The program counter is cleared to 0000, the random number counter is reinitialized, the I/O lines are set to a high-impedance state, and the state of the device is completely initialized. When the $\overline{\text{INIT}}$ signal returns high, the program begins executing at ROM location 0000 with the status bit set to 1.

2.5 MSP50C3x Clocks

The MSP50C3x can operate either off an internal clock requiring no external components or off an external clock requiring external components. Either the internal clock or the external clock can be selected as a mask option when the device code is released. The internal clock is used when economy is more important than absolute frequency accuracy. The external clock is used when the frequency accuracy requirements of the clock justify the added cost of the necessary external components, when the preset frequencies available with the internal clock are not applicable to the application, or when synchronization with other processors is important.

2.5.1 Internal Oscillator

The internal clock provides either 15.36 MHz or 19.2 MHz selectable by setting or clearing the SPEED bit in mode register 2. Both frequencies are trimmed during the manufacturing process. When using the internal clock, OSC IN should be tied to V_{SS}. The internal clock is disabled when either $\overline{\text{INIT}}$ is taken low or when a SETOFF instruction is executed. It is enabled when the $\overline{\text{INIT}}$ returns high or when a device wakeup occurs.

2.5.2 External Oscillator

The external oscillator provides the capability of providing either nonstandard or more accurate clock frequencies. Using the on-chip oscillator, either a ceramic resonator or a quartz crystal can be connected between OSC IN and the OSC OUT to provide a custom frequency. Alternatively, OSC IN can be driven with an externally derived clock signal. The on-chip oscillator circuit is disabled when the MSP50C3x is placed in any low-power mode. When the external oscillator mask option is selected, Port B1 is not available since an extra terminal is required to provide the oscillator output line (OSC OUT). See Figure 1–16 for a suggested external oscillator circuit.

2.5.3 External Clock

The external clock mask option allows an externally derived clock to be used by the device without loosing the Port B1 terminal. It is identical to the external oscillator except that terminal 7 is used for Port B1 and an oscillator return line is not provided.

2.5.4 Long Interval Monitor Timer – Timer-2 (MSP50x37 Only)

The MSP50x37 implements a set of timers in addition to timers incorporated in the CPU. Since this timer is designed for long interval time count, such as that required for external host processor communication, this timer does not initiate any interrupt. Table 2–9 shows the memory map of this timer.

Note: Long Interval Timer

The long interval timer does not decrement before it is loaded with the initial value from the upper 8-bit timer (FEEh).

Table 2–9. Memory Assignment for Additional Timer

RAM Address	Function
FECh	Prescaler (Write only)
FEDh	Lower 8-bit timer (Read/Write)
FEEh	Upper 8-bit timer (Read/Write)

To prevent count error, it is recommended to load the entire data when presetting the data.

Note: Transferring Data to the A Register

These additional timing registers are 8-bit registers. When transferring the data from these registers to the A register, the upper 8 bits of the A register are set to arbitrary values. The user should be careful so as to not depend upon the value of the upper 8 bits

Note: MSP50x37 Timer Registers and the Development Tools

The ANDCM, ORCM, TSTCM, SMAAN and AMAAC instructions do not work properly on these additional timing registers on the development tools. Due to the lack of development tool support, it is recommended that the only operations performed on these registers are transfers to and from memory using the TAM, TAMIX, TMA, TMAIX, and TBM instructions.

2.6 Analog-to-Digital Converter (MSP50x37 Only)

The MSP50C37 and the MSP50P37 incorporate an analog-to-digital converter (ADC). This circuit is disabled and draws no current when the the device is in SLEEP or STANDBY mode.

The 8 D-Port terminals are multiplexed to the input of the ADC. Writing a value of 0 through 7 to either RAM location 0xFFCh or 0xFFDh selects which of the D port terminals (D0 through D7) provides the ADC its input. Subsequent reads of either 0xFFCh or OxFFDh read the output of the ADC.

Reading either 0xFFCh or 0xFFDh initiates the next conversion. This conversion is available 40 instruction cycles later. Attempts to read the ADC sooner than that result in the ADC being reset and a new conversion being initiated without the results of the current conversion being either completed or written to the ADC buffer. The ADC does not convert continuously, only in response to reading the previous data. The converter output is 8 bits wide with an accuracy to 6 bits. A clock rate of 20 MHz results in a maximum conversion rate of 31.25K samples/second.

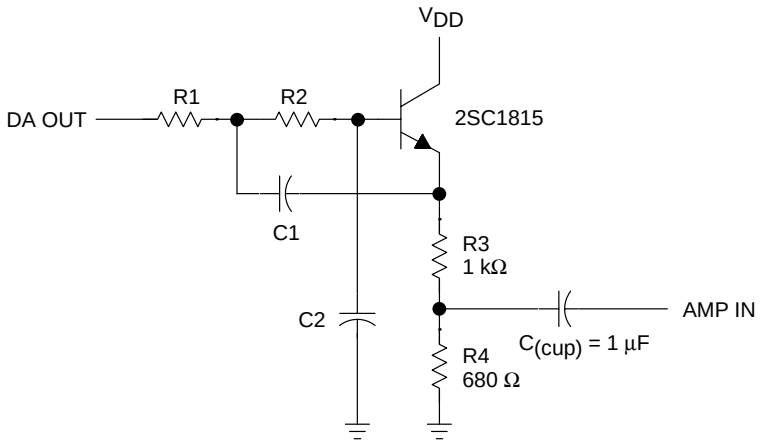
Table 2–10. D Port Terminal MUX of ADC Input for RAM

D Port Terminal Selected	RAM 0xFFCh or 0xFFDh Bits		
	2	**1**	**0**
PD0	0	0	0
PD1	0	0	1
PD2	0	1	0
PD3	0	1	1
PD4	1	0	0
PD5	1	0	1
PD6	1	1	0
PD7	1	1	1

2.7 Power Amplifier (MSP50x37 Only)

The MSP50x37 integrates a power amplifier that has either a one-pin analog option or a two-pin differential output. This amplifier can drive an 8-Ω speaker directly and has 0.5-W driving power (typical). The voltage gain of this amplifier is set to 9, the input signal should be kept within 1/2 V_{DD} ± 0.28 V to prevent exceeding 0.5 W. When set, the AMP_ENA bit (bit 0) in the power amplifier register (FEBh) activates the power amplifier. The contents of the power amplifier register are cleared upon a power up, reset, wake up, or when the $\overline{\text{INIT}}$ terminal is asserted low. See Figure 2–7 for an example of the application circuit.

Figure 2–7. Oversampling Output Filter Circuit



NOTE A: R1 = R2 = 6.8 kΩ
C1 = C2 = 6800 pF

$$f_O = \frac{1}{(2\ \pi\ RC)} = 3.4\ \text{kHz}$$

Output gain can be modified with R3 and/or R4.

Chapter 3

MSP50C3x Assembler

The MSP50C3x assembler chapter describes how to invoke the assembler, assembler command-line options, source-statement format, assembler symbols and characters, and assembler directives.

Topic **Page**

3.1 Description of Notation Used

The notation used in this document is as follows:

- An optional field is indicated by brackets; for example,

  ```
  [LABEL]
  ```

- User-supplied contents are indicated by braces; for example,

  ```
  <num>
  ```

- A reserved word is shown in capital letters.

- A required blank is indicated by a caret (^).

The following syntax example demonstrates the notational conventions used in this guide.

```
[<name>]^JMP^<number>^[<comment>]
```

3.2 Invoking the Assembler

The assembler is invoked from the DOS command line or from within the EMU software. The assembler is invoked from the DOS command line by typing:

```
ASMX^[<options>]^<source[.ext]>
```

where:

- Options represents a list of assembler options (see Section 3.3, *Command-Line Options*).
- Source is the name of the source file with the extension optional.
- If the extension is not given, then the default extension .asm is assumed. For example:

```
ASMX –L PROGRAM
```

runs the assembler using the source file program.asm and generates the output object file program.bin. No list file is generated.

The assembler is invoked within the EMU by selecting Assemble from the Assembler menu. A window appears allowing the user to select the file to assemble. Assembler options can be selected from the Options > Assembler Options menu.

3.3 Command-Line Options

Several options can be invoked from the command line (Table 3–1). They are invoked by listing their abbreviation prefixed by a minus sign. The following example:

```
ASMX -Lo PROGRAM.ASM
```

assembles the program in file program.asm but does not generate either a listing file or an object file; however, any errors are written to the console. The available options are detailed in Table 3–1. See subsection 3.9.10, *OPTION Directive*, for information on invoking options within the source code.

Table 3–1. Switches and Options

Character	Action
B or b	Lists only the first data byte in BYTE or RBYTE
D or d	Lists only the first data byte in DATA or RDATA
H, h, or ?	Displays usage information
I or i	Counts the number of times a valid instruction has been used
L or l	Displays error messages without generating a list
M or m	Expands macros in listing
O or o	Disables object file output
P or p	Prints listing without page breaks
R or r	Produces a reduced cross-reference list
S or s	Writes no errors on screen unless a listing file is generated
T or t	Lists only the first data byte in TEXT or RTEXT
V or v	Produces symbolic debugging information
W or w	Suppresses the warning message
X or x	Adds a cross-reference list at the end

3.3.1 BYTE Unlist Option

Placing a b or B in the command-line option field causes the assembler to list only the first opcode in a BYTE or RBYTE statement. Normally, if a BYTE or RBYTE statement has n arguments, they are listed in a column running down the page in the opcode column of the listing, taking n lines to completely list the resulting opcodes. When the BYTE unlist switch is set, then only the first line (which also contains the source line listing) is written to the listing file.

3.3.2 DATA Unlist Option

Placing a d or D in the command-line option field causes the assembler to list only the first opcode in a DATA or RDATA statement. Normally, if a DATA or RDATA statement has n arguments, they are listed in a column running down the page in the opcode column of the listing, taking n lines to completely list the resulting opcodes. When the DATA unlist switch is set, then only the first line (which also contains the source line listing) is written to the listing file.

3.3.3 XREF Unlist Option

Placing an x or X in the command-line option field causes the assembler to add a cross-reference listing at the end of the listing file (see Example 3–1). The cross reference listing condenses the output by printing multiple line numbers per line. The line that contains the symbol name shows where the symbol was defined. The second and following lines show where the symbol was used.

Example 3–1.XREF Unlist Option

```
Cross Reference                                          PAGE 0063
Symbol    Val    Filename          Line
--------  ----   ---------------   ------------------------------
ERR_CODE  0040   big.asm           276
                 big.asm           434 468 485 508 523 544 567 588 611 632
                 big.asm           653 674 695 716 738 765 788 809 832 853
```

3.3.4 TEXT Unlist Option

Placing a t or T in the command-line option field causes the assembler to list only the first opcode in a TEXT or RTEXT statement. Normally, if a TEXT or RTEXT statement has as an argument a string containing n characters, the ASCII representation of these n characters is written in a column in the opcode column of the program listing. When the TEXT unlist switch is set, then only the first line (which also contains the source line listing) is written to the listing file.

3.3.5 WARNING Unlist Option

Placing a w or W in the command-line option field causes the assembler to suppress WARNING messages. Warnings are still counted and error messages are still generated.

3.3.6 Complete XREF Switch

Placing an r or R in the command-line option field causes the assembler to produce a reduced XREF listing if one is produced. Normally, all symbols (whether

used or not) are listed in the XREF listing. The r option causes the assembler to omit from the XREF listing all symbols from the copy files that were never used.

3.3.7 Object Module Switch

Placing an o or O in the command-line option field causes the assembler to not generate any object output modules.

3.3.8 Listing File Switch

Placing an l or L in the command-line option field causes the assembler to not generate the listing file but to display any error messages to the screen.

3.3.9 Page-Eject Disable Switch

Placing a p or P in the command-line option field causes the assembler to print the listing in a continual manner without division into separate pages. When desired, a form feed can still be forced using the PAGE command.

3.3.10 Error-to-Screen Switch

Placing an s or S in the command-line option field causes the assembler to not write errors to the screen unless no listing file is being generated.

3.3.11 Instruction Count Switch

Placing an i or I in the command-line option field causes the assembler to generate a table containing the number of times each valid instruction is used in the program.

3.3.12 Show Usage Switch

Placing an h, H, or ? in the command line option field causes the assembler to display the proper DOS command line usage and a table of the command line switches that can be used.

3.3.13 Macro Switch

Placing an m or M in the command-line option field causes the assembler to expand macros in the listing file.

3.3.14 Symbolic Debugging Switch

Placing a v or V in the command-line option field causes the assembler to produce symbolic debugging information.

3.4 Assembler Input and Output Files

The assembler takes as input a file containing the assembler source and produces as output a listing file and an object file in binary format.

3.4.1 Assembler Source File

The assembly source file is specified in the command line. When the filename in the command line has an extension, then the file name is used as given. If no extension is specified, then the extension .asm is assumed.

For example:

```
ASMX PROGRAM.SRC
```

uses the file program.src as the assembly source file.

```
ASMX PROGRAM
```

uses the file program.asm as the assembly source file.

3.4.2 Assembler Binary Object File

The assembly process produces an object file in binary format. The object output is placed into a file with the same filename as the assembly source except that the extension is .bin. When the binary file is not desired, it can be disabled either as a command-line option or with an OPTION statement.

For example:

```
ASMX PROGRAM.SRC
```

uses the file program.src as the assembly source file and the file program.bin as the binary object output file.

```
ASMX -o PROGRAM.SRC
```

uses the file program.src as the assembly source file and produces no object output.

3.4.3 Assembler Listing File

The assembly process produces a listing file that contains the source instructions, the assembled code, and (optionally) a cross-reference table. The listing file is placed in a file with the same file name as the assembly source except that the extension is .lst.

For example:

```
ASMX PROGRAM.SRC
```

uses the file program.src as the assembly source file and the file program.lst as the assembly listing file.

3.5 Source Statement Format

An assembly-language source program consists of source statements contained in the assembly source file(s) that can contain assembler directives, machine instructions, or comments. Source statements can contain four ordered fields separated by one or more blanks. These fields (label, command, operand, and comment) are discussed in the following paragraphs.

The source statement can be as long as 80 characters. When the form width is set to 80 characters (the default), the assembler truncates the source line at 60 characters. The user should ensure that nothing other than comments extend past column 60.

Any source line starting with an asterisk (*) in the first character position or anything following a semicolon is treated as a comment in its entirety. It is ignored by the assembler and has no affect on the assembly process.

The syntax of the source statements is:

```
[<label>]^COMMAND^<operand>^;[<comment>]
```

A source statement can have an optional label that is defined by the user. One or more blanks separate the label from the COMMAND mnemonic. One or more blanks separate the mnemonic from the operand (if required by the command). One or more blanks separate the operand from the comment field. Comments are ignored by the assembler.

3.5.1 Label Field

The label field begins in character position one of the source line. When position one is a character other than a blank or an asterisk, the assembler assumes that the symbol is a label. When a label is omitted, then the first character position must be a blank. The label can contain up to ten characters consisting of alphabetic characters (a–z, A–Z), numbers (0–9), and some other characters (@,$,_). The first character should be an alphabetic character, and the remaining nine character positions can be any of the legal characters listed above.

3.5.2 Command Field

The command field begins after the blank that terminates the label field or in the first nonblank character past the first character position (which must be blank when the label is omitted). The command field is terminated by one or more blanks and can not extend past the sixtieth character position. The command field can contain either an assembler mnemonic (e.g. TAX) or an assem-

bler directive (e.g. OPTION). The assembler does not distinguish between capital and small letters in the command name; for example, TAX, Tax, and tAX are all identical names to the assembler.

3.5.3 Operand Field

The operand field begins following the blank that terminates the command field and can not extend past the sixtieth column position. The operand can contain one or more constants or expressions described in subsection 3.5.5, *Constants*, through subsection 3.5.5.5*, Assembly-Time Constants*. Terms in the operand field are separated by commas. The operand field is terminated by the first blank encountered.

3.5.4 Comment Field

The comment field begins either after the blank that terminates the operand field or after the blank that terminates the command field if no operand is required. The comment field must begin with a semicolon. The assembler generates errors if there are comments without the leading semicolon. The comment field can extend to the end of the source record and can contain any ASCII character including blanks.

3.5.5 Constants

The assembler recognizes the following five types of constants:

- Decimal integer constants
- Binary integer constants
- Hexadecimal integer constants
- Character constants
- Assembly-time constants

3.5.5.1 Decimal Integer Constants

A decimal integer constant is written as a string of decimal digits. The range of values of decimal integers is –32,768 to 65,535. Positive decimal integer constants greater than 32,767 are considered negative when interpreted as two's complement values.

The following are valid decimal constants:

- 1000 Constant equal 1000 or 03E8h
- –32768 Constant equal to –32,768 to 8000h
- 25 Constant equal to 25 or 0019h

3.5.5.2 Binary Integer Constants

A binary integer constant is written as a string of up to 16 binary digits (0/1) preceded by a question mark (?). When less than 16 digits are specified, the assembler right-justifies the given bits in the resulting constant.

The following are valid binary constants:

- ?0000000000010011 Constant equal 19 or 0013h
- ?0111111111111111 Constant equal to 32,767 to 7FFFh
- ?11110 Constant equal to 30 or 001Eh

3.5.5.3 Hexadecimal Integer Constants

A hexadecimal integer constant is written as a string of up to four hexadecimal digits preceded by a number sign (#) or a greater than sign (>). When less than four hexadecimal digits are specified, the assembler right-justifies the bits that are specified in the resulting constant. Hexadecimal digits include the decimal values 0 through 9 and the letters a (or A) through f (or F).

The following are valid hexadecimal constants:

- #07F Constant equal 127 or 07Fh
- >07f Constant equal 127 or 07Fh
- #307A Constant equal to 12,410 or 307Ah

3.5.5.4 Character Constants

A character constant is written as a string of one or two alphabetic characters enclosed in single quotes. A single quote can be represented within the character constant by two successive quotes. When less than two characters are specified, the assembler right-justifies the given bits in the resulting constant. The characters are represented internally as 8-bit ASCII characters. A character constant consisting of only two single quotes (no character) is valid and is assigned the value 0000h.

The following are valid character constants:

- ‘AB’ Constant equal 4142h
- ‘C’ Constant equal 0043h
- ‘”D’ Constant equal to 2744h

3.5.5.5 Assembly-Time Constants

An assembly-time constant is a symbol given a value by an EQU directive (see subsection 3.9.5, *EQU Directive*). The value of the symbol is determined at assembly time and can be assigned values with expressions using any of the constant types.

3.6 Symbols

Symbols are used in the label field and the operand field. A symbol is a string of ten or fewer alphanumeric characters (a–z, A–Z,0–9, and the characters @, _, and $). Upper-case and lower-case characters are not distinguished from one another; for example, A1 and a1 are treated identically by the assembler. No character can be blank. When more than ten characters are used in a symbol, the assembler prints all the characters but issues a warning message that the symbol has been truncated and uses only the first ten characters for processing.

Symbols used in the label field become symbolic addresses. They are associated with locations in the program and must not be used in the label field of other statements. Mnemonic operation codes and assembler directives can also be used as valid user-defined symbols when placed in the label field.

Symbols used in the operand field must be defined in the assembly, usually by appearing in the label field of a statement or in the operand field of an EQU directive.

The following are examples of valid symbols:

```
START
Start
Strt_1
```

Predefined Symbol $

The dollar sign ($) is a predefined symbol given the value of the current location within the program. It can be used in the operand field to indicate relative program offsets.

For example:

```
BR $+6
```

results in a branch to an address six bytes beyond the current location.

3.7 Character Strings

Several assembler directives require character strings in the operand field. A character string is written as a string of characters enclosed in single quotes. A quote can be represented in the string by two successive quotes. The maximum length of the string is defined for each directive that requires a character string. The characters are represented internally in 8-bit ASCII.

The following are valid character strings:

```
'SAMPLE PROGRAM'
'Plan 'C'''
```

3.8 Expressions

Expressions are used in the operand fields of assembler instructions and directives. An expression is a constant or symbol, a series of constants or symbols, or a series of constants and symbols separated by arithmetic operators.

Each constant or symbol can be preceded by a minus sign (unary minus) or a plus sign (unary plus). Unary minus is the same as taking the two's complement of the value. An expression must not contain embedded blanks. The valid range of values in an expression is –32,768 to 65,535. The value of all terms of an expression must be known at assembly time.

3.8.1 Arithmetic Operators in Expressions

The following arithmetic operators can be used in an expression:

- ~ inversion
- + addition
- – subtraction
- * multiplication
- / division (remainder is truncated)
- % modulo (remainder after division)
- & bitwise AND
- + + bitwise OR
- && bitwise EXCLUSIVE-OR

In evaluating an expression, the assembler first negates any constant or symbol preceded by a unary minus and then performs the arithmetic operations from left to right. The assembler does not assign arithmetic operation precedence to any operation other than unary plus and unary minus (so that the expression 4+4*2 is evaluated as 16, not 12).

3.8.2 Parentheses in Expressions

The assembler supports the use of parentheses in expressions to alter the order of evaluating the expression. Nesting parentheses within expressions is also supported. When parentheses are used, the portion of the expression within the innermost parentheses is evaluated first, and then the portion of the expression within the next innermost pair is evaluated. When evaluation of the portions of the expression within the parentheses has been completed, the evaluation is completed from left to right. Evaluation of portions of an expression within parentheses at the same nesting level is considered as simultaneous. Parenthetical expressions can not be nested more than eight deep.

3.9 Assembler Directives

Assembler directives (Table 3–2) are instructions that modify the assembler operation. They are invoked by placing the directive mnemonic in the command field and any modifying operands in the operand field. The valid directives are described in the following paragraphs.

Table 3–2. Summary of Assembler Directives

Mnemonic	Directive	Syntax
Directives That Affect the Location Counter		
AORG	Absolute origin	[<label>]^AORG^<expression>^[<comment>]
Directives That Affect Assembler Output		
IDT	Program identifier	[<label>]^IDT^'<string>'^[<comment>]
LIST	Restart source listing	[<label>]^LIST^[<comment>]
NARROW	80-column form width	[<label>]^NARROW^[<comment>]
OPTION	Output options	[<label>]^OPTION^<option list>^[<comment>]
PAGE	Page eject	[<label>]^PAGE^[<comment>]
TABSIZE	Tab stops	[<label>]^TABSIZE<expression>^[<comment>]
TITL	Page title	[<label>]^TITL^<string>^[<comment>]
UNL	Stop source listing	[<label>]^UNL^[<comment>]
WIDE	130-column form width	[<label>]^WIDE^[<comment>]
Directives That Initialize Constants		
BYTE	Initialize byte	[<label>]^BYTE^<expr-1>[,<expr-2>,....., <expr-n>]^[<comment>]
RBYTE	Reverse bit initialization of byte	[<label>]^RBYTE^<expr-1>[,<expr-2>,....., <expr-n>]^[<comment>]
DATA	Initialize word	[<label>]^DATA^<expr-1>[,<expr-2>,....., <expr-n>]^[<comment>]
RDATA	Reverse bit initialization of word	[<label>]^RDATA^<expr-1>[,<expr-2>,....., <expr-n>]^[<comment>]
EQU	Define assembly time constant	[<label>]^EQU^<expression>^[<comment>]
TEXT	Initialize text	[<label>]^TEXT^[–]'<string>'^[<comment>]
RTEXT	Reverse byte initialization of text	[<label>]^RTEXT^[–]'<string>'^[<comment>]
Miscellaneous Directives		
COPY	Copy source file	[<label>]^COPY^'<filename>'^[<comment>]
END	Program end	[<label>]^END^[<comment>]
MACRO/ ENDM	Macro definition	MACRO^'<name>' <statements> ENDM

3.9.1 AORG Directive

The AORG directive places the value found in the expression in the operand field into the location counter. Subsequent instructions have addresses starting at this value. The use of the label field is optional, but when a label is used, it is assigned the value found in the operand field.

The syntax of the AORG directive is as follows:

[<label>]^**AORG**^<expression>^[<comment>]

In the following statement:

```
AORG    #1000+offset
```

if the offset has a value of 8, AORG sets the location counter to #1008. If a label is included, it also is assigned the value of #1008. The symbol offset must be previously defined.

3.9.2 BYTE Directive

The BYTE directive places the value of one or more expressions into successive bytes of program memory. The range of each term is 0 to 255. The command field contains BYTE. The operand field contains a series of one or more terms separated by commas and terminated by a blank that represents the values to be placed in the successive bytes of program memory.

The syntax of the BYTE directive is as follows:

[<label>]^**BYTE**^<expr_1>[,<expr_2>,...,<expr_n>]^[<comment>]

In the following statement:

```
BYTE    #E0,5,data+5
```

places the numbers 224, 5, and the result of the arithmetic operation data + 5 into the next three bytes of program memory. The value of the symbol data must be defined in the assembly process.

3.9.3 COPY Directive

The COPY directive causes the assembler to read source statements from a different file. The assembler gets subsequent statements from the copy file until either the end-of-file marker is found or an END directive is found in the copy file. A copy file cannot contain another COPY directive. The command field contains COPY. The operand field contains the name of the file from which the source files are to be read.

The syntax of the COPY directive is as follows:

[<label>]^**COPY**^'<filename>'^[<comment>]

The directive in the following example:

```
COPY  'copy.fil'
```

causes the assembler to take its source statements from a file called copy.fil. At the end-of-file for copy.fil or when an END directive is encountered in copy.fil, the assembler resumes processing source statements from the original source file. The single quotes around the filename are required by the assembler.

3.9.4 DATA Directive

The DATA directive places the value of one or more expressions into successive words of program memory. The range of each term is 0 to 65,535. The command field contains DATA. The operand field contains a series of one or more terms separated by commas and terminated by a blank that represents the values to be placed in the successive words of program memory.

The syntax of the DATA directive is as follows:

[<label>]^**DATA**^<expr_1>[,<expr_2>,...,<expr_n>]^[<comment>]

In the following statement:

```
DATA  #E000,'AB'
```

places the following bytes into successive locations in program memory: E0h, 00h, 41h, 42h

3.9.5 EQU Directive

The EQU directive assigns a value to a symbol. The label field contains the name of the symbol to which a value is assigned. The command field contains EQU. The operand field contains the value to be assigned to the symbol.

The syntax of the EQU directive is as follows:

[<label>]^**EQU**^<expression>^[<comment>]

In the following statement:

```
Offset  EQU  #100
```

assigns the numeric value of 256 (100h) to the symbol Offset.

3.9.6 END Directive

The END directive signals the end of the source or copy file. It is treated by the program as an end-of-file marker. When it is found in a copy file, the copy file

is closed and subsequent statements are taken from the source file. When it is found in the source file, the assembly process terminates at that point in the file.

The syntax of the END directive is as follows:

[<label>]^**END**^<comment>]

In the following statement:

```
ACAAC       1
END
CLA
```

the ACAAC 1 instruction is assembled, but the CLA and any subsequent instructions are ignored.

3.9.7 IDT Directive

The IDT assigns a name to the object module produced. Use of the label field is optional. When used, a label assumes the current value of the location counter. The command field contains IDT. The operand field contains the module name <string>, a character string of up to eight characters within single quotes. When a character string of more than eight characters is entered, the assembler prints a truncation warning message and retains the first eight characters as the program name.

The syntax of the IDT directive is as follows:

[<label>]^**IDT**^<string>^[<comment>]

In the following example:

```
     AORG   20
L1   IDT   'Example'
```

assigns the value of 20 to the symbol L1 and assigns the name 'Example' to the module being assembled. The module name is then printed in the source listing as the operand of the IDT directive and appears in the page heading of the source listing.

3.9.8 LIST Directive

The LIST directive restores printing of the source listing. This directive is required only when a no-source-listing (UNL) directive is in effect and causes the assembler to resume listing. This directive is not printed in the source listing, but the line counter increments.

The syntax of the LIST directive is as follows:

[<label>]^**LIST**^[<comment>]

In the following statement:

```
    AORG  10
T1  LIST       ;Turn on source listing
```

the label T1 is assigned the value 10 and listing is resumed. The line is not printed out so that although the label T1 is entered into the symbol table and appears in the cross-reference listing, the line in which it is assigned a value does not appear in the listing file.

3.9.9 NARROW Directive

The NARROW directive causes the assembler to assume an 80-column form width for the listing file. The default is 80 columns (See subsection 3.9.18, *Wide Directive*).

The syntax of the NARROW directive is as follows:

[<label>]^**NARROW**^[<comment>]

The following example uses the NARROW directive:

```
    AORG 10
T1  NARROW        ;Switch to 80–column listing format.
```

3.9.10 OPTION Directive

The OPTION directive selects several options that affect assembler operation. The <option list> operand is a list of keywords separated by commas; each keyword selects an assembly feature. Only the first character of the keyword is significant. Use of the label field is optional. When used, the label assumes the current value of the location counter.

The syntax of the OPTION directive is as follows:

[<label>]^**OPTION**^<option–list>^[<comment>]

The following are examples of the OPTION directive:

```
   OPTION      XREF,SCRNOF
   OPTION      X,S
```

The two examples above have identical effects. The cross-reference listing is produced and the error messages are not sent to the screen (unless no source listing file is being produced). See Section 3.3, *Command-Line Options*, for information on invoking options from the command line.

The available options are listed in the following paragraphs.

3.9.10.1 BUNLST – Byte Unlist Option

Placing any valid symbol starting with B or b in the option list enables the byte unlist option. This option limits the listing of BYTE or RBYTE directives to one line. Normally, if a BYTE or RBYTE directive has more than one operand, the resulting object code is listed in a column in the opcode column of the source listing. BUNLST is used to avoid this.

3.9.10.2 DUNLST – Data Unlist Option

Placing any valid symbol starting with D or d in the option list enables the data unlist option. This option limits the listing of DATA or RDATA directives to one line. Normally, if a DATA or RDATA directive has more than one operand, the resulting object code is listed in a column in the opcode column of the source listing. DUNLST is used to avoid this.

3.9.10.3 FUNLST – Byte, Data and Text Unlist Option

Placing any valid symbol starting with F or f in the option limits the listing of BYTE, RBYTE, DATA, RDATA, TEXT, or RTEXT directives to one line. In effect, it is equivalent to calling the DUNLST, BUNLST, and TUNLST directives all at the same time.

3.9.10.4 LSTUNL – Listing Unlist Option

Placing any valid symbol starting with L or l in the option list inhibits the listing file from being produced. It takes precedence over the LIST directive.

3.9.10.5 OBJUNL – Object File Unlist Option

Placing any valid symbol starting with O or o in the option enables the object file unlist option. This option inhibits the generation of an object file.

3.9.10.6 PAGEOF – Page Break Inhibit Option

Placing any valid symbol starting with P or p in the option enables the page break inhibit option. This option causes the listing file to be printed in a continuous stream without page breaks.

3.9.10.7 RXREF – Reduced XREF Option

Placing any valid symbol starting with R or r in the option enables the reduced XREF option. This option causes symbols that were found in copy files but never used to be omitted from the cross-reference listing (if produced).

3.9.10.8 SCRNOF – Screen Error Message Unlist Option

Placing any valid symbol starting with S or s in the option enables the screen error message unlist option. This option causes the error messages to not be listed on the screen unless the listing file is not being produced.

3.9.10.9 TUNLST – Text Unlist Option

Placing any valid symbol starting with T or t in the option list enables the text unlist option. This option limits the listing of TEXT or RTEXT directives to one line. A TEXT or RTEXT directive normally takes as many lines to list as there are characters in the operand. TUNLST causes only the first line of the directive listing to be produced.

3.9.10.10 WARNOFF – Warning Message Unlist Option

Placing any valid symbol starting with W or w in the option list inhibits the listing of warning diagnostics. Warnings are still counted and the total is still printed at the end of the source listing.

3.9.10.11 XREF – Cross-Reference Listing Enable

Placing any valid symbol starting with X or x in the option list causes a cross-reference listing to be produced at the end of the source listing. When used, it should be placed at the start of the program.

3.9.11 PAGE Directive

The PAGE directive forces the assembler to continue the source program listing on a new page. The PAGE directive is not printed in the source listing, but the line counter increments. Use of the label field is optional. When used, a label assumes the current value of the location counter. The command field contains PAGE. The operand field is not used.

The syntax of the PAGE directive is as follows:

[<label>]^**PAGE**^[<comment>]

In the following statement:

```
    AORG 10
T1  PAGE        ;Force page eject
```

the label T1 is assigned the value 10 and the listing is resumed at the top of the next page. The line is not printed out so that although the label T1 is entered into the symbol table and appears in the cross-reference listing, the line in which it is assigned a value does not appear in the listing file.

3.9.12 RBYTE Directive

The RBYTE directive places the value of one or more expressions into successive bytes of program memory in a bit-reversed form. The range of each term

is 0 to 255. The command field contains RBYTE. The operand field contains a series of one or more terms separated by commas and terminated by a blank that represents the values to be placed in the successive bytes of program memory.

The syntax of the RBYTE directive is as follows:

[<label>]^**RBYTE**^<expr_1>[,<expr_2>,...,<expr_n>]^[<comment>]

In the following statement:

```
RBYTE #E0,5,data+5
```

places the numbers 7 (07h), 160 (A0h), and the bit reversed result of the arithmetic operation data+5 into the next three bytes of program memory. The value of the symbol data must be defined in the assembly process.

3.9.13 RDATA Directive

The RDATA directive places the value of one or more expressions into successive words of program memory in a bit-reversed form. The range of each term is 0 to 65,535. The command field contains RDATA. The operand field contains a series of one or more terms separated by commas and terminated by a blank that represents the values to be placed in the successive words of program memory.

The syntax of the RDATA directive is as follows:

[<label>]^**RDATA**^<expr_1>[,<expr_2>,...,<expr_n>]^[<comment>]

In the following statement:

```
RDATA #E000,'AB'
```

places the following bytes into successive locations in program memory: 00h, 07h, 42h, 82h

3.9.14 RTEXT Directive

The RTEXT directive writes an ASCII string to the object file in reverse order. When the string is preceded by a minus sign, the last character in the string to be written (which is the first character of the string as given) is written with its most significant bit set to 1. The use of the label field is optional. When used, the label assumes the current value of the location counter. The command field contains RTEXT. The operand field contains a character string of up to 52 characters long enclosed in single quotes (optionally preceded by a minus sign).

The syntax of the RTEXT directive is as follows:

[<label>]^**RTEXT**^[–]'<string>'^[<comment>]

In the following examples:

```
RTEXT –'This is a test'
RTEXT 'This is a test'
```

both write the string "tset a si sihT" to the output file. The first example writes the first T in the word "This", which is the last character to be written with its most significant bit set to 1 (that is, as a D4h instead of a 54h).

3.9.15 TEXT Directive

The TEXT directive writes an ASCII string to the object file. When the string is preceded by a minus sign, the last character in the string is written with its most significant bit set to 1. The use of the label field is optional. When used, the label assumes the current value of the location counter. The command field contains TEXT. The operand field contains a character string of up to 52 characters long enclosed in single quotes (optionally preceded by a minus sign).

The syntax of the TEXT directive is as follows:

[<label>]^**TEXT**^[–]'<string>'^[<comment>]

In the following examples:

```
RTEXT –'This is a test'
RTEXT 'This is a test'
```

both write the string This is a test to the output file. The first example writes the final 't' in the word "test" with its most significant bit set to 1 (that is, as a F4h instead of a 74h).

3.9.16 TITL Directive

The TITL directive inserts a title to be printed in the heading of each page of the source listing. When a title is desired in the heading of the listing page, a TITL directive must be the first source statement submitted to the assembler. Unlike the IDT directive, the TITL directive is not printed in the source listing. The assembler does not print the comment because the TITL directive is not printed, but the line counter does increment. Use of the label field is optional. When used, a label field assumes the current value of the location counter. The command field contains TITL. The operand field contains the title (string)–a character string of up to 50 characters in length enclosed in single quotes. When more than 50 characters are entered, the assembler retains the first

50 characters as the title and prints a syntax error message. The comment field is optional.

The syntax of the TITL directive is as follows:

[<label>]^**TITL**^[–]'<string>'^[<comment>]

In the following example:

```
    TITL   'Sample Program'      This is a sample line
```

causes the title, Sample Program, to be printed in the page heading of the source listing. When a TITL directive is the first source statement in a program, the title is printed on all pages until another TITL directive is processed. Otherwise, the title is printed on the page after the directive is processed and on subsequent pages until another TITL directive is processed. None of this line is printed to the listing file.

3.9.17 UNL Directive

The UNL directive inhibits printing of the source listing output until the occurrence of a LIST directive. It is not printed in the source listing, but the source line counter is incremented. The label field assumes the value of the location counter. The command field contains the symbol UNL. The operand field is not used.

The syntax of the UNL directive is as follows:

[<label>]^**UNL**^[<comment>]

In the following statement:

```
    AORG 10
T1  UNL          Turn off source listing
```

the label T1 is assigned the value 10 and listing is inhibited.

3.9.18 WIDE Directive

The WIDE directive causes the assembler to assume an 130-column form width for the listing file. The default is 80 columns (See subsection 3.9.9, *NARROW Directive*).

The syntax of the WIDE directive is as follows:

[<label>]^**WIDE**^[<comment>]

The following example uses the WIDE directive:

```
    AORG 10
T1  WIDE          Switch to 130–column listing format.
```

3.9.19 MACRO/ENDM Directive

The MACRO directive allows macros to be defined. Macro expansion allows for 10 parameters to be passed to the macro. To reference a parameter, use a leading dollar sign $ followed by a single decimal digit. To include a dollar sign in the expansion, use two dollar signs. Nested macro definitions are not supported.

The syntax of the MACRO directive is as follows:

```
    MACRO ‘name’
    <statements>
    ENDM
```

The following macro FWD jumps forward from the current location counter.

```
    MACRO ‘FWD’
    SBR  ($$+$0)
    ENDM
```

To execute the macro and cause a jump forward by 10 bytes, type

```
    FWD  10
```

All arguments passed to a macro are resolved before expansion. To prevent a symbol from being expanded, enclose the symbol in single quotes. The symbol in quotesmust be at least 3 characters in length or incorrect results may be obtained. Symbols less than 3 characters in length can be padded with spaces to obtain the required 3 character minimum length.

Another example:

```
TEN  EQU  10
     MACRO  ‘JMP’
L$0  SBR  L$1

     ENDM
```

Using the following calls,

```
    JMP  TEN,’TEN’
    JMP  ‘TEN’,TEN
```

expands to:

```
L10   SBR  LTEN
LTEN  SBR  L10
```

3.9.20 TABSIZE Directive

The TABSIZE directive sets the tab stops used when expanding source lines in the listing file. The directive sets the tab stops to the value of the expression. Valid tab sizes are 2 through 8. Multiple TABSIZE directives can appear in the source. Each entry changes the tab stops of the TABSIZE line and all subsequent lines. The default tab size is 4.

The following example:

```
TABSIZE     3
```

sets the tab stops in columns 4, 7, 10, etc.

3.10 Listing Formats

Column	Description
1 – 2	Reserved for the copy file number. When the source is coming from the main source file, then no entry is added; otherwise, the two-digit hexadecimal file number appears. There is a table printed after the source listing that contains copy file numbers crossreferenced to copy file references.
3	Always blank
4 – 7	Reserved for the line number of the source line.
8	Always blank
9 – 12	Contain the hexadecimal representation of the location counter.
13	Always blank
14 – 17	Contain the hexadecimal representation of the assembled statement.
18	Always blank
19 – last column	Contain source statement

Example:

The following source line appeared on line 217 in the main source file. The assembled code 4070 is placed at addresses 0301 and 0302 in the output file.

```
0217 0301 4070 BR   GO       ;branch to start of program
```

Message lines are indicated in the source listing by two '#' characters in columns 1 and 2. The letter in column 4 indicates the type of message: E for an error, F for a fatal error, and W for a warning.

Example:

E: Attempt to reference undefined symbol "INITKB".

3.11 Placing Binary Data Above #FFFF

The assembler allows data to be placed above #FFFF. This is for supporting the MSP50C34 64K-byte part. The excitation table resides at starting address #10000. To start placing data at #10000 do the following:

```
    AORG #FFFF
    BYTE #00   ;This is a DUMMY byte for the Assembler to act as AORG #10000
ETABLE  BYTE #24
         . . . .
```

Please be aware that if any labels are placed above #FFFF, their value is truncated to the low-order word. A warning is issued when data crosses the #FFFF boundary.

Note: AORG# 10000 Statement Restriction

This restriction means the byte at location #FFFF cannot be used by the user's program. Also, **do not** use the AORG #10000 statement. Unpredictable results can occur.

Chapter 4

MSP50x3x Instruction Set

This chapter describes the 61 different MSP50C3x instructions (Table 4–1 and Table 4–2). Each instruction takes either one or two instruction cycles to execute. Each instruction cycle consists of 16 clock cycles; therefore, a clock speed of 19.2 MHz translates to 1,200,000 instructions per second. When one synthesis channel is enabled for LPC, approximately one out of every four instruction cycles is taken for synthesis calculations. This causes the instruction cycle rate for the program to drop to approximately 900,000 cycles per second. When both synthesis channels are enabled for LPC, approximately two out of every four instruction cycles are taken for synthesis calculations. The program instruction-cycle rate for this case is approximately 600,000 cycles per second.

Topic **Page**

4.1 Instruction Syntax

The syntax for the source code instructions is:

```
[<label>]^<opcode mnemonic>^[<operand>]^ [;<comment>]
```

The fields are:

- A 10-character optional label field
- A 6-character opcode mnemonic field
- An opcode-dependent operand field
- An optional comment field

Each of the fields is separated by one or more tabs or spaces.

4.2 MSP50x3x Assembly Instructions

The following section contains descriptions, opcodes, source code (syntax), object code, execution results, status flag information, and examples for the assembly instructions used to program the MSP50x3x family. Table 4–1 lists the assembly instructions in alphabetical order with operand size in bits, instruction cycles required, status conditions, number of bytes required, opcode, and a brief description.

Table 4–1. MSP50x3x Instruction Set

Mnemonic	Operand Size (Bits)	Instruction Cycles Required	Status (1 Always Set, C Conditional, N/A Does Not Apply)	Number of Bytes Required	Opcode (Hex)	Description
ABAAC		1	C	1	2C	Add B register to A register
ACAAC	12	2	C	2	70	Add constant to A register
AGEC	8	2	C	2	63	A greater than or equal to constant
AMAAC		1	C	1	28	Add memory to A register
ANDCM	8	2	1	2	65	AND constant and memory
ANEC	8	2	C	2	60	A register not equal to constant
AXCA	8	2	1	2	68	A register times constant
AXMA		1	1	1	39	A register times memory
AXTM		1	1	1	38	A register times timer
BR	13	2	1	2	40	Branch if status set
BRA		1	1	1	1F	Branch always to address in A register
CALL	12	2	1	2	00	Call if status set
CLA		1	1	1	2F	Clear A register
CLB		1	1	1	24	Clear B register
CLX		1	1	1	20	Clear X register
DECMN		1	C	1	27	Decrement memory
DECXN		1	C	1	22	Decrement X register
EXTSG		1	1	1	3C	Extended-sign mode
GET	3	2	C	1	30	Get bits

Table 4–1. MSP50x3x Instruction Set (Continued)

Mnemonic	Operand Size (Bits)	Instruction Cycles Required	Status (1 Always Set, C Conditional, N/A Does Not Apply)	Number of Bytes Required	Opcode (Hex)	Description
IAC		1	C	1	3A	Increment A register
IBC		1	C	1	25	Increment B register
INCMC		1	C	1	26	Increment memory
INTGR		1	1	1	3B	Set integer mode
IXC		1	C	1	21	Increment X register
LUAA		2	1	1	6B	Look up A register, result to A register
LUAB		2	1	1	6D	Look up A register, result to B register
LUAPS		2	1	1	6C	Start parallel-to-serial transfer
ORCM	8	2	1	2	64	OR constant with memory
RETI		1	C	1	3E	Return from interrupt
RETN		1	1	1	3D	Return from subroutine
SALA		1	C	1	2E	Shift A register left
SALA4		1	1	1	1B	Shift A register left 4 bits
SARA		1	1	1	15	Shift A register right
SBAAN		1	C	1	2D	Subtract B register from A register
SBR	7	1	1	1	80	Short branch if status set
SETOFF		1	N/A	1	3F	Turn processor off
SMAAN		1	C	1	29	Subtract memory from A register
TAB		1	1	1	1A	Transfer A register to B register
TAM		1	1	1	16	Transfer A register to memory
TAMD	8	2	1	2	6A	Transfer A register to memory direct
TAMIX		1	1	1	13	Transfer A register to memory, increment X register
TAMODE		1	1	1	1D	Transfer A register to mode register
TAPSC		1	1	1	19	Transfer A register to prescale register
TASYN		1	1	1	1C	Transfer A register to synthesizer register

Table 4–1. MSP50x3x Instruction Set (Continued)

Mnemonic	Operand Size (Bits)	Instruction Cycles Required	Status (1 Always Set, C Conditional, N/A Does Not Apply)	Number of Bytes Required	Opcode (Hex)	Description
TATM		1	1	1	1E	Transfer A register to timer register
TAX		1	1	1	18	Transfer A register to X register
TBM		1	1	1	2A	Transfer B register to memory
TCA	8	2	1	2	6E	Transfer constant to A register
TCX	8	2	1	2	62	Transfer constant to X register
TMA		1	1	1	11	Transfer memory to A register
TMAD	8	2	1	2	69	Transfer memory to A register direct
TMAIX		1	1	1	14	Transfer memory to A register, increment X register
TMXD	8	2	1	2	6F	Transfer memory direct to X register
TRNDA		1	1	1	2B	Transfer random number to A register
TSTCA	8	2	C	2	67	Test constant and A register
TSTCM	8	2	C	2	66	Test constant and memory
TTMA		1	1	1	17	Transfer timer to A register
TXA		1	1	1	10	Transfer X register to A register
XBA		1	1	1	12	Exchange A register and B register
XBX		1	1	1	23	Exchange B register and X register
XGEC	8	2	C	2	61	X register greater than or equal to constant

Table 4–2 lists the instructions by opcode.

Table 4–2. MSP50x3x Instruction Table

	MSB								
LSB	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8-F**
0	CALL	TXA	CLX	GET 1	BR	BR	ANEC	ACAAC	SBR
1	CALL	TMA	IXC	GET 2	BR	BR	XGEC	ACAAC	SBR
2	CALL	XBA	DECXN	GET 3	BR	BR	TCX	ACAAC	SBR
3	CALL	TAMIX	XBX	GET 4	BR	BR	AGEC	ACAAC	SBR
4	CALL	TMAIX	CLB	GET 5	BR	BR	ORCM	ACAAC	SBR
5	CALL	SARA	IBC	GET 6	BR	BR	ANDCM	ACAAC	SBR
6	CALL	TAM	INCMC	GET 7	BR	BR	TSTCM	ACAAC	SBR
7	CALL	TTMA	DECMN	GET 8	BR	BR	TSTCA	ACAAC	SBR
8	CALL	TAX	AMAAC	AXTM	BR	BR	AXCA	ACAAC	SBR
9	CALL	TAPSC	SMAAN	AXMA	BR	BR	TMAD	ACAAC	SBR
A	CALL	TAB	TBM	IAC	BR	BR	TAMD	ACAAC	SBR
B	CALL	SALA4	TRNDA	INTGR	BR	BR	LUAA	ACAAC	SBR
C	CALL	TASYN	ABAAC	EXTSG	BR	BR	LUAPS	ACAAC	SBR
D	CALL	TAMODE	SBAAN	RETN	BR	BR	LUAB	ACAAC	SBR
E	CALL	TATM	SALA	RETI	BR	BR	TCA	ACAAC	SBR
F	CALL	BRA	CLA	SETOFF	BR	BR	TMXD	ACAAC	SBR

The remainder of this section describes each instruction in detail.

Description ABAAC – Add B Register to A Register

Action Adds the contents of the B register to the contents of the A register and stores the result in the A register.

Opcode 2C

Syntax [<label>]^**ABAAC**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	1	0	0

Execution A + B $\rightarrow$ A

Status Flag 1 if there is a carry into bit 8 of the ALU; 0 if not.

Note:

The addition is performed independent of the arithmetic mode (EXTSG or INTGR) as an unsigned addition of all 16 bits of the B register and A register.

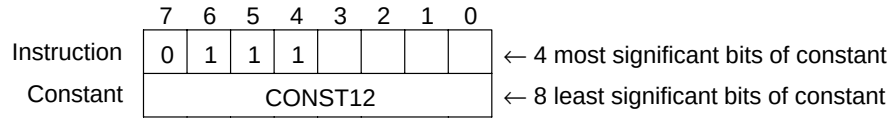
Description ACAAC – Add Constant to A Register

Action Adds the 12-bit constant specified by the operand to the contents of the A register and stores the result in the A register.

Opcode 70 – 7F

Syntax [<label>]^**ACAAC**^<CONST12>^...[<comment>]

Object Code



Execution A + CONST12 → A

Status Flag 1 if there is a carry into bit 8 of the ALU; 0 if not.

Note:

The results of the addition are dependent on the arithmetic mode. When the processor is in integer mode (INTGR), then the addition is of a 12-bit unsigned number with a 16-bit unsigned number. When the processor is in extended-sign mode (EXTSG), then the 12-bit constant is sign extended to a 16-bit two's complement number prior to the addition.

This instruction is useful when a table index has been placed in the A register. The base address of the table can be added to the index with this instruction, and a look-up can be completed to fetch the desired table element.

Example

```
      TMAD   INDEX     ;Bring table index in from memory
      ACAAC  TABLE     ;Add address of start of table
      LUAA             ;Bring table element into A register
TABLE
```

Description AGEC – A Register Greater Than or Equal to Constant

Action Compares the contents of the lower 8 bits of the A register and the 8-bit constant specified in the operand. AGEC sets the status flag if the contents of the lower 8 bits of the A register are greater than or equal to the operand.

Opcode 63

Syntax [<label>]^**AGEC**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	0	1	1
Constant	CONST8							

Execution A ≥ CONST8 → SF

Status Flag 1 if the lower 8 bits of the A register are greater than or equal to the 8-bit constant; 0 if not.

Note:

Comparison is always done on an unsigned basis (i.e., 0FFh is greater than 0FEh). Only the lower 8 bits of the A register are compared to the 8-bit constant value. The upper 8 bits of the A register are not considered, so the result is independent of the arithmetic mode (EXTSG or INTGR).

Example

```
      CLA             ;Prepare A register
LOOP  IAC             ;Increment A register
      AGEC   TEST     ;Is A reg greater than TEST
      SBR    TARGET   ;Yes, escape loop
      SBR    LOOP     ;No, continue loop
TARGET
```

Description AMAAC – Add Memory to A Register

Action Adds the contents of the RAM addressed by the X register to the A register and stores the result in the A register.

Opcode 28

Syntax [<label>]^**AMAAC**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	0	0	0

Execution A + *X $\rightarrow$ A

Status Flag 1 if there is a carry into bit 8 of the ALU; 0 if not.

Note:

When the most significant bit of the memory being used is set, the addition results are dependent on the arithmetic mode (EXTSG or INTGR). A carry into bit 8 sets the status flag in all cases.

This instruction may be used when the sum of two variables is desired.

Example

```
TMAD   VALUE1    ;Fetch value from memory
TCX    VALUE2    ;Point to second value
AMAAC            ;Add two values
TAMD   VALUE3    ;Store sum in memory
```

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EMU. While it does work on the actual part, it is not recommended that this be done due to the lack of debugging support.

Description ANDCM – Logical AND a Constant With Memory

Action Bit-wise performs an AND operation the contents of the memory addressed by the X register and an 8-bit constant and stores the results in the memory location addressed by the X register.

Opcode 65

Syntax [<label>]^**ANDCM**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	1	0	1
Constant	CONST8							

Execution *X && CONST8 → *X

Status Flag Always set to 1

Note:

The operation is performed independent of the arithmetic mode (EXTSG or INTGR) on the lower 8 bits of the RAM location; any other bits are unaffected.

Example

```
TCX    FLAGS     ;Point to FLAGS
ANDCM  #F0       ;Reset lower 4 bits to zero
```

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EMU. While it does work on the actual part, it is not recommended that this be done due to the lack of debugging support.

Description ANEC – A Register Not Equal to Constant

Action Compares the lower 8 bits of the A register to the constant specified by the operand and sets the status flag if they are not equal.

Opcode 60

Syntax [<label>]^**ANEC**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	0	0	0
Constant	CONST8							

Execution A ≠ CONST8 → SF

Status Flag 1 if the lower 8 bits of the A register are not equal to the 8-bit operand; 0 if they are equal.

> **Note:**
>
> Only the lower 8 bits of the A register are compared to the 8-bit constant value. This instruction is independent of the arithmetic mode (EXTSG or INTGR).

Example

```
      CLA             ;Prepare A register
LOOP  IAC             ;Increment A register
      ANEC  TEST      ;Is A register equal to TEST
      SBR   LOOP      ;No, continue loop
      SBR   TARGET    ;Yes, escape loop
TARGET
```

Description AXCA – A Register Times Constant

Action Multiplies the contents of the A register and the operand and leaves the results (right shifted 7 bits) in the A register.

Opcode 68

Syntax [<label>]^**AXCA**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	0	0	0
Constant	CONST8							

Execution $(A \times CONST8)/128 \rightarrow A$

Status Flag Always set to 1

Notes:

1) The operation is performed independent of the arithmetic mode (EXTSG or INTGR) as a 2's complement multiplication of all 16 bits of the A register and the 8-bit constant. The result is right shifted 7 bits so that the most significant 16 bits of the 23-bit result are available for further use.

   This operation can be used to perform a right shift on the value in the A register (see example below). Multiplying by 1 gives a right shift of 7 bits, multiplying by 2 gives a right shift of 6 bits, multiplying by 4 gives a right shift of 5 bits, multiplying by 8 gives a right shift of 4 bits, etc.

2) When the A register contains the value 8000h, the results of the AXCA instruction are not reliable.

Example

```
TCA   #80   ;Load first value
AXCA  #4    ;Multiply by second value
            ;(result is #04)
```

Description AXMA – A Register Times Memory

Action Multiplies the contents of the A register and the lower 8 bits of the contents of the memory location addressed by the X register; leaves the results (right shifted by 7 bits) in the A register.

Opcode 39

Syntax [<label>]^**AXMA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	0	0	1

Execution $(A \times *X)/128 \rightarrow A$

Status Flag Always set to 1

Notes:

1) The operation is performed independent of the arithmetic mode (EXTSG or INTGR) as a two's complement multiplication of all 16 bits of the A register and the 8-bit value fetched from memory. The result is right shifted 7 bits so that the most significant 16 bits of the 23-bit result are available for further use.

2) When the A register contains the value 8000h, the results of the AXMA instruction are not reliable.

Example

```
TCA    #3F       ;Load first value
TCX    RAMLOC    ;Point to memory
TAM              ;Store value in RAM
TCA    #1F       ;Load second value
AXMA             ;Multiply first value by second
                 ;value (result is #0F)
```

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EMU. While it does work on the actual part, it is not recommended that this be done due to the lack of debugging support.

Description AXTM – A Register Times Timer

Action Multiplies the contents of the A register and the contents of the timer register and stores the results (right shifted by 7 bits) in the A register.

Opcode 38

Syntax [<label>]^**AXTM**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	0	0	0

Execution $(A \times TM)/128 \rightarrow A$

Status Flag Always set to 1

Notes:

1) The operation is performed independent of the arithmetic mode (EXTSG/INTGR) as a two's complement multiplication of all 16 bits of the A register and the 8-bit value of the timer register. The result is right shifted 7 bits so that the most significant 16 bits of the 23-bit result are available for further use.

2) When the A register contains the value 8000h, the results of the AXTM instruction are not reliable.

Example

```
TCA   #3F   ;Load first value
TATM        ;Store first value in timer register
TCA   #1F   ;Load second value
AXTM        ;Multiply first value by second value
            ;(result is #0F if timer has
            ;not decremented)
```

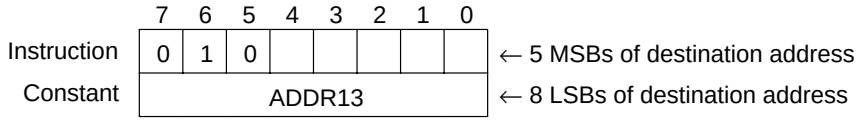
Description BR – Branch If Status Set

Action If the status flag is set to 1, the bottom 13 bits of the program counter are replaced by the 13-bit address specified by the operand and execution proceeds from that address. If the status flag is cleared to 0, the instruction following the BR instruction executes.

Opcode 40 – 5F

Syntax [<label>]^**BR**^<ADDR13>^...[<comment>]

Object Code



Execution if SF = 1, then ADDR13 → Program Counter [12:0]

if SF = 0, then Program Counter → Program Counter

Status Flag Always set to 1

Note:

The branch instruction is a conditional instruction. When a branch is used following an instruction that always leaves the status flag set to 1, the branch can be viewed as unconditional. To execute an unconditional branch after a command that affects the status flag, repeat the branch as shown in the example.

The branch instruction is local to a 8K byte 'page' specified by the upper 3 bits of the program counter. To move between pages, load the A register with the desired target address and execute the BRA instruction.

Example

```
      ACAAC #3F   ;Perform addition
      BR    LOC
      BR    LOC
LOC
```

Description BRA – Branch Always to Address in A Register

Action The program counter is loaded with the 16-bit address contained in the A register, and execution proceeds from that address.

Opcode 1F

Syntax [<label>]^**BRA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	1	1	1

Execution A → Program Counter

Status Flag Always set to 1

Notes:

1) This instruction is useful when a subroutine address has been placed in a table. The base address of the table can be added to the index and the address contained in the table can be fetched to the A register.

2) This instruction is also useful in moving between 8K-byte pages. Since the BR instruction is a local branch within an 8K-byte page specified by the upper 3 bits of the program counter, the only way to move between pages is with the BRA instruction.

3) While the extended-sign mode does not affect the operation of this instruction, it does affect the operation of many other instructions, including most instructions used to transfer values to the A register. Care should be taken that sign extension is not in effect when transferring values to the A register that are subsequently used by the BRA instruction, because the value can be changed during the transfer and unexpected results obtained.

Example

```
      TMAD  INDEX ;Bring table index in from memory
      ACAAC TABLE ;Add address of start of table
      LUAA        ;Bring new address into A register
      BRA         ;Branch to new address
TABLE
```

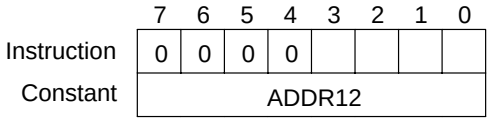
Description CALL – Call Subroutine If Status Set

Action If the status flag is set to 1, the contents of the program counter are pushed onto the stack, and the program counter is loaded with the address specified by the operand. Execution proceeds from that address. If the status flag is cleared to 0, the instruction following the CALL instruction executes.

Opcode 00 – 0F

Syntax [<label>]^**CALL**^<ADDR12>^...[<comment>]

Object Code



Execution If SF = 1, then Program Counter → Stack, and ADDR12 → Program Counter

If SF = 0, then Program Counter → Program Counter

Status Flag Always set to 1

Notes:

1) The program counter stack is capable of storing addresses up to seven levels deep. An address is pushed onto the stack whenever a CALL instruction occurs or whenever a hardware interrupt is executed. As addresses are pushed to the stack more than seven levels deep, the last seven addresses pushed to the stack are retained, and previous addresses are lost.

2) The CALL instruction is a conditional instruction. When a call is used following an instruction that always leaves STATUS high, it can be viewed as unconditional.

3) Because the CALL address is only 12 bits, subroutines should be placed in the lower 4K bytes of ROM. The BR instruction has 13 bits of address, making it possible to branch within an 8K block of ROM. Subroutines can, therefore, be located in the second 4K bytes of ROM by having the entry point in the lower 4K bytes with an immediate branch to the higher 4K bytes.

Description CLA – Clear A Register

Action Clears the contents of the A register to 0.

Opcode 2F

Syntax [<label>]^**CLA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	1	1	1

Execution $0 \rightarrow A$

Status Flag Always set to 1

Note:

This instruction is used to initialize the A register.

Description CLB – Clear B Register

Action Clears the contents of the B register to 0.

Opcode 24

Syntax [<label>]^**CLB**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	1	0	0

Execution $0 \rightarrow B$

Status Flag Always set to 1

Note:

This instruction is used to initialize the B register.

Description CLX – Clear X Register

Action Clears the contents of the X register to 0.

Opcode 20

Syntax [<label>]^**CLX**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	0	0	0

Execution $0 \rightarrow X$

Status Flag Always set to 1

Note:

This instruction is used to initialize the X register.

Description DECMN – Decrement Memory

Action Decrements the contents of the RAM location pointed to by the X register. If the bits are all zero, they are set to 1. If the bottom 8 bits are zero, they are set to 1 and the status flag is set. If not, they are simply decremented and the status flag is cleared.

Opcode 27

Syntax [<label>]^**DECMN**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	1	1	1

Execution *X – 1 → *X

Status Flag 1 if the lower 8 bits of memory went from all 00h to all FFh; 0 if not.

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EMU. While it does work on the actual part, it is not recommended that this be done due to the lack of debugging support.

Description DECXN – Decrement X Register

Action Decrements the contents of the X register. If the X register contains 000h, it is set to FFFh, otherwise, the X register is decremented. The status bit is conditional upon the value of the lower 8 bits of the X register.

Opcode 22

Syntax [<label>] ^**DECXN**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	0	1	0

Execution $X - 1 \rightarrow X$

Status Flag 1 if lower 8 bits of the X register went from 00h to FFh; 0 if not.

Description EXTSG – Change to Extended-Sign Mode

Action Changes MSP50x3x to extended-sign mode

Opcode 3C

Syntax [<label>]^**EXTSG**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	1	0	0

Execution The MSP50C3x is put into extended-sign mode. All data less than 16 bits in length are sign extended when being added to, subtracted from, or transferred to the A and B registers.

Status Flag Always set to 1

Note:

Sign extension means that the most significant bit of the data is copied into bits from 15 to the most significant bit of the data. For example, a 12-bit RAM location's most significant bit is bit 11. In extended-sign mode, bit 11 is copied into bits 12 through 15 when the data is transferred from the RAM location to the A register. This mode is useful if signed arithmetic must be done on values greater than 8 bits. Refer to each instruction description to determine if the arithmetic mode affects that particular instruction.

Description GET – Get Data From ROM/RAM

Action Transfers 1 to 8 bits of data from internal ROM or internal RAM to the A register by way of the parallel-to-serial register.

Opcode 30 to 37

Syntax [<label>]^**GET**^<N>^...[<comment>]

Object Code

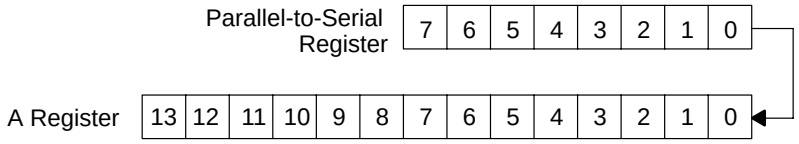
	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	0	N – 1		

Execution N bits of data are shifted from the LSB of the parallel–to–serial register into the LSB of the A register. This reverses the order of the bits in the A register from the order in the parallel-to-serial register. If more bits are required than are in the parallel–to–serial register, an additional byte is fetched from ROM or RAM.

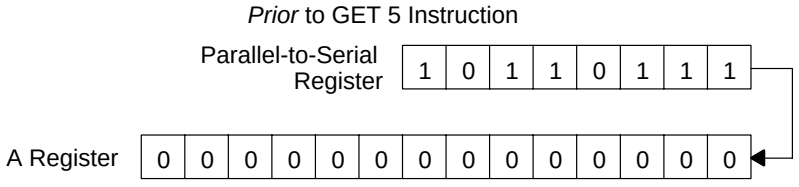
Status Flag 1 if the parallel-to-serial register buffer was emptied and needs to be reloaded on the next GET; 0 if not. The status flag result is not reliable for GET 7 and GET 8.

Notes:

1) The data is shifted out of the LSB of the parallel-to-serial register and into the LSB of the A register, resulting in a bit reversal of any single byte of data transferred into the A register from the order stored in the ROM.

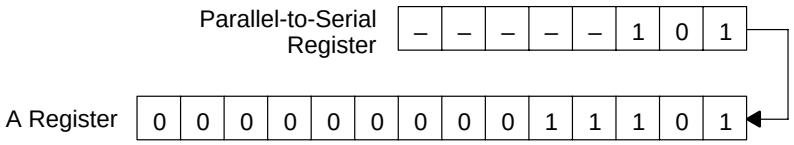


2) If more bits are requested than are immediately available in the parallel-to-serial register, the next data byte is loaded to the parallel-to-serial register and the remaining bits are transferred to the A register to satisfy the GET instruction.



Notes (continued):

After GET 5 Instruction



3) When the parallel-to-serial register is reloaded from ROM, the SAR (which is the address pointer for the GET instruction) is auto-incremented as needed. When the parallel-to-serial register is reloaded from RAM, the X register is the address pointer and is not auto-incremented.

4) Prior to the first use of the GET instruction, the GET counter, the parallel-to-serial register, and the mode register must be initialized. This initialization is accomplished by the TAMODE instruction and the LUAPS instruction, in that order. When using the GET instruction from RAM, a dummy GET 8 instruction must be performed after the LUAPS instruction and before the real GET. See subsection 5.8.2, *GET From Internal RAM*, for a sample program using RAM GET.

5) The source for the data fetched by the GET instruction can be either internal or external ROM or internal RAM. The TAMODE instruction is used to control the source of the data.

6) If the LPC bit is set to 1 and the first GET instruction after the LUAPS loads the maximum number of bits allowed (i.e., a GET 8 from internal ROM or RAM), the same data is loaded twice in a row. To avoid this problem, either perform the first GET before entering LPC mode or do a dummy GET (in the case of a GET from internal RAM, a total of two dummy GET 8 commands is required). Refer to Section 6.8, *Use of the GET Instruction*, for more information.

7) The GET 7 and GET 8 commands may result in an incorrect status flag result. If the status flag usage is significant to the application, then do the GETn commands in smaller amounts (substituting a GET 4,..., GET 3 combination for a GET 7, for example).

Description IAC – Increment A Register

Action Increments the contents of the A register by 1

Opcode 3A

Syntax [<label>]^**IAC**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	0	1	0

Execution A + 1 → A

Status Flag 1 if the lower 8 bits of the A register go from 0FFh to 000h; 0 if not.

Note:

This instruction increments all 16 bits of the A register, but only the lower 8 bits are used for status-flag determination.

Example

```
LOOP
      IAC          ;Increment loop counter
      SBR   DONE   ;Branch if loop counter overflow
      SBR   LOOP   ;Branch if no loop counter overflow
DONE
```

Description IBC – Increment B Register

Action Increments the contents of the B register by 1

Opcode 25

Syntax [<label>]^**IBC**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	1	0	1

Execution B + 1 → B

Status Flag 1 if the lower 8 bits of the B register go from 0FFh to 000h; 0 if not.

Note:

This instruction increments all 16 bits of the B register, but only the lower 8 bits are used for status-flag determination.

Example

```
LOOP
      IBC          ;Increment loop counter
      SBR   DONE   ;Branch if loop counter overflow
      SBR   LOOP   ;Branch if no loop counter overflow
DONE
```

Description INCMC – Increment Memory

Action Increments the contents of the RAM location pointed to by the X register. If the lower 8 bits are all 1s, they are cleared to all 0s and the status flag is set to 1. When this instruction is used with 12–bit RAM locations, the upper 4 bits increment whenever the lowest 8 bits change from all 1s to all 0s.

Opcode 26

Syntax [<label>]^**INCMC**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	1	1	0

Execution *X + 1 → *X

Status Flag 1 if the lower 8 bits of memory go from #FFh to 00h; 0 if not.

Description INTGR – Change to Integer Mode

Action Changes MSP50x3x to integer mode

Opcode 3B

Syntax [<label>]^**INTGR**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	0	1	1

Execution The upper bits of data less than 16 bits in length are 0 filled when being transferred to, added to, or subtracted from the A and B registers.

Status Flag Always set to 1

Note:

This instruction affects all data from RAM, the X register, or the timer register that are transferred to, added to, or subtracted from the A and B registers. It is used when only positive numbers are being used.

Description IXC – Increment X Register

Action Increments the contents of the X register by 1

Opcode 21

Syntax [<label>]^**IXC**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	0	0	1

Execution $X + 1 \rightarrow X$

Status Flag 1 if the contents of the X register go from FFh to 00h; 0 if not.

Note:

This instruction increments all 12 bits of the X register, but only the lower 8 bits are used for status–flag determination.

Example

```
LOOP
      IXC          ;Increment loop counter
      SBR   DONE   ;Branch if loop counter overflow
      SBR   LOOP   ;Branch if no loop counter overflow
DONE
```

Description LUAA – Look Up With A Register

Action Replaces the contents of the A register with the contents of the ROM addressed by the A register. When in extended-sign mode, the value fetched is sign extended to 16 bits.

Opcode 6B

Syntax [<label>]^**LUAA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	0	1	1

Execution *A → A

Status Flag Always set to 1

Extended-Sign Mode

When in extended-sign mode (EXTSG), the value loaded to the A register is sign extended. This can cause problems in two areas: 1) when loading the target address to the A register, the address may be changed if bit 7 is high, causing incorrect data to be loaded with the LUAA instruction; and 2) the data fetched may be sign extended. These problems can be avoided by ensuring that the processor is in integer mode (INTGR) prior to loading the A register.

Example

```
INTGR        ;Ensure integer mode
TCA   TABLE  ;Load table address
ACAAC INDEX  ;Add table offset
LUAA         ;Fetch table entry
```

Description LUAB – Look Up With B Register

Action Replaces the contents of the B register with the contents of the ROM addressed by the A register. When in extended-sign mode, the value fetched is sign extended to 16 bits.

Opcode 6D

Syntax [<label>]^**LUAB**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	1	0	1

Execution *A → B

Status Flag Always set to 1

Extended-Sign Mode

When in extended-sign mode (EXTSG) the value loaded to either the B register or the A register is sign extended. This can cause problems in two areas: when loading the target address to the A register, the address may be changed if bit 7 is high, causing incorrect data to be loaded with the LUAB instruction; and the data fetched to the B register may be sign extended. These problems can be avoided by ensuring that the processor is in integer mode (INTGR) prior to loading the A register.

Example

```
INTGR        ;Ensure integer mode
TCA   TABLE  ;Load table address
ACAAC INDEX  ;Add table offset
LUAB         ;Fetch table entry to B register
```

Description LUAPS – Indirect Look Up With A Register

Action Transfers A register contents to speech address register (SAR) and uses the resulting address to look up a speech data word. The data word is placed into the parallel-to-serial buffer and the SAR is incremented.

Opcode 6C

Syntax [<label>]^**LUAPS**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	1	0	0

Execution A → SAR; *SAR → PS; SAR + 1 → SAR

Status Flag Always set to 1

Note:

This instruction is used to initialize the parallel-to-serial register prior to the GET instructions. It should be used even if the data are coming from internal RAM. In these cases, the SAR does not need initialization, but the bit counter in the parallel-to-serial register still does.

Example

```
TCA   SPEECH   ;Load address of data
LUAPS          ;Initialize PS register
GET 4          ;Get first data
```

Description ORCM – OR Constant With Memory

Action Logically performs an OR operation the contents of RAM pointed to by the X register with the 8-bit operand and stores the results in RAM.

Opcode 64

Syntax [<label>]^**ORCM**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	1	0	0
Constant	CONST8							

Execution *X || CONST8 → *X

Status Flag Always set to 1

Note:

This instruction can be used to set an individual bit in RAM to 1.

Example

```
SILENCE   EQU    #01
          .
          .
          .
          TCX    FLAGS     ;Point to flags variable
          ORCM   SILENCE   ;Set silence bit high
```

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EMU. While it does work on the actual part, it is not recommended that this be done due to the lack of debugging support.

Description RETI – Return From Interrupt

Action If the interrupt is a level 1 interrupt, RETI retrieves the old contents of the A register, B register, status flag, integer mode bit, and X register from the interrupt storage locations; pops the top value from the stack to the program counter; and resumes execution from the new address in the program counter. If the interrupt is a level–2 interrupt, only the status flag, integer mode bit, and program counter are retrieved. If a RETI instruction is executed without interrupts being enabled, the instruction has no effect except to reset the interrupt pending latch. The mode register is not saved and restored during interrupts. Any changes made to the mode register during interrupts stay in effect after the RETI instruction.

Opcode 3E

Syntax [<label>]^**RETI**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	1	1	0

Execution If interrupts are not enabled: reset interrupt pending latch

If interrupts are enabled:

level 1: (A') → (A);(B')→ (B);(X')→ (X);(SF')→ (SF);(IF')→(IF);
(Top of Program Counter Stack)→ (Program Counter)

level 2: (SF')→ (SF);(IF')→ (IF)
(Top of Program Counter Stack)→ (Program Counter)

Status Flag Restored to value before interrupt

RETI Executed With Interrupts Enabled

If a RETI instruction is executed with interrupts enabled, but without an interrupt first occurring, the stack control may become corrupted.

Description RETN – Return From Subroutine

Action Pops the top value from the stack and resumes execution from the new address.

Opcode 3D

Syntax [<label>] ^**RETN**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	1	0	1

Execution Top of Stack → Program Counter

Status Flag Always set to 1

Notes:

If the stack is underflowed, RETN functions as a no-operation command. Control goes to the next consecutive address. Calls can be made indefinitely, but calls can only return seven levels.

Description SALA – Shift A Register Left

Action Shifts the A register left towards MSB by one bit and fills the LSB with a 0.

Opcode 2E

Syntax [<label>]^**SALA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	1	1	0

Execution A.i $\rightarrow$ A.i+1; 0 $\rightarrow$ A.0

Status Flag 1 if A.7 was a 1 before execution; 0 if A.7 was a 0 before execution.

Note:

The bit shifted out of bit 15 of the A register is lost. The results do not depend on the arithmetic mode (EXTSG or INTGR).

Description SALA4 – Shift A Register Left Four Bits

Action Shifts the A register left towards MSB by four bits and fills the lower 4 bits with zeros.

Opcode 1B

Syntax [<label>]^**SALA4**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	0	1	1

Execution A.i → A.i+4; 0 → A.0, A.1, A.2, A.3

Status Flag Always set to 1

Note:

Bits 12 to 15 of the A register are lost. The results do not depend on the arithmetic mode (EXTSG or INTGR).

Note:

When any multiply instruction is immediately followed by a SALA4 instruction, the result of the SALA4 instruction may be incorrect. This combination should be avoided.

Description SARA – Shift A Register Right One Bit

Action Shifts the A register right towards the LSB by 1 bit and fills the MSB with its old value.

Opcode 15

Syntax [<label>]^**SARA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	1	0	1

Execution A.i → A.i–1; A.15 → A.15

Status Flag Always set to 1

Note:

Data shifted out of bit 0 of the A register is lost. The results do not depend on the arithmetic mode (EXTSG or INTGR).

Description SBAAN – Subtract B Register From A Register

Action Subtracts the contents of the B register from the contents of the A register and stores the result in the A register. If the subtraction requires a borrow operation from bit 8 of the A register, the status flag is set to 1. Otherwise, the status flag is cleared to 0.

Opcode 2D

Syntax [<label>]^**SBAAN**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	1	0	1

Execution A – B → A

Status Flag 1 if the lower 8 bits of A register are less than the lower 8 bits of the B register; 0 if not.

Note:

The addition is performed independent of the arithmetic mode (EXTSG or INTGR) as a 2's complement subtraction of all 16 bits of the B register from the A register.

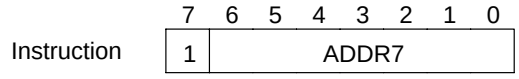
Description SBR – Short Branch If Status Set

Action When the status flag is set to 1, the lower 7 bits of the program counter are replaced by the value specified and execution proceeds from that address. Otherwise, the instruction following the SBR instruction is executed.

Opcode 80 to FF

Syntax [<label>]^**SBR**^<ADDR7>^...[<comment>]

Object Code



Execution If SF = 1, ADDR7 + Program Counter Page → PC

If SF = 0, Program Counter → Program Counter

Status Flag Always set to 1

Note:

1) The short branch instruction is a conditional instruction. When a short branch is used following an instruction that always leaves the status flag set to 1, the short branch can be viewed as unconditional.

2) The program counter is incremented when the instruction is fetched. If the program counter value is 0080h when the instruction is executed, placing an SBR with an operand of 1 at address 007Fh results in a branch to 81.

3) An SBR instruction executed at XX7Fh or XXFFh with status cleared (branch not taken) goes to XX80h or (XX+1)00h, respectively. Version 1.06 or greater of the assembler generates a warning message for all SBR instructions that occur at addresses ending in 7Fh or FFh.

Description SETOFF – Set Processor to OFF Mode

Action Places the processor in a low-power mode. The internal clock is stopped. The I/O ports retain their last programmed state.

Opcode 3F

Syntax [<label>]^**SETOFF**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	1	1	1	1	1

Execution Processor powered down

Status Flag State at power up is 1.

Note:

The processor may be restarted by either pulsing the INIT terminal low or by a negative transition on one of the designated wakeup terminals. If the processor is restarted using the INIT terminal, I/O ports are reset to a high impedance and program execution begins at ROM location 0000. If the processor is restarted using one of the designated wakeup terminals, the I/O ports retain their last programmed state and program execution begins at ROM location 0002. In either case, the register values are not retained, but the RAM values are retained provided that power continues to be applied to the chip.

Description SMAAN – Subtract Memory From A Register

Action Subtracts the contents of RAM addressed by the X register from the contents of the A register and stores the result in the A register. If the initial value in the lower 8 bits of the A register is less than the value in the lower 8 bits of RAM, the status bit is set to 1; otherwise, the status bit is cleared to 0. If the processor is in extended-sign mode, the value stored in memory is sign extended to a 16-bit value prior to the subtraction.

Opcode 29

Syntax [<label>]^**SMAAN**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	0	0	1

Execution A – *X → A

Status Flag 1 if the lower 8 bits of A register are less than the lower 8 bits in the RAM; 0 if not.

Note:

When the most significant bit of the memory being used is set, the subtraction results are dependent on the arithmetic mode (EXTSG or INTGR). A borrow from bit 8 sets the status flag in all cases.

This instruction may be used when the difference between two variables is desired. It subtracts the contents of the memory indexed by the X register from the A register.

Example

```
TMAD   VALUE1    ;Fetch value from memory
TCX    VALUE2    ;Point to second value
SMAAN            ;Subtract two values
TAMD   VALUE3    ;Store result in memory
```

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EMU. While it does work on the actual part, it is not recommended that this be done due to the lack of debugging support.

Description TAB – Transfer A Register to B Register

Action Copies the contents of the A register into the B register

Opcode 1A

Syntax [<label>]^**TAB**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	0	1	0

Execution A $\rightarrow$ B

Status Flag Always set to 1

Description TAM – Transfer A Register to Memory

Action Copies the contents of the A register into the memory location addressed by the X register. Since the memory location is too small to hold the complete contents of the A register, the most significant bits are lost in the transfer.

Opcode 16

Syntax [<label>]^**TAM**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	1	1	0

Execution A $\rightarrow$ *X

Status Flag Always set to 1

Description TAMD – Transfer A Register to Memory Direct

Action Copies the contents of the A register into the memory location addressed by the operand. Since the memory location is too small to hold the complete contents of the A register, the most significant bits are lost in the transfer.

Opcode 6A

Syntax [<label>]^**TAMD**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	0	1	0
Constant	CONST8							

Execution A → *CONST8

Status Flag Always set to 1

Description TAMIX – Transfer A Register to Memory and Increment X Register

Action Copies the contents of the A register into the memory location addressed by the X register and then increments the X register. Since the memory location is too small to hold the complete contents of the A register, the most significant bits are lost in the transfer.

Opcode 13

Syntax [<label>]^**TAMIX**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	0	1	1

Execution A $\rightarrow$ *X; X + 1 $\rightarrow$ X

Status Flag Always set to 1

Description TAMODE – Transfer A Register to Mode Register 1

Action Copies the lower 8 bits of the A register into the mode register 1

Opcode 1D

Syntax [<label>]^**TAMODE**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	1	0	1

Execution A → Mode Register 1

Status Flag Always set to 1

Note:

The bit definition for both mode registers is in Section 2.1.18, *Mode Registers*. Mode register 1 is memory mapped into RAM address FFEh. Mode register 2 is memory mapped into RAM address FFFh. Both registers can be written to using the TAM instruction, read using the TMA instruction, modified using the ORCM and ANDCM instructions, and tested using the TSTCM instruction. Only mode register 1 can be written to using the TAMODE instruction.

Description TAPSC – Transfer A Register to Prescale Register

Action Copies the lower 8 bits of the A register into the prescale register

Opcode 19

Syntax [<label>]^**TAPSC**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	0	0	1

Execution A → Prescale Register

Status Flag Always set to 1

Note:

The prescale circuit divides the timer clock by the value set by this instruction plus 1. The output of the prescale circuit is used as a clock for the timer register. Refer to subsection 2.1.14, *Timer Prescale Register*, for more information.

Description TASYN – Transfer A Register to Synthesizer Register

Action Copies the 16-bit A register to the speech channel selected by the CHANNEL bit in mode register 2. The destination and function of the value copied depends upon the operating mode of the channel. If the channel is set to LPC mode, the value is loaded to the pitch register for that channel. If the channel is set to PCM excited LPC, the value is loaded as the excitation input to the LPC synthesis filter. If the channel is set to PCM mode, the value is used directly as the output of the channel. If none of these modes is active (the channel is off), the value is loaded to the pitch register for the channel. This is done to allow preloading the pitch before turning on LPC mode.

Opcode 1C

Syntax [<label>]^**TASYN**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	1	0	0

Execution A → Speech-Processor Register

Status Flag Always set to 1

Note:

The TASYN copies the 16-bit contents of the A register to the following destinations depending on the contents of the mode registers (see subsection 2.1.18, *Mode Registers*).

Mode Bit		
LPC	**PCM**	**TASYN Destination**
0	0	Pitch register
0	1	DAC register
1	0	Pitch register
1	1	Excitation register

When loading the pitch register:

- The least significant bit and most significant bit of the A register are required to be cleared to zero.
- For voiced frames, the value in the A register:
  - Is required to be 0042h or higher
  - Is strongly recommended to be 0142h or higher
  - Is recommended to be 0202h or higher (see subsection 2.14, *Pitch Register and Pitch Period Counter (PPC)*)
- For unvoiced frames, the value in the A register is required to be between 0042h and 03FEh. Note that even when a frame is unvoiced, a pitch register value must be loaded.

Note:

If the CHANNEL bit in mode register 2 is cleared to 0, the TASYN transfers the A register contents to synthesizer channel 1. If the CHANNEL bit is set to 1, the TASYN instruction transfers the A register contents to synthesizer channel 2.

Description TATM – Transfer A Register to Timer Register

Action Copies the lower 8 bits of the A register into the timer register

Opcode 1E

Syntax [<label>]^**TATM**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	1	1	0

Execution A → Timer Register

Status Flag Always set to 1

Description TAX – Transfer A Register to X Register

Action Copies the contents of the lower 12 bits of the A register into the X register

Opcode 18

Syntax [<label>]^**TAX**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	1	0	0	0

Execution A $\rightarrow$ X

Status Flag Always set to 1

Description TBM – Transfer B Register to Memory

Action Copies the contents of the B register into the memory location addressed by the X register. Since the memory location is too small to hold the complete contents of the B register, the most significant bits are lost in the transfer.

Opcode 2A

Syntax [<label>]^**TBM**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	0	1	0

Execution B $\rightarrow$ *X

Status Flag Always set to 1

Description TCA – Transfer Constant to A Register

Action Copies the 8-bit constant specified by the operand into the A register. When in extended-sign mode, the 8-bit value is sign extended to a 16-bit two's complement value in the A register.

Opcode 6E

Syntax [<label>]^**TCA**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	1	1	0
Constant	CONST8							

Execution Extended-sign mode: CONST8 → A.7 – A.0; CONST8 (7) → A.15 – A.8

Integer mode: CONST8 → A.7 – A.0; 0 → A.15 – A.8

Status Flag Always set to 1

Description TCX – Transfer Constant to X Register

Action Copies the 8-bit constant specified by the operand into the X register

Opcode 62

Syntax [<label>]^**TCX**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	0	1	0
Constant	CONST8							

Execution Extended-sign mode: CONST8 → X.7 – X.0; CONST8 (7) → X.11 – X.8
Integer mode: CONST8 → X.7 – X.0; 0 → X.11 – X.8

Status Flag Always set to 1

Description TMA – Transfer Memory to A Register

Action Copy the contents of the memory addressed by the X register into the A register. When in extended-sign mode, the value fetched from RAM is sign extended to a 16-bit 2's complement value in the A register.

Opcode 11

Syntax [<label>]^**TMA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	0	0	1

Execution *X $\rightarrow$ A

Result depends on whether the TSP50x3x is in integer mode or extended-sign mode.

Status Flag Always set to 1

Note:

This instruction does not clear to zero the upper 8 bits of the A register when used on RAM locations 0xFEC, 0xFED, or 0xFEE. The user should be careful to not write code that depends upon the state of the upper 8 bits of the A register after moving data from these locations.

Description TMAD – Transfer Memory to A Register Direct

Action Copies the contents of the memory addressed by the operand into the A register. When in extended-sign mode, the value fetched from memory is sign extended to a 16-bit two's complement value in the A register.

Opcode 69

Syntax [<label>]^**TMAD**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	0	0	1
Constant	CONST8							

Execution If 8-bit RAM location then *CONST8 → A

If 12-bit RAM location: (*CONST12)→ (A)

The result depends on whether the MSP50C3x is in integer mode or extended-sign mode.

Status Flag Always set to 1

Description TMAIX – Transfer Memory to A Register and Increment X Register

Action Copies the contents of the memory location addressed by the X register into the A register and then increments the X register. When the processor is in extended-sign mode, the value fetched from RAM is sign extended to a 16-bit two's complement in the A register.

Opcode 14

Syntax [<label>]^**TMAIX**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	1	0	0

Execution *X $\rightarrow$ A

The result depends on whether the MSP50x3x is in integer mode or extended-sign mode

Status Flag Always set to 1

Note:

This instruction does not clear to zero the upper 8 bits of the A register when used on RAM locations 0xFEC, 0xFED, 0xFEE, or 0xFEF. The user should be careful to not write code that depends upon the state of the upper 8 bits of the A register after moving data from these locations.

Description TMXD – Transfer Memory Directly to X Register

Action Copies the contents of memory addressed by the operand into the X register

Opcode 6F

Syntax [<label>]^**TMXD**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	1	1	1	1
Constant	CONST8							

Execution If 8-bit RAM location: *CONST8 → X
If 12-bit RAM location: *CONST12 → X

The result depends on whether the MSP50C3x is in integer mode or extended-sign mode.

Status Flag Always set to 1

Description TRNDA – Transfer Random Number into A Register

Action Copies an 8-bit random number into the A register. Extended-sign mode does not affect the operation of this instruction. The value is not sign extended.

Opcode 2B

Syntax [<label>]^**TRNDA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	1	0	1	1

Execution Random Number → A

Status Flag Always set to 1

Notes:

1) The random number register generates a pseudorandom count with 32,767 states. The algorithm is summarized in the following paragraph.

2) At power up, the random number is initialized to 0. At every subsequent instruction cycle, the register is left shifted once, and bit 0 is set to the exclusive NOR of bits 13 and 14. The transfer to the A register in response to TRNDA is done prior to this operation.

3) The value is initialized to zero by the INIT terminal going low. It is not initialized by the SETOFF instruction and subsequent wakeup operation.

Description TSTCA – Test Constant With A Register

Action Compares the constant specified by the operand and the contents of the A register. If any bit in the operand is high with the corresponding bit in the A register low, the status flag is cleared to 0. Otherwise, the status flag is set to 1.

Opcode 67

Syntax [<label>]^**TSTCA**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	1	1	1
Constant	CONST8							

Execution (A && CONST8 == CONST8) → SF

Status Flag Conditionally set to 1 if every bit that is set to 1 in the operand has a corresponding bit set to 1 in the A register; otherwise it is cleared to 0. If the operand is 0, then the status flag is set to 1.

Note:

This instruction logically performs an AND operation the value stored in the A register with the value of the 8-bit constant and sets the status flag if the result is equal to the 8-bit constant. The value in the A register does not change.

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EVM. While it does work on the actual part, it is not recommended due to the lack of debugging support.

Description TSTCM – Test Constant With Memory

Action Compares the constant specified by the operand and the contents of the memory location addressed by the X register. If any bit in the operand is high with the corresponding bit in the memory location low, the status flag is cleared to zero. Otherwise, the status flag is set to 1.

Opcode 66

Syntax [<label>]^**TSTCM**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	1	1	0
Constant	CONST8							

Execution (*X && CONST8 == CONST8) → SF

Status Flag Conditionally set to 1 if every bit that is set in the operand has a corresponding set bit in the memory addressed by the X register; otherwise it is cleared to 0. If the operand is 0, then the status flag is set to 1.

Note:

This instruction logically ANDs the value stored in the RAM location pointed to by the X register with the value of the 8-bit constant and sets the status flag if the result is equal to the 8-bit constant. The value in memory is not affected.

Note:

This instruction does not work on RAM locations 0xFEC, 0xFED, or 0xFEE on the EVM. While it does work on the actual part, it is not recommended that this be done due to the lack of debugging support.

Description TTMA – Transfer Timer Register to A Register

Action Copies the contents of the timer register into the A register. When in extended-sign mode, the value fetched from the timer register is sign extended to a 16-bit 2's complement number in the A register.

Opcode 17

Syntax [<label>]^**TTMA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	1	1	1

Execution Extended-sign mode: Timer Register → A.7 – A.0;
Timer Register7 → A.15 – A.8

Integer mode: Timer Register → A.7 – A.0; 0 → A.15 – A.8

Status Flag Always set to 1

Description TXA – Transfer X Register to A Register

Action Copies the contents of the X register into the A register. When in extended-sign mode, the value transferred from the X register is sign extended into a 16-bit two's complement number in the A register.

Opcode 10

Syntax [<label>]^**TXA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	0	0	0

Execution extended-sign mode: X → A.11 – A.0; X.11 → A.15 – A.12

integer mode: X → A.11 – A.0; 0 → A.15 – A.12

Status Flag Always set to 1

Description XBA – Exchange Contents of B Register and A Register

Action Exchanges the contents of the B register with the contents of the A register

Opcode 12

Syntax [<label>]^**XBA**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	0	1	0	0	1	0

Execution $A \leftrightarrow B$

Status Flag Always set to 1

Description XBX – Exchange Contents of B Register and X Register

Action Exchanges the contents of the B register with the contents of the X register. The upper 4 bits of the B register are truncated in the move to the X register. When in extended–sign mode, the value transferred from the X register is sign extended into a 16-bit two's complement number in the B register. When in integer mode, the upper 4 bits of the B register are filled with 0s.

Opcode 23

Syntax [<label>]^**XBX**^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	0	1	0	0	0	1	1

Execution sign-extended mode: X $\rightarrow$ B.11 – B.0; X.11 $\rightarrow$ B.15 – B.12; B $\rightarrow$ X

integer mode: X $\rightarrow$ B.11 – B.0; 0 $\rightarrow$ B.15 – B.12; B $\rightarrow$ X

Status Flag Always set to 1

Description XGEC – X Register Greater Than or Equal to Constant

Action Compares the contents of the lower 8 bits of the X register and the constant specified by the operand and sets the status flag if the contents of the lower 8 bits of the X register are greater than or equal to the operand.

Opcode 61

Syntax [<label>]^**XGEC**^<CONST8>^...[<comment>]

Object Code

	7	6	5	4	3	2	1	0
Instruction	0	1	1	0	0	0	0	1
Constant	CONST8							

Execution $X \geq CONST8 \rightarrow SF$

Status Flag 1 if the contents of the lower 8 bits of the X register are greater than or equal to the operand; 0 if not.

Note:

Comparison is always done on an unsigned basis (i.e., FF is greater than FE). Only the lower 8 bits of the X register are compared to the 8-bit constant value. The upper 4 bits of the X register are not considered, so the result is independent of the arithmetic mode (EXTSG or INTGR).

Example

```
      XGEC   TESTV ;Is X ≥ TESTV
      SBR    GTE   ;Branch if so
LESS
GTE
```

Chapter 5

Applications

This chapter contains application information on the synthesizers, dual-channel synthesis, arithmetic modes, standby mode, slave mode, and generating tones with the MSP50C3x.

Topic **Page**

5.1 Synthesizer Control

In this section, a sample program demonstrates how to control the synthesizer in a MSP50C3x device. This program causes the device to synthesize speech from data stored in D6 format. It is described in the following sections.

5.1.1 Speech Coding and Decoding

The MSP50C3x device supports linear predictive coding (LPC) with ten or twelve K parameters. The LPC model requires the following three types of information: (1) pitch, (2) energy, and (3) up to 12 K parameters. The pitch parameter controls the input into the LPC system by providing one of two excitation signals. When the coded pitch code is nonzero, a periodic pulse similar to that produced by human vocal cords is created. A good example of the periodic sound is the A vowel sound. When the coded pitch code is zero, a white noise source similar to the turbulence generated by constricted airflow in the mouth is used. An example of this is the F sound. The LPC model is entirely digital; thus, the excitation function is a series of digital data samples.

The excitation function specified by the pitch code is multiplied by the energy parameter. The output of the multiplication is put into a filter whose resonance is determined by a number of K parameters (normally 10 or 12) to model the resonance of the human vocal tract. The output of the LPC model is a series of digital samples, typically at an 8-kHz or 10-kHz clock rate, that are put into the digital-to-analog converter (DAC).

The pitch, energy, and K parameters are stored in a coded form in a series of frames of various bit lengths. The sample program uses the D6 format for storing the speech data. In this format, each frame represents 200 samples. For a 10-kHz sampling rate, this corresponds to 20 ms per frame. Each parameter is stored using a set number of bits (see Table 5–1).

Table 5–1. D6 Parameter Size

Parameter	Energy	Repeat	Pitch	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	K12
# of Bits	4	1	7	6	6	5	5	4	4	4	3	3	3	0	0

As shown in Figure 5–1, the different frame sizes range from 4 bits to 55 bits depending on which parameters are needed for the specific frame type. The D6 format is an LPC-10 model, meaning that it uses ten K parameters to control the digital filter. K11 and K12 are, therefore, always cleared to 0 and no bits are needed to specify them.

Figure 5–1. D6 Frame Decoding

Frame	Energy	Repeat	Pitch	K1	K2	K3	K4	K5	K6	K7	K8	K9	K10
Voiced 55 Bits		0											
Un-voiced 34 Bits		0	0000000										
Repeat 12 Bits		1											
Silent 4 Bits	0000												
Stop 4 Bits	1111												

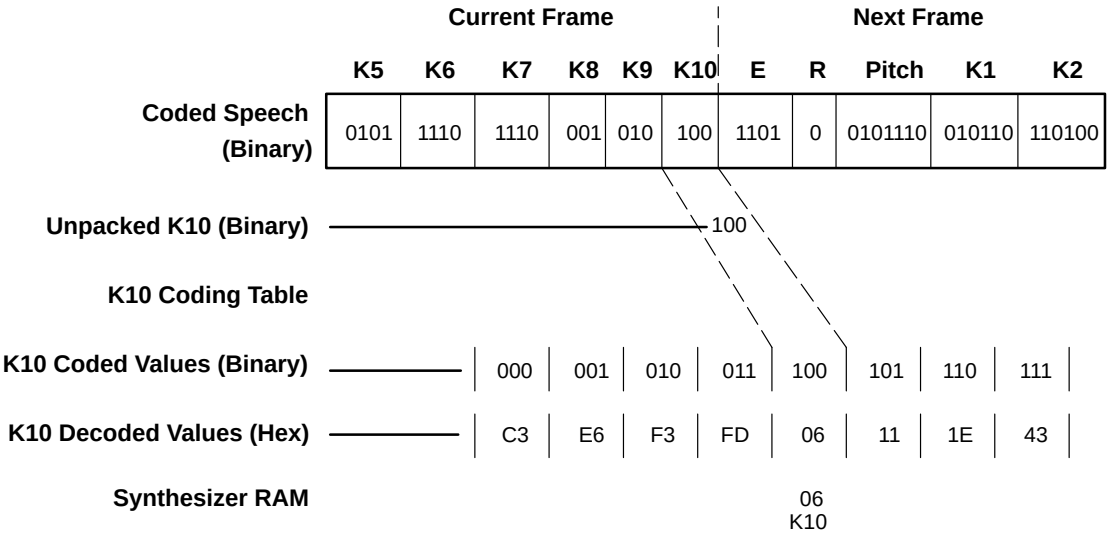
A silence is represented with a silent frame (specified by an energy of zero). No additional information is needed. A stop frame, indicated by an energy value of 1111 (binary), tells the processor that a particular word or phrase has ended and that control must be returned to the phrase selection program. Because a zero energy indicates a silence frame and a coded energy of 15 represents a stop frame, valid audible energies can range from 1 to 14.

The voiced frame is the longest frame type. All ten K parameters are used together with energy and pitch to specify a voiced frame. An unvoiced frame is indicated by a 0 pitch value. It is specified by a nonzero energy, a 0 pitch, and the first four K parameters.

When the vocal tract resonances change relatively slowly (e.g., with long vowels), two or more frames in a row may have the same values for their K parameters. When this occurs, the repeat bit is set to 1, and the K parameters are omitted. This is called a repeat frame.

All of the frames are arranged as serial bit streams. This means that a frame can start at any bit position within a given byte of memory. The GET instruction is used to get bits from memory in a serial fashion, freeing the programmer from bit-manipulation tasks. Once the bits for a particular parameter are extracted from the bit stream, they must be decoded before use in the synthesizer. The K10 unpacking and decoding process is shown in Figure 5–2.

Figure 5–2. Speech Parameter Unpacking and Decoding



To decode speech, the processor must do the following three things: (1) determine the frame type, (2) unpack each parameter, and (3) using a table lookup, decode each parameter. The specific details of these operations are given in Section 5.2, *Program Overview*,. The processor is also required to decide if each frame should be interpolated. Interpolation is used to smooth out the transitions between frames.

Most of the time, speech changes smoothly. If 20-ms frames are used without interpolation, changes occur abruptly and the speech sounds rough. The MSP50C3x devices require the program to interpolate the parameters. When speech changes quickly, as in the case of a transition between a voiced frame and an unvoiced frame, interpolation should not be performed. Therefore, the sample program disables interpolation at voicing transition.

5.1.2 RAM Usage

In the following discussion, all the addresses are given in hexadecimal notation.

The sample program uses 108 RAM locations. During synthesis, use of the 12-bit RAM locations 00h through 1Fh are fixed by the architecture of the MSP50C3x. As shown in Table 5–2, these locations are assumed by the synthesizer to contain the working values of the LPC speech parameters. The names given in parentheses are the variable names used in the sample program. When synthesis is disabled, these locations can be used at the programmer's discretion. Also, if only channel 1 is being used, the locations for channel 2 can be used elsewhere.

Table 5–2. Hardware-Fixed RAM Locations

RAM Location	Function
00	Temporary Value (TEMP)
01	Channel 1 Energy Working Value (EN_A)
02	Channel 1 K12 Working Value (K12_A)
03	Channel 1 K11 Working Value (K11_A)
04	Channel 1 K10 Working Value (K10_A)
05	Channel 1 K9 Working Value (K9_A)
06	Channel 1 K8 Working Value (K8_A)
07	Channel 1 K7 Working Value (K7_A)
08	Channel 1 K6 Working Value (K6_A)
09	Channel 1 K5 Working Value (K5_A)
0A	Channel 1 K4 Working Value (K4_A)
0B	Channel 1 K3 Working Value (K3_A)
0C	Channel 1 K2 Working Value (K2_A)
0D	Channel 1 K1 Working Value (K1_A)
0E	C1 Working Value (C1)
0F	C2 Working Value (C2)
10	Temporary Value (TEMP2)
11	Channel 2 Energy Working Value (EN_B)
12	Channel 2 K12 Working Value (K12_B)
13	Channel 2 K11 Working Value (K11_B)
14	Channel 2 K10 Working Value (K10_B)
15	Channel 2 K9 Working Value (K9_B)
16	Channel 2 K8 Working Value (K8_B)
17	Channel 2 K7 Working Value (K7_B)
18	Channel 2 K6 Working Value (K6_B)
19	Channel 2 K5 Working Value (K5_B)
1A	Channel 2 K4 Working Value (K4_B)
1B	Channel 2 K3 Working Value (K3_B)
1C	Channel 2 K2 Working Value (K2_B)
1D	Channel 2 K1 Working Value (K1_B)
1E	C3 Working Value (C3)
1F	Temporary Value (TEMP4)

Use of other RAM locations is detailed in Table 5–3. Pitch and the first two K factors are stored with 12 bits of precision, with the most significant byte stored in one location and the least significant nibble stored in the next consecutive location. The remaining K factors and energy are stored with 8 bits of precision.

Table 5–3. Other RAM Locations Used in Sample Program

RAM Location	Function	RAM Location	Function
20h	Ch.1 new energy value (EN_New_A)	21h	Ch.1 current energy value (EN_Old_A)
22h	Ch.1 new pitch value (PH_New_A)	23h	Ch.1 new fractional pitch value
24h	Ch.1 current pitch value (PH_Old_A)	25h	Ch.1 current fractional pitch value
26h	Ch.1 new K1 value (K1_New_A)	27h	Ch.1 new fractional K1 value
28h	Ch.1 current K1 value (K1_Old_A)	29h	Ch.1 current fractional K1 value
2Ah	Ch.1 new K2 value (K2_New_A)	2Bh	Ch.1 new fractional K2 value
2Ch	Ch.1 current K2 value (K2_Old_A)	2Dh	Ch.1 current fractional K2 value
2Eh	Ch.1 new K3 value (K3_New_A)	2Fh	Ch.1 current K3 value (K3_Old_A)
30h	Ch.1 new K4 value (K4_New_A)	31h	Ch.1 current K4 value (K4_Old_A)
32h	Ch.1 new K5 value (K5_New_A)	33h	Ch.1 current K5 value (K5_Old_A)
34h	Ch.1 new K6 value (K6_New_A)	35h	Ch.1 current K6 value (K6_Old_A)
36h	Ch.1 new K7 value (K7_New_A)	37h	Ch.1 current K7 value (K7_Old_A)
38h	Ch.1 new K8 value (K8_New_A)	39h	Ch.1 current K8 value (K8_Old_A)
3Ah	Ch.1 new K9 value (K9_New_A)	3Bh	Ch.1 current K9 value (K9_Old_A)
3Ch	Ch.1 new K10 value (K10_New_A)	3Dh	Ch.1 current K10 value (K10_Old_A)
40h	Ch.2 new energy value (EN_New_B)	41h	Ch.2 current energy value (EN_Old_B)
42h	Ch.2 new pitch value (PH_New_B)	43h	Ch.2 new fractional pitch value
44h	Ch.2 current pitch value (PH_Old_B)	45h	Ch.2 current fractional pitch value
46h	Ch.2 new K1 value (K1_New_B)	47h	Ch.2 new fractional K1 value
48h	Ch.2 current K1 value (K1_Old_B)	49h	Ch.2 current fractional K1 value
4Ah	Ch.2 new K2 value (K2_New_B)	4Bh	Ch.2 new fractional K2 value
4Ch	Ch.2 current K2 value (K2_Old_B)	4Dh	Ch.2 current fractional K2 value
4Eh	Ch.2 new K3 value (K3_New_B)	4Fh	Ch.2 current K3 value (K3_Old_B)
50h	Ch.2 new K4 value (K4_New_B)	51h	Ch.2 current K4 value (K4_Old_B)
52h	Ch.2 new K5 value (K5_New_B)	53h	Ch.2 current K5 value (K5_Old_B)
54h	Ch.2 new K6 value (K6_New_B)	55h	Ch.2 current K6 value (K6_Old_B)
56h	Ch.2 new K7 value (K7_New_B)	57h	Ch.2 current K7 value (K7_Old_B)
58h	Ch.2 new K8 value (K8_New_B)	59h	Ch.2 current K8 value (K8_Old_B)
5Ah	Ch.2 new K9 value (K9_New_B)	5Bh	Ch.2 current K9 value (K9_Old_B)

Table 5–3. Other RAM Locations Used in Sample Program (Continued)

5Ch	Ch.2 new K10 value (K10_New_B)	5Dh	Ch.2 current K10 value (K10_Old_B)
60h	Stored value of timer used to determine if an update is needed (TIMER)	61h	Stored value of timer used in INTP routine (SCALE)
62h	General LPC status and control flags (FLAGS)	63h	Channel 1 LPC status and control flags (FLAGS_1)
64h	Channel 2 LPC status and control flags (FLAGS_2)	65h	Stored phrase number for channel 1 (PHRASE1)
66h	Stored phrase number for channel 2 (PHRASE2)	67h	Most significant byte of channel 1 address for speech address register (ADR_MSB_1)
68h	Least significant byte of channel 1 address for speech address register (ADR_LSB_1)	69h	Bit offset into channel 1 address (OFFSET_1)
6Ah	Most significant byte of channel 2 address for speech address register (ADR_MSB_2)	6Bh	Least significant byte of channel 2 address for speech address register (ADR_LSB_2)
6Ch	Bit offset into channel 2 address (OFFSET_2)	6Dh	Most significant byte of channel 1 phrase address (ADR_MSB1)
6Eh	Least significant byte of channel 1 phrase address (ADR_LSB1)	6Fh	Most significant byte of channel 2 phrase address (ADR_MSB2)
70h	Least significant byte of channel 2 phrase address (ADR_LSB2)		

The program maintains copies of each decoded speech parameter for two separate frames: the current frame and the new frame. Normally, the synthesis routine interpolates smoothly between the current value of a given speech parameter and its new value. The interpolated value is written to the working value. However, in cases for which interpolation is not desired, the current value is written to the working value.

The value in the timer register is used for two purposes — to determine when a frame update needs to be performed and as a scale factor during interpolation. To serve these two purposes, three locations are reserved for freezing values read from the timer register. FLAGS contains the general status and control flags as detailed in Table 5–4. FLAGS_1 and FLAGS_2 contain the status and control flags for the respective channel as detailed in Table 5–5.

Table 5–4. FLAGS Bit Descriptions for Sample Program

Bit	Usage
0	Set if stop frame detected on channel 1.
1	Set if stop frame detected on channel 2.
2	Set if end of phrase reached on channel 1.
3	Set if end of phrase reached on channel 2.
4	Ch.1 inhibit interpolation bit 1
5	Ch.1 inhibit interpolation bit 2
6	Ch.2 inhibit interpolation bit 1
7	Ch.2 inhibit interpolation bit 2

Table 5–5. FLAGS_1 and FLAGS_2 Bit Descriptions for Sample Program

Bit	Usage
0	Set if disable INTP routine.
1	Set if new frame is a repeat frame.
2	Set if an update has been performed.
3	Set if current frame is a silent frame.
4	Set if current frame is an unvoiced frame.
5	Set if interpolation is not desired for frame.
6	Set if new frame is a silent frame.
7	Set if new frame is an unvoiced frame.

5.1.3 ROM Usage

The sample program uses approximately 2.6K bytes of ROM, leaving approximately 13.4K bytes (MSP50C32/37), 29.4K bytes (MSP50C33), or 61.4K bytes (MSP50C34) for other functions and speech data. Table 5–6 summarizes ROM usage.

Table 5–6. ROM Usage

ROM Locations				
‘32	‘33	‘34	‘37	Function
0000h	0000h	0000h	0000h	Execution start location after INIT rising edge
0000h – 000Fh	0000h – 000Fh	0000h – 000Fh	0000h – 000Fh	Device initialization code
0010h – 001Fh	0010h – 001Fh	0010h – 001Fh	0010h – 001Fh	Interrupt start locations
0020h – 0A12h	0020h – 0A12h	0020h – 0A12h	0020h – 0A12h	Synthesis program and tables
0A13h – 3FDFh	0A13h – 7FDFh	0A13h – FFDFh	0A13h – 3FDFh	Available for user program and speech data
3FDBh – 3FFFh	7FDBh – 7FFFh	FFDBh – FFFFh	3FDBh – 3FFFh	Test codes (not available to user)
4000h – 4200h	8000h – 8200h	10000h – 10200h	4000h – 4200h	Excitation codes (not available to user)

5.2 Program Overview

The sample dual synthesis program, parts of which are used in this section for explanation, is reproduced in its entirety in Appendix C, *MSP50C3x Sample Dual Synthesis Program*. The following is an outline of the program flow:

- ❑ Initialize processor
- ❑ Initialize speech address register, pitch, C1, and C2
- ❑ Decode first two frames of speech
- ❑ Start the synthesizer
- ❑ Until the stop frame is reached:
  - ■ Decode each frame
  - ■ When aninterrupt occurs, recalculate working parameter values
- ❑ Wait two frames, then stop the synthesizer
- ❑ Return to calling routine

The five main sections of the program are summarized in the following sections.

5.2.1 Initialization

The device state is unknown at device power up. The initialization section initializes the RAM and the mode register to a known state. In the sample program, this is accomplished by writing zeros to all RAM locations and to the mode register.

5.2.2 Phrase Selection

In general, the phrase section contains all application-specific code. In the sample program, this section merely contains repeated calls to the subroutine SPEAK, causing successive utterances to be spoken.

5.2.3 Speech Initialization

The speech initialization section consists of the subroutine SPEAK. It decodes the numbers contained in the A and B registers for a series of words into the addresses in ROM. It initializes the TSP50C3x for LPC synthesis on both channels and speaks the series of words that comprise the desired sentences. For each word in the sentence, SPEAK enables synthesis and loops until the utterance is complete. It then branches back to speak the next word in the sentence. This continues until the sentence is complete.

During each branch of the loop, the value in the timer register is polled. The next frame of speech data is read in each time the timer register underflows.

5.2.4 Interpolation Routine

Once a synthesizer is enabled by SPEAK, it writes a new value to the DAC approximately once every 60 or 90 instruction cycles. The value that is written is calculated from the values contained in RAM locations 01h to 0Fh for channel 1 or 11h to 1Fh for channel 2 and the contents of the pitch period counters (PPC1 and PPC2). Loading these locations with the correct values is the responsibility of the program checking the PPC bits (PPC1 and PPC2) and calling the appropriate interpolation routine (INTP1 or INTP2). The PPC bits are set hto 1 whenever the PPC decrements below 200h.

When interpolation is not inhibited, INTP performs a linear interpolation between the current value of each speech parameter and its new value using the value in the timer register to scale the interpolation. While it is possible to simply load the frame data to the working data, in practice, this results in speech that sounds rough due to the sharp transition between the different frames. To minimize this problem, INTP normally performs a linear interpolation between the current and new frames for each of the speech parameters. However, this is not always appropriate.

There are two cases in which the interpolation is inhibited and the transition is handled abruptly. When done, INTP simply copies the current values into the working values.

- Transition between voiced and unvoiced frames or between unvoiced and voiced frames — A different number of K parameters are used in voiced frames than in unvoiced frames, and the K parameters are different. Attempting to interpolate across the voicing transition results in strange sounds.
- An unvoiced frame following a silence — Plosives (e.g., the |Phaa| in the letter |P| are abrupt unvoiced sounds). Trying to interpolate this case results in a gradual ramp up of a plosive, which is incorrect. In the corresponding case of a voiced frame following a silence, it is acceptable to interpolate.

5.2.5 Frame-Update Routine

LPC speech is coded as a series of frames spaced in time. Periodically, the next frame must be read so that INTP (the interpolation routine) has new data to work with. This is the responsibility of the UPDATE routine. It reads the coded speech data contained in the next frame, determines what type of frame it is, decodes the speech data contained in the frame, and determines whether or not interpolation should be performed by INTP.

When a stop frame is encountered, a flag is set that causes the INTP routine to interpolate down to a silence. The synthesizer and the interpolation are both inhibited by the next pass through the update routine.

5.3 Dual Synthesis Program Walk-Through

The following is a walk-through of a simple MSP50C3x speech dual synthesis program. The approach used in this program is not the only possible approach, but it has the advantage of being relatively easy to explain. The complete listing of this program can be found in Appendix C, *MSP50C3x Sample Dual Synthesis Program*.

On power up, the processor begins executing code at location 0000h. On wakeup, the processor begins executing code at location 0002h. This way, the user can distinguish between a power up and a wakeup by putting different code at these locations. To make things simple, the sample program has the same code at these locations.

```
0005           ********************************************************
0006           *   Start of program
0007           ********************************************************
0008 0000                  AORG    #0000
0009        84             SBR     GOGO          ;Power up Vector
0010 0001   84             SBR     GOGO          ;(twice makes unconditional)
0011
0012 0002   84             SBR     GOGO          ;Wake up vector (same as power up)
0013 0003   84             SBR     GOGO          ;(twice makes unconditional)
0014
0015 0004 4022 GOGO        BR      GO            ;Branch to start of program
```

In this sample program, ROM addresses 0006h to 000Fh are not used. ROM addresses 0010h to 001F contain branches to the interrupt service routines. This section of the ROM address space is dedicated to these service routines by the MSP50x3x architecture. When an interrupt condition is generated, if the interrupt is enabled in the mode register, the contents of the program counter are replaced by the address of the appropriate interrupt vector. Normally, the instruction placed at the interrupt vector address is a branch to the actual routine. Because any branch instruction is conditional upon the value of the status bit and the value of the status bit is unknown, two short branches to the interrupt routine are used instead of a long branch. If the interrupt service routine is not within reach of a short branch, the target of the short branches should be a long branch to the interrupt service routine.

In this sample program, none of the possible eight interrupt conditions are used. The eight unused vectors point to a dummy routine that has no effect.

```
0017           ********************************************************
0018           *   Interrupt vectors
```

```
0019          ********************************************************
0020 0010                AORG    #0010
0021       A0            SBR     INT2_00      ;Timer Underflow, PCM=0, LPC = 0
0022 0011  A0            SBR     INT2_00      ;Timer Underflow, PCM=0, LPC = 0
0023 0012  A0            SBR     INT2_01      ;Timer Underflow, PCM=0, LPC = 1
0024 0013  A0            SBR     INT2_01      ;Timer Underflow, PCM=0, LPC = 1
0025 0014  A0            SBR     INT2_10      ;Timer Underflow, PCM=1, LPC = 0
0026 0015  A0            SBR     INT2_10      ;Timer Underflow, PCM=1, LPC = 0
0027 0016  A0            SBR     INT2_11      ;Timer Underflow, PCM=1, LPC = 1
0028 0017  A0            SBR     INT2_11      ;Timer Underflow, PCM=1, LPC = 1
0029 0018  A0            SBR     INT1_00      ;Pin (B1) goes low interrupt
0030 0019  A0            SBR     INT1_00      ;Pin (B1) goes low interrupt
0031 001A  A0            SBR     INT1_01      ;Clock interrupt, PCM=0, LPC = 1
0032 001B  A0            SBR     INT1_01      ;Clock interrupt, PCM=0, LPC = 1
0033 001C  A0            SBR     INT1_10      ;Clock interrupt, PCM=1, LPC = 0
0034 001D  A0            SBR     INT1_10      ;Clock interrupt, PCM=1, LPC = 0
0035 001E  A0            SBR     INT1_11      ;Clock interrupt, PCM=1, LPC = 1
0036 001F  A0            SBR     INT1_11      ;Clock interrupt, PCM=1, LPC = 1
0037
0038          INT1_10
0039          INT1_01
0040          INT2_10
0041          INT2_00
0042          INT2_01
0043          INT2_11
0044          INT1_00
0045 0020  2F INT1_11     CLA
0046 0021  3E             RETI
```

The following code initializes the processor by clearing the mode registers and the RAM. Since the first TMAD instruction after power up is not guaranteed to function correctly, a TMAD instruction is included in the initialization code. This ensures that the internal addressing is initialized correctly.

```
0293          ********************************************************
0294          *   Do program INITs
0295          ********************************************************
0296 0022 6900 GO         TMAD 0
```

```
0297
0298 0024   2F             CLA                 ;clear A to zero
0299 0025   20             CLX                 ;clear X to zero
0300 0026   22             DECXN               ;point to location #FFF
0301 0027   16             TAM                 ;Initialize Mode Register 2
0302 0028   22             DECXN               ;point to location #FFE
0303 0029   16             TAM                 ;Initialize Mode Register 1
0304
0305 002A   20             CLX                 ;clear X to zero
0306 002B   13 RAM_LOOP    TAMIX               ;Initialize All RAM to zeros
0307 002C 6180             XGEC    MAX_RAM+1   ;all RAM initialized?
0308 002E 4032             BR      RAM_EXIT    ;yes, exit loop
0309 0030 402B             BR      RAM_LOOP    ;no, continue looping
```

Generally, user programs have several levels of indirection in their use of speech address tables. Often, there are three levels of pointers:

- Sentence pointers that point to the start addresses of entries in the concatenation tables
- Concatenation tables that contain lists of word numbers that define specific sentences (each word number is used as an index into the word address table)
- A word address table containing the actual address of the start of each word in memory

Sometimes there are several sentences randomly selected for a given situation. This can lead to a fourth level of pointers that point to sentence groups. All of these levels of pointers are easily accessed using either the GET, LUAA, or LUAB instructions. The structure is dependent on the specific application.

This sample program uses three levels of indirection as previously described. The three tables are shown in the following code. Note that the use of single bytes to store the word numbers in the concatenation table restricts the vocabulary to 255 words. When a larger vocabulary is required, the BYTE directive should be replaced with a DATA directive and the appropriate changes made in the routine SPEAK.

The label VOC has the value of the start of the speech data. The number that is added to it is the offset into the speech data where a given word begins. Each of these word addresses occupies two bytes of memory.

```
2780            ********************************************************
```

```
2781           *                                                     *
2782           *    This is the lookup table giving the starting     *
2783           *    address of each concatenation list.              *
2784           *                                                     *
2785           *******************************************************
2786 0A13 0A17 SENTENCE    DATA    PHRASEa
2787 0A15 0A2D             DATA    PHRASEb
2788
2789
2790           *******************************************************
2791           *                                                     *
2792           *    This is the concatenation table giving the lists *
2793           *    of word numbers that define each phrase.         *
2794           *                                                     *
2795           *******************************************************
2796 0A17 0107 PHRASEa    BYTE    1,7,2,7,3,7,4,7,5,7,7,5,7,4,8,3,9,2,10,1,11,#FF
2797 0A2D 0701 PHRASEb    BYTE  7,1,7,2,7,3,7,4,7,5,7,7,5,8,4,9,3,10,2,11,1,10,#FF
2798
2799
2800           *******************************************************
2801           *                                                     *
2802           *    This is the lookup table for the speech stored at*
2803           *    VOC.                                             *
2804           *                                                     *
2805           *******************************************************
2806 0A44 0E5C SPEECH      DATA    SILENCE      ;Word 0       silence
2807 0A46 0A5C             DATA    #0000+VOC    ;Word 1       "One"  MALE
2808 0A48 0AE0             DATA    #0084+VOC    ;Word 2       "Two"
2809 0A4A 0B52             DATA    #00F6+VOC    ;Word 3       "Three"
2810 0A4C 0BD6             DATA    #017A+VOC    ;Word 4       "Four"
2811 0A4E 0C3C             DATA    #01E0+VOC    ;Word 5       "Five"
2812 0A50 0CE8             DATA    #028C+VOC    ;Word 6       "Six"
2813 0A52 0E7A             DATA    PAUSE        ;Word 7       pause for echo
2814 0A54 0E88             DATA    PAUSE2       ;Word 8       pause  "
2815 0A56 0E96             DATA    PAUSE3       ;Word 9       pause  "
2816 0A58 0EA4             DATA    PAUSE4       ;Word 10      pause  "
```

```
2817 0A5A 0EB2              DATA    PAUSE5        ;Word 11      pause  ”
```

The following code speaks two sentences, one on each channel, and then turns off the processor. The sentences are spoken at the same time (with some pauses in between words) to present an echo effect. The number of the desired sentence for channel 1 is loaded into the A register and the number of the desired sentence for channel 2 is loaded into the B register. The routine SPEAK is then called to process the sentences. The sentences are spoken, and then the processor is turned off.

```
0311 0032 6E01 RAM_EXIT     TCA     1             ;Speak 2nd phrase on channel 2
0312 0034   1A              TAB
0313 0035 6E00              TCA     0             ;Speak 1st phrase on channel 1
0314 0037 003B              CALL    SPEAK
0315
0316 0039   3F              setoff                ;shut down
```

What follows is the routine SPEAK that is called to speak each of the sentences. Before this routine is entered, the desired sentence numbers for channels 1 and 2 are loaded into the A and B registers respectively. The phrase numbers are stored into RAM because each one needs to be processed separately in order to get the addresses.

```
0317            ********************************************************
0318            *   Speak Utterance – Phrase numbers in A & B registers
0319            ********************************************************
0320 003B   3B SPEAK       INTGR
0321
0322 003C 6A65              TAMD    PHRASE1       ;Store channel 1 phrase number
0323 003E   12              XBA                   ;retrieve 2nd phrase number
0324 003F 6A66              TAMD    PHRASE2       ;Store channel 2 phrase number
0325
0326 0041   2F              CLA
0327 0042 6A62              TAMD    FLAGS         ;Init flags for speech
0328 0044 6A63              TAMD    FLAGS_1
0329 0046 6A64              TAMD    FLAGS_2
```

Because each sentence pointer is two bytes long, the sentence number is doubled to get the correct offset into the sentence pointer table. This offset is added to the start address of the table to get the address of the table entry. The LUAA and LUAB instructions are used to get the two byte address of the concatenation table entry. This selected concatenation table entry contains the

word number of the first word in the selected sentence. The address of the selected concatenation table entry is stored in ADR_MSB1 and ADR_LSB1 for channel 1 and ADR_MSB2 and ADR_LSB2 for channel 2.

```
0331           *-----Initialize address for channel 1-----*
0332
0333 0048 6965 LOOKUP1     TMAD    PHRASE1      ;Fetch stored phrase number
0334 004A   2E             SALA                 ;Double index to get offset
0335 004B 7A13             ACAAC   SENTENCE     ;Add base of table
0336
0337 004D   6D             LUAB                 ;get address MSB
0338 004E   3A             IAC                  ;point to address LSB
0339 004F   6B             LUAA                 ;get address LSB
0340 0050   12             XBA                  ;move LSB to B & MSB to A
0341 0051   1B             SALA4                ;combine MSB and LSB
0342 0052   1B             SALA4                ;into complete address
0343 0053   2C             ABAAC
0344 0054   1A             TAB                  ;save copy of address
0345 0055 6A6E             TAMD    ADR_LSB1     ;save address LSB
0346 0057 6801             AXCA    1        ;isolate next address MSB (shift by 8)
0347 0059   15             SARA
0348 005A 6A6D             TAMD    ADR_MSB1     ;save address MSB
0349
0350           *-----Initialize address for channel 2
0351
0352           LOOKUP2
0353 005C 6966             TMAD    PHRASE2      ;Fetch stored phrase number
0354 005E   2E             SALA                 ;Double index to get offset
0355 005F 7A13             ACAAC   SENTENCE     ;Add base of table
0356
0357 0061   6D             LUAB                 ;get address MSB
0358 0062   3A             IAC                  ;point to address LSB
0359 0063   6B             LUAA                 ;get address LSB
0360 0064   12             XBA                  ;move LSB to B & MSB to A
0361 0065   1B             SALA4                ;combine MSB and LSB
0362 0066   1B             SALA4                ;into complete address
0363 0067   2C             ABAAC
```

```
0364 0068   1A            TAB                 ;save copy of address
0365 0069 6A70            TAMD    ADR_LSB2    ;save address LSB
0366 006B 6801            AXCA    1        ;isolate next address MSB (shift by 8)
0367 006D   15            SARA
0368 006E 6A6F            TAMD    ADR_MSB2    ;save address MSB
```

The following code checks to see if the end of the phrase on channel 1 has been reached. If so, it branches to check channel 2. If not, it loads the current word address, increments the address to prepare for the next word in the phrase, saves the next address, and then restores the current address.

```
0370          *----Load next word address for channel 1
0371
0372 0070 6262 Nxt_Word1  TCX     FLAGS       ;point to general synthesis flags
0373 0072 6604            TSTCM   CH1END  ;has end of phrase on ch.1 been
                                             reached?
0374 0074 4091            BR      ChkCh2      ;yes, test channel 2
0375
0376 0076 696E            TMAD    ADR_LSB1    ;fetch address from
0377 0078   1A            TAB                 ;  memory
0378 0079 696D            TMAD    ADR_MSB1
0379 007B   1B            SALA4               ;merge MSB and LSB into
0380 007C   1B            SALA4               ;complete address
0381 007D   2C            ABAAC
0382 007E   1A            TAB                 ;save copy of address
0383 007F   3A            IAC                 ;generate next address
0384 0080 6A6E            TAMD    ADR_LSB1    ;save next address LSB
0385 0082 6801            AXCA    1           ;isolate next address MSB
0386 0084   15            SARA
0387 0085 6A6D            TAMD    ADR_MSB1    ;save next address MSB
0388 0087   12            XBA                 ;restore current address
```

The next word number is read from the current address in the next segment of code. It is tested to see if it is a stop word. If so, the end-of-phrase flag is set for channel 1. When the end-of-phrase flag is set for channel 2 also, the synthesis is turned off. When it is not set, silence is put on channel 1 until channel 2 finishes speaking.

```
0390 0088   6B            LUAA                ;get word number
0391 0089 60FF            ANEC    StopWord    ;is it a stop word?
0392 008B 4097            BR      SPEAK1a     ;no, continue
```

```
0393
0394 008D 6262             TCX     FLAGS        ;stop word reached
0395 008F 6404             ORCM    CH1END    ;signal end of phrase reached on ch.1
0396 0091 6608 ChkCh2      TSTCM   CH2END       ;is channel 2 already done?
0397 0093 4604             BR      STOP         ;yes, return
0398 0095 6E00             TCA     SIL          ;no, substitute silence
```

The following code doubles the word number to get the correct offset into the word table. The address of the word data is acquired and stored away. Offset is also initialized to prepare for the first GET from ROM. See Section 5.8, *Use of the GET Instruction*, for more information on the offset.

```
0400 0097   2E SPEAK1a     SALA                 ;Double index to get offset
0401 0098 7A44             ACAAC   SPEECH       ;Add base of table
0402 009A   6D             LUAB                 ;get address MSB
0403 009B   3A             IAC
0404 009C   6B             LUAA                 ;Get address LSB
0405 009D 6A68             TAMD    ADR_LSB_1    ;Save speech address LSB
0406 009F   12             XBA
0407 00A0 6A67             TAMD    ADR_MSB_1    ;Save speech address MSB
0408 00A2   2F             CLA
0409 00A3 6A69             TAMD    OFFSET_1     ;Initialize bit offset
```

The following code returns to zero the K parameters for the previously ended word and continues so that there is no break in the other channel.

```
0411 00A5 6262             TCX     FLAGS
0412 00A7 6620             TSTCM   Ch1Inh_2     ;is second inhibit bit set?
0413 00A9 44AC             BR      ZeroKs_1b    ;yes, return to ZeroKs
0414
```

There is also a routine to load the next word address for channel 2 (Nxt_Word2). This code is similar to the above code for channel 1, so it is not discussed here. It can be seen in the complete program listing in Appendix C.

Because LPC-10 coding is used in this example, the parameters K11 and K12 are not used. This section of code clears K11 and K12.

```
0461 00E6   2F             CLA                  ;Kill K11 and K12 parameters
0462 00E7 6A03             TAMD    K11_A
0463 00E9 6A13             TAMD    K11_B
0464 00EB 6A02             TAMD    K12_A
```

```
0465 00ED 6A12          TAMD    K12_B
```

The following code initializes the Y1, TEMP, and PITCH registers (#FF0–#FF4), the mode registers, and the speech flags.

```
0467 00EF 7FF0          ACAAC   #FF0           ;load address to X register
0468 00F1   18          TAX
0469
0470 00F2   2F          CLA
0471 00F3   13          TAMIX                 ;initialize Y1S (X=FF0)
0472 00F4   13          TAMIX                 ;initialize PITCH2 (X=FF1)
0473 00F5   13          TAMIX                 ;initialize Y1 (X=FF2)
0474
0475 00F6 7162          ACAAC   #162
0476 00F8   13          TAMIX                 ;initialize PITCH (X=FF3)
0477 00F9   2F          CLA
0478 00FA   13          TAMIX                 ;initialize TEMP (X=FF4)
0479
0480 00FB   20          CLX
0481 00FC   22          DECXN                 ;point to mode register 2 (X=FFF)
0482 00FD   16          TAM                   ;initialize mode register 2
0483 00FE   22          DECXN                 ;point to mode register 1 (X=FFE)
0484 00FF   16          TAM                   ;initialize mode register 1
0485
0486 0100 6A63          TAMD    FLAGS_1       ;Init flags for speech
0487 0102 6A64          TAMD    FLAGS_2
0488 0104 6262          TCX     FLAGS
0489 0106 65FE          ANDCM   ~STOP1
0490 0108 65FD          ANDCM   ~STOP2
```

The values in C1 and C2 control the behavior of the output filter. For most applications, the values should be set as shown in the following code. In addition, the C3 value controls the scaling of channel 2 to channel 1 when both channels are operating. Therefore, the smaller the value, the softer channel 2 output is.

```
0492 010A   2F          CLA                   ;Load C2 parameter
0493 010B 7B67          ACAAC   C2_Value      ;(a device constant)
0494 010D 6A0F          TAMD    C2
0495
0496 010F   2F          CLA                   ;Load C1 parameter
```

```
0497 0110 7F61            ACAAC   C1_Value     ;(a device constant)
0498 0112 6A0E            TAMD    C1
0499
0500 0114   2F            CLA                  ;Load C3 parameter
0501 0115 73FF            ACAAC   C3_Value     ;(channel scaling)
0502 0117 6A1E            TAMD    C3
```

The following code assigns a default pitch to cover the case in which the first frame is a silence frame. If this is the case, and no pitch was otherwise loaded, the pitch period counter is loaded with a zero and the synthesis of the first frame is then incorrect.

```
0507 0119   2F            CLA
0508 011A 700C            ACAAC   #0C
0509 011C 6A24            TAMD    PH_Old_A     ;save as old pitch, channel 1
0510 011E 6A22            TAMD    PH_New_A     ;save as new pitch, channel 1
0511 0120 6A44            TAMD    PH_Old_B     ;save as old pitch, channel 2
0512 0122 6A42            TAMD    PH_New_B     ;save as new pitch, channel 2
```

In the following code, the timer and prescale values are initialized.

```
0517 0124 6E5D            TCA     PSVALUE      ;Initialize prescale
0518 0126   19            TAPSC
0519 0127   1E            TATM
```

The first two frames for each channel are now preloaded. In the following code, each call to UPDATE loads one frame of speech for each channel. With two frames loaded into memory, the synthesis routine can properly do its interpolation function.

```
0524 0128 038C            CALL    UPDATE       ;Load first frame
0525 012A 038C            CALL    UPDATE       ;Load 2nd frame
```

In the following code, the first interpolation for channel 1 is done by calling the routine INTP1 and the first interpolation for channel 2 is done by calling the routine INTP2. Before each INTP is called, however, the timer values need to be initialized so that the interpolation function of INTP yields the correct value.

```
0532 012C 6E7F            TCA     #7F          ;Pretend there was a previous
0533 012E 6A60            TAMD    TIMER        ;update
0534
0535 0130 6E7F            TCA     #7F          ;Set timer to max value to
0536 0132   1E            TATM                 ;disable interpolation
0537
```

```
0538 0133 015A            CALL    INTP1     ;Do first interpolation for channel 1
0539
0540 0135 6E7F            TCA     #7F          ;Pretend there was a previous
0541 0137 6A60            TAMD    TIMER        ;update
0542
0543 0139 6E7F            TCA     #7F          ;Set timer to max value to
0544 013B   1E            TATM                 ;disable interpolation
0545
0546 013C 0274            CALL    INTP2     ;Do first interpolation for channel 2
```

The last step before the start of speech is to turn on the synthesizer. This is done by setting the appropriate bits in the mode registers.

```
0555 013E   20            CLX
0556 013F   22            DECXN
0557 0140   22            DECXN                ;point to mode register 1 (X=FFE)
0558 0141 6402            ORCM    LPC          ;turn on LPC for channel 1
0559 0143   21            IXC                  ;point to mode register 2 (X=FFF)
0560 0144 6402            ORCM    LPC          ;turn on LPC for channel 2
```

In the following code, when the synthesis routine detects a stop frame, it branches back to Nxt_Word1 or Nxt_Word2 (depending on which channel has the stop frame) to start speaking the next word. Until then, the program polls the value of the timer register and updates the frame data whenever the timer decrements below zero. It also checks the PPC bits to see if it is time for an interpolation. First, it tests whether the timer has already underflowed; if true, an update is performed. Second, it tests whether the channel 1 PPC bit is set; if true, INTP1 is immediately called. Third, it tests whether the channel 2 PPC bit is set; if true, INTP2 is immediately called.

```
0570 0146   20 SPEAK_LP   CLX
0571 0147   17            TTMA                 ;Fetch timer value
0572 0148 6380            AGEC    #80          ;Has it underflowed?
0573 014A 438C            BR      UPDATE       ;yes, do an update
0574
0575 014C   20            CLX
0576 014D   22            DECXN
0577 014E 6601            TSTCM   PPC1  ;has PPC for ch.1 decremented below 200h?
0578 0150 015A            CALL    INTP1        ;yes, interpolate
0579 0152   20            CLX
0580 0153   22            DECXN
```

```
0581 0154 6608            TSTCM   PPC2  ;has PPC for ch.2 decremented below 200h?
0582 0156 0274            CALL    INTP2       ;yes, interpolate
0583 0158 4146            BR      SPEAK_LP    ;no, wait some more
```

The following routine, INTP1, is the interpolation routine for channel 1. The interpolation routine for channel 2 (INTP2) is similar to this one; therefore, only INTP1 is discussed in this chapter. The routine INTP2 can be found within the complete program in Appendix C.

In the following code, INTP1 is reached whenever the pitch period counter for channel 1 decrements below 200h. Its purpose is to do any necessary interpolation of the reflection coefficients (K parameters) and to load the result into the working registers.

On entry to INTP1, the PPC1 bit is cleared to reset, and the CHANNEL bit is cleared to select channel 1.

```
0591 015A   20 INTP1      CLX
0592 015B   22            DECXN               ;point to Mode Register 2
0593 015C 65FE            ANDCM   ~PPC1       ;reset PPC1 bit
0594 015E 65EF            ANDCM   ~CHANNEL    ;select channel 1
```

If a word has just been loaded (besides the first word in a phrase), then two frames must be loaded by UPDATE before interpolation can occur. The following code checks to see if those two frames have been loaded yet (inhibit bits 1 and 2 are cleared when ready). If not, the energy must be cleared and INTP1 must be exited.

```
0600 0160 6262            TCX     FLAGS
0601 0162 6620            TSTCM   Ch1Inh_2  ;is second inhibit bit set?
0602 0164 4168            BR      INTP1a    ;yes, not ready for interpolation yet
0603 0166 416D            BR      INTP1b    ;no, continue with interpolation
0604
0605           INTP1a
0606 0168   2F            CLA                 ;clear energy while waiting
0607 0169 6A01            TAMD    EN_A
0608 016B 4272            BR      IRETI1      ;return from call
```

In the following code, the current value of the timer register is stored. This value is used later when interpolation is performed.

```
0610 016D   17 INTP1b     TTMA                ;Get timer register contents
0611 016E 6380            AGEC    #80         ;Has timer underflowed?
0612 0170 4174            BR      Adjust1    ;yes, Trick Scale to avoid problems
```

```
0613 0172 4175            BR      DoScale1     ;no, Use Timer as is
0614
0615 0174   2F Adjust1    CLA                  ;Pretend no underflow
0616 0175 6A61 DoScale1   TAMD    SCALE        ;Load timer val into scale
```

When interpolation has been turned off by the UPDATE routine, INTP is exited. This is shown in the following code.

```
0622 0177 6263            TCX     FLAGS_1      ;Point to channel 1 flags
0623 0179 6601            TSTCM   Int_Off      ;If routine disabled...
0624 017B 4272            BR      IRETI1       ;...branch to exit point
```

If there is a transition between a voiced frame and an unvoiced frame, then no interpolation should be performed between the two frames because the K parameters of a voiced frame are not compatible with the K parameters of an unvoiced frame. Any transition between frame types is detected in the UPDATE routine and signaled by setting the Int_Inh bit in FLAGS_1. The following code tests FLAGS_1 to see if interpolation should be performed between frames.

```
0632 017D 6263 TINTP1     TCX     FLAGS_1      ;Point to channel 1 status flags
0633 017F 6620            TSTCM   Int_Inh      ;Is interpolation inhibited?
0634 0181 4185            BR      NOINT1       ;yes, use inhibit code
0635 0183 41A4            BR      INTPCH1      ;no, use interpolation code
```

The following code is reached if interpolation is inhibited. It sets the stored value of the timer register to 7F, which effectively forces the interpolation to yield the old values for the working values, thereby effectively disabling interpolation.

```
0643 0185 6E7F NOINT1     TCA     #7F          ;Set Scale factor to
0644 0187 6A61            TAMD    SCALE        ;highest value
```

If there is a transition between a voiced and an unvoiced frame, the energy needs to be cleared until the K parameters and the unvoiced bit in the mode register all have been updated. This prevents the processor's LPC filter from using a mixture of voiced and unvoiced parameters. If the unvoiced bit in the mode register does not match the unvoiced bit in FLAGS_1, the energy is cleared. This is done in the following code.

```
0654 0189 6263            TCX     FLAGS_1
0655 018B 6680            TSTCM   Unv_Flg_Old  ;Is new frame unvoiced?
0656 018D 4198            BR      Uv1          ;yes, go to unvoiced branch
0657
0658 018F   20            CLX
0659 0190   22            DECXN                ;point to Mode Register 1
```

```
0660 0191   22             DECXN                ;New frame is voiced
0661 0192 6680             TSTCM   UNV          ;Has mode changed to unvoiced
0662 0194 419F             BR      ClrEN1       ;yes, clear the energy
0663 0196 41A4             BR      INTPCH1      ;no, no action required
0664
0665 0198   20 Uv1         CLX                  ;New frame is unvoiced
0666 0199   22             DECXN
0667 019A   22             DECXN
0668 019B 6680             TSTCM   UNV          ;Has voicing mode changed?
0669 019D 41A4             BR      INTPCH1      ;no, no action required
0670
0671 019F   2F ClrEN1      CLA                  ;Zero Energy during update
0672 01A0 6A01             TAMD    EN_A
0673 01A2 41A4             BR      INTPCH1
```

We are now ready to do the interpolation. Interpolation is done at this point with the standard linear equation:

$y = mx + b$

rewritten to:

$P = (Pcurrent - Pnew) \times TIMER + Pnew$

where:

P = interpolated parameter

Pcurrent = value of parameter in current frame

Pnew = value of parameter in new frame

TIMER = value in TIMER

The multiplication using the AXMA function scales the result by 080h. The value in TIMER ranges from 07Fh to 000h. When interpolation is inhibited, TIMER contains 07Fh and the interpolation results in P = Pcurrent.

The following code interpolates pitch. Pitch (as well as K1 and K2) is stored using two bytes. The program reads the most significant byte, left shifts it by one nibble, and then adds the least significant nibble of the value (stored in the second byte). The result is a 12-bit value. This is done both for the current and new values.

Unlike the K parameters, decoded pitch is always positive. The INTGR instruction at the start of INTP1 ensures the integer mode so that when the program

gets the decoded pitch from the decoding tables, it is not sign extended. See Section 5.4, *Arithmetic Modes*, for additional information on the integer and extended-sign modes.

```
0679 01A4 6222 INTPCH1     TCX     PH_New_A     ;Combine new pitch and fractional
0680 01A6   14             TMAIX                ;pitch and leave in
0681 01A7   1B             SALA4                ;the B register
0682 01A8   28             AMAAC
0683 01A9   21             IXC
0684 01AA   1A             TAB
0685 01AB   14             TMAIX                ;Combine current pitch and
0686 01AC   1B             SALA4                ; current fractional pitch
0687 01AD   28             AMAAC                ; and leave in A register
0688
0689 01AE   2D             SBAAN                ;(Pcurrent – Pnew)
0690 01AF 6261             TCX     SCALE
0691 01B1   39             AXMA                 ;(Pcurrent – Pnew) * Timer
0692 01B2   2C             ABAAC                ;Pnew + (Pcurrent – Pnew)* Timer
```

Unlike the other speech parameters, the interpolated pitch is not written to RAM. Instead, it is written to the pitch register using the TASYN instruction. In this particular code, the TASYN is commented out to avoid clicking. The pitch is loaded directly to the pitch register (X = #FF3). Because the value in the PPC is used to address the excitation function values, each of which is two bytes long, the interpolated pitch needs to be multiplied by two before writing it to the PPC. This is done in the following code using the SALA instruction.

```
0693 01B3   2E             SALA                 ;Double value to index excitation
0694
0695          **           TASYN                ;Write to pitch register
0696
0697 01B4   1A             tab                  ;Write to pitch register (#FF3)
0698 01B5   2F             cla            ;NOTE: TASYN instruction causes problems
0699 01B6 7FF3             acaac   #ff3         ;so we must write directly to
0700 01B8   18             tax                  ;the pitch register
0701 01B9   12             xba
0702 01BA   16             tam
```

The decoded energy, like the pitch, is always positive. The following code interpolates the energy.

```
0708 01BB 6220            TCX     EN_New_A     ;Put New Energy (adjusted to
0709 01BD   14            TMAIX                ;12 bits) in B register
0710 01BE   1B            SALA4
0711 01BF   1A            TAB
0712 01C0   14            TMAIX
0713 01C1   1B            SALA4                ;Put Current Energy (adjusted to
0714          *                                ;12 bits) in A register
0715 01C2   2D            SBAAN
0716 01C3 6261            TCX     SCALE
0717 01C5   39            AXMA                 ;(Ecurrent – Enew) * Timer
0718 01C6   2C            ABAAC                ;Enew + (Ecurrent – Enew)*Timer
0719 01C7   15            SARA
0720 01C8 6A01            TAMD    EN_A        ;Store Energy till mode is switched
```

Because the decoded K parameters can be both positive and negative, the program goes to extended-sign mode so that the values do not change sign when they are read into the A or B registers. This is done with the following line of code.

```
0722 01CA   3C            EXTSG                ;Allow negative K parameters
```

K1 and K2 are interpolated in the same manner as energy. The interpolation of K1 is shown in the following code. K2 is not shown.

```
0727 01CB 6226            TCX     K1_New_A     ;Combine New K1 and New
0728 01CD   14            TMAIX                ;fractional K1 and
0729 01CE   1B            SALA4                ;leave in the B register
0730 01CF   28            AMAAC
0731 01D0   21            IXC
0732 01D1   1A            TAB
0733
0734 01D2   14            TMAIX                ;Combine current K1 and
0735 01D3   1B            SALA4                ;current fractional K1 and
0736 01D4   28            AMAAC                ;leave in the A register
0737
0738 01D5   2D            SBAAN                ;(K1current – K1new)
0739 01D6 6261            TCX     SCALE
0740 01D8   39            AXMA                 ;(K1current – K1new) * Timer
0741 01D9   2C            ABAAC                ;K1new+(K1current–K1new)* Timer
0742 01DA 6A0D            TAMD    K1_A         ;Load interpolated K1 value
```

Since K3 through K10 are stored using an 8-bit precision instead of the 12-bit precision used for K1 and K2, the interpolation is simpler. The following fragment of code shows the interpolation used for K3. The code for K4 through K10 is similar.

```
0767 01ED 622E            TCX     K3_New_A      ;Put New K3 (adjusted to
0768 01EF   14            TMAIX                 ;12 bits) in B register
0769 01F0   1B            SALA4
0770 01F1   1A            TAB
0771
0772 01F2   14            TMAIX                 ;Put Current K3 (adjusted to
0773 01F3   1B            SALA4                 ;12 bits) in A register
0774
0775 01F4   2D            SBAAN                 ;(K3current – K3new)
0776 01F5 6261            TCX     SCALE
0777 01F7   39            AXMA                  ;(K3current – K3new) * Timer
0778 01F8   2C            ABAAC                 ;K3new+(K3current–K3new)* Timer
0779 01F9 6A0B            TAMD    K3_A          ;Load interpolated K3 value
```

If there has been a voicing change, the mode register needs to be changed to reflect the new value. The following code fragment changes the voicing bit in the mode register to reflect the state of the current frame (which is stored in FLAGS_1 for channel 1 and FLAGS_2 for channel 2). The LPC bit should also be reset when the unvoiced bit is being set on channel 1. This is done to avoid a glitch in the LPC bit. After changing the mode register, the program then exits from the INTP routine with RETN.

```
0942 025D   3B STMODE1    INTGR                 ;Back to integer mode
0943 025E 6263            TCX     FLAGS_1
0944 0260 65FB            ANDCM   ~Update_Flg   ;Signal that interp. done
0945 0262 6680            TSTCM   Unv_Flg_Old   ;Is current frame unvoiced?
0946 0264 426D            BR      SETUV1        ;yes, set mode to unvoiced
0947 0266   20            CLX                   ;no, ...
0948 0267   22            DECXN                 ;point to mode register 1
0949 0268   22            DECXN
0950 0269 657F            ANDCM   ~UNV          ;...set mode to voiced
0951
0952 026B   3E            RETI                  ;Return from interrupt
0953 026C   3D            RETN                  ;Return from first call
0954
```

```
0955 026D   20 SETUV1      CLX
0956 026E   22             DECXN               ;point to mode register 1
0957 026F   22             DECXN               ;Current frame is unvoiced, so
0958 0270 6480             ORCM    UNV+LPC     ;set mode to unvoiced and reset
0959                                           ;LPC bit to avoid glitch
0960 0272   3E IRETI1      RETI                ;Return from interrupt
0961 0273   3D             RETN                ;Return from first call
```

The last major section in this sample program is the routine that reads in the next frame and decodes it. The routine is called both from the speech initialization section (where it is used to preload the first two frames before enabling synthesis) and from the SPEAK_LP loop (where it is used to refresh the speech parameters when necessary).

The routine UPDATE does the following:

- ❏ If the stop frame is encountered on last pass, then stop speaking
- ❏ Copies new unvoiced flag to the current unvoiced flag
- ❏ Copies new silence flag to the current silence flag
- ❏ Sets new silence flag, new unvoiced flag, and interpolation flag to zero
- ❏ Copies new speech parameters to the current speech parameters
- ❏ Gets coded energy
- ❏ If silence frame, then sets new silence flag
- ❏ If stop frame, then sets stop flag
- ❏ Looks up decoded energy from table and puts in new energy
- ❏ If last frame was silent, then inhibits interpolation (this one is not silent)
- ❏ Gets repeat bit, if repeat bit is set to 1, then set the repeat flag
- ❏ Gets coded pitch
- ❏ If unvoiced frame, then sets new unvoiced flag
- ❏ Looks up decoded pitch from table and store as new pitch
- ❏ If new voicing is different from current voicing, then inhibits interpolation
- ❏ Gets coded K parameters
- ❏ Looks up decoded K parameters from table and store as new values

To prevent double updates, if the stored value of the timer register is zero, then it needs to be changed to 7Fh. This is done in the following code. If this is not done, then the polling routine discovers an underflow and calls UPDATE a second time.

```
1351 038C 6260 UPDATE      TCX     TIMER       ;Get stored value
1352 038E   11             TMA                 ;of Timer into A
1353
1354 038F 6000             ANEC    0           ;Is it zero?
```

```
1355 0391 4396             BR      UPDT00a      ;no, do nothing
1356 0393 6E7F             TCA     #7F          ;yes, replace value
1357 0395   16             TAM
```

The following code checks to see if the first two frames of a word have been loaded yet. When the end of a word is reached, the two inhibit bits are set for the next word. The first time through this segment of code, the first inhibit bit is cleared and the first frame is loaded. The second time through this segment, the second inhibit bit is cleared and the second frame is loaded. The reason for this is that interpolation cannot happen until the two first frames of the word have been loaded by UPDATE.

```
1359 0396 6262 UPDT00a     TCX     FLAGS
1360 0398 6610             TSTCM   Ch1Inh_1     ;is the first inhibit bit set?
1361 039A 439E             BR      UPDT00b      ;yes, clear it
1362 039C 43A2             BR      UPDT00c      ;no, test second bit
1363
1364 039E 65EF UPDT00b     ANDCM   ~Ch1Inh_1    ;clear first inhibit bit
1365 03A0 43AA             BR      UPDT00
1366
1367 03A2 6620 UPDT00c     TSTCM   Ch1Inh_2     ;is the second inhibit bit set?
1368 03A4 43A8             BR      UPDT00d      ;yes, clear it
1369 03A6 43AA             BR      UPDT00       ;no, continue with update
1370
1371 03A8 65DF UPDT00d     ANDCM   ~Ch1Inh_2    ;clear second inhibit bit
```

Now the program tests the stop flag. If it was set on the last pass through UPDATE, then the end of the current utterance has been reached, and the program needs to reset flags, zero the K parameters, and branch back to prepare for the next utterance in the phrase. This is done in the following code:

```
1380 03AA 6262 UPDT00      TCX     FLAGS        ;Point to flags
1381 03AC 6601             TSTCM   STOP1        ;Was stop frame encountered
1382 03AE 4495             BR      ZeroKs_1a    ;yes, zero channel 1 K parameters
```

Now before the next frame is loaded in, the flags from the new frame (the ones that tell the voicing of the frame and whether the frame is silent or not) need to be copied into the flags for the current frame. This is done in the following code:

```
1388 03B0 6263 UPDT01      TCX     FLAGS_1      ;Point to channel 1 flags
1389 03B2 0648             CALL    XFER_FLAGS   ;and xfer "new" to "old"
```

The following section of code is the subroutine called above. Besides copying the appropriate flags, this subroutine also resets the repeat flag, silence flag,

unvoiced flag and interpolation inhibit flag to 0. They are set later if the next frame requires them to be set.

```
2104           XFER_FLAGS
2107 0648 6610             TSTCM   Unv_Flg_New  ;Was new frame unvoiced?
2108 064A 4650             BR      SUNVL        ;  yes, set old frame unvoic
2109 064C 657F             ANDCM   ~Unv_Flg_Old ;  no, clear old frame voice
2110 064E 4652             BR      TSIL         ;and continue
2111
2112 0650 6480 SUNVL       ORCM    Unv_Flg_Old  ;Set old frame unvoiced.
2113
2116 0652 6608 TSIL        TSTCM   Sil_Flg_New  ;Was new frame silent?
2117 0654 465A             BR      SSIL         ;  yes, set old frame silent
2118 0656 65BF             ANDCM   ~Sil_Flg_Old ;  no, clear old frame not silent
2119 0658 465C             BR      ZROFLG       ;and continue
2120
2121 065A 6440 SSIL        ORCM    Sil_Flg_Old  ;Set old frame silent
2122
2128 065C 65C5 ZROFLG      ANDCM   #C5          ;reset flags
2129 065E   3D             RETN
```

In the following code, the new speech parameters are saved as current speech parameters prior to loading the next frame. The X register is loaded with the first parameter location prior to calling the subroutine. This allows the subroutine to be used for both channels.

```
1394 03B4 6220             TCX     EN_New_A     ;Point to ch. 1 data
1395 03B6 065F             CALL    XFER_PARMS   ;  and transfer parameters
```

The following section of code is the subroutine to transfer parameters which is called above. Pitch, K1 and K10 are shown.

```
2143 065F   14 XFER_PARMS  TMAIX                ;Transfer new frame En parameter
2144 0660   13             TAMIX                ;  to current frame location
2145           *-----PITCH-----
2146 0661   14             TMAIX                ;Transfer new frame pitch
2147 0662   21             IXC                  ;MSB from new to
2148 0663   16             TAM                  ;old frame location
2149
2150 0664   22             DECXN                ;Point to fractional pitch
2151
```

```
2152 0665   14            TMAIX              ;Transfer new fractional pitch
2153 0666   21            IXC                ;from new to old
2154 0667   13            TAMIX              ;frame location
2155          *-----K1-----
2156 0668   14            TMAIX              ;Transfer new frame K1
2157 0669   21            IXC                ;MSB from new to
2158 066A   16            TAM                ;old frame location
2159
2160 066B   22            DECXN              ;Point to fractional K1
2161
2162 066C   14            TMAIX              ;Transfer new fractional K1
2163 066D   21            IXC                ;from new to old
2164 066E   13            TAMIX              ;frame location
.
.
.
2196          *-----K10-----
2197 0684   14            TMAIX              ;Transfer new frame K10 parameter
2198 0685   13            TAMIX              ;to current frame location
.
2212 0686   3D            RETN               ;return from call
```

Before the new frame can be read in, the channel must be selected and the speech address register must be loaded with the address for the channel 1 speech. This must be done in dual synthesis because there is only one speech address register, so it must be loaded each time the channel changes in order to perform a GET from the correct location.

```
1400 03B8 0687            CALL    INIT1
```

The following section of code is the subroutine called above to turn on channel 1 and load the correct address to the speech address register. Since there is only one speech address register, it must be reloaded each time the channel changes in order to do a GET from the correct address. Normally with one channel going, the speech address register keeps track of the offset within the speech data. With two channels, when loading the speech address register, the offset must be kept track of. See Section 5.8, *Use of the GET Instruction*, for more information on the GET instruction and the offset for the speech address register.

INIT2 is called when it is time to load channel 2. The code for INIT2 can be seen in the complete program listing in Appendix C.

```
2222           INIT1
2223 0687   20         CLX
2224 0688   22         DECXN                    ; point to mode register 2
2225 0689 65EF         ANDCM   ~CHANNEL         ; turn on channel 1
2226
2227 068B 6968         TMAD    ADR_LSB_1        ; fetch address LSB from memory
2228 068D   12         XBA                      ; store in B register
2229 068E 6967         TMAD    ADR_MSB_1        ; fetch address MSB from memory
2230 0690   1B         SALA4                    ; combine MSB and LSB
2231 0691   1B         SALA4                    ;into complete
2232 0692   2C         ABAAC                    ;address
2233
2234 0693   6C         LUAPS                    ; prepare to get data
2235
2236 0694 6969         TMAD    OFFSET_1         ; retreive offset
2237 0696 6301         AGEC    1                ; is there an offset?
2238 0698 469C         BR      INIT1_1          ;yes, do a dummy get
2239 069A 46A3         BR      INIT1_X          ;no, return
2240
2241 069C   1A INIT1_1 TAB                      ; save offset in B register
2242 069D   2F         CLA                  ; clear offset memory location to avoid
2243 069E 6A69         TAMD    OFFSET_1         ;counting offset twice
2244 06A0   12         XBA                      ; retrieve offset
2245 06A1 06C1         CALL    C1GETN           ; do a dummy get
2246
2247 06A3   3D INIT1_X RETN                     ;return from call
```

The program is now ready to read in the new frame, decode it, and store the decoded values. Energy and pitch require special handling because of the special significance attached to certain values.

If energy has a value of zero, then the new frame is a silence frame. If the energy has a coded value of 15 (in this example), then the new frame is a stop frame. In the case of a stop frame, the program interpolates down to zero and then stops speaking. Between these two values, energy is decoded using a table look-up. The decoded value is stored in RAM.

The following code fragment reads the coded energy, sets the silence flag if the energy is zero, and sets the stop-frame flag and the silent-frame flag if the

energy is 15. If the coded energy is either zero or 15, the processor branches to a section of code that clears the energy and the K parameters.

The C1GETN subroutine gets the number of bits in the A register. See the subroutine code for more explanation on this subroutine and Section 5.8, *Use of the GET Instruction*, on how GET works with dual channels.

```
1408           *----- ENERGY -----
1409 03BA   2F             CLA
1410 03BB 6263             TCX     FLAGS_1      ;point to channel 1 flags
1411 03BD 6E04             TCA     EBITS
1412 03BF 06C1             CALL    C1GETN       ;Get coded energy
1413 03C1 6000             ANEC    ESILENCE     ;Is it a silent frame?
1414 03C3 43CB             BR      UPDT0        ;  No, continue
1415 03C5 6263             TCX     FLAGS_1
1416 03C7 6428             ORCM    Sil_Flg_New  ;Yes, set silence flag
1417 03C9 44B0             BR      ZeroKs_1     ;and zero K params
1418
1419 03CB 600F UPDT0       ANEC    ESTOP        ;Is it a stop frame?
1420 03CD 43D9             BR      UPDT1        ;no, continue
1421 03CF 6263             TCX     FLAGS_1
1422 03D1 6428             ORCM    Sil_Flg_New  ;yes, set flags
1423 03D3 6262             tcx     flags
1424 03D5 6401             orcm    stop1        ;signal stop frame reached on ch.1
1425 03D7 44B0             BR      ZeroKs_1     ;  and zero K params
```

In the following code fragment, the energy is decoded. The LUAA instruction is used to get the decoded energy.

```
1427 03D9 777B UPDT1       ACAAC   TBLEN        ;Add table offset to energy
1428 03DB   6B             LUAA                 ;Get decoded energy
1429 03DC 6A20             TAMD    EN_New_A     ;Store the Energy in RAM
```

If this is a silent frame (tested for earlier), no more parameters need to be read. In this code fragment, the program branches to the update routine for channel 2. At this same point in UPDATE_2, the program branches to the routine exit point.

```
1436 03DE 6263             TCX     FLAGS_1
1437 03E0 6608             TSTCM   Sil_Flg_New  ;Is this a silent frame?
1438 03E2 44CE             BR      Update_2     ;yes, go to channel 2 update
```

The next code fragment is reached if the new frame is not silent. It reads the repeat bit first. This bit is set to indicate that all of the K parameters between

the new frame and the previous one are identical. When this is so, the K parameters are not provided. A flag is set indicating that this is a repeat frame. Later, this flag is tested, and if this flag is not set, new K parameters are read in.

```
1443 03E4 6E01 UPDT2       TCA     RBITS
1444 03E6 06C1             CALL    C1GETN       ;Get the Repeat bit
1445 03E8 6701             TSTCA   #01          ;Is this a repeat frame?
1446 03EA 43EE             BR      SFLG1        ;  yes, set repeat flag
1447 03EC 43F2             BR      UPDT3        ;  no, continue
1448
1449 03EE 6263 SFLG1       TCX     FLAGS_1
1450 03F0 6402             ORCM    RepeatFlag   ;Set repeat flag
```

The next step is to read the coded pitch. This value is zero for an unvoiced frame and nonzero for a voiced frame. If it is unvoiced, then the unvoiced flag is set. This is done in the following code:

```
1452           *----- PITCH -----
1454 03F2 6E07 UPDT3       TCA     PBITS
1455 03F4 06C1             CALL    C1GETN       ;Get coded pitch
1456 03F6 6000             ANEC    PUnVoiced    ;Is the frame unvoiced?
1457 03F8 43FE             BR      UPDT3A       ;no, continue
1458 03FA 6263             TCX     FLAGS_1
1459 03FC 6410             ORCM    Unv_Flg_New  ;yes, set unvoiced flag
```

In the next fragment of code, the pitch is decoded. The SALA instruction doubles the index to compensate for the fact that pitch is stored as two bytes. The LUAB instruction gets the most significant byte of the decoded pitch. The LUAA gets the least significant nibble of the decoded pitch.

```
1461 03FE   2E UPDT3A      SALA                 ;Double coded pitch and
1462 03FF 778B             ACAAC   TBLPH        ;  add table offset to point
1463
1464 0401   6D             LUAB                 ;Get decoded pitch
1465 0402   3A             IAC
1466 0403   6B             LUAA                 ;Get decoded fractional pitch
1467 0404 6222             TCX     PH_New_A     ;Store the pitch and fractional
1468 0406   2A             TBM                  ;  pitch in RAM
1469 0407   21             IXC
1470 0408   16             TAM
```

If the voicing has changed between voiced and unvoiced or vice versa, interpolation needs to be inhibited because the tonal qualities of an unvoiced frame

differ markedly from those of a voiced frame. It is inappropriate to blend them with an interpolation. The following code tests for a change in voicing and sets a flag to inhibit interpolation if necessary. First, the new frame is tested.

```
1475 0409 6263             TCX     FLAGS_1
1476 040B 6610             TSTCM   Unv_Flg_New  ;Is the new frame unvoiced?
1477 040D 4411             BR      UPDT3B       ;yes, test previous frame
1478 040F 441B             BR      VOICE        ;no, go to voiced code
```

If the frame is unvoiced, the program reaches the following code. It tests the current frame to see if it is silent or voiced. If either condition is true, then a flag is set to inhibit interpolation. If the previous frame was silent, interpolation should be inhibited to avoid distorting a plosive that follows a silence. A plosive is an abrupt unvoiced sound that should not be interpolated. First, the program tests to see if the previous frame was silent.

```
1486 0411 6640 UPDT3B      TSTCM   Sil_Flg_Old  ;Was the previous frame silent
1487 0413 4421             BR      UPDT5        ;yes, inhibit interpolation
```

Then the program tests to see if the previous frame was voiced.

```
1489 0415 6680             TSTCM   Unv_Flg_Old  ;Was the previous frame unvoiced
1490 0417 4423             BR      UPDT4        ;yes, no need to change anything
1491 0419 4421             BR      UPDT5        ;no, inhibit interpolation
```

The following code is reached if the new frame is voiced. It simply tests to see if the previous frame was also voiced. If it was not, then interpolation is inhibited. Because it is acceptable to ramp up a voiced frame, the program does not need to test for a leading silent frame as with the unvoiced frame.

```
1498 041B 6680 VOICE       TSTCM   Unv_Flg_Old  ;Was the previous frame voiced
1499 041D 4421             BR      UPDT5        ;no, set no interpolation
1500 041F 4423             BR      UPDT4        ;yes, no need to change anything
```

The following code inhibits interpolation.

```
1502 0421 6420 UPDT5       ORCM    Int_Inh      ;Inhibit interpolation
```

Previously, the repeat bit was read to see if this is a repeat frame. If it is a repeat frame, then the new K parameters are the same as the current K parameters and no further action needs to be taken. If it is not a repeat frame, the program needs to continue reading the new K parameters. This section of code branches to the update routine for channel 2 if this is a repeat frame. At this same point in UPDATE_2, this section branches to the general routine exit if this is a repeat frame.

```
1509 0423 6602 UPDT4       TSTCM   RepeatFlag   ;Is repeat flag set?
1510 0425 44CE             BR      Update_2     ;yes, go to channel 2 update
```

The first four K parameters (K1 through K4) are now loaded. The first two of these decoded K parameters are 12-bit values that are stored in two bytes of RAM. The most significant 8 bits are contained in the first byte, and the least significant 4 bits are contained in the second byte.

In the following code, the C1GETN subroutine reads the coded K factor into the A register. For K1 and K2, it is left shifted (multiplied by 2) to convert it into an offset in the table that contains the two-byte uncoded K parameters. The offset is added to the starting address of the table with the ACAAC instruction. The LUAB instruction reads the most significant byte of the K factor, and the LUAA instruction reads the byte containing the least significant nibble. For K3 and K4, the coded K factor is added directly to the start of the table since they only consist of one byte. K1 and K3 are shown in the following code. K2 is similar to K1 and K4 is similar to K3.

```
1530           *-----K1-----
1531 0427 6E06             TCA     K1BITS
1532 0429 06C1             CALL    C1GETN       ;Get coded K1
1533 042B   2E             SALA                 ;Convert it to a
1534 042C 788B             ACAAC   TBLK1        ;  pointer to table element
1535 042E   6D             LUAB                 ;Fetch MSB of uncoded K1
1536 042F   3A             IAC
1537 0430   6B             LUAA                 ;Fetch fractional K1
1538 0431 6226             TCX     K1_New_A
1539 0433   2A             TBM                  ;Store uncoded K1
1540 0434   21             IXC
1541 0435   16             TAM                  ;Store fractional K1
.
.
1556           *-----K3-----
1557 0445 6E05             TCA     K3BITS
1558 0447 06C1             CALL    C1GETN       ;Get Index into K3 table
1559 0449 798B             ACAAC   TBLK3        ;  and add offset of table
1560
1561 044B   6B             LUAA                 ;Get uncoded K3
1562 044C 6A2E             TAMD    K3_New_A     ;and store it in RAM
```

Now the program tests to see if the new frame is unvoiced. Unvoiced frames use only four K parameters, and the remaining K parameters are set to zero. At this point, the first four K parameters are already loaded. The following code

fragment tests to see if the new frame is unvoiced, and, if it is, branches to code that zeroes the rest of the K parameters.

```
1574 0457 6263            TCX     FLAGS_1
1575 0459 6610            TSTCM   Unv_Flg_New  ;Is this an unvoiced frame?
1576 045B 44BF            BR      UNVC          ;yes, zero rest of K factors
```

The remaining K parameters are similar to the second two K parameters (K3 and K4) in that they have only an 8-bit precision for their decoded values instead of the 12-bit precision used for the first two K parameters. This precision reduction simplifies the code. K5 is shown in the following code fragment. K6 through K10 are similar to K5.

```
1585          *-----K5-----
1586 045D 6E04            TCA     K5BITS
1587 045F 06C1            CALL    C1GETN        ;Get Index into K5 table
1588 0461 79CB            ACAAC   TBLK5         ;  and add offset of table
1589
1590 0463   6B            LUAA                  ;Get uncoded K5
1591 0464 6A32            TAMD    K5_New_A      ;  and store it in RAM
```

After all the K parameters for a voiced frame have been loaded, the UPDATE routine continues with the update for channel 2. This is done in the following code. After all the K parameters for a voice frame have been loaded in UPDATE_2 the routine can be exited by branching to the general routine exit.

```
1639 0493 44CE            BR      UPDATE_2      ;Continue with channel 2 upd
```

This section of code clears K parameters that are not used. ZeroKs_1a is branched to when the end of a word is reached. Energy is cleared, flags are reset, pitch is initialized, and the inhibit bits are set to disable interpolation until two frames are read in. After the next word has been read in, we return to finish zeroing the K parameters. Silent and stop frames result in a branch to ZeroKs_1. Unvoiced frames result in a branch to UNVC.

```
1651          ZeroKs_1a
1652 0495   2F            CLA
1653 0496 6A01            TAMD    EN_A          ;Clear energy
1654 0498 6A21            TAMD    EN_Old_A
1655 049A 6A63            TAMD    FLAGS_1       ;Clear channel 1 flags
1656 049C 6262            TCX     FLAGS
1657 049E 65FE            ANDCM   ~Stop1        ;Reset Stop flag for next word
1658 04A0 6430            ORCM    Ch1Inh_1+Ch1Inh_2    ;Set interp inhibit flags
1659 04A2   2F            CLA
```

```
1660 04A3 7FF3            ACAAC   #FF3         ;Initialize pitch
1661 04A5   18            TAX
1662 04A6   2F            CLA
1663 04A7 7162            ACAAC   #162
1664 04A9   16            TAM
1665 04AA 4070            BR      Nxt_Word1    ;Get next word
1666          ZeroKs_1b
1667 04AC 660C        TSTCM   CH1END+CH2END;Have we reached the end for both ch1&2
1668 04AE 4604            BR      STOP           ;  yes, stop synthesis
1669
1670 04B0   2F ZeroKs_1    CLA
1671 04B1 6A20            TAMD    EN_New_A     ;Kill Energy
1672 04B3 6A26            TAMD    K1_New_A     ;Kill K1
1673 04B5 6A27            TAMD    K1_New_A+1
1674 04B7 6A2A            TAMD    K2_New_A     ;Kill K2
1675 04B9 6A2B            TAMD    K2_New_A+1
1676 04BB 6A2E            TAMD    K3_New_A     ;Kill K3
1677 04BD 6A30            TAMD    K4_New_A     ;Kill K4
1678 04BF   2F UNVC        CLA
1679 04C0 6A32            TAMD    K5_New_A     ;Kill K5
1680 04C2 6A34            TAMD    K6_New_A     ;Kill K6
1681 04C4 6A36            TAMD    K7_New_A     ;Kill K7
1682 04C6 6A38            TAMD    K8_New_A     ;Kill K8
1683 04C8 6A3A            TAMD    K9_New_A     ;Kill K9
1684 04CA 6A3C            TAMD    K10_New_A    ;Kill K10
1685          *           TAMD    K11_New_A    ;Kill K11
1686          *           TAMD    K12_New_A    ;Kill K12
1687 04CC 44CE            BR      UPDATE_2     ;Continue with channel 2 update
```

At this point the program would continue on to UPDATE_2, which is the update routine for channel 2. Because this code is similar to the above update routine for channel 1, it is not shown. See the complete program in Appendix C, *MSP50C3x Sample Dual Synthesis Program*.

If the stop flag has been set, the following code is reached. It turns off the synthesizer, writes a zero to the DAC in PCM mode, disables the interrupt, sets the voicing to voiced as a default for the next utterance, and then branches to SPEAK1 to begin the next utterance.

```
2036 0604   20 STOP        CLX
2037 0605   22              DECXN               ;point to mode register 2
2038 0606 65FD              ANDCM   ~LPC        ;Turn off LPC synthesis for ch.2
2039 0608 657F              ANDCM   ~UNV        ;Go back to voiced for next word
2040
2041 060A   2F              cla
2042 060B   3A loop1c       iac                 ; temporary fix for 3X problem
2043 060C 637F              agec    #7F       ; |   wait to turn off channel 1 LPC
2044 060E 4612              br      outc        ; |
2045 0610 460B              br      loop1c      ; |
2046                                            ; |
2047           outc
2048 0612   22              DECXN               ;point to mode register 1
2049 0613 65FD              ANDCM   ~LPC        ;Turn off LPC synthesis for ch.1
2050 0615 657F              ANDCM   ~UNV        ;Go back to voiced for next word
2051 0617 6404              ORCM    PCM         ;Enable PCM mode for channel 1
2052 0619   2F              CLA
2053 061A   1C              TASYN               ;Write a zero to the DAC
2054 061B 6EFA              TCA     #FA
2055 061D   3A BACK         IAC                 ;Wait for minimum of 30
2056 061E 4622              BR      out         ;  instruction cycles
2057 0620 461D              BR      back
2058 0622   20 OUT          CLX
2059 0623   22              DECXN               ;point to mode register 1
2060 0624   22              DECXN
2061 0625 65FB              ANDCM   ~PCM        ;disable PCM for channel 1
2062 0627   3D              RETN                ;return from routine
```

The following code sets a flag to indicate that a new frame has been loaded and then tests to see if LPC synthesis is enabled. If it is enabled, the processor reenables the level-1 interrupt and branches back to SPEAK_LP where it waits until the next interrupt and periodically polls the timer register until the next frame update is required. If LPC synthesis is not enabled, then the UPDATE routine was reached by a CALL instruction to preload the first two frames, and a RETN is executed to exit the UPDATE routine.

```
2064 0628   17 RTN          TTMA                ;Get current timer
2065 0629   1A              TAB                 ;  and store it away
```

```
2066
2067 062A   17 NOTDIFF     TTMA                ;Wait for timer to decrement
2068 062B   2D              SBAAN
2069 062C 6000              ANEC    0
2070 062E 4632              BR      DIFF
2071 0630 462A              BR      NOTDIFF
2072
2073 0632   17 DIFF         TTMA                ;Subtract 128 from value
2074 0633 7080              ACAAC   #80
2075 0635   1E              TATM
2076
2077 0636 6263              TCX     FLAGS_1     ;Set a flag indicating that
2078 0638 6404              ORCM    Update_Flg  ;the parameters are updated for
                                                ;ch.1
2079 063A 6264              TCX     FLAGS_2     ;Set a flag indicating that
2080 063C 6404              ORCM    Update_Flg  ;the parameters are updated for
                                                ;ch.2
2083
2084 063E   20              CLX
2085 063F   22              DECXN               ;point to mode register 1
2086 0640   22              DECXN
2087 0641 6602              TSTCM   LPC         ;Are we speaking yet?
2088 0643 4646              BR      RTN1        ;yes, reenable interrupt
2089 0645   3D              RETN                ;no, return for more data
2090
2091 0646 4146 RTN1         BR      SPEAK_LP    ;Go back to loop
2092
```

The speech data decoding tables can be seen in the complete sample program shown in Appendix C, *MSP50C3x Sample Dual Synthesis Program*.

The following routine takes the place of the GET instruction by itself. Whenever the speech address register is loaded (using LUAPS), the first GET following the LUAPS starts with the first bit at that address. Because the speech address is changing when going between channel 1 and channel 2, the offset would normally get lost. For example, if a GET obtains 4 bits after the LUAPS on channel 1, then GET 3 bits after the LUAPS on channel 2, the next LUAPS on channel 1 does not start where it left off (OFFSET = 4). Therefore, the user must keep track of the OFFSET for each channel. This routine gets the number

of bits in the A register, adds that number to the offset, and updates the address that was sent to the speech address register (stored in ADR_MSB_1 & ADR_LSB_1) if the offset goes above 8. See Section 5.8, *Use of the GET Instruction*, for more information on using the GET instruction with dual channels. The routine for the channel 2 GETs (C2GETN) is similar to this one, therefore, it is not shown here. See Appendix C, *MSP50C3x Sample Dual Synthesis Program*, for the complete program listing.

```
2285           ************************************************
2286           *  C1GETN                                      *
2287           *                                              *
2288           *  This subroutine gets the number of bits in  *
2289           *  the A register and updates the address sent *
2290           *  to the speech address register for channel 1. *
2291           ************************************************
2292           C1GETN
2293 06C1   1A         TAB                  ; save # of bits to get
2294 06C2   2E         SALA                 ; adjust to 2 byte vector size
2295 06C3 76C4         ACAAC   $+1          ; add table start location
2296 06C5   1F         BRA                  ; vector to jump table
2297
2298 06C6 46D6         BR      C1Get1       ; branch to get 1 bit
2299 06C8 46DC         BR      C1Get2       ; branch to get 2 bits
2300 06CA 46E2         BR      C1Get3       ; branch to get 3 bits
2301 06CC 46E8         BR      C1Get4       ; branch to get 4 bits
2302 06CE 46EE         BR      C1Get5       ; branch to get 5 bits
2303 06D0 46F4         BR      C1Get6       ; branch to get 6 bits
2304 06D2 46FA         BR      C1Get7       ; branch to get 7 bits
2305 06D4 4700         BR      C1Get8       ; branch to get 8 bits
2306
2307 06D6   2F C1Get1  CLA
2308 06D7   30 C1Get1a GET     1            ; get 1 bit
2309 06D8 4706         BR      C1GetN1
2310 06DA 4706         BR      C1GetN1
2311
2312 06DC   2F C1Get2  CLA
2313 06DD   31 C1Get2a GET     2            ; get 2 bits
2314 06DE 4706         BR      C1GetN1
```

```
2315 06E0 4706         BR      C1GetN1
2316
2317 06E2   2F C1Get3  CLA
2318 06E3   32 C1Get3a GET     3               ; get 3 bits
2319 06E4 4706         BR      C1GetN1
2320 06E6 4706         BR      C1GetN1
2321
2322 06E8   2F C1Get4  CLA
2323 06E9   33 C1Get4a GET     4               ; get 4 bits
2324 06EA 4706         BR      C1GetN1
2325 06EC 4706         BR      C1GetN1
2326
2327 06EE   2F C1Get5  CLA
2328 06EF   33         GET     4               ; get 4 bits
2329 06F0 46D7         BR      C1Get1a         ;+ 1 bit
2330 06F2 46D7         BR      C1Get1a
2331
2332 06F4   2F C1Get6  CLA
2333 06F5   33         GET     4               ; get 4 bits
2334 06F6 46DD         BR      C1Get2a         ;+ 2 bits
2335 06F8 46DD         BR      C1Get2a
2336
2337 06FA   2F C1Get7  CLA
2338 06FB   33         GET     4               ; get 4 bits
2339 06FC 46E3         BR      C1Get3a         ;+ 3 bits
2340 06FE 46E3         BR      C1Get3a
2341
2342 0700   2F C1Get8  CLA
2343 0701   33         GET     4               ; get 4 bits
2344 0702 46E9         BR      C1Get4a         ;+ 4 bits
2345 0704 46E9         BR      C1Get4a
2346
2347 0706   12 C1GetN1 XBA                     ; retrieve # of bits to get
2348 0707 6269         TCX     OFFSET_1        ; point to offset
2349 0709   28         AMAAC                   ; add offset to # of bits
2350 070A 6308         AGEC    8               ; is offset > 8?
```

```
2351 070C 4711         BR      C1GetN2         ;yes, increment address
2352
2353 070E   16         TAM                     ;no, store new offset
2354 070F 471C         BR      C1GetNX         ; and exit
2355
2356 0711 7FF8 C1GetN2 ACAAC   –8              ; subtract 8 from offset
2357 0713   16         TAM                     ; store new offset
2358 0714   22         DECXN                   ; point to address LSB
2359 0715   26         INCMC                   ; increment address LSB, overflow
2360 0716 471A         BR      C1GetN3         ;yes, increment address MSB
2361 0718 471C         BR      C1GetNX         ;no, exit
2362
2363 071A   22 C1GetN3 DECXN                   ; point to address MSB
2364 071B   26         INCMC                   ; increment address MSB
2365
2366 071C   12 C1GetNX XBA                     ; retrieve data
2367 071D   3D         RETN                    ; return from call
```

5.4 Arithmetic Modes

The interpretation of the value stored in a register or memory location is arbitrary and depends on the assumptions that programmers put into their software. A given value can represent a series of flags, a character value, a fractional number, or a range of integers. Normally, multiplication instructions assume a fractional value interpretation, and addition/subtraction instructions assume a range-of-integers interpretation.

Even if it is known that the value represents a range of integers, a problem remains — what range of integers is represented? If it is assumed that the contents of an 8-bit register represent a value ranging from -128_{10} to 127_{10} with 00h representing the most negative value and FFh representing the most positive value, the following problem arises: the addition of -127_{10} and 5_{10} should yield -122_{10} instead of:

$0000\ 0001_2 + 1000\ 0101_2 = 1000\ 0110_2$, or 6_{10}.

To solve this problem, negative numbers are usually represented with twos complement notation. Using this notation, a negative value is represented by one plus the inversion of its positive equivalent. Thus, to represent a negative one, its positive equivalent 0000 0001 is inverted to 1111 1110 and one is added to it:

$1111\ 1110_2 + 0000\ 0001_2 = 1111\ 1111_2$

Following is the calculation of the sum of –127 and 5 using this notation:

$1000\ 0001_2 + 0000\ 0101_2 = 1000\ 0110_2$, or -122_{10}

This is the correct result and solves the problem with negative values, but it restricts the range of positive values. The most significant bit now operates as a sign bit, leaving the remaining 7 bits to represent the absolute value. Only 127_{10} discrete positive values can be represented with those 7 bits, which is too restrictive in many applications.

To solve this problem, the MSP50C3x allows two different arithmetic modes. Upon initialization, the processor is in integer mode. In the integer mode, numbers are presumed by the processor to be integers ranging positive from zero. In the extended-sign mode, numbers are presumed by the processor to be values ranging positive or negative from zero, with negative numbers represented by twos complement notation.

The EXTSG and INTGR instructions are used to control the arithmetic mode of the MSP50C3x. The EXTSG instruction puts the processor in extended-sign mode, and the INTGR instruction puts the processor in integer mode.

Please note that the integer mode and the extended-sign mode are mutually exclusive; the processor is either in extended-sign mode or in integer mode but cannot be in both at the same time.

Transferring a value between the X register and the A register illustrates the difference in operation between the two modes. The X register has a size of 12 bits, and the A register has a size of 16 bits. A value of FFFh in the X register represents 4095 in integer mode or –1 in extended-sign mode. To maintain these values, the value left in the A register needs to be different between the two modes. Table 5–7 illustrates the difference.

Table 5–7. TXA Operation

MODE	X REGISTER		A REGISTER		VALUE
Integer Mode	FFFh	→	0FFFh	=	4095_{10}
Extended-Sign Mode	FFFh	→	FFFFh	=	-1_{10}
Integer Mode	005h	→	0005h	=	5_{10}
Extended-Sign Mode	005h	→	0005h	=	5_{10}

In extended-sign mode, the most significant bit acts as a sign bit. Because the value needs to be maintained over the transfer, the high-order bits of the A register are set to the state of the most significant bit of the X register. In integer mode, the high-order bits of the A register are simply set to 0.

Note that there is no difference in the operation between the two modes if the value represented is positive because in extended-sign mode, the most significant bit of a positive value is 0. When the value is transferred, the high-order bits are set to 0 the same as in the integer mode.

The operation of the following instructions are modified by the arithmetic mode:

- ACAAC — Add 12-bit constant to A register
- AMAAC — Add memory data to A register
- LUAA — Look up memory addressed by A register, result in A register
- LUAB — Look up memory addressed by A register, result in B register
- SMAAN — Subtract memory data from A register
- TCA — Transfer 8-bit constant to A register
- TMA — Transfer memory data to A register (indirect)
- TMAD — Transfer memory data to A register (direct)
- TMAIX — Transfer memory data to A register, increment X register
- TXA — Transfer X register contents to A register
- XBX — Exchange B register and X register contents

In general, these instructions transfer a value to the 16-bit A or B registers from a smaller register or memory location. Figure 5–3 illustrates the operation of

the ACAAC instruction in extended-sign mode. The 12-bit constant must be sign-extended to 16 bits (to match the size of the A register) prior to the addition. This modifies the value of the constant added to the A register from FFFh to FFFFh.

Figure 5–3. ACAAC in Extended-Sign Mode

CARRY		1111	1111	1111	1100_2
A REGISTER	3202h	0011	0010	0000	0010_2
CONSTANT	FFFFh	1111	1111	1111	1111$_2$
RESULT	3201h	0011	0010	0000	0001_2

Figure 5–4 illustrates the same operation in integer mode. In integer mode, the sign extension is not performed; consequently, the value added to the A register remains FFFh.

Figure 5–4. ACAAC in Integer Mode

CARRY		1111	1111	1111	1100_2
A REGISTER	3202h	0011	0010	0000	0010_2
CONSTANT	0FFFh	0000	1111	1111	1111_2
RESULT	4201h	0100	0010	0000	0001_2

5.5 Operation of the Multiply Instruction

On digital computers, a multiplication frequently results in a value that is much larger than either multiplicand. An example is the multiplication of two 2-bit numbers:

$11_2 \times 11_2 = 1001_2$

The result of multiplying two 2-bit numbers is a 4-bit number. Similarly, multiplying the 16-bit A register with the contents of an 8-bit memory location would result in a 24-bit value. This creates a problem because a value this large cannot be stored. One solution would be to limit the size of the multiplicands, but this would severely restrict the utility of the multiply instruction. A better solution is to interpret the multiplicands as fractions and to truncate the least significant part of the result. This solution minimizes overflow problems, and truncation affects the least significant portion of the result instead of the most significant part. In this scheme, an n-bit binary number is interpreted as follows:

value = $(-A_1 \times 2^0) + (A_2 \times 2^{-1}) + \ldots + (A_n \times 2^{1-n})$,

where $A_1 \ldots A_n$ are the bit values of the number. For example, the 4-bit number 1010 would be interpreted to have the following value:

value = $-1 + (0 \times 0.5) + (1 \times 0.25) + (0 \times 0.125) = -0.75$

Several points need to be emphasized:

- The possible values using this scheme range from –1 to slightly less than 1.
- Since the MSP50C3x instructions are all 8-bit by 16-bit multiply instructions, the lower 8 bits of the result are truncated.
- Since the lower 8 bits of the result are truncated, many multiplications give a zero result; for example:

(0000 0000 0000 1111) × (0000 0011) = 0000 0000 0000 0000 | 0010 1101
= 0000 0000 0000 0000

5.6 Power-Saving Modes

The MSP50C3x can be put into one of two power-saving modes after its initial power-up.

5.6.1 Standby Mode

Executing a SETOFF instruction places the MSP50C3x in the standby mode. When this occurs, the program stops executing and the device is put into a low-power state. The device can be brought to an active state by pulsing $\overline{\text{INIT}}$ low and then high or by a negative transition on a designated wakeup line. In standby mode, the I/O lines retain their the last programmed state, RAM is retained, and the program counter is cleared to 0000h on $\overline{\text{INIT}}$ or 0002h on wakeup (see Section 2.4, *MSP50C3x Power Control and Initialization*, for more information).

5.6.2 Sleep Mode

Taking the $\overline{\text{INIT}}$ terminal low reinitializes the MSP50C3x and puts the device in the sleep mode. The I/O lines are set to a high-impedance state and the device is completely initialized. The device can be brought to an active state by taking $\overline{\text{INIT}}$ high (see Section 2.4, *MSP50C3x Power Control and Initialization*, for more information).

5.7 Slave Mode

Setting bit 6 (SLAVE) of mode register 1 to 1 places the MSP50C3x in the slave mode. This specialized mode is intended for applications in which the MSP50C3x device needs to be controlled by a master microprocessor.

When in slave mode, the functionality of the following ports is modified:

- B0 becomes a chip enable strobe. It is normally held high. When it is taken low, data is read from or written to the A0 –A7 ports depending on the state of B1.
- B1 becomes a read/write select input. When B1 is low, data is written to the MSP50C3x when B0 goes low. When B1 is high, data may be read from the MSP50C3x when B0 goes low.
- Port A becomes a general bidirectional port controlled by B0 and B1. A7 is used as a busy signal. When bit 7 in the output latch is set to 1 by the software, A7 is reset to a low state when B0 goes low to write data to the MSP50C3x.

Because A7 is used as a busy flag, leaving only A0 –A6 for data, normally only 7 bits of data may be exchanged between the master and the slave in any one read operation from the MSP50C3x. In write operations to the MSP50C3x, all eight terminals of port A can transfer data.

During read operations from the slave MSP50C3x, the master is responsible for maintaining its outputs connected to the MSP50C3x port A in a high-impedance state. Otherwise, bus contention results.

The MSP50C3x I/O ports must be configured in input mode for slave mode to work properly. Port A7 can be put in output mode, if desired. It then functions as a handshaking line rather than a polled handshake bit.

5.7.1 Slave-Mode Write Operation

A typical sequence for an 8-bit write operation to the MSP50C3x in the slave mode is shown in Figure 5–5. At the beginning of the operation, the MSP50C3x has a low in the A7 output latch. It is there either because it was written there with software or because it was set low by the hardware on completion of a previous write operation. The data transfer occurs as follows:

1) The master microprocessor sets R/W high to indicate a read operation.

2) The master polls the output state of A7 by pulsing STR (on B0) low and reading the state of A7 while STR is low.

3) Eventually, the MSP50C3x completes processing any previous data or instructions from the master. When it does, it writes a 1 to the A7 output latch.

4) When the master senses that A7 has gone high, it sets the R/W signal low to indicate a write operation.

5) The master presents valid data to port A0 – A6.

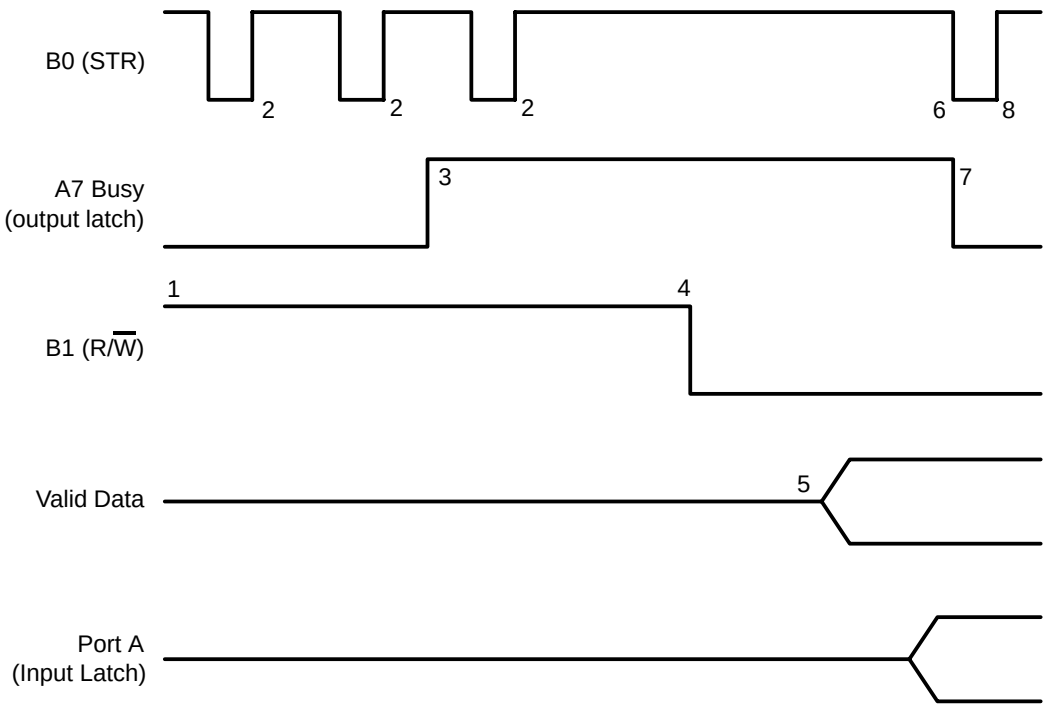
6) The master pulses STR (on B0) low, which causes the data on port A0 – A6 to be latched to the port A input latch. The MSP50C3x hardware causes the A7 output latch to be cleared to 0, indicating that the MSP50C3x has accepted the data.

7) The MSP50C3x polls the A7 output latch. When the MSP50C3x sees A7 go low, it knows that data is being written to the port A input latch.

8) The MSP50C3x polls the B0 (STR) input line. When B0 goes high, the write is complete, and the data in A is valid.

9) When it is ready to accept another command, the MSP50C3x writes a 1 to the A7 output latch, thus starting another cycle.

Figure 5–5. Slave-Mode Write Operation



5.7.2 Slave-Mode Read Operation

A typical sequence for an 8-bit read operation from the MSP50C3x in the slave mode is shown in Figure 5–6.

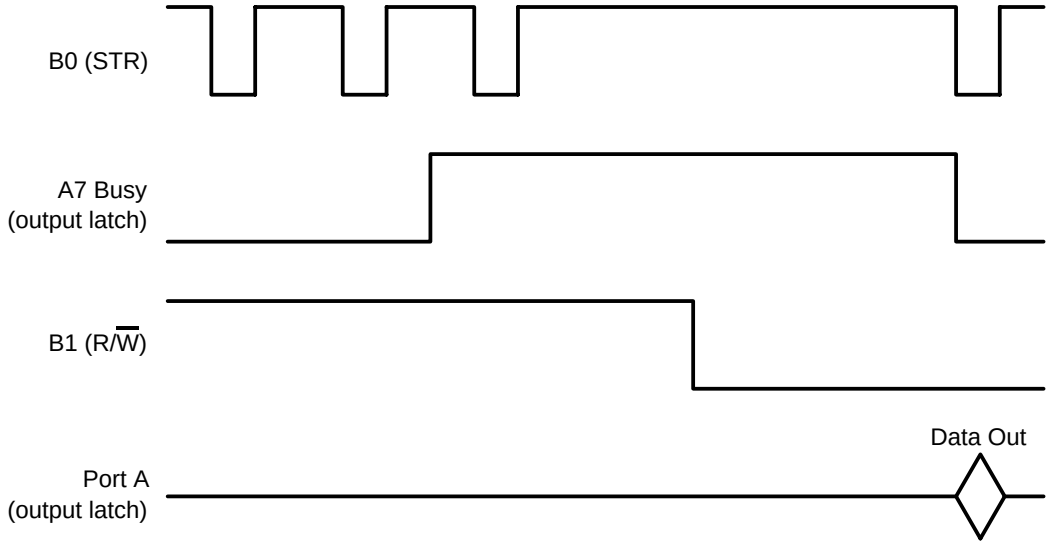
At the beginning of the operation, the MSP50C3x has a low in the A7 output latch. It has received a command or a request for information from the master. When the MSP50C3x is ready to respond, the data transfer occurs as follows:

1) The MSP50C3x writes the data to A0 – A6 and a 1 to port A7. The 1 on port A7 is a signal that valid data is available in the terminals connected to port A.

2) The master periodically polls port A. When it finds A7 has gone high, it knows that A0 – A6 contains valid data.

3) A7 remains high, indicating that the slave is prepared for another command. The master can write to the slave at any time. When the slave polls the A7 output latch and finds it low, the slave knows that a new command from the master is in the port A latch.

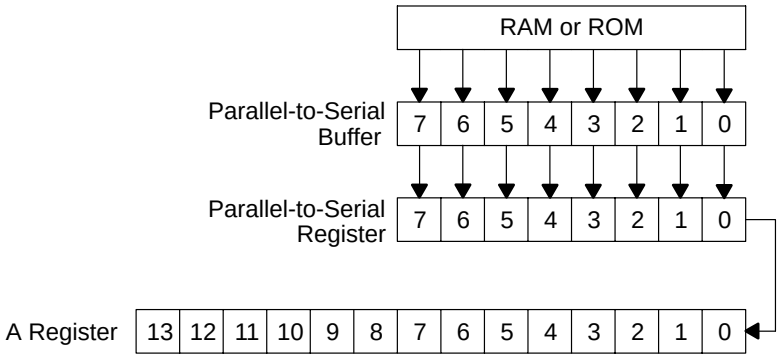
Figure 5–6. Slave-Mode Read-Then-Write Operation



5.8 Use of the GET Instruction

The GET instruction is used to retrieve a bit stream from the RAM or the internal ROM. It allows the program to unpack speech data in a time-efficient manner. As shown in Figure 5–7, it is implemented through the use of a parallel-to-serial shift register.

Figure 5–7. Register Connections for GET Instruction

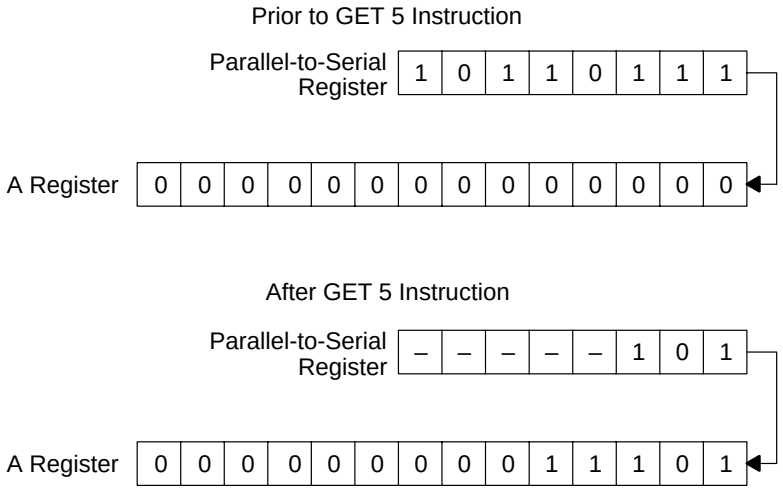


The parallel-to-serial register (P/S register) is loaded in a parallel manner from the parallel-to-serial buffer (P/S buffer), which is in turn parallel loaded from the source of the data (which could be internal ROM or internal RAM). When the GET instruction is executed, the number of bits specified in the operand of the GET instruction are shifted out of the LSB of the P/S register into the LSB of the A register.

When the number of valid bits in the P/S register is less than the specified number of bits, the contents of the P/S buffer are loaded during operation to the P/S register and the contents of the P/S buffer are refreshed from the data source the next time that a GET instruction is executed. The status bit is set to 1. If the buffer did not need to be reloaded, the status bit is cleared to 0.

Note that because the data is shifted out of the LSB of the P/S register and into the LSB of the A register, there is a byte reflection of the data in this process as illustrated in Figure 5–8. This figure shows the state of the P/S register and the A register both before and after a GET 5 instruction. Prior to the GET 5, the P/S register contains B716, and the A register contains all zeros. After the instruction, the least significant 5 bits of the P/S register are shifted into the A register. Because of the bit flip, the A register contains 1D16 after the shift operation. The P/S register has only 3 valid bits left after the operation. When more than 3 bits are requested in the next GET operation, the P/S register is reloaded from the P/S buffer.

Figure 5–8. Parallel-to-Serial Operation for GET 5 Instruction



Note: GET Instruction

Because of timing problems that can cause the same data to be fetched from the data source twice in a row the first two times the GET instruction is executed; unless special precautions are taken, initialization for the GET instruction should not be done while the mode register LPC bit is set.

Specifically, if the LPC bit is set and the first GET instruction is a GET 8 from internal ROM or RAM, the P/S is loaded with the same data twice in a row. To avoid this problem, either do a double GET in this situation, or, more simply, never be in LPC mode during the interval between the LUAPS instruction and the first GET instruction.

5.8.1 GET From Internal ROM

When the RAMROM bit of mode register 1 is 0, the data source for GET instructions is the internal ROM (see Table 5–8). As detailed in Section 5.8, the data is read into the A register in a byte-flipped form referenced to the value stored in ROM, meaning that the LSB of the ROM data byte is shifted into the

Table 5–8. Mode Register Control of GET Data Source

RAMROM	DATA SOURCE
0	Internal ROM
1	Internal RAM

A register first. The recommended sequence for preparing to GET from internal ROM is as follows:

1) Load the starting address of the first desired GET source into the A register.

2) Execute a LUAPS instruction, which performs all required initialization. The processor is now ready to execute a GET instruction starting at the address loaded in step 1.

3) When a nonsequential address is desired for a GET, repeat steps 1 and 2 for the new address.

When using dual channel synthesis, the speech address register changes as the channels alternate getting data from ROM. Because of this, the address along with the offset must be saved for the next GET on that channel. The steps above are extended for dual channels to the following:

4) Load the starting address of the desired GET source into the A register.

5) Execute a LUAPS instruction.

6) Load the offset of the address into the A register.

7) Execute a GET instruction. This adjusts the P/S buffer when needed. The processor is now ready to execute a GET instruction starting at the address + bit offset.

5.8.2 GET From Internal RAM

When the RAMROM bit is set in mode register 1, the data source for GET instructions is the internal RAM. As detailed in Section 5.8, the data is read into the A register in a byte-flipped form from the value stored in RAM, meaning that the LSB of the RAM data byte is shifted into the A register first.

The usage of the GET instruction while in RAM mode is somewhat more complicated than when the data source is from ROM because the burden of providing the address used to refresh the P/S buffer falls on the software.

When a GET instruction exhausts the P/S register, the value stored in the P/S buffer is loaded into the P/S register, and the GET instruction returns with the status set. When the next GET instruction is executed, the P/S buffer is loaded with the value stored in the RAM location pointed to by the X register.

The recommended sequence for preparing to GET from RAM is as follows:

1) Place the MSP50C3x in internal RAM mode.

2) Place the RAM address of the first desired GET source in the X register.

3) Execute LUAPS to initialize the counters in the MSP50C3x and to load the first byte from RAM into the P/S register. When preparing to GET from the RAM, the value in the A register during the LUAPS is unimportant. After the GET, the P/S buffer is empty and the P/S register is full.

4) Execute a dummy GET 8 instruction.

5) Load the X register with the RAM address of the second desired GET source.

6) The following sequence occurs when the first GET is executed:

   a) The P/S buffer is empty, so it is loaded with the value stored in the RAM location pointed to by the X register.

   b) The number of bits specified by the operand of the GET instruction is shifted into the A register.

7) On all subsequent GET operations, the status at completion should be tested by software. When the status is set, the P/S buffer is empty and the software should ensure that the X register contains the next desired address before the next GET is executed.

Following is a sample program that uses the GET from RAM:

```
        TCA     #20          ;turn on RAMROM bit
        TAMODE
        TCX     #10         ;set X register to start location
        LUAPS               ;init P/S register
        GET     8           ;do a dummy GET
        IXC                 ;increment X register to next location
LOOP    CLA
        GET     8           ;get data from RAM
        IXC                 ;increment X register to next location
        BR      LOOP        ;continue to read from RAM
        BR      LOOP
```

5.9 Generating Tones Using PCM

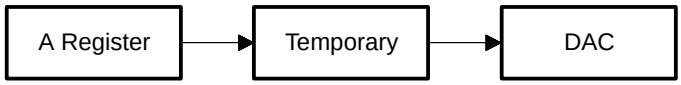
The MSP50C3x can generate speech and tones using pulse-code modulation (PCM) as well as LPC. When using PCM, a periodically sampled waveform can be loaded directly into the DAC, providing the ability to synthesize arbitrary waveforms. The value that is loaded into the DAC can be derived using a calculation, a table look-up, or a combination of the two methods. Smoothing between the data points is provided by the external low-pass filter.

When running PCM on one channel only, it should be run on channel 1. Then, PCM mode is enabled by setting the PCM1 bit to 1 and the LPC1 bit to 0 in mode register 1. When doing LPC + PCM, the PCM2 bit should be set to 1 in mode register 2 and the LPC1 bit should be set to 1 in mode register 1. Once PCM mode is enabled, the software must load the DAC with a value every 60 or 120 instruction cycles using the TASYN instruction.

5.9.1 Operation of the TASYN Instruction in PCM Mode

While in PCM mode, executing the TASYN instruction transfers the contents of the A register to the input of the DAC as shown in Figure 5–9. TASYN transfers the data in the A register to a temporary buffer register whose contents are periodically transferred to the DAC once every 120 instruction cycles.

Figure 5–9. Operation of TASYN in PCM Mode



The data in the A register should be in a modified 2's complement format, described as follows:

The A register is 16 bits long. When the contents of the A register are transferred to the DAC, the bits are interpreted as shown in Figure 5–10. The least significant bits (bits 0 and 1) are ignored and are generally cleared to zero. Bits 13 and 14 are the overflow bits. Bit 15 is the sign bit. When the sign bit is set to 1, the value being loaded to the DAC is negative. The remaining 11 bits of the A register (bits 2–12) contain magnitude data. The greatest magnitude is +512. Any greater magnitude is clipped. The relative weights of the magnitude bits are listed in Table 5–9.

Figure 5–10. Format of Data in A Register before TASYN

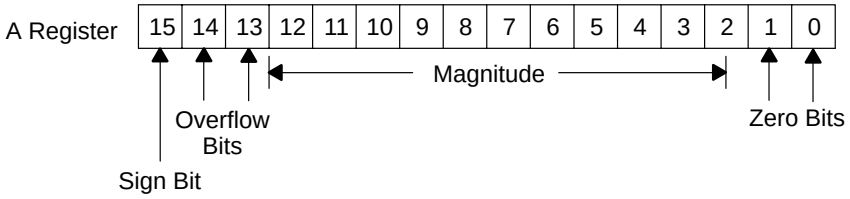


Table 5–9. Relative Weights of DAC Magnitude Bits

Bit Position	11	10	9	8	7	6	5	4	3	2
Relative Magnitude	256	128	64	32	16	8	4	2	1	1

While in PCM mode, the contents of the DAC are refreshed every 120 instruction cycles. The new data must be reloaded with TASYN instructions at an integer multiple of this rate. When the new data is not synchronous with the 1200-cycle refresh rate, samples may be missed or doubled, thereby resulting in tone deterioration.

There are two approaches to keeping the TASYN instruction synchronous to the DAC. With the PCM1 bit set to 1, the LPC1 bit cleared to 0, and the INT1 bit set to 1, a level-1 interrupt is generated every 120 instruction cycles. When the interrupt-service routine is longer than 120 instruction cycles, the interrupt is generated every 240 instruction cycles (or the next integer multiple of 120 greater than the length of the interrupt service routine). The second approach is to program a tight loop using exactly 120 or 240 instruction cycles per loop. This method works and avoids the instruction-cycle overhead associated with the interrupt but is more difficult to program reliably.

5.10 PCM + LPC

The MSP50C3x can also generate speech and tones using PCM + LPC. To accomplish this, the LPC must be run on channel 1 and the PCM must be on channel 2.

A simple way to do PCM + LPC is to use the standard single channel LPC routine (found in Appendix D) and add to it the PCM interrupt routine and the appropriate initializations. The PCM interrupt routine controls the sample rate. PCM + LPC mode can be enabled by setting the LPC1 bit (for channel 1) to 1, the ENA1 bit to 1, and the PCM2 bit (for channel 2) to 1. One thing to remember, as with dual LPC, the channel bit must be set or cleared to select the appropriate channel before the TASYN instruction is given.

Chapter 6

Customer Information

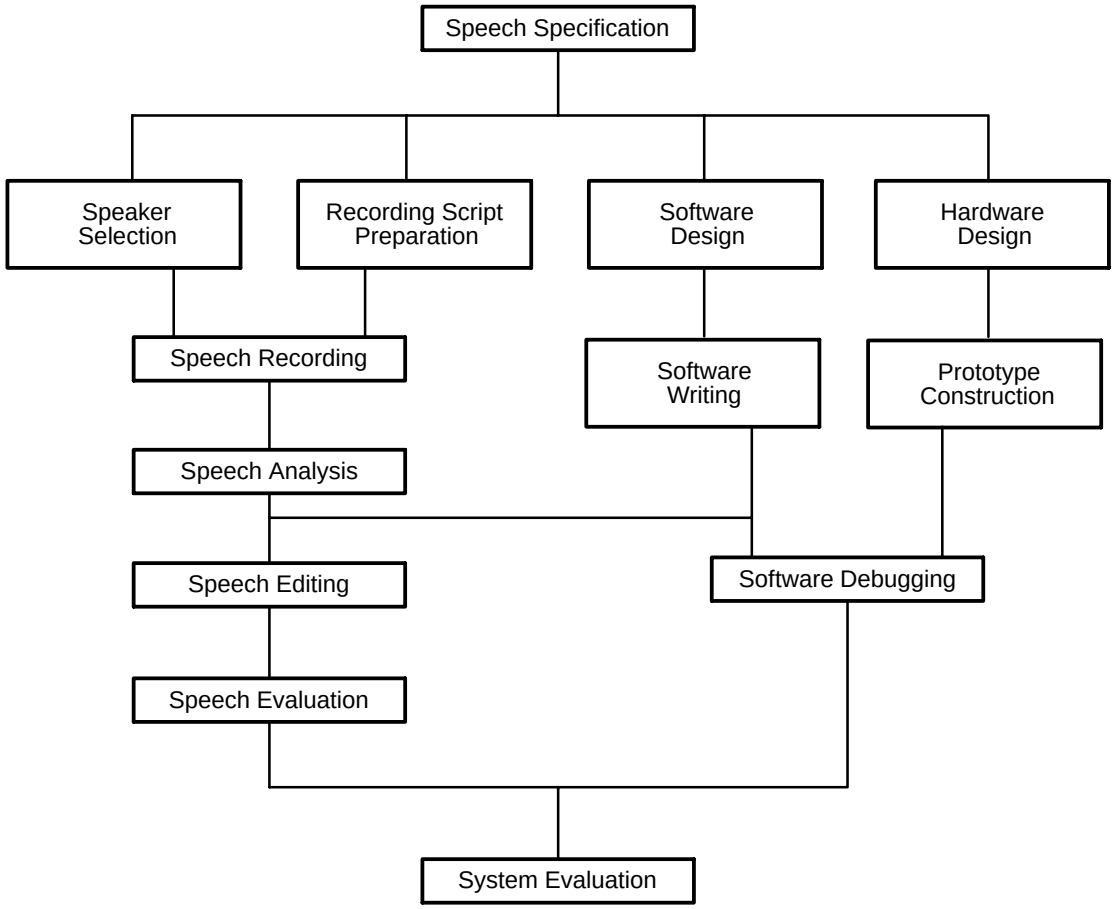
Customer information on development cycle organization, development and production sequence, mechanical information and packaging availability, ordering information, and example ordering forms are all included in this chapter.

Topic **Page**

6.1 Development Cycle

The MSP50x3x development cycle is more complex than microprocessor development, because it adds speech development to the normal microprocessor development cycle (Figure 6–1). The software design cycle is similar to that for other microprocessors. Speech development is discussed in Appendix A, *Script Preparation and Speech Development Tools*.

Figure 6–1. Speech Development Cycle



6.2 Summary of Speech Development/Production Sequence

The following is a summary of the speech development /production sequence:

1) For the Speech Development Group at TI to accept a custom device program, the customer must submit a new product release form (NPRF). This form describes the custom features of the device (e.g., customer information, prototype and production qualities, symbolization, etc.). Copies of blank NPRF forms can be found in subsections of 6.5.

2) TI generates the prototype photomask and processes, manufactures, and tests 25 packaged or 196 die initial prototype devices for shipment to the customer. Limited quantities of prototype devices in addition to the initial prototypes can be purchased for use in customer evaluation. All prototype devices are shipped against the following disclaimer: “It is understood that, for expediency purposes, the initial prototype devices (and any additional prototype devices purchased) were assembled on a prototype (i.e., not production-qualified) manufacturing line whose reliability has not been characterized. Therefore, the anticipated inherent reliability of these devices cannot be expressly defined.”

3) The customer verifies the operation and quality of these prototypes and responds with either written customer prototype approval or disapproval.

4) A nonrecurring mask charge that includes the initial prototype devices is incurred by the customer.

5) A minimum purchase may be required during the first year of production.

Note: Using Prototype Devices in Production Systems

Texas Instruments recommends that prototype devices not be used in production systems because their expected end-use failure rate is undefined but is predicted to be greater than standard qualified production.

6.3 Mechanical Information

Most of the MSP50C3x family is available in either a 16-pin plastic small-outline wide-body (SOWB) DW package, a 16-pin plastic dual-in-line N package (DIP), or in die form. The MSP50P34 is also available in a 40-pin plastic DIP NW package. The MSP50C37 is available in a 28-pin small-outline wide-body (SOWB) DW package or a 28-pin plastic dual-in-line N package (DIP).

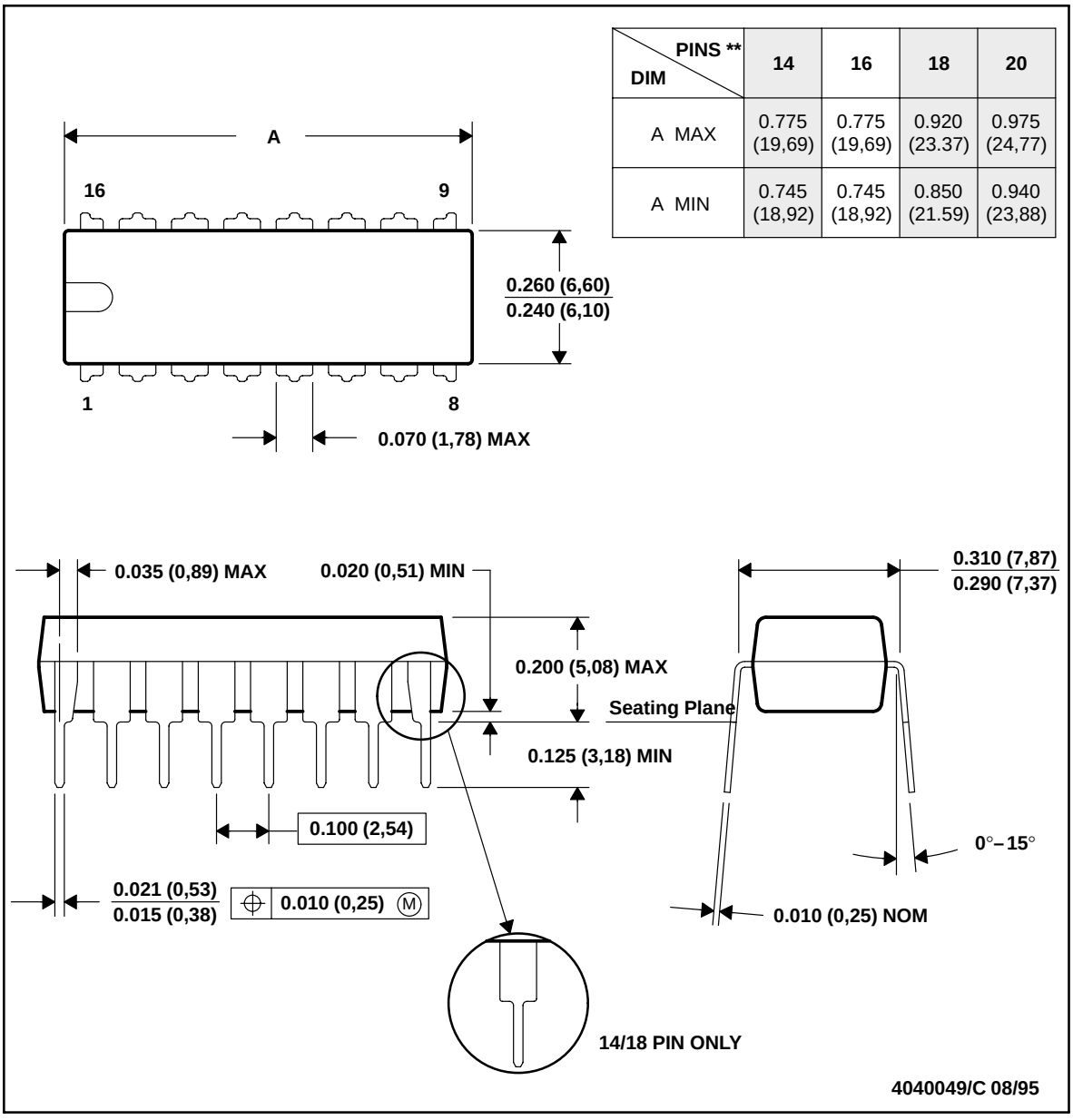
6.3.1 N and NW Plastic Dual-In-Line Packages

N (Figure 6–2) 16-pin plastic DIP packages of the MSP50C32/33/34, N (Figure 6–3) 28-pin plastic DIP packages of the MSP50x37, and the NW040 40-pin plastic DIP package of the MSP50P34 (Figure 6–4) consist of a circuit mounted in a lead frame and encapsulated within an electrically nonconductive plastic compound. The compound withstands soldering temperature with no deformation, and circuit performance characteristics remain stable when operated in high-humidity conditions. Once the leads are compressed and inserted, sufficient tension is provided to secure the package in the board during soldering. Leads require no additional cleaning or processing when used in the soldered assembly.

Figure 6–2. MSP50C32/33/34 16-Pin N Package

N (R-PDIP-T)** **PLASTIC DUAL-IN-LINE PACKAGE**

16 PIN SHOWN



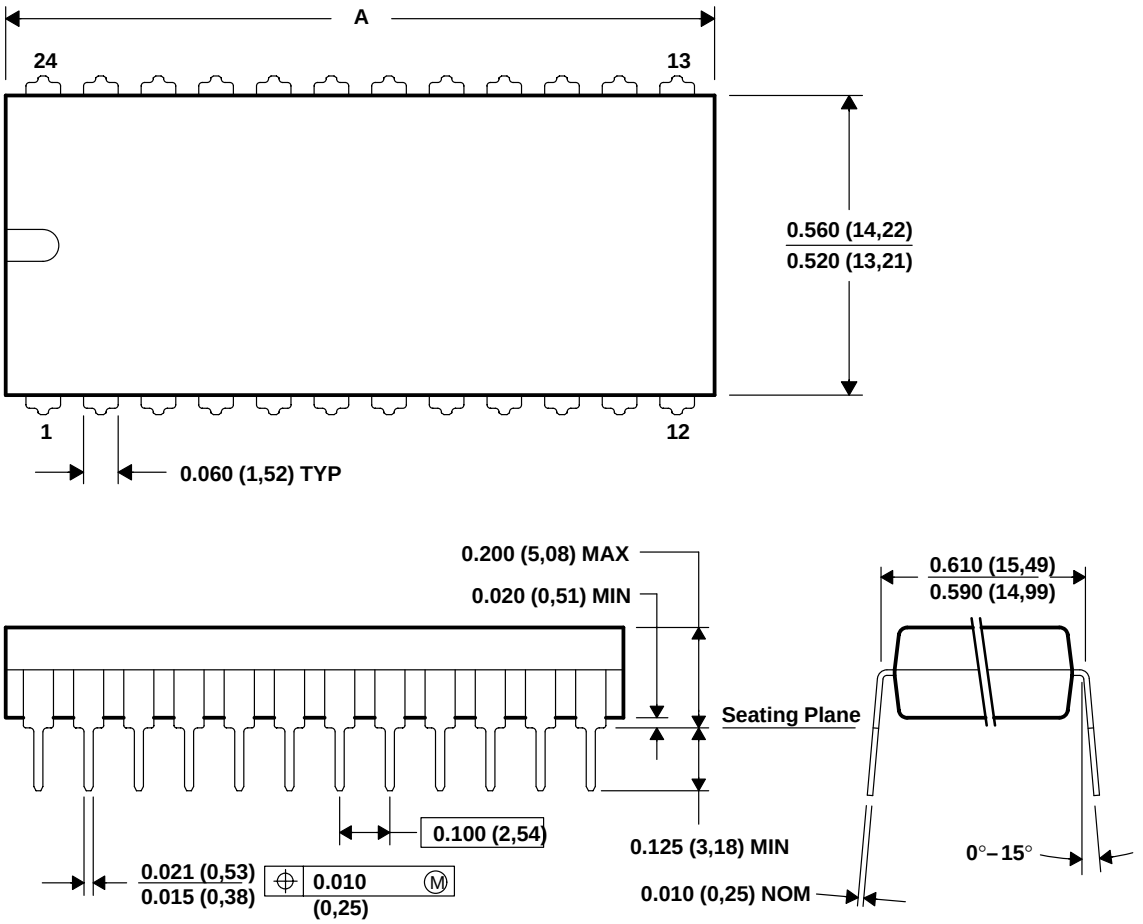
DIM \ PINS **	14	16	18	20
A MAX	0.775 (19,69)	0.775 (19,69)	0.920 (23.37)	0.975 (24,77)
A MIN	0.745 (18,92)	0.745 (18,92)	0.850 (21.59)	0.940 (23,88)

NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-001 (20 pin package is shorter then MS-001.)

Figure 6–3. MSP50x37 28-Pin N Package

N (R-PDIP-T)** **PLASTIC DUAL-IN-LINE PACKAGE**

24 PIN SHOWN



DIM \ PINS **	24	28	32	40	48	52
A MAX	1.270 (32,26)	1.450 (36,83)	1.650 (41,91)	2.090 (53,09)	2.450 (62,23)	2.650 (67,31)
A MIN	1.230 (31,24)	1.410 (35,81)	1.610 (40,89)	2.040 (51,82)	2.390 (60,71)	2.590 (65,79)

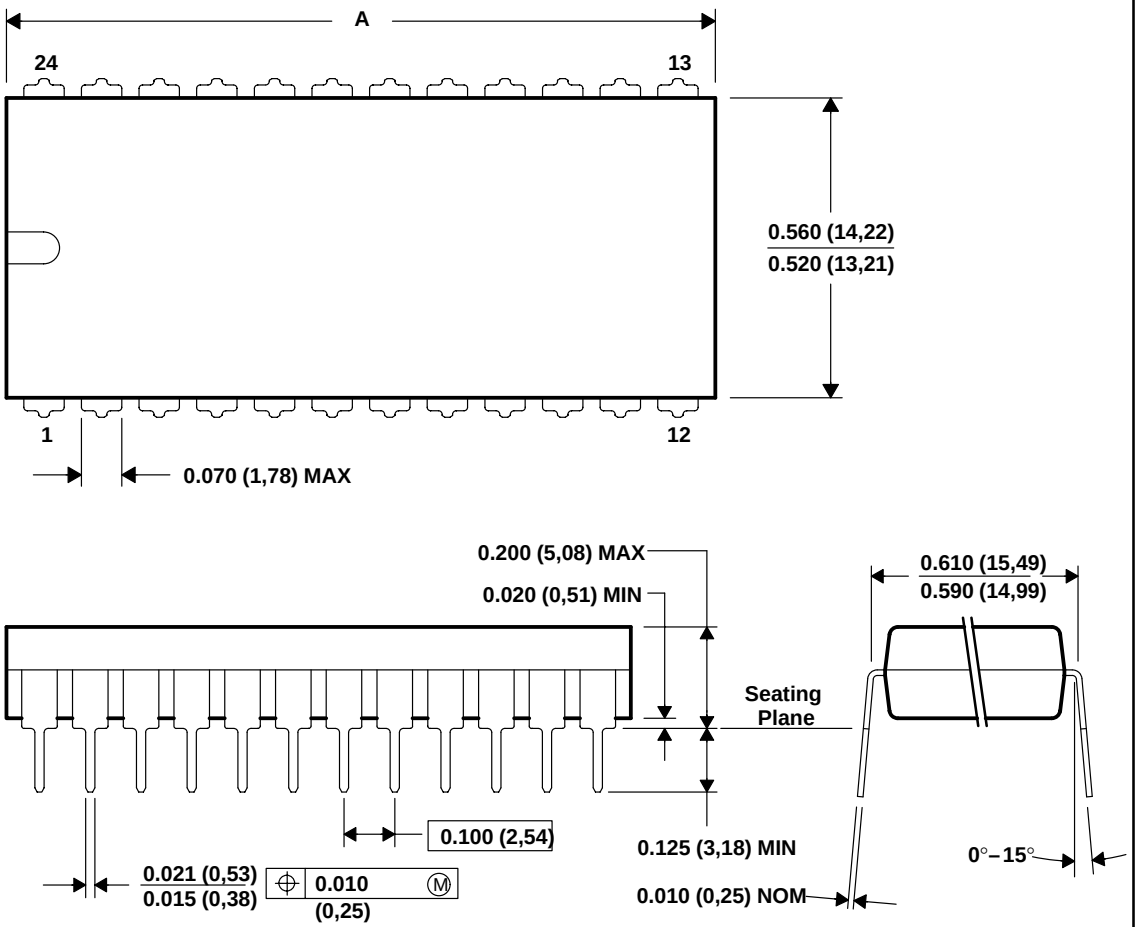
4040053/B 04/95

NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-011
D. Falls within JEDEC MS-015 (32 pin only)

Figure 6–4. MSP50P34 40-Pin NW Package

NW (R-PDIP-T)** **PLASTIC DUAL-IN-LINE PACKAGE**

24 PIN SHOWN



DIM \ PINS **	24	28	40	48
A MAX	1.290 (32,77)	1.440 (36,60)	2.090 (53,09)	2.450 (62,23)
A MIN	1.260 (32,00)	1.400 (35,56)	2.040 (51,82)	2.390 (60,71)

4040079/B 04/95

NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-011

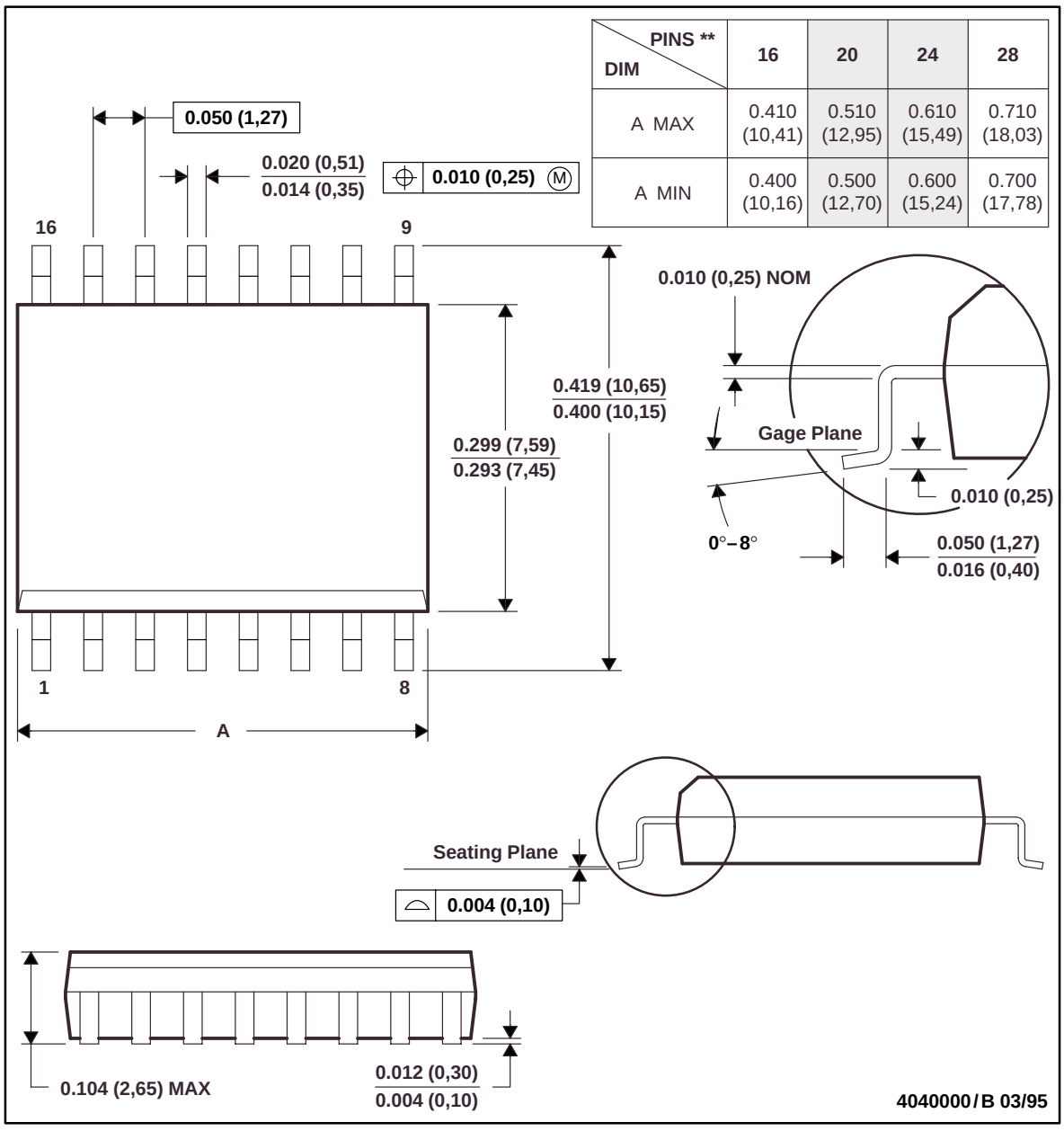
6.3.2 DW016 and DW028 Plastic Small-Outline Wide-Body (SOWB) Packages

The DW016 16-pin plastic SOWB package of the MSP50C32/33/34, MSP50P34, and the DW028 28-pin plastic SOWB package of the MSP50C37 and MSP50C37 (Figure 6–5) consist of a circuit mounted on a lead frame and encapsulated within a plastic compound. The compound withstands soldering temperature with no deformation, and circuit performance characteristics remain stable when operated in high-humidity conditions. Leads require no additional cleaning or processing when used in the soldered assembly.

Figure 6–5. MSP50C32/33/34/37 and MSP50P34/37 DW Package

DW (R-PDSO-G)** **PLASTIC SMALL-OUTLINE PACKAGE**

16 PIN SHOWN

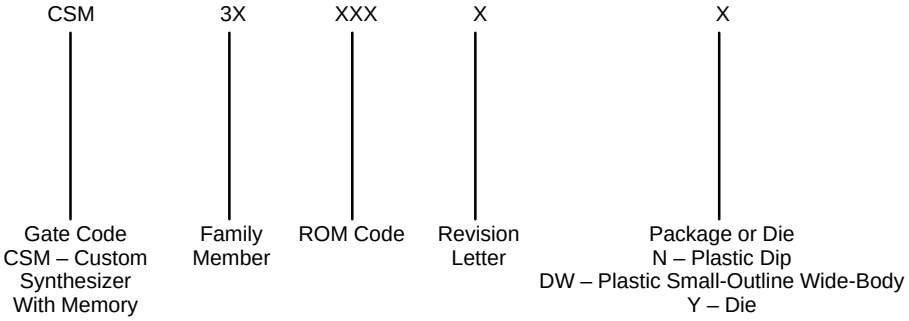


DIM \ PINS **	16	20	24	28
A MAX	0.410 (10,41)	0.510 (12,95)	0.610 (15,49)	0.710 (18,03)
A MIN	0.400 (10,16)	0.500 (12,70)	0.600 (15,24)	0.700 (17,78)

NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion not to exceed 0.006 (0,15).
D. Falls within JEDEC MS-013

6.4 Ordering Information

Because the MSP50C3x are custom devices, they receive a distinct identification as follows:



6.5 New Product Release Forms (MSP50C3x)

The new product release form is used to track and document all the steps involved in implementing a new speech code onto one of the parent speech devices. Blank forms are provided in subsections 6.5.1, *New Product Release Form for MSP50C32/33/34* and 6.5.2, *New Product Release Form for MSP50C37* (note that the addresses on these forms are subject to change). Copy the new product release forms (NPRF) provided or get one from your TI field sales office to initiate the implementation process. The next step is to complete Section 1. As seen on the blank forms, Section 1 allows you to choose the parent device for your particular code, as well as the options pertinent to the parent device you wish to use. Section 1 also allows you to choose your own customer part number used for ordering your parts. If no customer part number is indicated, then TI defaults to the CSM3xxxxxx part number for ordering purposes. Completion of the company name, project name, and option fields is mandatory. Completion of all other fields in Section 1 is optional. After completion of Section 1, you must submit the NPRF (along with your speech code) to the speech products group by way of your local TI field sales office.

Once the speech products group receives the speech code and the NPRF, you have completed the initial steps involved in implementing this code onto production devices. Since all parent speech devices are mask programmable, the speech code must first be converted into a format that the speech products mask vendor can use to generate this new mask. This format is called a PG output. Once this PG output is generated, the original speech code is reconstructed from the PG output file and sent back to you for recheck. This recheck ensures that the PG output file was generated correctly. Along with the reconstructed speech code, the NPRF is also returned to you with Section 2 completed by TI. In this section, TI assigns your own CSM3xxxxx part number and, in the case of packaged devices, TI also proposes a symbol format to you. If you wish to deviate from the suggested symbol format, you must consult TI for requested changes.

After you verify the reconstructed speech code and accept the proposed symbol format, you are required to sign Section 3 as authorization for TI to generate the mask, prototypes, and risk units in accordance with the pertinent purchase order. You then must send or fax the NPRF to the speech products group via the local TI field sales office. TI usually ships the prototypes to you approximately six weeks after receiving the NPRF with Section 3 signed. Once you receive these prototypes, you must verify the functionality of the prototypes, sign section 4, and send the NPRF (with Section 4 signed) back to TI. At this point, you can start ordering production units.

6.5.1 New Product Release Form for MSP50C32

```
                    NEW PRODUCT RELEASE FORM FOR MSP50C32

SECTION 1.  OPTION SELECTION
     This section is to be completed by the customer and sent to TI along with
     the microprocessor code and speech data.

Company:_____________________________   Division:_____________________________
Project Name:________________________   Purchase Order #:____________________
Management Contact: _____________________________  Phone:(___) _______________
Technical Contact : _____________________________  Phone:(___) _______________
Customer Part Number:_____________________________

D/A Output (check one):

           ___ 2 pin push-pull (2D)

           ___ Single pin double ended (1A)

Oscillator (check one):

           ___ Internal (Two ranges selected by software:
                       1) 15.36 MHz (14.89 MHz – 15.82 MHz)
                       2) 19.2 MHz (18.62 MHz – 19.77 MHz))

           ___ External

Package Type (check one):

           ___ N (16 pin)

           ___ SOWB (16 pin)
                  ___ Tube
                  ___ Reel

           ___ Die
*****************************************************************************

SECTION 2A. ASSIGNMENT OF TI PRODUCTION PART NUMBER
     This section is to be completed by TI.

TI Part Number:_______________

SECTION 2B. ASSIGNMENT OF SYMBOLIZATION FORMAT
     The section is to be completed by the customer.
     The first line of the symbolization is fixed. Except EIA#/Logo.
     The second and third lines are to be filled in by the customer.

Top Side Symbolization (16 pin 'N')
                          +----------------------+  LLLL: LOT TRACE CODE
                          |     ??? YMLLLLT      |    YM: DATE CODE
                          |    <optional 13 char> |     T: ASSY SITE
                          |    <optional 11 char> |   ???: TI EIA NO. or
                          +----------------------+          TI LOGO

For '16N' package, the customer may choose between 980 or the TI LOGO on the first
line.

Top Side Symbolization (16 pin 'SOWB')
                          +----------------------+  LLLL: LOT TRACE CODE
                          |     \T/ YMLLLLT      |    YM: DATE CODE
                          |    <optional  9 char> |     T: ASSY SITE
                          |    <optional  6 char> |   \T/: TI LOGO
                          +----------------------+
```

****

SECTION 3. AUTHORIZATION TO GENERATE MASKS, PROTOTYPES, AND RISK UNITS
This section is to be completed by the customer and sent to TI after the following criteria have been met:

1) The customer has verified that the TI computer generated data matches the original data.

2) The customer approves of the symbolization format in Section 2B. (Applies to packaged devices only)

I hereby certify that the TI generated verification data has been checked and found to be correct, and I authorize TI to generate masks, prototypes, and risk units in accordance with purchase order in section 1 above. In addition, in the instance that this is a packaged device, I also authorize TI to use the symbolization format illustrated in section 2B on all devices.

By:_____ Title:_____

Date:_____

(FAX this form to (214)480–7301. Attn: Code Release Team)

****

SECTION 4. APPROVAL OF PROTOTYPES AND AUTHORIZATION TO START PRODUCTION
This section is to be completed by the customer after prototype devices have been received and tested.

I hereby certify that the prototype devices have been received and tested and found to be acceptable, and I authorize TI to start normal production in accordance with purchase order #_____.

By:_____ Title:_____

Date:_____

****

Return to: Texas Instruments, Inc.
Attn: Code Release Team
P.O. Box 660199, M/S 8718
Dallas, TX 75266–0199

OR Fax to: (214)480–7301
Attn: Code Release Team

Have Questions?:
CALL: Code Release Team
(214)480–4444

OR E–MAIL: code–rel@msp.sc.ti.com

6.5.2 New Product Release Form for MSP50C33

```
                    NEW PRODUCT RELEASE FORM FOR MSP50C33

SECTION 1.  OPTION SELECTION
     This section is to be completed by the customer and sent to TI along with
     the microprocessor code and speech data.

Company:_____________________________   Division:_____________________________
Project Name:________________________   Purchase Order #:____________________
Management Contact: _____________________________  Phone:(___) _______________
Technical Contact : _____________________________  Phone:(___) _______________
Customer Part Number:_____________________________

D/A Output (check one):

           ___ 2 pin push-pull (2D)

           ___ Single pin double ended (1A)

Oscillator (check one):

           ___ Internal (Two ranges selected by software:
                       1) 15.36 MHz (14.89 MHz – 15.82 MHz)
                       2) 19.2 MHz (18.62 MHz – 19.77 MHz))

           ___ External

Package Type (check one):

           ___ N (16 pin)

           ___ SOWB (16 pin)
                  ___ Tube
                  ___ Reel

           ___ Die
*****************************************************************************

SECTION 2A. ASSIGNMENT OF TI PRODUCTION PART NUMBER
     This section is to be completed by TI.

TI Part Number:_______________

SECTION 2B. ASSIGNMENT OF SYMBOLIZATION FORMAT
     The section is to be completed by the customer.
     The first line of the symbolization is fixed. Except EIA#/Logo.
     The second and third lines are to be filled in by the customer.

Top Side Symbolization (16 pin 'N')
                          +----------------------+  LLLL: LOT TRACE CODE
                          |     ??? YMLLLLT      |    YM: DATE CODE
                          |    <optional 13 char> |     T: ASSY SITE
                          |    <optional 11 char> |   ???: TI EIA NO. or
                          +----------------------+          TI LOGO

For '16N' package, the customer may choose between 980 or the TI LOGO on the first
line.

Top Side Symbolization (16 pin 'SOWB')
                          +----------------------+  LLLL: LOT TRACE CODE
                          |     \T/ YMLLLLT      |    YM: DATE CODE
                          |    <optional  9 char> |     T: ASSY SITE
                          |    <optional  6 char> |   \T/: TI LOGO
                          +----------------------+
```

****

SECTION 3. AUTHORIZATION TO GENERATE MASKS, PROTOTYPES, AND RISK UNITS
This section is to be completed by the customer and sent to TI after the following criteria have been met:

1) The customer has verified that the TI computer generated data matches the original data.

2) The customer approves of the symbolization format in Section 2B. (Applies to packaged devices only)

I hereby certify that the TI generated verification data has been checked and found to be correct, and I authorize TI to generate masks, prototypes, and risk units in accordance with purchase order in section 1 above. In addition, in the instance that this is a packaged device, I also authorize TI to use the symbolization format illustrated in section 2B on all devices.

By:_____ Title:_____

Date:_____

(FAX this form to (214)480–7301. Attn: Code Release Team)

****

SECTION 4. APPROVAL OF PROTOTYPES AND AUTHORIZATION TO START PRODUCTION
This section is to be completed by the customer after prototype devices have been received and tested.

I hereby certify that the prototype devices have been received and tested and found to be acceptable, and I authorize TI to start normal production in accordance with purchase order #_____.

By:_____ Title:_____

Date:_____

****

Return to: Texas Instruments, Inc.
Attn: Code Release Team
P.O. Box 660199, M/S 8718
Dallas, TX 75266–0199

OR Fax to: (214)480–7301
Attn: Code Release Team

Have Questions?:
CALL: Code Release Team
(214)480–4444

OR E–MAIL: code–rel@msp.sc.ti.com

6.5.3 New Product Release Form for MSP50C34

NEW PRODUCT RELEASE FORM FOR MSP50C34

SECTION 1. OPTION SELECTION
This section is to be completed by the customer and sent to TI along with the microprocessor code and speech data.

Company:_____ Division:_____
Project Name:_____ Purchase Order #:_____
Management Contact: _____ Phone:(___) _____
Technical Contact : _____ Phone:(___) _____
Customer Part Number:_____

D/A Output (check one):

___ 2 pin push–pull (2D)

___ Single pin double ended (1A)

Oscillator (check one):

___ Internal (Two ranges selected by software:
1) 15.36MHz (14.89MHz – 15.82MHz)
2) 19.2MHz (18.62MHz – 19.77MHz))

___ External

Package Type (check one):

___ N (16 pin)

___ SOWB (16 pin)
___ Tube
___ Reel

___ Die
Number of I/O (Die Sales Only)
___ 10
___ 24

****

SECTION 2A. ASSIGNMENT OF TI PRODUCTION PART NUMBER
This section is to be completed by TI.

TI Part Number:_____

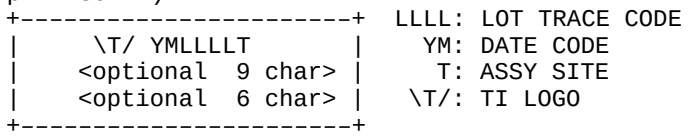
SECTION 2B. ASSIGNMENT OF SYMBOLIZATION FORMAT
The section is to be completed by the customer.
The first line of the symbolization is fixed. Except EIA#/Logo.
The second and third lines are to be filled in by the customer.

Top Side Symbolization (16 pin 'N')

```
                          +----------------------+  LLLL: LOT TRACE CODE
                          |     ??? YMLLLLT      |    YM: DATE CODE
                          |    <optional 13 char> |     T: ASSY SITE
                          |    <optional 11 char> |   ???: TI EIA NO. or
                          +----------------------+          TI LOGO
```

For '16N' package, the customer may choose between 980 or the TI LOGO on the first line.

Top Side Symbolization (16 pin 'SOWB')



****

SECTION 3. AUTHORIZATION TO GENERATE MASKS, PROTOTYPES, AND RISK UNITS
This section is to be completed by the customer and sent to TI after the following criteria have been met:

1) The customer has verified that the TI computer generated data matches the original data.

2) The customer approves of the symbolization format in Section 2B. (Applies to packaged devices only)

I hereby certify that the TI generated verification data has been checked and found to be correct, and I authorize TI to generate masks, prototypes, and risk units in accordance with purchase order in section 1 above. In addition, in the instance that this is a packaged device, I also authorize TI to use the symbolization format illustrated in section 2B on all devices.

By:_____ Title:_____

Date:_____

(FAX this form to (214)480–7301. Attn: Code Release Team)

****

SECTION 4. APPROVAL OF PROTOTYPES AND AUTHORIZATION TO START PRODUCTION
This section is to be completed by the customer after prototype devices have been received and tested.

I hereby certify that the prototype devices have been received and tested and found to be acceptable, and I authorize TI to start normal production in accordance with purchase order #_____.

By:_____ Title:_____

Date:_____

****

Return to: Texas Instruments, Inc.
Attn: Code Release Team
P.O. Box 660199, M/S 8718
Dallas, TX 75266–0199

OR Fax to: (214)480–7301
Attn: Code Release Team

Have Questions?:
CALL: Code Release Team
(214)480–4444

OR E–MAIL: code–rel@msp.sc.ti.com

6.5.4 New Product Release Form for MSP50C37

```
                          NEW PRODUCT RELEASE FORM
                                FOR MSP50C37

 SECTION 1.  OPTION SELECTION
     This section is to be completed by the customer and sent to TI along with
     the microprocessor code and speech data.

Company:_____________________________   Division:_____________________________
Project Name:________________________   Purchase Order #:____________________
Management Contact: _____________________________  Phone:(___) _____________
Technical Contact : _____________________________  Phone:(___) _____________
Customer Part Number:_____________________________

 POWER AMP (CHECK ONE)
           ___  DUAL OUTPUT
           ___  SINGLE OUTPUT

 Package Type (check one):
           ___ N (28 pin)
           ___ SOWB (28 pin)
                 ___ Tube
                 ___ Reel

 ******************************************************************************
SECTION 2A. ASSIGNMENT OF TI PRODUCTION PART NUMBER
     This section is to be completed by TI.

TI Part Number:_______________

SECTION 2B. ASSIGNMENT OF SYMBOLIZATION FORMAT
     The section is to be completed by the customer.
     The first line of the symbolization is fixed.
     The second and third lines are to be filled in by the customer.

Top Side Symbolization (28 pin 'N')
                           +----------------------+  LLLL: LOT TRACE CODE
                           |     ??? YMLLLLT      |    YM: DATE CODE
                           |    <optional 13 char> |     T: ASSY SITE
                           |    <optional 11 char> |   ???: TI EIA NO. or
                           +----------------------+          TI LOGO

For '28N' package, the customer may choose between 980 or the TI LOGO on the first
line.

Top Side Symbolization (28 pin 'SOWB')
                           +----------------------+  LLLL: LOT TRACE CODE
                           |     \T/ YMLLLLT      |    YM: DATE CODE
                           |   <optional  10 char> |     T: ASSY SITE
                           |    <optional  6 char> |   \T/: TI LOGO
                           +----------------------+

******************************************************************************

SECTION 3.  AUTHORIZATION TO GENERATE MASKS, PROTOTYPES, AND RISK UNITS
     This section is to be completed by the customer and sent to TI after
     the following criteria have been met:

1) The customer has verified that the TI computer generated data matches the
   original data.
```

2) The customer approves of the symbolization format in Section 2B.

I hereby certify that the TI generated verification data has been checked and found to be correct, and I authorize TI to generate masks, prototypes, and risk units in accordance with purchase order in section 1 above. In addition, I also authorize TI to use the symbolization format illustrated in section 2B on all devices.

By:_____ Title:_____

Date:_____

(FAX this form to (214)480–7301. Attn: Code Release Team)

****

SECTION 4. APPROVAL OF PROTOTYPES AND AUTHORIZATION TO START PRODUCTION
This section is to be completed by the customer after prototype devices have been received and tested.

I hereby certify that the prototype devices have been received and tested and found to be acceptable, and I authorize TI to start normal production in accordance with purchase order #_____.

By:_____ Title:_____

Date:_____

****

Return to: Texas Instruments, Inc.
Attn: Code Release Team
P.O. Box 660199, M/S 8718
Dallas, TX 75266–0199

OR Fax to: (214)480–7301
Attn: Code Release Team

Have Questions?:
CALL: Code Release Team
(214)480–4444

OR E–MAIL: code–rel@msp.sc.ti.com

6.5.5 New Product Release Form for MSP50P34

NEW PRODUCT RELEASE FORM FOR MSP50C34

SECTION 1. OPTION SELECTION
This section is to be completed by the customer and sent to TI along with the microprocessor code and speech data.

Company:_____ Division:_____
Project Name:_____ Purchase Order #:_____
Management Contact: _____ Phone:(___) _____
Technical Contact : _____ Phone:(___) _____
Customer Part Number:_____

D/A Output (check one):

___ 2 pin push–pull (2D)

___ Single pin double ended (1A)

Oscillator (check one):

___ Internal (Two ranges selected by software:
1) 15.36MHz (14.89MHz – 15.82MHz)
2) 19.2MHz (18.62MHz – 19.77MHz))

___ External

Package Type (check one):

___ N (16 pin)

___ SOWB (16 pin)
___ Tube
___ Reel

___ Die
Number of I/O (Die Sales Only)
___ 10
___ 24

****

SECTION 2A. ASSIGNMENT OF TI PRODUCTION PART NUMBER
This section is to be completed by TI.

TI Part Number:_____

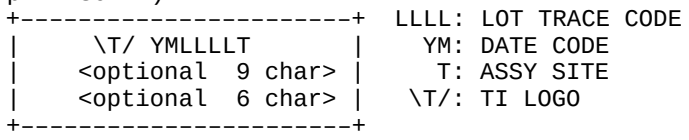
SECTION 2B. ASSIGNMENT OF SYMBOLIZATION FORMAT
The section is to be completed by the customer.
The first line of the symbolization is fixed. Except EIA#/Logo.
The second and third lines are to be filled in by the customer.

Top Side Symbolization (16 pin 'N')

```
                          +----------------------+  LLLL: LOT TRACE CODE
                          |     ??? YMLLLLT      |    YM: DATE CODE
                          |    <optional 13 char> |     T: ASSY SITE
                          |    <optional 11 char> |   ???: TI EIA NO. or
                          +----------------------+          TI LOGO
```

For '16N' package, the customer may choose between 980 or the TI LOGO on the first line.

Top Side Symbolization (16 pin 'SOWB')



****

SECTION 3. AUTHORIZATION TO GENERATE MASKS, PROTOTYPES, AND RISK UNITS
This section is to be completed by the customer and sent to TI after the following criteria have been met:

1) The customer has verified that the TI computer generated data matches the original data.

2) The customer approves of the symbolization format in Section 2B. (Applies to packaged devices only)

I hereby certify that the TI generated verification data has been checked and found to be correct, and I authorize TI to generate masks, prototypes, and risk units in accordance with purchase order in section 1 above. In addition, in the instance that this is a packaged device, I also authorize TI to use the symbolization format illustrated in section 2B on all devices.

By:_____ Title:_____

Date:_____

(FAX this form to (214)480–7301. Attn: Code Release Team)

****

SECTION 4. APPROVAL OF PROTOTYPES AND AUTHORIZATION TO START PRODUCTION
This section is to be completed by the customer after prototype devices have been received and tested.

I hereby certify that the prototype devices have been received and tested and found to be acceptable, and I authorize TI to start normal production in accordance with purchase order #_____.

By:_____ Title:_____

Date:_____

****

Return to: Texas Instruments, Inc.
Attn: Code Release Team
P.O. Box 660199, M/S 8718
Dallas, TX 75266–0199

OR Fax to: (214)480–7301
Attn: Code Release Team

Have Questions?:
CALL: Code Release Team
(214)480–4444

OR E–MAIL: code–rel@msp.sc.ti.com

Appendix A

Script Preparation and Speech Development Tools

Script preparation and speech development can be done either by the customer, a third-party consultant, or Texas Instruments. The following sections discuss major considerations during the script preparation and speech development process.

Topic **Page**

A.1 Script Generation

The first step in designing a system using LPC is the generation of a system specification, including a script. A coding table that yields the best data rate for the voice selected at the level of quality required needs to be selected. The voice that is selected needs to be tested to verify that it synthesizes well. TI can recommend voices, or new voices can be auditioned. Each coding table and voice has its characteristic data rate. This can be used with a word count to determine the amount of memory required to store the speech for the system.

There are three approaches to word use in a speech script: maximal reuse, partial concatenation, and no concatenation. The original synthetic products tended to use maximal reuse because memory was expensive and quality expectations were low. In maximal reuse systems, only one sample of each word is used regardless of the context in which the word occurs. The speech sounds robotic; it is flat with no inflection and there are delays between words. This yields good intelligibility at low data rates but does not provide natural quality. Natural speech has different inflections depending on the position of the word in a sentence and on whether the sentence is a question, statement, or an order. Additionally, all the words are run together; each word is changed by the last sound of the word before it and the first sound of the word after it.

Recording and synthesizing each phrase separately is the easiest way to get natural speech, but memory constraints often force compromises. An expert speech editor can look at a script that lists each word in each context in which it occurs and determine what contexts are similar enough to permit reuse.

Once a script is defined and the coding table selected, a recording script must be generated. For systems with partial reuse, this script must include a recording of each word in all necessary contexts. The other two approaches are more straight forward with a word list or a phrase list being all that is required.

A.1.1 Speaker Selection

While the scripts are being generated, a speaker should be selected to read the script. If possible, several voices should be recorded and analyzed, as all voices do not analyze equally well.

A.1.2 Speech Collection

Collecting speech for any medium, be it LPC or digital tape, requires significant effort. For high-quality speech, a recording studio and a professional speaker are required. It is possible to achieve acceptable quality with a professional

speaker and a quiet room. Nonprofessional speakers have trouble maintaining uniform pitch and energy levels, speaking properly, and providing the expression and inflection required. Additionally, the strain of speaking for long periods of time in a controlled manner is considerable. Nonprofessional speakers are best used only for prototyping.

During the session, it may be necessary to experiment with inflection and expression to find the best approach. Ideally, the person making the final decision on product content and aesthetics should be at the recording session. Leaving this task to others leads to repeat visits to the studio.

There are various techniques that can be used to ensure that the speech analyzes and synthesizes properly. Certain consonants need to be emphasized and spoken more clearly than they are in normal speech. The TI WINSDS development tool (see Section A.2, *Speech Development Tools*) provides immediate feedback for synthetic speech, making the speech collection process much easier for inexperienced users.

The actual collection process is fairly simple. The speech is converted into digital form and then analyzed with a computationally intensive algorithm. The WINSDS uses a TMS32031 digital signal processing chip to permit very rapid analysis. It consists of one board that fits into an AT class computer, software, and a documentation package. The board contains the TMS32031 and related circuitry, an analog-to-digital converter (ADC), a digital-to-analog converter (DAC), digital filters, amplifiers, and speech synthesizers to record and play digitized and synthetic speech. The software supports speech collection, analysis, and editing with extensive use of menus, windows, and other user-friendly interfaces. TI uses an algorithm that provides high-quality speech but also requires low levels of phase distortion. For this reason, digital audio tape, rather than cassette audio tape, should be used to collect speech.

A.1.3 LPC Editing

The speech often needs to be edited, both to define the boundaries of the words and to mask imperfections in the model, the analysis, and the speaker. Limited changes can be made to alter the stress and intonation, but the best quality is achieved by having the desired sound and inflection well recorded. Skillful editors can also reduce data rates significantly from those of analyzed speech. Good editing is a difficult skill to learn, requiring a good ear, linguistic knowledge, and a familiarity with computers. TI offers the WINSDS speech development system, which eases many of these tasks by analyzing the speech immediately to provide quick feedback and to permit rerecording if the synthetic speech does not offer the desired quality.

A.1.4 Pitfalls

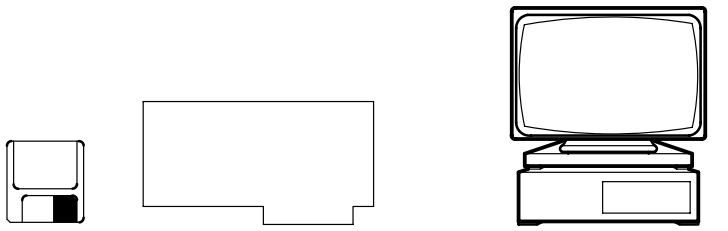
All speech interfaces, LPC or not, are human interfaces, so they are hard to design. Building a prototype system is often useful. The WINSDS supports quick prototyping.

LPC provides very low data rate speech by virtue of its close modeling of the human vocal tract. Nonspeech sounds, such as sound effects, may or may not be modeled accurately by LPC. The best way to find out whether or not a sound can be modeled is to record and analyze the sound on the WINSDS.

A.2 Speech Development Tools

The following figures show the various development tools and list the features of each.

Figure A–1. WINSDS



WINSDS PC (AT-Class Computer)

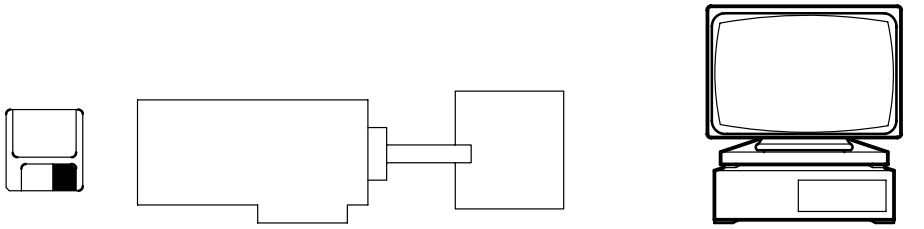
A.2.1 WINSDS Features

- ❑ High-speed speech analysis (real time)
- ❑ Graphical and numerical speech editing
- ❑ Microphone and line-level inputs
- ❑ Headphone and line outputs
- ❑ Supports TSP50C1x and MSP50C3x devices
- ❑ Requires an AT-class computer (a 100-MHz 486 is recommended) with a VGA card and Windows™ 3.1, 3.11, or Windows 95™
- ❑ Uses TMS32031 digital signal processor

Note: Required and Recommended Equipment

A hard disk drive is required and a tape backup system is strongly suggested for the WINSDS development system.

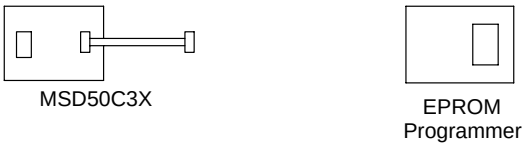
Figure A–2. EMU50C3x



A.2.2 EMU50C3x Features

- ❑ In-circuit emulation
- ❑ Hardware breakpoints
- ❑ Single step
- ❑ Examine/modify registers/memory
- ❑ Includes assembler
- ❑ Requires one ISA card slot in AT-class computers (386SX-33 or better recommended)
- ❑ Optional real-time trace

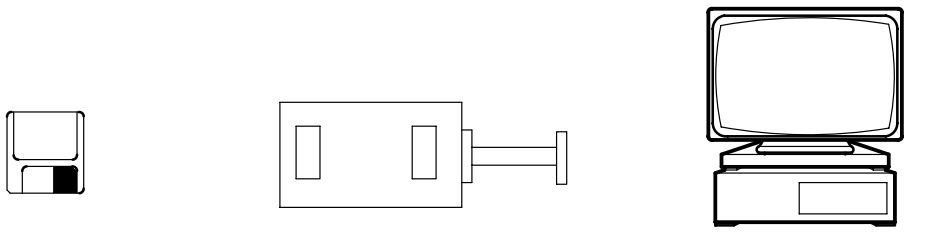
Figure A–3. MSD50C3x



A.2.3 MSD50C3x Features

- ❑ In-circuit emulation
- ❑ Small size, low power consumption
- ❑ Ideal for demonstration and field test
- ❑ Requires an industry-standard EPROM (such as a TMS27C256 or TMS27C512)

Figure A–4. OTP-PROG



A.2.4 OTP-PROG2 Features

- Programs the MSP50P34, MSP50P37, and the TSP50P11
- User-friendly software
- Requires a parallel port on an AT-class computer

Appendix B

MSP50C3x Versus TSP50C1x

This appendix contains information about switching from a TSP50C1x family device to a MSP50C3x famly device.

Topic **Page**

B.1 Summary of Changes from TSP50C1x Family

- Support dual LPC capability
  - RAM size is 256 locations
  - 32 12-bit RAM locations
  - Design part for double clock speed (19.2 MHz)
  - Double mode register to 2 memory mapped 8-bit registers
  - Add dual LPC algorithm
  - Add 2nd Pitch period counter
  - PPC < 0x200 sets the bit in the mode register instead of triggering an interrupt
  - Level 1 interrupt in LPC mode happens every 30 instruction cycles.
- Increase ROM size
  - Increase Program Counter register width to 16 bits
  - Increase A register width to 16 bits
  - Increase B register width to 16 bits
  - Increase SAR width to 16 bits
  - Increase ROM size to 64 Kbytes
- Miscellaneous
  - No support for external ROM mode
  - Improved internal oscillator design
  - Improved D/A output design
  - Designed part for increased V_{DD} range (3.3V – 6.5V)
  - Remap A port to 0xFCh – 0xFFh RAM range
  - Remap B port to 0xF8h – 0xFBh RAM range
  - Added wakeup function
  - 3 oscillator modes (internal, external, and crystal/ceramic)
  - Status gets set by SETOFF, $\overline{\text{INIT}}$ terminal low, and interrupt
  - Fix DECMN instruction so that true decrement happens on 12-bit RAM
  - Fix ANDCM instruction so that operation on 12-bit RAM location does not affect the upper 4 bits
  - SETOFF does not change the I/O state
  - Memory map mode registers
  - I/O ports are paged in and out of the memory map by the IO_MAP bit in the Mode Register
  - Interrupt vector location is changed

B.2 Upgrading a TSP50C1x Program to a MSP50C3x Program

This section documents the changes necessary to upgrade a TSP50C1x program to a MSP50C3x program. This file assumes that none of the additional features of the MSP50C3x are being used. for example, the wakeup function does not exist on the TSP50C1x, so code changes necessary to accommodate the wakeup function are not included.

Assumptions:

- ❏ The wakeup function is not used
- ❏ Only the A and B ports are used
- ❏ Only 128 RAM locations or less are used
- ❏ Only 16-K ROM locations are used
- ❏ All program code is in the lower 4 Kbytes of ROM

B.2.1 Normal Operation

Remap interrupt vectors

The interrupt vectors are I/O mapped in a different sequence on the MSP50C3x than they are on the TSP50C1x.

Table B–1.Interrupt Vectors for the TSP50C1x and the MSP50C3x

Address	Type	TSP50C1x	MSP50C3x
0x0010	Level 2 Interrupt	PCM = 0, LPC = 1	PCM = 0, LPC = 0
0x0012	Level 2 Interrupt	PCM = 0, LPC = 0	PCM = 0, LPC = 1
0x0014	Level 2 Interrupt	PCM = 1, LPC = 1	PCM = 1, LPC = 0
0x0016	Level 2 Interrupt	PCM = 1, LPC = 0	PCM = 1, LPC = 1
0x0018	Level 1 Interrupt	PCM = 0, LPC = 1	PCM = 0, LPC = 0
0x001A	Level 1 Interrupt	PCM = 0, LPC = 0	PCM = 0, LPC = 1
0x001C	Level 1 Interrupt	PCM = 1, LPC = 1	PCM = 1, LPC = 0
0x001E	Level 1 Interrupt	PCM = 1, LPC = 0	PCM = 1, LPC = 1

Remap I/O ports

The I/O ports are I/O mapped to different RAM locations on the MSP50C3x than they are on the TSP50C1x.

Table B–2.I/O Ports for the TSP50C1x and the MSP50C3x

Port	TSP50C1x Address	MSP50C3x Address
Port A Input Register	0x80	0xFC
Port A Pullup Register	0x81	0xFD
Port A Tristate Register	0x82	0xFE
Port A Output Register	0x83	0xFF
Port B Input Register	0x84	0xF8
Port B Pullup Register	0x85	0xF9
Port B Tristate Register	0x86	0xFA
Port B Output Register	0x87	0xFB

Adjust Code for 16-bit register size

The A and B ports are 14 bits wide on the TSP50C1x family and are 16 bits wide on the MSP50C3x family. Any code depending on the 14-bit size of the TSP50C1x family must be adjusted. For example, a common way of excising bits at the most significant portion of the data word is to left shift the data in the A register, then right shift to properly justify the data. An additional 2 bits of shift would be needed on the MSP50C3x family.

Select clock speed

On the TSP50C1x family, the clock speed is fixed. On the MSP50C3x family, the internal clock speed is switchable. If the internal clock option is used on the MSP50C3x, then the speed needs to be set in software.

Adjust code for 12-bit RAM location

On the TSP50C1x family, RAM locations 0x10h through 0x1Fh are 8 bits wide. On the MSP50C3x family, these RAM locations are 12 bits wide. Any use of these RAM locations that assumes an 8-bit width needs to be fixed.

Adjust for clock speed

The MSP50C3x family operates at a clock speed that is twice that of the TSP50C1x family. Any clock speed dependence must be adjusted.

TSP60C18 support removed

The support for the TSP60C18/81 speech ROM has been removed. The external ROM bit has been removed from the mode register. If it is desired to use an external ROM, the interface must be done in software.

Adjust code for 12-bit X register size

On the TSP50C1x family, the X register is 8 bits wide. On the MSP50C3x family, the X register is 12 bits wide. Any use of the X register that assumes an 8-bit width needs to be fixed.

Converting TSP50C19 code

There is no paging on the MSP50C3x family.

B.2.2 LPC

Adjust code for different LPC interrupt

On the TSP50C1x family, the LPC interrupt was caused by the underflow of the pitch period counter. On the MSP50C3x family, the LPC interrupt is invoked at a 20-kHz rate by the system clock. The underflow of the pitch period counter causes a bit to be set to 1 in mode register 2.

Modify the MSP50C3x code to periodically poll the mode register for the PPC bit to be set and then clear the bit and either branch or call the interpolation routine.

Clear Memory-Mapped Register prior to turning on LPC

RAM locations 0xFF0h, 0xFF1h, 0xFF2h, and 0xFF4h should be cleared to 0 prior to turning on the LPC. The RAM location 0xFF3h should be set to 0x162h.

Replace the excitation function with the MSP50C3x excitation function.

Thie excitation function is the same as the TSP50C1x excitation function except that the unvoiced portion (#3A80) is sign extended to 16 bits (#FA80). The voiced portion of the excitation function may need to be multiplied by 2 to get the same volume as the TSP50C1x.

Move the excitation function

In TSP50C1x programs (except the TSP50C19), the excitation function needs to be moved from 0x4000 to 0x8000 when using the MSP50C33 or to 0x10000 when using the MSP50C34.

Note: TSP50C19

On the TSP50C19, the excitation function already has been moved to 0x8000h.

B.2.3 PCM

Adjust data for the different register size

On the TSP50C1x family, the TASYN instruction loaded the PCM data to a register 14 bits wide. The least significant 2 bits were unused and the most significant 2 bits were used for sign extension.

On the MSP50C3x family, the TASYN instruction loads the PCM data to a register 16 bits wide. The least-significant two bits are unused. The most significant two bits are used for overflow with the third most-significant bit being used for sign extension.

Appendix C

MSP50C3x Sample Dual Synthesis Program

Topic Page

C.1 MSP50C3x Sample Dual Synthesis Program

The following is a sample speech synthesis program that runs on the MSP50C3x family of speech synthesis microprocessors. This program uses both LPC channels to speak an echo.

```
dualdemo.asm   TSP50CXX Assembler Version 2.1d       Wed Oct 09 15:33:52 1996
                                                                  PAGE 0001
0001                    OPTION BUNLIST,DUNLIST,PAGEOF
0002                    WIDE
0003
0004                    TABSIZE 8
0005            *-------------------------------------------------------*
0006            *   MSP50C3x DUAL LPC SYNTHESIS PROGRAM                 *
0007            *                                                       *
0008            *   This is a sample speech synthesis program           *
0009            *   which runs on the MSP50C3x family of speech         *
0010            *   synthesis microprocessors.  It uses both            *
0011            *   LPC channels to speak an echo.                      *
0012            *                                                       *
0013            *   This program uses the D6 Coding table format.       *
0014            *                                                       *
0015            *-------------------------------------------------------*
0016            *   COPYRIGHT 1995       TI – SPEECH PRODUCTS           *
0017            *-------------------------------------------------------*
0018            *-------------------------------------------------------*
0019            *      ADDRESS LABELS FOR SYNTHESIS ROUTINE             *
0020            *-------------------------------------------------------*
0021            ****************************************
0022            *      SYNTHESIZER RAM LOCATIONS
0023            ****************************************
0024            * NOTE – NEVER CHANGE LOCATIONS #01 TO #1F
0025            *
0026      0000 TEMP        EQU     #00          ;Temp storage
0027      0001 EN_A        EQU     #01          ;Energy working value
0028      0002 K12_A       EQU     #02          ;K12 Working Value – Ch. 1
0029      0003 K11_A       EQU     #03          ;K11 Working Value – Ch. 1
0030      0004 K10_A       EQU     #04          ;K10 Working Value – Ch. 1
0031      0005 K9_A        EQU     #05          ;K9 Working Value – Ch. 1
0032      0006 K8_A        EQU     #06          ;K8 Working Value – Ch. 1
0033      0007 K7_A        EQU     #07          ;K7 Working Value – Ch. 1
0034      0008 K6_A        EQU     #08          ;K6 Working Value – Ch. 1
0035      0009 K5_A        EQU     #09          ;K5 Working Value – Ch. 1
0036      000A K4_A        EQU     #0A          ;K4 Working Value – Ch. 1
0037      000B K3_A        EQU     #0B          ;K3 Working Value – Ch. 1
0038      000C K2_A        EQU     #0C          ;K2 Working Value – Ch. 1
0039      000D K1_A        EQU     #0D          ;K1 Working Value – Ch. 1
0040      000E C1          EQU     #0E          ;C1 Parameter
0041      000F C2          EQU     #0F          ;C2 Parameter
0042      0010 TEMP2       EQU     #10          ;Temp storage
0043      0011 EN_B        EQU     #11          ;Energy working value
0044      0012 K12_B       EQU     #12          ;K12 Working Value – Ch. 2
0045      0013 K11_B       EQU     #13          ;K11 Working Value – Ch. 2
0046      0014 K10_B       EQU     #14          ;K10 Working Value – Ch. 2
0047      0015 K9_B        EQU     #15          ;K9 Working Value – Ch. 2
0048      0016 K8_B        EQU     #16          ;K8 Working Value – Ch. 2
0049      0017 K7_B        EQU     #17          ;K7 Working Value – Ch. 2
0050      0018 K6_B        EQU     #18          ;K6 Working Value – Ch. 2
0051      0019 K5_B        EQU     #19          ;K5 Working Value – Ch. 2
0052      001A K4_B        EQU     #1A          ;K4 Working Value – Ch. 2
0053      001B K3_B        EQU     #1B          ;K3 Working Value – Ch. 2
```

```
0054      001C K2_B        EQU     #1C          ;K2 Working Value – Ch. 2
0055      001D K1_B        EQU     #1D          ;K1 Working Value – Ch. 2
0056      001E C3          EQU     #1E          ;C3 Parameter
0057      001F TEMP4       EQU     #1F          ;Temp storage
0058           *
0059           *---Channel 1 interpolation endpoints
0060           *
0061      0020 EN_New_A    EQU     #20          ;ENERGY New Value MSB
0062      0021 EN_Old_A    EQU     #21          ;ENERGY Current Value MSB
0063      0022 PH_New_A    EQU     #22          ;PITCH New Value MSB
0064      0024 PH_Old_A    EQU     #24          ;PITCH Current Value MSB
0065      0026 K1_New_A    EQU     #26          ;K1 New Value MSB
0066      0028 K1_Old_A    EQU     #28          ;K1 Current Value MSB
0067      002A K2_New_A    EQU     #2A          ;K2 New Value MSB
0068      002C K2_Old_A    EQU     #2C          ;K2 Current Value MSB
0069      002E K3_New_A    EQU     #2E          ;K3 New Value MSB
0070      002F K3_Old_A    EQU     #2F          ;K3 Current Value MSB
0071      0030 K4_New_A    EQU     #30          ;K4 New Value MSB
0072      0031 K4_Old_A    EQU     #31          ;K4 Current Value MSB
0073      0032 K5_New_A    EQU     #32          ;K5 New Value
0074      0033 K5_Old_A    EQU     #33          ;K5 Current Value
0075      0034 K6_New_A    EQU     #34          ;K6 New Value
0076      0035 K6_Old_A    EQU     #35          ;K6 Current Value
0077      0036 K7_New_A    EQU     #36          ;K7 New Value
0078      0037 K7_Old_A    EQU     #37          ;K7 Current Value
0079      0038 K8_New_A    EQU     #38          ;K8 New Value
0080      0039 K8_Old_A    EQU     #39          ;K8 Current Value
0081      003A K9_New_A    EQU     #3A          ;K9 New Value
0082      003B K9_Old_A    EQU     #3B          ;K9 Current Value
0083      003C K10_New_A   EQU     #3C          ;K10 New Value
0084      003D K10_Old_A   EQU     #3D          ;K10 Current Value
0085           *
0086           *---Channel 2 interpolation endpoints
0087           *
0088      0040 EN_New_B    EQU     #40          ;ENERGY New Value MSB
0089      0041 EN_Old_B    EQU     #41          ;ENERGY Current Value MSB
0090      0042 PH_New_B    EQU     #42          ;PITCH New Value MSB
0091      0044 PH_Old_B    EQU     #44          ;PITCH Current Value MSB
0092      0046 K1_New_B    EQU     #46          ;K1 New Value MSB
0093      0048 K1_Old_B    EQU     #48          ;K1 Current Value MSB
0094      004A K2_New_B    EQU     #4A          ;K2 New Value MSB
0095      004C K2_Old_B    EQU     #4C          ;K2 Current Value MSB
0096      004E K3_New_B    EQU     #4E          ;K3 New Value MSB
0097      004F K3_Old_B    EQU     #4F          ;K3 Current Value MSB
0098      0050 K4_New_B    EQU     #50          ;K4 New Value MSB
0099      0051 K4_Old_B    EQU     #51          ;K4 Current Value MSB
0100      0052 K5_New_B    EQU     #52          ;K5 New Value
0101      0053 K5_Old_B    EQU     #53          ;K5 Current Value
0102      0054 K6_New_B    EQU     #54          ;K6 New Value
0103      0055 K6_Old_B    EQU     #55          ;K6 Current Value
0104      0056 K7_New_B    EQU     #56          ;K7 New Value
0105      0057 K7_Old_B    EQU     #57          ;K7 Current Value
0106      0058 K8_New_B    EQU     #58          ;K8 New Value
0107      0059 K8_Old_B    EQU     #59          ;K8 Current Value
0108      005A K9_New_B    EQU     #5A          ;K9 New Value
0109      005B K9_Old_B    EQU     #5B          ;K9 Current Value
0110      005C K10_New_B   EQU     #5C          ;K10 New Value
0111      005D K10_Old_B   EQU     #5D          ;K10 Current Value
0112           *
0113           *
0114           *    LPC status variable locations
0115           *
```

```
0116      0060 TIMER       EQU     #60          ;Stored Timer value for update
0117      0061 SCALE       EQU     #61          ;Timer register value used in INTP
0118      0062 FLAGS       EQU     #62          ;General synthesis flags
0119      0063 FLAGS_1     EQU     #63          ;Channel 1 synthesis flags
0120      0064 FLAGS_2     EQU     #64          ;Channel 2 synthesis flags
0121      0065 PHRASE1     EQU     #65          ;Phrase number for channel 1
0122      0066 PHRASE2     EQU     #66          ;Phrase number for channel 2
0123           *
0124           *   Channel 1 address for speech address register
0125           *
0126      0067 ADR_MSB_1   EQU     #67          ;MSB of channel 1 address
0127      0068 ADR_LSB_1   EQU     #68          ;LSB of channel 1 address
0128      0069 OFFSET_1    EQU     #69          ;Bit offset into channel 1 address
0129           *
0130           *   Channel 2 address for speech address register
0131           *
0132      006A ADR_MSB_2   EQU     #6A          ;MSB of channel 2 address
0133      006B ADR_LSB_2   EQU     #6B          ;LSB of channel 2 address
0134      006C OFFSET_2    EQU     #6C          ;Bit offset into channel 2 address
0135           *
0136           *   Phrase addresses for Channels 1 & 2
0137           *
0138      006D ADR_MSB1    EQU     #6D          ;MSB of channel 1 address
0139      006E ADR_LSB1    EQU     #6E          ;LSB of channel 1 address
0140      006F ADR_MSB2    EQU     #6F          ;MSB of channel 2 address
0141      0070 ADR_LSB2    EQU     #70          ;LSB of channel 2 address
0142           ************************************************************
0143           *   Constant Definitions
0144           ************************************************************
0145           *
0146           *   Bit Size of Speech parameters
0147           *
0148      0004 EBITS       EQU     4            ;Number of Energy Bits
0149      0007 PBITS       EQU     7            ;Number of Pitch Bits
0150      0001 RBITS       EQU     1            ;Number of Repeat Bits
0151      0006 K1BITS      EQU     6            ;Number of K1 Bits
0152      0006 K2BITS      EQU     6            ;Number of K2 Bits
0153      0005 K3BITS      EQU     5            ;Number of K3 Bits
0154      0005 K4BITS      EQU     5            ;Number of K4 Bits
0155      0004 K5BITS      EQU     4            ;Number of K5 Bits
0156      0004 K6BITS      EQU     4            ;Number of K6 Bits
0157      0004 K7BITS      EQU     4            ;Number of K7 Bits
0158      0003 K8BITS      EQU     3            ;Number of K8 Bits
0159      0003 K9BITS      EQU     3            ;Number of K9 Bits
0160      0003 K10BITS     EQU     3            ;Number of K10 Bits
0161      0000 K11BITS     EQU     0            ;Number of K11 Bits
0162      0000 K12BITS     EQU     0            ;Number of K12 Bits
0163           *
0164           *   Prescale Values
0165           *
0166           *   PSvalue = Round(Samples * 2 * 30)/128
0167           *
0168           *   This comes from the fact that samples come every 30
0169           *   instruction cycles in LPC mode.  The factor of 2
0170           *   accounts for the cycle steal that happens in
0171           *   LPC mode.  When not in LPC mode, samples come
0172           *   every 60 instruction cycles, so it comes out the
0173           *   same.  The 128 divider is the full scale Timer
0174           *   register value.  The 0.5 addition is there
0175           *   to take care of rounding.
0176           *
0177      00C8 SAMPLES     EQU     200          ;Samples per frame
```

```
0178      005D PSVALUE     EQU     (SAMPLES*60/128)+(1/2)   ;Prescale Value
0179           *
0180           *   Device Constants
0181           *
0182      0F61 C1_Value    EQU     #F61          ;C1 Value (low pass filter value)
0183      0B67 C2_Value    EQU     #B67          ;C2 Value         ″
0184      03FF C3_Value    EQU     #3FF          ;C3 Value (channel scaling value)
0185      00FF MAX_RAM     EQU     #FF           ;Highest RAM location
0186      0032 ISAMPLE     EQU     #32           ;Instruction cycles per sample
0187           *
0188           *   Special Energy Values
0189           *
0190      000F ESTOP       EQU     15            ;Stop code
0191      0000 ESILENCE    EQU     0             ;Silence Code
0192           *
0193           *   Special Pitch Value
0194           *
0195      0000 PUnVoiced   EQU     0             ;UnVoiced Frame Code
0196      0018 UNV_PITCH   EQU     #18           ;Pitch used for Unvoiced Frames
0197           *
0198           *   End of sentence signal
0199           *
0200      00FF StopWord    EQU     #FF           ;Sentence stop code
0201           *
0202           *   FLAGS_1 and FLAGS_2 bit usage (and Set Masks)
0203           *
0204      0001 Int_Off     EQU     #01           ;Interpolation inhibit flag
0205      0002 RepeatFlag  EQU     #02           ;Repeat Frame = 1
0206      0004 Update_Flg  EQU     #04           ;Set high on update
0207      0008 Sil_Flg_New EQU     #08           ;New frame is silent = 1
0208      0010 Unv_Flg_New EQU     #10           ;New frame is unvoiced = 1
0209      0020 Int_Inh     EQU     #20           ;Change between Voice and UnV=1
0210      0040 Sil_Flg_Old EQU     #40           ;Current frame is silent = 1
0211      0080 Unv_Flg_Old EQU     #80           ;Current frame is unvoiced = 1
0212           *
0213           *   FLAGS bit usage (and Set Masks)
0214           *
0215      0001 STOP1       EQU     #01           ;Stop flag for channel 1
0216      0002 STOP2       EQU     #02           ;Stop flag for channel 2
0217      0004 CH1END      EQU     #04           ;end of channel 1 phrase
0218      0008 CH2END      EQU     #08           ;end of channel 2 phrase
0219      0010 Ch1Inh_1    EQU     #10           ;channel 1 inhibit interpolation bit 1
0220      0020 Ch1Inh_2    EQU     #20           ;channel 1 inhibit interpolation bit 2
0221      0040 Ch2Inh_1    EQU     #40           ;channel 2 inhibit interpolation bit 1
0222      0080 Ch2Inh_2    EQU     #80           ;channel 2 inhibit interpolation bit 2
0223           *
0224           *   MODE Register Bit Definitions
0225           *
0226      0001 INT1        EQU     #01           ;Enable Level 1 interrupt
0227      0002 LPC         EQU     #02           ;Enable LPC synthesis
0228      0004 PCM         EQU     #04           ;Enable PCM synthesis
0229      0008 INT2        EQU     #08           ;Enable Level 2 interrupt
0230      0010 EXTROM      EQU     #10           ;Set external ROM mode
0231      0020 RAMROM      EQU     #20           ;Enable GETs from RAM
0232      0040 MASTER      EQU     #40           ;Master/Slave Toggle
0233      0080 UNV         EQU     #80           ;Enable Unvoiced excitation
0234           *
0235           *  MODE Register 2 bit definitions
0236           *
0237      0001 PPC1        EQU     #01           ;PPC for channel 1
0238      0008 PPC2        EQU     #08           ;PPC for channel 1
0239      0010 CHANNEL     EQU     #10           ;channel select (0=ch.1, 1=ch.2)
```

```
0240           *
0241           *  SILENCE phrase defined in SPEECH lookup table
0242           *     !! This number must be changed to match the location of the
0243           *        silence phrase in the lookup table. When channel 1 finishes
0244           *        speaking before channel 2, we insert silence on channel 1
0245           *        because channel 2 cannot be on without channel 1!!
0246           *
0247      0000 SIL         EQU     0
0248
0249           ************************************************************
0250           *   Start of program
0251           ************************************************************
0252 0000                 AORG    #0000
0253        84             SBR     GOGO          ;Power up Vector
0254 0001   84             SBR     GOGO          ;   (twice makes unconditional)
0255
0256 0002   84             SBR     GOGO          ;Wake up vector
0257 0003   84             SBR     GOGO          ;   (twice makes unconditional)
0258
0259 0004 4022 GOGO        BR      GO            ;Branch to start of program
0260
0261           ************************************************************
0262           *   Interrupt vectors
0263           ************************************************************
0264 0010                 AORG    #0010
0265        A0             SBR     INT2_00       ;Timer Underflow, PCM=0, LPC=0
0266 0011   A0             SBR     INT2_00       ;Timer Underflow, PCM=0, LPC=0
0267 0012   A0             SBR     INT2_01       ;Timer Underflow, PCM=0, LPC=1
0268 0013   A0             SBR     INT2_01       ;Timer Underflow, PCM=0, LPC=1
0269 0014   A0             SBR     INT2_10       ;Timer Underflow, PCM=1, LPC=0
0270 0015   A0             SBR     INT2_10       ;Timer Underflow, PCM=1, LPC=0
0271 0016   A0             SBR     INT2_11       ;Timer Underflow, PCM=1, LPC=1
0272 0017   A0             SBR     INT2_11       ;Timer Underflow, PCM=1, LPC=1
0273 0018   A0             SBR     INT1_00       ;Pin (B1) goes low interrupt
0274 0019   A0             SBR     INT1_00       ;Pin (B1) goes low interrupt
0275 001A   A0             SBR     INT1_01       ;Clock interrupt, PCM=0, LPC=1
0276 001B   A0             SBR     INT1_01       ;Clock interrupt, PCM=0, LPC=1
0277 001C   A0             SBR     INT1_10       ;Clock interrupt, PCM=1, LPC=0
0278 001D   A0             SBR     INT1_10       ;Clock interrupt, PCM=1, LPC=0
0279 001E   A0             SBR     INT1_11       ;Clock interrupt, PCM=1, LPC=1
0280 001F   A0             SBR     INT1_11       ;Clock interrupt, PCM=1, LPC=1
0281
0282              INT1_10
0283              INT1_01
0284              INT2_10
0285              INT2_00
0286              INT2_01
0287              INT2_11
0288              INT1_00
0289 0020   2F INT1_11     CLA
0290 0021   3E             RETI
0291
0292
0293           ************************************************************
0294           *   Do program INITs
0295           ************************************************************
0296 0022 6900 GO          TMAD    0
0297
0298 0024   2F             CLA                   ;clear A to zero
0299 0025   20             CLX                   ;clear X to zero
0300 0026   22             DECXN                 ;point to location #FFF
0301 0027   16             TAM                   ;Initialize Mode Register 2
```

```
0302 0028   22             DECXN                ;point to location #FFE
0303 0029   16             TAM                  ;Initialize Mode Register 1
0304
0305 002A   20             CLX                  ;clear X to zero
0306 002B   13 RAM_LOOP    TAMIX                ;Initialize All RAM to zeros
0307 002C 61FF             XGEC    MAX_RAM      ;all RAM initialized?
0308 002E 4032             BR      RAM_EXIT     ;  yes, exit loop
0309 0030 402B             BR      RAM_LOOP     ;  no, continue looping
0310
0311 0032   16 RAM_EXIT    TAM                  ;initialize last RAM loaction (FF)
0312
0313 0033 6E01             TCA     1            ;Speak 2nd phrase on channel 2
0314 0035   1A             TAB
0315 0036 6E00             TCA     0            ;Speak 1st phrase on channel 1
0316 0038 003B             CALL    SPEAK
0317
0318 003A   3F             setoff               ;shut down
0319           ********************************************************
0320           *   Speak Utterance – Phrase numbers in A & B registers
0321           ********************************************************
0322 003B   3B SPEAK       INTGR
0323
0324 003C 6A65             TAMD    PHRASE1      ;Store channel 1 phrase number
0325 003E   12             XBA                  ;retrieve 2nd phrase number
0326 003F 6A66             TAMD    PHRASE2      ;Store channel 2 phrase number
0327
0328 0041   2F             CLA
0329 0042 6A62             TAMD    FLAGS        ;Init flags for speech
0330 0044 6A63             TAMD    FLAGS_1
0331 0046 6A64             TAMD    FLAGS_2
0332
0333           *-----Initialize address for channel 1-----*
0334
0335 0048 6965 LOOKUP1     TMAD    PHRASE1      ;Fetch stored phrase number
0336 004A   2E             SALA                 ;Double index to get offset
0337 004B 79E9             ACAAC   SENTENCE     ;Add base of table
0338
0339 004D   6D             LUAB                 ;get address MSB
0340 004E   3A             IAC                  ;point to address LSB
0341 004F   6B             LUAA                 ;get address LSB
0342 0050   12             XBA                  ;move LSB to B & MSB to A
0343 0051   1B             SALA4                ;combine MSB and LSB
0344 0052   1B             SALA4                ;   into complete address
0345 0053   2C             ABAAC
0346 0054 6A6E             TAMD    ADR_LSB1     ;save address LSB
0347 0056 6801             AXCA    1            ;isolate next address MSB (shift by 8)
0348 0058   15             SARA
0349 0059 6A6D             TAMD    ADR_MSB1     ;save address MSB
0350
0351           *-----Initialize address for channel 2
0352
0353           LOOKUP2
0354 005B 6966             TMAD    PHRASE2      ;Fetch stored phrase number
0355 005D   2E             SALA                 ;Double index to get offset
0356 005E 79E9             ACAAC   SENTENCE     ;Add base of table
0357
0358 0060   6D             LUAB                 ;get address MSB
0359 0061   3A             IAC                  ;point to address LSB
0360 0062   6B             LUAA                 ;get address LSB
0361 0063   12             XBA                  ;move LSB to B & MSB to A
0362 0064   1B             SALA4                ;combine MSB and LSB
0363 0065   1B             SALA4                ;  into complete address
```

```
0364 0066   2C             ABAAC
0365 0067 6A70             TAMD    ADR_LSB2     ;save address LSB
0366 0069 6801             AXCA    1            ;isolate next address MSB (shift by 8)
0367 006B   15             SARA
0368 006C 6A6F             TAMD    ADR_MSB2     ;save address MSB
0369
0370           *----Load next word address for channel 1
0371
0372 006E 6262 Nxt_Word1   TCX     FLAGS        ;point to general synthesis flags
0373 0070 6604             TSTCM   CH1END     ;has end of phrase on ch.1 been reached?
0374 0072 408F             BR      ChkCh2       ;  yes, test channel 2
0375
0376 0074 696E             TMAD    ADR_LSB1     ;fetch address from
0377 0076   1A             TAB                  ;  memory
0378 0077 696D             TMAD    ADR_MSB1
0379 0079   1B             SALA4                ;merge MSB and LSB into
0380 007A   1B             SALA4                ;  complete address
0381 007B   2C             ABAAC
0382 007C   1A             TAB                  ;save copy of address
0383 007D   3A             IAC                  ;generate next address
0384 007E 6A6E             TAMD    ADR_LSB1     ;save next address LSB
0385 0080 6801             AXCA    1            ;isolate next address MSB
0386 0082   15             SARA
0387 0083 6A6D             TAMD    ADR_MSB1     ;save next address MSB
0388 0085   12             XBA                  ;restore current address
0389
0390 0086   6B             LUAA                 ;get word number
0391 0087 60FF             ANEC    StopWord     ;is it a stop word?
0392 0089 4095             BR      SPEAK1a      ;  no, continue
0393
0394 008B 6262             TCX     FLAGS        ;stop word reached
0395 008D 6404             ORCM    CH1END       ;signal end of phrase reached on ch.1
0396 008F 6608 ChkCh2      TSTCM   CH2END       ;is channel 2 already done?
0397 0091 45E2             BR      STOP         ;  yes, return
0398 0093 6E00             TCA     SIL          ;  no, substitute silence
0399
0400           *****change code to switch channel 2 info to channel 1
0401
0402 0095   2E SPEAK1a     SALA                 ;Double index to get offset
0403 0096 7A1A             ACAAC   SPEECH       ;Add base of table
0404 0098   6D             LUAB                 ;get address MSB
0405 0099   3A             IAC
0406 009A   6B             LUAA                 ;Get address LSB
0407 009B 6A68             TAMD    ADR_LSB_1    ;Save speech address LSB
0408 009D   12             XBA
0409 009E 6A67             TAMD    ADR_MSB_1    ;Save speech address MSB
0410 00A0   2F             CLA
0411 00A1 6A69             TAMD    OFFSET_1     ;Initialize bit offset
0412
0413 00A3 6262             TCX     FLAGS
0414 00A5 6620             TSTCM   Ch1Inh_2     ;is second inhibit bit set?
0415 00A7 448D             BR      ZeroKs_1b    ;   yes, return to ZeroKs
0416
0417           *----Load next word address for channel 2
0418
0419 00A9 6262 Nxt_Word2   TCX     FLAGS        ;point to general synthesis flags
0420 00AB 6608             TSTCM   CH2END      ;has end of phrase on ch.2 been reached?
0421 00AD 40CA             BR      ChkCh1       ;  yes, test channel 1
0422
0423 00AF 6970             TMAD    ADR_LSB2     ;fetch address from
0424 00B1   1A             TAB                  ;  memory
0425 00B2 696F             TMAD    ADR_MSB2
```

```
0426 00B4   1B            SALA4               ;merge MSB and LSB into
0427 00B5   1B            SALA4               ;  complete address
0428 00B6   2C            ABAAC
0429 00B7   1A            tab                 ;save copy of address
0430 00B8   3A            IAC                 ;generate next address
0431 00B9 6A70            TAMD    ADR_LSB2    ;save next address LSB
0432 00BB 6801            AXCA    1           ;isolate next address MSB
0433 00BD   15            SARA
0434 00BE 6A6F            TAMD    ADR_MSB2    ;save next address MSB
0435 00C0   12            XBA                 ;restore current address
0436
0437 00C1   6B            LUAA                ;get word number
0438 00C2 60FF            ANEC    StopWord    ;is it a stop word?
0439 00C4 40D0            BR      SPEAK2a     ;  no, continue
0440
0441 00C6 6262            TCX     FLAGS       ;stop word reached
0442 00C8 6408            ORCM    CH2END      ;signal end of phrase reached on ch.2
0443 00CA 6604 ChkCh1     TSTCM   CH1END      ;is channel 1 already done?
0444 00CC 45E2            BR      STOP        ;  yes, return from routine
0445 00CE 6E00            TCA     SIL         ;  no, substitute silence
0446
0447           ***** change code to turn channel 2 off if channel 1 is still on
0448
0449 00D0   2E SPEAK2a    SALA                ;Double index to get offset
0450 00D1 7A1A            ACAAC   SPEECH      ;Add base of table
0451 00D3   6D            LUAB                ;Get address MSB
0452 00D4   3A            IAC
0453 00D5   6B            LUAA                ;Get address LSB
0454 00D6 6A6B            TAMD    ADR_LSB_2   ;Save speech address LSB
0455 00D8   12            XBA
0456 00D9 6A6A            TAMD    ADR_MSB_2   ;Save speech address MSB
0457 00DB   2F            CLA
0458 00DC 6A6C            TAMD    OFFSET_2    ;Initialize bit offset
0459
0460 00DE 6262            TCX     FLAGS
0461 00E0 6680            TSTCM   Ch2Inh_2    ;is second inhibit bit set?
0462 00E2 45C0            BR      ZeroKs_2b   ;   yes, return to ZeroKs
0463
0464           *-----Initialize memory mapped synthesis registers
0465 00E4   2F            CLA                 ;Kill K11 and K12 parameters
0466 00E5 6A03            TAMD    K11_A
0467 00E7 6A13            TAMD    K11_B
0468 00E9 6A02            TAMD    K12_A
0469 00EB 6A12            TAMD    K12_B
0470
0471 00ED 7FF0            ACAAC   #FF0        ;load address to X register
0472 00EF   18            TAX
0473
0474 00F0   2F            CLA
0475 00F1   13            TAMIX               ;initialize Y1S (X=FF0)
0476 00F2   13            TAMIX               ;initialize PITCH2 (X=FF1)
0477 00F3   13            TAMIX               ;initialize Y1 (X=FF2)
0478
0479 00F4 7162            ACAAC   #162
0480 00F6   13            TAMIX               ;initialize PITCH (X=FF3)
0481 00F7   2F            CLA
0482 00F8   13            TAMIX               ;initialize TEMP (X=FF4)
0483
0484 00F9 6A63            TAMD    FLAGS_1     ;Init flags for speech
0485 00FB 6A64            TAMD    FLAGS_2
0486 00FD 6262            TCX     FLAGS
0487 00FF 65FC            ANDCM   ~(STOP1+STOP2)
```

```
0488
0489 0101   2F             CLA                 ;Load C2 parameter
0490 0102 7B67             ACAAC   C2_Value    ;(a device constant)
0491 0104 6A0F             TAMD    C2
0492
0493 0106   2F             CLA                 ;Load C1 parameter
0494 0107 7F61             ACAAC   C1_Value    ;(a device constant)
0495 0109 6A0E             TAMD    C1
0496
0497 010B   2F             CLA                 ;Load C3 parameter
0498 010C 73FF             ACAAC   C3_Value    ;(channel scaling)
0499 010E 6A1E             TAMD    C3
0500            *
0501            * Now we give an initial value to the Pitch in case the
0502            * utterance starts with a silent frame.
0503            *
0504 0110   2F             CLA
0505 0111 700C             ACAAC   #0C
0506 0113 6A24             TAMD    PH_Old_A    ;save as old pitch, channel 1
0507 0115 6A22             TAMD    PH_New_A    ;save as new pitch, channel 1
0508 0117 6A44             TAMD    PH_Old_B    ;save as old pitch, channel 2
0509 0119 6A42             TAMD    PH_New_B    ;save as new pitch, channel 2
0510            *
0511            * Now we initialize the prescale and timer so that they
0512            * will count during the coming calls to UPDATE
0513            *
0514 011B 6E5D             TCA     PSVALUE     ;Initialize prescale
0515 011D   19             TAPSC
0516 011E   1E             TATM
0517            *
0518            * Now we preload the first two frames.
0519            *
0520
0521 011F 0370             CALL    UPDATE      ;Load first frame
0522 0121 0370             CALL    UPDATE      ;Load 2nd frame
0523            *
0524            * Now we give a full scale value to the Timer so that the
0525            * interpolation in this first call to INTP will be correctly
0526            * scaled.  Then we do the first call to INTP to preload the
0527            * first valid interpolation.
0528            *
0529 0123 6E7F             TCA     #7F         ;Pretend there was a previous
0530 0125 6A60             TAMD    TIMER       ;    update
0531 0127   1E             TATM                ;Set timer to max value to
0532                                           ;    disable interpolation
0533
0534 0128 014C             CALL    INTP1       ;Do first interpolation for ch.1
0535
0536 012A 6E7F             TCA     #7F         ;Pretend there was a previoua
0537 012C 6A60             TAMD    TIMER       ;    update
0538 012E   1E             TATM                ;Set timer to max value to
0539                                           ;    disable interpolation
0540 012F 025F             CALL    INTP2       ;Do first interpolation for ch.2
0541
0542            *
0543            * Now we enable the synthesizer for speech
0544            *
0545 0131   20             CLX
0546 0132   22             DECXN
0547 0133   22             DECXN               ;point to mode register 1 (X=FFE)
0548 0134 6402             ORCM    LPC         ;turn on LPC for channel 1
0549 0136   21             IXC                 ;point to mode register 2 (X=FFF)
```

```
0550 0137 6402            ORCM    LPC          ;turn on LPC for channel 2
0551
0552           *
0553           * Now we loop until the utterance is complete.  When the
0554           * utterance is finished, the routine UPDATE will execute a
0555           * RETN instruction which will exit this routine.  In the
0556           * mean time, this loop will poll the Timer register and
0557           * update the frame whenever it underflows and perform
0558           * interpolations when approprite.
0559           *
0560           * * * * *
0561           SPEAK_LP
0562 0139   17            TTMA                 ;Fetch timer value
0563 013A 6380            AGEC    #80          ;Has it underflowed?
0564 013C 4370            BR      UPDATE       ;    yes, do an update
0565
0566 013E   20            CLX
0567 013F   22            DECXN
0568 0140 6601            TSTCM   PPC1       ;has PPC for ch.1 decremented below 200h?
0569 0142 014C            CALL    INTP1        ;  yes, interpolate
0570
0571 0144   20            CLX
0572 0145   22            DECXN
0573 0146 6608            TSTCM   PPC2       ;has PPC for ch.2 decremented below 200h?
0574 0148 025F            CALL    INTP2        ;   yes, interpolate
0575 014A 4139            BR      SPEAK_LP     ;   no, wait some more
0576
0577
0578           * ------------------------------------------------------*
0579           * INTP1                                                 *
0580           * This routine performs the interpolation function for    *
0581           * the channel 1 synthesizer.                            *
0582           * ------------------------------------------------------*
0583 014C   20 INTP1       CLX
0584 014D   22            DECXN                ;point to Mode Register 2
0585 014E 65EE            ANDCM   ~(PPC1+CHANNEL)  ;reset PPC1 bit & select ch.1
0586
0587           * * * * *
0588           * First we must check to see if we are ready for
0589           * interpolation.
0590           * * * * *
0591 0150 6262            TCX     FLAGS
0592 0152 6620            TSTCM   Ch1Inh_2     ;is second inhibit bit set?
0593 0154 4158            BR      INTP1a     ;   yes, not ready for interpolation yet?
0594 0156 415C            BR      INTP1b       ;   no, continue with interpolation
0595
0596           INTP1a
0597 0158   2F            CLA                  ;clear energy while waiting
0598 0159 6A01            TAMD    EN_A
0599 015B   3D            RETN                 ;return from call
0600
0601 015C   17 INTP1b      TTMA                 ;Get timer register contents
0602 015D 6380            AGEC    #80          ;Has timer underflowed?
0603 015F 4163            BR      Adjust1      ;   yes, Trick Scale to avoid prob
0604 0161 4164            BR      DoScale1     ;   no, Use Timer as is
0605
0606 0163   2F Adjust1     CLA                  ;Pretend no underflow
0607 0164 6A61 DoScale1    TAMD    SCALE        ;Load timer val into scale
0608
0609           * * * * *
0610           * See if this routine is enabled.  If it is not, exit
0611           * the routine.
```

```
0612           * * * * *
0613 0166 6263             TCX     FLAGS_1      ;Point to channel 1 flags
0614 0168 6601             TSTCM   Int_Off      ;If routine disabled...
0615 016A 425E             BR      IRETN1       ; ...branch to exit point
0616           * * * * *
0617           * Next we need to see if the frame type has changed between
0618           * voiced and unvoiced frames.  If it has, we do not want to
0619           * interpolate between them; we just want to use the current
0620           * frame values until we have two frames of the same type to
0621           * interpolate between.
0622           * * * * *
0623 016C 6263 TINTP1        TCX     FLAGS_1      ;Point to channel 1 status flags
0624 016E 6620             TSTCM   Int_Inh      ;Is interpolation inhibited?
0625 0170 4174             BR      NOINT1       ;   yes, use inhibit code
0626 0172 4191             BR      INTPCH1      ;   no, use interpolation code
0627           * * * * *
0628           * The following code is reached if interpolation is
0629           * inhibited. It sets the stored timer value to #7F which
0630           * effectively forces the interpolation to yield the old
0631           * values for the working values, thus effectively disabling
0632           * interpolation.
0633           * * * * *
0634 0174 6E7F NOINT1        TCA     #7F          ;Set Scale factor to
0635 0176 6A61             TAMD    SCALE        ;   highest value
0636           *
0637           * If the new frame has a voicing different from the last
0638           * frame, we want to zero the energy until the Unvoiced bit
0639           * in the mode register is changed and the K parameters are
0640           * all to the correct values.  We therefore check in this
0641           * section of code to see if the frame voicing is different
0642           * from the setting in the Mode Register.  If it is, we zero
0643           * the energy until after the Mode Register is modified.
0644           *
0645 0178 6263             TCX     FLAGS_1
0646 017A 6680             TSTCM   Unv_Flg_Old  ;Is new frame unvoiced?
0647 017C 4187             BR      Uv1          ;    yes, go to unvoiced branch
0648
0649 017E   20             CLX
0650 017F   22             DECXN                ;point to Mode Register 1
0651 0180   22             DECXN                ;New frame is voiced
0652 0181 6680             TSTCM   UNV          ;Has mode changed to unvoiced
0653 0183 418E             BR      ClrEN1       ;     yes, clear the energy
0654 0185 4191             BR      INTPCH1      ;     no, no action required
0655
0656 0187   20 Uv1          CLX                  ;New frame is unvoiced
0657 0188   22             DECXN
0658 0189   22             DECXN
0659 018A 6680             TSTCM   UNV          ;Has voicing mode changed?
0660 018C 4191             BR      INTPCH1      ;     no, no action required
0661
0662 018E   2F ClrEN1        CLA                  ;Zero Energy during update
0663 018F 6A01             TAMD    EN_A
0664
0665           *
0666           * Interpolate Pitch and write the result to the pitch
0667           * register
0668           *
0669 0191 6222 INTPCH1      TCX     PH_New_A     ;Combine new pitch and fractional
0670 0193   14             TMAIX                ;     pitch and leave in
0671 0194   1B             SALA4                ;     the B register
0672 0195   28             AMAAC
0673 0196   21             IXC
```

```
0674 0197   1A            TAB
0675 0198   14            TMAIX               ;Combine current pitch and
0676 0199   1B            SALA4               ;    current fractional pitch
0677 019A   28            AMAAC               ;    and leave in A register
0678
0679 019B   2D            SBAAN               ;(Pcurrent – Pnew)
0680 019C 6261            TCX     SCALE
0681 019E   39            AXMA                ;(Pcurrent – Pnew) * Timer
0682 019F   2C            ABAAC               ;Pnew + (Pcurrent – Pnew)* Timer
0683 01A0   2E            SALA                ;Double value to index excitation
0684
0685          **          TASYN               ;Write to pitch register
0686
0687 01A1   1A            tab                 ;Write to pitch register (#FF3)
0688 01A2   2F            cla                 ;  NOTE: TASYN instruction may cause
0689 01A3 7FF3            acaac   #ff3        ;        problems, so we must write
0690 01A5   18            tax                 ;        directly to the pitch reg.
0691 01A6   12            xba
0692 01A7   16            tam
0693
0694          *
0695          * Interpolate Energy and store the result in the working
0696          * register
0697          *
0698 01A8 6220            TCX     EN_New_A    ;Put New Energy (adjusted to
0699 01AA   14            TMAIX               ;   12 bits) in B register
0700 01AB   1B            SALA4
0701 01AC   1A            TAB
0702 01AD   14            TMAIX
0703 01AE   1B            SALA4               ;Put Current Energy (adjusted to
0704                                          ;    12 bits) in A register
0705 01AF   2D            SBAAN
0706 01B0 6261            TCX     SCALE
0707 01B2   39            AXMA                ;(Ecurrent – Enew) * Timer
0708 01B3   2C            ABAAC               ;Enew + (Ecurrent – Enew)*Timer
0709 01B4   15            SARA
0710 01B5 6A01            TAMD    EN_A        ;Store Energy till mode is switched
0711
0712 01B7   3C            EXTSG               ;Allow negative K parameters
0713          *
0714          * Interpolate K1 and store the result in the working K1
0715          * register
0716          *
0717 01B8 6226            TCX     K1_New_A    ;Combine New K1 and New
0718 01BA   14            TMAIX               ;   fractional K1 and
0719 01BB   1B            SALA4               ;   leave in the B register
0720 01BC   28            AMAAC
0721 01BD   21            IXC
0722 01BE   1A            TAB
0723
0724 01BF   14            TMAIX               ;Combine current K1 and
0725 01C0   1B            SALA4               ;   current fractional K1 and
0726 01C1   28            AMAAC               ;   leave in the A register
0727
0728 01C2   2D            SBAAN               ;(K1current – K1new)
0729 01C3 6261            TCX     SCALE
0730 01C5   39            AXMA                ;(K1current – K1new) * Timer
0731 01C6   2C            ABAAC               ;K1new+(K1current–K1new)* Timer
0732 01C7 6A0D            TAMD    K1_A        ;Load interpolated K1 value
0733          *
0734          * Interpolate K2 and store the result in the
0735          * working K2 register
```

```
0736           *
0737 01C9 622A             TCX     K2_New_A     ;Put New K2 (adjusted to
0738 01CB   14             TMAIX                ;   12 bits) in B register
0739 01CC   1B             SALA4
0740 01CD   28             AMAAC
0741 01CE   21             IXC
0742 01CF   1A             TAB
0743
0744 01D0   14             TMAIX                ;Put Current K2 (adjusted to
0745 01D1   1B             SALA4                ;   12 bits) in A register
0746 01D2   28             AMAAC
0747
0748 01D3   2D             SBAAN                ;(K2current – K2new)
0749 01D4 6261             TCX     SCALE
0750 01D6   39             AXMA                 ;(K2current – K2new) * Timer
0751 01D7   2C             ABAAC                ;K2new+(K2current–K2new)* Timer
0752 01D8 6A0C             TAMD    K2_A         ;Load interpolated K2 value
0753           *
0754           * Interpolate K3 and store the result in the working K3
0755           * register
0756           *
0757 01DA 622E             TCX     K3_New_A     ;Put New K3 (adjusted to
0758 01DC   14             TMAIX                ;   12 bits) in B register
0759 01DD   1B             SALA4
0760 01DE   1A             TAB
0761
0762 01DF   14             TMAIX                ;Put Current K3 (adjusted to
0763 01E0   1B             SALA4                ;   12 bits) in A register
0764
0765 01E1   2D             SBAAN                ;(K3current – K3new)
0766 01E2 6261             TCX     SCALE
0767 01E4   39             AXMA                 ;(K3current – K3new) * Timer
0768 01E5   2C             ABAAC                ;K3new+(K3current–K3new)* Timer
0769 01E6 6A0B             TAMD    K3_A         ;Load interpolated K3 value
0770           *
0771           * Interpolate K4 and store the result in the working K4
0772           * register
0773           *
0774 01E8 6230             TCX     K4_New_A     ;Put New K4 (adjusted to
0775 01EA   14             TMAIX                ;   12 bits) in B register
0776 01EB   1B             SALA4
0777 01EC   1A             TAB
0778
0779 01ED   14             TMAIX                ;Put Current K4 (adjusted to
0780 01EE   1B             SALA4                ;   12 bits) in A register
0781
0782 01EF   2D             SBAAN                ;(K4current – K4new)
0783 01F0 6261             TCX     SCALE
0784 01F2   39             AXMA                 ;(K4current – K4new) * Timer
0785 01F3   2C             ABAAC                ;K4new+(K4current–K4new)* Timer
0786 01F4 6A0A             TAMD    K4_A         ;Load interpolated K4 value
0787           *
0788           * Interpolate K5 and store the result in the working K5
0789           * register
0790           *
0791 01F6 6232             TCX     K5_New_A     ;Put New K5 (adjusted to
0792 01F8   14             TMAIX                ;   12 bits) in B register
0793 01F9   1B             SALA4
0794 01FA   1A             TAB
0795 01FB   14             TMAIX                ;Put Current K5 (adjusted to
0796 01FC   1B             SALA4                ;   12 bits) in A register
0797
```

```
0798 01FD   2D            SBAAN               ;(K5current – K5new)
0799 01FE 6261            TCX     SCALE
0800 0200   39            AXMA                ;(K5current – K5new) * Timer
0801 0201   2C            ABAAC               ;K5new+(K5current–K5new)* Timer
0802 0202 6A09            TAMD    K5_A        ;Load interpolated K5 value
0803                *
0804                * Interpolate K6 and store the result in the working K6
0805                * register
0806                *
0807 0204 6234            TCX     K6_New_A    ;Put New K6 (adjusted to
0808 0206   14            TMAIX               ;   12 bits) in B register
0809 0207   1B            SALA4
0810 0208   1A            TAB
0811 0209   14            TMAIX               ;Put Current K6 (adjusted to
0812 020A   1B            SALA4               ;   12 bits) in A register
0813
0814 020B   2D            SBAAN               ;(K6current – K6new)
0815 020C 6261            TCX     SCALE
0816 020E   39            AXMA                ;(K6current – K6new) * Timer
0817 020F   2C            ABAAC               ;K6new+(K6current–K6new)* Timer
0818 0210 6A08            TAMD    K6_A        ;Load interpolated K6 value
0819                *
0820                * Interpolate K7 and store the result in the working K7
0821                * register
0822                *
0823 0212 6236            TCX     K7_New_A    ;Put New K7 (adjusted to
0824 0214   14            TMAIX               ;   12 bits) in B register
0825 0215   1B            SALA4
0826 0216   1A            TAB
0827 0217   14            TMAIX               ;Put Current K7 (adjusted to
0828 0218   1B            SALA4               ;   12 bits) in A register
0829
0830 0219   2D            SBAAN               ;(K7current – K7new)
0831 021A 6261            TCX     SCALE
0832 021C   39            AXMA                ;(K7current – K7new) * Timer
0833 021D   2C            ABAAC               ;K7new+(K7current–K7new)* Timer
0834 021E 6A07            TAMD    K7_A        ;Load interpolated K7 value
0835                *
0836                * Interpolate K8 and store the result in the working K8
0837                * register
0838                *
0839 0220 6238            TCX     K8_New_A    ;Put New K8 (adjusted to
0840 0222   14            TMAIX               ;   12 bits) in B register
0841 0223   1B            SALA4
0842 0224   1A            TAB
0843
0844 0225   14            TMAIX               ;Put Current K8 (adjusted to
0845 0226   1B            SALA4               ;   12 bits) in A register
0846
0847 0227   2D            SBAAN               ;(K8current – K8new)
0848 0228 6261            TCX     SCALE
0849 022A   39            AXMA                ;(K8current – K8new) * Timer
0850 022B   2C            ABAAC               ;K8new+(K8current–K8new)* Timer
0851 022C 6A06            TAMD    K8_A        ;Load interpolated K8 value
0852                *
0853                * Interpolate K9 and store the result in the working K9
0854                * register
0855                *
0856 022E 623A            TCX     K9_New_A    ;Put New K9 (adjusted to
0857 0230   14            TMAIX               ;   12 bits) in B register
0858 0231   1B            SALA4
0859 0232   1A            TAB
```

```
0860
0861 0233   14            TMAIX               ;Put Current K9 (adjusted to
0862 0234   1B            SALA4               ;   12 bits) in A register
0863
0864 0235   2D            SBAAN               ;(K9current – K9new)
0865 0236 6261            TCX     SCALE
0866 0238   39            AXMA                ;(K9current – K9new) * Timer
0867 0239   2C            ABAAC               ;K9new+(K9current–K9new)* Timer
0868 023A 6A05            TAMD    K9_A        ;Load interpolated K9 value
0869           *
0870           * Interpolate K10 and store the result in the working K10
0871           * register
0872           *
0873 023C 623C            TCX     K10_New_A   ;Put New K10 (adjusted to
0874 023E   14            TMAIX               ;   12 bits) in B register
0875 023F   1B            SALA4
0876 0240   1A            TAB
0877
0878 0241   14            TMAIX               ;Put Current K10 (adjusted to
0879 0242   1B            SALA4               ;   12 bits) in A register
0880
0881 0243   2D            SBAAN               ;(K10current – K10new)
0882 0244 6261            TCX     SCALE
0883 0246   39            AXMA                ;(K10current – K10new) * Timer
0884 0247   2C            ABAAC               ;K10new+(K10current–K10new)* Timer
0885 0248 6A04            TAMD    K10_A       ;Load interpolated K10 value
0886           *
0887           * K11 and K12 are not needed for LPC 10, so they have
0888           * been commented out.
0889           *
0890           * Interpolate K11 and store the result in the working K11
0891           * register
0892           *
0893           *       TCX     K11_New_A       ;Put New K11 (adjusted to
0894           *       TMAIX                   ;  12 bits) in B register
0895           *       SALA4
0896           *       TAB
0897           *
0898           *       TMAIX                   ;Put Current K11 (adjusted to
0899           *       SALA4                   ;  12 bits) in A register
0900           *
0901           *       SBAAN                   ;(K11current – K11new)
0902           *       TCX     SCALE
0903           *       AXMA                    ;(K11current – K11new) * Timer
0904           *       ABAAC                   ;K11new+(K11current–K11new)* Timer
0905           *       TAMD    K11_A           ;Load interpolated K11 value
0906           *
0907           * Interpolate K12 and store the result in the working
0908           * K12 register
0909           *
0910           *       TCX     K12_New_A       ;Put New K12 (adjusted to
0911           *       TMAIX                   ;  12 bits) in B register
0912           *       SALA4
0913           *       TAB
0914           *
0915           *       TMAIX                   ;Put Current K12 (adjusted to
0916           *       SALA4                   ;  12 bits) in A register
0917           *
0918           *       SBAAN                   ;(K12current – K12new)
0919           *       TCX     SCALE
0920           *       AXMA                    ;(K12current – K12new) * Timer
0921           *       ABAAC                   ;K12new+(K12current–K12new)* Timer
```

```
0922           *       TAMD    K12_A           ;Load interpolated K12 value
0923
0924           *
0925           * Set voiced/unvoiced mode according to current frame type.
0926           * This is done by loading Mode Register 1 address into the
0927           * X Register and adjusting the bit with an AND or OR operation.
0928           *
0929 024A   3B STMODE1     INTGR               ;Back to integer mode
0930 024B 6263             TCX     FLAGS_1
0931 024D 65FB             ANDCM   ~Update_Flg ;Signal that interp. done
0932 024F 6680             TSTCM   Unv_Flg_Old ;Is current frame unvoiced?
0933 0251 4259             BR      SETUV1      ;   yes, set mode to unvoiced
0934 0253   20             CLX                 ;   no, ...
0935 0254   22             DECXN               ;point to mode register 1
0936 0255   22             DECXN
0937 0256 657F             ANDCM   ~UNV        ;...set mode to voiced
0938
0939 0258   3D             RETN                ;Return from first call
0940
0941 0259   20 SETUV1      CLX
0942 025A   22             DECXN               ;point to mode register 1
0943 025B   22             DECXN               ;set mode to unvoiced and reset
0944 025C 6482             ORCM    UNV+LPC     ;   LPC bit to avoid glitch
0945
0946           IRETN1
0947 025E   3D             RETN                ;Return from first call
0948
0949           * ----------------------------------------------------------*
0950           * INTP2                                                     *
0951           * This routine performs the interpolation function for      *
0952           * the channel 2 synthesizer.                                *
0953           * ----------------------------------------------------------*
0954 025F   20 INTP2       CLX
0955 0260   22             DECXN               ;point to Mode Register 2
0956 0261 65F7             ANDCM   ~PPC2       ;reset PPC2 bit
0957 0263 6410             ORCM    CHANNEL     ;select channel 2
0958
0959           * * * * *
0960           * First we must check to see if we are ready for
0961           * interpolation.
0962           * * * * *
0963 0265 6262             TCX     FLAGS
0964 0267 6680             TSTCM   Ch2Inh_2    ;is second inhibit bit set?
0965 0269 426D             BR      INTP2a      ;   yes, not ready for interolation
0966 026B 4271             BR      INTP2b      ;   no, continue with interpolation
0967
0968           INTP2a
0969 026D   2F             CLA                 ;clear energy while waiting
0970 026E 6A11             TAMD    EN_B
0971 0270   3D             RETN                ;return from call
0972
0973 0271   17 INTP2b      TTMA                ;Get timer register contents
0974 0272 6380             AGEC    #80         ;Has timer underflowed?
0975 0274 4278             BR      Adjust2     ;   yes, Trick Scale to avoid prob
0976 0276 4279             BR      DoScale2    ;   no, Use Timer as is
0977
0978 0278   2F Adjust2     CLA                 ;Pretend no underflow
0979 0279 6A61 DoScale2    TAMD    SCALE       ;Load timer val into scale
0980
0981           * * * * *
0982           * See if this routine is enabled.  If it is not, exit
0983           * the routine.
```

```
0984           * * * * *
0985 027B 6264             TCX     FLAGS_2      ;Point to channel 2 flags
0986 027D 6601             TSTCM   Int_Off      ;If routine disabled...
0987 027F 436F             BR      IRETN2       ; ...branch to exit point
0988           * * * * *
0989           * Next we need to see if the frame type has changed between
0990           * voiced and unvoiced frames.  If it has, we do not want to
0991           * interpolate between them; we just want to use the current
0992           * frame values until we have two frames of the same type to
0993           * interpolate between.
0994           * * * * *
0995 0281 6264 TINTP2       TCX     FLAGS_2      ;Point to status flags
0996 0283 6620             TSTCM   Int_Inh      ;Is interpolation inhibited?
0997 0285 4289             BR      NOINT2       ;   yes, use inhibit code
0998 0287 42A4             BR      INTPCH2      ;   no, use interpolation code
0999           * * * * *
1000           * The following code is reached if interpolation is
1001           * inhibited. It sets the stored timer value to #7F which
1002           * effectively forces the interpolation to yield the old
1003           * values for the working values, thus effectively disabling
1004           * interpolation.
1005           * * * * *
1006 0289 6E7F NOINT2       TCA     #7F          ;Set Scale factor to
1007 028B 6A61             TAMD    SCALE        ;   highest value
1008           *
1009           * If the new frame has a voicing different from the last
1010           * frame, we want to zero the energy until the Unvoiced bit
1011           * in the mode register is changed and the K parameters are
1012           * all to the correct values.  We therefore check in this
1013           * section of code to see if the frame voicing is different
1014           * from the setting in the Mode Register.  If it is, we zero
1015           * the energy until after the Mode Register is modified.
1016           *
1017 028D 6264             TCX     FLAGS_2
1018 028F 6680             TSTCM   Unv_Flg_Old  ;Is new frame unvoiced?
1019 0291 429B             BR      Uv2          ;   yes, go to unvoiced branch
1020
1021 0293   20             CLX
1022 0294   22             DECXN                ;point to mode register 2
1023 0295 6680             TSTCM   UNV          ;Has mode changed to unvoiced?
1024 0297 42A1             BR      ClrEN2       ;    yes, clear the energy
1025 0299 42A4             BR      INTPCH2      ;    no, no action required
1026
1027 029B   20 Uv2         CLX                  ;New frame is unvoiced
1028 029C   22             DECXN                ;point to mode register 2
1029 029D 6680             TSTCM   UNV          ;Has voicing mode changed?
1030 029F 42A4             BR      INTPCH2      ;    no, no action required
1031
1032 02A1   2F ClrEN2      CLA                  ;Zero Energy during update
1033 02A2 6A11             TAMD    EN_B
1034
1035           *
1036           * Interpolate Pitch and write the result to the pitch
1037           * register
1038           *
1039 02A4 6242 INTPCH2      TCX     PH_New_B     ;Combine new pitch and fractional
1040 02A6   14             TMAIX                ;    pitch and leave in
1041 02A7   1B             SALA4                ;    the B register
1042 02A8   28             AMAAC
1043 02A9   21             IXC
1044 02AA   1A             TAB
1045 02AB   14             TMAIX                ;Combine current pitch and
```

```
1046 02AC   1B            SALA4               ;    current fractional pitch
1047 02AD   28            AMAAC               ;    and leave in A register
1048
1049 02AE   2D            SBAAN               ;(Pcurrent – Pnew)
1050 02AF 6261            TCX     SCALE
1051 02B1   39            AXMA                ;(Pcurrent – Pnew) * Timer
1052 02B2   2C            ABAAC               ;Pnew + (Pcurrent – Pnew)* Tmer
1053 02B3   2E            SALA                ;Double value to index excitation
1054
1055          **          TASYN               ;Write to pitch register
1056
1057 02B4   1A            tab                 ;Write to pitch register
1058 02B5   2F            cla                 ; !NOTE: TASYN instruction may cause
1059 02B6 7FF1            acaac   #ff1        ;        problems, so we must write
1060 02B8   18            tax                 ;        directly to the pitch reg.
1061 02B9   12            xba
1062 02BA   16            tam
1063
1064          *
1065          * Interpolate Energy and store the result in the working
1066          * register
1067          *
1068 02BB 6240            TCX     EN_New_B    ;Put New Energy (adjusted to
1069 02BD   14            TMAIX               ;   12 bits) in B register
1070 02BE   1B            SALA4
1071 02BF   1A            TAB
1072 02C0   14            TMAIX
1073 02C1   1B            SALA4               ;Put Current Energy (adjusted to
1074                                          ;   12 bits) in A register
1075 02C2   2D            SBAAN
1076 02C3 6261            TCX     SCALE
1077 02C5   39            AXMA                ;(Ecurrent – Enew) * Timer
1078 02C6   2C            ABAAC               ;Enew + (Ecurrent – Enew)*Timer
1079 02C7   15            SARA
1080 02C8 6A11            TAMD    EN_B        ;Store Energy till mode is switched
1081
1082 02CA   3C            EXTSG               ;Allow negative K parameters
1083          *
1084          * Interpolate K1 and store the result in the working K1
1085          * register
1086          *
1087 02CB 6246            TCX     K1_New_B    ;Combine New K1 and New
1088 02CD   14            TMAIX               ;   fractional K1 and
1089 02CE   1B            SALA4               ;   leave in the B register
1090 02CF   28            AMAAC
1091 02D0   21            IXC
1092 02D1   1A            TAB
1093
1094 02D2   14            TMAIX               ;Combine current K1 and
1095 02D3   1B            SALA4               ;   current fractional K1 and
1096 02D4   28            AMAAC               ;   leave in the A register
1097
1098 02D5   2D            SBAAN               ;(K1current – K1new)
1099 02D6 6261            TCX     SCALE
1100 02D8   39            AXMA                ;(K1current – K1new) * Timer
1101 02D9   2C            ABAAC               ;K1new+(K1current–K1new)* Timer
1102 02DA 6A1D            TAMD    K1_B        ;Load interpolated K1 value
1103          *
1104          * Interpolate K2 and store the result in the
1105          * working K2 register
1106          *
1107 02DC 624A            TCX     K2_New_B    ;Put New K2 (adjusted to
```

```
1108 02DE   14            TMAIX               ;   12 bits) in B register
1109 02DF   1B            SALA4
1110 02E0   28            AMAAC
1111 02E1   21            IXC
1112 02E2   1A            TAB
1113
1114 02E3   14            TMAIX               ;Put Current K2 (adjusted to
1115 02E4   1B            SALA4               ;   12 bits) in A register
1116 02E5   28            AMAAC
1117
1118 02E6   2D            SBAAN               ;(K2current – K2new)
1119 02E7 6261            TCX     SCALE
1120 02E9   39            AXMA                ;(K2current – K2new) * Timer
1121 02EA   2C            ABAAC               ;K2new+(K2current–K2new)* Timer
1122 02EB 6A1C            TAMD    K2_B        ;Load interpolated K2 value
1123           *
1124           * Interpolate K3 and store the result in the working K3
1125           * register
1126           *
1127 02ED 624E            TCX     K3_New_B    ;Put New K3 (adjusted to
1128 02EF   14            TMAIX               ;   12 bits) in B register
1129 02F0   1B            SALA4
1130 02F1   1A            TAB
1131
1132 02F2   14            TMAIX               ;Put Current K3 (adjusted to
1133 02F3   1B            SALA4               ;   12 bits) in A register
1134
1135 02F4   2D            SBAAN               ;(K3current – K3new)
1136 02F5 6261            TCX     SCALE
1137 02F7   39            AXMA                ;(K3current – K3new) * Timer
1138 02F8   2C            ABAAC               ;K3new+(K3current–K3new)* Timer
1139 02F9 6A1B            TAMD    K3_B        ;Load interpolated K3 value
1140           *
1141           * Interpolate K4 and store the result in the working K4
1142           * register
1143           *
1144 02FB 6250            TCX     K4_New_B    ;Put New K4 (adjusted to
1145 02FD   14            TMAIX               ;   12 bits) in B register
1146 02FE   1B            SALA4
1147 02FF   1A            TAB
1148
1149 0300   14            TMAIX               ;Put Current K4 (adjusted to
1150 0301   1B            SALA4               ;   12 bits) in A register
1151
1152 0302   2D            SBAAN               ;(K4current – K4new)
1153 0303 6261            TCX     SCALE
1154 0305   39            AXMA                ;(K4current – K4new) * Timer
1155 0306   2C            ABAAC               ;K4new+(K4current–K4new)* Timer
1156 0307 6A1A            TAMD    K4_B        ;Load interpolated K4 value
1157           *
1158           * Interpolate K5 and store the result in the working K5
1159           * register
1160           *
1161 0309 6252            TCX     K5_New_B    ;Put New K5 (adjusted to
1162 030B   14            TMAIX               ;   12 bits) in B register
1163 030C   1B            SALA4
1164 030D   1A            TAB
1165 030E   14            TMAIX               ;Put Current K5 (adjusted to
1166 030F   1B            SALA4               ;   12 bits) in A register
1167
1168 0310   2D            SBAAN               ;(K5current – K5new)
1169 0311 6261            TCX     SCALE
```

```
1170 0313   39            AXMA                ;(K5current – K5new) * Timer
1171 0314   2C            ABAAC               ;K5new+(K5current–K5new)* Timer
1172 0315 6A19            TAMD    K5_B        ;Load interpolated K5 value
1173           *
1174           * Interpolate K6 and store the result in the working K6
1175           * register
1176           *
1177 0317 6254            TCX     K6_New_B    ;Put New K6 (adjusted to
1178 0319   14            TMAIX               ;   12 bits) in B register
1179 031A   1B            SALA4
1180 031B   1A            TAB
1181 031C   14            TMAIX               ;Put Current K6 (adjusted to
1182 031D   1B            SALA4               ;   12 bits) in A register
1183
1184 031E   2D            SBAAN               ;(K6current – K6new)
1185 031F 6261            TCX     SCALE
1186 0321   39            AXMA                ;(K6current – K6new) * Timer
1187 0322   2C            ABAAC               ;K6new+(K6current–K6new)* Timer
1188 0323 6A18            TAMD    K6_B        ;Load interpolated K6 value
1189           *
1190           * Interpolate K7 and store the result in the working K7
1191           * register
1192           *
1193 0325 6256            TCX     K7_New_B    ;Put New K7 (adjusted to
1194 0327   14            TMAIX               ;   12 bits) in B register
1195 0328   1B            SALA4
1196 0329   1A            TAB
1197 032A   14            TMAIX               ;Put Current K7 (adjusted to
1198 032B   1B            SALA4               ;   12 bits) in A register
1199
1200 032C   2D            SBAAN               ;(K7current – K7new)
1201 032D 6261            TCX     SCALE
1202 032F   39            AXMA                ;(K7current – K7new) * Timer
1203 0330   2C            ABAAC               ;K7new+(K7current–K7new)* Timer
1204 0331 6A17            TAMD    K7_B        ;Load interpolated K7 value
1205           *
1206           * Interpolate K8 and store the result in the working K8
1207           * register
1208           *
1209 0333 6258            TCX     K8_New_B    ;Put New K8 (adjusted to
1210 0335   14            TMAIX               ;   12 bits) in B register
1211 0336   1B            SALA4
1212 0337   1A            TAB
1213
1214 0338   14            TMAIX               ;Put Current K8 (adjusted to
1215 0339   1B            SALA4               ;   12 bits) in A register
1216
1217 033A   2D            SBAAN               ;(K8current – K8new)
1218 033B 6261            TCX     SCALE
1219 033D   39            AXMA                ;(K8current – K8new) * Timer
1220 033E   2C            ABAAC               ;K8new+(K8current–K8new)* Timer
1221 033F 6A16            TAMD    K8_B        ;Load interpolated K8 value
1222           *
1223           * Interpolate K9 and store the result in the working K9
1224           * register
1225           *
1226 0341 625A            TCX     K9_New_B    ;Put New K9 (adjusted to
1227 0343   14            TMAIX               ;   12 bits) in B register
1228 0344   1B            SALA4
1229 0345   1A            TAB
1230
1231 0346   14            TMAIX               ;Put Current K9 (adjusted to
```

```
1232 0347   1B            SALA4               ;   12 bits) in A register
1233
1234 0348   2D            SBAAN               ;(K9current – K9new)
1235 0349 6261            TCX     SCALE
1236 034B   39            AXMA                ;(K9current – K9new) * Timer
1237 034C   2C            ABAAC               ;K9new+(K9current–K9new)* Timer
1238 034D 6A15            TAMD    K9_B        ;Load interpolated K9 value
1239          *
1240          * Interpolate K10 and store the result in the working K10
1241          * register
1242          *
1243 034F 625C            TCX     K10_New_B   ;Put New K10 (adjusted to
1244 0351   14            TMAIX               ;   12 bits) in B register
1245 0352   1B            SALA4
1246 0353   1A            TAB
1247
1248 0354   14            TMAIX               ;Put Current K10 (adjusted to
1249 0355   1B            SALA4               ;   12 bits) in A register
1250
1251 0356   2D            SBAAN               ;(K10current – K10new)
1252 0357 6261            TCX     SCALE
1253 0359   39            AXMA                ;(K10current – K10new) * Timer
1254 035A   2C            ABAAC               ;K10new+(K10current–K10new)* Timer
1255 035B 6A14            TAMD    K10_B       ;Load interpolated K10 value
1256          *
1257          * K11 and K12 are not needed for LPC 10, so they have
1258          * been commented out.
1259          *
1260          * Interpolate K11 and store the result in the working K11
1261          * register
1262          *
1263          *       TCX     K11_New_B       ;Put New K11 (adjusted to
1264          *       TMAIX                   ;  12 bits) in B register
1265          *       SALA4
1266          *       TAB
1267
1268          *       TMAIX                   ;Put Current K11 (adjusted to
1269          *       SALA4                   ;  12 bits) in A register
1270
1271          *       SBAAN                   ;(K11current – K11new)
1272          *       TCX     SCALE
1273          *       AXMA                    ;(K11current – K11new) * Timer
1274          *       ABAAC                   ;K11new+(K11current–K11new)* Timer
1275          *       TAMD    K11_B           ;Load interpolated K11 value
1276          *
1277          * Interpolate K12 and store the result in the working
1278          * K12 register
1279          *
1280          *       TCX     K12_New_B       ;Put New K12 (adjusted to
1281          *       TMAIX                   ;  12 bits) in B register
1282          *       SALA4
1283          *       TAB
1284
1285          *       TMAIX                   ;Put Current K12 (adjusted to
1286          *       SALA4                   ;  12 bits) in A register
1287
1288          *       SBAAN                   ;(K12current – K12new)
1289          *       TCX     SCALE
1290          *       AXMA                    ;(K12current – K12new) * Timer
1291          *       ABAAC                   ;K12new+(K12current–K12new)* Timer
1292          *       TAMD    K12_B           ;Load interpolated K12 value
1293          *
```

```
1294           * Set voiced/unvoiced mode according to current frame type.
1295           * This is done by loading Mode Register 2 address into the
1296           * X Register and adjusting the bit with an AND or OR operation.
1297           *
1298 035D   3B STMODE2     INTGR               ;Back to integer mode
1299 035E 6264             TCX     FLAGS_2
1300 0360 65FB             ANDCM   ~Update_Flg ;Signal that interp done
1301 0362 6680             TSTCM   Unv_Flg_Old ;Is current frame unvoiced?
1302 0364 436B             BR      SETUV2      ;   yes, set mode to unvoiced
1303 0366   20             CLX                 ;   no, ...
1304 0367   22             DECXN               ;point to mode register 2
1305 0368 657F             ANDCM   ~UNV        ;...set mode to voiced
1306
1307 036A   3D             RETN                ;Return from first call
1308
1309 036B   20 SETUV2      CLX
1310 036C   22             DECXN               ;point to mode register 2
1311 036D 6480             ORCM    UNV         ;Current frame is unvoiced, so
1312                                            ;   set mode to unvoiced.
1313           IRETN2
1314 036F   3D             RETN                ;Return from first call
1315
1316
1317
1318           * * * * *
1319           * Update the parameters for a new frame – CHANNEL 1
1320           * * * * *
1321           * * * * *
1322           * To prevent double updates, if the stored value of the
1323           * timer register is zero, then we need to change it to #7F.
1324           * If we do not do this, then the polling routine will
1325           * discover an underflow and call Update a second time.
1326           * We also need to check if inhibit bits are set (if new
1327           * word has just been started) because two frames need to
1328           * be pre-loaded before interpolation can happen.
1329           * * * * *
1330 0370 6260 UPDATE      TCX     TIMER       ;Get stored value
1331 0372   11             TMA                 ;  of Timer into A
1332
1333 0373 6000             ANEC    0           ;Is it zero?
1334 0375 437A             BR      UPDT00a     ;   no, do nothing
1335 0377 6E7F             TCA     #7F         ;   yes, replace value
1336 0379   16             TAM
1337
1338 037A 6262 UPDT00a     TCX     FLAGS
1339 037C 6610             TSTCM   Ch1Inh_1    ;is the first inhibit bit set?
1340 037E 4382             BR      UPDT00b     ;   yes, clear it
1341 0380 4386             BR      UPDT00c     ;   no, test second bit
1342
1343 0382 65EF UPDT00b     ANDCM   ~Ch1Inh_1   ;clear first inhibit bit
1344 0384 438E             BR      UPDT00
1345
1346 0386 6620 UPDT00c     TSTCM   Ch1Inh_2    ;is the second inhibit bit set?
1347 0388 438C             BR      UPDT00d     ;   yes, clear it
1348 038A 438E             BR      UPDT00      ;   no, continue with update
1349
1350 038C 65DF UPDT00d     ANDCM   ~Ch1Inh_2   ;clear second inhibit bit
1351
1352
1353           * * * * *
1354           * First we need to test to see if a stop frame was
1355           * encountered on the last pass through the routine.   If the
```

```
1356           * previous frame was a stop frame, we need to turn off the
1357           * synthesizer and stop speaking.
1358           * * * * *
1359 038E 6601 UPDT00      TSTCM   STOP1        ;Was stop frame encountered
1360 0390 4476             BR      ZeroKs_1a    ;   yes, zero channel 1 K parameters
1361
1362           * * * * *
1363           * Transfer the state of the previous frame to the Unvoiced
1364           * flag (Current).
1365           * * * * *
1366 0392 6263 UPDT01      TCX     FLAGS_1      ;Point to channel 1 flags
1367 0394 061E             CALL    XFER_FLAGS   ;  and xfer "new" to "old"
1368           * * * * *
1369           * Transfer the new frame parameters into the
1370           * storage location used for the old frame parameters.
1371           * * * * *
1372 0396 6220             TCX     EN_New_A     ;Point to ch. 1 data
1373 0398 06EF             CALL    XFER_PARMS   ;  and transfer parameters
1374           * * * * *
1375           * Turn on channel 1 and load address to speech
1376           * address register
1377           * * * * *
1378 039A 0717             CALL    INIT1
1379
1380           * * * * *
1381           * We have now discarded the "old" values by replacing
1382           * them with the "new" values.  We now need to read in
1383           * another frame of speech data and use them as the
1384           * new "new" values.
1385           * * * * *
1386           *----- ENERGY -----
1387 039C 6263             TCX     FLAGS_1      ;point to channel 1 flags
1388 039E 6E04             TCA     EBITS
1389 03A0 0635             CALL    C1GETN       ;Get coded energy
1390 03A2 6000             ANEC    ESILENCE     ;Is it a silent frame?
1391 03A4 43AC             BR      UPDT0        ;  No, continue
1392 03A6 6263             TCX     FLAGS_1
1393 03A8 6408             ORCM    Sil_Flg_New  ;  Yes, set silence flag
1394 03AA 4491             BR      ZeroKs_1     ;  and zero K params
1395
1396 03AC 600F UPDT0       ANEC    ESTOP        ;Is it a stop frame?
1397 03AE 43BA             BR      UPDT1        ;   no, continue
1398 03B0 6263             TCX     FLAGS_1
1399 03B2 6408             ORCM    Sil_Flg_New  ;yes, set flags
1400 03B4 6262             tcx     flags
1401 03B6 6401             orcm    stop1        ;signal stop frame reached on ch.1
1402 03B8 4491             BR      ZeroKs_1     ;  and zero K params
1403
1404 03BA 7751 UPDT1       ACAAC   TBLEN        ;Add table offset to energy
1405 03BC   6B             LUAA                 ;Get decoded energy
1406 03BD 6A20             TAMD    EN_New_A     ;Store the Energy in RAM
1407           * * * * *
1408           * If this is a silent frame, we are done with the Update
1409           * for channel 1, skip to the Update for channel 2.  If
1410           * the previous frame was silent, the new frame should be
1411           * spoken immediately with no ramp up due to interpolation.
1412           * * * * *
1413 03BF 6263             TCX     FLAGS_1
1414 03C1 6608             TSTCM   Sil_Flg_New  ;Is this a silent frame?
1415 03C3 44AD             BR      Update_2     ;   yes, go to channel 2 update
1416           * * * * *
1417           * A repeat frame will use the K parameters from the previous
```

```
1418           * frame.  If this is a repeat frame, we need to set a flag.
1419           * * * * *
1420 03C5 6E01 UPDT2       TCA     RBITS
1421 03C7 0635             CALL    C1GETN       ;Get the Repeat bit
1422 03C9 6701             TSTCA   #01          ;Is this a repeat frame?
1423 03CB 43CF             BR      SFLG1        ;  yes, set repeat flag
1424 03CD 43D3             BR      UPDT3        ;  no, continue
1425
1426 03CF 6263 SFLG1       TCX     FLAGS_1
1427 03D1 6402             ORCM    RepeatFlag   ;Set repeat flag
1428
1429           *----- PITCH -----
1430
1431 03D3 6E07 UPDT3       TCA     PBITS
1432 03D5 0635             CALL    C1GETN       ;Get coded pitch
1433 03D7 6000             ANEC    PUnVoiced    ;Is the frame unvoiced?
1434 03D9 43DF             BR      UPDT3A       ;   no, continue
1435 03DB 6263             TCX     FLAGS_1
1436 03DD 6410             ORCM    Unv_Flg_New  ;   yes, set unvoiced flag
1437
1438 03DF   2E UPDT3A      SALA                 ;Double coded pitch and
1439 03E0 7761             ACAAC   TBLPH        ;  add table offset to point
1440
1441 03E2   6D             LUAB                 ;Get decoded pitch
1442 03E3   3A             IAC
1443 03E4   6B             LUAA                 ;Get decoded fractional pitch
1444 03E5 6222             TCX     PH_New_A     ;Store the pitch and fractional
1445 03E7   2A             TBM                  ;  pitch in RAM
1446 03E8   21             IXC
1447 03E9   16             TAM
1448           * * * * *
1449           * If the voicing has changed with the new frame, then we
1450           * need to change the voicing in the mode register.
1451           * * * * *
1452 03EA 6263             TCX     FLAGS_1
1453 03EC 6610             TSTCM   Unv_Flg_New  ;Is the new frame unvoiced?
1454 03EE 43F2             BR      UPDT3B       ;   yes, test previous frame
1455 03F0 43FC             BR      VOICE        ;   no, go to voiced code
1456           * * * * *
1457           * The following code is reached if the new frame is
1458           * unvoiced. We inspect the flags to see if the previous
1459           * frame was either silent or voiced. If either condition
1460           * applies, then we branch to code which inhibits
1461           * interpolation.
1462           * * * * *
1463 03F2 6640 UPDT3B      TSTCM   Sil_Flg_Old  ;Was the previous frame silent?
1464 03F4 4402             BR      UPDT5        ;   yes, inhibit interpolation
1465
1466 03F6 6680             TSTCM   Unv_Flg_Old  ;Was the previous frame unvoiced
1467 03F8 4404             BR      UPDT4        ;   yes, no need to change anything
1468 03FA 4402             BR      UPDT5        ;   no, inhibit interpolation
1469           * * * * *
1470           * The following code is reached if the new frame is
1471           * voiced. We inspect the flags to see if the previous
1472           * frame was also voiced. If it was not, we need to inhibit
1473           * interpolation.
1474           * * * * *
1475 03FC 6680 VOICE       TSTCM   Unv_Flg_Old  ;Was the previous frame unvoiced?
1476 03FE 4402             BR      UPDT5        ;   yes, set no interpolation
1477 0400 4404             BR      UPDT4        ;   no, no need to change anything
1478
1479 0402 6420 UPDT5       ORCM    Int_Inh      ;Inhibit interpolation
```

```
1480           * * * * *
1481           * Now we test the repeat flag.  If the new frame is a repeat
1482           * frame, then the current values are used for the K factors,
1483           * so new values do not need to be loaded and we can exit the
1484           * routine now.
1485           * * * * *
1486 0404 6602 UPDT4       TSTCM   RepeatFlag   ;Is repeat flag set?
1487 0406 44AD             BR      Update_2     ;   yes, go to channel 2 update
1488           * * * * *
1489           * Now we need to load the "new" K factors (K1 through K10).
1490           * Each K factor is a 12 bit value which will be stored in
1491           * two bytes.  The most significant 8 bits in the first byte,
1492           * and the least significant 4 bits (called the fractional
1493           * value) in the second byte. For K3 through K12, the
1494           * fractional part is assumed to be zero.  K11 and K12 are
1495           * not used in LPC10 synthesis, and the code loading them is
1496           * commented out.  A coded factor is read into the A
1497           * register.  It is then converted to a pointer to a table
1498           * element which contains the uncoded factor. Since the K1
1499           * and K2 table elements consist of two bytes, the conversion
1500           * consists of doubling the coded factor and adding the
1501           * result to the start of the table.  Since the K3 & K4 table
1502           * elements consist of one byte, the coded factor is added
1503           * directly to the start of the table.  Once the pointer has
1504           * been set up, the uncoded factor is fetched and stored into
1505           * RAM.
1506           * * * * *
1507           *-----K1-----
1508 0408 6E06             TCA     K1BITS
1509 040A 0635             CALL    C1GETN       ;Get coded K1
1510 040C   2E             SALA                 ;Convert it to a
1511 040D 7861             ACAAC   TBLK1        ;  pointer to table element
1512 040F   6D             LUAB                 ;Fetch MSB of uncoded K1
1513 0410   3A             IAC
1514 0411   6B             LUAA                 ;Fetch fractional K1
1515 0412 6226             TCX     K1_New_A
1516 0414   2A             TBM                  ;Store uncoded K1
1517 0415   21             IXC
1518 0416   16             TAM                  ;Store fractional K1
1519           *-----K2-----
1520 0417 6E06             TCA     K2BITS
1521 0419 0635             CALL    C1GETN       ;Get coded K2
1522
1523 041B   2E             SALA                 ;Convert it to a
1524 041C 78E1             ACAAC   TBLK2        ;  pointer to table element
1525
1526 041E   6D             LUAB                 ;Fetch MSB of uncoded K2
1527 041F   3A             IAC
1528 0420   6B             LUAA                 ;Fetch fractional K2
1529 0421 622A             TCX     K2_New_A
1530 0423   2A             TBM                  ;Store uncoded K2
1531 0424   21             IXC
1532 0425   16             TAM                  ;Store fractional K2
1533           *-----K3-----
1534 0426 6E05             TCA     K3BITS
1535 0428 0635             CALL    C1GETN       ;Get Index into K3 table
1536 042A 7961             ACAAC   TBLK3        ;  and add offset of table
1537
1538 042C   6B             LUAA                 ;Get uncoded K3
1539 042D 6A2E             TAMD    K3_New_A     ;   and store it in RAM
1540           *-----K4-----
1541 042F 6E05             TCA     K4BITS
```

```
1542 0431 0635            CALL    C1GETN       ;Get Index into K4 table
1543 0433 7981            ACAAC   TBLK4        ;  and add offset of table
1544 0435   6B            LUAA                 ;Get uncoded K4
1545 0436 6A30            TAMD    K4_New_A     ;  and store it in RAM
1546           * * * * *
1547           * If this is an unvoiced frame, we only use four K factors,
1548           * so we load zeros to the rest of the K factors.  If this
1549           * is a voiced frame, load the rest of the uncoded factors.
1550           * * * * *
1551 0438 6263            TCX     FLAGS_1
1552 043A 6610            TSTCM   Unv_Flg_New  ;Is this an unvoiced frame?
1553 043C 44A0            BR      UNVC         ;   yes, zero rest of K factors
1554           * * * * *
1555           * The following code is executed if the new frame is
1556           * voiced.  Since we assume that the fractional parameter is
1557           * zero for the remaining K factors, the table elements are
1558           * only one byte long.  As with K3 and K4, the conversion to
1559           * a table pointer consists of adding the coded factor to the
1560           * start of the table.
1561           * * * * *
1562           *-----K5-----
1563 043E 6E04            TCA     K5BITS
1564 0440 0635            CALL    C1GETN       ;Get Index into K5 table
1565 0442 79A1            ACAAC   TBLK5        ;  and add offset of table
1566
1567 0444   6B            LUAA                 ;Get uncoded K5
1568 0445 6A32            TAMD    K5_New_A     ;  and store it in RAM
1569           *-----K6-----
1570 0447 6E04            TCA     K6BITS
1571 0449 0635            CALL    C1GETN       ;Get Index into K6 table
1572 044B 79B1            ACAAC   TBLK6        ;  and add offset of table
1573 044D   6B            LUAA                 ;Get uncoded K6
1574 044E 6A34            TAMD    K6_New_A     ;  and store it in RAM
1575           *-----K7-----
1576 0450 6E04            TCA     K7BITS
1577 0452 0635            CALL    C1GETN       ;Get Index into K7 table
1578 0454 79C1            ACAAC   TBLK7        ;  and add offset of table
1579 0456   6B            LUAA                 ;Get uncoded K7
1580 0457 6A36            TAMD    K7_New_A     ;  and store it in RAM
1581           *-----K8-----
1582 0459 6E03            TCA     K8BITS
1583 045B 0635            CALL    C1GETN       ;Get Index into K8 table
1584 045D 79D1            ACAAC   TBLK8        ;  and add offset of table
1585 045F   6B            LUAA                 ;Get uncoded K8
1586 0460 6A38            TAMD    K8_New_A     ;  and store it in RAM
1587           *-----K9-----
1588 0462 6E03            TCA     K9BITS
1589 0464 0635            CALL    C1GETN       ;Get Index into K9 table
1590 0466 79D9            ACAAC   TBLK9        ;  and add offset of table
1591 0468   6B            LUAA                 ;Get uncoded K9
1592 0469 6A3A            TAMD    K9_New_A     ;  and store it in RAM
1593           *-----K10-----
1594 046B 6E03            TCA     K10BITS
1595 046D 0635            CALL    C1GETN       ;Get Index into K10 table
1596 046F 79E1            ACAAC   TBLK10       ;  and add offset of table
1597 0471   6B            LUAA                 ;Get uncoded K10
1598 0472 6A3C            TAMD    K10_New_A    ;  and store it in RAM
1599           * * * * *
1600           * Since K11 and K12 are not used in LPC 10, the K11 and K12
1601           * code is commented out.
1602           * * * * *
1603           *-----K11-----
```

```
1604           *           TCA     K11BITS
1605           *           CALL    C1GETN       ;Get Index into K11 table
1606           *           ACAAC   TBLK11       ;  and add offset of table
1607           *           LUAA                 ;Get uncoded K11
1608           *           TAMD    K11_New_A    ;  and store it in RAM
1609           *-----K12-----
1610           *           TCA     K11BITS
1611           *           CALL    C1GETN       ;Get Index into K12 table
1612           *           ACAAC   TBLK12       ;  and add offset of table
1613           *           LUAA                 ;Get uncoded K12
1614           *           TAMD    K12_New_A    ;  and store it in RAM
1615           *-----
1616 0474 44AD             BR      UPDATE_2     ;Continue with channel 2 update
1617
1618           * * * * *
1619           * The following code (ZeroKs_1) is executed if the K parameters
1620           * need to be zeroed out.  If the new frame is a stop frame
1621           * or a silent frame, we zero out all K parameters and set
1622           * the energy to zero.  If the new frame is an unvoiced frame,
1623           * then we need to zero out the unused upper K parameters.
1624           * If the end of a word is reached, then we set appropriate
1625           * flags, clear energy, etc. (ZeroKs_1a)
1626           * * * * *
1627           *
1628           ZeroKs_1a
1629 0476   2F              CLA
1630 0477 6A01              TAMD    EN_A         ;Clear energy
1631 0479 6A21              TAMD    EN_Old_A
1632 047B 6A63              TAMD    FLAGS_1      ;Clear channel 1 flags
1633 047D 6262              TCX     FLAGS
1634 047F 65FE              ANDCM   ~Stop1       ;Reset Stop flag for next word
1635 0481 6430              ORCM    Ch1Inh_1+Ch1Inh_2    ;Set interp inhibit flags
1636 0483   2F              CLA
1637 0484 7FF3              ACAAC   #FF3         ;Initialize pitch
1638 0486   18              TAX
1639 0487   2F              CLA
1640 0488 7162              ACAAC   #162
1641 048A   16              TAM
1642 048B 406E              BR      Nxt_Word1    ;Get next word
1643           ZeroKs_1b
1644 048D 660C              TSTCM   CH1END+CH2END  ;Have we reached the end for both chan-
nels
1645 048F 45E2              BR      STOP           ;  yes, stop synthesis
1646
1647 0491   2F ZeroKs_1     CLA
1648 0492 6A20              TAMD    EN_New_A     ;Kill Energy
1649 0494 6A26              TAMD    K1_New_A     ;Kill K1
1650 0496 6A27              TAMD    K1_New_A+1
1651 0498 6A2A              TAMD    K2_New_A     ;Kill K2
1652 049A 6A2B              TAMD    K2_New_A+1
1653 049C 6A2E              TAMD    K3_New_A     ;Kill K3
1654 049E 6A30              TAMD    K4_New_A     ;Kill K4
1655 04A0   2F UNVC         CLA
1656 04A1 6A32              TAMD    K5_New_A     ;Kill K5
1657 04A3 6A34              TAMD    K6_New_A     ;Kill K6
1658 04A5 6A36              TAMD    K7_New_A     ;Kill K7
1659 04A7 6A38              TAMD    K8_New_A     ;Kill K8
1660 04A9 6A3A              TAMD    K9_New_A     ;Kill K9
1661 04AB 6A3C              TAMD    K10_New_A    ;Kill K10
1662           *           TAMD    K11_New_A    ;Kill K11
1663           *           TAMD    K12_New_A    ;Kill K12
1664
```

```
1665                                            ;Continue with channel 2 update
1666           * * * * *
1667           * Update the parameters for a new frame – CHANNEL 2
1668           * * * * *
1669           * * * * *
1670           * We need to check if inhibit bits are set (if new
1671           * word has just been started) because two frames need to
1672           * be pre–loaded before interpolation can happen.
1673           * * * * *
1674           UPDATE_2
1675 04AD 6262             TCX     FLAGS
1676 04AF 6640             TSTCM   Ch2Inh_1     ;is the first inhibit bit set?
1677 04B1 44B5             BR      UPDT02a      ;   yes, clear it
1678 04B3 44B9             BR      UPDT02b      ;   no, test second bit
1679
1680 04B5 65BF UPDT02a     ANDCM   ~Ch2Inh_1    ;clear first inhibit bit
1681 04B7 44C1             BR      UPDT02
1682
1683 04B9 6680 UPDT02b     TSTCM   Ch2Inh_2     ;is the second inhibit bit set?
1684 04BB 44BF             BR      UPDT02c      ;   yes, clear it
1685 04BD 44C1             BR      UPDT02       ;   no, continue with update
1686
1687 04BF 657F UPDT02c     ANDCM   ~Ch2Inh_2    ;clear second inhibit bit
1688
1689
1690           * * * * *
1691           * First we need to test to see if a stop frame was
1692           * encountered on the last pass through the routine.   If the
1693           * previous frame was a stop frame, we need to turn off the
1694           * synthesizer and stop speaking.
1695           * * * * *
1696 04C1 6262 UPDT02      TCX     FLAGS        ;Point to flags
1697 04C3 6602             TSTCM   STOP2        ;Was channel 2 stop frame encountered
1698 04C5 45AB             BR      ZeroKs_2a    ;  no, zero channel 2 K parameters
1699
1700           * * * * *
1701           * Transfer the state of the previous frame to the Unvoiced
1702           * flag (Current).
1703           * * * * *
1704 04C7 6264 UPDT_2a     TCX     FLAGS_2      ;Point to channel 2 flags
1705 04C9 061E             CALL    XFER_FLAGS   ;  and transfer ”new” to ”old”
1706           * * * * *
1707           * Transfer the new frame parameters into the
1708           * storage location used for the old frame parameters.
1709           * * * * *
1710 04CB 6240             TCX     EN_New_B     ;Point to ch. 2 data
1711 04CD 06EF             CALL    XFER_PARMS   ;  and transfer parameters
1712           * * * * *
1713           * Turn on channel 2 and load address to speech
1714           * address register
1715           * * * * *
1716 04CF 0734             CALL    INIT2
1717
1718           * * * * *
1719           * We have now discarded the ”old” values by replacing
1720           * them with the ”new” values.  We now need to read in
1721           * another frame of speech data and use them as the
1722           * new ”new” values.
1723           * * * * *
1724           *----- ENERGY -----
1725 04D1 6264             TCX     FLAGS_2
1726 04D3 6E04             TCA     EBITS
```

```
1727 04D5 0692            CALL    C2GETN       ;Get coded energy
1728 04D7 6000            ANEC    ESILENCE     ;Is it a silent frame?
1729 04D9 44E1            BR      UPDT0_2      ;  No, continue
1730 04DB 6264            TCX     FLAGS_2
1731 04DD 6408            ORCM    Sil_Flg_New  ;  Yes, set silence flag
1732 04DF 45C4            BR      ZeroKs_2     ;  and zero K params
1733
1734 04E1 600F UPDT0_2    ANEC    ESTOP        ;Is it a stop frame?
1735 04E3 44EF            BR      UPDT1_2      ;  no, continue
1736 04E5 6264            TCX     FLAGS_2
1737 04E7 6408            ORCM    Sil_Flg_New  ;  yes, set flags
1738 04E9 6262            tcx     flags
1739 04EB 6402            orcm    stop2
1740 04ED 45C4            BR      ZeroKs_2     ;  and zero K params
1741
1742 04EF 7751 UPDT1_2    ACAAC   TBLEN        ;Add table offset to energy
1743 04F1   6B            LUAA                 ;Get decoded energy
1744 04F2 6A40            TAMD    EN_New_B     ;Store the Energy in RAM
1745          * * * * *
1746          * If this is a silent frame, we are done with the update If
1747          * the previous frame was silent, the new frame should be
1748          * spoken immediately with no ramp up due to interpolation
1749          * * * * *
1750 04F4 6264            TCX     FLAGS_2
1751 04F6 6608            TSTCM   Sil_Flg_New  ;Is this a silent frame?
1752 04F8 45FE            BR      RTN          ;   yes, exit
1753          * * * * *
1754          * A repeat frame will use the K parameter from the previous
1755          * frame.  If this is a repeat frame, we need to set a flag.
1756          * * * * *
1757 04FA 6E01 UPDT2_2     TCA     RBITS
1758 04FC 0692            CALL    C2GETN       ;Get the Repeat bit
1759 04FE 6701            TSTCA   #01          ;Is this a repeat frame?
1760 0500 4504            BR      SFLG1_2      ;  yes, set repeat flag
1761 0502 4508            BR      UPDT3_2      ;  no, continue
1762
1763 0504 6264 SFLG1_2    TCX     FLAGS_2
1764 0506 6402            ORCM    RepeatFlag   ;Set repeat flag
1765
1766          *----- PITCH -----
1767
1768 0508 6E07 UPDT3_2     TCA     PBITS
1769 050A 0692            CALL    C2GETN       ;Get coded pitch
1770 050C 6000            ANEC    PUnVoiced    ;Is the frame unvoiced?
1771 050E 4514            BR      UPDT3A_2     ;  no, continue
1772 0510 6264            TCX     FLAGS_2
1773 0512 6410            ORCM    Unv_Flg_New  ;  yes, set unvoiced flag
1774
1775 0514   2E UPDT3A_2    SALA                 ;Double coded pitch and
1776 0515 7761            ACAAC   TBLPH        ;  add table offset
1777
1778 0517   6D            LUAB                 ;Get decoded pitch
1779 0518   3A            IAC
1780 0519   6B            LUAA                 ;Get decoded fractional pitch
1781 051A 6242            TCX     PH_New_B     ;Store the pitch and fractional
1782 051C   2A            TBM                  ;  pitch in RAM
1783 051D   21            IXC
1784 051E   16            TAM
1785          * * * * *
1786          * If the voicing has changed with the new frame, then we
1787          * need to change the voicing in the mode register.
1788          * * * * *
```

```
1789 051F 6264             TCX     FLAGS_2
1790 0521 6610             TSTCM   Unv_Flg_New  ;Is the new frame unvoiced?
1791 0523 4527             BR      UPDT3B_2     ;  yes, continue
1792 0525 4531             BR      VOICE_2      ;  no, go to voiced code
1793           * * * * *
1794           * The following code is reached if the new frame is
1795           * unvoiced. We inspect the flags to see if the previous
1796           * frame was either silent or voiced. If either condition
1797           * applies, then we branch to code which inhibits
1798           * interpolation.
1799           * * * * *
1800 0527 6640 UPDT3B_2    TSTCM   Sil_Flg_Old  ;Was the previous frame silent?
1801 0529 4537             BR      UPDT5_2      ;  yes, inhibit interpolation
1802
1803 052B 6680             TSTCM   Unv_Flg_Old  ;Was the previous frame unvoiced?
1804 052D 4539             BR      UPDT4_2      ;  yes, no need to change anything
1805 052F 4537             BR      UPDT5_2      ;  no, inhibit interpolation
1806           * * * * *
1807           * The following code is reached if the new frame is
1808           * voiced. We inspect the flags to see if the previous
1809           * frame was also voiced. If it was not, we need to inhibit
1810           * interpolation.
1811           * * * * *
1812 0531 6680 VOICE_2     TSTCM   Unv_Flg_Old  ;Was the previous frame unvoiced?
1813 0533 4537             BR      UPDT5_2      ;  yes, set no interpolation
1814 0535 4539             BR      UPDT4_2      ;  no, no need to change anything
1815
1816 0537 6420 UPDT5_2     ORCM    Int_Inh      ;Inhibit interpolation
1817           * * * * *
1818           * Now we test the repeat flag.  If the new frame is a repeat
1819           * frame, then the current values are used for the K factors,
1820           * so new values do not need to be loaded and we can exit the
1821           * routine now.
1822           * * * * *
1823 0539 6602 UPDT4_2     TSTCM   RepeatFlag   ;Is repeat flag set?
1824 053B 45FE             BR      RTN          ;   yes, exit routine
1825           * * * * *
1826           * Now we need to load the ”new” K factors (K1 through K10).
1827           * Each K factor is a 12 bit value which will be stored in
1828           * two bytes.  The most significant 8 bits in the first byte,
1829           * and the least significant 4 bits (called the fractional
1830           * value) in the second byte. For K3 through K12, the
1831           * fractional part is assumed to be zero.  K11 and K12 are
1832           * not used in LPC10 synthesis, and the code loading them is
1833           * commented out.  A coded factor is read into the A
1834           * register.  It is then converted to a pointer to a table
1835           * element which contains the uncoded factor. Since the K1
1836           * and K2 table elements consist of two bytes, the conversion
1837           * consists of doubling the coded factor and adding the
1838           * result to the start of the table.  Since the K3 & K4 table
1839           * elements consist of one byte, the coded factor is added
1840           * directly to the start of the table.  Once the pointer has
1841           * been set up, the uncoded factor is fetched and stored into
1842           * RAM.
1843           * * * * *
1844           *-----K1-----
1845 053D 6E06             TCA     K1BITS
1846 053F 0692             CALL    C2GETN       ;Get coded K1
1847 0541   2E             SALA                 ;Convert it to a
1848 0542 7861             ACAAC   TBLK1        ;  pointer to table element
1849 0544   6D             LUAB                 ;Fetch MSB of uncoded K1
1850 0545   3A             IAC
```

```
1851 0546   6B             LUAA                ;Fetch fractional K1
1852 0547 6246             TCX     K1_New_B
1853 0549   2A             TBM                 ;Store uncoded K1
1854 054A   21             IXC
1855 054B   16             TAM                 ;Store fractional K1
1856            *-----K2-----
1857 054C 6E06             TCA     K2BITS
1858 054E 0692             CALL    C2GETN      ;Get coded K2
1859
1860 0550   2E             SALA                ;Convert it to a
1861 0551 78E1             ACAAC   TBLK2       ;  pointer to table element
1862
1863 0553   6D             LUAB                ;Fetch MSB of uncoded K2
1864 0554   3A             IAC
1865 0555   6B             LUAA                ;Fetch fractional K2
1866 0556 624A             TCX     K2_New_B
1867 0558   2A             TBM                 ;Store uncoded K2
1868 0559   21             IXC
1869 055A   16             TAM                 ;Store fractional K2
1870            *-----K3-----
1871 055B 6E05             TCA     K3BITS
1872 055D 0692             CALL    C2GETN      ;Get Index into K3 table
1873 055F 7961             ACAAC   TBLK3       ;  and add offset of table
1874
1875 0561   6B             LUAA                ;Get uncoded K3
1876 0562 6A4E             TAMD    K3_New_B    ;  and store it in RAM
1877            *-----K4-----
1878 0564 6E05             TCA     K4BITS
1879 0566 0692             CALL    C2GETN      ;Get Index into K4 table
1880 0568 7981             ACAAC   TBLK4       ;  and add offset of table
1881 056A   6B             LUAA                ;Get uncoded K4
1882 056B 6A50             TAMD    K4_New_B    ;  and store it in RAM
1883            * * * * *
1884            * If this is an unvoiced frame, we only use four K factors,
1885            * so we load zeroes to the rest of the K factors.  If this
1886            * is a voiced frame, load the rest of the uncoded factors.
1887            * * * * *
1888 056D 6264             TCX     FLAGS_2
1889 056F 6610             TSTCM   Unv_Flg_New ;Is this an unvoiced frame?
1890 0571 45D3             BR      UNVC_2      ;  Yes, zero rest of factors
1891            * * * * *
1892            * The following code is executed if the new frame is
1893            * voiced.  Since we assume that the fractional parameter is
1894            * zero for the remaining K factors, the table elements are
1895            * only one byte long.  As with K3 and K4, the conversion to
1896            * a table pointer consists of adding the coded factor to the
1897            * start of the table.
1898            * * * * *
1899            *-----K5-----
1900 0573 6E04             TCA     K5BITS
1901 0575 0692             CALL    C2GETN      ;Get Index into K5 table
1902 0577 79A1             ACAAC   TBLK5       ;  and add offset of table
1903
1904 0579   6B             LUAA                ;Get uncoded K5
1905 057A 6A52             TAMD    K5_New_B    ;  and store it in RAM
1906            *-----K6-----
1907 057C 6E04             TCA     K6BITS
1908 057E 0692             CALL    C2GETN      ;Get Index into K6 table
1909 0580 79B1             ACAAC   TBLK6       ;  and add offset of table
1910 0582   6B             LUAA                ;Get uncoded K6
1911 0583 6A54             TAMD    K6_New_B    ;  and store it in RAM
1912            *-----K7-----
```

```
1913 0585 6E04            TCA     K7BITS
1914 0587 0692            CALL    C2GETN       ;Get Index into K7 table
1915 0589 79C1            ACAAC   TBLK7        ;  and add offset of table
1916 058B   6B            LUAA                 ;Get uncoded K7
1917 058C 6A56            TAMD    K7_New_B     ;  and store it in RAM
1918           *-----K8-----
1919 058E 6E03            TCA     K8BITS
1920 0590 0692            CALL    C2GETN       ;Get Index into K8 table
1921 0592 79D1            ACAAC   TBLK8        ;  and add offset of table
1922 0594   6B            LUAA                 ;Get uncoded K8
1923 0595 6A58            TAMD    K8_New_B     ;  and store it in RAM
1924           *-----K9-----
1925 0597 6E03            TCA     K9BITS
1926 0599 0692            CALL    C2GETN       ;Get Index into K9 table
1927 059B 79D9            ACAAC   TBLK9        ;  and add offset of table
1928 059D   6B            LUAA                 ;Get uncoded K9
1929 059E 6A5A            TAMD    K9_New_B     ;  and store it in RAM
1930           *-----K10-----
1931 05A0 6E03            TCA     K10BITS
1932 05A2 0692            CALL    C2GETN       ;Get Index into K10 table
1933 05A4 79E1            ACAAC   TBLK10       ;  and add offset of table
1934 05A6   6B            LUAA                 ;Get uncoded K10
1935 05A7 6A5C            TAMD    K10_New_B    ;  and store it in RAM
1936           * * * * *
1937           * Since K11 and K12 are not used in LPC 10, the K11 and K12
1938           * code is commented out.
1939           * * * * *
1940           *-----K11-----
1941           *           TCA     K11BITS
1942           *           CALL    C2GETN       ;Get Index into K11 table
1943           *           ACAAC   TBLK11       ;  and add offset of table
1944           *           LUAA                 ;Get uncoded K11
1945           *           TAMD    K11_New_B    ;  and store it in RAM
1946           *-----K12-----
1947           *           TCA     K11BITS
1948           *           CALL    C2GETN       ;Get Index into K12 table
1949           *           ACAAC   TBLK12       ;  and add offset of table
1950           *           LUAA                 ;Get uncoded K12
1951           *           TAMD    K12_New_B    ;  and store it in RAM
1952           *-----
1953 05A9 45FE            BR      RTN          ;Exit
1954
1955           * * * * *
1956           * The following code is executed if the K parameters need to
1957           * be zeroed out.  If  the new frame is a stop frame or a
1958           * silent frame, we zero out all K parameters and set the
1959           * energy to zero.  If the new frame is an unvoiced frame,
1960           * then we need to zero out the unused upper K parameters.
1961           * If the end of a word is reached, then we set appropriate
1962           * flags, clear energy, etc. (ZeroKs_2a)
1963           * * * * *
1964           *
1965 05AB   2F ZeroKs_2a   CLA
1966 05AC 6A11            TAMD    EN_B         ;clear energy
1967 05AE 6A41            TAMD    EN_Old_B
1968 05B0 6A64            TAMD    FLAGS_2      ;clear channel 2 flags
1969 05B2 6262            TCX     FLAGS
1970 05B4 64C0            ORCM    Ch2Inh_1+Ch2Inh_2  ;set interp inhibit flags
1971 05B6 65FD            ANDCM   ~Stop2       ;reset stop flag for next word
1972 05B8   2F            CLA                  ;initialize pitch
1973 05B9 7FF1            ACAAC   #FF1
1974 05BB   18            TAX
```

```
1975 05BC   2F             CLA
1976 05BD   16             TAM
1977 05BE 40A9             BR      Nxt_Word2    ;get next word
1978           ZeroKs_2b
1979 05C0 660C             TSTCM   CH1END+CH2END  ;have we reached the end for both chan-
nels?
1980 05C2 45E2             BR      STOP           ;  yes, stop synthesis
1981
1982 05C4   2F ZeroKs_2    CLA
1983 05C5 6A40             TAMD    EN_New_B     ;Kill Energy
1984 05C7 6A46             TAMD    K1_New_B     ;Kill K1
1985 05C9 6A47             TAMD    K1_New_B+1
1986 05CB 6A4A             TAMD    K2_New_B     ;Kill K2
1987 05CD 6A4B             TAMD    K2_New_B+1
1988 05CF 6A4E             TAMD    K3_New_B     ;Kill K3
1989 05D1 6A50             TAMD    K4_New_B     ;Kill K4
1990 05D3   2F UNVC_2      CLA
1991 05D4 6A52             TAMD    K5_New_B     ;Kill K5
1992 05D6 6A54             TAMD    K6_New_B     ;Kill K6
1993 05D8 6A56             TAMD    K7_New_B     ;Kill K7
1994 05DA 6A58             TAMD    K8_New_B     ;Kill K8
1995 05DC 6A5A             TAMD    K9_New_B     ;Kill K9
1996 05DE 6A5C             TAMD    K10_New_B    ;Kill K10
1997           *           TAMD    K11_New_B    ;Kill K11
1998           *           TAMD    K12_New_B    ;Kill K12
1999
2000 05E0 45FE             BR      RTN          ;Exit
2001
2002
2003           * * * * *
2004           * STOP AND RETURN
2005           * * * * *
2006           * The following code has two entry points. STOP is reached
2007           * if the current frame is a stop flag, it turns off
2008           * synthesis and returns to the program.  RTN is the general
2009           * exit point for the UPDATE routine, it sets the Update flag
2010           * and leaves the routine.
2011           * * * * *
2012 05E2   20 STOP         CLX
2013 05E3   22             DECXN                ;point to mode register 2
2014 05E4 657D             ANDCM   ~(LPC+UNV)   ;Turn off LPC synthesis for ch.2
2015
2016 05E6   2F             cla
2017 05E7   3A loop1c      iac                  ; temporary fix for 3X problem
2018 05E8   EA             sbr      outc        ; |
2019 05E9   E7             sbr      loop1c      ; |
2020                                            ; |
2021           outc
2022 05EA   22             DECXN                ;point to mode register 1
2023 05EB 657D             ANDCM   ~(LPC+UNV)   ;Turn off LPC synthesis for ch.1
2024                                            ;  and go back to voiced for next word
2025 05ED 6404             ORCM    PCM          ;Enable PCM mode for channel 1
2026 05EF   2F             CLA
2027 05F0   1C             TASYN                ;Write a zero to the DAC
2028 05F1 6EFA             TCA     #FA
2029 05F3   3A BACK        IAC                  ;Wait for minimum of 30
2030 05F4 45F8             BR      out          ;  instruction cycles
2031 05F6 45F3             BR      back
2032 05F8   20 OUT         CLX
2033 05F9   22             DECXN                ;point to mode register 1
2034 05FA   22             DECXN
2035 05FB 65FB             ANDCM   ~PCM         ;disable PCM for channel 1
```

```
2036 05FD   3D             RETN                ;return from routine
2037
2038 05FE   17 RTN         TTMA                ;Get current timer
2039 05FF   1A             TAB                 ;  and store it away
2040
2041 0600   17 NOTDIFF     TTMA                ;Wait for timer to decrement
2042 0601   2D             SBAAN
2043 0602 6000             ANEC    0
2044 0604 4608             BR      DIFF
2045 0606 4600             BR      NOTDIFF
2046
2047 0608   17 DIFF        TTMA                ;Subtract 128 from value
2048 0609 7080             ACAAC   #80
2049 060B   1E             TATM
2050
2051 060C 6263             TCX     FLAGS_1     ;Set a flag indicating that
2052 060E 6404             ORCM    Update_Flg  ;  the parameters are updated for ch.1
2053 0610 6264             TCX     FLAGS_2     ;Set a flag indicating that
2054 0612 6404             ORCM    Update_Flg  ;  the parameters are updated for ch.2
2055
2056           *-------Return to current location------------*
2057
2058 0614   20             CLX
2059 0615   22             DECXN               ;point to mode register 1
2060 0616   22             DECXN
2061 0617 6602             TSTCM   LPC         ;Are we speaking yet?
2062 0619 461C             BR      RTN1        ;   yes, reenable interrupt
2063 061B   3D             RETN                ;   no, return for more data
2064
2065 061C 4139 RTN1        BR      SPEAK_LP    ;Go back to loop
2066
2067
2068           *-----------------------------------------------------*
2069           * XFER_FLAGS                                          *
2070           *                                                     *
2071           * Transfers the "New" frame flags to "Old" frame flags    *
2072           * and zeros the "New" flags so that they can be           *
2073           * conditionally set later.                            *
2074           *                                                     *
2075           * ASSUMES:  X register is set point appropriate flags     *
2076           *           register                                  *
2077           *                                                     *
2078           *-----------------------------------------------------*
2079           XFER_FLAGS
2080           *-----------Transfer silent frame flag--------------------*
2081
2082 061E 6610             TSTCM   Unv_Flg_New ;Was new frame unvoiced?
2083 0620 4626             BR      SUNVL       ;  yes, set old frame unvoiced
2084 0622 657F             ANDCM   ~Unv_Flg_Old ;  no, clear old frame voiced
2085 0624 4628             BR      TSIL        ;      and continue
2086
2087 0626 6480 SUNVL       ORCM    Unv_Flg_Old ;Set old frame unvoiced.
2088
2089           *-----------Transfer silent frame flag--------------------*
2090
2091 0628 6608 TSIL        TSTCM   Sil_Flg_New ;Was new frame silent?
2092 062A 4630             BR      SSIL        ;  yes, set old frame silent
2093 062C 65BF             ANDCM   ~Sil_Flg_Old ;  no, clear old frame not silent
2094 062E 4632             BR      ZROFLG      ;      and continue
2095
2096 0630 6440 SSIL        ORCM    Sil_Flg_Old ;Set old frame silent
2097
```

```
2098           * * * * *
2099           * Reset the Repeat Flag, new Silence Flag, new Unvoiced
2100           * Flag, and Interpolation Inhibit flag so that new
2101           * values can be loaded in this routine.
2102           * * * * *
2103 0632 65C5 ZROFLG      ANDCM   #C5
2104 0634   3D             RETN
2105
2106
2107           ************************************************
2108           *  C1GETN                                      *
2109           *                                              *
2110           *  This subroutine gets the number of bits in  *
2111           *  the A register and updates the address sent *
2112           *  to the speech address register for channel 1. *
2113           ************************************************
2114           C1GETN
2115 0635   1A         TAB                     ; save # of bits to get
2116 0636   2E         SALA                    ; adjust to 2 byte vector siz
2117 0637 7638         ACAAC   $+1             ; add table start location
2118 0639   1F         BRA                     ; vector to jump table
2119
2120 063A 464A         BR      C1Get1          ; branch to get 1 bit
2121 063C 4650         BR      C1Get2          ; branch to get 2 bits
2122 063E 4656         BR      C1Get3          ; branch to get 3 bits
2123 0640 465C         BR      C1Get4          ; branch to get 4 bits
2124 0642 4662         BR      C1Get5          ; branch to get 5 bits
2125 0644 4668         BR      C1Get6          ; branch to get 6 bits
2126 0646 466E         BR      C1Get7          ; branch to get 7 bits
2127 0648 4674         BR      C1Get8          ; branch to get 8 bits
2128
2129 064A   2F C1Get1  CLA
2130 064B   30 C1Get1a GET     1               ; get 1 bit
2131 064C 467A         BR      C1GetN1
2132 064E 467A         BR      C1GetN1
2133
2134 0650   2F C1Get2  CLA
2135 0651   31 C1Get2a GET     2               ; get 2 bits
2136 0652 467A         BR      C1GetN1
2137 0654 467A         BR      C1GetN1
2138
2139 0656   2F C1Get3  CLA
2140 0657   32 C1Get3a GET     3               ; get 3 bits
2141 0658 467A         BR      C1GetN1
2142 065A 467A         BR      C1GetN1
2143
2144 065C   2F C1Get4  CLA
2145 065D   33 C1Get4a GET     4               ; get 4 bits
2146 065E 467A         BR      C1GetN1
2147 0660 467A         BR      C1GetN1
2148
2149 0662   2F C1Get5  CLA
2150 0663   33         GET     4               ; get 4 bits
2151 0664 464B         BR      C1Get1a         ;   + 1 bit
2152 0666 464B         BR      C1Get1a
2153
2154 0668   2F C1Get6  CLA
2155 0669   33         GET     4               ; get 4 bits
2156 066A 4651         BR      C1Get2a         ;   + 2 bits
2157 066C 4651         BR      C1Get2a
2158
2159 066E   2F C1Get7  CLA
```

```
2160 066F   33         GET     4               ; get 4 bits
2161 0670 4657         BR      C1Get3a         ;   + 3 bits
2162 0672 4657         BR      C1Get3a
2163
2164 0674   2F C1Get8  CLA
2165 0675   33         GET     4               ; get 4 bits
2166 0676 465D         BR      C1Get4a         ;   + 4 bits
2167 0678 465D         BR      C1Get4a
2168
2169 067A   12 C1GetN1 XBA                     ; retrieve # of bits to get
2170 067B 6269         TCX     OFFSET_1        ; point to offset
2171 067D   28         AMAAC                   ; add offset to # of bits
2172 067E 6308         AGEC    8               ; is offset > 8?
2173 0680 4685         BR      C1GetN2         ;   yes, increment address
2174
2175 0682   16         TAM                     ;   no, store new offset
2176 0683 4690         BR      C1GetNX         ;    and exit
2177
2178 0685 7FF8 C1GetN2 ACAAC   -8              ; subtract 8 from offset
2179 0687   16         TAM                     ; store new offset
2180 0688   22         DECXN                   ; point to address LSB
2181 0689   26         INCMC                   ; increment address LSB, overflow
2182 068A 468E         BR      C1GetN3         ;   yes, increment address MSB
2183 068C 4690         BR      C1GetNX         ;   no, exit
2184
2185 068E   22 C1GetN3 DECXN                   ; point to address MSB
2186 068F   26         INCMC                   ; increment address MSB
2187
2188 0690   12 C1GetNX XBA                     ; retrieve data
2189 0691   3D         RETN                    ; return from call
2190
2191
2192           ************************************************
2193           *  C2GETN                                      *
2194           *                                              *
2195           *  This subroutine gets the number of bits in  *
2196           *  the A register and updates the address sent *
2197           *  to the speech address register for channel 2. *
2198           ************************************************
2199           C2GETN
2200 0692   1A         TAB                     ; save # of bits to get
2201 0693   2E         SALA                    ; adjust to 2 byte vector size
2202 0694 7695         ACAAC   $+1             ; add table start location
2203 0696   1F         BRA                     ; vector to jump table
2204
2205 0697 46A7         BR      C2Get1          ; branch to get 1 bit
2206 0699 46AD         BR      C2Get2          ; branch to get 2 bits
2207 069B 46B3         BR      C2Get3          ; branch to get 3 bits
2208 069D 46B9         BR      C2Get4          ; branch to get 4 bits
2209 069F 46BF         BR      C2Get5          ; branch to get 5 bits
2210 06A1 46C5         BR      C2Get6          ; branch to get 6 bits
2211 06A3 46CB         BR      C2Get7          ; branch to get 7 bits
2212 06A5 46D1         BR      C2Get8          ; branch to get 8 bits
2213
2214 06A7   2F C2Get1  CLA
2215 06A8   30 C2Get1a GET     1               ; get 1 bit
2216 06A9 46D7         BR      C2GetN1
2217 06AB 46D7         BR      C2GetN1
2218
2219 06AD   2F C2Get2  CLA
2220 06AE   31 C2Get2a GET     2               ; get 2 bits
2221 06AF 46D7         BR      C2GetN1
```

```
2222 06B1 46D7         BR      C2GetN1
2223
2224 06B3   2F C2Get3  CLA
2225 06B4   32 C2Get3a GET     3               ; get 3 bits
2226 06B5 46D7         BR      C2GetN1
2227 06B7 46D7         BR      C2GetN1
2228
2229 06B9   2F C2Get4  CLA
2230 06BA   33 C2Get4a GET     4               ; get 4 bits
2231 06BB 46D7         BR      C2GetN1
2232 06BD 46D7         BR      C2GetN1
2233
2234 06BF   2F C2Get5  CLA
2235 06C0   33         GET     4               ; get 4 bits
2236 06C1 46A8         BR      C2Get1a         ;   + 1 bit
2237 06C3 46A8         BR      C2Get1a
2238
2239 06C5   2F C2Get6  CLA
2240 06C6   33         GET     4               ; get 4 bits
2241 06C7 46AE         BR      C2Get2a         ;   + 2 bits
2242 06C9 46AE         BR      C2Get2a
2243
2244 06CB   2F C2Get7  CLA
2245 06CC   33         GET     4               ; get 4 bits
2246 06CD 46B4         BR      C2Get3a         ;   + 3 bits
2247 06CF 46B4         BR      C2Get3a
2248
2249 06D1   2F C2Get8  CLA
2250 06D2   33         GET     4               ; get 4 bits
2251 06D3 46BA         BR      C2Get4a         ;   + 4 bits
2252 06D5 46BA         BR      C2Get4a
2253
2254 06D7   12 C2GetN1 XBA                     ; retrieve # of bits to get
2255 06D8 626C         TCX     OFFSET_2        ; point to offset
2256 06DA   28         AMAAC                   ; add offset to # of bits
2257 06DB 6308         AGEC    8               ; is offset > 8?
2258 06DD 46E2         BR      C2GetN2         ;   yes, increment address
2259
2260 06DF   16         TAM                     ;   no, store new offset
2261 06E0 46ED         BR      C2GetNX         ;    and exit
2262
2263 06E2 7FF8 C2GetN2 ACAAC   -8              ; subtract 8 from offset
2264 06E4   16         TAM                     ; store new offset
2265 06E5   22         DECXN                   ; point to address LSB
2266 06E6   26         INCMC                   ; increment address LSB, overflow
2267 06E7 46EB         BR      C2GetN3         ;   yes, increment address MSB
2268 06E9 46ED         BR      C2GetNX         ;   no, exit
2269
2270 06EB   22 C2GetN3 DECXN                   ; point to address MSB
2271 06EC   26         INCMC                   ; increment address MSB
2272
2273 06ED   12 C2GetNX XBA                     ; retrieve data
2274 06EE   3D         RETN                    ; return from call
2275
2276
2277           *-------------------------------------------------------*
2278           * XFER_PARMS                                            *
2279           *                                                       *
2280           * Transfers the "New" frame synthesis parameters to the  *
2281           * "Old" frame synthesis parameters so that a new set of  *
2282           * new parameters can be read in.                         *
2283           *                                                       *
```

```
2284           * ASSUMES:  X register is set point to start of data      *
2285           *           block                                         *
2286           *                                                         *
2287           *---------------------------------------------------------*
2288 06EF   14 XFER_PARMS  TMAIX               ;Transfer new frame En parameter
2289 06F0   13             TAMIX               ;  to current frame location
2290           *-----PITCH-----
2291 06F1   14             TMAIX               ;Transfer new frame pitch
2292 06F2   21             IXC                 ;   MSB from new to
2293 06F3   16             TAM                 ;   old frame location
2294
2295 06F4   22             DECXN               ;Point to fractional pitch
2296
2297 06F5   14             TMAIX               ;Transfer new fractional pitch
2298 06F6   21             IXC                 ;   from new to old
2299 06F7   13             TAMIX               ;   frame location
2300           *-----K1-----
2301 06F8   14             TMAIX               ;Transfer new frame K1
2302 06F9   21             IXC                 ;   MSB from new to
2303 06FA   16             TAM                 ;   old frame location
2304
2305 06FB   22             DECXN               ;Point to fractional K1
2306
2307 06FC   14             TMAIX               ;Transfer new fractional K1
2308 06FD   21             IXC                 ;   from new to old
2309 06FE   13             TAMIX               ;   frame location
2310           *-----K2-----
2311 06FF   14             TMAIX               ;Transfer new frame K2
2312 0700   21             IXC                 ;   MSB from new to
2313 0701   16             TAM                 ;   old frame location
2314
2315 0702   22             DECXN               ;Point to fractional K2
2316
2317 0703   14             TMAIX               ;Transfer new fractional K2
2318 0704   21             IXC                 ;   from new to old
2319 0705   13             TAMIX               ;   frame location
2320           *-----K3-----
2321 0706   14             TMAIX               ;Transfer new frame K3 parameter
2322 0707   13             TAMIX               ;   to current frame location
2323           *-----K4-----
2324 0708   14             TMAIX               ;Transfer new frame K4 parameter
2325 0709   13             TAMIX               ;   to current frame location
2326           *-----K5-----
2327 070A   14             TMAIX               ;Transfer new frame K5 parameter
2328 070B   13             TAMIX               ;   to current frame location
2329           *-----K6-----
2330 070C   14             TMAIX               ;Transfer new frame K6 parameter
2331 070D   13             TAMIX               ;   to current frame location
2332           *-----K7-----
2333 070E   14             TMAIX               ;Transfer new frame K7 parameter
2334 070F   13             TAMIX               ;   to current frame location
2335           *-----K8-----
2336 0710   14             TMAIX               ;Transfer new frame K8 parameter
2337 0711   13             TAMIX               ;   to current frame location
2338           *-----K9-----
2339 0712   14             TMAIX               ;Transfer new frame K9 parameter
2340 0713   13             TAMIX               ;   to current frame location
2341           *-----K10-----
2342 0714   14             TMAIX               ;Transfer new frame K10 parameter
2343 0715   13             TAMIX               ;   to current frame location
2344
2345           * * * * *
```

```
2346           * K11 and K12 are not used in LPC 10 synthesis.  the code
2347           * has been commented out.
2348           * * * * *
2349           *-----K11-----
2350           *       TMAIX             ;Transfer new frame K11 parameter
2351           *       TAMIX             ;to current frame location
2352           *-----K12-----
2353           *       TMAIX             ;Transfer new frame K12 parameter
2354           *       TAMIX             ;to current frame location
2355           *-----
2356
2357 0716   3D            RETN                ;return from call
2358
2359
2360           ************************************************
2361           *  INIT1                                       *
2362           *                                              *
2363           *  This subroutine turns on channel 1 and      *
2364           *  loads the correct address to the speech     *
2365           *  address register                            *
2366           ************************************************
2367           INIT1
2368 0717   20        CLX
2369 0718   22        DECXN                   ; point to mode register 2
2370 0719 65EF        ANDCM   ~CHANNEL        ; turn on channel 1
2371
2372 071B 6968        TMAD    ADR_LSB_1       ; fetch address LSB from memory
2373 071D   12        XBA                     ; store in B register
2374 071E 6967        TMAD    ADR_MSB_1       ; fetch address MSB from memory
2375 0720   1B        SALA4                   ; combine MSB and LSB
2376 0721   1B        SALA4                   ;   into complete
2377 0722   2C        ABAAC                   ;   address
2378
2379 0723   6C        LUAPS                   ; prepare to get data
2380
2381 0724 6969        TMAD    OFFSET_1        ; retreive offset
2382 0726 6301        AGEC    1               ; is there an offset?
2383 0728 472C        BR      INIT1_1         ;   yes, do a dummy get
2384 072A 4733        BR      INIT1_X         ;   no, return
2385
2386 072C   1A INIT1_1 TAB                    ; save offset in B register
2387 072D   2F        CLA                     ; clear offset memory location to avoid
2388 072E 6A69        TAMD    OFFSET_1        ;   counting offset twice (GETN restores
2389 0730   12        XBA                     ; retrieve offset
2390 0731 0635        CALL    C1GETN          ; do a dummy get
2391
2392 0733   3D INIT1_X RETN                   ;return from call
2393
2394
2395           ************************************************
2396           *  INIT2                                       *
2397           *                                              *
2398           *  This subroutine turns on channel 2 and      *
2399           *  loads the correct address to the speech     *
2400           *  address register                            *
2401           ************************************************
2402           INIT2
2403 0734   20        CLX
2404 0735   22        DECXN                   ; point to mode register 2
2405 0736 6410        ORCM    CHANNEL         ; turn on channel 2
2406
2407 0738 696B        TMAD    ADR_LSB_2       ; fetch address LSB from memory
```

```
2408 073A   12         XBA                    ; store in B register
2409 073B 696A         TMAD    ADR_MSB_2      ; fetch address MSB from memory
2410 073D   1B         SALA4                  ; combine MSB and LSB
2411 073E   1B         SALA4                  ;   into complete
2412 073F   2C         ABAAC                  ;   address
2413
2414 0740   6C         LUAPS                  ; prepare to get data
2415
2416 0741 696C         TMAD    OFFSET_2       ; retreive offset
2417 0743 6301         AGEC    1              ; is there an offset?
2418 0745 4749         BR      INIT2_1        ;   yes, do a dummy get
2419 0747 4750         BR      INIT2_X        ;   no, return
2420
2421 0749   1A INIT2_1 TAB                    ; save offset in B register
2422 074A   2F         CLA                    ; clear offset memory location to avoid
2423 074B 6A6C         TAMD    OFFSET_2       ;   counting offset twice (GETN restores
2424 074D   12         XBA                    ; retrieve offset
2425 074E 0692         CALL    C2GETN         ; do a dummy get
2426
2427 0750   3D INIT2_X RETN                   ; return from call
2428
2429
2430
2431           * * * * *
2432           * D6 SPEECH DECODING TABLES.
2433           * * * * *
2434           * Energy decoding table
2435           * * * * *
2436 0751 0001 TBLEN       BYTE    #00,#01,#02,#03,#04,#05,#07,#0B
2437 0759 111A             BYTE    #11,#1A,#29,#3F,#55,#70,#7F,#00
2438
2439           * * * * *
2440           * Pitch period decoding table
2441           * * * * *
2442 0761 0C00 TBLPH       BYTE    #0C,#00      ;0
2443 0763 1000             BYTE    #10,#00      ;1
2444 0765 1004             BYTE    #10,#04      ;2
2445 0767 1008             BYTE    #10,#08      ;3
2446 0769 1100             BYTE    #11,#00      ;4
2447 076B 1104             BYTE    #11,#04      ;5
2448 076D 1108             BYTE    #11,#08      ;6
2449 076F 110C             BYTE    #11,#0C      ;7
2450 0771 1204             BYTE    #12,#04      ;8
2451 0773 1208             BYTE    #12,#08      ;9
2452 0775 120C             BYTE    #12,#0C      ;10
2453 0777 1304             BYTE    #13,#04      ;1
2454 0779 1308             BYTE    #13,#08      ;2
2455 077B 1400             BYTE    #14,#00      ;3
2456 077D 1404             BYTE    #14,#04      ;4
2457 077F 140C             BYTE    #14,#0C      ;5
2458 0781 1500             BYTE    #15,#00      ;6
2459 0783 1508             BYTE    #15,#08      ;7
2460 0785 150C             BYTE    #15,#0C      ;8
2461 0787 1604             BYTE    #16,#04      ;9
2462 0789 160C             BYTE    #16,#0C      ;20
2463 078B 1700             BYTE    #17,#00      ;1
2464 078D 1708             BYTE    #17,#08      ;2
2465 078F 1800             BYTE    #18,#00      ;3
2466 0791 1804             BYTE    #18,#04      ;4
2467 0793 180C             BYTE    #18,#0C      ;5
2468 0795 1904             BYTE    #19,#04      ;6
2469 0797 190C             BYTE    #19,#0C      ;7
```

```
2470 0799 1A04            BYTE    #1A,#04      ;8
2471 079B 1A0C            BYTE    #1A,#0C      ;9
2472 079D 1B04            BYTE    #1B,#04      ;30
2473 079F 1B0C            BYTE    #1B,#0C      ;1
2474 07A1 1C04            BYTE    #1C,#04      ;2
2475 07A3 1C0C            BYTE    #1C,#0C      ;3
2476 07A5 1D04            BYTE    #1D,#04      ;4
2477 07A7 1D0C            BYTE    #1D,#0C      ;5
2478 07A9 1E04            BYTE    #1E,#04      ;6
2479 07AB 1F00            BYTE    #1F,#00      ;7
2480 07AD 1F08            BYTE    #1F,#08      ;8
2481 07AF 2000            BYTE    #20,#00      ;9
2482 07B1 200C            BYTE    #20,#0C      ;40
2483 07B3 2104            BYTE    #21,#04      ;1
2484 07B5 210C            BYTE    #21,#0C      ;2
2485 07B7 2208            BYTE    #22,#08      ;3
2486 07B9 2300            BYTE    #23,#00      ;4
2487 07BB 230C            BYTE    #23,#0C      ;5
2488 07BD 2408            BYTE    #24,#08      ;6
2489 07BF 2500            BYTE    #25,#00      ;7
2490 07C1 250C            BYTE    #25,#0C      ;8
2491 07C3 2608            BYTE    #26,#08      ;9
2492 07C5 2704            BYTE    #27,#04      ;50
2493 07C7 2800            BYTE    #28,#00      ;1
2494 07C9 280C            BYTE    #28,#0C      ;2
2495 07CB 2908            BYTE    #29,#08      ;3
2496 07CD 2A04            BYTE    #2A,#04      ;4
2497 07CF 2B00            BYTE    #2B,#00      ;5
2498 07D1 2B0C            BYTE    #2B,#0C      ;6
2499 07D3 2C08            BYTE    #2C,#08      ;7
2500 07D5 2D04            BYTE    #2D,#04      ;8
2501 07D7 2E04            BYTE    #2E,#04      ;9
2502 07D9 2F00            BYTE    #2F,#00      ;60
2503 07DB 3000            BYTE    #30,#00      ;1
2504 07DD 300C            BYTE    #30,#0C      ;2
2505 07DF 310C            BYTE    #31,#0C      ;3
2506 07E1 3208            BYTE    #32,#08      ;4
2507 07E3 3308            BYTE    #33,#08      ;5
2508 07E5 3408            BYTE    #34,#08      ;6
2509 07E7 3508            BYTE    #35,#08      ;7
2510 07E9 3608            BYTE    #36,#08      ;8
2511 07EB 3708            BYTE    #37,#08      ;9
2512 07ED 3808            BYTE    #38,#08      ;70
2513 07EF 3908            BYTE    #39,#08      ;1
2514 07F1 3A08            BYTE    #3A,#08      ;2
2515 07F3 3B0C            BYTE    #3B,#0C      ;3
2516 07F5 3C0C            BYTE    #3C,#0C      ;4
2517 07F7 3D0C            BYTE    #3D,#0C      ;5
2518 07F9 3F00            BYTE    #3F,#00      ;6
2519 07FB 4004            BYTE    #40,#04      ;7
2520 07FD 4104            BYTE    #41,#04      ;8
2521 07FF 4208            BYTE    #42,#08      ;9
2522 0801 430C            BYTE    #43,#0C      ;80
2523 0803 4500            BYTE    #45,#00      ;1
2524 0805 4604            BYTE    #46,#04      ;2
2525 0807 4708            BYTE    #47,#08      ;3
2526 0809 4900            BYTE    #49,#00      ;4
2527 080B 4A04            BYTE    #4A,#04      ;5
2528 080D 4B0C            BYTE    #4B,#0C      ;6
2529 080F 4D00            BYTE    #4D,#00      ;7
2530 0811 4E08            BYTE    #4E,#08      ;8
2531 0813 5000            BYTE    #50,#00      ;9
```

```
2532 0815 5104            BYTE    #51,#04      ;90
2533 0817 520C            BYTE    #52,#0C      ;1
2534 0819 5408            BYTE    #54,#08      ;2
2535 081B 5600            BYTE    #56,#00      ;3
2536 081D 5708            BYTE    #57,#08      ;4
2537 081F 5904            BYTE    #59,#04      ;5
2538 0821 5A0C            BYTE    #5A,#0C      ;6
2539 0823 5C08            BYTE    #5C,#08      ;7
2540 0825 5E04            BYTE    #5E,#04      ;8
2541 0827 6000            BYTE    #60,#00      ;9
2542 0829 610C            BYTE    #61,#0C      ;100
2543 082B 6308            BYTE    #63,#08      ;1
2544 082D 6504            BYTE    #65,#04      ;2
2545 082F 6704            BYTE    #67,#04      ;3
2546 0831 6900            BYTE    #69,#00      ;4
2547 0833 6B00            BYTE    #6B,#00      ;5
2548 0835 6D00            BYTE    #6D,#00      ;6
2549 0837 6F00            BYTE    #6F,#00      ;7
2550 0839 7100            BYTE    #71,#00      ;8
2551 083B 7304            BYTE    #73,#04      ;9
2552 083D 7504            BYTE    #75,#04      ;110
2553 083F 7708            BYTE    #77,#08      ;111
2554 0841 790C            BYTE    #79,#0C      ;112
2555 0843 7C00            BYTE    #7C,#00      ;113
2556 0845 7E04            BYTE    #7E,#04      ;114
2557 0847 8008            BYTE    #80,#08      ;115
2558 0849 820C            BYTE    #82,#0C      ;116
2559 084B 8504            BYTE    #85,#04      ;117
2560 084D 870C            BYTE    #87,#0C      ;118
2561 084F 8A04            BYTE    #8A,#04      ;119
2562 0851 8C0C            BYTE    #8C,#0C      ;120
2563 0853 8F08            BYTE    #8F,#08      ;121
2564 0855 9200            BYTE    #92,#00      ;122
2565 0857 940C            BYTE    #94,#0C      ;123
2566 0859 9708            BYTE    #97,#08      ;124
2567 085B 9A04            BYTE    #9A,#04      ;125
2568 085D 9D00            BYTE    #9D,#00      ;126
2569 085F A000            BYTE    #A0,#00      ;127
2570
2571           * * * * *
2572           * K1 parameter decoding table
2573           * * * * *
2574 0861 8100 TBLK1      BYTE    #81,#00
2575 0863 8204            BYTE    #82,#04
2576 0865 8304            BYTE    #83,#04
2577 0867 8408            BYTE    #84,#08
2578 0869 850C            BYTE    #85,#0C
2579 086B 8700            BYTE    #87,#00
2580 086D 8804            BYTE    #88,#04
2581 086F 890C            BYTE    #89,#0C
2582 0871 8B04            BYTE    #8B,#04
2583 0873 8C0C            BYTE    #8C,#0C
2584 0875 8E04            BYTE    #8E,#04
2585 0877 9000            BYTE    #90,#00
2586 0879 910C            BYTE    #91,#0C
2587 087B 9308            BYTE    #93,#08
2588 087D 9508            BYTE    #95,#08
2589 087F 9704            BYTE    #97,#04
2590 0881 9908            BYTE    #99,#08
2591 0883 9B08            BYTE    #9B,#08
2592 0885 9D08            BYTE    #9D,#08
2593 0887 9F0C            BYTE    #9F,#0C
```

```
2594 0889 A200            BYTE    #A2,#00
2595 088B A404            BYTE    #A4,#04
2596 088D A60C            BYTE    #A6,#0C
2597 088F A904            BYTE    #A9,#04
2598 0891 AB08            BYTE    #AB,#08
2599 0893 AE00            BYTE    #AE,#00
2600 0895 B00C            BYTE    #B0,#0C
2601 0897 B308            BYTE    #B3,#08
2602 0899 B604            BYTE    #B6,#04
2603 089B B900            BYTE    #B9,#00
2604 089D BC00            BYTE    #BC,#00
2605 089F BF04            BYTE    #BF,#04
2606 08A1 C204            BYTE    #C2,#04
2607 08A3 C508            BYTE    #C5,#08
2608 08A5 C80C            BYTE    #C8,#0C
2609 08A7 CC04            BYTE    #CC,#04
2610 08A9 CF0C            BYTE    #CF,#0C
2611 08AB D308            BYTE    #D3,#08
2612 08AD D708            BYTE    #D7,#08
2613 08AF DB04            BYTE    #DB,#04
2614 08B1 DF04            BYTE    #DF,#04
2615 08B3 E308            BYTE    #E3,#08
2616 08B5 E70C            BYTE    #E7,#0C
2617 08B7 EC00            BYTE    #EC,#00
2618 08B9 F004            BYTE    #F0,#04
2619 08BB F40C            BYTE    #F4,#0C
2620 08BD F90C            BYTE    #F9,#0C
2621 08BF FE0C            BYTE    #FE,#0C
2622 08C1 0404            BYTE    #04,#04
2623 08C3 090C            BYTE    #09,#0C
2624 08C5 0F04            BYTE    #0F,#04
2625 08C7 1508            BYTE    #15,#08
2626 08C9 1C08            BYTE    #1C,#08
2627 08CB 2308            BYTE    #23,#08
2628 08CD 2A0C            BYTE    #2A,#0C
2629 08CF 3208            BYTE    #32,#08
2630 08D1 3A08            BYTE    #3A,#08
2631 08D3 420C            BYTE    #42,#0C
2632 08D5 4B08            BYTE    #4B,#08
2633 08D7 5400            BYTE    #54,#00
2634 08D9 5C04            BYTE    #5C,#04
2635 08DB 6500            BYTE    #65,#00
2636 08DD 6E00            BYTE    #6E,#00
2637 08DF 7808            BYTE    #78,#08
2638
2639           * * * * *
2640           * K2 parameter decoding table
2641           * * * * *
2642 08E1 8A00 TBLK2       BYTE    #8A,#00
2643 08E3 9800            BYTE    #98,#00
2644 08E5 A30C            BYTE    #A3,#0C
2645 08E7 AD0C            BYTE    #AD,#0C
2646 08E9 B408            BYTE    #B4,#08
2647 08EB BA08            BYTE    #BA,#08
2648 08ED C000            BYTE    #C0,#00
2649 08EF C500            BYTE    #C5,#00
2650 08F1 C90C            BYTE    #C9,#0C
2651 08F3 CE04            BYTE    #CE,#04
2652 08F5 D20C            BYTE    #D2,#0C
2653 08F7 D60C            BYTE    #D6,#0C
2654 08F9 DA0C            BYTE    #DA,#0C
2655 08FB DE08            BYTE    #DE,#08
```

```
2656 08FD E200            BYTE    #E2,#00
2657 08FF E50C            BYTE    #E5,#0C
2658 0901 E904            BYTE    #E9,#04
2659 0903 EC0C            BYTE    #EC,#0C
2660 0905 F000            BYTE    #F0,#00
2661 0907 F304            BYTE    #F3,#04
2662 0909 F608            BYTE    #F6,#08
2663 090B F90C            BYTE    #F9,#0C
2664 090D FD00            BYTE    #FD,#00
2665 090F 0000            BYTE    #00,#00
2666 0911 0304            BYTE    #03,#04
2667 0913 0604            BYTE    #06,#04
2668 0915 0904            BYTE    #09,#04
2669 0917 0C04            BYTE    #0C,#04
2670 0919 0F04            BYTE    #0F,#04
2671 091B 1208            BYTE    #12,#08
2672 091D 1508            BYTE    #15,#08
2673 091F 1808            BYTE    #18,#08
2674 0921 1B08            BYTE    #1B,#08
2675 0923 1E08            BYTE    #1E,#08
2676 0925 2108            BYTE    #21,#08
2677 0927 240C            BYTE    #24,#0C
2678 0929 270C            BYTE    #27,#0C
2679 092B 2A0C            BYTE    #2A,#0C
2680 092D 2D0C            BYTE    #2D,#0C
2681 092F 300C            BYTE    #30,#0C
2682 0931 3400            BYTE    #34,#00
2683 0933 3700            BYTE    #37,#00
2684 0935 3A04            BYTE    #3A,#04
2685 0937 3D00            BYTE    #3D,#00
2686 0939 4000            BYTE    #40,#00
2687 093B 4300            BYTE    #43,#00
2688 093D 4600            BYTE    #46,#00
2689 093F 4900            BYTE    #49,#00
2690 0941 4C00            BYTE    #4C,#00
2691 0943 4F04            BYTE    #4F,#04
2692 0945 5204            BYTE    #52,#04
2693 0947 5504            BYTE    #55,#04
2694 0949 5804            BYTE    #58,#04
2695 094B 5B04            BYTE    #5B,#04
2696 094D 5E00            BYTE    #5E,#00
2697 094F 6100            BYTE    #61,#00
2698 0951 630C            BYTE    #63,#0C
2699 0953 6608            BYTE    #66,#08
2700 0955 6904            BYTE    #69,#04
2701 0957 6C00            BYTE    #6C,#00
2702 0959 6F00            BYTE    #6F,#00
2703 095B 7200            BYTE    #72,#00
2704 095D 7604            BYTE    #76,#04
2705 095F 7C00            BYTE    #7C,#00
2706
2707          * * * * *
2708          * K3 parameter decoding table
2709          * * * * *
2710 0961 8B9A TBLK3      BYTE    #8B,#9A,#A2,#A9,#AF,#B5,#BB,#C0
2711 0969 C5CA            BYTE    #C5,#CA,#CF,#D4,#D9,#DE,#E2,#E7
2712 0971 ECF1            BYTE    #EC,#F1,#F6,#FB,#01,#07,#0D,#14
2713 0979 1A22            BYTE    #1A,#22,#29,#32,#3B,#45,#53,#6D
2714
2715          * * * * *
2716          * K4 parameter decoding table
2717          * * * * *
```

```
2718 0981 94B0 TBLK4       BYTE    #94,#B0,#C2,#CB,#D3,#D9,#DF,#E5
2719 0989 EAEF             BYTE    #EA,#EF,#F4,#F9,#FE,#03,#07,#0C
2720 0991 1115             BYTE    #11,#15,#1A,#1F,#24,#29,#2E,#33
2721 0999 383E             BYTE    #38,#3E,#44,#4B,#53,#5A,#64,#74
2722
2723           * * * * *
2724           * K5 parameter decoding table
2725           * * * * *
2726 09A1 A3C5 TBLK5       BYTE    #A3,#C5,#D4,#E0,#EA,#F3,#FC,#04
2727 09A9 0C15             BYTE    #0C,#15,#1E,#27,#31,#3D,#4C,#66
2728
2729           * * * * *
2730           * K6 parameter decoding table
2731           * * * * *
2732 09B1 AAD7 TBLK6       BYTE    #AA,#D7,#E7,#F2,#FC,#05,#0D,#14
2733 09B9 1C24             BYTE    #1C,#24,#2D,#36,#40,#4A,#55,#6A
2734
2735           * * * * *
2736           * K7 parameter decoding table
2737           * * * * *
2738 09C1 A3C8 TBLK7       BYTE    #A3,#C8,#D7,#E3,#ED,#F5,#FD,#05
2739 09C9 0D14             BYTE    #0D,#14,#1D,#26,#31,#3C,#4B,#67
2740
2741           * * * * *
2742           * K8 parameter decoding table
2743           * * * * *
2744 09D1 C5E4 TBLK8       BYTE    #C5,#E4,#F6,#05,#14,#27,#3E,#58
2745
2746           * * * * *
2747           * K9 parameter decoding table
2748           * * * * *
2749 09D9 B9DC TBLK9       BYTE    #B9,#DC,#EC,#F9,#04,#10,#1F,#45
2750
2751           * * * * *
2752           * K10 parameter decoding table
2753           * * * * *
2754 09E1 C3E6 TBLK10      BYTE    #C3,#E6,#F3,#FD,#06,#11,#1E,#43
2755
2756
2757           ************************************************************
2758           *                                                          *
2759           *    This is the lookup table giving the starting          *
2760           *    address of each concatenation list.                   *
2761           *                                                          *
2762           ************************************************************
2763 09E9 09ED SENTENCE    DATA    PHRASEa
2764 09EB 0A03             DATA    PHRASEb
2765
2766
2767           ************************************************************
2768           *                                                          *
2769           *    This is the concatenation table giving the lists      *
2770           *    of word numbers that define each phrase.              *
2771           *                                                          *
2772           ************************************************************
2773 09ED 0107 PHRASEa    BYTE    1,7,2,7,3,7,4,7,5,7,7,5,7,4,8,3,9,2,10,1,11,#FF
2774 0A03 0701 PHRASEb    BYTE    7,1,7,2,7,3,7,4,7,5,7,7,5,8,4,9,3,10,2,11,1,10,#FF
2775
2776
2777           ************************************************************
2778           *                                                          *
2779           *    This is the lookup table for the speech stored at     *
```

```
2780           *    voc.                                                  *
2781           *                                                          *
2782           ************************************************************
2783 0A1A 0E32 SPEECH      DATA    SILENCE      ;Word 0       silence
2784 0A1C 0A32             DATA    #0000+VOC    ;Word 1       ″One″  MALE
2785 0A1E 0AB6             DATA    #0084+VOC    ;Word 2       ″Two″
2786 0A20 0B28             DATA    #00F6+VOC    ;Word 3       ″Three″
2787 0A22 0BAC             DATA    #017A+VOC    ;Word 4       ″Four″
2788 0A24 0C12             DATA    #01E0+VOC    ;Word 5       ″Five″
2789 0A26 0CBE             DATA    #028C+VOC    ;Word 6       ″Six″
2790 0A28 0E50             DATA    PAUSE        ;Word 7       pause for echo effect
2791 0A2A 0E5E             DATA    PAUSE2       ;Word 8       pause  ″
2792 0A2C 0E6C             DATA    PAUSE3       ;Word 9       pause  ″
2793 0A2E 0E7A             DATA    PAUSE4       ;Word 10      pause  ″
2794 0A30 0E88             DATA    PAUSE5       ;Word 11      pause  ″
2795
2796           ************************************************************
2797           *                                                          *
2798           *    This is the DTS speech coded with the D6 coding       *
2799           *    table                                                 *
2800           *                                                          *
2801           ************************************************************
2802           VOC
2803 0A32 6889             BYTE    #68,#89,#84,#FB,#1A,#53,#64,#B2
2804 0A3A 8487             BYTE    #84,#87,#33,#C9,#35,#28,#9B,#A1
2805 0A42 D1BA             BYTE    #D1,#BA,#22,#3A,#94,#8D,#08,#BD
2806 0A4A BE40             BYTE    #BE,#40,#1C,#6D,#BA,#BC,#14,#7E
2807 0A52 33CE             BYTE    #33,#CE,#4E,#75,#8D,#EE,#2F,#03
2808 0A5A BB96             BYTE    #BB,#96,#4A,#46,#D7,#CF,#4A,#DD
2809 0A62 4A23             BYTE    #4A,#23,#54,#CE,#26,#B7,#74,#A5
2810 0A6A 9B49             BYTE    #9B,#49,#7B,#62,#44,#B7,#32,#2D
2811 0A72 95D9             BYTE    #95,#D9,#C8,#B4,#5B,#9A,#35,#5A
2812 0A7A 8DC2             BYTE    #8D,#C2,#DC,#2C,#CC,#5A,#CC,#0A
2813 0A82 2B6E             BYTE    #2B,#6E,#EE,#66,#19,#69,#98,#27
2814 0A8A 7533             BYTE    #75,#33,#CB,#80,#36,#AC,#94,#E6
2815 0A92 A985             BYTE    #A9,#85,#CE,#4B,#1B,#EC,#CD,#D4
2816 0A9A 2C50             BYTE    #2C,#50,#71,#52,#F5,#76,#AA,#1B
2817 0AA2 9B38             BYTE    #9B,#38,#98,#58,#33,#56,#B6,#35
2818 0AAA D258             BYTE    #D2,#58,#A3,#99,#C8,#7B,#AE,#D5
2819 0AB2 A85E             BYTE    #A8,#5E,#FB,#01,#04,#B0,#78,#BA
2820 0ABA 2BC0             BYTE    #2B,#C0,#5D,#1B,#6D,#00,#F7,#65
2821 0AC2 BA01             BYTE    #BA,#01,#64,#BA,#13,#29,#B7,#06
2822 0ACA 3681             BYTE    #36,#81,#C9,#FE,#92,#DB,#5C,#15
2823 0AD2 20B8             BYTE    #20,#B8,#7F,#29,#AF,#8A,#CA,#10
2824 0ADA DC3F             BYTE    #DC,#3F,#35,#12,#56,#47,#2A,#FA
2825 0AE2 9FFA             BYTE    #9F,#FA,#26,#61,#97,#0C,#ED,#77
2826 0AEA 439A             BYTE    #43,#9A,#6E,#97,#9A,#F7,#8A,#01
2827 0AF2 2ECE             BYTE    #2E,#CE,#8D,#29,#7B,#48,#17,#B1
2828 0AFA CF86             BYTE    #CF,#86,#B4,#4E,#64,#04,#47,#77
2829 0B02 A14B             BYTE    #A1,#4B,#26,#32,#83,#9B,#13,#31
2830 0B0A AD23             BYTE    #AD,#23,#59,#E3,#DA,#5E,#90,#B2
2831 0B12 85AC             BYTE    #85,#AC,#68,#65,#0D,#70,#E9,#4D
2832 0B1A 3644             BYTE    #36,#44,#38,#13,#87,#74,#12,#BB
2833 0B22 8D52             BYTE    #8D,#52,#59,#90,#E4,#3D,#08,#60
2834 0B2A CA86             BYTE    #CA,#86,#13,#40,#66,#1A,#46,#00
2835 0B32 B9EC             BYTE    #B9,#EC,#8B,#00,#14,#59,#B7,#0A
2836 0B3A 905A             BYTE    #90,#5A,#35,#9A,#EC,#1E,#D9,#86
2837 0B42 A4EA             BYTE    #A4,#EA,#5C,#41,#69,#85,#B2,#A6
2838 0B4A EE21             BYTE    #EE,#21,#AF,#CC,#24,#46,#63,#F7
2839 0B52 9453             BYTE    #94,#53,#26,#E1,#65,#B1,#7B,#C9
2840 0B5A 3BA5             BYTE    #3B,#A5,#77,#B8,#92,#3E,#E5,#9B
2841 0B62 B47B             BYTE    #B4,#7B,#18,#EE,#9F,#0A,#5B,#52
```

```
2842 0B6A 02B4          BYTE    #02,#B4,#EE,#4F,#8D,#23,#CF,#06
2843 0B72 2AB7          BYTE    #2A,#B7,#A7,#FE,#96,#04,#0A,#DD
2844 0B7A DFD2          BYTE    #DF,#D2,#70,#B6,#24,#C6,#9D,#25
2845 0B82 613C          BYTE    #61,#3C,#F0,#1C,#F3,#ED,#A4,#30
2846 0B8A 5974          BYTE    #59,#74,#8E,#70,#E7,#96,#9B,#4C
2847 0B92 0A47          BYTE    #0A,#47,#74,#3B,#D1,#CC,#07,#95
2848 0B9A 21BE          BYTE    #21,#BE,#19,#65,#A6,#B3,#27,#20
2849 0BA2 CE4C          BYTE    #CE,#4C,#62,#93,#58,#41,#B4,#77
2850 0BAA 0A3E          BYTE    #0A,#3E,#80,#00,#A6,#6A,#03,#01
2851 0BB2 54A6          BYTE    #54,#A6,#4F,#0C,#10,#C6,#D1,#0B
2852 0BBA 8097          BYTE    #80,#97,#D4,#E0,#12,#2A,#D7,#37
2853 0BC2 8758          BYTE    #87,#58,#09,#E9,#18,#B7,#3F,#0D
2854 0BCA BD87          BYTE    #BD,#87,#74,#8A,#99,#9F,#86,#DE
2855 0BD2 43D9          BYTE    #43,#D9,#26,#EA,#37,#C5,#EC,#A1
2856 0BDA A9B0          BYTE    #A9,#B0,#F3,#91,#71,#FE,#30,#60
2857 0BE2 83B3          BYTE    #83,#B3,#B1,#C4,#7F,#1A,#B3,#ED
2858 0BEA 8ED4          BYTE    #8E,#D4,#A2,#3F,#CC,#84,#AD,#4A
2859 0BF2 1BE8          BYTE    #1B,#E8,#1F,#D6,#EA,#38,#A4,#1C
2860 0BFA E60F          BYTE    #E6,#0F,#5B,#63,#49,#D4,#0F,#F3
2861 0C02 B983          BYTE    #B9,#83,#B1,#7B,#E2,#87,#7B,#DD
2862 0C0A D5BA          BYTE    #D5,#BA,#A8,#E8,#C5,#5D,#0F,#00
2863 0C12 0890          BYTE    #08,#90,#FB,#51,#23,#80,#AB,#19
2864 0C1A 4A00          BYTE    #4A,#00,#B9,#97,#0D,#01,#34,#59
2865 0C22 490C          BYTE    #49,#0C,#D0,#A5,#29,#11,#80,#E5
2866 0C2A 8658          BYTE    #86,#58,#EA,#BE,#32,#36,#27,#F5
2867 0C32 69B5          BYTE    #69,#B5,#4C,#18,#CB,#9B,#DA,#B5
2868 0C3A 7AAA          BYTE    #7A,#AA,#EC,#61,#45,#6B,#4B,#33
2869 0C42 F06F          BYTE    #F0,#6F,#D1,#94,#25,#A5,#ED,#15
2870 0C4A 3768          BYTE    #37,#68,#EA,#9C,#D4,#75,#BA,#ED
2871 0C52 346D          BYTE    #34,#6D,#4E,#19,#7B,#CD,#76,#9A
2872 0C5A 7ABB          BYTE    #7A,#BB,#CC,#A2,#F2,#18,#4D,#B9
2873 0C62 5996          BYTE    #59,#96,#59,#71,#B4,#A4,#3C,#2A
2874 0C6A CBBC          BYTE    #CB,#BC,#5A,#5C,#52,#67,#A6,#4D
2875 0C72 3636          BYTE    #36,#36,#AA,#61,#17,#D3,#2E,#6F
2876 0C7A 2293          BYTE    #22,#93,#F4,#05,#61,#1F,#56,#52
2877 0C82 69E7          BYTE    #69,#E7,#41,#B3,#0F,#32,#E1,#AC
2878 0C8A E2B0          BYTE    #E2,#B0,#D9,#EB,#95,#34,#5C,#7E
2879 0C92 52EC          BYTE    #52,#EC,#E5,#44,#1B,#4A,#79,#C1
2880 0C9A F63A          BYTE    #F6,#3A,#6D,#1C,#9A,#76,#66,#BB
2881 0CA2 5132          BYTE    #51,#32,#16,#89,#94,#99,#DD,#96
2882 0CAA 8F69          BYTE    #8F,#69,#C9,#6A,#D5,#6E,#F2,#52
2883 0CB2 2162          BYTE    #21,#62,#6A,#62,#37,#24,#2D,#22
2884 0CBA 1197          BYTE    #11,#97,#07,#00,#04,#F0,#2A,#08
2885 0CC2 13C0          BYTE    #13,#C0,#BF,#F9,#44,#00,#FF,#EE
2886 0CCA 9500          BYTE    #95,#00,#7C,#A5,#D3,#02,#F0,#B5
2887 0CD2 DA94          BYTE    #DA,#94,#62,#C6,#17,#8D,#D9,#B7
2888 0CDA 4BBE          BYTE    #4B,#BE,#97,#8B,#25,#CB,#D7,#A5
2889 0CE2 5AAA          BYTE    #5A,#AA,#4D,#72,#F7,#DB,#D4,#2F
2890 0CEA BD4C          BYTE    #BD,#4C,#75,#EA,#6B,#5A,#84,#15
2891 0CF2 D1DD          BYTE    #D1,#DD,#BD,#11,#00,#80,#01,#1C
2892 0CFA 6F6B          BYTE    #6F,#6B,#01,#78,#AC,#BE,#05,#E0
2893 0D02 5F75          BYTE    #5F,#75,#62,#80,#7F,#D0,#9D,#01
2894 0D0A BE8F          BYTE    #BE,#8F,#7B,#02,#78,#3B,#5D,#1E
2895 0D12 08F0          BYTE    #08,#F0,#15,#3E,#13,#C0,#57,#F3
2896 0D1A 4C00          BYTE    #4C,#00,#7F,#CF,#38,#01,#FC,#81
2897 0D22 320C          BYTE    #32,#0C,#F0,#5F,#C2,#85,#62,#C5
2898 0D2A 0D85          BYTE    #0D,#85,#59,#9B,#5A,#25,#D5,#87
2899 0D32 A4AA          BYTE    #A4,#AA,#67,#A5,#5A,#04,#5B,#62
2900 0D3A D7DC          BYTE    #D7,#DC,#52,#4B,#9A,#C9,#A9,#F2
2901 0D42 49E9          BYTE    #49,#E9,#46,#6D,#37,#94,#FE,#C4
2902 0D4A 8C75          BYTE    #8C,#75,#B3,#58,#52,#CB,#64,#A6
2903 0D52 2C53          BYTE    #2C,#53,#23,#47,#A6,#35,#6B,#DE
```

```
2904 0D5A C89A             BYTE    #C8,#9A,#23,#6B,#A5,#55,#E0,#36
2905 0D62 C91A             BYTE    #C9,#1A,#B7,#D2,#3E,#0E,#26,#67
2906 0D6A 8D4B             BYTE    #8D,#4B,#66,#AF,#26,#99,#BB,#D5
2907 0D72 40B5             BYTE    #40,#B5,#97,#2D,#36,#95,#3A,#E6
2908 0D7A 0300             BYTE    #03,#00,#A6,#2A,#5A,#BE,#D6,#45
2909 0D82 E850             BYTE    #E8,#50,#C9,#5C,#A9,#EC,#7A,#76
2910 0D8A A98C             BYTE    #A9,#8C,#91,#65,#B8,#FD,#B6,#54
2911 0D92 D63C             BYTE    #D6,#3C,#52,#AC,#D9,#5A,#8A,#9B
2912 0D9A E911             BYTE    #E9,#11,#6D,#3F,#2D,#E5,#29,#96
2913 0DA2 50AE             BYTE    #50,#AE,#E7,#A6,#FE,#92,#2B,#28
2914 0DAA 75AB             BYTE    #75,#AB,#DD,#A6,#8F,#29,#D4,#D9
2915 0DB2 5900             BYTE    #59,#00,#0C,#B0,#08,#D4,#0A,#C0
2916 0DBA 13E6             BYTE    #13,#E6,#AE,#00,#6B,#7D,#9B,#02
2917 0DC2 8CE5             BYTE    #8C,#E5,#32,#0F,#A4,#25,#53,#73
2918 0DCA 5750             BYTE    #57,#50,#53,#D1,#93,#C5,#3C,#5B
2919 0DD2 6599             BYTE    #65,#99,#18,#CA,#7C,#99,#65,#BC
2920 0DDA CE8D             BYTE    #CE,#8D,#65,#4A,#0F,#4D,#9D,#53
2921 0DE2 C69E             BYTE    #C6,#9E,#D3,#1C,#65,#4E,#2C,#23
2922 0DEA 3F3B             BYTE    #3F,#3B,#52,#D2,#4F,#95,#9E,#9F
2923 0DF2 1D29             BYTE    #1D,#29,#E9,#A7,#4A,#37,#B7,#4F
2924 0DFA A5B2             BYTE    #A5,#B2,#35,#A5,#9B,#DB,#A7,#52
2925 0E02 D99A             BYTE    #D9,#9A,#D2,#C9,#93,#93,#A8,#74
2926 0E0A 4DE9             BYTE    #4D,#E9,#96,#D9,#F2,#54,#B2,#BA
2927 0E12 8CF2             BYTE    #8C,#F2,#BA,#09,#69,#9D,#59,#46
2928 0E1A 651E             BYTE    #65,#1E,#99,#96,#56,#4C,#A3,#38
2929 0E22 54CC             BYTE    #54,#CC,#5B,#0B,#B9,#91,#AD,#1B
2930 0E2A 9EC5             BYTE    #9E,#C5,#45,#D9,#18,#73,#C2,#0C
2931
2932
2933 0E32 0000 SILENCE     BYTE    #00,#00,#00,#00,#00,#00,#00,#00
2934 0E3A 0000             BYTE    #00,#00,#00,#00,#00,#00,#00,#00
2935 0E42 0000             BYTE    #00,#00,#00,#00,#00,#00,#00,#00
2936 0E4A 0000             BYTE    #00,#00,#00,#00,#00,#FF
2937
2938 0E50 00FF PAUSE       BYTE    #00,#FF,#00,#00,#FF,#00,#00,#00
2939 0E58 0000             BYTE    #00,#00,#00,#00,#00,#FF
2940
2941 0E5E 0000 PAUSE2      BYTE    #00,#00,#FF,#00,#FF,#00,#00,#00
2942 0E66 0000             BYTE    #00,#00,#00,#00,#00,#FF
2943
2944 0E6C 0000 PAUSE3      BYTE    #00,#00,#00,#FF,#FF,#00,#00,#00
2945 0E74 0000             BYTE    #00,#00,#00,#00,#00,#FF
2946
2947 0E7A 0000 PAUSE4      BYTE    #00,#00,#00,#00,#FF,#00,#00,#00
2948 0E82 0000             BYTE    #00,#00,#00,#00,#00,#FF
2949
2950 0E88 0000 PAUSE5      BYTE    #00,#00,#00,#00,#00,#FF,#00,#00
2951 0E90 0000             BYTE    #00,#00,#00,#00,#00,#FF
2952
2953
2954           *EXCITATION FUNCTION
2955
2956 8000                  AORG    #8000
2957
2958      0144    BYTE  #01,#44,#01,#5E,#01,#74,#01,#84,#01,#8E,#01,#92,#01,#94,#01,#8C
2959 8010 0184    BYTE  #01,#84,#01,#78,#01,#6A,#01,#5A,#01,#4A,#01,#3C,#01,#34,#01,#2A
2960 8020 012A    BYTE  #01,#2A,#01,#30,#01,#3E,#01,#50,#01,#70,#01,#94,#01,#C6,#01,#FC
2961 8030 023E    BYTE  #02,#3E,#02,#82,#02,#D2,#03,#22,#03,#7A,#03,#D0,#04,#2C,#04,#80
2962 8040 04D8    BYTE  #04,#D8,#05,#24,#05,#72,#05,#B2,#05,#F0,#06,#1E,#06,#4A,#06,#64
2963 8050 067E    BYTE  #06,#7E,#06,#86,#06,#8E,#06,#8A,#06,#8A,#06,#7E,#06,#7A,#06,#74
2964 8060 067A    BYTE  #06,#7A,#06,#82,#06,#9C,#06,#BE,#06,#F6,#07,#40,#07,#A4,#08,#1A
2965 8070 08AE    BYTE  #08,#AE,#09,#5A,#0A,#22,#0B,#04,#0C,#00,#0D,#14,#0E,#3E,#0F,#7A
```

```
2966 8080 10C8    BYTE  #10,#C8,#12,#22,#13,#82,#14,#E8,#16,#4C,#17,#AA,#18,#FE,#1A,#40
2967 8090 1B6E    BYTE  #1B,#6E,#1C,#80,#1D,#76,#1E,#48,#1E,#F4,#1F,#78,#1F,#D2,#1F,#FE
2968 80A0 1FFE    BYTE  #1F,#FE,#1F,#D2,#1F,#78,#1E,#F4,#1E,#48,#1D,#76,#1C,#80,#1B,#6E
2969 80B0 1A40    BYTE  #1A,#40,#18,#FE,#17,#AA,#16,#4C,#14,#E8,#13,#82,#12,#22,#10,#C8
2970 80C0 0F7A    BYTE  #0F,#7A,#0E,#3E,#0D,#14,#0C,#00,#0B,#04,#0A,#22,#09,#5A,#08,#AE
2971 80D0 081A    BYTE  #08,#1A,#07,#A4,#07,#40,#06,#F6,#06,#BE,#06,#9C,#06,#82,#06,#7A
2972 80E0 0674    BYTE  #06,#74,#06,#7A,#06,#7E,#06,#8A,#06,#8A,#06,#8E,#06,#86,#06,#7E
2973 80F0 0664    BYTE  #06,#64,#06,#4A,#06,#1E,#05,#F0,#05,#B2,#05,#72,#05,#24,#04,#D8
2974 8100 0480    BYTE  #04,#80,#04,#2C,#03,#D0,#03,#7A,#03,#22,#02,#D2,#02,#82,#02,#3E
2975 8110 01FC    BYTE  #01,#FC,#01,#C6,#01,#94,#01,#70,#01,#50,#01,#3E,#01,#30,#01,#2A
2976 8120 012A    BYTE  #01,#2A,#01,#34,#01,#3C,#01,#4A,#01,#5A,#01,#6A,#01,#78,#01,#84
2977 8130 018C    BYTE  #01,#8C,#01,#94,#01,#92,#01,#8E,#01,#84,#01,#74,#01,#5E,#01,#44
2978 8140 0B00    BYTE  #0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00
2979 8150 0B00    BYTE  #0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00
2980 8160 F500    BYTE  #F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00
2981 8170 F500    BYTE  #F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00
2982 8180 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2983 8190 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2984 81A0 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2985 81B0 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2986 81C0 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2987 81D0 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2988 81E0 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2989 81F0 FFFF    BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
2990 8200   FF    BYTE  #FF
Assembly Messages
Warnings(0) Errors(0) Fatal(0)
```

Appendix D

Sample Synthesis Program

This appendix contains the code for a sample synthesis program that runs on the MSP50C3x family of speech synthesizers. It has the MSP50C3x device speak the numbers from one to five on one channel only.

Topic **Page**

D.1 Sample Synthesis Program

```
d6single.asm   TSP50CXX Assembler Version 2.1d         Thu Sep 26 10:42:26 1996
                                                                    PAGE 0001
0001                       OPTION  BUNLIST,DUNLIST,PAGEOF
0002                       WIDE
0003
0004                       TABSIZE 8
0005                *-------------------------------------------------*
0006                *   TSP50C1x LPC SYNTHESIS PROGRAM                *
0007                *                                                 *
0008                *   This is a sample speech synthesis program     *
0009                *   which runs on the TSP50C3x family of speech   *
0010                *   synthesis microprocessors.  It simply         *
0011                *   counts from one to five on one channel.       *
0012                *                                                 *
0013                *   This program uses the D6 Coding table format. *
0014                *                                                 *
0015                *-------------------------------------------------*
0016                *   COPYRIGHT 1995, TI – SPEECH PRODUCTS        *
0017                *-------------------------------------------------*
0018                *   RAM MAP                                       *
0019                *+------------------------------------------------*
0020                *    +----+----+----+----+----+----+----+----+
0021                *    | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
0022                *    +----+----+----+----+----+----+----+----+
0023                *    |EN- | EN | K12| K11| K10| K9 | K8 | K7 |
0024                *    |TMP |    |    |  |    |    | |  |    |
0025                *    +----+----+----+----+----+----+----+----+
0026                *    | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
0027                *    +----+----+----+----+----+----+----+----+
0028                *    | K6 | K5 | K4 | K3 | K2 | K1 | C1 | C2 |
0029                *    |    |    |    |    |    |    |    |    |
0030                *    +----+----+----+----+----+----+----+----+
0031                *    | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
0032                *    +----+----+----+----+----+----+----+----+
0033                *    |    |    |    |    |    |    |    |    |
0034                *    |    |    |    |    |    |    |    |    |
0035                *    +----+----+----+----+----+----+----+----+
0036                *    | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
0037                *    +----+----+----+----+----+----+----+----+
0038                *    |    |    |    |    |    |    |    |    |
0039                *    |    |    |    |    |    |    |    |    |
0040                *    +----+----+----+----+----+----+----+----+
0041                *    | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
0042                *    +----+----+----+----+----+----+----+----+
0043                *    | EN | EN | PH      | PH      | K1      |
0044                *    | V2 | V1 | V2      | V1      | V2      |
0045                *    +----+----+----+----+----+----+----+----+
0046                *    | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
0047                *    +----+----+----+----+----+----+----+----+
0048                *    | K1      | K2      | K2      | K3 | K3 |
0049                *    | V1      | V2      | V1      | V2 | V1 |
0050                *    +----+----+----+----+----+----+----+----+
0051                *    | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
0052                *    +----+----+----+----+----+----+----+----+
0053                *    | K4 | K4 | K5 | K5 | K6 | K6 | K7 | K7 |
0054                *    | V2 | V1 | V2 | V1 | V2 | V1 | V2 | V1 |
0055                *    +----+----+----+----+----+----+----+----+
0056                *    | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
0057                *    +----+----+----+----+----+----+----+----+
```

```
0058          *    | K8 | K8 | K9 | K9 | K10| K10|TIMR|SCAL|
0059          *    | V2 | V1 | V2 | V1 | V2 | V1 |    |    |
0060          *    +----+----+----+----+----+----+----+----+
0061          *    | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
0062          *    +----+----+----+----+----+----+----+----+
0063          *    |FLAG|FLAG|MODE|    |    |    |    |    |
0064          *    |    |  1 | BUF|    |    |    |    |    |
0065          *    +----+----+----+----+----+----+----+----+
0066          *    | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
0067          *    +----+----+----+----+----+----+----+----+
0068          *    |    |    |    |    |    |    |    |    |
0069          *    |    |    |    |    |    |    |    |    |
0070          *    +----+----+----+----+----+----+----+----+
0071          *    | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |
0072          *    +----+----+----+----+----+----+----+----+
0073          *    |    |    |    |    |    |    |    |    |
0074          *    |    |    |    |    |    |    |    |    |
0075          *    +----+----+----+----+----+----+----+----+
0076          *    | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
0077          *    +----+----+----+----+----+----+----+----+
0078          *    |    |    |    |    |    |    |    |    |
0079          *    |    |    |    |    |    |    |    |    |
0080          *    +----+----+----+----+----+----+----+----+
0081          *    | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
0082          *    +----+----+----+----+----+----+----+----+
0083          *    |    |    |    |    |    |    |    |    |
0084          *    |    |    |    |    |    |    |    |    |
0085          *    +----+----+----+----+----+----+----+----+
0086          *    | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
0087          *    +----+----+----+----+----+----+----+----+
0088          *    |    |    |    |    |    |    |    |    |
0089          *    |    |    |    |    |    |    |    |    |
0090          *    +----+----+----+----+----+----+----+----+
0091          *    | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
0092          *    +----+----+----+----+----+----+----+----+
0093          *    |    |    |    |    |    |    |    |    |
0094          *    |    |    |    |    |    |    |    |    |
0095          *    +----+----+----+----+----+----+----+----+
0096          *    | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |
0097          *    +----+----+----+----+----+----+----+----+
0098          *    |    |    |    |    |    |    |    |    |
0099          *    |    |    |    |    |    |    |    |    |
0100          *    +----+----+----+----+----+----+----+----+
0101          *
0102          *-------------------------------------------------*
0103          *      ADDRESS LABELS FOR SYNTHESIS ROUTINE       *
0104          *-------------------------------------------------*
0105          ****************************************
0106          *      SYNTHESIZER RAM LOCATIONS
0107          ****************************************
0108          * NOTE – NEVER CHANGE LOCATIONS #01 TO #0F
0109          *
0110     0000 EN_TEMP     EQU     #00          ;Temp storage for EN value
0111     0001 EN          EQU     #01          ;Energy working value
0112     0002 K12         EQU     #02          ;K12 Working Value
0113     0003 K11         EQU     #03          ;K11 Working Value
0114     0004 K10         EQU     #04          ;K10 Working Value
0115     0005 K9          EQU     #05          ;K9 Working Value
0116     0006 K8          EQU     #06          ;K8 Working Value
0117     0007 K7          EQU     #07          ;K7 Working Value
0118     0008 K6          EQU     #08          ;K6 Working Value
0119     0009 K5          EQU     #09          ;K5 Working Value
```

```
0120      000A K4          EQU     #0A          ;K4 Working Value
0121      000B K3          EQU     #0B          ;K3 Working Value
0122      000C K2          EQU     #0C          ;K2 Working Value
0123      000D K1          EQU     #0D          ;K1 Working Value
0124      000E C1          EQU     #0E          ;C1 Parameter
0125      000F C2          EQU     #0F          ;C2 Parameter
0126      0020 ENV2        EQU     #20          ;ENERGY New Value MSB
0127      0021 ENV1        EQU     #21          ;ENERGY Current Value MSB
0128      0022 PHV2        EQU     #22          ;PITCH New Value MSB
0129      0024 PHV1        EQU     #24          ;PITCH Current Value MSB
0130      0026 K1V2        EQU     #26          ;K1 New Value MSB
0131      0028 K1V1        EQU     #28          ;K1 Current Value MSB
0132      002A K2V2        EQU     #2A          ;K2 New Value MSB
0133      002C K2V1        EQU     #2C          ;K2 Current Value MSB
0134      002E K3V2        EQU     #2E          ;K3 New Value MSB
0135      002F K3V1        EQU     #2F          ;K3 Current Value MSB
0136      0030 K4V2        EQU     #30          ;K4 New Value MSB
0137      0031 K4V1        EQU     #31          ;K4 Current Value MSB
0138      0032 K5V2        EQU     #32          ;K5 New Value
0139      0033 K5V1        EQU     #33          ;K5 Current Value
0140      0034 K6V2        EQU     #34          ;K6 New Value
0141      0035 K6V1        EQU     #35          ;K6 Current Value
0142      0036 K7V2        EQU     #36          ;K7 New Value
0143      0037 K7V1        EQU     #37          ;K7 Current Value
0144      0038 K8V2        EQU     #38          ;K8 New Value
0145      0039 K8V1        EQU     #39          ;K8 Current Value
0146      003A K9V2        EQU     #3A          ;K9 New Value
0147      003B K9V1        EQU     #3B          ;K9 Current Value
0148      003C K10V2       EQU     #3C          ;K10 New Value
0149      003D K10V1       EQU     #3D          ;K10 Current Value
0150           *
0151           *
0152           *    LPC status variable locations
0153           *
0154      003E TIMER       EQU     #3E          ;Stored Timer value for update
0155      003F SCALE       EQU     #3F          ;Timer register value used in INTP
0156      0040 FLAGS       EQU     #40          ;Flags used in LPC synthesis
0157      0041 FLAG1       EQU     #41          ;Flags used in LPC synthesis
0158      0042 MODE_BUF    EQU     #42          ;Stored value of Mode register
0159           ************************************************************
0160           *   Constant Definitions
0161           ************************************************************
0162           *
0163           *   Bit Size of Speech parameters
0164           *
0165      0004 EBITS       EQU     4            ;Number of Energy Bits
0166      0007 PBITS       EQU     7            ;Number of Pitch Bits
0167      0001 RBITS       EQU     1            ;Number of Repeat Bits
0168      0006 K1BITS      EQU     6            ;Number of K1 Bits
0169      0006 K2BITS      EQU     6            ;Number of K2 Bits
0170      0005 K3BITS      EQU     5            ;Number of K3 Bits
0171      0005 K4BITS      EQU     5            ;Number of K4 Bits
0172      0004 K5BITS      EQU     4            ;Number of K5 Bits
0173      0004 K6BITS      EQU     4            ;Number of K6 Bits
0174      0004 K7BITS      EQU     4            ;Number of K7 Bits
0175      0003 K8BITS      EQU     3            ;Number of K8 Bits
0176      0003 K9BITS      EQU     3            ;Number of K9 Bits
0177      0003 K10BITS     EQU     3            ;Number of K10 Bits
0178      0000 K11BITS     EQU     0            ;Number of K11 Bits
0179      0000 K12BITS     EQU     0            ;Number of K12 Bits
0180           *
0181           *   Prescale Values
```

```
0182           *
0183           *   PSvalue = Round(Samples * 2 * 30)/256  – 1
0184           *
0185           *   This comes from the fact that samples come every 30
0186           *   instruction cycles in LPC mode.  The factor of 2
0187           *   accounts for the cycle steal that happens in
0188           *   LPC mode.  When not in LPC mode, samples come
0189           *   every 60 instruction cycles, so it comes out the
0190           *   same.  The 128 divider is the full scale Timer
0191           *   register value.  The 0.5 addition is there
0192           *   to take care of rounding.
0193           *
0194      00C8 SAMPLES     EQU     200          ;Samples per frame
0195      005D PSVALUE     EQU     (SAMPLES*60/128)+(1/2)   ;Prescale Value
0196           *
0197           *   Device Constants
0198           *
0199      0F61 C1_Value    EQU     #F61         ;C1 Value
0200      0B67 C2_Value    EQU     #B67         ;C2 Value
0201      00FF MAX_RAM     EQU     #FF          ;Highest RAM location
0202      0032 ISAMPLE     EQU     #32          ;Instruction cycles per sample
0203           *
0204           *   Special Energy Values
0205           *
0206      000F ESTOP       EQU     15           ;Stop code
0207      0000 ESILENCE    EQU     0            ;Silence Code
0208           *
0209           *   Special Pitch Value
0210           *
0211      0000 PUnVoiced   EQU     0            ;UnVoiced Frame Code
0212      0018 UNV_PITCH   EQU     #18          ;Pitch used for Unvoiced Frames
0213           *
0214           *   End of sentence signal
0215           *
0216      00FF StopWord    EQU     #FF
0217           *
0218           *   FLAGS bit usage (and Set Masks)
0219           *
0220      0001 STOPFLAG    EQU     #01          ;Stop frame reached = 1
0221      0002 R_FLAG      EQU     #02          ;Repeat Frame = 1
0222      0004 Update_Flg  EQU     #04          ;Set high on update
0223      0008 Sil_Flg1    EQU     #08          ;New frame is silent = 1
0224      0010 Unv_Flg1    EQU     #10          ;New frame is unvoiced = 1
0225      0020 Int_Inh     EQU     #20          ;Change between Voice and Unv=1
0226      0040 Sil_Flg2    EQU     #40          ;Current frame is silent = 1
0227      0080 Unv_Flg2    EQU     #80          ;Current frame is unvoiced = 1
0228           *
0229           *   FLAG1 bit usage (and Set Masks)
0230           *
0231      0001 Int_Off     EQU     #01          ;Disable INTP routine = 1
0232           *
0233           *   MODE Register Bit Definitions
0234           *
0235      0001 INT1        EQU     #01          ;Enable Level 1 interrupt
0236      0002 LPC         EQU     #02          ;Enable LPC synthesis
0237      0004 PCM         EQU     #04          ;Enable PCM synthesis
0238      0008 INT2        EQU     #08          ;Enable Level 2 interrupt
0239      0020 RAMROM      EQU     #20          ;Enable GETs from RAM
0240      0040 MASTER      EQU     #40          ;Master/Slave Toggle
0241      0080 UNV         EQU     #80          ;Enable Unvoiced excitation
0242           *
0243           *  MODE Register 2 bit definitions
```

```
0244                *
0245          0001 PPC1        EQU     #01          ;PPC for channel 1
0246                *
0247                *   Port definitions
0248                *
0249          00FC Input_A     EQU     #FC          ;Port A input buffer
0250          00FD Pullup_A    EQU     #FD          ;Port A pullup select switch
0251          00FE Tristate_A  EQU     #FE          ;Port A Input/Output select switch
0252          00FF Output_A    EQU     #FF          ;Port A output buffer
0253
0254          00F8 Input_B     EQU     #F8          ;Port B input buffer
0255          00F9 Pullup_B    EQU     #F9          ;Port B pullup select switch
0256          00FA Tristate_B  EQU     #FA          ;Port B Input/Output select switch
0257          00FB Output_B    EQU     #FB          ;Port B output buffer
0258
0259                ************************************************************
0260                *   Start of program
0261                ************************************************************
0262 0000                      AORG    #0000
0263            84             SBR     GOGO         ;Power up Vector
0264 0001       84             SBR     GOGO         ;   (twice makes unconditional)
0265
0266 0002       84             SBR     GOGO         ;Wake up vector (same as power up)
0267 0003       84             SBR     GOGO         ;   (twice makes unconditional)
0268
0269 0004 4022 GOGO            BR      GO           ;Branch to start of program
0270                *
0271                ************************************************************
0272                *   Interrupt vectors
0273                ************************************************************
0274 0010                      AORG    #0010
0275            A0             SBR     INT2_00      ;Timer Underflow, PCM=0, LPC=0
0276 0011       A0             SBR     INT2_00      ;Timer Underflow, PCM=0, LPC=0
0277 0012       A0             SBR     INT2_01      ;Timer Underflow, PCM=0, LPC=1
0278 0013       A0             SBR     INT2_01      ;Timer Underflow, PCM=0, LPC=1
0279 0014       A0             SBR     INT2_10      ;Timer Underflow, PCM=1, LPC=0
0280 0015       A0             SBR     INT2_10      ;Timer Underflow, PCM=1, LPC=0
0281 0016       A0             SBR     INT2_11      ;Timer Underflow, PCM=1, LPC=1
0282 0017       A0             SBR     INT2_11      ;Timer Underflow, PCM=1, LPC=1
0283 0018       A0             SBR     INT1_00      ;Pin (B1) goes low interrupt
0284 0019       A0             SBR     INT1_00      ;Pin (B1) goes low interrupt
0285 001A       A0             SBR     INT1_01      ;Clock interrupt, PCM=0, LPC=1
0286 001B       A0             SBR     INT1_01      ;Clock interrupt, PCM=0, LPC=1
0287 001C       A0             SBR     INT1_10      ;Clock interrupt, PCM=1, LPC=0
0288 001D       A0             SBR     INT1_10      ;Clock interrupt, PCM=1, LPC=0
0289 001E       A0             SBR     INT1_11      ;Clock interrupt, PCM=1, LPC=1
0290 001F       A0             SBR     INT1_11      ;Clock interrupt, PCM=1, LPC=1
0291
0292                INT1_10
0293                INT1_01
0294                INT2_10
0295                INT2_00
0296                INT2_01
0297                INT2_11
0298                INT1_00
0299 0020       2F INT1_11     CLA
0300 0021       3E             RETI
0301
0302
0303                ************************************************************
0304                *   Do program INITs
0305                ************************************************************
```

```
0306 0022 6900 GO          TMAD    0
0307
0308 0024   2F             CLA                 ;Initialize mode register
0309 0025   1D             TAMODE
0310
0311 0026   20             CLX
0312 0027   13 RAM_LOOP    TAMIX               ;Initialize All RAM to zeros
0313 0028 61FF             XGEC    MAX_RAM
0314 002A 402E             BR      RAM_EXIT
0315 002C 4027             BR      RAM_LOOP
0316
0317 002E   16 RAM_EXIT    TAM                 ;initialize last RAM location (FF)
0318
0319 002F 6E01             TCA     1           ;Speak 1st phrase
0320 0031 0049             CALL    SPEAK
0321
0322 0033 6E02             TCA     2           ;Speak 2nd phrase
0323 0035 0049             CALL    SPEAK
0324           *
0325 0037 6E03             TCA     3           ;Speak 3rd phrase
0326 0039 0049             CALL    SPEAK
0327           *
0328 003B 6E04             TCA     4           ;Speak 4th phrase
0329 003D 0049             CALL    SPEAK
0330           *
0331 003F 6E05             TCA     5           ;Speak 5th phrase
0332 0041 0049             CALL    SPEAK
0333           *
0334 0043 6E06             TCA     6           ;Speak 5th phrase
0335 0045 0049             CALL    SPEAK
0336           *
0337 0047 4047 glop        BR      GLOP        ;loop forever
0338           ************************************************************
0339           *   Speak Utterance – Phrase number in A register
0340           ************************************************************
0341 0049   3B SPEAK       INTGR
0342 004A   2E             SALA                ;Double index to get offset
0343 004B 75A2             ACAAC   SPEECH      ;Add base of table
0344 004D   6D             LUAB                ;get address MSB
0345 004E   3A             IAC
0346 004F   6B             LUAA                ;Get address LSB
0347 0050   12             XBA
0348 0051   1B             SALA4               ;Combine MSB and LSB
0349 0052   1B             SALA4
0350 0053   2C             ABAAC
0351 0054   6C             LUAPS               ;Load Speech Address Register
0352
0353 0055   2F             CLA                 ;Kill K11 and K12 parameters
0354 0056 6A03             TAMD    K11
0355 0058 6A02             TAMD    K12
0356
0357 005A 7FF0             ACAAC   #FF0
0358 005C   18             TAX
0359
0360 005D   2F             CLA
0361 005E   13             TAMIX               ;FF0
0362 005F   13             TAMIX               ;FF1
0363 0060   13             TAMIX               ;FF2
0364
0365 0061 7162             ACAAC   #162
0366 0063   13             TAMIX               ;FF3
0367 0064   2F             CLA
```

```
0368 0065   13            TAMIX               ;FF4
0369
0370 0066 6A42            TAMD    MODE_BUF
0371 0068   1D            TAMODE
0372
0373 0069 6A40            TAMD    FLAGS       ;Init flags for speech
0374
0375 006B   2F            CLA                 ;Load C2 parameter
0376 006C 7B67            ACAAC   C2_Value    ;(a device constant)
0377 006E 6A0F            TAMD    C2
0378
0379 0070   2F            CLA                 ;Load C1 parameter
0380 0071 7F61            ACAAC   C1_Value    ;(a device constant)
0381 0073 6A0E            TAMD    C1
0382              *
0383              * Now we give an initial value to the Pitch in case the
0384              * utterance starts with a silent frame.
0385              *
0386 0075 700C            ACAAC   #0C
0387 0077 6A24            TAMD    PHV1
0388 0079 6A22            TAMD    PHV2
0389              *
0390              * Now I initialize the prescale and timer so that they
0391              * will count during the coming calls to UPDATE
0392              *
0393 007B 6E5D            TCA     PSVALUE     ;Initialize prescale
0394 007D   19            TAPSC
0395 007E   1E            TATM
0396              *
0397              * Now we preload the first two frames.
0398              *
0399
0400 007F 01A7            CALL    UPDATE      ;Load first frame
0401 0081 01A7            CALL    UPDATE      ;Load 2nd frame
0402              *
0403              * Now we give a full scale value to the Timer so that the
0404              * interpolation in this first call to INTP will be correctly
0405              * scaled.  Then I do the first call to INTP to preload the
0406              * first valid interpolation.
0407              *
0408 0083 6E7F            TCA     #7F         ;Pretend there was a previous
0409 0085 6A3E            TAMD    TIMER       ;    update
0410 0087   1E            TATM                ;Set timer to max value to
0411                                          ;    disable interpolation
0412
0413 0088 009E            CALL    INTP        ;Do first interpolation
0414
0415              *
0416              * Now we enable the synthesizer for speech
0417              *
0418              * We do this in two stages so that we can reset the
0419              * interrupt pending latch without it being immediately
0420              * set again by the B1(low) interrupt.
0421              *
0422 008A 6242            TCX     MODE_BUF    ;Turn on LPC synthesizer
0423 008C 6402            ORCM    LPC
0424 008E   11            TMA
0425 008F   1D            TAMODE
0426
0427              *
0428              * Now we loop until the utterance is complete.  When the
0429              * utterance is finished, the routine UPDATE will execute a
```

```
0430           * RETN instruction which will exit this routine.  In the
0431           * mean time, this loop will poll the Timer register and
0432           * update the frame whenever it underflows.
0433           *
0434           * * * * *
0435 0090   20 SPEAK_LP    CLX
0436 0091   17             TTMA                ;Fetch timer value
0437 0092 6380             AGEC    #80         ;Has it underflowed?
0438 0094 41A7             BR      UPDATE      ;    yes, do an update
0439
0440 0096   20             CLX
0441 0097   22             DECXN
0442 0098 6601             TSTCM   PPC1
0443 009A 009E             CALL    INTP
0444 009C 4090             BR      SPEAK_LP    ;    no, wait some more
0445
0446
0447           * * * * *
0448           * INTERPOLATION ROUTINE
0449           * * * * *
0450           * First we need to get the current value of the timer
0451           * register and store it away.
0452           * * * * *
0453 009E   20 INTP        CLX
0454 009F   22             DECXN
0455 00A0 65FE             ANDCM   ~PPC1
0456 00A2   20             CLX
0457
0458 00A3   17             TTMA                ;Get timer register contents
0459 00A4 6380             AGEC    #80         ;Has time underflowed?
0460 00A6 40AA             BR      Adjust      ;   yes, Trick Scale to avoid prob
0461 00A8 40AB             BR      DoScale     ;   no, Use Timer as is
0462
0463 00AA   2F Adjust      CLA                 ;Pretend no underflow
0464 00AB 6A3F DoScale     TAMD    SCALE       ;Load timer val into scale
0465           * * * * *
0466           * See if this routine is enabled.  If it is not, exit
0467           * the routine.
0468           * * * * *
0469 00AD 6241             TCX     FLAG1       ;Point to flag
0470 00AF 6601             TSTCM   Int_Off     ;If routine disabled...
0471 00B1 41A6             BR      IRETN       ; ...branch to exit point
0472           * * * * *
0473           * Next we need to see if the frame type has changed between
0474           * voiced and unvoiced frames.  If it has, we do not want to
0475           * interpolate between them; we just want to use the current
0476           * frame values until we have two frames of the same type to
0477           * interpolate between.
0478           * * * * *
0479 00B3 6240 TINTP         TCX     FLAGS       ;Point to status flags
0480 00B5 6620             TSTCM   Int_Inh     ;Is interpolation inhibited?
0481 00B7 40BB             BR      NOINT       ;   yes, use inhibit code
0482 00B9 40D6             BR      INTPCH      ;   no, use interpolation code
0483           * * * * *
0484           * The following code is reached if interpolation is
0485           * inhibited. It sets the stored timer value to #7F which
0486           * effectively forces the interpolation to yield the old
0487           * values for the working values, thus effectively disabling
0488           * interpolation.
0489           * * * * *
0490 00BB 6E7F NOINT         TCA     #7F         ;Set Scale factor to
0491 00BD 6A3F             TAMD    SCALE       ;   highest value
```

```
0492           *
0493           * If the new frame has a voicing different from the last
0494           * frame, we want to zero the energy until the Unvoiced bit
0495           * in the mode register is changed and the K parameters are
0496           * all to the correct values.  We therefore check in this
0497           * section of code to see if the frame voicing is different
0498           * from the setting in the Mode Register.  If it is, we zero
0499           * the energy until after the Mode Register is modified.
0500           *
0501 00BF 6240             TCX     FLAGS
0502 00C1 6680             TSTCM   Unv_Flg2     ;Is new frame unvoiced?
0503 00C3 40CD             BR      Uv           ;    yes, go to unvoiced branch
0504
0505 00C5 6242             TCX     Mode_Buf     ;New frame is voiced
0506 00C7 6680             TSTCM   UNV          ;Has mode changed to unvoiced?
0507 00C9 40D3             BR      ClrEN        ;     yes, clear the energy
0508 00CB 40D6             BR      INTPCH       ;     no, no action required
0509
0510 00CD 6242 Uv          TCX     Mode_Buf     ;New frame is unvoiced
0511 00CF 6680             TSTCM   UNV          ;Has voicing mode changed?
0512 00D1 40D6             BR      INTPCH       ;     no, no action required
0513
0514 00D3   2F ClrEN       CLA                  ;Zero Energy during update
0515 00D4 6A01             TAMD    EN
0516
0517           *
0518           * Interpolate Pitch and write the result to the pitch
0519           * register
0520           *
0521 00D6 6222 INTPCH      TCX     PHV2         ;Combine new pitch and fractional
0522 00D8   14             TMAIX                ;     pitch and leave in
0523 00D9   1B             SALA4                ;     the B register
0524 00DA   28             AMAAC
0525 00DB   21             IXC
0526 00DC   1A             TAB
0527 00DD   14             TMAIX                ;Combine current pitch and
0528 00DE   1B             SALA4                ;    current fractional pitch
0529 00DF   28             AMAAC                ;    and leave in A register
0530
0531 00E0   2D             SBAAN                ;(Pcurrent – Pnew)
0532 00E1 623F             TCX     SCALE
0533 00E3   39             AXMA                 ;(Pcurrent – Pnew) * Timer
0534 00E4   2C             ABAAC                ;Pnew + (Pcurrent – Pnew)* Timer
0535 00E5   2E             SALA                 ;Double value to index excitation
0536 00E6   1C             TASYN                ;Write to pitch register
0537
0538           *
0539           * Interpolate Energy and store the result in the working
0540           * register
0541           *
0542 00E7 6220             TCX     ENV2         ;Put New Energy (adjusted to
0543 00E9   14             TMAIX                ;12 bits) in B register
0544 00EA   1B             SALA4
0545 00EB   1A             TAB
0546 00EC   14             TMAIX
0547 00ED   1B             SALA4                ;Put Current Energy (adjusted to
0548                                            ;    12 bits) in A register
0549 00EE   2D             SBAAN
0550 00EF 623F             TCX     SCALE
0551 00F1   39             AXMA                 ;(Ecurrent – Enew) * Timer
0552 00F2   2C             ABAAC                ;Enew + (Ecurrent – Enew)*Timer
0553 00F3 6A00             TAMD    EN_TEMP      ;Store Energy till mode is switched
```

```
0554
0555 00F5   3C            EXTSG               ;Allow negative K parameters
0556           *
0557           * Interpolate K1 and store the result in the working K1
0558           * register
0559           *
0560 00F6 6226            TCX     K1V2        ;Combine New K1 and New
0561 00F8   14            TMAIX               ;fractional K1 and
0562 00F9   1B            SALA4               ;leave in the B register
0563 00FA   28            AMAAC
0564 00FB   21            IXC
0565 00FC   1A            TAB
0566
0567 00FD   14            TMAIX               ;Combine current K1 and
0568 00FE   1B            SALA4               ;current fractional K1 and
0569 00FF   28            AMAAC               ;leave in the A register
0570
0571 0100   2D            SBAAN               ;(K1current – K1new)
0572 0101 623F            TCX     SCALE
0573 0103   39            AXMA                ;(K1current – K1new) * Timer
0574 0104   2C            ABAAC               ;K1new+(K1current–K1new)* Timer
0575 0105 6A0D            TAMD    K1          ;Load interpolated K1 value
0576           *
0577           * Interpolate K2 and store the result in the
0578           * working K2 register
0579           *
0580 0107 622A            TCX     K2V2        ;Put New K2 (adjusted to
0581 0109   14            TMAIX               ;12 bits) in B register
0582 010A   1B            SALA4
0583 010B   28            AMAAC
0584 010C   21            IXC
0585 010D   1A            TAB
0586
0587 010E   14            TMAIX               ;Put Current K5 (adjusted to
0588 010F   1B            SALA4               ;12 bits) in A register
0589 0110   28            AMAAC
0590
0591 0111   2D            SBAAN               ;(K2current – K2new)
0592 0112 623F            TCX     SCALE
0593 0114   39            AXMA                ;(K2current – K2new) * Timer
0594 0115   2C            ABAAC               ;K2new+(K2current–K2new)* Timer
0595 0116 6A0C            TAMD    K2          ;Load interpolated K2 value
0596           *
0597           * Interpolate K3 and store the result in the working K3
0598           * register
0599           *
0600 0118 622E            TCX     K3V2        ;Put New K3 (adjusted to
0601 011A   14            TMAIX               ;12 bits) in B register
0602 011B   1B            SALA4
0603 011C   1A            TAB
0604
0605 011D   14            TMAIX               ;Put Current K3 (adjusted to
0606 011E   1B            SALA4               ;12 bits) in A register
0607
0608 011F   2D            SBAAN               ;(K3current – K3new)
0609 0120 623F            TCX     SCALE
0610 0122   39            AXMA                ;(K3current – K3new) * Timer
0611 0123   2C            ABAAC               ;K3new+(K3current–K3new)* Timer
0612 0124 6A0B            TAMD    K3          ;Load interpolated K3 value
0613           *
0614           * Interpolate K4 and store the result in the working K4
0615           * register
```

```
0616           *
0617 0126 6230             TCX     K4V2          ;Put New K4 (adjusted to
0618 0128   14             TMAIX                 ;12 bits) in B register
0619 0129   1B             SALA4
0620 012A   1A             TAB
0621
0622 012B   14             TMAIX                 ;Put Current K5 (adjusted to
0623 012C   1B             SALA4                 ;12 bits) in A register
0624
0625 012D   2D             SBAAN                 ;(K4current – K4new)
0626 012E 623F             TCX     SCALE
0627 0130   39             AXMA                  ;(K4current – K4new) * Timer
0628 0131   2C             ABAAC                 ;K4new+(K4current–K4new)* Timer
0629 0132 6A0A             TAMD    K4            ;Load interpolated K4 value
0630           *
0631           * Interpolate K5 and store the result in the working K5
0632           * register
0633           *
0634 0134 6232             TCX     K5V2          ;Put New K5 (adjusted to
0635 0136   14             TMAIX                 ;12 bits) in B register
0636 0137   1B             SALA4
0637 0138   1A             TAB
0638 0139   14             TMAIX                 ;Put Current K5 (adjusted to
0639 013A   1B             SALA4                 ;12 bits) in A register
0640
0641 013B   2D             SBAAN                 ;(K5current – K5new)
0642 013C 623F             TCX     SCALE
0643 013E   39             AXMA                  ;(K5current – K5new) * Timer
0644 013F   2C             ABAAC                 ;K5new+(K5current–K5new)* Timer
0645 0140 6A09             TAMD    K5            ;Load interpolated K5 value
0646           *
0647           * Interpolate K6 and store the result in the working K6
0648           * register
0649           *
0650 0142 6234             TCX     K6V2          ;Put New K6 (adjusted to
0651 0144   14             TMAIX                 ;12 bits) in B register
0652 0145   1B             SALA4
0653 0146   1A             TAB
0654 0147   14             TMAIX                 ;Put Current K6 (adjusted to
0655 0148   1B             SALA4                 ;12 bits) in A register
0656
0657 0149   2D             SBAAN                 ;(K6current – K6new)
0658 014A 623F             TCX     SCALE
0659 014C   39             AXMA                  ;(K6current – K6new) * Timer
0660 014D   2C             ABAAC                 ;K6new+(K6current–K6new)* Timer
0661 014E 6A08             TAMD    K6            ;Load interpolated K6 value
0662           *
0663           * Interpolate K7 and store the result in the working K7
0664           * register
0665           *
0666 0150 6236             TCX     K7V2          ;Put New K7 (adjusted to
0667 0152   14             TMAIX                 ;12 bits) in B register
0668 0153   1B             SALA4
0669 0154   1A             TAB
0670 0155   14             TMAIX                 ;Put Current K7 (adjusted to
0671 0156   1B             SALA4                 ;12 bits) in A register
0672
0673 0157   2D             SBAAN                 ;(K7current – K7new)
0674 0158 623F             TCX     SCALE
0675 015A   39             AXMA                  ;(K7current – K7new) * Timer
0676 015B   2C             ABAAC                 ;K7new+(K7current–K7new)* Timer
0677 015C 6A07             TAMD    K7            ;Load interpolated K7 value
```

```
0678           *
0679           * Interpolate K8 and store the result in the working K8
0680           * register
0681           *
0682 015E 6238             TCX     K8V2         ;Put New K8 (adjusted to
0683 0160   14             TMAIX                ;12 bits) in B register
0684 0161   1B             SALA4
0685 0162   1A             TAB
0686
0687 0163   14             TMAIX                ;Put Current K8 (adjusted to
0688 0164   1B             SALA4                ;12 bits) in A register
0689
0690 0165   2D             SBAAN                ;(K8current – K8new)
0691 0166 623F             TCX     SCALE
0692 0168   39             AXMA                 ;(K8current – K8new) * Timer
0693 0169   2C             ABAAC                ;K8new+(K8current–K8new)* Timer
0694 016A 6A06             TAMD    K8           ;Load interpolated K8 value
0695           *
0696           * Interpolate K9 and store the result in the working K9
0697           * register
0698           *
0699 016C 623A             TCX     K9V2         ;Put New K9 (adjusted to
0700 016E   14             TMAIX                ;12 bits) in B register
0701 016F   1B             SALA4
0702 0170   1A             TAB
0703
0704 0171   14             TMAIX                ;Put Current K9 (adjusted to
0705 0172   1B             SALA4                ;12 bits) in A register
0706
0707 0173   2D             SBAAN                ;(K9current – K9new)
0708 0174 623F             TCX     SCALE
0709 0176   39             AXMA                 ;(K9current – K9new) * Timer
0710 0177   2C             ABAAC                ;K9new+(K9current–K9new)* Timer
0711 0178 6A05             TAMD    K9           ;Load interpolated K8 value
0712           *
0713           * Interpolate K10 and store the result in the working K10
0714           * register
0715           *
0716 017A 623C             TCX     K10V2        ;Put New K10 (adjusted to
0717 017C   14             TMAIX                ;12 bits) in B register
0718 017D   1B             SALA4
0719 017E   1A             TAB
0720
0721 017F   14             TMAIX                ;Put Current K10 (adjusted to
0722 0180   1B             SALA4                ;12 bits) in A register
0723
0724 0181   2D             SBAAN                ;(K10current – K10new)
0725 0182 623F             TCX     SCALE
0726 0184   39             AXMA                 ;(K10current – K10new) * Timer
0727 0185   2C             ABAAC                ;K10new+(K10current–K10new)* Timer
0728 0186 6A04             TAMD    K10          ;Load interpolated K10 value
0729           *
0730           * K11 and K12 are not needed for LPC 10, so I have
0731           * commented them out.
0732           *
0733           * Interpolate K11 and store the result in the working K11
0734           * register
0735           *
0736           *       TCX     K11V2     ;Put New K11 (adjusted to
0737           *       TMAIX             ;  12 bits) in B register
0738           *       SALA4
0739           *       TAB
```

```
0740
0741           *       TMAIX             ;Put Current K11 (adjusted to
0742           *       SALA4             ;  12 bits) in A register
0743
0744           *       SBAAN             ;(K11current – K11new)
0745           *       TCX     SCALE
0746           *       AXMA              ;(K11current – K11new) * Timer
0747           *       ABAAC             ;K11new+(K11current–K11new)*Timer
0748           *       TAMD    K11       ;Load interpolated K11 value
0749           *
0750           * Interpolate K12 and store the result in the working
0751           * K12 register
0752           *
0753           *       TCX     K12V2     ;Put New K12 (adjusted to
0754           *       TMAIX             ;  12 bits) in B register
0755           *       SALA4
0756           *       TAB
0757
0758           *       TMAIX             ;Put Current K12 (adjusted to
0759           *       SALA4             ;  12 bits) in A register
0760
0761           *       SBAAN             ;(K12current – K12new)
0762           *       TCX     SCALE
0763           *       AXMA              ;(K12current – K12new) * Timer
0764           *       ABAAC             ;K12new+(K12current–K12new)*Timer
0765           *       TAMD    K12       ;Load interpolated K12 value
0766           *
0767           *
0768           * Set voiced/unvoiced mode according to current frame type.
0769           * This is done in a two step fashion:  first the value in
0770           * the MODE_BUF register is adjusted with an AND or OR
0771           * operation, then the result is written to the synthesizer
0772           * with a TAMODE operation.  We do it this way to keep a copy
0773           * of the current status of the synthesizer mode at all time.
0774           *
0775 0188   3B STMODE      INTGR               ;Back to integer mode
0776 0189 6240             TCX     FLAGS
0777 018B 65FB             ANDCM   ~Update_Flg  ;Signal that interp done
0778 018D 6680             TSTCM   Unv_Flg2     ;Is current frame unvoiced?
0779 018F 419C             BR      SETUV        ;yes, set mode to unvoiced
0780 0191 6242             TCX     MODE_BUF     ;no, ...
0781 0193 657F             ANDCM   ~UNV         ;...set mode to voiced
0782 0195   11             TMA
0783 0196   1D             TAMODE
0784
0785 0197 6900             TMAD    EN_TEMP      ;Change Energy parameter
0786 0199 6A01             TAMD    EN           ;to correct value
0787
0788 019B   3D             RETN                 ;Return from first call
0789
0790 019C 6242 SETUV       TCX     MODE_BUF     ;Current frame is unvoiced, so
0791 019E 6480             ORCM    UNV          ;set mode to unvoiced.
0792 01A0   11             TMA
0793 01A1   1D             TAMODE
0794
0795 01A2 6900             TMAD    EN_TEMP      ;Change Energy parameter
0796 01A4 6A01             TAMD    EN           ;to correct value.
0797
0798 01A6   3D IRETN       RETN                 ;Return from first call
0799
0800           * * * * *
0801           * Update the parameters for a new frame
```

```
0802                * * * * *
0803                * * * * *
0804                * To prevent double updates, if the stored value of the
0805                * timer register is zero, then we need to change it to #7F.
0806                * If we do not do this, then the polling routine will
0807                * discover an underflow and call Update a second time.
0808                * * * * *
0809 01A7 623E UPDATE      TCX     TIMER        ;Get stored value
0810 01A9   11             TMA                  ;of Timer into A
0811
0812 01AA 6000             ANEC    0            ;Is it zero?
0813 01AC 41B1             BR      UPDT00       ;no, do nothing
0814 01AE 6E7F             TCA     #7F          ;yes, replace value
0815 01B0   16             TAM
0816                * * * * *
0817                * First we need to test to see if a stop frame was
0818                * encountered on the last pass through the routine.   If the
0819                * previous frame was a stop frame, we need to turn off the
0820                * synthesizer and stop speaking.
0821                * * * * *
0822 01B1 6240 UPDT00      TCX     FLAGS
0823 01B3 6601             TSTCM   STOPFLAG     ;Was stop frame encountered
0824 01B5 42C9             BR      STOP         ;yes, stop speaking
0825                * * * * *
0826                * Transfer the state of the previous frame to the Unvoiced
0827                * flag (Current).
0828                * * * * *
0829 01B7 6610             TSTCM   Unv_Flg1     ;Was previous frame unvoiced
0830 01B9 41BF             BR      SUNVL        ;yes, current frame=unvoiced
0831 01BB 657F             ANDCM   ~Unv_Flg2    ;no, current frame=voiced
0832 01BD 41C1             BR      TSIL         ;and continue
0833
0834 01BF 6480 SUNVL       ORCM    Unv_Flg2     ;Set current frame unvoiced.
0835                * * * * *
0836                * Transfer the state of the previous frame to the
0837                * Silence flag (Current).
0838                * * * * *
0839 01C1 6608 TSIL        TSTCM   Sil_Flg1     ;Was previous frame silent?
0840 01C3 41C9             BR      SSIL         ;yes, current frame  = silent
0841 01C5 65BF             ANDCM   ~Sil_Flg2    ;no, current frame not silent
0842 01C7 41CB             BR      ZROFLG       ;and continue
0843
0844 01C9 6440 SSIL        ORCM    Sil_Flg2     ;Set current frame silent
0845                * * * * *
0846                * Reset the Repeat Flag, new Silence Flag, new Unvoiced
0847                * Flag, and Interpolation Inhibit flag so that new
0848                * values can be loaded in this routine.
0849                * * * * *
0850 01CB 6240 ZROFLG      TCX     FLAGS
0851 01CD 65C5             ANDCM   #C5
0852                * * * * *
0853                * Transfer the new frame parameters into the
0854                * storage location used for the current frame parameters.
0855                * * * * *
0856 01CF 6220             TCX     ENV2         ;Point to energy
0857 01D1   14             TMAIX                ;Transfer new frame En parameter
0858 01D2   13             TAMIX                ;to current frame location
0859                *-----PITCH-----
0860 01D3   14             TMAIX                ;Transfer new frame pitch
0861 01D4 6A24             TAMD    PHV1         ;to current frame location
0862
0863 01D6   14             TMAIX                ;Transfer new fractional pitch
```

```
0864 01D7   21            IXC                 ;to current frame location
0865 01D8   13            TAMIX
0866                 *-----K1-----
0867 01D9   14            TMAIX               ;Transfer new frame K1 parameter
0868 01DA 6A28            TAMD    K1V1        ;to current frame location
0869 01DC   14            TMAIX               ;Transfer new fractional K1
0870 01DD   21            IXC                 ;to current frame location
0871 01DE   13            TAMIX
0872                 *-----K2-----
0873 01DF   14            TMAIX               ;Transfer new frame K2 parameter
0874 01E0 6A2C            TAMD    K2V1        ;to current frame location
0875 01E2   14            TMAIX               ;Transfer new fractional K2
0876 01E3   21            IXC                 ;to current frame location
0877 01E4   13            TAMIX
0878                 *-----K3-----
0879 01E5   14            TMAIX               ;Transfer new frame K3 parameter
0880 01E6   13            TAMIX               ;to current frame location
0881                 *-----K4-----
0882 01E7   14            TMAIX               ;Transfer new frame K4 parameter
0883 01E8   13            TAMIX               ;to current frame location
0884                 *-----K5-----
0885 01E9   14            TMAIX               ;Transfer new frame K5 parameter
0886 01EA   13            TAMIX               ;to current frame location
0887                 *-----K6-----
0888 01EB   14            TMAIX               ;Transfer new frame K6 parameter
0889 01EC   13            TAMIX               ;to current frame location
0890                 *-----K7-----
0891 01ED   14            TMAIX               ;Transfer new frame K7 parameter
0892 01EE   13            TAMIX               ;to current frame location
0893                 *-----K8-----
0894 01EF   14            TMAIX               ;Transfer new frame K8 parameter
0895 01F0   13            TAMIX               ;to current frame location
0896                 *-----K9-----
0897 01F1   14            TMAIX               ;Transfer new frame K9 parameter
0898 01F2   13            TAMIX               ;to current frame location
0899                 *-----K10-----
0900 01F3   14            TMAIX               ;Transfer new frame K10 parameter
0901 01F4   13            TAMIX               ;to current frame location
0902                 * * * * *
0903                 * K11 and K12 are not used in LPC 10 synthesis.  the code
0904                 * has been commented out.
0905                 * * * * *
0906                 *-----K11-----
0907                 *       TMAIX             ;Transfer new frame K11 parameter
0908                 *       TAMIX             ;to current frame location
0909                 *-----K12-----
0910                 *       TMAIX             ;Transfer new frame K12 parameter
0911                 *       TAMIX             ;to current frame location
0912                 *-----
0913                 * * * * *
0914                 * We have now discarded the "current" values by replacing
0915                 * them with the "new" values.  We now need to read in
0916                 * another frame of speech data and use them as the
0917                 * new "new" values.
0918                 * * * * *
0919                 *----- ENERGY -----
0920 01F5   2F            CLA
0921 01F6 6240            TCX     FLAGS
0922 01F8   33            GET     EBITS       ;Get coded energy
0923 01F9 6000            ANEC    ESILENCE    ;Is it a silent frame?
0924 01FB 4201            BR      UPDT0       ;No, continue
0925 01FD 6408            ORCM    Sil_Flg1    ;Yes, set silence flag
```

```
0926 01FF 42AB             BR      ZeroKs       ;and zero K params
0927
0928 0201 600F UPDT0       ANEC    ESTOP        ;Is it a stop frame?
0929 0203 4209             BR      UPDT1        ;no, continue
0930 0205 6409             ORCM    STOPFLAG+Sil_Flg1  ;yes, set flags
0931 0207 42AB             BR      ZeroKs       ;and zero Ks
0932
0933 0209 730A UPDT1       ACAAC   TBLEN        ;Add table offset to energy
0934 020B   6B             LUAA                 ;Get decoded energy
0935 020C 6A20             TAMD    ENV2         ;Store the Energy in RAM
0936           * * * * *
0937           * If this is a silent frame, we are done with the update If
0938           * the previous frame was silent, the new frame should be
0939           * spoken immediately with no ramp up due to interpolation
0940           * * * * *
0941 020E 6240             TCX     FLAGS
0942 0210 6608             TSTCM   Sil_Flg1     ;Is this a silent frame?
0943 0212 42E1             BR      RTN          ;yes, exit
0944           * * * * *
0945           * A repeat frame will use the K parameter from the previous
0946           * frame.  If it is, we need to set a flag.
0947           * * * * *
0948 0214   30 UPDT2       GET     RBITS        ;Get the Repeat bit
0949 0215 6701             TSTCA   #01          ;Is this a repeat frame?
0950 0217 421B             BR      SFLG1        ;yes, set repeat flag
0951 0219 421D             BR      UPDT3
0952
0953 021B 6402 SFLG1       ORCM    R_FLAG       ;Set repeat flag
0954
0955           *----- PITCH -----
0956
0957 021D   2F UPDT3       CLA
0958 021E   33             GET     4            ;Get coded pitch
0959 021F   32             GET     3            ;Get coded pitch
0960 0220 6000             ANEC    PUnVoiced    ;Is the frame unvoiced?
0961 0222   A5             SBR     UPDT3A       ;no, continue
0962 0223 6410             ORCM    Unv_Flg1     ;yes, set unvoiced flag
0963
0964 0225   2E UPDT3A      SALA                 ;Double coded pitch and
0965 0226 731A             ACAAC   TBLPH        ;add table offset to point
0966
0967 0228   6D             LUAB                 ;Get decoded pitch
0968 0229   3A             IAC
0969 022A   6B             LUAA                 ;Get decoded fractional pitch
0970 022B 6222             TCX     PHV2         ;Store the pitch and fractional
0971 022D   2A             TBM                  ;pitch in RAM
0972 022E   21             IXC
0973 022F   16             TAM
0974           * * * * *
0975           * If the voicing has changed with the new frame, then we
0976           * need to change the voicing in the mode register.
0977           * * * * *
0978 0230 6240             TCX     FLAGS
0979 0232 6610             TSTCM   Unv_Flg1     ;Is the new frame unvoiced?
0980 0234   B7             SBR     UPDT3B       ;yes, continue
0981 0235 4241             BR      VOICE        ;no, go to voiced code
0982           * * * * *
0983           * The following code is reached if the new frame is
0984           * unvoiced. We inspect the flags to see if the previous
0985           * frame was either silent or voiced. If either condition
0986           * applies, then we branch to code which inhibits
0987           * interpolation.
```

```
0988                    * * * * *
0989 0237 6640 UPDT3B   TSTCM   Sil_Flg2     ;Was the previous frame silent
0990 0239 4247          BR      UPDT5        ;yes, inhibit interpolation
0991
0992 023B 6680          TSTCM   Unv_Flg2     ;Was the previous frame unvoiced?
0993 023D 4249          BR      UPDT4        ;yes, no need to change anything
0994 023F 4247          BR      UPDT5        ;no, inhibit interpolation
0995                    * * * * *
0996                    * The following code is reached if the new frame is
0997                    * voiced. We inspect the flags to see if the previous
0998                    * frame was also voiced. If it was not, we need to inhibit
0999                    * interpolation.
1000                    * * * * *
1001 0241 6680 VOICE    TSTCM   Unv_Flg2     ;Was the previous frame voiced?
1002 0243 4247          BR      UPDT5        ;no, set no interpolation flag
1003 0245 4249          BR      UPDT4        ;yes, no need to change anything
1004
1005 0247 6420 UPDT5    ORCM    Int_Inh      ;Inhibit interpolation
1006                    * * * * *
1007                    * Now we test the repeat flag.  If the new frame is a repeat
1008                    * frame, then the current values are used for the K factors,
1009                    * so new values do not need to be loaded and we can exit the
1010                    * routine now.
1011                    * * * * *
1012 0249 6602 UPDT4    TSTCM   R_FLAG       ;Is repeat flag set?
1013 024B 42E1          BR      RTN          ;yes, exit routine
1014                    * * * * *
1015                    * Now we need to load the "new" K factors (K1 through K10).
1016                    * Each K factor is a 12 bit value which will be stored in
1017                    * two bytes.  The most significant 8 bits in the first byte,
1018                    * and the least significant 4 bits (called the fractional
1019                    * value) in the second byte. For K3 through K12, the
1020                    * fractional part is assumed to be zero.  K11 and K12 are
1021                    * not used in LPC10 synthesis, and the code loading them is
1022                    * commented out.  A coded factor is read into the A
1023                    * register.  It is then converted to a pointer to a table
1024                    * element which contains the uncoded factor. Since the K1
1025                    * and K2 table elements consist of two bytes, the conversion
1026                    * consists of doubling the coded factor and adding the
1027                    * result to the start of the table.  Since the K4 & K4 table
1028                    * elements consist of one byte, the coded factor is added
1029                    * directly to the start of the table.  Once the pointer has
1030                    * been set up, the uncoded factor is fetched and stored into
1031                    * RAM.
1032                    * * * * *
1033                    *-----K1-----
1034 024D   2F              CLA
1035 024E   33              GET     4            ;Get coded K1
1036 024F   31              GET     2            ;Get coded K1
1037 0250   2E              SALA                 ;Convert it to a
1038 0251 741A              ACAAC   TBLK1        ;pointer to table element
1039 0253   6D              LUAB                 ;Fetch MSB of uncoded K1
1040 0254   3A              IAC
1041 0255   6B              LUAA                 ;Fetch fractional K1
1042 0256 6226              TCX     K1V2
1043 0258   2A              TBM                  ;Store uncoded K1
1044 0259   21              IXC
1045 025A   16              TAM                  ;Store fractional K1
1046                    *-----K2-----
1047 025B   2F              CLA
1048 025C   33              GET     4            ;Get coded K2
1049 025D   31              GET     2            ;Get coded K2
```

```
1050
1051 025E   2E             SALA                ;Convert it to a
1052 025F 749A             ACAAC   TBLK2        ;pointer to table element
1053
1054 0261   6D             LUAB                ;Fetch MSB of uncoded K2
1055 0262   3A             IAC
1056 0263   6B             LUAA                ;Fetch fractional K2
1057 0264 622A             TCX     K2V2
1058 0266   2A             TBM                 ;Store uncoded K2
1059 0267   21             IXC
1060 0268   16             TAM                 ;Store fractional K2
1061           *-----K3-----
1062 0269   2F             CLA
1063 026A   33             GET     4           ;Get Index into K3 table
1064 026B   30             GET     1           ;Get Index into K3 table
1065 026C 751A             ACAAC   TBLK3       ;and add offset of table to it
1066
1067 026E   6B             LUAA                ;Get uncoded K3
1068 026F 6A2E             TAMD    K3V2        ;and store it in RAM
1069           *-----K4-----
1070 0271   2F             CLA
1071 0272   33             GET     4           ;Get Index into K4 table
1072 0273   30             GET     1           ;Get Index into K4 table
1073 0274 753A             ACAAC   TBLK4       ;and add offset of table to it
1074 0276   6B             LUAA                ;Get uncoded K4
1075 0277 6A30             TAMD    K4V2        ;and store it in RAM
1076           * * * * *
1077           * If this is an unvoiced frame, we only use four K factors,
1078           * so we load zeroes to the rest of the K factors.  If this
1079           * is a voiced frame, load the rest of the uncoded factors.
1080           * * * * *
1081 0279 6240             TCX     FLAGS
1082 027B 6610             TSTCM   Unv_Flg1    ;Is this an unvoiced frame?
1083 027D 42BA             BR      UNVC        ;Yes, zero rest of factors
1084           * * * * *
1085           * The following code is executed if the new frame is
1086           * voiced.  Since we assume that the fractional parameter is
1087           * zero for the remaining K factors, the table elements are
1088           * only one byte long.  As with K3 and K4, the conversion to
1089           * a table pointer consists of adding the coded factor to the
1090           * start of the table.
1091           * * * * *
1092           *-----K5-----
1093 027F   2F             CLA
1094 0280   33             GET     K5BITS      ;Get Index into K5 table
1095 0281 755A             ACAAC   TBLK5       ;and add offset of table to it
1096
1097 0283   6B             LUAA                ;Get uncoded K5
1098 0284 6A32             TAMD    K5V2        ;and store it in RAM
1099           *-----K6-----
1100 0286   2F             CLA
1101 0287   33             GET     K6BITS      ;Get Index into K6 table
1102 0288 756A             ACAAC   TBLK6       ;and add offset of table to it
1103 028A   6B             LUAA                ;Get uncoded K6
1104 028B 6A34             TAMD    K6V2        ;and store it in RAM
1105           *-----K7-----
1106 028D   2F             CLA
1107 028E   33             GET     K7BITS      ;Get Index into K7 table
1108 028F 757A             ACAAC   TBLK7       ;and add offset of table to it
1109 0291   6B             LUAA                ;Get uncoded K7
1110 0292 6A36             TAMD    K7V2        ;and store it in RAM
1111           *-----K8-----
```

```
1112 0294   2F             CLA
1113 0295   32             GET     K8BITS       ;Get Index into K8 table
1114 0296 758A             ACAAC   TBLK8        ;and add offset of table to it
1115 0298   6B             LUAA                 ;Get uncoded K8
1116 0299 6A38             TAMD    K8V2         ;and store it in RAM
1117           *-----K9-----
1118 029B   2F             CLA
1119 029C   32             GET     K9BITS       ;Get Index into K9 table
1120 029D 7592             ACAAC   TBLK9        ;and add offset of table to it
1121 029F   6B             LUAA                 ;Get uncoded K9
1122 02A0 6A3A             TAMD    K9V2         ;and store it in RAM
1123           *-----K10-----
1124 02A2   2F             CLA
1125 02A3   32             GET     K10BITS      ;Get Index into K10 table
1126 02A4 759A             ACAAC   TBLK10       ;and add offset of table to it
1127 02A6   6B             LUAA                 ;Get uncoded K10
1128 02A7 6A3C             TAMD    K10V2        ;and store it in RAM
1129           * * * * *
1130           * Since K11 and K12 are not used in LPC 10, the K11 and K12
1131           * code is commented out.
1132           * * * * *
1133           *-----K11-----
1134           *       CLA
1135           *       GET     K11BITS      ;Get Index into K11 table
1136           *       ACAAC   TBLK11       ;  and add offset of table to it
1137           *       LUAA                 ;Get uncoded K11
1138           *       TAMD    K11V2        ;  and store it in RAM
1139           *-----K12-----
1140           *       CLA
1141           *       GET     K12BITS      ;Get Index into K12 table
1142           *       ACAAC   TBLK12       ;  and add offset of table to it
1143           *       LUAA                 ;Get uncoded K12
1144           *       TAMD    K12V2        ;  and store it in RAM
1145           *-----
1146 02A9 42E1             BR      RTN
1147           * * * * *
1148           * The following code is executed if the K parameters need to
1149           * be zeroed out.  If  the new frame is a stop frame or a
1150           * silent frame, we zero out all K parameters and set the
1151           * energy to zero.  If the new frame is an unvoiced frame,
1152           * then we need to zero out the unused upper K parameters.
1153           * * * * *
1154           *
1155 02AB   2F ZeroKs      CLA
1156 02AC 6A20             TAMD    ENV2         ;Kill Energy
1157 02AE 6A26             TAMD    K1V2         ;Kill K1
1158 02B0 6A27             TAMD    K1V2+1
1159 02B2 6A2A             TAMD    K2V2         ;Kill K2
1160 02B4 6A2B             TAMD    K2V2+1
1161 02B6 6A2E             TAMD    K3V2         ;Kill K3
1162 02B8 6A30             TAMD    K4V2         ;Kill K4
1163 02BA   2F UNVC        CLA
1164 02BB 6A32             TAMD    K5V2         ;Kill K5
1165 02BD 6A34             TAMD    K6V2         ;Kill K6
1166 02BF 6A36             TAMD    K7V2         ;Kill K7
1167 02C1 6A38             TAMD    K8V2         ;Kill K8
1168 02C3 6A3A             TAMD    K9V2         ;Kill K9
1169 02C5 6A3C             TAMD    K10V2        ;Kill K10
1170           *       TAMD    K11V2            ;Kill K11
1171           *       TAMD    K12V2            ;Kill K12
1172 02C7 42E1             BR      RTN
1173           * * * * *
```

```
1174           * STOP AND RETURN
1175           * * * * *
1176           * The following code has two entry points. STOP is reached
1177           * if the current frame is a stop flag, it turns off
1178           * synthesis and returns to the program.  RTN is the general
1179           * exit point for the UPDATE routine, it sets the Update flag
1180           * and leaves the routine.
1181           * * * * *
1182 02C9 6242 STOP        TCX     MODE_BUF     ;Turn off synthesis, disable inter-
rupt
1183 02CB 657C            ANDCM   ~(LPC+INT1+UNV) ;  and go back to voiced
1184 02CD 6404            ORCM    PCM          ;Enable PCM mode
1185 02CF   11            TMA
1186 02D0   1D            TAMODE               ;Set mode per above setting
1187 02D1   2F            CLA
1188 02D2   1C            TASYN                ;Write a zero to the DAC
1189 02D3 6EFA            TCA     #FA
1190 02D5   3A BACK       IAC                  ;Wait for minimum of 30
1191 02D6 42DA            BR      out          ;instruction cycles
1192 02D8 42D5            BR      back
1193 02DA 6242 OUT        TCX     MODE_BUF     ;Disable PCM
1194 02DC 65FB            ANDCM   ~PCM
1195 02DE   11            TMA
1196 02DF   1D            TAMODE               ;Set mode per above setting
1197 02E0   3D            RETN                 ;Go back for next word
1198
1199 02E1   17 RTN        TTMA                 ;Get current timer
1200 02E2   1A            TAB                  ;and store it away
1201
1202 02E3   17 NOTDIFF    TTMA                 ;Wait for timer to decrement
1203 02E4   2D            SBAAN
1204 02E5 6000            ANEC    0
1205 02E7 42EB            BR      DIFF
1206 02E9 42E3            BR      NOTDIFF
1207
1208 02EB   17 DIFF       TTMA                 ;Subtract 128 from value
1209 02EC 7080            ACAAC   #80
1210 02EE   1E            TATM
1211
1212 02EF 6240            TCX     FLAGS        ;Set a flag indicating that
1213 02F1 6404            ORCM    Update_Flg   ;the parameters are update
1214
1215           *-------Return to current location------------*
1216
1217 02F3 6242            TCX     MODE_BUF     ;Get mode
1218 02F5 6602            TSTCM   LPC          ;Are we speaking yet?
1219 02F7 42FA            BR      RTN1         ;yes, reenable interrupt
1220 02F9   3D            RETN                 ;no, return for more data
1221
1222 02FA 6241 RTN1       TCX     FLAG1        ;Inhibit the execution of the
1223 02FC 6401            ORCM    Int_Off      ;pending interpolation interrupt
1224
1225 02FE 6242            TCX     MODE_BUF     ;Reenable the interrupt
1226 0300 6401            ORCM    INT1
1227 0302   11            TMA
1228 0303   1D            TAMODE
1229
1230 0304 6241            TCX     FLAG1        ;Reenable execution
1231 0306 65FE            ANDCM   ~Int_Off     ;of the interpolation routine
1232 0308 4090            BR      SPEAK_LP     ;Go back to loop
1233
1234           * * * * *
```

```
1235           * D6 SPEECH DECODING TABLES.
1236           * * * * *
1237           * Energy decoding table
1238           * * * * *
1239 030A 0001 TBLEN       BYTE    #00,#01,#02,#03,#04,#05,#07,#0B
1240 0312 111A             BYTE    #11,#1A,#29,#3F,#55,#70,#7F,#00
1241
1242           * * * * *
1243           * Pitch period decoding table
1244           * * * * *
1245 031A 0C00 TBLPH       BYTE    #0C,#00      ;0
1246 031C 1000             BYTE    #10,#00      ;1
1247 031E 1004             BYTE    #10,#04      ;2
1248 0320 1008             BYTE    #10,#08      ;3
1249 0322 1100             BYTE    #11,#00      ;4
1250 0324 1104             BYTE    #11,#04      ;5
1251 0326 1108             BYTE    #11,#08      ;6
1252 0328 110C             BYTE    #11,#0C      ;7
1253 032A 1204             BYTE    #12,#04      ;8
1254 032C 1208             BYTE    #12,#08      ;9
1255 032E 120C             BYTE    #12,#0C      ;10
1256 0330 1304             BYTE    #13,#04      ;1
1257 0332 1308             BYTE    #13,#08      ;2
1258 0334 1400             BYTE    #14,#00      ;3
1259 0336 1404             BYTE    #14,#04      ;4
1260 0338 140C             BYTE    #14,#0C      ;5
1261 033A 1500             BYTE    #15,#00      ;6
1262 033C 1508             BYTE    #15,#08      ;7
1263 033E 150C             BYTE    #15,#0C      ;8
1264 0340 1604             BYTE    #16,#04      ;9
1265 0342 160C             BYTE    #16,#0C      ;20
1266 0344 1700             BYTE    #17,#00      ;1
1267 0346 1708             BYTE    #17,#08      ;2
1268 0348 1800             BYTE    #18,#00      ;3
1269 034A 1804             BYTE    #18,#04      ;4
1270 034C 180C             BYTE    #18,#0C      ;5
1271 034E 1904             BYTE    #19,#04      ;6
1272 0350 190C             BYTE    #19,#0C      ;7
1273 0352 1A04             BYTE    #1A,#04      ;8
1274 0354 1A0C             BYTE    #1A,#0C      ;9
1275 0356 1B04             BYTE    #1B,#04      ;30
1276 0358 1B0C             BYTE    #1B,#0C      ;1
1277 035A 1C04             BYTE    #1C,#04      ;2
1278 035C 1C0C             BYTE    #1C,#0C      ;3
1279 035E 1D04             BYTE    #1D,#04      ;4
1280 0360 1D0C             BYTE    #1D,#0C      ;5
1281 0362 1E04             BYTE    #1E,#04      ;6
1282 0364 1F00             BYTE    #1F,#00      ;7
1283 0366 1F08             BYTE    #1F,#08      ;8
1284 0368 2000             BYTE    #20,#00      ;9
1285 036A 200C             BYTE    #20,#0C      ;40
1286 036C 2104             BYTE    #21,#04      ;1
1287 036E 210C             BYTE    #21,#0C      ;2
1288 0370 2208             BYTE    #22,#08      ;3
1289 0372 2300             BYTE    #23,#00      ;4
1290 0374 230C             BYTE    #23,#0C      ;5
1291 0376 2408             BYTE    #24,#08      ;6
1292 0378 2500             BYTE    #25,#00      ;7
1293 037A 250C             BYTE    #25,#0C      ;8
1294 037C 2608             BYTE    #26,#08      ;9
1295 037E 2704             BYTE    #27,#04      ;50
1296 0380 2800             BYTE    #28,#00      ;1
```

```
1297 0382 280C            BYTE    #28,#0C      ;2
1298 0384 2908            BYTE    #29,#08      ;3
1299 0386 2A04            BYTE    #2A,#04      ;4
1300 0388 2B00            BYTE    #2B,#00      ;5
1301 038A 2B0C            BYTE    #2B,#0C      ;6
1302 038C 2C08            BYTE    #2C,#08      ;7
1303 038E 2D04            BYTE    #2D,#04      ;8
1304 0390 2E04            BYTE    #2E,#04      ;9
1305 0392 2F00            BYTE    #2F,#00      ;60
1306 0394 3000            BYTE    #30,#00      ;1
1307 0396 300C            BYTE    #30,#0C      ;2
1308 0398 310C            BYTE    #31,#0C      ;3
1309 039A 3208            BYTE    #32,#08      ;4
1310 039C 3308            BYTE    #33,#08      ;5
1311 039E 3408            BYTE    #34,#08      ;6
1312 03A0 3508            BYTE    #35,#08      ;7
1313 03A2 3608            BYTE    #36,#08      ;8
1314 03A4 3708            BYTE    #37,#08      ;9
1315 03A6 3808            BYTE    #38,#08      ;70
1316 03A8 3908            BYTE    #39,#08      ;1
1317 03AA 3A08            BYTE    #3A,#08      ;2
1318 03AC 3B0C            BYTE    #3B,#0C      ;3
1319 03AE 3C0C            BYTE    #3C,#0C      ;4
1320 03B0 3D0C            BYTE    #3D,#0C      ;5
1321 03B2 3F00            BYTE    #3F,#00      ;6
1322 03B4 4004            BYTE    #40,#04      ;7
1323 03B6 4104            BYTE    #41,#04      ;8
1324 03B8 4208            BYTE    #42,#08      ;9
1325 03BA 430C            BYTE    #43,#0C      ;80
1326 03BC 4500            BYTE    #45,#00      ;1
1327 03BE 4604            BYTE    #46,#04      ;2
1328 03C0 4708            BYTE    #47,#08      ;3
1329 03C2 4900            BYTE    #49,#00      ;4
1330 03C4 4A04            BYTE    #4A,#04      ;5
1331 03C6 4B0C            BYTE    #4B,#0C      ;6
1332 03C8 4D00            BYTE    #4D,#00      ;7
1333 03CA 4E08            BYTE    #4E,#08      ;8
1334 03CC 5000            BYTE    #50,#00      ;9
1335 03CE 5104            BYTE    #51,#04      ;90
1336 03D0 520C            BYTE    #52,#0C      ;1
1337 03D2 5408            BYTE    #54,#08      ;2
1338 03D4 5600            BYTE    #56,#00      ;3
1339 03D6 5708            BYTE    #57,#08      ;4
1340 03D8 5904            BYTE    #59,#04      ;5
1341 03DA 5A0C            BYTE    #5A,#0C      ;6
1342 03DC 5C08            BYTE    #5C,#08      ;7
1343 03DE 5E04            BYTE    #5E,#04      ;8
1344 03E0 6000            BYTE    #60,#00      ;9
1345 03E2 610C            BYTE    #61,#0C      ;100
1346 03E4 6308            BYTE    #63,#08      ;1
1347 03E6 6504            BYTE    #65,#04      ;2
1348 03E8 6704            BYTE    #67,#04      ;3
1349 03EA 6900            BYTE    #69,#00      ;4
1350 03EC 6B00            BYTE    #6B,#00      ;5
1351 03EE 6D00            BYTE    #6D,#00      ;6
1352 03F0 6F00            BYTE    #6F,#00      ;7
1353 03F2 7100            BYTE    #71,#00      ;8
1354 03F4 7304            BYTE    #73,#04      ;9
1355 03F6 7504            BYTE    #75,#04      ;110
1356 03F8 7708            BYTE    #77,#08      ;111
1357 03FA 790C            BYTE    #79,#0C      ;112
1358 03FC 7C00            BYTE    #7C,#00      ;113
```

```
1359 03FE 7E04            BYTE    #7E,#04      ;114
1360 0400 8008            BYTE    #80,#08      ;115
1361 0402 820C            BYTE    #82,#0C      ;116
1362 0404 8504            BYTE    #85,#04      ;117
1363 0406 870C            BYTE    #87,#0C      ;118
1364 0408 8A04            BYTE    #8A,#04      ;119
1365 040A 8C0C            BYTE    #8C,#0C      ;120
1366 040C 8F08            BYTE    #8F,#08      ;121
1367 040E 9200            BYTE    #92,#00      ;122
1368 0410 940C            BYTE    #94,#0C      ;123
1369 0412 9708            BYTE    #97,#08      ;124
1370 0414 9A04            BYTE    #9A,#04      ;125
1371 0416 9D00            BYTE    #9D,#00      ;126
1372 0418 A000            BYTE    #A0,#00      ;127
1373
1374           * * * * *
1375           * K1 parameter decoding table
1376           * * * * *
1377 041A 8100 TBLK1       BYTE    #81,#00
1378 041C 8204            BYTE    #82,#04
1379 041E 8304            BYTE    #83,#04
1380 0420 8408            BYTE    #84,#08
1381 0422 850C            BYTE    #85,#0C
1382 0424 8700            BYTE    #87,#00
1383 0426 8804            BYTE    #88,#04
1384 0428 890C            BYTE    #89,#0C
1385 042A 8B04            BYTE    #8B,#04
1386 042C 8C0C            BYTE    #8C,#0C
1387 042E 8E04            BYTE    #8E,#04
1388 0430 9000            BYTE    #90,#00
1389 0432 910C            BYTE    #91,#0C
1390 0434 9308            BYTE    #93,#08
1391 0436 9508            BYTE    #95,#08
1392 0438 9704            BYTE    #97,#04
1393 043A 9908            BYTE    #99,#08
1394 043C 9B08            BYTE    #9B,#08
1395 043E 9D08            BYTE    #9D,#08
1396 0440 9F0C            BYTE    #9F,#0C
1397 0442 A200            BYTE    #A2,#00
1398 0444 A404            BYTE    #A4,#04
1399 0446 A60C            BYTE    #A6,#0C
1400 0448 A904            BYTE    #A9,#04
1401 044A AB08            BYTE    #AB,#08
1402 044C AE00            BYTE    #AE,#00
1403 044E B00C            BYTE    #B0,#0C
1404 0450 B308            BYTE    #B3,#08
1405 0452 B604            BYTE    #B6,#04
1406 0454 B900            BYTE    #B9,#00
1407 0456 BC00            BYTE    #BC,#00
1408 0458 BF04            BYTE    #BF,#04
1409 045A C204            BYTE    #C2,#04
1410 045C C508            BYTE    #C5,#08
1411 045E C80C            BYTE    #C8,#0C
1412 0460 CC04            BYTE    #CC,#04
1413 0462 CF0C            BYTE    #CF,#0C
1414 0464 D308            BYTE    #D3,#08
1415 0466 D708            BYTE    #D7,#08
1416 0468 DB04            BYTE    #DB,#04
1417 046A DF04            BYTE    #DF,#04
1418 046C E308            BYTE    #E3,#08
1419 046E E70C            BYTE    #E7,#0C
1420 0470 EC00            BYTE    #EC,#00
```

```
1421 0472 F004             BYTE    #F0,#04
1422 0474 F40C             BYTE    #F4,#0C
1423 0476 F90C             BYTE    #F9,#0C
1424 0478 FE0C             BYTE    #FE,#0C
1425 047A 0404             BYTE    #04,#04
1426 047C 090C             BYTE    #09,#0C
1427 047E 0F04             BYTE    #0F,#04
1428 0480 1508             BYTE    #15,#08
1429 0482 1C08             BYTE    #1C,#08
1430 0484 2308             BYTE    #23,#08
1431 0486 2A0C             BYTE    #2A,#0C
1432 0488 3208             BYTE    #32,#08
1433 048A 3A08             BYTE    #3A,#08
1434 048C 420C             BYTE    #42,#0C
1435 048E 4B08             BYTE    #4B,#08
1436 0490 5400             BYTE    #54,#00
1437 0492 5C04             BYTE    #5C,#04
1438 0494 6500             BYTE    #65,#00
1439 0496 6E00             BYTE    #6E,#00
1440 0498 7808             BYTE    #78,#08
1441
1442           * * * * *
1443           * K2 parameter decoding table
1444           * * * * *
1445 049A 8A00 TBLK2       BYTE    #8A,#00
1446 049C 9800             BYTE    #98,#00
1447 049E A30C             BYTE    #A3,#0C
1448 04A0 AD0C             BYTE    #AD,#0C
1449 04A2 B408             BYTE    #B4,#08
1450 04A4 BA08             BYTE    #BA,#08
1451 04A6 C000             BYTE    #C0,#00
1452 04A8 C500             BYTE    #C5,#00
1453 04AA C90C             BYTE    #C9,#0C
1454 04AC CE04             BYTE    #CE,#04
1455 04AE D20C             BYTE    #D2,#0C
1456 04B0 D60C             BYTE    #D6,#0C
1457 04B2 DA0C             BYTE    #DA,#0C
1458 04B4 DE08             BYTE    #DE,#08
1459 04B6 E200             BYTE    #E2,#00
1460 04B8 E50C             BYTE    #E5,#0C
1461 04BA E904             BYTE    #E9,#04
1462 04BC EC0C             BYTE    #EC,#0C
1463 04BE F000             BYTE    #F0,#00
1464 04C0 F304             BYTE    #F3,#04
1465 04C2 F608             BYTE    #F6,#08
1466 04C4 F90C             BYTE    #F9,#0C
1467 04C6 FD00             BYTE    #FD,#00
1468 04C8 0000             BYTE    #00,#00
1469 04CA 0304             BYTE    #03,#04
1470 04CC 0604             BYTE    #06,#04
1471 04CE 0904             BYTE    #09,#04
1472 04D0 0C04             BYTE    #0C,#04
1473 04D2 0F04             BYTE    #0F,#04
1474 04D4 1208             BYTE    #12,#08
1475 04D6 1508             BYTE    #15,#08
1476 04D8 1808             BYTE    #18,#08
1477 04DA 1B08             BYTE    #1B,#08
1478 04DC 1E08             BYTE    #1E,#08
1479 04DE 2108             BYTE    #21,#08
1480 04E0 240C             BYTE    #24,#0C
1481 04E2 270C             BYTE    #27,#0C
1482 04E4 2A0C             BYTE    #2A,#0C
```

```
1483 04E6 2D0C             BYTE    #2D,#0C
1484 04E8 300C             BYTE    #30,#0C
1485 04EA 3400             BYTE    #34,#00
1486 04EC 3700             BYTE    #37,#00
1487 04EE 3A04             BYTE    #3A,#04
1488 04F0 3D00             BYTE    #3D,#00
1489 04F2 4000             BYTE    #40,#00
1490 04F4 4300             BYTE    #43,#00
1491 04F6 4600             BYTE    #46,#00
1492 04F8 4900             BYTE    #49,#00
1493 04FA 4C00             BYTE    #4C,#00
1494 04FC 4F04             BYTE    #4F,#04
1495 04FE 5204             BYTE    #52,#04
1496 0500 5504             BYTE    #55,#04
1497 0502 5804             BYTE    #58,#04
1498 0504 5B04             BYTE    #5B,#04
1499 0506 5E00             BYTE    #5E,#00
1500 0508 6100             BYTE    #61,#00
1501 050A 630C             BYTE    #63,#0C
1502 050C 6608             BYTE    #66,#08
1503 050E 6904             BYTE    #69,#04
1504 0510 6C00             BYTE    #6C,#00
1505 0512 6F00             BYTE    #6F,#00
1506 0514 7200             BYTE    #72,#00
1507 0516 7604             BYTE    #76,#04
1508 0518 7C00             BYTE    #7C,#00
1509
1510           * * * * *
1511           * K3 parameter decoding table
1512           * * * * *
1513 051A 8B9A TBLK3       BYTE    #8B,#9A,#A2,#A9,#AF,#B5,#BB,#C0
1514 0522 C5CA             BYTE    #C5,#CA,#CF,#D4,#D9,#DE,#E2,#E7
1515 052A ECF1             BYTE    #EC,#F1,#F6,#FB,#01,#07,#0D,#14
1516 0532 1A22             BYTE    #1A,#22,#29,#32,#3B,#45,#53,#6D
1517
1518           * * * * *
1519           * K4 parameter decoding table
1520           * * * * *
1521 053A 94B0 TBLK4       BYTE    #94,#B0,#C2,#CB,#D3,#D9,#DF,#E5
1522 0542 EAEF             BYTE    #EA,#EF,#F4,#F9,#FE,#03,#07,#0C
1523 054A 1115             BYTE    #11,#15,#1A,#1F,#24,#29,#2E,#33
1524 0552 383E             BYTE    #38,#3E,#44,#4B,#53,#5A,#64,#74
1525
1526           * * * * *
1527           * K5 parameter decoding table
1528           * * * * *
1529 055A A3C5 TBLK5       BYTE    #A3,#C5,#D4,#E0,#EA,#F3,#FC,#04
1530 0562 0C15             BYTE    #0C,#15,#1E,#27,#31,#3D,#4C,#66
1531
1532           * * * * *
1533           * K6 parameter decoding table
1534           * * * * *
1535 056A AAD7 TBLK6       BYTE    #AA,#D7,#E7,#F2,#FC,#05,#0D,#14
1536 0572 1C24             BYTE    #1C,#24,#2D,#36,#40,#4A,#55,#6A
1537
1538           * * * * *
1539           * K7 parameter decoding table
1540           * * * * *
1541 057A A3C8 TBLK7       BYTE    #A3,#C8,#D7,#E3,#ED,#F5,#FD,#05
1542 0582 0D14             BYTE    #0D,#14,#1D,#26,#31,#3C,#4B,#67
1543
1544           * * * * *
```

```
1545           * K8 parameter decoding table
1546           * * * * *
1547 058A C5E4 TBLK8       BYTE    #C5,#E4,#F6,#05,#14,#27,#3E,#58
1548
1549           * * * * *
1550           * K9 parameter decoding table
1551           * * * * *
1552 0592 B9DC TBLK9       BYTE    #B9,#DC,#EC,#F9,#04,#10,#1F,#45
1553
1554           * * * * *
1555           * K10 parameter decoding table
1556           * * * * *
1557 059A C3E6 TBLK10      BYTE    #C3,#E6,#F3,#FD,#06,#11,#1E,#43
1558
1559
1560           ************************************************************
1561           *                                                          *
1562           *    This is the lookup table for the speech stored at     *
1563           *    voc.                                                  *
1564           *                                                          *
1565           ************************************************************
1566 05A2 0000 SPEECH      DATA    #0000
1567 05A4 05B0            DATA    #0000+VOC    ;Word 1       "One"
1568 05A6 0634            DATA    #0084+VOC    ;Word 2       "Two"
1569 05A8 06A6            DATA    #00F6+VOC    ;Word 3       "Three"
1570 05AA 072A            DATA    #017A+VOC    ;Word 4       "Four"
1571 05AC 0790            DATA    #01E0+VOC    ;Word 5       "Five"
1572 05AE 083C            DATA    #028C+VOC    ;Word 6       "Six"
1573           ************************************************************
1574           *                                                          *
1575           *    This is the DTS speech coded with the D6 coding       *
1576           *    table                                                 *
1577           *                                                          *
1578           ************************************************************
1579           VOC
1580 05B0 6889            BYTE    #68,#89,#84,#FB,#1A,#53,#64,#B2
1581 05B8 8487            BYTE    #84,#87,#33,#C9,#35,#28,#9B,#A1
1582 05C0 D1BA            BYTE    #D1,#BA,#22,#3A,#94,#8D,#08,#BD
1583 05C8 BE40            BYTE    #BE,#40,#1C,#6D,#BA,#BC,#14,#7E
1584 05D0 33CE            BYTE    #33,#CE,#4E,#75,#8D,#EE,#2F,#03
1585 05D8 BB96            BYTE    #BB,#96,#4A,#46,#D7,#CF,#4A,#DD
1586 05E0 4A23            BYTE    #4A,#23,#54,#CE,#26,#B7,#74,#A5
1587 05E8 9B49            BYTE    #9B,#49,#7B,#62,#44,#B7,#32,#2D
1588 05F0 95D9            BYTE    #95,#D9,#C8,#B4,#5B,#9A,#35,#5A
1589 05F8 8DC2            BYTE    #8D,#C2,#DC,#2C,#CC,#5A,#CC,#0A
1590 0600 2B6E            BYTE    #2B,#6E,#EE,#66,#19,#69,#98,#27
1591 0608 7533            BYTE    #75,#33,#CB,#80,#36,#AC,#94,#E6
1592 0610 A985            BYTE    #A9,#85,#CE,#4B,#1B,#EC,#CD,#D4
1593 0618 2C50            BYTE    #2C,#50,#71,#52,#F5,#76,#AA,#1B
1594 0620 9B38            BYTE    #9B,#38,#98,#58,#33,#56,#B6,#35
1595 0628 D258            BYTE    #D2,#58,#A3,#99,#C8,#7B,#AE,#D5
1596 0630 A85E            BYTE    #A8,#5E,#FB,#01,#04,#B0,#78,#BA
1597 0638 2BC0            BYTE    #2B,#C0,#5D,#1B,#6D,#00,#F7,#65
1598 0640 BA01            BYTE    #BA,#01,#64,#BA,#13,#29,#B7,#06
1599 0648 3681            BYTE    #36,#81,#C9,#FE,#92,#DB,#5C,#15
1600 0650 20B8            BYTE    #20,#B8,#7F,#29,#AF,#8A,#CA,#10
1601 0658 DC3F            BYTE    #DC,#3F,#35,#12,#56,#47,#2A,#FA
1602 0660 9FFA            BYTE    #9F,#FA,#26,#61,#97,#0C,#ED,#77
1603 0668 439A            BYTE    #43,#9A,#6E,#97,#9A,#F7,#8A,#01
1604 0670 2ECE            BYTE    #2E,#CE,#8D,#29,#7B,#48,#17,#B1
1605 0678 CF86            BYTE    #CF,#86,#B4,#4E,#64,#04,#47,#77
1606 0680 A14B            BYTE    #A1,#4B,#26,#32,#83,#9B,#13,#31
```

```
1607 0688 AD23            BYTE    #AD,#23,#59,#E3,#DA,#5E,#90,#B2
1608 0690 85AC            BYTE    #85,#AC,#68,#65,#0D,#70,#E9,#4D
1609 0698 3644            BYTE    #36,#44,#38,#13,#87,#74,#12,#BB
1610 06A0 8D52            BYTE    #8D,#52,#59,#90,#E4,#3D,#08,#60
1611 06A8 CA86            BYTE    #CA,#86,#13,#40,#66,#1A,#46,#00
1612 06B0 B9EC            BYTE    #B9,#EC,#8B,#00,#14,#59,#B7,#0A
1613 06B8 905A            BYTE    #90,#5A,#35,#9A,#EC,#1E,#D9,#86
1614 06C0 A4EA            BYTE    #A4,#EA,#5C,#41,#69,#85,#B2,#A6
1615 06C8 EE21            BYTE    #EE,#21,#AF,#CC,#24,#46,#63,#F7
1616 06D0 9453            BYTE    #94,#53,#26,#E1,#65,#B1,#7B,#C9
1617 06D8 3BA5            BYTE    #3B,#A5,#77,#B8,#92,#3E,#E5,#9B
1618 06E0 B47B            BYTE    #B4,#7B,#18,#EE,#9F,#0A,#5B,#52
1619 06E8 02B4            BYTE    #02,#B4,#EE,#4F,#8D,#23,#CF,#06
1620 06F0 2AB7            BYTE    #2A,#B7,#A7,#FE,#96,#04,#0A,#DD
1621 06F8 DFD2            BYTE    #DF,#D2,#70,#B6,#24,#C6,#9D,#25
1622 0700 613C            BYTE    #61,#3C,#F0,#1C,#F3,#ED,#A4,#30
1623 0708 5974            BYTE    #59,#74,#8E,#70,#E7,#96,#9B,#4C
1624 0710 0A47            BYTE    #0A,#47,#74,#3B,#D1,#CC,#07,#95
1625 0718 21BE            BYTE    #21,#BE,#19,#65,#A6,#B3,#27,#20
1626 0720 CE4C            BYTE    #CE,#4C,#62,#93,#58,#41,#B4,#77
1627 0728 0A3E            BYTE    #0A,#3E,#80,#00,#A6,#6A,#03,#01
1628 0730 54A6            BYTE    #54,#A6,#4F,#0C,#10,#C6,#D1,#0B
1629 0738 8097            BYTE    #80,#97,#D4,#E0,#12,#2A,#D7,#37
1630 0740 8758            BYTE    #87,#58,#09,#E9,#18,#B7,#3F,#0D
1631 0748 BD87            BYTE    #BD,#87,#74,#8A,#99,#9F,#86,#DE
1632 0750 43D9            BYTE    #43,#D9,#26,#EA,#37,#C5,#EC,#A1
1633 0758 A9B0            BYTE    #A9,#B0,#F3,#91,#71,#FE,#30,#60
1634 0760 83B3            BYTE    #83,#B3,#B1,#C4,#7F,#1A,#B3,#ED
1635 0768 8ED4            BYTE    #8E,#D4,#A2,#3F,#CC,#84,#AD,#4A
1636 0770 1BE8            BYTE    #1B,#E8,#1F,#D6,#EA,#38,#A4,#1C
1637 0778 E60F            BYTE    #E6,#0F,#5B,#63,#49,#D4,#0F,#F3
1638 0780 B983            BYTE    #B9,#83,#B1,#7B,#E2,#87,#7B,#DD
1639 0788 D5BA            BYTE    #D5,#BA,#A8,#E8,#C5,#5D,#0F,#00
1640 0790 0890            BYTE    #08,#90,#FB,#51,#23,#80,#AB,#19
1641 0798 4A00            BYTE    #4A,#00,#B9,#97,#0D,#01,#34,#59
1642 07A0 490C            BYTE    #49,#0C,#D0,#A5,#29,#11,#80,#E5
1643 07A8 8658            BYTE    #86,#58,#EA,#BE,#32,#36,#27,#F5
1644 07B0 69B5            BYTE    #69,#B5,#4C,#18,#CB,#9B,#DA,#B5
1645 07B8 7AAA            BYTE    #7A,#AA,#EC,#61,#45,#6B,#4B,#33
1646 07C0 F06F            BYTE    #F0,#6F,#D1,#94,#25,#A5,#ED,#15
1647 07C8 3768            BYTE    #37,#68,#EA,#9C,#D4,#75,#BA,#ED
1648 07D0 346D            BYTE    #34,#6D,#4E,#19,#7B,#CD,#76,#9A
1649 07D8 7ABB            BYTE    #7A,#BB,#CC,#A2,#F2,#18,#4D,#B9
1650 07E0 5996            BYTE    #59,#96,#59,#71,#B4,#A4,#3C,#2A
1651 07E8 CBBC            BYTE    #CB,#BC,#5A,#5C,#52,#67,#A6,#4D
1652 07F0 3636            BYTE    #36,#36,#AA,#61,#17,#D3,#2E,#6F
1653 07F8 2293            BYTE    #22,#93,#F4,#05,#61,#1F,#56,#52
1654 0800 69E7            BYTE    #69,#E7,#41,#B3,#0F,#32,#E1,#AC
1655 0808 E2B0            BYTE    #E2,#B0,#D9,#EB,#95,#34,#5C,#7E
1656 0810 52EC            BYTE    #52,#EC,#E5,#44,#1B,#4A,#79,#C1
1657 0818 F63A            BYTE    #F6,#3A,#6D,#1C,#9A,#76,#66,#BB
1658 0820 5132            BYTE    #51,#32,#16,#89,#94,#99,#DD,#96
1659 0828 8F69            BYTE    #8F,#69,#C9,#6A,#D5,#6E,#F2,#52
1660 0830 2162            BYTE    #21,#62,#6A,#62,#37,#24,#2D,#22
1661 0838 1197            BYTE    #11,#97,#07,#00,#04,#F0,#2A,#08
1662 0840 13C0            BYTE    #13,#C0,#BF,#F9,#44,#00,#FF,#EE
1663 0848 9500            BYTE    #95,#00,#7C,#A5,#D3,#02,#F0,#B5
1664 0850 DA94            BYTE    #DA,#94,#62,#C6,#17,#8D,#D9,#B7
1665 0858 4BBE            BYTE    #4B,#BE,#97,#8B,#25,#CB,#D7,#A5
1666 0860 5AAA            BYTE    #5A,#AA,#4D,#72,#F7,#DB,#D4,#2F
1667 0868 BD4C            BYTE    #BD,#4C,#75,#EA,#6B,#5A,#84,#15
1668 0870 D1DD            BYTE    #D1,#DD,#BD,#11,#00,#80,#01,#1C
```

```
1669 0878 6F6B             BYTE    #6F,#6B,#01,#78,#AC,#BE,#05,#E0
1670 0880 5F75             BYTE    #5F,#75,#62,#80,#7F,#D0,#9D,#01
1671 0888 BE8F             BYTE    #BE,#8F,#7B,#02,#78,#3B,#5D,#1E
1672 0890 08F0             BYTE    #08,#F0,#15,#3E,#13,#C0,#57,#F3
1673 0898 4C00             BYTE    #4C,#00,#7F,#CF,#38,#01,#FC,#81
1674 08A0 320C             BYTE    #32,#0C,#F0,#5F,#C2,#85,#62,#C5
1675 08A8 0D85             BYTE    #0D,#85,#59,#9B,#5A,#25,#D5,#87
1676 08B0 A4AA             BYTE    #A4,#AA,#67,#A5,#5A,#04,#5B,#62
1677 08B8 D7DC             BYTE    #D7,#DC,#52,#4B,#9A,#C9,#A9,#F2
1678 08C0 49E9             BYTE    #49,#E9,#46,#6D,#37,#94,#FE,#C4
1679 08C8 8C75             BYTE    #8C,#75,#B3,#58,#52,#CB,#64,#A6
1680 08D0 2C53             BYTE    #2C,#53,#23,#47,#A6,#35,#6B,#DE
1681 08D8 C89A             BYTE    #C8,#9A,#23,#6B,#A5,#55,#E0,#36
1682 08E0 C91A             BYTE    #C9,#1A,#B7,#D2,#3E,#0E,#26,#67
1683 08E8 8D4B             BYTE    #8D,#4B,#66,#AF,#26,#99,#BB,#D5
1684 08F0 40B5             BYTE    #40,#B5,#97,#2D,#36,#95,#3A,#E6
1685 08F8 0300             BYTE    #03,#00,#A6,#2A,#5A,#BE,#D6,#45
1686 0900 E850             BYTE    #E8,#50,#C9,#5C,#A9,#EC,#7A,#76
1687 0908 A98C             BYTE    #A9,#8C,#91,#65,#B8,#FD,#B6,#54
1688 0910 D63C             BYTE    #D6,#3C,#52,#AC,#D9,#5A,#8A,#9B
1689 0918 E911             BYTE    #E9,#11,#6D,#3F,#2D,#E5,#29,#96
1690 0920 50AE             BYTE    #50,#AE,#E7,#A6,#FE,#92,#2B,#28
1691 0928 75AB             BYTE    #75,#AB,#DD,#A6,#8F,#29,#D4,#D9
1692 0930 5900             BYTE    #59,#00,#0C,#B0,#08,#D4,#0A,#C0
1693 0938 13E6             BYTE    #13,#E6,#AE,#00,#6B,#7D,#9B,#02
1694 0940 8CE5             BYTE    #8C,#E5,#32,#0F,#A4,#25,#53,#73
1695 0948 5750             BYTE    #57,#50,#53,#D1,#93,#C5,#3C,#5B
1696 0950 6599             BYTE    #65,#99,#18,#CA,#7C,#99,#65,#BC
1697 0958 CE8D             BYTE    #CE,#8D,#65,#4A,#0F,#4D,#9D,#53
1698 0960 C69E             BYTE    #C6,#9E,#D3,#1C,#65,#4E,#2C,#23
1699 0968 3F3B             BYTE    #3F,#3B,#52,#D2,#4F,#95,#9E,#9F
1700 0970 1D29             BYTE    #1D,#29,#E9,#A7,#4A,#37,#B7,#4F
1701 0978 A5B2             BYTE    #A5,#B2,#35,#A5,#9B,#DB,#A7,#52
1702 0980 D99A             BYTE    #D9,#9A,#D2,#C9,#93,#93,#A8,#74
1703 0988 4DE9             BYTE    #4D,#E9,#96,#D9,#F2,#54,#B2,#BA
1704 0990 8CF2             BYTE    #8C,#F2,#BA,#09,#69,#9D,#59,#46
1705 0998 651E             BYTE    #65,#1E,#99,#96,#56,#4C,#A3,#38
1706 09A0 54CC             BYTE    #54,#CC,#5B,#0B,#B9,#91,#AD,#1B
1707 09A8 9EC5             BYTE    #9E,#C5,#45,#D9,#18,#73,#C2,#0C
1708
1709           *EXCITATION FUNCTION
1710
1711 8000                  AORG    #8000
1712
1713      0144 BYTE  #01,#44,#01,#5E,#01,#74,#01,#84,#01,#8E,#01,#92,#01,#94,#01,#8C
1714 8010 0184 BYTE  #01,#84,#01,#78,#01,#6A,#01,#5A,#01,#4A,#01,#3C,#01,#34,#01,#2A
1715 8020 012A BYTE  #01,#2A,#01,#30,#01,#3E,#01,#50,#01,#70,#01,#94,#01,#C6,#01,#FC
1716 8030 023E BYTE  #02,#3E,#02,#82,#02,#D2,#03,#22,#03,#7A,#03,#D0,#04,#2C,#04,#80
1717 8040 04D8 BYTE  #04,#D8,#05,#24,#05,#72,#05,#B2,#05,#F0,#06,#1E,#06,#4A,#06,#64
1718 8050 067E BYTE  #06,#7E,#06,#86,#06,#8E,#06,#8A,#06,#8A,#06,#7E,#06,#7A,#06,#74
1719 8060 067A BYTE  #06,#7A,#06,#82,#06,#9C,#06,#BE,#06,#F6,#07,#40,#07,#A4,#08,#1A
1720 8070 08AE BYTE  #08,#AE,#09,#5A,#0A,#22,#0B,#04,#0C,#00,#0D,#14,#0E,#3E,#0F,#7A
1721 8080 10C8 BYTE  #10,#C8,#12,#22,#13,#82,#14,#E8,#16,#4C,#17,#AA,#18,#FE,#1A,#40
1722 8090 1B6E BYTE  #1B,#6E,#1C,#80,#1D,#76,#1E,#48,#1E,#F4,#1F,#78,#1F,#D2,#1F,#FE
1723 80A0 1FFE BYTE  #1F,#FE,#1F,#D2,#1F,#78,#1E,#F4,#1E,#48,#1D,#76,#1C,#80,#1B,#6E
1724 80B0 1A40 BYTE  #1A,#40,#18,#FE,#17,#AA,#16,#4C,#14,#E8,#13,#82,#12,#22,#10,#C8
1725 80C0 0F7A BYTE  #0F,#7A,#0E,#3E,#0D,#14,#0C,#00,#0B,#04,#0A,#22,#09,#5A,#08,#AE
1726 80D0 081A BYTE  #08,#1A,#07,#A4,#07,#40,#06,#F6,#06,#BE,#06,#9C,#06,#82,#06,#7A
1727 80E0 0674 BYTE  #06,#74,#06,#7A,#06,#7E,#06,#8A,#06,#8A,#06,#8E,#06,#86,#06,#7E
1728 80F0 0664 BYTE  #06,#64,#06,#4A,#06,#1E,#05,#F0,#05,#B2,#05,#72,#05,#24,#04,#D8
1729 8100 0480 BYTE  #04,#80,#04,#2C,#03,#D0,#03,#7A,#03,#22,#02,#D2,#02,#82,#02,#3E
1730 8110 01FC BYTE  #01,#FC,#01,#C6,#01,#94,#01,#70,#01,#50,#01,#3E,#01,#30,#01,#2A
```

```
1731 8120 012A BYTE  #01,#2A,#01,#34,#01,#3C,#01,#4A,#01,#5A,#01,#6A,#01,#78,#01,#84
1732 8130 018C BYTE  #01,#8C,#01,#94,#01,#92,#01,#8E,#01,#84,#01,#74,#01,#5E,#01,#44
1733 8140 0B00 BYTE  #0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00
1734 8150 0B00 BYTE  #0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00,#0B,#00
1735 8160 F500 BYTE  #F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00
1736 8170 F500 BYTE  #F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00,#F5,#00
1737 8180 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1738 8190 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1739 81A0 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1740 81B0 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1741 81C0 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1742 81D0 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1743 81E0 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1744 81F0 FFFF BYTE  #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1745 8200   FF BYTE  #FF
1746
```

Appendix E

MSP50C3x Family Data Sheet

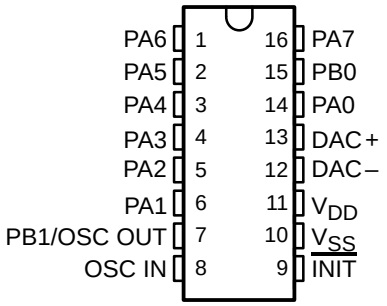
Topic Page

E.1 MSP50C3x Family Data Sheet

This appendix contains data sheet information for the MSP50C3x mixed-signal processor family.

- **Dual Programmable LPC-12 Speech Synthesizers**
- **Simultaneous LPC and PCM Waveforms**
- **8-Bit Microprocessor with 61 instructions**
- **32 Twelve-Bit Words and 224 Bytes of RAM**
- **3.3V to 6.5V CMOS Technology for Low Power Dissipation**
- **Direct Speaker Drive Capability**
- **Mask Selectable Internal or External Clock**
- **Internal Clock Generator that Requires No External Components**
- **Two Software-Selectable Clock Speeds**
- **10-kHz or 8-kHz Speech Sample Rate**

N PACKAGE
(TOP VIEW)



description

The MSP50x3x family uses a revolutionary architecture to combine an 8-bit microprocessor, two speech synthesizers, ROM, RAM, and I/O in a low-cost single-chip system. The architecture uses the same arithmetic logic unit (ALU) for the two synthesizers and the microprocessor, thus reducing chip area and cost and enabling the microprocessor to do a multiply operation in 0.8 μs. The MSP50x3x family features two independent channels of linear predictive coding (LPC), which synthesize high-quality speech at a low data rate. Pulse-code modulation (PCM) can produce music or sound effects. LPC and PCM can be added together to produce a composite result. For more information, see the MSP50x3x User's Guide (literature number SPSU006).

Table 1. MSP50x3x Family

DEVICE	AMOUNT OF ROM/PROM	FEATURES
MSP50C32	16K bytes mask ROM	9/10 I/O lines
MSP50C33	32K bytes mask ROM	9/10 I/O lines
MSP50C34	64K bytes mask ROM	9/10 I/O lines, 24 I/O lines in die form
MSP50P34	64K bytes PROM	9/10 I/O lines
MSP50C37	16K bytes mask ROM	18 I/O lines, A/D converter/analog amplifier
MSP50P37	16K bytes PROM	18 I/O lines, A/D converter/analog amplifier

absolute maximum ratings over operating free-air temperature range†

Supply voltage range, V_{DD} (see Note 1) . . . . . –0.3 V to 8 V
Supply current, I_{DD} or I_{SS} (see Note 2) . . . . . 100 mA
Input voltage range, V_I (see Note 1) . . . . . –0.3 V to V_{DD} + 0.3 V
Output voltage range, V_O (see Note 1) . . . . . –0.3 V to V_{DD} + 0.3 V
Storage temperature range . . . . . –30°C to 125°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. All voltages are with respect to ground.
2. The total supply current includes the current out of all the I/O terminals and DAC terminals as well as the operating current of the device.

recommended operating conditions (MSP50C32, MSP50C33, MSP50x34)

			MIN	MAX	UNIT
V_{DD}	Supply voltage†		3.3	6.5	V
V_{IH}	High-level input voltage	V_{DD} = 3.3 V	2.5	3.3	V
		V_{DD} = 5 V	3.8	5	
		V_{DD} = 6 V	4.5	6	
V_{IL}	Low-level input voltage	V_{DD} = 3.3 V	0	0.65	V
		V_{DD} = 5 V	0	1	
		V_{DD} = 6 V	0	1.3	
T_A	Operating free-air temperature	Device functionality	0	70	°C
Rspeaker	Minimum speaker impedance	Direct speaker drive using 2 pin push-pull DAC option	32		Ω

† Unless otherwise noted, all voltages are with respect to V_{SS}.

recommended operating conditions (MSP50x37)

			MIN	MAX	UNIT
V_{DD}	Supply voltage†		4	6.5	V
V_{IH}	High-level input voltage	V_{DD} = 4 V	3	4	V
		V_{DD} = 5 V	3.8	5	
		V_{DD} = 6 V	4.5	6	
V_{IL}	Low-level input voltage	V_{DD} = 4 V	0	1	V
		V_{DD} = 5 V	0	1.2	
		V_{DD} = 6 V	0	1.5	
	MUX input voltage	Reference voltage = 6.5 V	0	6.5	V
T_A	Operating free-air temperature	Device functionality	–10	70	°C
Rspeaker	Minimum speaker impedance	Direct speaker drive using power amp	8		Ω

MSP50C32, MSP50C33, MSP50x34 electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{T+}	Positive-going threshold voltage (INIT)	V_{DD} = 3.5 V			2		V
		V_{DD} = 6 V			3.4		
V_{T-}	Negative-going threshold voltage (INIT)	V_{DD} = 3.5 V			1.6		V
		V_{DD} = 6 V			2.3		
V_{hys}	Hysteresis ($V_{T+} - V_{T-}$) (INIT)	V_{DD} = 3.5 V			0.4		V
		V_{DD} = 6 V			1.1		
I_{lkg}	Input leakage current (except for OSC IN)					2	µA
$I_{standby}$	Standby current ($\overline{INIT}$ low, SETOFF)					10	µA
I_{DD}†	Supply current	V_{DD} = 3.3 V,	V_{OH} = 2.75 V		2.1		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V		3.1		
		V_{DD} = 6 V,	V_{OH} = 5.5 V		4.5		
I_{OH}	High-level output current (PA, PB)	V_{DD} = 3.3 V,	V_{OH} = 2.75 V	–4	–12		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V	–5	–14		
		V_{DD} = 6 V,	V_{OH} = 5.5 V	–6	–15		
		V_{DD} = 3.3 V,	V_{OH} = 2.2 V	–8	–20		mA
		V_{DD} = 5 V,	V_{OH} = 3.33 V	–14	–40		
		V_{DD} = 6 V,	V_{OH} = 4 V	–20	–51		
I_{OL}	Low-level output current (PA, PB)	V_{DD} = 3.3 V,	V_{OL} = 0.5 V	5	9		mA
		V_{DD} = 5 V,	V_{OL} = 0.5 V	5	9		
		V_{DD} = 6 V,	V_{OL} = 0.5 V	5	9		
		V_{DD} = 3.3 V,	V_{OL} = 1.1 V	10	19		mA
		V_{DD} = 5 V,	V_{OL} = 1.67 V	20	29		
		V_{DD} = 6 V,	V_{OL} = 2 V	25	35		
I_{OH}	High-level output current (D/A)	V_{DD} = 3.3 V,	V_{OH} = 2.75 V	–30	–50		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V	–35	–60		
		V_{DD} = 6 V,	V_{OH} = 5.5 V	–40	–65		
		V_{DD} = 3.3 V,	V_{OH} = 2.3 V	–50	–90		mA
		V_{DD} = 5 V,	V_{OH} = 4 V	–90	–140		
		V_{DD} = 6 V,	V_{OH} = 5 V	–100	–150		
I_{OL}	Low-level output current (D/A)	V_{DD} = 3.3 V,	V_{OL} = 0.5 V	50	80		mA
		V_{DD} = 5 V,	V_{OL} = 0.5 V	70	90		
		V_{DD} = 6 V,	V_{OL} = 0.5 V	80	110		
		V_{DD} = 3.3 V,	V_{OL} = 1 V	100	140		mA
		V_{DD} = 5 V,	V_{OL} = 1 V	140			
		V_{DD} = 6 V,	V_{OL} = 1 V	150			
	Pullup resistance	Resistors selected by software and connected between terminal and V_{DD}		10	20	50	kΩ
$f_{osc(low)}$	Oscillator frequency‡	V_{DD} = 5 V, T_A = 25°C, Target frequency = 15.36 MHz		14.89	15.36	15.86	MHz
$f_{osc(high)}$	Oscillator frequency‡	V_{DD} = 5 V, T_A = 25°C, Target frequency = 19.2 MHz		18.62	19.2	19.7	MHz

† Operating current assumes all inputs are tied to either V_{SS} or V_{DD} with no input currents due to programmed pullup resistors. The DAC output and other outputs are open circuited.

‡ The frequency of the internal clock has a temperature coefficient of approximately –0.2 %/°C and a V_{DD} coefficient of approximately ±1%/V.

MSP50x37 electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{T+}	Positive-going threshold voltage (INIT)	V_{DD} = 4.5 V		2.7		V
		V_{DD} = 6 V		3.65		
V_{T-}	Negative-going threshold voltage (INIT)	V_{DD} = 4.5 V		2.3		V
		V_{DD} = 6 V		3.15		
V_{hys}	Hysteresis ($V_{T+} - V_{T-}$) (INIT)	V_{DD} = 4.5 V		0.4		V
		V_{DD} = 6 V		0.5		
I_{lkg}	Input leakage current (except for OSC IN)				1	µA
$I_{standby}$	Standby current ($\overline{INIT}$ low, SETOFF)			10		µA
I_{DD}†	Supply current	Power amplifier is on		25		mA
		Power amplifier is off		10		
I_{OH}	High-level output current (PA, PB, PD)	V_{DD} = 4 V, V_{OH} = 3.5 V	–4	–6		mA
		V_{DD} = 5 V, V_{OH} = 4.5 V	–5	–7.5		
		V_{DD} = 6 V, V_{OH} = 5.5 V	–6	–9.2		
		V_{DD} = 4 V, V_{OH} = 2.65 V	–8	–13		mA
		V_{DD} = 5 V, V_{OH} = 3.33 V	–14	–20		
		V_{DD} = 6 V, V_{OH} = 4 V	–20	–29		
I_{OL}	Low-level output current (PA4 – PA7)	V_{DD} = 4 V, V_{OL} = 0.5 V	20	28		mA
		V_{DD} = 5 V, V_{OL} = 0.5 V	26	34		
		V_{DD} = 6 V, V_{OL} = 0.5 V	30	39		
		V_{DD} = 4 V, V_{OL} = 1.33 V	40	54		mA
		V_{DD} = 5 V, V_{OL} = 1.67 V	60	74		
		V_{DD} = 6 V, V_{OL} = 2 V	82	103		
I_{OL}	Low-level output current (PA0 – PA3, PB, PD))	V_{DD} = 4 V, V_{OL} = 0.5 V	10	17		mA
		V_{DD} = 5 V, V_{OL} = 0.5 V	13	20		
		V_{DD} = 6 V, V_{OL} = 0.5 V	15	25		
		V_{DD} = 4 V, V_{OL} = 1.33 V	20	32		mA
		V_{DD} = 5 V, V_{OL} = 1.67 V	30	52		
		V_{DD} = 6 V, V_{OL} = 2 V	41	71		
	Pullup resistance	Resistors selected by software and connected between terminal and V_{DD}	15	30	60	kΩ
$f_{osc(low)}$	Oscillator frequency‡	V_{DD} = 5 V, T_A = 25°C, Target frequency = 15.36 MHz	14.89	15.36	15.82	MHz
$f_{osc(high)}$	Oscillator frequency‡	V_{DD} = 5 V, T_A = 25°C, Target frequency = 19.2 MHz	18.62	19.2	19.77	MHz

† Operating current assumes all inputs are tied to either V_{SS} or V_{DD} with no input currents due to programmed pullup resistors. The DAC output and other outputs are open circuited.

‡ The frequency of the internal clock has a temperature coefficient of approximately –0.2 %/°C and a V_{DD} coefficient of approximately ±1.4%/V.

MSP50x37 Power Amplifier Electrical Characteristics Over Recommended Operating Free-Air Temperature Range

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Differential output power	V_{DD} = 5 V, f = 1 kHz, R_L = 8 Ω		500		mW
Bandwidth				3.5	kHz

MSP50x37 ADC Electrical Characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER	MIN	TYP	MAX	UNIT
Linearity		±0.5		LSB
Offset		±1.5		LSB
Full scale error		±1.5		LSB
Conversion time		40		Instructions

switching characteristics (MSP50C32, MSP50C33, MSP50x34)

PARAMETER		TEST CONDITIONS	MIN	NOM	MAX	UNIT
t_r	Rise time, PA, PB, D/A	V_{DD} = 3.3 V, C_L = 100 pF, 10% to 90%		50		ns
t_f	Fall time, PA, PB, D/A	V_{DD} = 3.3 V, C_L = 100 pF, 10% to 90%		50		ns

switching characteristics (MSP50x37)

PARAMETER		TEST CONDITIONS	MIN	NOM	MAX	UNIT
t_r	Rise time, PA, PB, PD	V_{DD} = 4 V, C_L = 100 pF, 10% to 90%		22		ns
t_f	Fall time, PA, PB, PD	V_{DD} = 4 V, C_L = 100 pF, 10% to 90%		10		ns

timing requirements

			MIN	MAX	UNIT
Initialization					
t_{INIT}	$\overline{INIT}$ pulsed low while the MSP50x3x has power applied (see Figure 1)		1		µs
Wakeup					
$t_{su(wakeup)}$	Setup time prior to wakeup terminal negative transition (see Figure 2)		1		µs
External Interrupt					
$t_{su(interrupt)}$	Setup time prior to B1 terminal negative transition (see Figure 3)	f_{clock} = 15.36 MHz	1		µs
		f_{clock} = 19.2 MHz	1.5		
Writing (Slave Mode)					
$t_{su1(B1)}$	Setup time, B1 low before B0 goes low (see Figure 4)		20		ns
$t_{su(d)}$	Setup time, data valid before B0 goes high (see Figure 4)		100		ns
$t_{h1(B1)}$	Hold time, B1 low after B0 goes high (see Figure 4)		20		ns
$t_{h(d)}$	Hold time, data valid after B0 goes high (see Figure 4)		30		ns
t_w	Pulse duration, B0 low (see Figure 4)		100		ns
t_r	Rise time, B0 (see Figure 4)			50	ns
t_f	Fall time, B0 (see Figure 4)			50	ns
Reading (Slave Mode)					
$t_{su2(B1)}$	Setup time, B1 before B0 goes low (see Figure 5)		20		ns
$t_{h2(B1)}$	Hold time, B1 after B0 goes high (see Figure 5)		20		ns
t_{dis}	Output disable time, data valid after B0 goes high (see Figure 5)		0	30	ns
t_w	Pulse duration, B0 low (see Figure 5)		100		ns
t_r	Rise time, B0 (see Figure 5)			50	ns
t_f	Fall time, B0 (see Figure 5)			50	ns
t_d	Delay time for B0 low to data valid (see Figure 5)			50	ns

PARAMETER MEASUREMENT INFORMATION

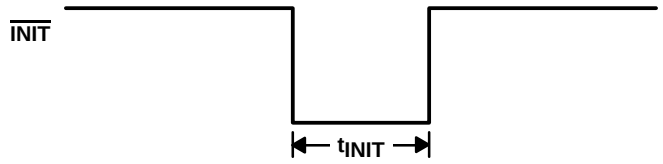


Figure 1. Initialization Timing Diagram

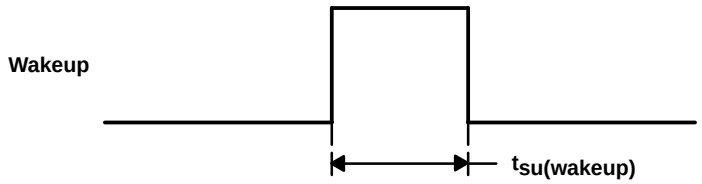


Figure 2. Wakeup Terminal Setup Timing Diagram

PARAMETER MEASUREMENT INFORMATION

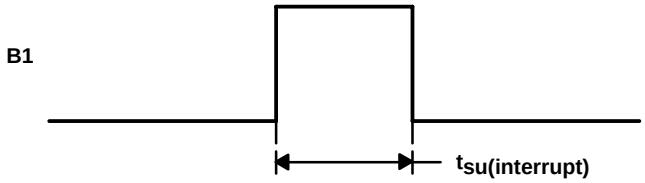


Figure 3. External Interrupt Terminal Setup Timing Diagram

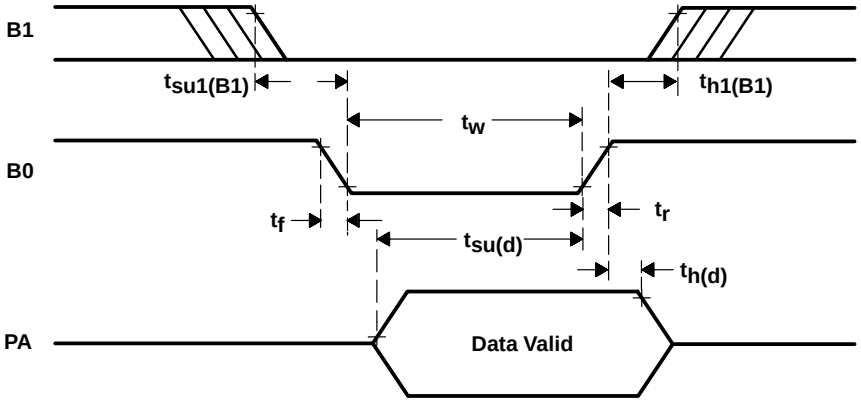


Figure 4. Write Timing Diagram (Slave Mode)

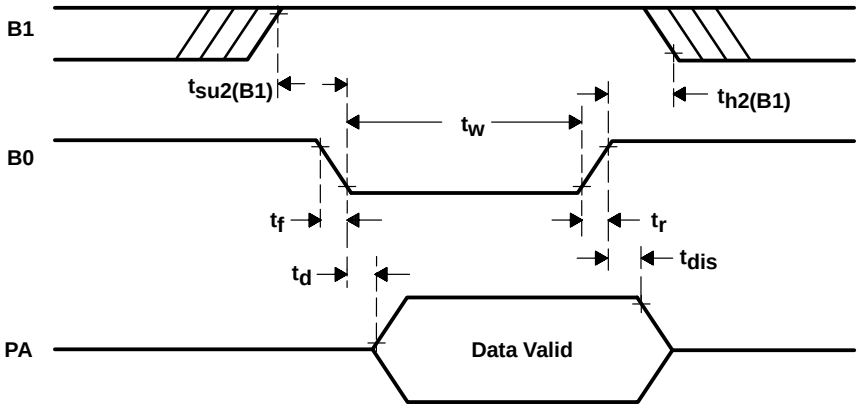


Figure 5. Read Timing Diagram (Slave Mode)

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W

X