TEXAS
INSTRUMENTS

TSP50C1x Family

Speech Synthesizer

Design Manual

1993

Linear Products

Design Manual

TSP50C1x Family
Speech Synthesizer

1993

TSP50C1x Family

Speech Synthesizer

Design Manual

Texas Instruments

IMPORTANT NOTICE

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

TI warrants performance of its semiconductor products and related software to current specifications 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.

Please be aware that TI products are not intended for use in life-support appliances, devices, or systems. Use of TI product in such applications requires the written approval of the appropriate TI officer. Certain applications using semiconductor devices may involve potential risks of personal injury, property damage, or loss of life. In order to minimize these risks, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards. Inclusion of TI products in such applications is understood to be fully at the risk of the customer using TI devices or systems.

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

Printed in the U.S.A.

Contents

Section *Title* *Page*

Contents (Continued)

Contents (Continued)

Contents (Continued)

Section *Title* *Page*

Contents (Continued)

Section *Title* *Page*

Contents (Continued)

Contents (Continued)

List of Illustrations

List of Illustrations (Continued)

List of Tables

1 Introduction to the TSP50C1x Family of Devices

The TSP50C1x uses a revolutionary architecture to combine an 8-bit microprocessor, a speech synthesizer, ROM, RAM, and I/O in a low-cost single-chip system. The architecture uses the same ALU (Arithmetic Logic Unit) for the synthesizer and the microprocessor, thus reducing chip area and cost and enabling the microprocessor to do a multiply operation in 1.6 µs. Linear Predictive Coding (LPC) is used to synthesize high-quality speech at a low data rate.

1.1 Applications

The TSP50C1x 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. They include:

- Talking Clocks
- Toys
- Telephone Answering Machines
- Home Monitors
- Navigation Aids
- Laboratory Instruments
- Personal Computers
- Inspection Controls
- Inventory Controls
- Machine Controls
- Warehouse Systems
- Warning Systems
- Appliances
- Mailboxes
- Equipment for the Handicapped
- Learning Aids
- Computer-Aided Instruction
- Magazine and Direct-Mail Advertisements
- Point-of-Sale Displays

1.2 Description

The TSP50C1x can be divided into several functional blocks (Figure 1–1, 1–2, 1–3). The ALU and RAM are shared by the speech synthesizer and the microcomputer.

The TSP50C1x implements an LPC-12 speech synthesis algorithm using a 12-pole lattice filter. The internal microprocessor fetches speech data from the internal or external ROM (TSP60C18), decodes the speech data, and sends the decoded data to the synthesizer. The microprocessor also interpolates (smooths) the speech data between fetches. The output of the synthesizer can be used to drive transistor or integrated-circuit amplifiers. Some digital low-pass filtering is provided inside the TSP50C1x.

The general-purpose microprocessor in the TSP50C1x is also capable of a variety of logical, arithmetic, and control functions and can often be used for the nonsynthesis tasks of the customer's application as well.

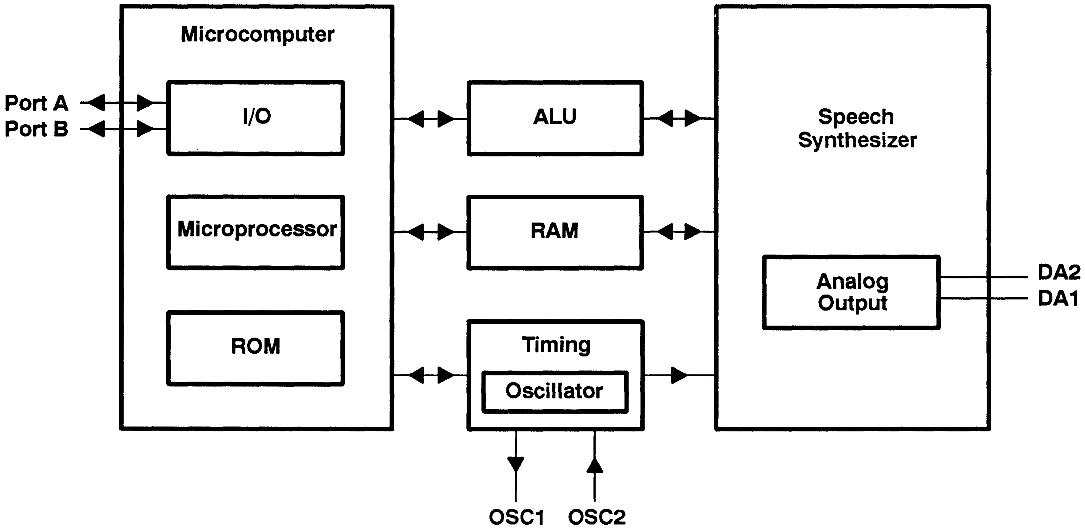


Figure 1–1. TSP50C10/11 Functional Block Diagram

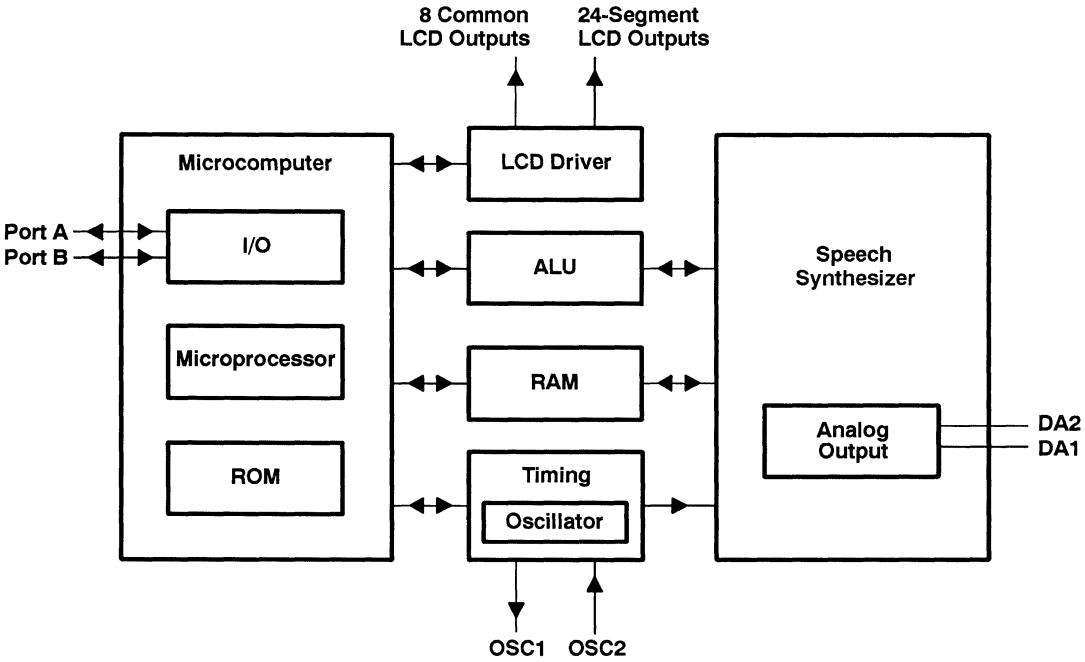


Figure 1–2. TSP50C12 Functional Block Diagram

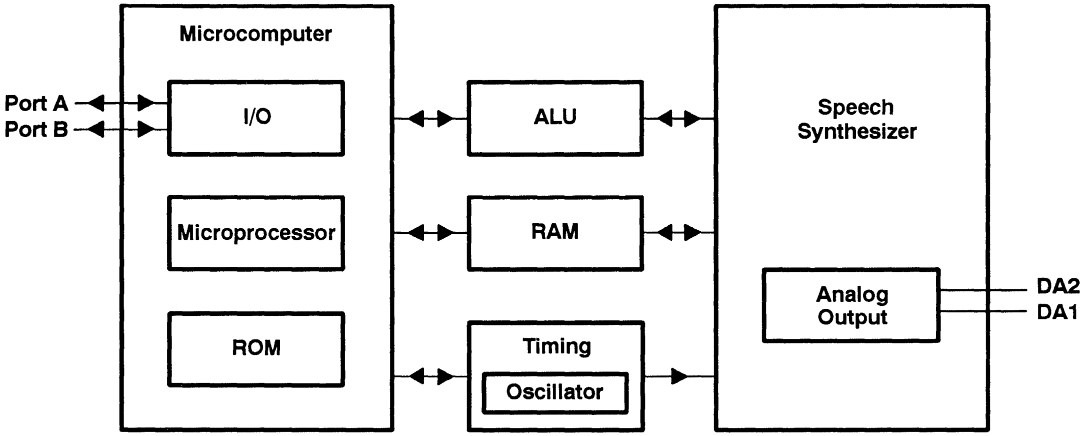


Figure 1–3. TSP50C14 Functional Block Diagram

1.3 TSP50C1x Family Features

- Programmable LPC-12 Speech Synthesizer
- 8-Bit Microprocessor With 61 Instructions
- 16 Twelve-Bit Words and 112 Bytes of RAM
- 4-V to 6-V CMOS Technology for Low Power Dissipation
- 3 D/A Configurations – Mask Selectable
- 10-kHz or 8-kHz Speech Sample Rate
- 8K-Byte ROM (TSP50C10) or 16K-Byte ROM (TSP50C11/12/14)
- 10 Software Controllable I/O Lines (9 I/O Lines With Two-Pin D/A Output)
- Internal Timer
- External Interrupt
- Single-Cycle Multiply Instruction
- Executes Up to 600,000 Instructions Per Second
- Built-in Interface to TSP60C18 Speech ROM
- Built-In Slave Mode to Act as Microprocessor Peripheral

1.4 TSP50C12 Features

- Direct LCD Drive Capability for an 8 × 24 (192-Segment) Display
- 1/8 Duty Cycle and 1/4 Bias Drive With On-Chip Voltage Reference
- Internal Contrast Adjustment
- 24 Bytes of Display RAM
- Two D/A Configurations – Mask Selectable
- Limited Direct Speaker Drive Capability
- RC Oscillator Option

1.5 TSP50C14 Features

- Direct Speaker Drive Capability
- Internal Clock Generator That Requires No External Components
- Two-Pin D/A Output and 10 Pins of I/O Simultaneously Possible
- Two D/A Configurations – Mask Selectable
- Optional Doubling of the D/A Output
- 16 Twelve-Bit Words and 48 Bytes of RAM

1.6 D/A Options

The TSP50C1x offers three D/A (digital-to-analog) output options to match different applications. The DAC (digital-to-analog converter) is a pulse-width-modulated type with 9 bits or 10 bits of resolution and a 16-kHz or 20-kHz sampling rate. Each option has a range of 480 to −480 segments per sample period, with two options having a resolution of ± 1/2 LSB and the third having a resolution of ± 1 LSB.

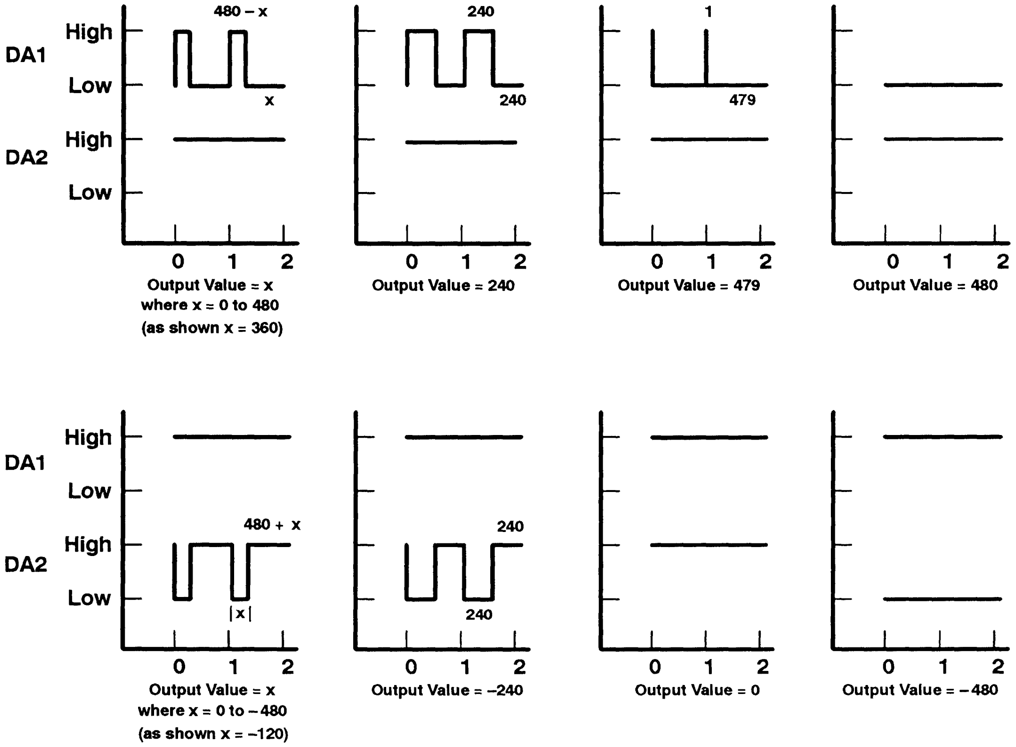
The DAC produces samples at twice the rate that data is received from the LPC filter. For example, if the LPC filter is running at approximately 10 kHz, then the DAC is running at approximately 20 kHz.

The TSP50C12 and TSP50C14 can be used with a normal-sized pulse width or with the PW2 option. The PW2 option causes the processor to produce a double-sized pulse width. This results in a higher volume output, which includes some risk of clipping the output.

1.6.1 Two-Pin Push Pull (Option 1) – Accurate to 10 Bits (±1/2 LSB)

Option 1 works well with a very efficient and inexpensive four-transistor amplifier. It requires two pins, so the I/O pin B1 is used for the second pin, meaning that only 9 bits of I/O are available. When the DAC is idle, or the output value is 0, both pins are high. When the output value is positive, DA1 goes low with a duty cycle proportional to the output value, while DA2 stays high. When the output value is negative, DA2 goes low with a duty cycle proportional to the output value, while DA1 stays high. This option offers a resolution of 10 bits.

Figure 1–4 shows examples of D/A output waveforms with different output values. Each pulse of the DAC is divided into 480 segments per sample period. For a positive output value x = 0 to 480, DA1 will go low for x segments while DA2 stays high. When the DAC is idle or the output value is 0, both DA1 and DA2 are high. For a negative value x = 0 to −480, DA2 will go low for |x| segments while DA1 stays high.



Output Value = x where x = 0 to 480 (as shown x = 360)

Output Value = 240

Output Value = 479

Output Value = 480

Output Value = x where x = 0 to –480 (as shown x = –120)

Output Value = –240

Output Value = 0

Output Value = –480

Figure 1–4. D/A Output Waveform for Two-Pin Push Pull (Option 1)

Figures 1–5, 1–6, and 1–7 show examples of circuits that can be used with this option.

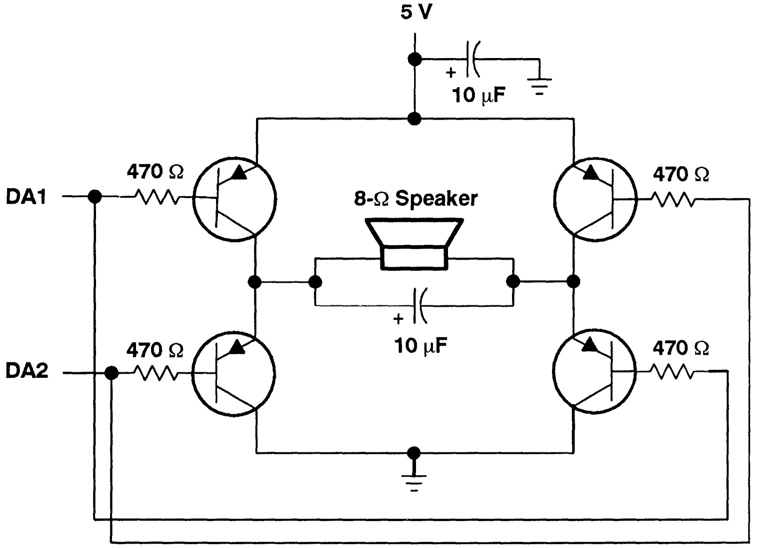


Figure 1–5. Four-Transistor Amplifier Circuit

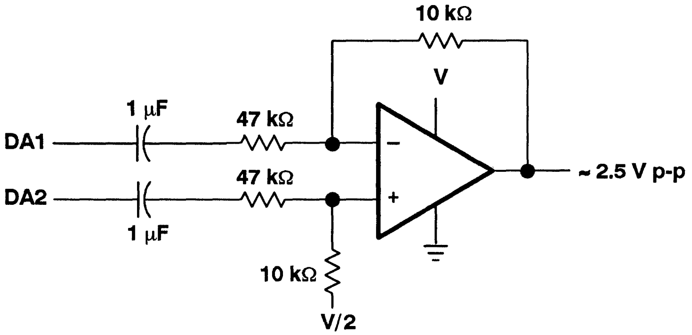
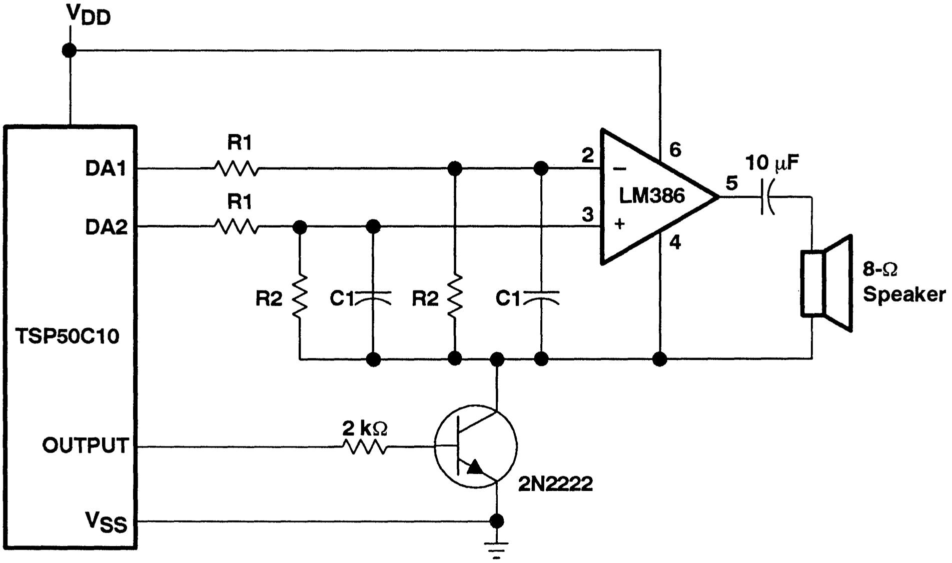


Figure 1–6. Operational Amplifier Interface Circuit



NOTES: R1 ~ 56 kΩ 10%

R2 = 2 kΩ 10%

C1 = 0.022 µF 20%

R2 and C1 set low-pass cutoff frequency: $f_C = 1/(2\pi R2 \times C1)$

For values given above, f_C = 3.6 kHz

Gain control can be added by connecting a 10-µF capacitor in series with a 10-kΩ pot. This series combination is connected between pins 1 and 8. When this is done, R1 should be increased to approximately 250 kΩ.

Figure 1–7. Power Amplifier Interface Circuit

1.6.2 Single-Pin Single Ended (Option 2) – Accurate to Only 9 Bits (±1 LSB)

Option 2 is designed for use with a single-transistor amplifier, offering the lowest-cost solution and still retaining all 10 I/O pins. It has only 9 bits of resolution, and the amplifier power consumption is higher than the four-transistor amplifier mentioned above. It is available on the TSP50C10, TSP50C11, and the TSP50C14. The duty cycle of the output is proportional to the output value. If the output value is 0, the duty cycle is 50%. As the output value increases from 0 to the maximum, the duty cycle goes from being high 50% of the time up to 100% high. As the value goes from 0 to the most negative value, the duty cycle decreases from 50% high to 0%.

Each pulse of the DAC is divided into 480 segments per sample period. As shown in Figure 1–8, when the output value is x = –480 to 480, DA1 will go low for |x/2–240| segments. When the output value is 0, DA1 goes low for 240 segments.

NOTE: Using Option 2 causes a click at the beginning and end of speech and (under certain conditions) during synthesis. Software is available to minimize these clicks.

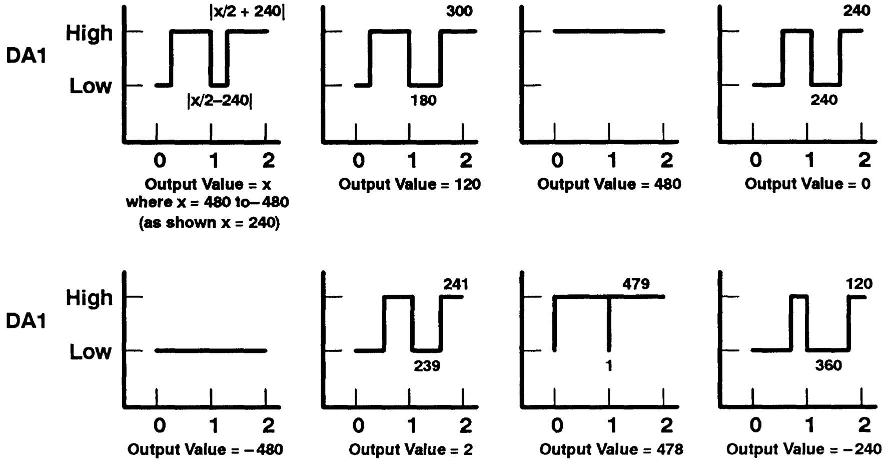


Figure 1–8. D/A Output Waveform for Single Ended (Option 2)

Figure 1–9 shows an example of a circuit that can be used with option 2.

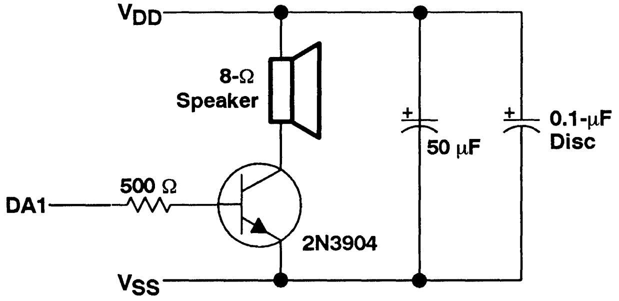


Figure 1–9. One-Transistor Amplifier Circuit

1.6.3 Single-Pin Double Ended (Option 3) – Accurate to 10 Bits (±1/2 LSB)

Option 3 is provided for use with operational and power amplifiers. It offers both 10 bits of resolution and 10 I/O pins and is available on the TSP50C10, TSP50C11, and the TSP50C12. 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 about 1/2 V_{DD} – 1 V. The duty cycle is proportional to the output value. When the output value is negative, the D/A output pulses to 1/2 V_{DD}+1 V with a duty cycle proportional to the output value.

Figure 1–10 shows examples of D/A output waveforms with different output values. Each pulse of the DAC is divided into 480 segments per sample period. For a positive output value x = 0 to 480, DA1 will go low to 1/2 V_{DD}–1 V for x segments. When the DAC is idle, or the output value is 0, DA1 will go to 1/2 V_{DD}. For a negative value x = 0 to –480, DA1 will go high to 1/2 V_{DD} + 1 V for |x| segments.

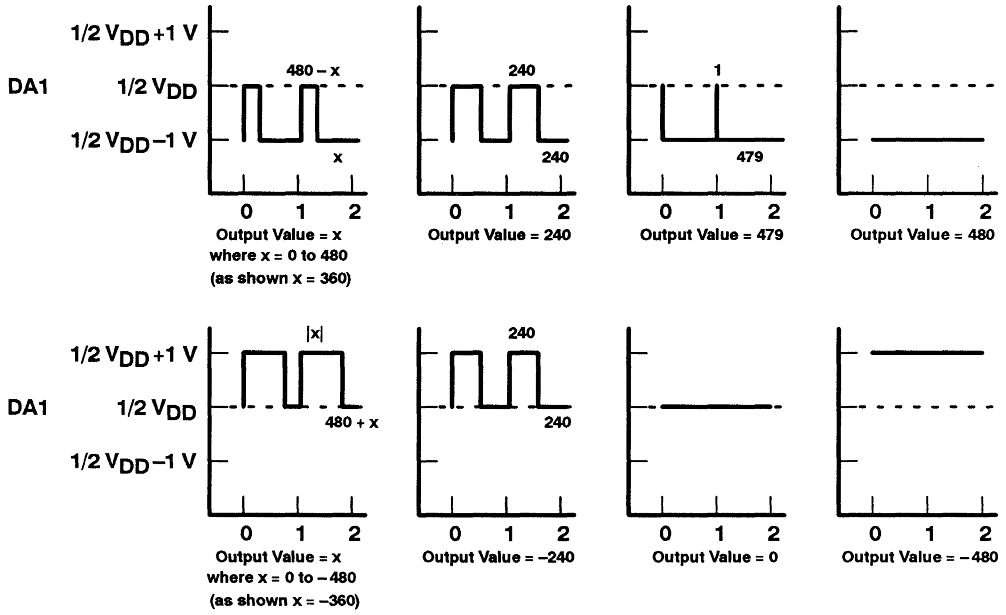


Figure 1–10. D/A Output Waveform – Single-Pin Double Ended (Option 3)

Figure 1–11 shows an example of a circuit that can be used with option 3.

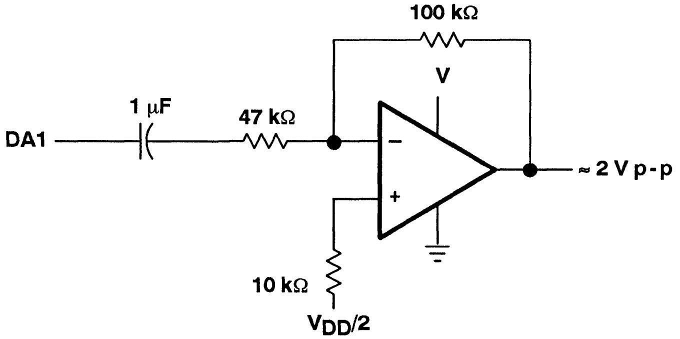


Figure 1–11. Operational Amplifier Interface Circuit

1.7 TSP50C10/11 Pin Assignments and Descriptions

Figure 1–12 shows the pin assignments for the TSP50C10/11. Table 1–1 provides terminal functional descriptions. Table 1–2 shows the possible TSP50C10/11 I/O configurations. Figure 1–13 illustrates the recommended power-up initialization circuit. Note that the pullup resistor is required to be lower than 50 kΩ. Figure 1–14 illustrates the recommended clock circuit. Refer to Section 6 for more information on I/O configuration.

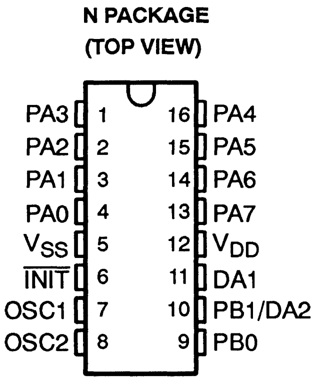


Figure 1–12. TSP50C10/11 Pin Assignments

Table 1–1. TSP50C10/11 Terminal Functions

PIN		I/O	DESCRIPTION
NAME	NO.		
DA1	11	O	D/A output. Three mask options are available.
DA2	10[†]	O	D/A output. Three mask options are available.
INIT	6	I	Initialize input. When $\overline{\text{INIT}}$ goes low, the clock stops, the TSP50C10/11 goes into low-power mode, the program counter is set to zero, and the contents of the RAM are retained. An $\overline{\text{INIT}}$ pulse of 1 µs is sufficient to reset the processor.
OSC1	7	I	Clock input. Crystal or ceramic resonator between OSC1 and OSC2, or signal into OSC1. 9.6 MHz for 10-kHz sampling rate or 7.68 MHz for 8-kHz sampling rate.
OSC2	8	–	Clock return
PA0–PA7	1–4, 13–16	I/O	8-bit bidirectional I/O port
PB0–PB1	9–10[†]	I/O	2-bit bidirectional I/O port
V_{DD}	12	–	5-V supply voltage
V_{SS}	5	–	Ground terminal

[†] The operation of this pin depends on the D/A option selected.

Table 1–2. TSP50C10/11 I/O Configurations

PIN	MASTER			SLAVE 1-PIN D/A	MASTER 1-PIN D/A 60C18
	1-PIN D/A	1-PIN D/A†	2-PIN D/A		
4	PA0	PA0	PA0	D0	C0
3	PA1	PA1	PA1	D1	C1
2	PA2	PA2	PA2	D2	C2
1	PA3	PA3	PA3	D3	C3
16	PA4	PA4	PA4	D4	PA4
15	PA5	PA5	PA5	D5	PA5
14	PA6	PA6	PA6	D6	PA6
13	PA7	PA7	PA7	BUSY/D7	SRCK
9	PB0	PB0	PB0	$\overline{\text{CE}}$	STR
10	PB1	$\overline{\text{IRQ}}$	DA2	R/$\overline{\text{W}}$	R/$\overline{\text{W}}$

† With external interrupt

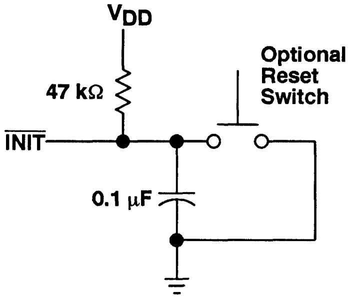


Figure 1–13. Power-Up Initialization Circuit

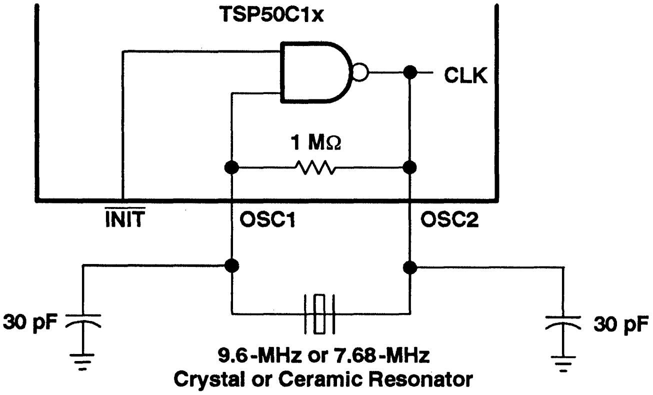
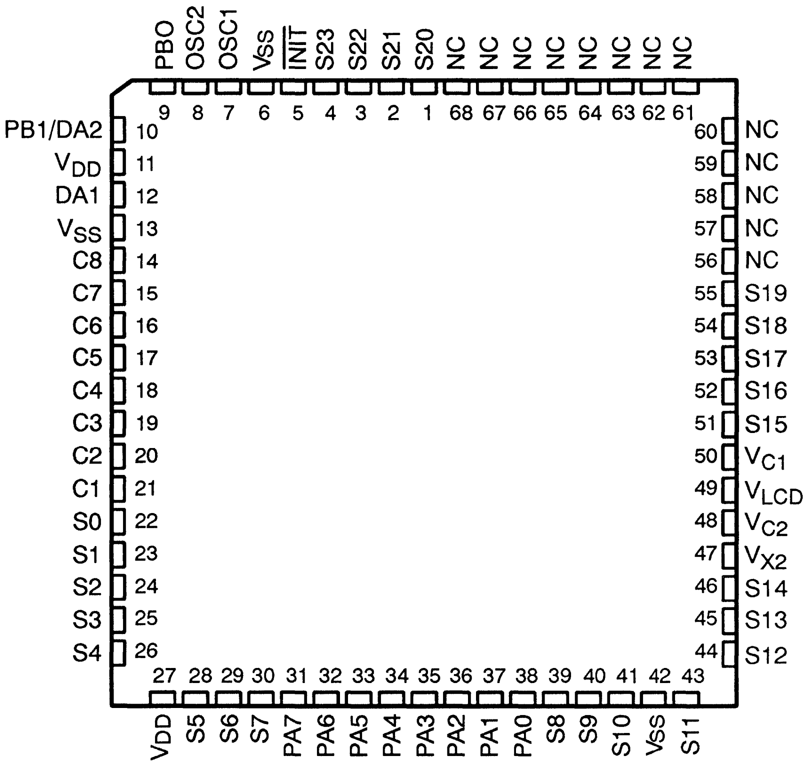


Figure 1–14. Oscillator Circuit

1.8 TSP50C12 Pin Assignments and Descriptions

Figure 1–15 shows the pin assignments for the TSP50C12. Table 1–3 provides terminal functional descriptions. The I/O configurations in Table 1–2 also applies to the TSP50C12, but the pin numbers given are different. Figure 1–13 illustrates the recommended power-up initialization circuit, and Figure 1–14 illustrates the recommended clock circuit. The TSP50C12 is available as a 68-pin PLCC or as a die. Refer to Section 6 for more information on I/O configuration.

PLCC PACKAGE
(TOP VIEW)



NC – No internal connection

Figure 1–15. TSP50C12 Pin Assignments

Table 1–3. TSP50C12 Terminal Functions

PIN		I/O	DESCRIPTION
NAME	NO.		
DA2	10†	O	D/A output. D/A options 1 and 3 are available.
DA1	12	O	D/A output. D/A options 1 and 3 are available.
PB1	10†	I/O	Bidirectional I/O pin
PB0	9	I/O	Bidirectional I/O pin
$\overline{\text{INIT}}$	5	I	Initialize input. When $\overline{\text{INIT}}$ goes low, the clock stops, the TSP50C12 goes into low-power mode, the program counter is set to zero, and the contents of the RAM are retained. An $\overline{\text{INIT}}$ pulse of 1 µs is sufficient to reset the processor.
OSC1‡	7	I	Clock input. Crystal or ceramic resonator between OSC1 and OSC2, or signal into OSC1. 9.6 MHz for 10-kHz sampling rate or 7.68 MHz for 8-kHz sampling rate.
OSC2‡	8	–	Clock return
PA0 – PA7	31–38	I/O	8-bit bidirectional I/O port
C1–C8	14–21	O	LCD common lines (rows)
SEG1–SEG24	1–4, 22–26, 28–30, 39–41, 43–46, 51–55	O	LCD segment lines (columns)
V_{C1}	50	–	Voltage doubler capacitor connection
V_{C2}	48	–	
V_{X2}	47	–	
V_{LCD}	49	–	LCD supply voltage
V_{DD}	11, 27	–	5-V supply voltage
V_{SS}	6, 13, 42	–	Ground terminals

† The operation of this pin depends on the D/A option selected.

‡ Ceramic resonator requires two pins. RC oscillator requires one pin for timing and one buffered clock output for trim monitoring.

1.9 TSP50C14 Pin Assignments and Descriptions

Figure 1–16 shows the pin assignments for the TSP50C14. Table 1–4 provides terminal functional descriptions. The I/O configurations in Table 1–2 apply to the TSP50C14 with the exception of the pin numbering and the DA2 pin assignment. Figure 1–13 illustrates the recommended power-up initialization circuit for the TSP50C14. For most configurations, OSC1 should be tied to either V_{SS} or V_{DD}. Refer to Section 6 for more information on I/O configurations.

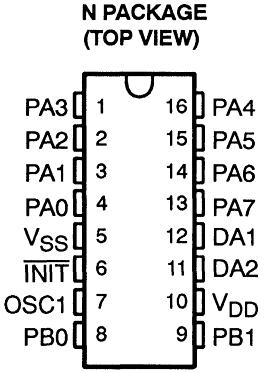


Figure 1–16. TSP50C14 Pin Assignments

Table 1–4. TSP50C14 Terminal Functions

PIN NAME	 NO.	I/O	DESCRIPTION
DA1	11	O	D/A output. D/A options 1 and 2 are available.†
DA2	12	O	D/A output. D/A options 1 and 2 are available.†
$\overline{\text{INIT}}$	6	I	Initialize input. When $\overline{\text{INIT}}$ goes low, the clock stops, the TSP50C14 goes into low-power mode, the program counter is set to zero, and the contents of the RAM are retained. An $\overline{\text{INIT}}$ pulse of 1 µs is sufficient to reset the processor.
OSC1	7	I	Clock input. When not in use, OSC1 should be tied to V_{SS} or V_{DD}.
PA0–PA7	1–4, 13–16	I/O	8-bit bidirectional I/O port
PB0–PB1	8–9	I/O	2-bit bidirectional I/O port
V_{DD}	10	–	5-V supply voltage
V_{SS}	5	–	Ground terminal

† Both DA1 and DA2 are driven with the same levels, if option 2 is selected.

1.10 Introduction to LPC (Linear Predictive Coding)

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. A short discussion of the techniques and pitfalls of collecting, analyzing, and editing speech for LPC synthesis is included in Appendix A. For more information, contact your TI Field Sales Representative or Regional Technology Center.

1.10.1 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 air flow 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 that produces 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.

1.10.2 The LPC Model

The LPC model incorporates elements analogous to each of the elements of the vocal tract previously described. 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 1–17 shows the LPC 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 TSP50C1x, this filter has twelve poles, so the synthesis is referred to as LPC-12.

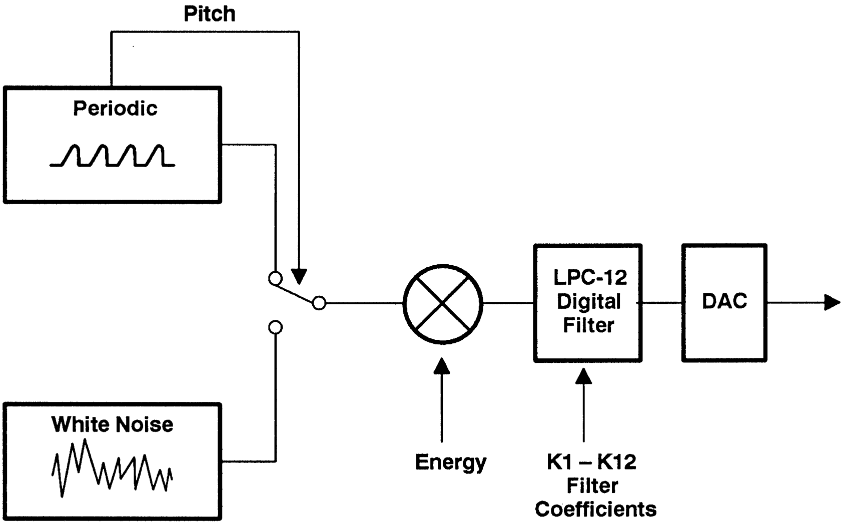


Figure 1–17. LPC-12 Vocal Tract Model

1.10.3 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 than others. The speech is analyzed in 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 not an 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 seven to three bits for each coefficient. The coding table is designed so that more coverage is given to the coefficient values that occur frequently.

2 TSP50C1x Family Architecture

As shown in the block diagram in Figure 2–1, the major components of the TSP50C1x are a speech synthesizer, an 8-bit microprocessor, an internal 8K-byte (TSP50C10) or 16K-byte (TSP50C11/12/14) ROM, and input/output ports.

When synthesis is disabled, instructions are fetched by the microprocessor from the ROM 600,000 (10-kHz speech sample rate) or 480,000 (8-kHz speech sample rate) times per second. These instructions control the actions of the TSP50C1x. By placing different instruction patterns in the ROM, the TSP50C1x can be programmed to accomplish a wide variety of tasks. To generate speech, the processor accesses speech data from either the internal ROM or an external source such as a TSP60C18 speech ROM, an EPROM, or a host processor. 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. The instruction execution rate slows to 280,000 or 224,000 instruction cycles per second during synthesis because the synthesizer also shares the ALU (Arithmetic Logic Unit) and ROM data paths with the microprocessor. The microprocessor must perform interpolation during each frame as well as fetch the data for the next frame.

The I/O consists of one 8-bit bidirectional port (Port A) and one 2-bit bidirectional port (Port B). Each bit can be software configured for input or output and for push pull or open drain (no pullup driver). There are two specialized I/O modes for specific functions. Slave mode configures the TSP50C1x to act as a peripheral to a host microprocessor. External ROM mode allows the TSP50C1x to interface with a TSP60C18 speech ROM.

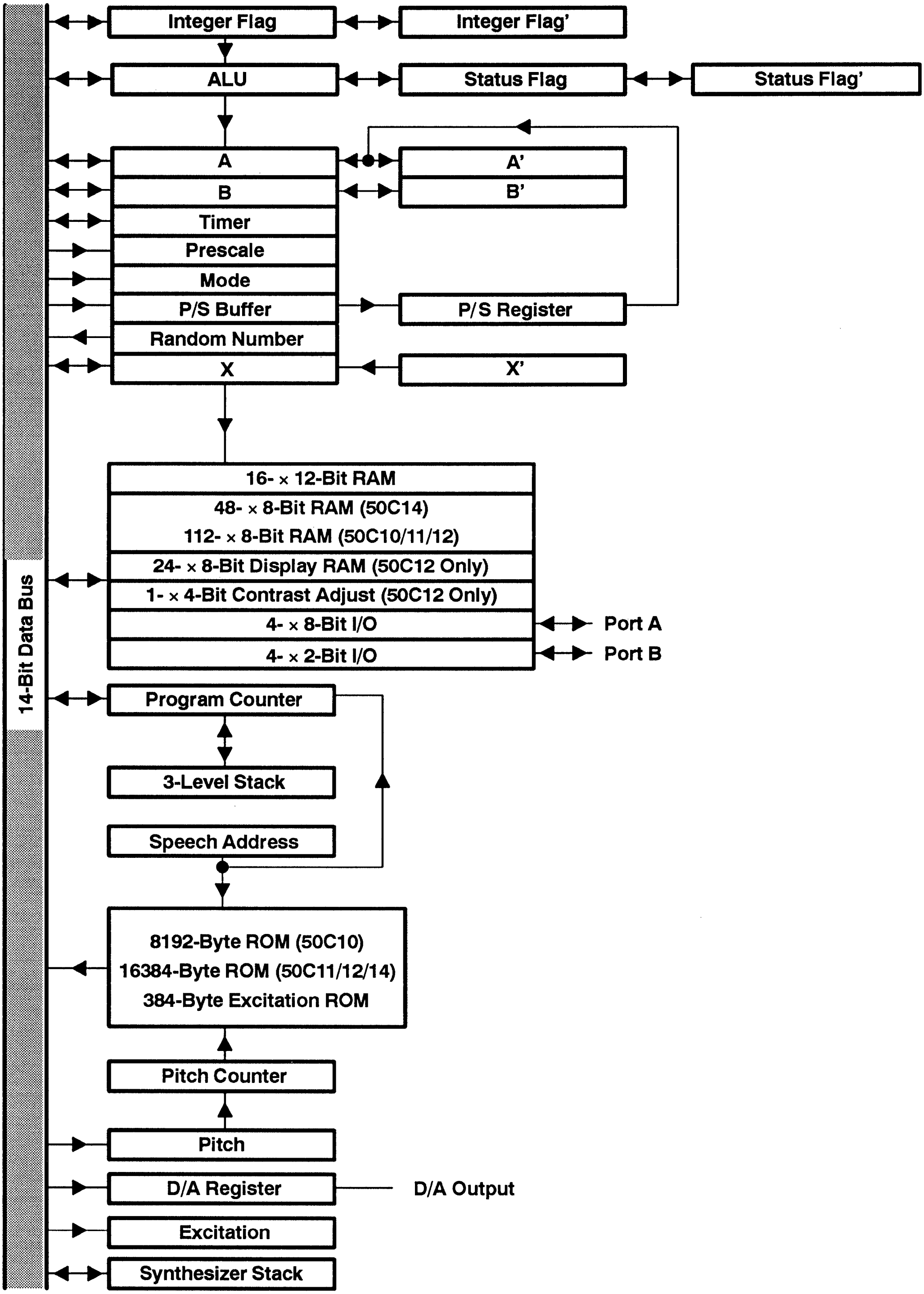


Figure 2–1. TSP50C1x System Block Diagram

2.1 Read-Only Memory (ROM)

The TSP50C10 has an 8K-byte ROM. The TSP50C11/12/14 each have a 16K-byte ROM. It can be used for 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
0000	Execution start location after $\overline{\text{INIT}}$ rising edge
0010–001F	Interrupt start locations (see Section 2.21)
1FE0–1FFF	Texas Instruments test code for TSP50C10
3FE0–3FFF	Texas Instruments test code for TSP50C11/12/14

The ROM may be accessed in the following four ways:

1. The program counter is used to address processor instructions.
2. The GET instruction can be used to transfer 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.
3. The LUAA instruction can be used to transfer 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.
4. The LUAB instruction can be used to transfer 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

2.2 Program Counter

The TSP50C1x has a 14-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.

The following instructions modify the program counter:

BR	— branch
BRA	— branch to address in A register
SBR	— short branch
CALL	— call subroutine
RETN	— return from subroutine
RETI	— return from interrupt

2.3 Program Counter Stack

The program counter stack has three levels. When a subroutine is called or an interrupt occurs, the contents of the program counter are pushed onto the stack. When an 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.

2.4 TSP50C10/11 Random-Access Memory (RAM)

The TSP50C10/11 RAM has 128 locations (Figure 2–2). The first 16 RAM locations are used by the synthesizer and are 12 bits long. The remaining 112 locations are 8 bits long. When not synthesizing speech, the entire RAM may be used for algorithm data storage. The I/O control registers are also mapped into the RAM address space from 80 to 87. For more information, see Section 2.18.

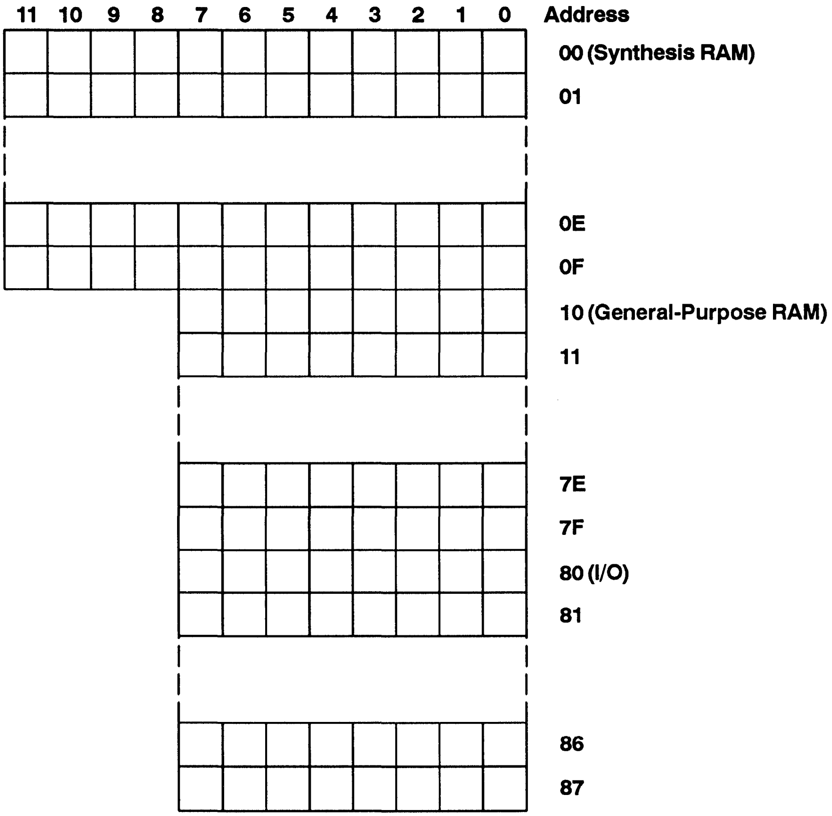


Figure 2–2. TSP50C10/11 RAM Map

2.5 TSP50C12 Random-Access Memory (RAM)

The TSP50C12 RAM has 16 12-bit synthesizer RAM locations and 112 8-bit general purpose RAM locations (Figure 2–3). The RAM also has 24 8-bit display RAM locations and one 4-bit contrast adjustment register. The I/O ports are mapped into RAM address space from F0–F7.

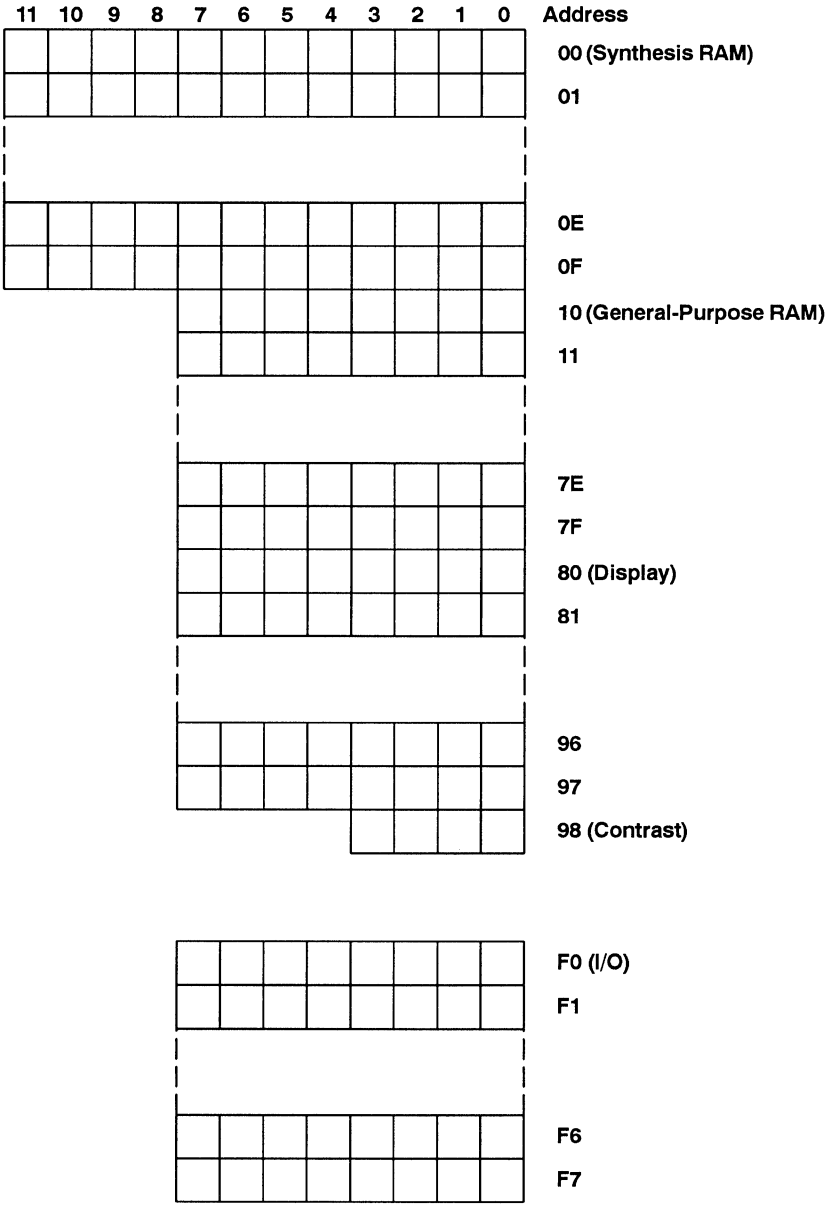


Figure 2–3. TSP50C12 RAM Map

2.6 TSP50C14 Random-Access Memory (RAM)

The TSP50C14 RAM has the same RAM layout as the TSP50C10/11 (see Figure 2–2) with one exception. The general-purpose RAM location range is from 10 to 3F.

2.7 Arithmetic Logic Unit (ALU)

The ALU performs arithmetic and logic functions for the microprocessor and the synthesizer. The ALU is 14 bits in length, providing the resolution needed for speech synthesis. When 8-bit data are transferred to the ALU, they are right justified. The input to the upper 6 bits may be either zeros (integer mode) or equal to the MSB of the 8-bit data (extended-sign mode) depending on the arithmetic mode selected using the EXTSG and INTGR instructions. See the description of each instruction for specific information. All bit and comparison operations are performed on the lower 8 bits. The ALU is capable of doing an 8-bit by 14-bit multiply with a 14-bit scaled result in a single instruction cycle.

2.8 A Register

The A register or accumulator is the primary 14-bit register and is used for arithmetic and logical operations. Its contents can be transferred to or from ROM, RAM, and most of the other registers. The contents are saved in a dedicated storage register during level-1 interrupts and restored by the RETI instruction.

A Register

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

2.9 X Register

The X register is an 8-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. If a RAM location with an illegal address is loaded via the X register, the EVM board with the TSE chip will accept it, but a problem will appear on the TSP chip.

X Register

7	6	5	4	3	2	1	0

2.10 B Register

The 14-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.

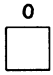
B Register

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

2.11 Status Flag

The status flag is set or cleared by various instructions depending on the result of the instruction. Refer to the individual description of instructions in Section 5 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 unknown at power up. Therefore, if the first instruction after power up is one of these conditional instructions, the execution of the instruction cannot be predicted. The value of the status flag is saved during interrupts and restored by the RETI instruction.

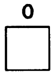
Status Flag



2.12 Integer Mode Flag

The 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 data less then 14 bits in length will be zero filled when being transferred to, added to, or subtracted from the A and B registers. When the integer mode flag is cleared (extended-sign mode), the upper bits of data less than 14 bits in length will be sign extended when being transferred to, added to, or subtracted from the A and B registers. The value of the integer mode flag is saved during interrupts and restored by the RETI instruction.

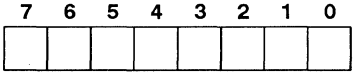
Integer Mode Flag



2.13 Timer Register

The 8-bit timer register is used for generating interrupts and for counting events. It decrements once each time the timer prescale register goes from 00 to FF. It can be loaded using the TATM instruction and examined with the TTMA instruction. When it decrements from 00 to FF, a level-2 interrupt request is generated. If interrupts are enabled and no interrupt is being processed already, an immediate interrupt occurs; if not, the interrupt request remains pending until interrupts are enabled. The timer continues to count whether or not it is reloaded. The timer will not decrement before it is initialized. However, on the EVM, the timer will decrement after a STOP/RUN.

Timer Register

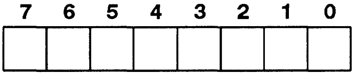


2.14 Timer Prescale Register

The 8-bit timer prescale register is a programmable divider between the processor clock and the timer register. When it decrements from 00 to FF, 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.

The timer prescale register clock comes from an internal clock. The internal clock runs at 1/16 the clock frequency of the chip; thus, the timer prescale register decrements once every instruction cycle when not in LPC mode. The TAPSC instruction loads the timer prescale register's 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



2.15 Pitch Register

Although the 14-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 TSP50C1x, the pitch period counter is also used to control the interpolation of all speech 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 the speech and by making it a harmonic of the speech 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. When speech starts, the pitch period counter is cleared. The pitch period counter is decremented by 20_{16} for each speech sample, with speech 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 200_{16} or when a pitch register is added to it with a result less than 200_{16}, a level-1 interrupt occurs. This interrupt can be used for interpolation. The excitation function is put out when the pitch period counter is between 140_{16} and 00. For further information, see Section 6.

Pitch Register

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

For voiced or unvoiced frames, the LSB and the MSB of the A register must be zero 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 TSP50C1x chip may occur. Also, these problems may not be apparent when using the TSE50C1x chip.

A Register

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

↓

Pitch Register

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 $1FFE_{16}$. 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 42_{16} or higher. If this is not done, problems with the TSP chip may occur which are not apparent with the TSE chip. Second, it is strongly recommended that the pitch register be loaded with a value of 142_{16} 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 202_{16} or higher is recommended. The requirement that the pitch register value be less than or equal to $1FFE_{16}$ and the recommendation of a value greater than or equal to 142_{16} result in a pitch range of 39 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 42_{16} and $3FE_{16}$. If this is not done, problems with the TSP chip may occur which are not apparent with the TSE chip.

2.16 Speech Address Register

The speech address register (SAR) is a 14-bit register that is used to point to data in internal ROM. The LUAPS instruction transfers the value in A to the speech address register and loads the parallel-to-serial register (see Section 2.17) with the internal ROM value pointed to by the SAR. The GET instruction can then be used to 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

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

2.17 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 internal ROM pointed to by the speech address register, internal RAM pointed to by the X register, or external TSP60C18 speech ROM pointed to by the SAR in the TSP60C18. The LUAPS instruction is used to initialize the parallel-to-serial register and zero its bit counter. GET instructions can then be used to transfer one to eight 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 RAM, however, the X register is not automatically incremented. The EXTROM and RAMROM bits in the mode register control the source for the parallel-to-serial register. See the speech address register description in Section 2.16 for more information.

Parallel-to-Serial Register

7	6	5	4	3	2	1	0

2.18 Input/Output Ports

Ten bidirectional lines – 8-bit Port A and 2-bit Port B – are available for interfacing with external devices. 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 two bits of Port B are available on the outside of the chip, and the states of the upper six bits of its registers are undetermined. Transfers from any of the I/O port registers to the A register leave the upper six bits (bits 13–9) of the A register undetermined. Details of the I/O registers are shown in Table 2–2.

Table 2–2. I/O Registers

REGISTER	TYPE	LOCATION†	
		PORT A	PORT B
Data Input Register (DIR)	Read Only	80	84
Pullup Enable Register (PER)	Read/Write	81	85
Data Direction Register (DDR)	Read/Write	82	86
Data Output Register (DOR)	Read/Write	83	87

† For the TSP50C12, the register locations are F0–F7.

DESIRED PIN FUNCTION	DOR	DDR	PER	PIN 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 level on the I/O pin, which is true even when the DDR is set for 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 pins set for open drain with a 1 written to the data register.

Leaving a high-impedance I/O pin unconnected could cause power consumption to rise while the processor is in run mode. The power consumption will be 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 standby mode, unconnected high-impedance pins have no effect on either power consumption or device functionality.

The I/O can also be put in slave mode, making the TSP50C1x usable as a peripheral to a host microprocessor. Port A can be connected to an 8-bit data bus and controlled by $R/\overline{W}$ (Port B1) and chip enable (Port B0). A read ($R/\overline{W}$ high and chip enable low) puts the Port A output latch values out on Port A. A write ($R/\overline{W}$ 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 that are to be used on the data bus in this mode should be configured as inputs.

In external ROM mode, the TSP50C1x can be interfaced easily to a TSP60C18 speech ROM. Port B0 is used as a chip enable strobe output to the TSP60C18, and Port A7 is used as a clock. Port A0 to A3 are used for address and data transfer, and one other bit must be used for read/write control of the TSP60C18.

When the two-pin push-pull option is selected for the D/A output on the TSP50C10/11/12, Port B1 is used for the second D/A pin, making it unavailable for I/O. In this case, no attempt should be made to use the B1 interrupt.

If the PCM and LPC mode register bits are both cleared, a high-to-low transition on B1 causes a level-1 interrupt. This can be used to generate an interrupt with an external event.

2.19 Mode Register

The mode register (Table 2–3) is an 8-bit write-only register that controls the operating mode of the TSP50C1x. When the $\overline{INIT}$ pin goes low, all mode register bits are cleared. The mode register is not saved during a subroutine call or interrupt.

2.20 Speech Synthesizer

The task of generating synthetic speech is divided between the programmable microprocessor and the dedicated speech synthesizer. The four speech synthesizer modes, which are set by the LPC and PCM bits in the mode register, are discussed in the following paragraphs.

Table 2–3. Mode Register

7	6	5	4	3	2	1	0
UNV	MASTER	RAMROM	EXTROM	ENA2	PCM	LPC	ENA1

BIT NAME	BIT LOW	BIT HIGH
ENA1	Disables level-1 interrupt	Enables level-1 interrupt
LPC	Disables LPC processor – all instruction cycles used by the microprocessor.	Enables LPC processor – 53% of instruction cycles dedicated to LPC synthesis.
PCM	Disables PCM mode. Level-1 interrupt is either PPC < 200_{16} in LPC mode or pin B1 otherwise.	Enables PCM mode. LPC high causes an interrupt rate of fosc/960 and microprocessor control of LPC excitation value. LPC low causes an interrupt rate of fosc/480 and microprocessor control of D/A register.
ENA2	Disables level-2 interrupt	Enables level-2 interrupt
EXTROM	Disables operation of external ROM hardware interface.	Enables operation of external ROM hardware interface.
RAMROM	Enables data source for GET instructions to be either internal or external ROM.	Enables data source for GET instructions to be internal RAM.
MASTER	Enables I/O master operation. All available I/O pins are controlled by internal microprocessor.	Enables I/O slave operation. Pin B0 becomes hardware chip enable strobe, and B1 becomes $R/\overline{W}$. Port A is controlled by B0 and B1.
UNV	Enables pitch-controlled excitation sequence when in LPC mode (PCM low, voiced).	Enables random excitation sequence when in LPC mode (PCM low, unvoiced).

2.20.1 Synthesizer Mode 0 – OFF

When the PCM and LPC bits are both cleared, the synthesizer is disabled. All instruction cycles are devoted to the microprocessor.The TASYN instruction transfers the A register to the pitch register, making it easy to load the pitch register before starting the LPC synthesizer. In this mode, the level-1 interrupt is triggered by a high-to-low transition on pin B1.

2.20.2 Synthesizer Mode 1 – LPC

This is the normal speaking mode. The TASYN instruction loads the pitch register, and the level-1 interrupt is triggered by the pitch register going below 200_{16}. Fifty-three percent of the instruction cycles are used by the synthesizer.

The microprocessor controls speech synthesis by unpacking and decoding parameters, by setting the update interval (frame rate), and by 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. Speech parameter input is received from dedicated space in the microprocessor RAM, and speech samples are generated at 8 kHz or 10 kHz. Communication between the microprocessor and the speech synthesizer takes place via a shared memory space in the microprocessor RAM. Refer to Section 6 for more information.

2.20.3 Synthesizer Mode 2 – PCM

This mode is used for tone and music generation or for very-high-bit-rate speech. The microprocessor uses all the instruction cycles, and the TASYN instruction transfers the A register directly to the D/A register. The level-1 interrupt occurs at a rate twice the speech sample rate (16 kHz or 20 kHz), giving access to the unfiltered D/A output.

2.20.4 Synthesizer Mode 3 – PCM and LPC

When both the PCM and LPC bits are set, the LPC synthesizer runs normally with its excitation function 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. This mode is included for use with RELPS (Residual Encoded Linear Predictive Synthesis) and similar techniques. The synthesizer takes 50% of the instruction cycles in this mode.

2.20.5 Use of RAM by the Synthesizer

The synthesizer uses locations 01 to 0F in the RAM. When synthesis is taking place, the parameters for the synthesizer come directly from these RAM locations. The addresses are shown in Figure 2–4.

Address	11	10	9	8	7	6	5	4	3	2	1	0	Comments
00													not used for synthesis
01													Energy
02													K12 (LPC-12 values)
03													K11
04													K10
05													K9
06													K8
07													K7
08													K6
09													K5
0A													K4
0B													K3
0C													K2
0D													K1
0E													C1 (low-pass filter)
0F													C2

Figure 2–4. RAM Map During Speech Generation

2.20.6 Frame-Length Control

The frame length is controlled by: a) The value put into the prescale register and b) 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, so the prescale value should be selected to give the proper frame duration based on the timer's range.

2.20.7 Digital-to-Analog Converter

The TSP50C1x contains an internal digital-to-analog converter (DAC) connected to the output of the synthesizer. The DAC is available in three pulse-width-modulated forms for the TSP50C10/11 and two pulse-width modulated forms for the TSP50C12/14. See Section 1.6 for more information. The DAC outputs samples at a rate given by fosc/480. For a 9.6-MHz oscillator, this results in an output sample rate of 20 kHz. For a 7.68-MHz oscillator, 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. This state is the same as a zero state for the two-pin and double-ended one-pin modes, but in the single-pin single-ended mode, the DAC goes to the maximum negative value. This fact must be taken into account to minimize clicks during speech.

2.21 Interrupts

The TSP50C1x has two interrupts: interrupt-1 and interrupt-2. Both are enabled and disabled by bits in the mode register. Interrupt-1 is a synthesis interrupt and has a higher priority. It also has more hardware support. When an interrupt-1 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. The mode register is not saved and restored during interrupts. 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.

Interrupt-1 is caused by 1 of 4 conditions depending on the state of the two mode-register bits PCM and LPC. These conditions, as well as the interrupt routine start address for each case, are shown in Table 2–4.

Table 2–4. Interrupt-1 Vectors

ADDRESS	PCM	LPC	INTERRUPT TRIGGER
0018_{16}	0	1	Pitch counter less than 200_{16} (see Section 2.15)
$001A_{16}$	0	0	Pin B1 goes from high to low (see Section 2.18)
$001C_{16}$	1	1	fosc/960 clock (see Section 2.20.4)
$001E_{16}$	1	0	fosc/480 clock (see Section 2.20.3)

Interrupt-2 has a lower priority and cannot interrupt the interrupt-1 routine. It can be interrupted by interrupt-1. 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. Interrupt-2 is always caused by a timer underflow – the timer going from 0 to FF – but it starts at different addresses depending on the state of two mode-register bits. Table 2–5 shows the interrupt-2 vectors.

Table 2–5. Interrupt-2 Vectors

ADDRESS	PCM	LPC	INTERRUPT TRIGGER
0010_{16}	0	1	All level-2 interrupts caused by timer underflow
0012_{16}	0	0	
0014_{16}	1	1	
0016_{16}	1	0	

NOTE: All addresses in this manual are in hexadecimal format unless otherwise noted. All other numbers are in decimal format unless otherwise noted.

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

Interrupts will not be 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 BRANCH, BRA, CALL, or SBR to itself) should be avoided since an interrupt will never be 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 will be 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 may 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 will remain in effect after the return, including the enabling and disabling of interrupts.

2.22 TSP50C12 LCD Functional Description

The LCD functionality of the TSP50C12 was included without adding instructions to the instruction set. An additional 192 bits of RAM were added to serve as the display RAM. The display RAM is physically placed at RAM addresses 80 – 97. As a result, Port A's registers are mapped from F0 to F3 and Port B's registers are mapped from F4 to F7. This RAM mapping is consistent with the SE50C10 emulator device used in the extended RAM mode (pin controllable).

When data is stored into the display RAM locations, it may immediately affect the voltage levels on the LCD segment outputs. Because the microprocessor access of RAM is time multiplexed with LCD access, there will be no asynchronous ambiguities on segment outputs. If the display RAM update routines are slow, it may be necessary to buffer the display data in another area of RAM and then transfer it to the display RAM in a more time efficient block move.

An LCD voltage reference generator is also included on the TSP50C12. This circuit eliminates the need for an external voltage reference generator.

2.22.1 TSP50C12 LCD Driver

The TSP50C12 can drive an 8 x 24 (192-segment) LCD display with 1/8 duty cycle. The driver function for the LCD is controlled by internal timing hardware. Display data for the LCD is stored in a dedicated section of RAM. This data is stored in pixel form with 24 consecutive 8-bit words. Table 2–6 shows the memory locations for each pixel.

Table 2–6. TSP50C12 Display RAM Map

ADDRESS	msb							lsb
80	S23c1	S22c1	S21c1	S20c1	S19c1	S18c1	s17c1	S16c1
81	S15c1	S14c1	S13c1	S12c1	S11c1	S10c1	S9c1	S8c1
82	S7c1	S6c1	S5c1	S4c1	S3c1	S2c1	S1c1	S0c1
83	S23c2	S22c2	S21c2	S20c2	S19c2	S18c2	s17c2	S16c2
84	S15c2	S14c2	S13c2	S12c2	S11c2	S10c2	S9c2	S8c2
85	S7c2	S6c2	S5c2	S4c2	S3c2	S2c2	S1c2	S0c2
86	S23c3	S22c3	S21c3	S20c3	S19c3	S18c3	s17c3	S16c3
87	S15c3	S14c3	S13c3	S12c3	S11c3	S10c3	S9c3	S8c3
88	S7c3	S6c3	S5c3	S4c3	S3c3	S2c3	S1c3	S0c3
89	S23c4	S22c4	S21c4	S20c4	S19c4	S18c4	s17c4	S16c4
8A	S15c4	S14c4	S13c4	S12c4	S11c4	S10c4	S9c4	S8c4
8B	S7c4	S6c4	S5c4	S4c4	S3c4	S2c4	S1c4	S0c4
8C	S23c5	S22c5	S21c5	S20c5	S19c5	S18c5	s17c5	S16c5
8D	S15c5	S14c5	S13c5	S12c5	S11c5	S10c5	S9c5	S8c5
8E	S7c5	S6c5	S5c5	S4c5	S3c5	S2c5	S1c5	S0c5
8F	S23c6	S22c6	S21c6	S20c6	S19c6	S18c6	s17c6	S16c6
90	S15c6	S14c6	S13c6	S12c6	S11c6	S10c6	S9c6	S8c6
91	S7c6	S6c6	S5c6	S4c6	S3c6	S2c6	S1c6	S0c6
92	S23c7	S22c7	S21c7	S20c7	S19c7	S18c7	s17c7	S16c7
93	S15c7	S14c7	S13c7	S12c7	S11c7	S10c7	S9c7	S8c7
94	S7c7	S6c7	S5c7	S4c7	S3c7	S2c7	S1c7	S0c7
95	S23c8	S22c8	S21c8	S20c8	S19c8	S18c8	s17c8	S16c8
96	S15c8	S14c8	S13c8	S12c8	S11c8	S10c8	S9c8	S8c8
97	S7c8	S6c8	S5c8	S4c8	S3c8	S2c8	S1c8	S0c8

NOTES: S–Segment or pixel on a given row (common time)
c:–Row (common time)
All addresses in this manual are in hexadecimal format unless otherwise noted. All other numbers are in decimal format unless otherwise noted.

2.22.2 TSP50C12 LCD Drive Type A

The Type A drive method places limitations on the series resistance and pixel capacitance of the display. This drive type requires a more complex LCD display. The Type A option must be selected by the customer and given to TI before releasing the device for mask tooling.

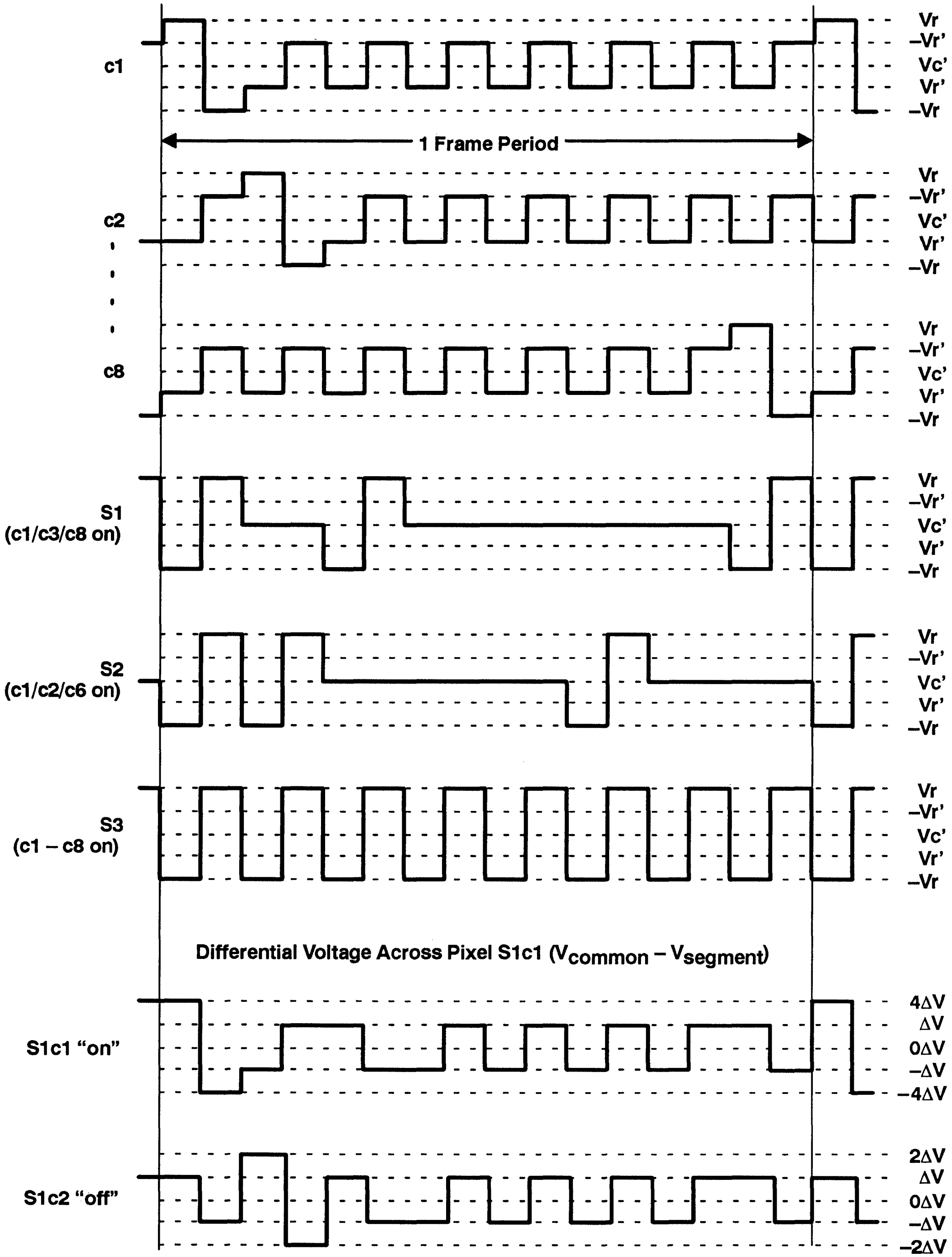


Figure 2–5. TSP50C12 LCD Driver Type A Timing Diagram

2.22.3 TSP50C12 LCD Drive Type B

The Type B drive method operates at a lower frequency, allowing the common signal to go high on the first frame and to go low on the next frame. This option is preferred for applications that have large capacitance pixel loads and high series trace resistances. This method also might be used if the microprocessor is operated at higher frequencies. The Type B option must be selected by the customer and given to TI before releasing the device for mask tooling.

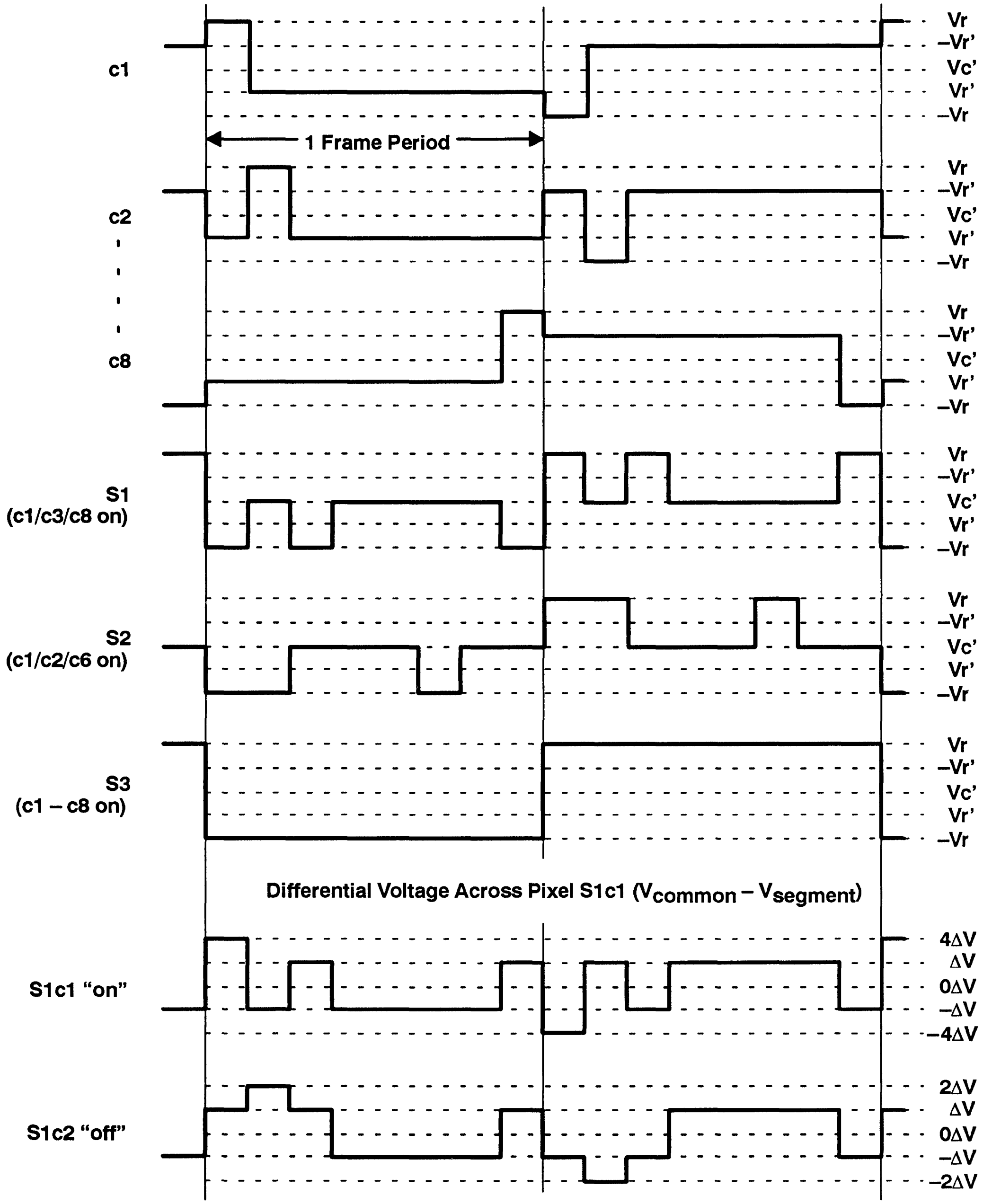


Figure 2–6. TSP50C12 LCD Driver Type B Timing Diagram

2.23 TSP50C12 LCD Reference Voltage and Contrast Adjustment

The TSP50C12 contains an internal voltage reference generator to regulate and adjust the LCD reference voltages. The voltage generator is comprised of a voltage doubler, a bandgap reference, a voltage regulator, and a final trim DAC. V_{LCD} provides an isolated voltage supply for the voltage doubler. V_{LCD} can be connected to V_{DD} or, for example, can be connected to a 4.5-V tap of a 4-cell battery supply to improve the power efficiency of the circuit. An external capacitor should be connected between V_{C1} and V_{C2}. An external capacitor should be connected between V_{X2} and V_{LCD}. The bandgap provides a reference voltage for the voltage regulator. The voltage regulator has a nominal output of 4.9 V (±200 mV). The reference voltage can be trimmed by writing to the DAC (memory-mapped to the lower four bits at RAM location 98). The trim control will range from −8 steps (0000) to +7 steps (1111) from nominal with each step being approximately 100 mV. The value of this RAM location will not be initialized and must be set by the initialization software routine.

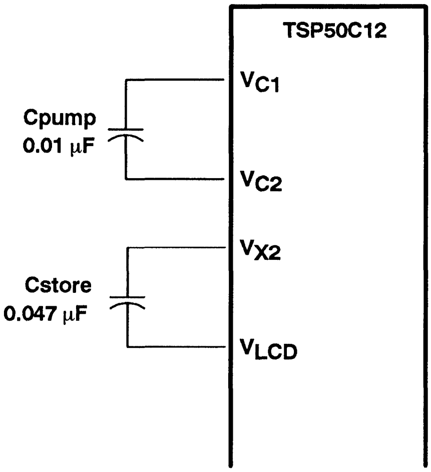


Figure 2–7. TSP50C12 Voltage Doubler

2.24 TSP50C12 Clock Options

The RC oscillator requires a single external resistor between V_{DD} and OSC1 with OSC2 left unconnected to set the operating frequency. The frequency shift, as V_{DD} changes, is limited to 10% over the operating range of 4 V to 6.5 V. The center frequency as a function of resistance will require trimming. For applications requiring greater clock precision, a ceramic resonator option is also available. The RC oscillator/ceramic resonator selection must be made by the customer and given to TI before releasing the device for mask tooling.

3 TSP50C1x Electrical Specifications

3.1 Absolute Maximum Ratings Over Operating Free-Air Temperature Range

Supply voltage range, V_{DD} (see Note 1)	–0.3 V to 8 V
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
Operating free-air temperature range, T_A	0°C to 70°C
Storage temperature range	−30°C to 125°C

NOTE 1: All voltages are with respect to ground.

3.2 TSP50C1x Recommended Operating Conditions

			MIN	NOM	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		0.8	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
		LCD reference spec (TSP50C12 only)	10		40	
f_{osc}	Clock frequency	10-kHz speech sample rate‡		9.6		MHz
		8-kHz speech sample rate‡		7.68		
f_{clock}	ROM clock frequency	External ROM mode interface to TSP60C18 speech ROMs		$f_{osc}/4$		MHz

† Unless otherwise noted, all voltages are with respect to V_{SS}.
‡ Speech sample rate = $f_{osc}/960$.

3.3 TSP50C1x D/A Options Timing Requirements

			MIN	NOM	MAX	UNIT
t_r	Rise time, PA, PB, D/A options 1, 2	V_{DD} = 4 V, C_L = 100 pF		22		ns
t_f	Fall time, PA, PB, D/A options 1, 2			10		ns

3.4 TSP50C1x Initialization Timing Requirement

		MIN	MAX	UNIT
t_{INIT}	$\overline{INIT}$ pulsed low while the TSP50C1x has power applied	1		µs

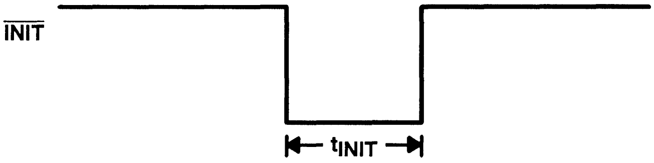


Figure 3–1. Initialization Timing Diagram

3.5 TSP50C1x Write Timing Requirements (Slave Mode)

		MIN	MAX	UNIT
$t_{su(B1)}$	Setup time, B1 low before B0 goes low	20		ns
$t_{su(d)}$	Setup time, data valid before B0 goes high	100		ns
$t_{h(B1)}$	Hold time, B1 low after B0 goes high	20		ns
$t_{h(d)}$	Hold time, data valid after B0 goes high	30		ns
t_w	Pulse duration, B0 low	100		ns
t_r	Rise time, B0		50	ns
t_f	Fall time, B0		50	ns

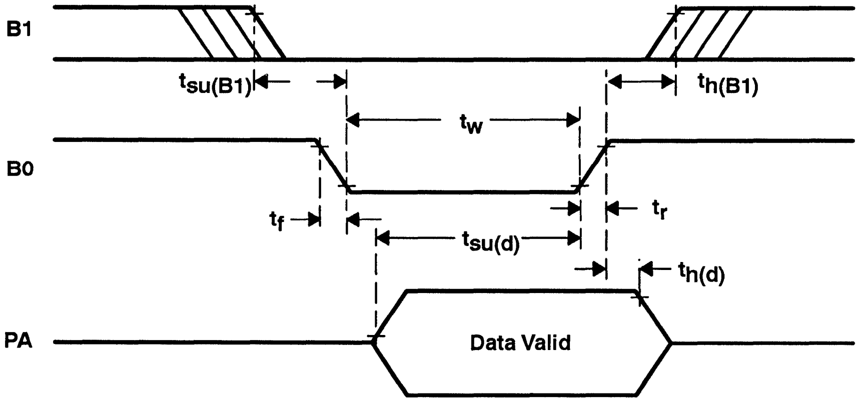


Figure 3–2. Write Timing Diagram (Slave Mode)

3.6 TSP50C1x Read Timing Requirements (Slave Mode)

		MIN	MAX	UNIT
$t_{su(B1)}$	Setup time, B1 before B0 goes low	20		ns
$t_{h(B1)}$	Hold time, B1 after B0 goes high	20		ns
t_{dis}	Output disable time, data valid after B0 goes high	0	30	ns
t_w	Pulse duration, B0 low	100		ns
t_r	Rise time, B0		50	ns
t_f	Fall time, B0		50	ns
t_d	Delay time for B0 low to data valid		50	ns

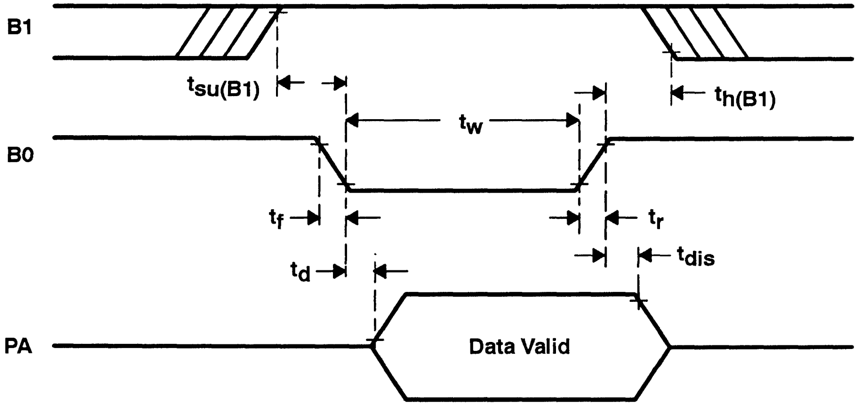


Figure 3–3. Read Timing Diagram (Slave Mode)

3.7 TSP50C10/11 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	V_{DD} = 4.5 V			2.7		V
		V_{DD} = 6 V			3.65		
V_{T-}	Negative-going threshold voltage	V_{DD} = 4.5 V			2.3		V
		V_{DD} = 6 V			3.15		
V_{hys}	Hysteresis ($V_{T+} - V_{T-}$)	V_{DD} = 4.5 V			0.4		V
		V_{DD} = 6 V			0.5		
I_{lkg}	Input leakage current					1	µA
$I_{standby}$	Standby current ($\overline{\text{INIT}}$ low)					10	µA
I_{DD}†	Supply current	D/A option 1, 2, or 3			5		mA
I_{OH}	High-level output current (PA, PB, D/A options 1, 2)	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} = 2.67 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 (PA, PB, D/A options 1, 2)	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		
	Pull-up resistance	Resistors selected with software and connected between pin and V_{DD}		15	30	60	kΩ

† 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.

3.8 TSP50C12 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		V_{DD} = 4.5 V		2.7		V
			V_{DD} = 6 V		3.65		
V_{T-}	Negative-going threshold voltage		V_{DD} = 4.5 V		2.3		V
			V_{DD} = 6 V		3.15		
V_{hys}	Hysteresis ($V_{T+} - V_{T-}$)		V_{DD} = 4.5 V		0.4		V
			V_{DD} = 6 V		0.5		
	LCD reference voltages	V_r	DAC register = 1000, T_A = 25°C, See Figures 2−5 and 2−6	4.7	4.9	5.1	V
		$-V_r'$		3.717	3.875	4.033	
		V_c'		2.734	2.85	2.966	
		V_r'		1.751	1.825	1.899	
		$-V_r$		0.767	0.8	0.833	
V_r	LCD temperature coefficient †		T_A = 0°C to 40°C		–2.5		mV/°C
	DAC step		DAC step control of V_r with respect to $-V_r$, V_{DD} = 5 V, T_A = 25°C	74	100	124	mV
I_{lkg}	Input leakage current					1	µA
$I_{standby}$	Standby current ($\overline{INIT}$ low)					10	µA
I_{DD}‡	Supply current		D/A option 1 or 3		5		mA
I_{OH}	High-level output current (PA, PB, D/A options 1, 2)		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} = 2.67 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 (PA, PB, D/A options 1, 2)		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		
	Pull up resistance		Resistors selected with software and connected between pin and V_{DD}	15	30	60	kΩ
	DAC buffer drive (D/A option 1)		32-Ω load connected across DA1 and DA2, VDD = 4.5 V		60		mA
	LCD frame rate		f_{OSC} = 9.6 MHz		96		Hz

† This negative temperature coefficient is normally advantageous because it tracks the temperature variation of most LCD materials.

‡ 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.

3.9 TSP50C14 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	V_{DD} = 4.5 V		2.7		V
		V_{DD} = 6 V		3.65		
V_{T-}	Negative-going threshold voltage	V_{DD} = 4.5 V		2.3		V
		V_{DD} = 6 V		3.15		
V_{hys}	Hysteresis ($V_{T+} - V_{T-}$)	V_{DD} = 4.5 V		0.4		V
		V_{DD} = 6 V		0.5		
I_{lkg}	Input leakage current				1	µA
$I_{standby}$	Standby current ($\overline{\text{INIT}}$ low)				10	µA
I_{DD}†	Supply current	DAC option 1 or 2		5		mA
I_{OH}	High-level output current (PA, PB, D/A options 1, 2)	V_{DD} = 4 V, V_{OH} = 3.5 V	–27	–41		mA
		V_{DD} = 5 V, V_{OH} = 4.5 V	–34	–51		
		V_{DD} = 6 V, V_{OH} = 5.5 V	–41	–63		
		V_{DD} = 4 V, V_{OH} = 2.67 V	–54	–88		mA
		V_{DD} = 5 V, V_{OH} = 3.33 V	–95	–136		
		V_{DD} = 6 V, V_{OH} = 4 V	–136	–197		
I_{OL}	Low-level output current (PA, PB, D/A options 1, 2)	V_{DD} = 4 V, V_{OL} = 0.5 V	27	41		mA
		V_{DD} = 5 V, V_{OL} = 0.5 V	34	51		
		V_{DD} = 6 V, V_{OL} = 0.5 V	41	63		
		V_{DD} = 4 V, V_{OL} = 1.33 V	54	88		mA
		V_{DD} = 5 V, V_{OL} = 1.67 V	95	136		
		V_{DD} = 6 V, V_{OL} = 2 V	136	197		
	Pullup resistance	Resistors selected with software and connected between pin and V_{DD}	15	30	60	kΩ
f_{osc}	Oscillator frequency‡	7.68-MHz target frequency, V_{DD} = 4 V, T_A = 25°C	7.21	7.68	8.15	MHz
		9.6-MHz target frequency, V_{DD} = 4 V, T_A = 25°C	9.02	9.6	10.2	

† 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.

4 TSP50C1x Assembler

4.1 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 keyword is given in capital letters.

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

EXAMPLE

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

4.2 Invoking the Assembler

The assembler is invoked by typing:

ASM10 ^ [<options>] ^ <source[.ext]>

where:

Options represents a list of assembler options (see Section 4.2.1).

Source is the name of the source file with the extension optional.

If the extension is not given, then the default extension of ".ASM" is

assumed.

For example:

```
ASM10 –l PROGRAM
```

would run the assembler using the source file "PROGRAM.ASM" and would generate the output object file "PROGRAM.BIN". No list file would be generated.

4.2.1 Command-Line Options

Several options can be invoked from the command line (Table 4–1). They are invoked by listing their abbreviation prefixed by a minus sign. For example:

```
ASM10 –Io PROGRAM.ASM
```

would assemble the program in file "PROGRAM.ASM" but would not generate either a listing file or an object file; however, any errors would be written to the console. The available options are detailed below. See Section 4.4.10 for information on invoking options from within the source code.

Table 4–1. Switches and Options

CHARACTER OR NUMBER	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
I or i	Counts the number of times a valid instruction has been used
L or l	Displays error messages without generating a list
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 listing file is generated
T or t	Lists only the first data byte in TEXT or RTEXT
W or w	Suppresses the warning message
X or x	Adds a cross-reference list at the end
9	Generates object file in TI-990 tagged object format

4.2.1.1 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. If the BYTE unlist switch is set, then only the first line (which also contains the source line listing) is written to the listing file.

4.2.1.2 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. If the DATA unlist switch is set, then only the first line (which also contains the source line listing) is written to the listing file.

4.2.1.3 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.

4.2.1.4 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 in the listing file. 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 listing. If the TEXT unlist switch is set, only the first line (also containing the source line listing) is written to the list file.

4.2.1.5 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.

4.2.1.6 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.

4.2.1.7 Object Module Switch

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

4.2.1.8 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.

4.2.1.9 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 may still be forced using the PAGE command.

4.2.1.10 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.

4.2.1.11 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 was used in the program.

4.2.1.12 Binary Code File Disable Switch

Placing a 9 in the command-line option field causes the assembler to generate the object module in tagged object format in a file with a ".MPO" extension instead of the normal binary formatted object module in a file with a ".BIN" extension.

4.2.2 Assembler Input and Output Files

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

4.2.2.1 Assembly Source File

The assembly source file is specified in the command line. If the file name given 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:

```
ASM10 PROGRAM.SRC
```

uses the file PROGRAM.SRC as the assembly source file.

```
ASM10 PROGRAM
```

uses the file PROGRAM.ASM as the assembly source file.

4.2.2.2 Assembly Binary Object File

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

For example:

```
        ASM10 PROGRAM.SRC
```

uses the file "PROGRAM.SRC" as the assembly source file and the file "PROGRAM.BIN" as the binary object output file.

```
        ASM10 –O PROGRAM.SRC
```

uses the file PROGRAM.SRC as the assembly source file and produces no object output.

4.2.2.3 Assembly Tagged Object File

If desired, the assembler can substitute an object file in tagged object format instead of the object file in binary format. If produced, the object output is placed in a file with the same file name as the assembly source except that the extension is ".mpo".

For example:

```
        ASM10 –9 PROGRAM.SRC
```

uses the file "PROGRAM.SRC" as the assembly source file and the file "PROGRAM.MPO" as the tagged object output file. No binary-formatted object file is produced.

4.2.2.4 Assembly 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:

```
        ASM10 PROGRAM.SRC
```

uses the file "PROGRAM.SRC" as the assembly source file and the file "PROGRAM.LST" as the assembly listing file.

4.3 Source-Statement Format

An assembly-language source program consists of source statements contained in the assembly source file(s) that may contain assembler directives, machine instructions, or comments. Source statements may 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 may be as long as 80 characters. If the form width is set to 80 characters (the default), the assembler will truncate 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 is treated as a comment in its entirety and as such is ignored by the assembler and has no effect on the assembly process.

The syntax of the source statements is:

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

A source statement may 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.

4.3.1 Label Field

The label field begins in character position one of the source line. If position one is a character other that a blank or an asterisk, the assembler assumes that the symbol is a label. If a label is omitted, then the first character position must be a blank. The label may 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 9 character positions may be any of the legal characters listed above.

4.3.2 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 may not extend past the sixtieth character position. The command field may contain either an assembler mnemonic (e.g., TAX) or an assembler 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.

4.3.3 Operand Field

The operand field begins following the blank that terminates the command field and may not extend past the sixtieth column position. The operand may contain one or more constants or expressions described below. Terms in the operand field are separated by commas. The operand field is terminated by the first blank encountered.

4.3.4 Comment Field

The comment field begins after the blank that terminates the operand field or the blank that terminates the command field if no operand is required. The comment field may extend to the end of the source record and may contain any ASCII character including blanks.

4.3.5 Constants

The assembler recognizes the following five types of constants:

Decimal integer constants

Binary integer constants

Hexadecimal integer constants

Character constants

Assembly-time constants

4.3.5.1 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 to 1000 or #03E8
P32768	Constant equal to –32768 or #8000
25	Constant equal to 25 or #0019

4.3.5.2 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 (?). If 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 to 19 or #0013
?0111111111111111	Constant equal to 32767 or #7FFF
?11110	Constant equal to 30 or #001E

4.3.5.3 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 (>). If 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:

#7F	Constant equal to 127 (or #007F)
>7f	Constant equal to 127 (or #007F)
#307a	Constant equal to 12410 (or #307A)

4.3.5.4 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. If 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 0000 (Hex).

The following are valid character constants:

’AB’	Constant equal to #4142
’C’	Constant equal to #0043
’’’D’	Constant equal to #2744

4.3.5.5 Assembly-Time Constants

An assembly-time constant is a symbol given a value by an EQU directive (see Section 4.4.5). The value of the symbol is determined at assembly time and may be assigned values with expressions using any of the above constant types.

4.3.6 Symbols

Symbols are used in the label field and the operand field. A symbol is a string of ten or less alphanumeric characters (a through z, A through Z, 0 through 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 may 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 may 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
```

would result in a branch to six locations beyond the current location.

4.3.7 Character String

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 may 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 as 8-bit ASCII.

The following are valid character strings:

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

4.3.8 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 may 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.

4.3.8.1 Arithmetic Operators in Expressions

The following arithmetic operators may be used in an expression :

+	for addition
–	for subtraction
*	for multiplication
/	for division (remainder is truncated)
%	for modulo (remainder after division)
&	for bitwise AND
++	for bitwise OR
&&	for 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 or unary minus (so that the expression 4+5*2 would be evaluated as 18, not 14).

4.3.8.2 Parentheses In Expressions

The assembler supports the use of parentheses in expressions to alter the order of evaluating the expression. Nesting of pairs of 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 may not be nested more than eight deep.

4.4 Assembler Directives

Assembler directives (Table 4–2) 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 4–2. Summary of Assembler Directives

<table>
<tr><th colspan="3">DIRECTIVES THAT AFFECT THE LOCATION COUNTER</th></tr>
<tr><th>Mnemonic</th><th>Directive</th><th>Syntax</th></tr>
<tr><td>AORG</td><td>Absolute origin</td><td>[&lt;label&gt;]^AORG^<expression>^[<comment&gt;]</td></tr>
<tr><th colspan="3">DIRECTIVES THAT AFFECT ASSEMBLER OUTPUT</th></tr>
<tr><th>Mnemonic</th><th>Directive</th><th>Syntax</th></tr>
<tr><td>IDT</td><td>Program identifier</td><td>[&lt;label>]^IDT^’<string>’^[<comment&gt;]</td></tr>
<tr><td>LIST</td><td>Restart source listing</td><td>[&lt;label&gt;]^LIST^&lt;expression>^[<comment&gt;]</td></tr>
<tr><td>NARROW</td><td>80-column form width</td><td>[&lt;label>]^NARROW^[<comment&gt;]</td></tr>
<tr><td>OPTION</td><td>Output options</td><td>[&lt;label&gt;]^OPTION^<option list>^[<comment&gt;]</td></tr>
<tr><td>PAGE</td><td>Page eject</td><td>[&lt;label>]^PAGE^[<comment&gt;]</td></tr>
<tr><td>TITL</td><td>Page title</td><td>[&lt;label&gt;]^TITL^’<string>’^[<comment&gt;]</td></tr>
<tr><td>UNL</td><td>Stop source listing</td><td>[&lt;label&gt;]^UNL^[<comment&gt;]</td></tr>
<tr><td>WIDE</td><td>130-column form width</td><td>[&lt;label>]^WIDE^[<comment&gt;]</td></tr>
<tr><th colspan="3">DIRECTIVES THAT INITIALIZE CONSTANTS</th></tr>
<tr><th>Mnemonic</th><th>Directive</th><th>Syntax</th></tr>
<tr><td>BYTE</td><td>Initialize byte</td><td>[&lt;label&gt;]^BYTE^<expr-1>^[,<expr-2>,....., <expr-n>]^[<comment&gt;]</td></tr>
<tr><td>RBYTE</td><td>Reverse bit initialization of byte</td><td>[&lt;label&gt;]^RBYTE^<expr-1>^[,<expr-2>,....., <expr-n>]^[<comment&gt;]</td></tr>
<tr><td>DATA</td><td>Initialize word</td><td>[&lt;label&gt;]^DATA^<expr-1>^[,<expr-2>,....., <expr-n>]^[<comment&gt;]</td></tr>
<tr><td>RDATA</td><td>Reverse bit initialization of word</td><td>[&lt;label>]^RDATA^<expr-1>^[,<expr-2>,....., <expr-n>]^[<comment&gt;]</td></tr>
<tr><td>EQU</td><td>Define assembly time</td><td>[&lt;label&gt;]^EQU^[<comment&gt;]</td></tr>
<tr><td>TEXT</td><td>Initialize text</td><td>[&lt;label>]^TEXT^[–]’<string>’^[<comment&gt;]</td></tr>
<tr><td>RTEXT</td><td>Reverse byte initialization of text</td><td>[&lt;label&gt;]^RTEXT^[–]’&lt;string>’^[<comment&gt;]</td></tr>
<tr><th colspan="3">MISCELLANEOUS DIRECTIVES</th></tr>
<tr><th>Mnemonic</th><th>Directive</th><th>Syntax</th></tr>
<tr><td>COPY</td><td>Copy source file</td><td>[&lt;label&gt;]^COPY^<filename>^[<comment&gt;]</td></tr>
<tr><td>END</td><td>Program end</td><td>[&lt;label&gt;]^END^<symbol>^[<comment&gt;]</td></tr>
</table>

4.4.1 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>]

EXAMPLE

```
AORG #1000+Offset
```

The symbol Offset must be previously defined. If Offset has a value of 8, the location counter is set to #1008 by this directive. Had a label been included, the label also would have been assigned the value of #1008.

4.4.2 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 127. 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>]

EXAMPLE

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

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

4.4.3 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 an 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 ^ <file name> ^ [<comment>]

EXAMPLE

```
COPY copy.fil
```

The directive in the example 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 found in copy.fil, the assembler resumes processing source statements from the original source file.

4.4.4 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 bytes of program memory.

The syntax of the DATA directive is as follows:

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

EXAMPLE

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

The example places the following bytes into successive locations in program memory: #E0, #00, #41, #42

4.4.5 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>]

EXAMPLE

```
Offset EQU #100
```

The example assigns the numeric value of 256 (100 Hex) to the symbol Offset.

4.4.6 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. If it is found in a copy file, the copy file is closed and subsequent statements are taken from the source file. If 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>]

EXAMPLE

```
ACAAC 1
END
CLA
```

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

4.4.7 IDT Directive

The IDT directive 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>]

EXAMPLE

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

The 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. The module name is also placed in the object code (if the tagged object format code is being produced).

4.4.8 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>]

EXAMPLE

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

In the example, 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.

4.4.9 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 WIDE.)

The syntax of the NARROW directive is as follows:

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

EXAMPLE

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

4.4.10 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 available options are listed in the paragraphs below.

The syntax of the OPTION directive is as follows:

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

EXAMPLE

```
        OPTION    990,XREF,SCRNOF
        OPTION    990,XREF,SCREEN
        OPTION    9,X,S
```

The three examples above have an identical effect. The binary object file is replaced by an object file in tagged object format. 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 4.2.1 for information on invoking options from the command line.

4.4.10.1 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. If the directive has ten operands, ten lines are required in the source listing to list it. The BUNLST is used to avoid this.

4.4.10.2 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. If the directive has ten operands, ten lines are required in the source listing to list it. The DUNLST is used to avoid this.

4.4.10.3 FUNLST – Byte, Data, and Text Unlist Option

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

4.4.10.4 I COUNT – Instruction Count List Option

Placing any valid symbol starting with I or i in the option list causes the program to generate a table containing the number of times each valid instruction was used in the program. If used, it should be placed at the start of the program.

4.4.10.5 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.

4.4.10.6 OBJUNL – Object File Unlist Option

Placing any valid symbol starting with O or o in the option list enables the object file unlist option. This option inhibits either of the object output files from being produced.

4.4.10.7 PAGEOF – Page Break Inhibit Option

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

4.4.10.8 RXREF – Reduced XREF Option

Placing any valid symbol starting with R or r in the option list 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).

4.4.10.9 SCRNOF – Screen Error Message Unlist Option

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

4.4.10.10 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. The TUNLST causes only the first line of the directive listing to be produced.

4.4.10.11 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.

4.4.10.12 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. If used, it should be placed at the start of the program.

4.4.10.13 990 – Tagged Object Output Switch

Placing any valid symbol starting with 9 in the option list causes the assembler to omit the binary coded object module (normally produced as a .BIN file) and to produce a tagged object module (as a .MPO file) instead.

4.4.11 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>]

EXAMPLE

```
        AORG 10
Tl PAGE   Force Page Eject
```

In the example, 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.

4.4.12 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 127. 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>]

EXAMPLE

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

The value of the symbol data must be defined in the assembly process. The example places the numbers 7 (07 Hex), 160 (A0 Hex), and the bit-reversed result of the arithmetic operation (data+5) in successive bytes of program memory.

4.4.13 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>]

EXAMPLE

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

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

4.4.14 RTEXT Directive

The RTEXT directive writes an ASCII string to the object file in reverse order. If 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 high. 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>]

EXAMPLE

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

Both examples 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 high (that is, as a #D4 instead of a #54).

4.4.15 TEXT Directive

The TEXT directive writes an ASCII string to the object file. If the string is preceded by a minus sign, the last character in the string is written with its most significant bit set high. 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 in length enclosed in single quotes (optionally preceded by a minus sign).

The syntax of the TEXT directive is as follows:

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

EXAMPLE

```
TEXT –'This is a test'
TEXT 'This is a test'
```

Both examples 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 high (that is, as a #F4 instead of a #74)

4.4.16 TITL Directive

The TITL directive supplies 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's 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 that 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>]

EXAMPLE

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

The example 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 next 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.

4.4.17 UNL Directive

The UNL directive inhibits the 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 is optional.

When used, the label 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>]

EXAMPLE

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

In the example, the label T1 is assigned the value 10, and listing is inhibited.

4.4.18 WIDE Directive

The WIDE directive causes the assembler to assume a 130-column form width for the listing file. The default is 80 columns. (See NARROW.)

The syntax of the WIDE directive is as follows:

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

EXAMPLE

```
        AORG 10
T1      WIDE                Switch to 130—column listing format
```

5 Instruction Set

There are 61 different TSP50C1x instructions (Tables 5–1 and 5–2). Most of them require only one instruction cycle to execute, although a few require two. Each instruction cycle consists of 16 clock cycles; therefore, a clock speed of 9.6 MHz translates to 600,000 instructions per second. When LPC synthesis is enabled, every other instruction cycle is taken for synthesis calculations, and two additional cycles are used for excitation function look up. This causes the instruction cycle rate for the program to drop to 280,000 cycles per second.

Table 5–1. TSP50C1x Instruction Set

MNEMONIC	OPERAND SIZE (BITS)	INSTRUCTION CYCLES REQUIRED	STATUS (1 ALWAYS SET, C CONDITIONAL	NUMBER OF BYTES REQUIRED	OPCODE (HEX)	DESCRIPTION
ABAAC		1	C	1	2C	Add A to B
ACAAC	12	2	C	1	70	Add constant to A
AGEC	8	2	C	2	63	A greater than or equal to constant
AMAAC		1	C	1	28	Add memory to A
ANDCM	8	2	1	2	65	AND constant and memory
ANEC	8	2	C	2	60	A not equal to constant
AXCA	8	2	1	2	68	A times constant
AXMA		1	1	1	39	A times memory
AXTM		1	1	1	38	A 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
CLB		1	1	1	24	Clear B
CLX		1	1	1	20	Clear X
DECMN		1	C	1	27	Decrement memory
DECXN		1	C	1	22	Decrement X
EXTSG		1	1	1	3C	Extended-sign mode
GET	3	2	C	1	30	Get bits
IAC		1	C	1	3A	Increment A
IBC		1	C	1	25	Increment B
INCMC		1	C	1	26	Increment memory
INTGR		1	C	1	3B	Set integer mode
IXC		1	C	1	21	Increment X
LUAA		2	1	1	6B	Look up A, result to A
LUAB		2	1	1	6D	Look up A, result to B

Table 5–1. TSP50C1x Instruction Set (continued)

MNEMONIC	OPERAND SIZE (BITS)	INSTRUCTION CYCLES REQUIRED	STATUS (1 ALWAYS SET, C CONDITIONAL	NUMBER OF BYTES REQUIRED	OPCODE (HEX)	DESCRIPTION
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 left
SALA4		1	1	1	1B	Shift A left 4 bits
SARA		1	1	1	15	Shift A right
SBAAN		1	C	1	2D	Subtract B from A
SBR	7	1	1	1	80	Short branch if status set
SETOFF		1	1	1	3F	Turn processor off
SMAAN		1	C	1	29	Subtract memory from A
TAB		1	1	1	1A	Transfer A to B
TAM		1	1	1	16	Transfer A to memory
TAMD	8	2	1	2	6A	Transfer A to memory direct
TAMIX		1	1	1	13	Transfer A to memory, increment X
TAMODE		1	1	1	1D	Transfer A to mode register
TAPSC		1	1	1	19	Transfer A to prescale register
TASYN		1	1	1	1C	Transfer A to synthesizer register
TATM		1	1	1	1E	Transfer A to timer register
TAX		1	1	1	18	Transfer A to X
TBM		1	1	1	2A	Transfer B to memory
TCA	8	2	1	2	6E	Transfer constant to A
TCX	8	2	1	2	62	Transfer constant to X
TMA		1	1	1	11	Transfer memory to A
TMAD	8	2	1	2	69	Transfer memory to A direct
TMAIX		1	1	1	14	Transfer memory to A, increment X
TMXD	8	2	1	2	6F	Transfer memory direct to X
TRNDA		1	1	1	2B	Transfer random number to A
TSTCA	8	2	C	2	67	Test constant and A
TSTCM	8	2	C	2	66	Test constant and memory
TTMA		1	1	1	17	Transfer timer to A
TXA		1	1	1	10	Transfer X to A

Table 5–1. TSP50C1x Instruction Set (continued)

MNEMONIC	OPERAND SIZE (BITS)	INSTRUCTION CYCLES REQUIRED	STATUS (1 ALWAYS SET, C CONDITIONAL	NUMBER OF BYTES REQUIRED	OPCODE (HEX)	DESCRIPTION
XBA		1	1	1	12	Exchange A and B
XBX		1	1	1	23	Exchange B and X
XGEC	8	2	C	2	61	X greater than or equal to constant

Table 5–2. TSP50C1x Instruction Table

LSB	MSB								
	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
B	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

5.1 Instruction Format

The source code instruction format is:

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

The fields are:

a 10-character optional label field,

a 6-character opcode field,

an opcode-dependent operand field,

and an optional comment field.

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

5.2 ABAAC – Add B to A

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

SOURCE CODE: [<LABEL>]^ABAAC^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) + (B) → (A)

STATUS FLAG: 1 if there is a carry into bit eight 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 14 bits of the B register and A register.

5.3 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

SOURCE CODE: [<LABEL>]^ACAAC^<CONST12>^...[<COMMENT>]

OBJECT CODE:

	7	6	5	4	3	2	1	0	
INSTRUCTION	0	1	1	1					← 4 most significant bits of constant
CONSTANT	CONST12								← 8 least significant bits of constant

EXECUTION RESULTS: (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. If the processor is in integer mode (INTGR), then the addition is of a 12-bit unsigned number with a 14-bit unsigned number. If the processor is in extended-sign mode (EXTSG), then the 12-bit constant is sign extended to a 14-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
```

5.4 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. Sets the status flag if the contents of the lower 8 bits of the A register are greater than the operand.

OPCODE: 63

SOURCE CODE: [<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 RESULTS: (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., FF is greater than FE. Only the lower eight bits of the A register are compared to the 8-bit constant value. The upper 6 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
```

5.5 AMAAC – Add Memory to A Register

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

OPCODE: 28

SOURCE CODE: [<LABEL>]^AMAAC^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) + (*X) → (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 eight 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
```

5.6 ANDCM – Logical AND a Constant With Memory

ACTION: Bit-wise ANDs 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

SOURCE CODE: [<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 RESULTS: (*X) && CONST8 → (*X)

STATUS FLAG: 1 always

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
```

5.7 ANEC – A 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

SOURCE CODE: [<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 RESULTS: (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 eight 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
```

5.8 AXCA – A 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

SOURCE CODE: [<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 RESULTS: ((A) × CONST8)/128 → (A)

STATUS FLAG: 1 always

NOTE: The operation is performed independent of the arithmetic mode (EXTSG or INTGR) as a two's complement multiplication of all 14 bits of the A register and the 8-bit constant. The result is right shifted 7 bits so that the most significant 14 bits of the 21-bit result are available for further use.

EXAMPLE:

```
TCA    #3F    Load first value
AXCA   #1F    Multiply by second value
              (result is #0F)
```

5.9 AXMA – A 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

SOURCE CODE: [<LABEL>]^AXMA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: ((A) × (*X))/128 → (A)

STATUS FLAG: 1 always

NOTE: The operation is performed independent of the arithmetic mode (EXTSG or INTGR) as a two's complement multiplication of all 14 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 14 bits of the 21-bit result are available for further use.

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)
```

5.10 AXTM – A 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

SOURCE CODE: [<LABEL>]^AXTM^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: ((A) × (TM))/128 → (A)

STATUS FLAG: 1 always

NOTE: The operation is performed independent of the arithmetic mode (EXTSG/INTGR) as a two's complement multiplication of all 14 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 14 bits of the 21-bit result are available for further use.

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 decrementd)
```

5.11 BR – Branch If Status Set

ACTION: If the status flag is set to 1, the program counter is loaded with the address specified by the operand and execution proceeds from that address. If the status flag is set to 0, the instruction following the BR instruction executes.

OPCODE: 40

SOURCE CODE: [<LABEL>]^BR^<ADDR13>^...[<COMMENT>]

OBJECT CODE:

	7	6	5	4	3	2	1	0	
INSTRUCTION	0	1	0						← 5 most significant bits of destination address
CONSTANT	ADDR13								← 8 least significant bits of destination address

EXECUTION RESULTS: if SF = 1, then ADDR13 → Program Counter

if SF = 0, then Program Counter → Program Counter

STATUS FLAG: 1 always

NOTE: The branch instruction is a conditional instruction. When a branch is used following an instruction that always leaves the status flag set high, 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 below.

EXAMPLE:

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

5.12 BRA – Branch Always to Address in A Register

ACTION: The program counter is loaded with the 14-bit address contained in the A register, and execution proceeds from that address.

OPCODE: 1F

SOURCE CODE: [<LABEL>]^BRA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → Program Counter

STATUS FLAG: 1 always

NOTE: 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.

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
```

The BRA instruction is an unconditional instruction. The branch is always taken, regardless of the value of the status register.

NOTE: 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 may be changed during the transfer and unexpected results obtained.

5.13 CALL – Call Subroutine If Status Set

ACTION: If the status flag is 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 0, the instruction following the CALL instruction executes.

OPCODE: 00

SOURCE CODE: [<LABEL>]^CALL^<ADDR12>^...[<COMMENT>]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION	0	0	0	0				
CONSTANT	ADDR12							

EXECUTION RESULTS: If SF = 1, then Program Counter → Stack, and ADDR12 → Program Counter

If SF = 0, then Program Counter → Program Counter

STATUS FLAG: 1 always

NOTE: The program counter stack is capable of storing addresses up to three 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 three levels deep, the last three addresses pushed to the stack are retained, and previous addresses are lost.

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. 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 to the lower 8K bytes 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.

5.14 CLA – Clear A Register

ACTION: Sets the contents of the A register to 0.

OPCODE: 2F

SOURCE CODE: [<LABEL>] ^CLA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: 0 → (A)

STATUS FLAG: 1 always

5.15 CLB – Clear B Register

ACTION: Sets the contents of the B register to 0.

OPCODE: 24

SOURCE CODE: [<LABEL>]^CLB^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: 0 → (B)

STATUS FLAG: 1 always

NOTE: This instruction is used to initialize the B register.

5.16 CLX – Clear X Register

ACTION: Sets the contents of the X register to 0.

OPCODE: 20

SOURCE CODE: [<LABEL>]^CLX^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: 0 → (X)

STATUS FLAG: 1 always

NOTE: This instruction is used to initialize the X register.

5.17 DECMN – Decrement Memory

ACTION: Decrements the contents of the 8-bit RAM location pointed to by the X register. If the 8 bits are all zero, they are set to one and the status flag is set. If not, they are simply decremented and the status flag is cleared.

Because the action taken by the DECMN instruction is to add #0FF to the RAM value, when this instruction is used with 12-bit RAM locations, the lower 8 bits are decremented and the upper 4 bits are incremented whenever there is an overflow from the lower 8 bits.

OPCODE: 27

SOURCE CODE: [<LABEL>] ^DECMN^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (*X) + #0FF → (*X)

STATUS FLAG: 1 if the lower 8 bits of memory went from all zeros to all ones; 0 if not.

5.18 DECXN – Decrement X Register

ACTION: Decrements the contents of the X register. If the X register contains 00, it is set to FF and the status flag is set to 1. Otherwise, the X register is decremented and the status flag is cleared to zero.

OPCODE: 22

SOURCE CODE: [<LABEL>]^DECXN^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (X) – 1 → (X)

STATUS FLAG: 1 if X register went from 00 to FF ; 0 if not.

5.19 EXTSG – Extended-Sign Mode

ACTION: Changes TSP50C1x to extended-sign mode

OPCODE: 3C

SOURCE CODE: [<LABEL>]^EXTSG^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: The TSP50C1x is put into extended-sign mode. All data less than 14 bits in length will be sign extended when being added to, subtracted from, or transferred to the A and B registers.

STATUS FLAG: 1 always

NOTE: Sign extension means that the most significant bit of the data is copied into bits from 13 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 and 13 when the data are 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.

5.20 GET – Get Data From ROM/RAM

ACTION: Transfers 1 to 8 bits of data from internal ROM, external ROM (TSP60C18), or internal RAM to the A register via the parallel-to-serial register.

OPCODE: 30 to 37

SOURCE CODE: [<LABEL>]^GET^<N>^...[<COMMENT>]

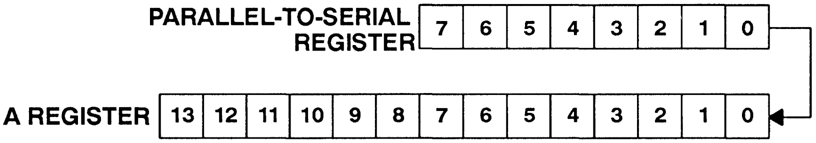
OBJECT CODE:

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

EXECUTION RESULTS: 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.

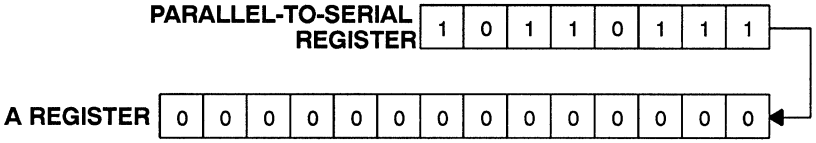
NOTE: The data is shifted out of the LSB of the parallel-to-serial register and into the LSB of the A register, resulting in a bit reversal of any single byte of data transferred into the A register from the order stored in the ROM.



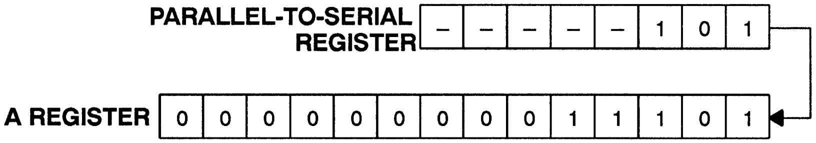
If more bits are requested than are immediately available in the parallel-to-serial register, the next data byte is loaded to the parallel-to-serial register and the remaining bits are transferred to the A register to satisfy the GET instruction.

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

PRIOR TO GET 5 INSTRUCTION



AFTER GET 5 INSTRUCTION



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 Section 6.7.3 for a sample program using RAM GET.

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.

When used to fetch data from external ROM, the GET instruction cannot fetch more than 4 bits of data at a time.

If the LPC bit is set and the first GET instruction after the LUAPS loads the maximum number of bits allowed (i.e., a GET 4 from external ROM or a GET 8 from internal ROM or RAM), the same data will be 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 would be required). Refer to section 6.6 and 6.7 for more information.

5.21 IAC – Increment A Register

ACTION: Increments the contents of the A register by 1

OPCODE: 3A

SOURCE CODE: [<LABEL>]^IAC^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) + 1 → (A)

STATUS FLAG: 1 if the lower 8 bits of the A register go from FF to 00; 0 if not.

NOTE: This instruction increments all 14 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
```

5.22 IBC – Increment B Register

ACTION: Increments the contents of the B register by 1

OPCODE: 25

SOURCE CODE: [<LABEL>]^IBC^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (B) + 1 → (B)

STATUS FLAG: 1 if the lower 8 bits of the B register go from FF to 00; 0 if not.

NOTE: This instruction increments all 14 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
```

5.23 INCMC – Increment Memory

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

OPCODE: 26

SOURCE CODE: [<LABEL>]^INCMC^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (*X) + 1 → (*X)

STATUS FLAG: 1 if the lower 8 bits of memory go from #FF to 00; 0 if not.

5.24 INTGR – Integer Mode

ACTION: Changes TSP50C1x to integer mode

OPCODE: 3B

SOURCE CODE: [<LABEL>]^INTGR^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: The upper bits of data less than 14 bits in length will be zero filled when being transferred to, added to, or subtracted from the A and B registers.

STATUS FLAG: 1 always

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, or numbers of 8 bits or less, are being used.

5.25 IXC – Increment X Register

ACTION: Increments the contents of the X register by 1

OPCODE: 21

SOURCE CODE: [<LABEL>]^IXC^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (X) + 1 → (X)

STATUS FLAG: 1 if the contents of the X register go from FF to 00; 0 if not.

NOTE: The status flag is only set when the X register contains #FF prior to the execution of the IXC instruction. In this case, the status flag is set and the X register value is 0.

EXAMPLE:

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

5.26 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 14 bits.

OPCODE: 6B

SOURCE CODE: [<LABEL>]^LUAA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (*A) → (A)

STATUS FLAG: 1 always.

WARNING: When in extended-sign mode (EXTSG), the value loaded to 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 LUAA instruction; and 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
```

5.27 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 14 bits.

OPCODE: 6D

SOURCE CODE: [<LABEL>]^LUAB^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (*A) → (B)

STATUS FLAG: 1 always.

WARNING: 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
```

5.28 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 SAR is incremented.

OPCODE: 6C

SOURCE CODE: [<LABEL>]^LUAPS^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (SAR); (*SAR) → (PS); (SAR) + 1 → (SAR)

STATUS FLAG: 1 always

NOTE: This instruction is used to initialize the parallel-to-serial register prior to GET instructions. It should be used even if the data are coming from external ROM or 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
```

5.29 ORCM – OR Constant With Memory

ACTION: Logically ORs the contents of RAM pointed to by the X register with the 8-bit operand and stores the results in RAM.

OPCODE: 64

SOURCE CODE: [<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 RESULTS: (*X) || CONST8 → (*X)

STATUS FLAG: 1 always

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
```

5.30 RETI – Return From Interrupt

ACTION: If the interrupt is a level-1 interrupt, 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 an RETI instruction is executed with interrupts enabled and without an interrupt first occurring, the stack control can be corrupted. 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

SOURCE CODE: [<LABEL>] ^RETI^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: 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

5.31 RETN – Return From Subroutine

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

OPCODE: 3D

SOURCE CODE: [<LABEL>]^RETN^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (Top of Stack) → (Program Counter)

STATUS FLAG: 1 always

NOTE: If stack is underflowed, RETN will function as a no-operation command. Control will go to the next consecutive address. Calls can be made indefinitely, but calls can only return three levels.

5.32 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

SOURCE CODE: [<LABEL>]^SALA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A)i → (A)i + 1; 0 → (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 13 of the A register is lost. The results do not depend on the arithmetic mode (EXTSG or INTGR).

5.33 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

SOURCE CODE: [<LABEL>] ^SALA4^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A)i → (A)i + 4; 0 → (A)0 – (A)3

STATUS FLAG: 1 always.

NOTE: Bits 10 to 13 of the A register are lost. The results do not depend on the arithmetic mode (EXTSG or INTGR).

5.34 SARA – Shift A Register Right One Bit

ACTION: Shifts the A register right towards LSB by one bit and fills the MSB with its old value.

OPCODE: 15

SOURCE CODE: [<LABEL>] ^SARA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A)i → (A)i – 1; (A)13 → (A)13

STATUS FLAG: 1 always.

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).

5.35 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

SOURCE CODE: [<LABEL>]^SBAAN^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (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 two's complement subtraction of all 14 bits of the B register from the A register.

5.36 SBR – Short Branch If Status Set

ACTION: When the status flag is set to 1, the lower seven 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

SOURCE CODE: [<LABEL>]^SBR^<ADDR7>^...[<COMMENT>]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION	1	ADDR7						

EXECUTION RESULTS: If SF = 1, ADDR7 + Program Counter PAGE → PC

If SF = 0, Program Counter → Program Counter

STATUS FLAG: 1 always

NOTE: The short branch instruction is a conditional instruction. When a short branch is used following an instruction that always leaves the status flag high, the short branch can be viewed as unconditional.

The program counter is incremented when the instruction is fetched. Because the program counter value is 80 when the instruction is executed, placing an SBR with an operand of 1 at address 7F results in a branch to 81.

An SBR instruction executed at XX7F or XXFF with status cleared (branch not taken) goes to XX00 or XX80, respectively. Version 1.06 or greater of the assembler generates a warning message for all SBR instructions that occur at addresses ending in 7F or FF.

5.37 SETOFF – Set Processor to Off Mode

ACTION: Places the processor in a low-power mode. The clock is stopped and I/O ports are placed in a high-impedance state.

OPCODE: 3F

SOURCE CODE: [<LABEL>] ^SETOFF^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: Processor powered down

STATUS FLAG: State at power up not guaranteed

NOTE: A rising edge on the $\overline{\text{INIT}}$ pin is necessary to restart the processor. The register values are not retained, but the RAM values are retained provided that power continues to be applied to the chip.

5.38 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 14-bit value prior to the subtraction.

OPCODE: 29

SOURCE CODE: [<LABEL>]^SMAAN^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (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
```

5.39 TAB – Transfer A Register to B Register

ACTION: Copies the contents of the A register into the B register

OPCODE: 1A

SOURCE CODE: [<LABEL>]^TAB^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (B)

STATUS FLAG: 1 always

5.40 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

SOURCE CODE: [<LABEL>] ^TAM^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (*X)

STATUS FLAG: 1 always

5.41 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

SOURCE CODE: [<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 RESULTS: (A) → (*CONST8)

STATUS FLAG: 1 always

5.42 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

SOURCE CODE: [<LABEL>]^TAMIX^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (*X); (X) + 1 → (X)

STATUS FLAG: 1 always

5.43 TAMODE – Transfer A Register to Mode Register

ACTION: Copies the lower 8 bits of the A register into the mode register

OPCODE: 1D

SOURCE CODE: [<LABEL>]^TAMODE^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (Mode Register)

STATUS FLAG: 1 always

NOTE: The bit definition for the mode register is in Section 2.19, Mode Register.

5.44 TAPSC – Transfer A Register to Prescale Register

ACTION: Copies the lower 8 bits of the A register into the prescale register

OPCODE: 19

SOURCE CODE: [<LABEL>]^TAPSC^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (Prescale Register)

STATUS FLAG: 1 always

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 Section 2.14, Timer Prescale Register, for more information.

5.45 TASYN – Transfer A Register to Synthesizer Register

ACTION: Copies the 14-bit A register to a speech-processor register. The specific register and resulting control function depend on the operating mode: LPC (load 14-bit pitch register; MSB and LSB of A register must be set to zero), PCM/LPC (load 14-bit LPC excitation register), and PCM (load 12-bit DAC register; see Section 6.9). If none of these modes are active, the value goes into the pitch register just as if LPC mode were active. This is done to allow preloading the pitch before turning on LPC mode.

OPCODE: 1C

SOURCE CODE: [<LABEL>]^TASYN^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (Speech-Processor Register)

STATUS FLAG: 1 always

NOTE: The TASYN copies the 14-bit contents of the A register to the following destinations depending on the contents of the MODE register (see Section 2.19).

MODE BIT		TASYN DESTINATION
LPC	PCM	
0	0	Pitch register
0	1	DAC register
1	0	Pitch register
1	1	Excitation register

When loading the pitch register:

1. The least significant bit and most significant bit of the A register are required to be zero.
2. For voiced frames, the value in the A register:
   a. is required to be 0042_{16} or higher
   b. is strongly recommended to be 0142_{16} or higher
   c. is recommended to be 202_{16} or higher (see Section 2.15)
3. For unvoiced frames, the value in the A register is required to be between 0042_{16} and $03FE_{16}$. Note that even when a frame is unvoiced, a pitch register value must be loaded.

5.46 TATM – Transfer A Register to Timer Register

ACTION: Copies the lower 8 bits of the A register into the timer register

OPCODE: 1E

SOURCE CODE: [<LABEL>] ^TATM^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (Timer Register)

STATUS FLAG: 1 always

5.47 TAX – Transfer A Register to X Register

ACTION: Copies the contents of the lower 8 bits of the A register into the X register

OPCODE: 18

SOURCE CODE: [<LABEL>]^TAX^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (X)

STATUS FLAG: 1 always

5.48 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

SOURCE CODE: [<LABEL>] ^TBM^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (B) → (*X)

STATUS FLAG: 1 always

5.49 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 14-bit two's complement value in the A register.

OPCODE: 6E

SOURCE CODE: [<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 RESULTS:

extended-sign mode: CONST8 → (A)7–0; CONST8 (7) → (A)13–8

integer mode: CONST8 → (A)7–0; 0 → (A)13–8

STATUS FLAG: 1 always

5.50 TCX – Transfer Constant to X Register

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

OPCODE: 62

SOURCE CODE: [<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 RESULTS: CONST8 → (X)

STATUS FLAG: 1 always

5.51 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 14-bit two's complement value in the A register.

OPCODE: 11

SOURCE CODE: [<LABEL>]^TMA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (*X) → (A)

Result depends on whether the TSP50C1x is in integer mode or extended-sign mode.

STATUS FLAG: 1 always

5.52 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 14-bit two's complement value in the A register

OPCODE: 69

SOURCE CODE: [<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 RESULTS: (*CONST8) → (A)

Result depends on whether the TSP50C1x is in integer mode or extended-sign mode

STATUS FLAG: 1 always

5.53 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 14-bit two's complement in the A register.

OPCODE: 14

SOURCE CODE: [<LABEL>] ^TMAIX^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (*X) → (A)

Result depends on whether the TSP50C1x is in integer mode or extended-sign mode

STATUS FLAG: 1 always

5.54 TMXD – Transfer Memory Direct to X Register

ACTION: Copies the lower 8 bits of the memory addressed by the operand into the X register

OPCODE: 6F

SOURCE CODE: [<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 RESULTS: (*CONST8) → (X)

STATUS FLAG: 1 always

5.55 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

SOURCE CODE: [<LABEL>]^TRNDA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (Random Number) → (A)

STATUS FLAG: 1 always

NOTE: The random number register generates a pseudo-random count with 32,767 states. The algorithm is summarized in the following paragraph.

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 OR of bits 13 and 14. The transfer to the A register in response to TRNDA is done prior to this operation.

5.56 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 zero. Otherwise, the status flag is set to 1.

OPCODE: 67

SOURCE CODE: [<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 RESULTS: ((A) && CONST8 == CONST8) → (SF)

STATUS FLAG: Conditionally set to 1 if every bit that is high in the operand has a corresponding high bit in the A register; otherwise set to 0.

NOTE: This instruction logically ANDs 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.

5.57 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

SOURCE CODE: [<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 RESULTS: ((*X) && CONST8 == CONST8) → (SF)

STATUS FLAG: Conditionally set to 1 if every bit that is high in the operand has a corresponding high bit in the memory addressed by the X register; otherwise set to 0.

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.

5.58 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 14-bit two's complement number in the A register.

OPCODE: 17

SOURCE CODE: [<LABEL>] ^TTMA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS:

extended-sign mode: (Timer Register) → (A)7–0; (Timer Register)7 → (A)13–8

integer mode: (Timer Register) → (A)7–0; 0 → (A)13–8

STATUS FLAG: 1 always

5.59 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 14-bit two's complement number in the A register.

OPCODE: 10

SOURCE CODE: [<LABEL>]^TXA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS:

extended-sign mode: (X) → (A)7–0; (X)7 → (A)13–8

integer mode: (X) → (A)7–0; 0 → (A)13–8

STATUS FLAG: 1 always

5.60 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

SOURCE CODE: [<LABEL>]^XBA^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS: (A) ↔ (B)

STATUS FLAG: 1 always

5.61 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 6 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 14-bit two's complement number in the B register.

OPCODE: 23

SOURCE CODE: [<LABEL>] ^XBX^...[<COMMENT>]

OBJECT CODE:

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

EXECUTION RESULTS:

sign-extended mode: (X) → (B)7–0; (X)7 → (B)13–8; (B) → (X)

integer mode: (X) → (B)7–0; 0 → (B)13–8; (B) → (X)

STATUS FLAG: 1 always

5.62 XGEC – X Register Greater Than or Equal to Constant

ACTION: Compares the contents of the X register and the constant specified by the operand and sets the status flag if the contents of the X register are greater than or equal to the operand.

OPCODE: 61

SOURCE CODE: [<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 RESULTS: SF = (X) ≥ CONST8

STATUS FLAG: 1 if the contents of the X register are greater than or equal to the operand; 0 if not.

EXAMPLE:

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

6 Applications

The following sections present programming techniques for specific parts of a TSP50C1x device:

6.1 Synthesizer Control

In this section, a sample program demonstrates how to control the synthesizer in a TSP50C1x device. This program causes the device to synthesize speech from data stored in D6 format. !t is described in the following sections:

6.1.1 Speech Coding and Decoding

The TSP50C1x 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. If 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. If the coded pitch code is 0, a white noise source similar to the turbulence generated by constricted air flow 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.

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 (Table 6–1).

As shown in Figure 6–1, 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 set to zero and no bits are needed to specify them.

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

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 zero pitch value. It is specified by a nonzero energy, a zero pitch, and the first four K parameters.

If 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. If this occurs, the repeat bit is set high, 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 6–2.

Table 6–1. D6 Parameter Size

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

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

Figure 6–1. D6 Frame Decoding

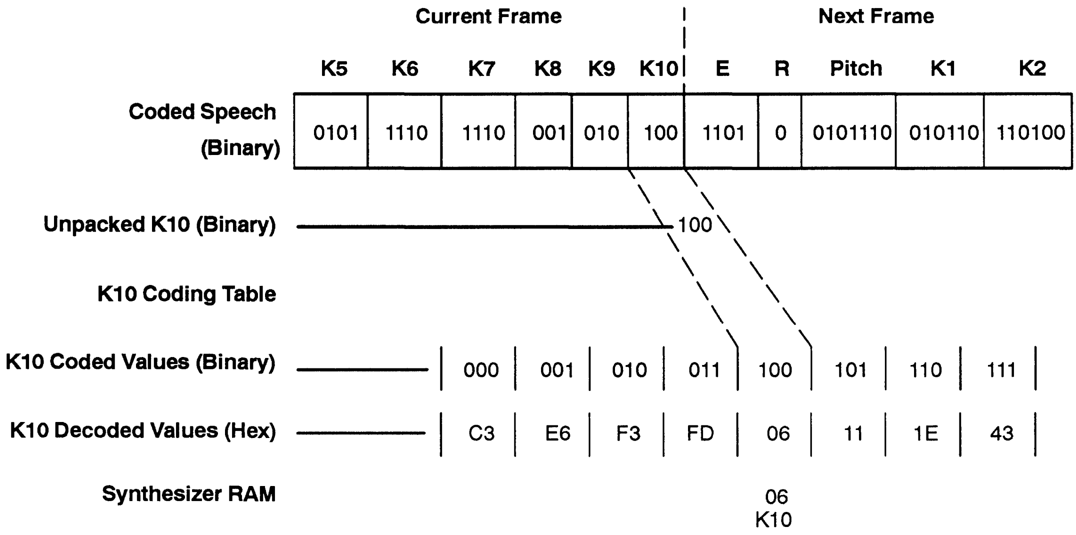


Figure 6–2. Speech Parameter Unpacking and Decoding

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 6.1.4. 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 TSP50C1x 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.

6.1.2 RAM Usage

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

The sample program uses $3C_{16}$ RAM locations. During synthesis, use of the 12-bit RAM locations 01_{16} through $0F_{16}$ is fixed by the architecture of the TSP50C1x. As shown in Table 6–2, 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 may be used at the programmer's discretion.

Use of other RAM locations is detailed in Table 6–3. Energy, pitch, and the first four 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 are stored with 8 bits of precision.

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, two locations are reserved for freezing values read from the timer register. FLAGS contains the status and control flags as detailed in Table 6–4. Because the mode register cannot be read, the sample program maintains a copy of it in RAM in the location MODE_BUF.

Table 6–2. Hardware-Fixed RAM Locations

RAM LOCATION	FUNCTION
01	Energy working value (EN)
02	K12 working value (K12)
03	K11 working value (K11)
04	K10 working value (K10)
05	K9 working value (K9)
06	K8 working value (K8)
07	K7 working value (K7)
08	K6 working value (K6)
09	K5 working value (K5)
0A	K4 working value (K4)
0B	K3 working value (K3)
0C	K2 working value (K2)
0D	K1 working value (K1)
0E	C1 working value (C1)
0F	C2 working value (C2)

Table 6–3. Other RAM Locations Used in Sample Program

10	New energy value (ENV2)	11	Current energy value (ENV1)
12	New pitch value (PHV2)	13	New fractional pitch value
14	Current pitch value (PHV1)	15	Current fractional pitch value
16	New K1 value (K1V2)	17	New fractional K1 value
18	Current K1 value (K1V1)	19	Current fractional K1 value
1A	New K2 value (K2V2)	1B	New fractional K2 value
1C	Current K2 value (K2V1)	1D	Current fractional K2 value
1E	New K3 value (K3V2)	1F	New fractional K3 value
20	Current K3 value (K3V1)	21	Current fractional K3 value
22	New K4 value (K4V2)	23	New fractional K4 value
24	Current K4 value (K4V1)	25	Current fractional K4 value
26	New K5 value (K5V2)	27	Current K5 value (K5V1)
28	New K6 value (K6V2)	29	Current K6 value (K6V1)
2A	New K7 value (K7V2)	2B	Current K7 value (K7V1)
2C	New K8 value (K8V2)	2D	Current K8 value (K8V1)
2E	New K9 value (K9V2)	2F	Current K9 value (K9V1)
30	New K10 value (K10V2)	31	Current K10 value (K10V1)
32	New K11 value (K11V2)	33	Current K11 value (K11V1)
34	New K12 value (K12V2)	35	Current K12 value (K12V1)
36	Stored value of timer used to determine if update needed (TIMER)		
37	Stored value of timer used in INTP routine (SCALE)		
38	LPC status and control flags (FLAGS)		
39	Miscellaneous flags (FLAG1)		
3A	Buffer used to store current mode register contents (MODE_BUF)		
3B	Most significant byte of phrase address (ADR_MSB)		
3C	Least significant byte of phrase address (ADR_LSB)		

Table 6–4. FLAGS Bit Descriptions for Sample Program

BIT	USAGE
0	Set if stop frame detected.
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.

6.1.3 ROM Usage

The sample program uses approximately 1.4K-bytes of ROM, leaving approximately 6.6K-bytes (50C10) or 14.6K-bytes (50C11/12/14) for other functions and speech data. Table 6–5 summarizes ROM usage.

Table 6–5. ROM Usage

ROM LOCATIONS		FUNCTION
TSP50C10	TSP50C11/12/14	
0000 – 000F	0000 – 000F	Device initialization code
0010 – 001F	0010 – 001F	Interrupt vector branches
0020 – 05BE	0020 – 05BE	Synthesis program and tables
05BF – 1FDF	05BF – 3FDF	Available for user program and speech data
1FE0	3FE0	Test codes and excitation codes (not available to user)

6.1.4 Program Overview

The sample synthesis program, parts of which are used in this section for explanation, is reproduced in its entirety in Appendix B. The following is an outline of the program flow.

```
Initialize processor
Initialize speech address register, pitch, C1, and C2
Decode second frame of speech
Start synthesizer
Enable interrupt.
Until stop frame reached:
    Decode each frame
    When interrupt occurs, recalculate working parameter values
Wait two frames, then stop synthesizer
Return to calling routine.
```

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

6.1.4.1 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.

6.1.4.2 Phrase Selection

In general, this 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.

6.1.4.3 Speech Initialization

This section consists of the subroutine SPEAK. It decodes the number contained in the A register for a series of words into the addresses in ROM. It initializes the TSP50C1x for LPC synthesis and speaks the series of words that comprises the desired sentence. For each word in the sentence, it enables synthesis and the level-1 interrupt 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.

6.1.4.4 Level-1-Interrupt Service Routine.

Once the synthesizer is enabled by SPEAK, it writes a new value to the digital-to-analog converter (DAC) once every 30 instruction cycles. The value that is written is calculated from the values contained in RAM locations 01 to 0F and the contents of the pitch period counter (PPC). Loading these locations with the correct values is the responsibility of the level-1-interrupt service routine (INTP). This routine is invoked whenever the interrupt is enabled and the PPC decrements below 200_{16}.

If 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 this is done, INTP simply copies the current values into the working values.

The first case is in transition between voiced and unvoiced frames or between unvoiced and voiced frames. A different number of K factors are used in voiced frames than in unvoiced frames, and the K factors are different. Attempting to interpolate across the voicing transition would result in strange sounds.

The second case is the case of an unvoiced frame following a silence. Plosives (e. g. the 'Phaa' in the letter P) are abrupt unvoiced sounds. Trying to interpolate this case would result in a gradual ramp up of a plosive, which would be incorrect. In the corresponding case of a voiced frame following a silence, it is acceptable to interpolate.

6.1.4.5 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 level-1-interrupt service 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.

If a stop frame is encountered, a flag is set that causes the INTP routine to interpolate down to a silence. The synthesizer and the level-1 interrupt are both inhibited on the next pass through the update routine.

6.1.5 Synthesis Program Walk-Through

What follows is a walk-through of a simple TSP50C1x speech 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 B.

On power up, the processor begins executing code at location 00_{16}. This code initializes the processor by clearing the mode register and the RAM. After that, the processor branches around the interrupt vectors. 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.

```
           ***********************************************************
           *   Start of program
           ***********************************************************
0000                   AORG    #0000
0000   69              TMAD    0
0001   00

           *-----------Initialize mode register----------------------*

0002   2F              CLA
0003   1D              TAMODE

           *-----------Clear all ram to zero-------------------------*

0004   20              CLX                -Start at bottom of RAM
0005   13  RAM_LOOP    TAMIX              -Clear RAM, increment pointer
0006   61              XGEC    MAX_RAM+1 -Finished all RAM?
0007   80
0008   40              BR      GO             yes, skip vector tables
0009   24
000A   40              BR      RAM_LOOP       no, loop back
000B   05
```

In this sample program, ROM addresses $0C_{16}$ to $0F_{16}$ are not used. ROM addresses 10 to 1F contain branches to the interrupt service routines. This section of the ROM address space is dedicated to this purpose by the TSP50C1x 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. For example, when the PPC is decremented below 200_{16}, the program counter is replaced with the value 18_{16}. When the next RETI instruction is encountered, the original value of the program counter is restored. 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, only one of the possible eight interrupt conditions is used. The remaining seven vectors point to a dummy routine that has no effect. Because the routine INTP is out of reach of a short branch, the interrupt vector points to INT_01, which is a long branch to INTP as previously discussed.

```
0263            ***********************************************************
0264            *   Interrupt vectors
0265            ***********************************************************
0266 0010                     AORG     #0010
0267 0010   A2               SBR      INT2_01   -Timer Underflow, PCM=0, LPC=1
0268 0011   A2               SBR      INT2_01   -Timer Underflow, PCM=0, LPC=1
0269 0012   A2               SBR      INT2_00   -Timer Underflow, PCM=0, LPC=0
0270 0013   A2               SBR      INT2_00   -Timer Underflow, PCM=0, LPC=0
0271 0014   A2               SBR      INT2_11   -Timer Underflow, PCM=1, LPC=1
0272 0015   A2               SBR      INT2_11   -Timer Underflow, PCM=1, LPC=1
0273 0016   A2               SBR      INT2_10   -Timer Underflow, PCM=1, LPC=0
0274 0017   A2               SBR      INT2_10   -Timer Underflow, PCM=1, LPC=0
0275 0018   A0               SBR      INT1_01   -PPC < 200 hex interrupt
0276 0019   A0               SBR      INT1_01   -PPC < 200 hex interrupt
0277 001A   A2               SBR      INT1_00   -Pin (B1) goes low interrupt
0278 001B   A2               SBR      INT1_00   -Pin (B1) goes low interrupt
0279 001C   A2               SBR      INT1_11   -10 kHz Clock interrupt
0280 001D   A2               SBR      INT1_11   -10 kHz Clock interrupt
0281 001E   A2               SBR      INT1_10   -20 kHz Clock interrupt
0282 001F   A2               SBR      INT1_10   -20 kHz Clock interrupt
0283            *
0284 0020   40  INT1_01      BR       INTP      -PPC < 200 hex interrupt
     0021   B4
0285            *
0286        0022 INT2_00
0287        0022 INT2_01
0288        0022 INT2_10
0289        0022 INT2_11
0290        0022 INT1_00
0291        0022 INT1_10
0292 0022   2F  INT1_11      CLA
0293 0023   3E               RETI
```

Generally, user programs have several levels of indirection in their use of speech address tables. Often, there are three levels of pointers: (1) sentence pointers that point to the start addresses of entries in the concatenation tables, (2) 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), and (3) 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 below. Note that the use of single bytes to store the word numbers in the concatenation table restricts the vocabulary to 255 words. If 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.

```
           ************************************************************
           *                                                          *
           *    This is the lookup table giving the starting address *
           *    of each concatenation list.                           *
           *                                                          *
           ************************************************************
05BF 05C5' SENTENCE    DATA    PHRASE0
05C1 05CA'             DATA    PHRASE1
05C3 05CF'             DATA    PHRASE2
           ************************************************************
           *                                                          *
           *    This is the concatenation table giving the lists      *
           *    of word numbers that define each phrase.  Each        *
           *    list is terminated by an #FF.                         *
           *                                                          *
           ************************************************************
05C5    01 PHRASE0     BYTE    1,2,3,4,#FF
05CA    04 PHRASE1     BYTE    4,3,2,1,#FF
05CF    05 PHRASE2     BYTE    5,4,3,2,1,#FF
           ************************************************************
           *                                                          *
           *    This is the lookup table for the speech stored at     *
           *    voc.                                                  *
           *                                                          *
           ************************************************************
05D5 0000' SPEECH      DATA    #0000
05D7 05E3'             DATA    #0000+VOC    Word 1      "One"
05D9 0667'             DATA    #0084+VOC    Word 2      "Two"
05DB 06D9'             DATA    #00F6+VOC    Word 3      "Three"
05DD 075D'             DATA    #017A+VOC    Word 4      "Four"
05DF 07C3'             DATA    #01E0+VOC    Word 5      "Five"
05E1 086F'             DATA    #028C+VOC    Word 6      "Six"
```

The following code speaks a series of sentences and then turns off the processor. The number of the desired sentence is loaded into the A register and the routine SPEAK is called to process the sentence. Three sentences are spoken, and then the processor is turned off.

```
           ************************************************************
           *  Speak phrases
           ************************************************************
0024    6E GO          TCA     0         -Speak 1st phrase
0025    00
0026    00             CALL    SPEAK
0027    31
           *
0028    6E             TCA     1         -Speak 2nd phrase
0029    01
002A    00             CALL    SPEAK
002B    31
```

```
            *
002C   6E               TCA     2          -Speak 3rd phrase
002D   02
002E   00               CALL    SPEAK
002F   31
            *
0030   3F               SETOFF             -Quit program
```

What follows is the routine SPEAK, which is called to speak each of the sentences. Before this routine is entered, the desired sentence number is loaded in the A register. 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 bytes of the address of the concatenation table entry.

```
            **********************************************************
            *   Speak Utterance - Phrase number in A register
            **********************************************************
0031   3B   SPEAK       INTGR
0032   2E               SALA               -Double index to get offset
0033   75               ACAAC   SENTENCE   -Add base of table
0034   BF
0035   6D               LUAB               -get address MSB
0036   3A               IAC
0037   6B               LUAA               -Get address LSB
0038   12               XBA
0039   1B               SALA4              -Combine MSB and LSB
003A   1B               SALA4
003B   2C               ABAAC
```

The 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_MSB and ADR_LSB.

```
003C   1A               TAB                -Save address
003D   6A               TAMD    ADR_LSB    -Save LSB of address
003E   3C

003F   68               AXCA    1          -Shift address right
0040   01
0041   15               SARA                   by 8 bits

0042   6A               TAMD    ADR_MSB    -Save MSB of address
0043   3B
0044   12               XBA
0045   40               BR      SPEAK2
0046   59

```

The following code gets the address of the current concatenation table entry, increments to the next entry, and then stores that address. This code is reached when the processor has finished speaking one word in a sentence and is ready to speak the next word.

```
0331 0047   69  SPEAK1      TMAD   ADR_LSB    -Fetch and combine
     0048   3C
0332 0049   1A              TAB                   address
0333 004A   69              TMAD   ADR_MSB
     004B   3B
0334 004C   1B              SALA4
0335 004D   1B              SALA4
0336 004E   2C              ABAAC
0337
0338 004F   3A              IAC               -Increment address
0339
0340 0050   1A              TAB               -Save new address
0341 0051   6A              TAMD   ADR_LSB    -Save LSB of address
     0052   3C
0342
0343 0053   68              AXCA   1          -Shift address right
     0054   01
0344 0055   15              SARA                  by 8 bits
0345
0346 0056   6A              TAMD   ADR_MSB    -Save MSB of address
     0057   3B
0347 0058   12              XBA
```

The next section of code uses the LUAA instruction to fetch the next byte of the concatenation table entry, tests to see if it marks the end of the concatenation table entry, and, if so, exits the routine.

```
0349 0059   6B  SPEAK2      LUAA              -Get word number
0350 005A   60              ANEC   StopWord   -End of phrase?
     005B   FF
0351 005C   40              BR     SPEAK3         no, continue
     005D   5F
0352 005E   3D              RETN                  yes, exit loop
```

Now a word number is in the A register. It needs to be used as an index into the word address table to get the starting address of the word in memory. Because each address in the table is two bytes long, the word number is doubled to get the correct offset into the table before adding the address of the start of the table to the offset.

```
0354 005F   2E  SPEAK3      SALA              -Double index to get offset
0355 0060   75              ACAAC  SPEECH     -Add base of table
```

The A register now contains the address in ROM where the starting address of the desired word is stored. The following fragment retrieves this address and loads it into the SAR (speech address register). The LUAB instruction gets the most significant byte of the address into the B register. The LUAA gets the least significant byte of the address into the A register. The most significant byte is then left shifted by 8 bits and the least significant byte is added to it. The complete address is now in the A register. The LUAPS transfers this address into the SAR. Speech begins at this address.

```
0356 0062   6D              LUAB                –Get address MSB
0357 0063   3A              IAC
0358 0064   6B              LUAA                –Get address LSB
0359 0065   12              XBA
0360 0066   1B              SALA4               –Combine LSB and MSB
0361 0067   1B              SALA4
0362 0068   2C              ABAAC
0363
0364 0069   6C              LUAPS               –Load Speech Address Register
```

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 and sets all the speech flags to the default condition (zero).

```
0366 006A   2F              CLA                 –Kill K11 and K12 parameters
0367 006B   6A              TAMD    K11
     006C   03
0368 006D   6A              TAMD    K12
     006E   02
0369
0370 006F   6A              TAMD    FLAGS       –Init flags for speech
     0070   38
```

The values in C1 and C2 control the behavior of the output filter. For most applications, the values should be set as shown below.

```
0372 0071   2F              CLA                 –Load C2 parameter
0373 0072   7B              ACAAC   C2_Value       (a device constant)
     0073   67
0374 0074   6A              TAMD    C2
     0075   0F
0375
0376 0076   2F              CLA                 –Load C1 parameter
0377 0077   7F              ACAAC   C1_Value       (a device constant)
     0078   61

0378 0079   6A              TAMD    C1
     007A   0E
```

A default pitch is now assigned 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 would be loaded with a zero and the synthesis of the first frame would be incorrect.

```
0383 007B   70              ACAAC   #0C
     007C   0C
0384 007D   6A              TAMD    PHV1
     007E   14
0385 007F   6A              TAMD    PHV2
     0080   12
```

The first two frames are now preloaded. Each call to UPDATE loads one frame of speech. With two frames loaded into memory, the synthesis routine can properly do its interpolation function.

```
0389 0081   01              CALL    UPDATE    -Load first frame
     0082   B5
0390 0083   01              CALL    UPDATE    -Load 2nd frame
     0084   B5
```

The first interpolation is done by calling the routine INTP. This is the same routine that is invoked by the interrupt after it is enabled later. Before INTP is called, however, the timer and prescaler values need to be initialized so that the interpolation function of INTP yields the correct value.

```
0397 0085   6E              TCA     PSVALUE   -Initialize prescale
     0086   2E
0398 0087   19              TAPSC
0399 0088   6E              TCA     #7F       -Pretend there was a previous
     0089   7F
0400 008A   6A              TAMD    TIMER        update
     008B   36
0401 008C   6E              TCA     #FF       -Set timer to max value to
     008D   FF
0402 008E   1E              TATM                 disable interpolation
0403 008F   00              CALL    INTP      -Do first interpolation
     0090   B4
```

The last step before the start of speech is to turn on the synthesizer and then enable the interrupt. This is done by setting the appropriate bits in the mode register with the TAMODE instruction.

There are many cases in which a program may need to know what value is currently in the mode register. This is a problem because there is no way to read directly from the mode register. The best way around this problem is to maintain a copy of the mode register that can be read. This program, therefore, designates a RAM location as MODE_BUF. Any changes to the mode register are made in the following three-step procedure: (1) the change is made in MODE_BUF, (2) the value in MODE_BUF is transferred to the A register, and (3) the mode register is changed with a TAMODE instruction.

In the following code, first the synthesizer is turned on and then a RETI is executed to ensure that the interrupt-pending latch is not set before interrupts are enabled. Once the interrupt is enabled, the routine INTP is reached whenever the pitch period counter decrements below 200_{16}.

```
0412 0091   62              TCX     MODE_BUF  -Turn on LPC synthesizer
     0092   3A
0413 0093   64              ORCM    LPC
     0094   02
0414 0095   11              TMA
0415 0096   1D              TAMODE
0416
0417 0097   3E              RETI              -Reset interrupt pending latch
0418
0419 0098   64              ORCM    INT1      -Enable interrupt
     0099   01
0420 009A   11              TMA
0421 009B   1D              TAMODE
```

When the synthesis routine detects the stop frame, it branches back to SPEAK1 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. First, it tests whether the timer has already decremented below zero; if so, an update is performed. Second, it tests whether the timer is equal to zero; if so, UPDATE is immediately called. By the time UPDATE completes processing, the timer register will have underflowed.

```
    009C SPEAK_LP
009C 62             TCX    FLAGS
009D 38
009E 66             TSTCM  Update_Flg –Is Update already done?
009F 04
00A0 40             BR     SPEAK_LP   yes, loop
00A1 9C

00A2 62             TCX    TIMER   –Get old timer
00A3 36
00A4 11             TMA               register value
00A5 1A             TAB               into B register

00A6 17             TTMA           –Get new timer register
00A7 15             SARA              value and scale it.

00A8 16             TAM            –Store new value
00A9 12             XBA            –Exchange new and old values
00AA 2D             SBAAN          –Subtract new from old
00AB 41             BR     UPDATE  –If underflowed, do an update
00AC B5

00AD 11             TMA            –Get new timer value again.
00AE 60             ANEC   0       –Is it about to underflow?
00AF 00
00B0 40             BR     SPEAK_LP   no, loop again
00B1 9C
00B2 41             BR     UPDATE     yes, do update now
00B3 B5
```

INTP is the interrupt service routine for the level-1 interrupt. It is reached whenever the pitch period counter decrements below 200_{16}. Its purpose is to do any necessary interpolation of the reflection coefficients (K factors) and to load the result into the working registers.

On entry to INTP, the current value of the timer register is stored. This value will be used later when interpolation is performed.

```
00B4 3B   INTP      INTGR          –Ensure we are in integer mode
00B5 17             TTMA           –Get timer register contents
00B6 15             SARA              shift to make positive
00B7 6A             TAMD   SCALE      and store it
00B8 37
```

If interpolation has been turned off by the UPDATE routine, INTP is exited.

```
0469 00B9    62                TCX      FLAG1     -Point to flag
     00BA    39
0470 00BB    66                TSTCM    Int_Off   -If routine disabled...
     00BC    01
0471 00BD    41                BR       IRETI        ...branch to exit point
     00BE    B3
```

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. The following code tests FLAGS to see if interpolation should be performed between frames.

```
0479 00BF    62   TINTP        TCX      FLAGS     -Point to status flags
     00C0    38
0480 00C1    66                TSTCM    Int_Inh   -Is interpolation inhibited?
     00C2    20
0481 00C3    40                BR       NOINT        yes, inhibit interpolation
     00C4    C7
0482 00C5    40                BR       INTPCH       no, interpolate
     00C6    E4
```

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.

```
0490 00C7    6E   NOINT        TCA      #7F       -Set Scale factor to
     00C8    7F
0491 00C9    6A                TAMD     SCALE        highest value
     00CA    37
```

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, the energy is cleared.

```
0501 00CB    62                TCX      FLAGS
     00CC    38
0502 00CD    66                TSTCM    Unv_Flg2  -Is current frame unvoiced?
     00CE    80
0503 00CF    40                BR       Uv           yes, go to unvoiced branch
     00D0    D9
0504
0505 00D1    62                TCX      Mode_Buf  -Current frame is voiced
     00D2    3A
0506 00D3    66                TSTCM    UNV       -Has mode changed to unvoiced?
     00D4    80
0507 00D5    40                BR       ClrEN        yes, clear the energy
```

```
     00D6   DF
0508 00D7   40                  BR       INTPCH         no, no action required
     00D8   E4
0509
0510 00D9   62  Uv              TCX      Mode_Buf   –New frame is unvoiced
     00DA   3A
0511 00DB   66                  TSTCM    UNV        –Has voicing mode changed?
     00DC   80
0512 00DD   40                  BR       INTPCH         no, no action required
     00DE   E4
0513
0514 00DF   2F  ClrEN           CLA                 –Zero Energy during update
0515 00E0   6A                  TAMD     EN
     00E1   01
0516 00E2   40                  BR       INTPCH
     00E3   E4
```

We are now ready to do the interpolation. The interpolation is done with the standard linear equation:

$$y = mx + b$$

rewritten to:

$$P = (\text{Pcurrent} - \text{Pnew}) \times \text{TIMER} + \text{Pnew}$$

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 80_{16}. The value in TIMER ranges from $7F_{16}$ to zero. If interpolation is inhibited, TIMER will contain $7F_{16}$ and the interpolation will result 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 will always be positive. The INTGR instruction ensures the integer mode so that when the program gets the decoded pitch from the decoding tables, it is not sign extended. (see Section 6.2 for additional information on arithmetic modes).

```
0522 00E4   62  INTPCH          TCX      PHV2       –Combine new pitch and new
     00E5   12
0523 00E6   14                  TMAIX                  fractional pitch and
0524 00E7   1B                  SALA4                  leave in the B register
0525 00E8   28                  AMAAC
0526 00E9   21                  IXC
0527 00EA   1A                  TAB
0528 00EB   14                  TMAIX               –Combine current pitch and
0529 00EC   1B                  SALA4                  current fractional pitch
```

```
00ED   28           AMAAC                  and leave in A register

00EE   2D           SBAAN               -(Pcurrent - Pnew)
00EF   62           TCX      SCALE
00F0   37
00F1   39           AXMA                -(Pcurrent-Pnew)*Timer
00F2   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 period counter using the TASYN instruction. 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 using the SALA instruction.

```
00F3   2E           SALA                -Adjust for 2 byte excitation
00F4   1C           TASYN               -Write to pitch register
```

Because the decoded K parameters can be both positive and negative, the program goes to extended-sign mode so that the values will not change sign when they are read into the A or B registers.

```
00F5   3C           EXTSG               -Allow negative K parameters
```

K1 through K4 are interpolated in the same manner as energy. The interpolation of K1 is shown below. K2 through K4 are not shown.

```
00F6   62           TCX      K1V2       -Combine New K1 and New
00F7   16
00F8   14           TMAIX                  fractional K1 and
00F9   1B           SALA4                  leave in the B register
00FA   28           AMAAC
00FB   21           IXC
00FC   1A           TAB

00FD   14           TMAIX               -Combine current K1 and
00FE   1B           SALA4                  current fractional K1 and
00FF   28           AMAAC                  leave in the A register

0100   2D           SBAAN               -(K1current - K1new)
0101   62           TCX      SCALE
0102   37
0103   39           AXMA                -(K1current - K1new) * Timer
0104   2C           ABAAC               -K1new+(K1current-K1new)*Timer
0105   6A           TAMD     K1         -Load interpolated K1 value
0106   0D
```

Since K5 through K10 are stored using an 8-bit precision instead of the 12-bit precision used for K1 through K4, the interpolation is simpler. The following fragment shows the interpolation used for K5. The code for K6 through K10 is similar.

```
0623 013A   62          TCX     K5V2     -Put New K5 (adjusted to
     013B   26
0624 013C   14          TMAIX               12 bits) in B register
0625 013D   1B          SALA4
0626 013E   1A          TAB
0627 013F   14          TMAIX            -Put Current K5 (adjusted to
0628 0140   1B          SALA4               12 bits) in A register
0629
0630 0141   2D          SBAAN            -(K5current - K5new)
0631 0142   62          TCX     SCALE
     0143   37
0632 0144   39          AXMA             -(K5current - K5new) * Timer
0633 0145   2C          ABAAC            -K5new+(K5current-K5new)*Timer
0634 0146   6A          TAMD    K5       -Load interpolated K5 value
     0147   09
```

The decoded energy, like the pitch, will always be positive. The INTGR instruction places the processor in integer mode so that the decoded energy will not be sign extended. The following code interpolates the energy.

```
0761 018E   3B          INTGR            -Back to integer mode for energy
0762 018F   62          TCX     ENV2     -Combine new energy and
     0190   10
0763 0191   14          TMAIX               fractional energy and
0764 0192   1B          SALA4               leave in the B register
0765 0193   1A          TAB
0766 0194   14          TMAIX            -Combine current energy and
0767 0195   1B          SALA4               current fractional energy
0768 0196   2D          SBAAN            -(Ecurrent - Enew)
0769 0197   62          TCX     SCALE
     0198   37
0770 0199   39          AXMA             -(Ecurrent - Enew) * Timer
0771 019A   2C          ABAAC            -Enew+(Ecurrent-Enew)*Timer
0772 019B   6A          XBA              -Save energy
```

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). After changing the mode register, the program stores the interpolated energy and then exits from the INTP routine with either RETN or RETI depending on whether this routine was reached using a subroutine call or in response to an interrupt.

```
0781 019C   62  STMODE  TCX     FLAGS
     019E   38
0782 019E   65          ANDCM   ~Update_Flg -Signal that interp done
     019F   FB
0783 01A0   66          TSTCM   Unv_Flg2 -Is current frame unvoiced?
     01A1   80
0784 01A2   41          BR      SETUV    -yes, set mode to unvoiced
     01A3   AA
0785 01A4   62          TCX     MODE_BUF    no, ...
```

```
     01A5   3A
0786 01A6   65                ANDCM    ~UNV          ...set mode to voiced
     01A7   7F
0787 01A8   41                BR       WRITEMODE
     01A9   AE
0788
0789 01AA   62  SETUV         TCX      MODE_BUF  -Current frame is unvoiced, so
     01AB   3A
0790 01AC   64                ORCM     UNV       -set mode to unvoiced.
     01AD   80
0790
0792 01B1   11  WRITEMODE     TMA                -Write mode information
0793 01AF   1D                TAMODE              to mode register
0794
0795 01B0   12                XBA                -Write energy
0796 01B1   62                TAMD      EN        to filter
     01B2   01
0797
0798 01B3   3E  IRETI          RETI               -Return from interrupt
0799 01B4   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 stop frame encountered on last pass, stop speaking

Copy new unvoiced flag to current unvoiced flag

Copy new silence flag to current silence flag

Set new silence flag, new unvoiced flag, and interpolation flag to zero

Copy new speech parameters to current speech parameters

Get coded energy

If silence frame, set new silence flag

If stop frame, set stop flag

Look up decoded energy from table and put in new energy

Inhibit interpolation if last frame was silent and this one is not.

Get repeat bit, set repeat flag if it is a one.

Get coded pitch

If unvoiced frame, set new unvoiced flag.

Look up decoded pitch from table and store as new pitch.

If new voicing is different from current voicing, inhibit interpolation.

Get coded K parameters

Look up decoded K parameters from table and store as new values.

First, the level-1 interrupt is disabled so that an interpolation is not attempted during the period that the frame data is not valid. The level-1 interrupt is reenabled before exiting UPDATE.

```
0805 01B5   62  UPDATE      TCX     MODE_BUF
     01B6   3A
0806 01B7   65              ANDCM   ~INT1
     01B8   FE
0807 01B9   11              TMA
0808 01BA   1D              TAMODE
```

To prevent double updates, if the stored value of the timer register is zero, then it needs to be changed to 7F. If this is not done, then the polling routine will discover an underflow and call UPDATE a second time.

```
0815 01BB   62              TCX     TIMER     —Get stored value
     01BC   36
0816 01BD   11              TMA                  of Timer into A
0817
0818 01BE   60              ANEC    0         —Is it zero?
     01BF   00
0819 01C0   41              BR      UPDT00       no, do nothing
     01C1   C5
0820 01C2   6E              TCA     #7F          yes, replace value
     01C3   7F
0821 01C4   16              TAM
```

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 disable synthesis and branch back to prepare for the next utterance in the phrase.

```
0828 01C5   62  UPDT00      TCX     FLAGS
     01C6   38
0829 01C7   66              TSTCM   STOPFLAG  —Was stop frame encountered
     01C8   01
0830 01C9   42              BR      STOP         yes, stop speaking
     01CA   EF
```

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.

```
0835 01CB   66              TSTCM   Unv_Flg1  —Was previous frame unvoiced?
     01CC   10
0836 01CD   41              BR      SUNVL        yes, current frame=unvoiced
     01CE   D3
0837 01CF   65              ANDCM   ~Unv_Flg2    no, current frame=voiced
     01D0   7F
0838 01D1   41              BR      TSIL         and continue
     01D2   D5
0839
```

```
0840 01D3    64   SUNVL         ORCM    Unv_Flg2  -Set current frame unvoiced.
     01D4    80
0845 01D5    66   TSIL          TSTCM   Sil_Flg1  -Was previous frame silent?
     01D6    08
0846 01D7    41                 BR      SSIL          yes, current frame silent
     01D8    DD
0847 01D9    65                 ANDCM   ~Sil_Flg2     no, current frame not sil.
     01DA    BF
0848 01DB    41                 BR      ZROFLG        and continue
     01DC    DF
0849
0850 01DD    64   SSIL          ORCM    Sil_Flg2  -Set current frame silent
     01DE    40
```

Now the program resets the repeat flag, silence flag, unvoiced flag, and interpolation-inhibit flag to zero. They will be set later if the next frame requires them to be set.

```
0856 01DF    62   ZROFLG        TCX     FLAGS
     01E0    38
0857 01E1    65                 ANDCM   #C5
     01E2    C5
```

Now the new speech parameters are saved as current speech parameters prior to loading the next frame.

```
0862 01E3    62                 TCX     ENV2      -Transfer new frame energy
     01E4    10
0863 01E5    14                 TMAIX                 from new frame location
0864 01E6    13                 TAMIX                 to current frame location
0865              *------PITCH------
0866 01E7    14                 TMAIX             -Transfer new frame pitch
0867 01E8    6A                 TAMD    PHV1          to current frame location
     01E9    14
0868
0869 01EA    14                 TMAIX             -Transfer new fractional pitch
0870 01EB    21                 IXC                   to current frame location
0871 01EC    13                 TAMIX
0872              *------K1------
0873 01ED    14                 TMAIX             -Transfer new frame K1 param.
0874 01EE    6A                 TAMD    K1V1          to current frame location
     01EF    18
0875 01F0    14                 TMAIX             -Transfer new fractional K1
0876 01F1    21                 IXC                   to current frame location
0877 01F2    13                 TAMIX
.
.
.
0911              *------K10------
0912 020F    14                 TMAIX             -Transfer new frame K10 param.
0913 0210    13                 TAMIX                 to current frame location
```

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 0, 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.

```
0932 0211  2F              CLA
0933 0212  62              TCX     FLAGS
     0213  38
0934 0214  33              GET     EBITS     -Get coded energy
0935 0215  60              ANEC    ESILENCE  -Is it a silent frame?
     0216  00
0936 0217  42              BR      UPDT0        No, continue
     0218  1D
0937 0219  64              ORCM    Sil_Flg1+Int_Inh   Yes, set silence flag
     021A  28
0938 021B  42              BR      ZeroKs       and zero K params
     021C  CD
0939           *
0940 021D  60  UPDT0       ANEC    ESTOP     -Is it a stop frame?
     021E  0F
0941 021F  42              BR      UPDT1        no, continue
     0220  25
0942 0221  64              ORCM    STOPFLAG+Sil_Flg1+Int_Inh yes, set flags
     0222  29
0943 0223  42              BR      ZeroKs       and zero Ks
     0224  CD
```

Now the energy is decoded. The LUAA instruction is used to get the decoded energy.

```
0945 0225  73  UPDT1       ACAAC   TBLEN     -Add table offset to energy
     0226  27
0946 0227  6B              LUAA              -Get decoded energy
0947 0228  6A              TAMD    ENV2      -Store the Energy in RAM
     0229  10
```

If this is a silent frame (tested for earlier), no more parameters need to be read. In this case, the program branches to the routine exit point.

```
0953 022A  62              TCX     FLAGS
     022B  38
0954 022C  66              TSTCM   Sil_Flg1  -Is this a silent frame?
     022D  08
0955 022E  43              BR      RTN          yes, exit
     022F  0C
```

The next code fragment is reached if the new frame is not silent. It reads the repeat bit. This bit is set to indicate that all of the K parameters between the new frame and the previous are identical. If this is so, the K factors 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 factors are read in.

```
0230   30  UPDT2     GET    RBITS     –Get the Repeat bit
0231   67            TSTCA  #01       –Is this a repeat frame?
0232   01
0233   42            BR     SFLG1        yes, set repeat flag
0234   37
0235   42            BR     UPDT3
0236   39

0237   64  SFLG1     ORCM   R_FLAG    –Set repeat flag
0238   02
```

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.

```
0239   2F  UPDT3     CLA
023A   33            GET    4         –Get coded pitch
023B   32            GET    3         –Get coded pitch
023C   60            ANEC   PUnVoiced –Is the frame unvoiced?
023D   00
023E   C1            SBR    UPDT3A       no, continue
023F   64            ORCM   Unv_Flg1     yes, set unvoiced flag
0240   10
```

Now 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.

```
0241   2E  UPDT3A    SALA             –Double coded pitch and
0242   73            ACAAC  TBLPH        add table offset to point
0243   37

0244   6D            LUAB             –Get decoded pitch
0245   3A            IAC
0246   6B            LUAA             –Get decoded fractional pitch

0247   62            TCX    PHV2      –Store the pitch and
0248   12
0249   2A            TBM                 fractional pitch in RAM
024A   21            IXC
024B   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.

```
0991 024C   62             TCX     FLAGS
     024D   38
0992 024E   66             TSTCM   Unv_Flg1  -Is the new frame unvoiced?
     024F   10
0993 0250   D3             SBR     UPDT3B       yes, continue
0994 0251   42             BR      VOICE        no, go to voiced code
     0252   5D
```

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.

```
1002 0253   66  UPDT3B     TSTCM   Sil_Flg2  -Was the last frame silent?
     0254   40
1003 0255   42             BR      UPDT5        yes, inhibit interpolation
     0256   63
```

Then the program tests to see if the previous frame was voiced.

```
1005 0257   66             TSTCM   Unv_Flg2  -Was the last frame unvoiced
     0258   80
1006 0259   42             BR      UPDT4        yes, don't change anything
     025A   65
1007 025B   42             BR      UPDT5        no, inhibit interpolation
     025C   63
```

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.

```
1014 025D   66  VOICE      TSTCM   Unv_Flg2  -Was the last frame voiced?
     025E   80
1015 025F   42             BR      UPDT5        no, disable interpolation
     0260   63
1016 0261   42             BR      UPDT4        yes, continue
     0262   65
```

The following code inhibits interpolation.

```
1018 0263   64  UPDT5      ORCM    Int_Inh   -Inhibit interpolation
     0264   20
```

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 factors. This section of code branches to the general routine exit if this is a repeat frame.

```
1025 0265   66  UPDT4      TSTCM   R_FLAG    -Is repeat flag set?
```

```
     0266   02
1026 0267   43             BR      RTN          yes, exit routine
     0268   0C
```

The first four K factors (K1 through K4) are now loaded. Each of these decoded K factors is a 12-bit value that is stored in two bytes. The most significant 8 bits are contained in the first byte, and the least significant 4 bits are contained in the second byte.

The GET instruction reads the coded K factor into the A register. It is left shifted (multiplied by two) to convert it into an offset in the table that contains the two-byte uncoded K factors. 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. K1 is shown below. K2 through K4 are similar to K1.

```
1046             *———K1———
1047 0269   2F             CLA
1048 026A   33             GET     4          –Get coded K1
1049 026B   31             GET     2          –Get coded K1
1050 026C   2E             SALA               –Convert it to a
1051 026D   74             ACAAC   TBLK1        pointer to table element
     026E   37
1052 026F   6D             LUAB               –Fetch MSB of uncoded K1
1053 0270   3A             IAC
1054 0271   6B             LUAA               –Fetch fractional K1
1055 0272   62             TCX     K1V2
     0273   16
1056 0274   2A             TBM                –Store uncoded K1
1057 0275   21             IXC
1058 0276   16             TAM                –Store fractional K1
```

Now the program needs to test to see if the new frame is unvoiced. Unvoiced frames use only four K factors, and the remaining K factors are set to zero. At this point, the first four K factors 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 factors.

```
1098 029B   62             TCX     FLAGS
     029C   38
1099 029D   66             TSTCM   Unv_Flg1   –Is this an unvoiced frame?
     029E   10
1100 029F   42             BR      UNVC         Yes, zero rest of factors
     02A0   E0
```

The remaining K factors differ from the first four K factors in that they have only an 8-bit precision for their decoded values instead of the 12-bit precision used for the first four K factors. This precision reduction simplifies the code. K5 is shown below. K6 through K10 are similar to K5.

```
1109             *———K5———
1110 02A1   2F             CLA
1111 02A2   33             GET     K5BITS     –Get Index into K5 table
1112 02A3   75             ACAAC   TBLK5        and add offset of table
     02A4   77
```

```
1113
1114 02A5   6B                LUAA               -Get uncoded K5
1115 02A6   6A                TAMD     K5V2         and store it in RAM
     02A7   26
```

After all the K factors for a voiced frame have been loaded, the UPDATE routine can be exited by branching to the general routine exit.

```
1163 02CB   43                BR       RTN
     02CC   0C
```

This section of code clears K parameters that are not used. Silent and stop frames result in a branch to ZeroKs. Unvoiced frames result in a branch to UNVC.

```
1172 02CD   2F   ZeroKs       CLA
1173 02CE   6A                TAMD     ENV2       -Kill Energy
     02CF   10
1174 02D0   6A                TAMD     K1V2       -Kill K1
     02D1   16
1175 02D2   6A                TAMD     K1V2+1
     02D3   17
1176 02D4   6A                TAMD     K2V2       -Kill K2
     02D5   1A
1177 02D6   6A                TAMD     K2V2+1
     02D7   1B
1178 02D8   6A                TAMD     K3V2       -Kill K3
     02D9   1E
1179 02DA   6A                TAMD     K3V2+1
     02DB   1F
1180 02DC   6A                TAMD     K4V2       -Kill K4
     02DD   22
1181 02DE   6A                TAMD     K4V2+1
     02DF   23
1182 02E0   2F   UNVC         CLA
1183 02E1   6A                TAMD     K5V2       -Kill K5
     02E2   26
1184 02E3   6A                TAMD     K6V2       -Kill K6
     02E4   28
1185 02E5   6A                TAMD     K7V2       -Kill K7
     02E6   2A
1186 02E7   6A                TAMD     K8V2       -Kill K8
     02E8   2C
1187 02E9   6A                TAMD     K9V2       -Kill K9
     02EA   2E
1188 02EB   6A                TAMD     K10V2      -Kill K10
     02EC   30
1189               *          TAMD     K11V2      -Kill K11
1190               *          TAMD     K12V2      -Kill K12
1191 02ED   43                BR       RTN
```

```
     02EE   0C
```

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.

```
1201 02EF   62  STOP        TCX     MODE_BUF
     02F0   3A
1202 02F1   65              ANDCM   ~LPC      —Turn off synthesis
     02F2   FD
1203 02F3   65              ANDCM   ~INT1     —Disable interrupt
     02F4   FE
1204 02F5   65              ANDCM   ~UNV      —Back to voiced for next word
     02F6   7F
1205 02F7   64              ORCM    PCM       —Enable PCM mode
     02F8   04
1206 02F9   11              TMA
1207 02FA   1D              TAMODE            —Set mode per above setting
1208 02FB   2F              CLA
1209 02FC   1C              TASYN             —Write a zero to the DAC
1210 02FD   6E              TCA     #FA
     02FE   FA
1211 02FF   3A  BACK        IAC               —Wait for minimum of 30
1212 0300   43              BR      out          instruction cycles
     0301   04
1213 0302   42              BR      back
     0303   FF
1214 0304   62  OUT         TCX     MODE_BUF  —Disable PCM
     0305   3A
1215 0306   65              ANDCM   ~PCM
     0307   FB
1216 0308   11              TMA
1217 0309   1D              TAMODE            —Set mode per above setting
1218 030A   40              BR      SPEAK1    —Go back for next word
     030B   47
```

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.

```
1220 030C   62  RTN         TCX     FLAGS     —Set a flag indicating that
     030D   38
1221 030E   64              ORCM    Update_Flg   the parameters are updated
     030F   04
1222
1223 0310   62              TCX     MODE_BUF  —Get mode
     0311   3A
1224 0312   66              TSTCM   LPC       —Are we speaking yet?
```

```
0313   02
0314   43               BR       RTN1          yes, reenable interrupt
0315   17
0316   3D               RETN                   no, return for more data

0317   62   RTN1        TCX      FLAG1      —Inhibit any pending
0318   39
0319   64               ORCM     Int_Off       interpolation interrupt
031A   01

031B   62               TCX      MODE_BUF   —Reenable the interrupt
031C   3A
031D   64               ORCM     INT1
031E   01
031F   11               TMA
0320   1D               TAMODE

0321   62               TCX      FLAG1      —Reenable execution
0322   39
0323   65               ANDCM    ~Int_Off     of the interpolation routine
0324   FE
0325   40               BR       SPEAK_LP   —Go back to loop
0326   9C
```

The speech data decoding tables can be seen in the complete sample program shown in Appendix B.

6.2 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 00_{16} representing the most negative value and FF_{16} 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 TSP50C1x 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 TSP50C1x. 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 8 bits, and the A register has a size of 14 bits. A value of FF_{16} in the X register represents 255 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 6–6 illustrates the difference.

Table 6–6. TXA Operation

MODE	X REGISTER		A REGISTER		VALUE
Integer Mode	FF_{16}	→	$00FF_{16}$	=	255_{10}
Extended-Sign Mode	FF_{16}	→	$3FFF_{16}$	=	-1_{10}
Integer Mode	05_{16}	→	0005_{16}	=	5_{10}
Extended-Sign Mode	05_{16}	→	0005_{16}	=	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 zero.

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 zero. When the value is transferred, the high-order bits are set to zero 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 14-bit A or B registers from a smaller register or memory location. Figure 6–3 illustrates the operation of the ACAAC instruction in extended-sign mode. The 12-bit constant must be sign-extended to 14 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 FFF_{16} to $3FFF_{16}$.

CARRY		$11\ 1111\ 1111\ 11_2$
A REGISTER	3202_{16}	$11\ 0010\ 0000\ 0010_2$
CONSTANT	FFF_{16}	$11\ 1111\ 1111\ 1111_2$
RESULT	3201_{16}	$11\ 0010\ 0000\ 0001_2$

Figure 6–3. ACAAC in Extended-Sign Mode

Figure 6−4 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 FFF_{16}.

CARRY		$11\ 1111\ 1111\ 11_2$
A REGISTER	3202_{16}	$11\ 0010\ 0000\ 0010_2$
CONSTANT	FFF_{16}	$00\ 1111\ 1111\ 1111_2$
RESULT	0201_{16}	$00\ 0010\ 0000\ 0001_2$

Figure 6−4. ACAAC in Integer Mode

6.3 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 four-bit number. Similarly, multiplying the 14-bit A register with the contents of an 8-bit memory location would result in a 22-bit value. This creates a problem because this large of a value 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:

$$\text{value} = (-A_1 \times 2^0) + (A_2 \times 2^{-1}) + \ldots + (A_n \times 2^{1-n}),$$

where A . . . A are the bit values of the number. For example, the four-bit number 1010 would be interpreted to have the following value:

$$\text{value} = -1 + (0 \times 0.5) + (1 \times 0.25) + (0 \times 0.125) = -0.75$$

Several points need to emphasized:

1. The possible values using this scheme range from −1 to slightly less than 1.
2. Since the TSP50C1x instructions are all 8-bit by 14-bit multiply instructions, the lower 8 bits of the result are truncated.
3. Since the lower 8 bits of the result are truncated, many multiplications will give a zero result; for example:

(00 0000 0000 1111) × (0000 0011) = 00 0000 0000 0000 | 0010 1101
= 00 0000 0000 0000

6.4 Standby Mode

The TSP50C1x can be put in a low-power-dissipation standby mode by either executing a SETOFF instruction or by taking $\overline{\text{INIT}}$ low. If the device is placed in standby with the SETOFF instruction, it may be brought to an active state by pulsing $\overline{\text{INIT}}$ low and high. If the device is placed in a standby state by taking $\overline{\text{INIT}}$ low, it may be brought to an active state by taking $\overline{\text{INIT}}$ high.

When the device is placed in the standby state, output data is cleared, the I/O pins are placed in input mode, the program counter is cleared to zero, the registers are left in an undefined state, and the values stored in RAM are retained. The clock stops running and no instructions are executed until $\overline{\text{INIT}}$ goes from low to high.

6.5 Slave Mode

Setting bit 6 of the mode register high places the TSP50C1x in the slave mode. This specialized mode is intended for applications in which the TSP50C1x 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 A(0–7) ports depending on the value of B1.

B1 becomes a read/write select input. If B1 is low, data is written to the TSP50C1x when B0 goes low. If B1 is high, data may be read from the TSP50C1x when B0 goes low.

Port A becomes a general bidirectional port controlled by B0 and B1. Pin A(7) is used as a busy signal. If bit 7 in the output latch is set high by the software, A(7) is reset to a low state when B0 goes low to write data to the TSP50C1x.

Because A(7) is used as a busy flag, leaving only A(0–6) for data, normally only seven bits of data may be exchanged between the master and the slave in any one read operation from the TSP50C1x. In write operations to the TSP50C1x, all 8 pins of port A can be used to transfer data.

During read operations from the slave TSP50C1x, the master is responsible for maintaining its outputs connected to the TSP50C1x port A in a high-impedance state. Otherwise, bus contention results.

The TSP50C1x I/O ports must be configured in input mode for slave mode to work properly. Port A(7) may be put in output mode, if desired. It will then function as a handshaking line rather than a polled handshake bit.

Please note that simultaneous configuration of SLAVE and EXTROM is not allowed. The ten I/O lines cannot be arranged to give both capabilities.

6.5.1 Slave-Mode Write Operation

A typical sequence for an 8-bit write operation to the TSP50C1x in the slave mode is shown in Figure 6–5.

At the beginning of the operation, the TSP50C1x has a low in the A(7) 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/$\overline{\text{W}}$ high to indicate a read operation.
2. The master polls the output state of A(7) by pulsing STR (on B(0)) low and reading the state of A(7) while STR is low.
3. Eventually, the TSP50C1x completes processing any previous data or instructions from the master. When it does, it writes a one to the A(7) output latch.
4. When the master senses that A(7) has gone high, it sets the R/$\overline{\text{W}}$ signal low to indicate a write operation.
5. The master presents valid data to port A(0–6).
6. The master pulses STR (on B(0)) low, which causes the data on port A(0–6) pins to be latched to the port A input latch. The TSP50C1x hardware causes the A(7) output latch to be cleared to zero, indicating that the TSP50C1x has accepted the data.
7. The TSP50C1x polls the A(7) output latch. When it sees it go low, it knows that data is being written to the port A input latch.
8. The TSP50C1x polls the B0 (STR) input line. When B0 goes high, the write is complete, and the data in A0 is valid.
9. When it is ready to accept another command, the TSP50C1x writes a one to the A(7) output latch, thus starting another cycle.

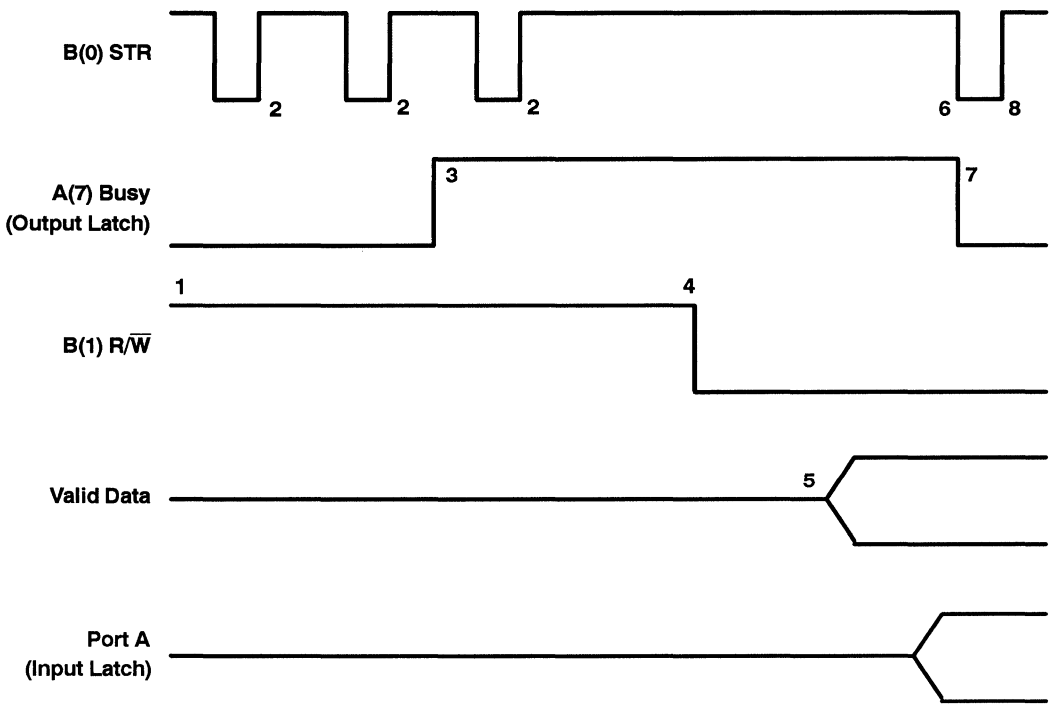


Figure 6–5. Slave-Mode Write Operation

6.5.2 Slave-Mode Read Operation

A typical sequence for an 8-bit read operation from the TSP50C1x in the slave mode is shown in Figure 6–6.

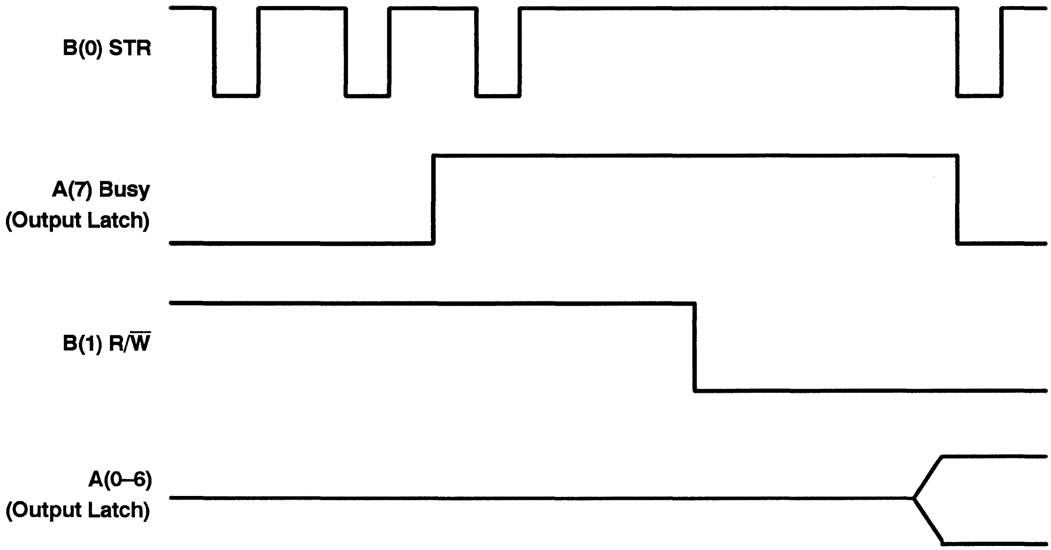


Figure 6–6. Slave-Mode Read-Then-Write Operation

At the beginning of the operation, the TSP50C1x has a low in the A(7) output latch. It has received a command or a request for information from the master. When the TSP50C1x is ready to respond, the data transfer occurs as follows:

1. The TSP50C1x writes the data to A(0–6) and a logic one to port A(7). The one on port A(7) is a signal that valid data is available in the pins connected to port A.
2. The master periodically polls port A. When it finds A(7) has gone high, it knows that A(0–6) contains valid data.
3. A(7) 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 A(7) output latch and finds it low, it knows that a new command from the master is in the port A latch.

6.6 TSP60C18 Interface

The TSP60C18 is 256 K-bit ROM organized internally as 16K-bits X 16 bits. It is designed specifically to provide additional low-cost ROM storage for the Texas Instruments family of speech chips.

6.6.1 External ROM Mode

Setting bit 4 of the mode register high places the TSP50C1x device in external ROM mode. When placed in this mode, the TSP50C1x port operation is modified to provide an efficient interface to the TSP60C18. The ports affected are summarized below:

> B(0) is dedicated as a strobe output. It should be configured as an output by the software. Its output value is the logical AND of the B(0) output latch and a hardware-generated strobe active signal. Software pulses this signal low to write addresses to the TSP60C18. Hardware pulses this signal low during GET instructions.
>
> A(7) is dedicated as a system clock signal going to the TSP60C18. It should be configured by software as an output with a logical one written to its output latch. Its value is the logical AND of the A(7) output latch and a clock that runs at one-fourth the rate of the master clock.

Control of other ports is necessary to complete communications with the TSP60C18, but the selection of which ports to use for which signal is optional.

NOTE: Simultaneous configuration of SLAVE and EXTROM is not allowed. The ten I/O lines cannot be arranged to give both capabilities.

6.6.2 TSP60C18 I/O Signals

The TSP60C18 has ten functional pins in addition to power and ground. Table 6–7 summarizes the function of each signal, and Table 6–8 details the pinout of the TSP60C18.

Table 6–7. TSP60C18 Pin Functional Descriptions

SIGNAL	DIRECTION	FUNCTION/ACTION
HCLB	Input	If this pin is low, the device is initialized and forced into an input mode (output buffers are put in the high-impedance state). This signal is not affected by the state of the CEB input.
CEB	Input	If this pin is high, the C(0–3) pins are unconditionally in the high-impedance state. This pin is provided to permit ROM expansion to greater than 1 Mbit.
STR	Input	When this pin is taken low, depending upon the state of the $R/\overline{W}$ signal, data is read from or an address is written to the TSP60C18.
$R/\overline{W}$	Input	When this pin is high, data is output from the device when STR goes low. When this pin is low, one nibble of the 16-bit address is input to the device when STR goes low.
C(0–3)	Input or Output	When STR goes low and $R/\overline{W}$ is low, the data present on these pins is latched into the device as one nibble of the four-nibble address. When STR goes low and $R/\overline{W}$ is high, one nibble of the currently addressed data is presented on these pins for output. C(0) is the least significant bit and C(3) is the most significant bit of the address/data nibble.
A0	Input	When this pin is low, the address that is loaded is understood to point directly to the data that is desired for output. When this pin is high, the address that is loaded is understood to point to a table entry that contains the address of the data that is desired for output. See Section 6.6.4 for more information.
SRCK	Input	Free-running system clock for internal sequential logic

Table 6–8. TSP60C18 Pinout

PIN		FUNCTION	PIN		FUNCTION
NAME	NO.		NAME	NO.	
A0	2	Address mode control pin	NC	5	No internal connection
C(0)	14	Address/data bit 0 (LSB)	NC	12	No internal connection
C(1)	15	Address/data bit 1	NC	13	No internal connection
C(2)	16	Address/data bit 2	$R/\overline{W}$	8	I/O direction control
C(3)	1	Address/data bit 3 (MSB)	SRCK	10	System clock
CEB	7	Chip enable input	STR	9	Chip enable strobe signal
HCLB	6	Hardware clear input	V_{DD}	3	Positive supply, 2.5 V to 6.5 V
NC	4	No internal connection	V_{SS}	11	Power return

6.6.3 TSP60C18 Addressing

The TSP60C18 address is a 16-bit address on 16-bit boundaries that provides addressing capabilities to 1M bit. Each TSP60C18 has a storage capability of 256K-bits. To achieve the full 1M-bit capability, the address space of each TSP60C18 is internally masked so that up to four TSP60C18s may be connected in parallel to produce a 1M-bit ROM system. When operated in this fashion, all like-numbered pins are connected together; the two most significant bits of the 16-bit address control which chip is addressed, and the remaining 14 bits control the relative address within the selected chip that is to be accessed.

6.6.4 TSP60C18 Addressing Modes

The TSP60C18 provides the following three addressing modes: 16-bit direct addressing, 16-bit indirect addressing, and 8-bit indirect addressing. The signal A0 determines which addressing mode is used.

When STR goes low to latch in the second and fourth nibbles of the address, A0 is sampled. As shown in Table 6–9, the state of A0 during the two samples determines the addressing mode.

Table 6–9. TSP60C18 Addressing Modes

A0 DURING		ADDRESS MODE
Nibble 2	Nibble 4	
0	0	16-bit direct address
0	1	16-bit indirect address
1	X	8-bit indirect address

If A0 is high as the second nibble of the address is latched in, no further nibbles are latched in; the two bytes that were clocked in are presumed to be the least significant byte of a two-byte address pointing to a 16-bit boundary in the ROM. The most significant byte of the address is zero. The ROM is in indirect address mode, meaning that the data located at the address that was clocked in is the address of the actual desired data. If any additional address nibbles are latched in, they are treated as the beginning of a new and different address.

If A0 is low as the second nibble of the address is latched in, the address is presumed to be a 16-bit address. The state of A0 is sampled as the fourth nibble of the address is being latched in to determine if the address is direct or indirect.

6.6.4.1 TSP60C18 Direct-Addressing Mode

If the TSP60C18 is loaded in the direct-addressing mode, the 16-bit address that is loaded is presumed to point directly to the desired data.

6.6.4.2 TSP60C18 Indirect-Addressing Mode

If the TSP60C18 is loaded in the indirect-addressing mode, the 8-bit or 16-bit address does not point directly to the desired data. Instead, it points to a location in the ROM that contains the address that points to the location of the desired data. The TSP60C18 then automatically sets the internal data pointer to the 16-bit address found in this location.

WARNING: Because the indirect-addressing mode is an internal function within each chip and not between chips, there are no special provisions made to use indirect addressing in multichip TSP60C18 systems. Unless the table data is repeated in each TSP60C18 device at the same lower 14-bit address location, the function will not work properly and device damage may result. If care is not taken to place identical tables within each chip, multiple devices may be enabled at the same time, causing bus contention on C(3–0).

As an example, assume that the ROM contains the data shown in Table 6–10.

Table 6–10. Indirect Address Example

ADDRESS	DATA
0000	05A2
0001	0200
0002	0302
.	.
.	.
.	.
0200	1234
0201	5678

If the address 0001 is latched into the TSP60C18 with the signal A0 placing the device in the indirect-addressing mode, the data that is fetched by subsequent GET operations is pointed to by the address found in location 0001, that is, the data contained in location 0200. The first word returned by subsequent GET statements is therefore 1234.

6.6.5 TSP60C18 Control

In the remaining discussion of the TSP60C18, the device is assumed to be connected to the TSP50C1x as shown in Figure 6–7. B0 *must* be used for the STR pin on the TSP60C18, and A7 *must* be used for SRCK. The interconnection of the remaining pins is optional depending on the application.

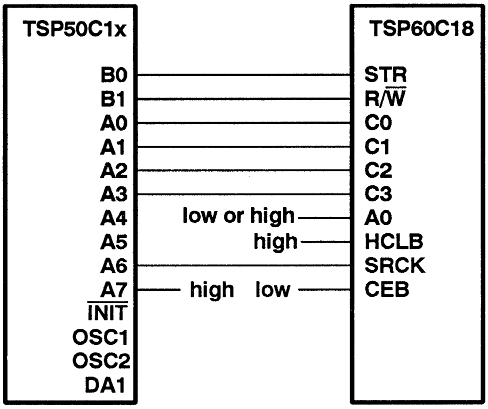


Figure 6–7. TSP60C18-to-TSP50C1x Hookup

6.6.5.1 Initialization of the TSP60C18

The TSP60C18 can be initialized with either hardware or software. The TSP60C18 can be initialized with hardware by taking the HCLB pin low and then high, which effectively does a power-up initialization of the device. This initializes the internal pointer counter, puts the device in load mode, and resets the internal chip enable. The desired starting address of data must still be loaded as described below.

In the hookup shown in Figure 6–7, the HCLB pin is not accessible to the TSP50C1x. The second way that the TSP60C18 can be initialized is through software, which is accomplished by the following sequence:

Configure Port B and A(0,1,2,3,7) as outputs

Take B(0), B(1), and A(7) high

Place TSP50C1x in external ROM mode

Execute LUAPS to initialize TSP50C1x

Do a dummy load address operation

Do a dummy read

Load a valid address

Burn 8 or 16 instruction cycles, depending on address mode

Prime the device with two GET2 commands.

The following three sections discuss the process.

6.6.5.2 Direct-Address Initialization of the TSP60C18

The TSP60C18 can be initialized in the 16-bit direct mode in the following manner:

- Hold A0 of the TSP60C18 low
- Configure Port B and A(0,1,2,3,7) as outputs
- Take B(0), B(1), and A(7) high
- Place the TSP50C1x in external ROM mode
- Execute LUAPS to initialize TSP50C1x
- Do a dummy read
  - Take $R/\overline{W}$ low
  - Pulse STR low
  - Take $R/\overline{W}$ high
  - Pulse STR low
- Load the valid address
  - Present the least significant nibble of the address on C0–C3
  - Pulse STR low
  - Present the second nibble of the address on C0–C3
  - Pulse STR low
  - Present the third nibble of the address on C0–C3
  - Pulse STR low
  - Present the most significant nibble of the address on C0–C3
  - Pulse STR low
- Burn eight instruction cycles
- Execute two GET2 instructions.

The TSP60C18 is now prepared to output data to the TSP50C1x in response to GET instructions. See Section 6.7.3 for a sample listing of a routine that performs this function.

6.6.5.3 8-Bit Indirect-Address Initialization of the TSP60C18

The TSP60C18 can be initialized in the 8-bit indirect mode in the following manner:

- Configure Port B and A(0,1,2,3,7) as outputs
- Take B(0), B(1), and A(7) high
- Place the TSP50C1x in external ROM mode
- Execute LUAPS to initialize TSP50C1x
- Do a dummy read
  - Take $R/\overline{W}$ low
  - Pulse STR low
  - Take $R/\overline{W}$ high

```
    Pulse STR low
Take A0 of the TSP60C18 high
Load the valid address
    Present the least significant nibble of the address on C0−C3
    Pulse STR low
    Present the second nibble of the address on C0−C3
    Pulse STR low
Burn 16 instruction cycles
Execute two GET2 instructions.
```

The TSP60C18 is now prepared to output data to the TSP50C1x in response to GET instructions. The data is pointed to by the table entry located at the address that was loaded.

6.6.5.4 16-Bit Indirect-Address Initialization of the TSP60C18

The TSP60C18 can be initialized in the 16-bit indirect mode in the following manner:

```
Configure Port B and A(0,1,2,3,7) as outputs
Take B(0), B(1), and A(7) high
Place the TSP50C1x in external ROM mode
Execute LUAPS to initialize TSP50C1x
Do a dummy read
    Take R/W̄ low
    Pulse STR low
    Take R/W̄ high
    Pulse STR low
Take A0 of the TSP60C18 low
Load the least significant byte of address
Present the least significant nibble of the byte on C0−C3
    Pulse STR low
    Present the most significant nibble of the byte on C0−C3
    Pulse STR low
Take A0 of the TSP60C18 high
Load the most significant byte of address
Present the least significant nibble of the byte on C0−C3
    Pulse STR low
    Present the most significant nibble of the byte on C0−C3
    Pulse STR low
Burn 16 instruction cycles
Execute two GET2 instructions.
```

The TSP60C18 is now prepared to output data to the TSP50C1x in response to GET instructions. The data is pointed to by the table entry located at the address that was loaded.

6.6.6 Placing the TSP60C18 In a Low-Power Standby Condition

The TSP60C18 can be placed in a low-power standby condition by removing the clock while the nodes of the device are in a precharged condition. This can be done in one of two ways.

1) Placing the TSP60C18 in a low-power mode by loading it with a partial address and maintaining R/$\overline{W}$ and STR high, as shown below:

    Configure Port B and A(0,1,2,3,7) as outputs

    Load B(0), B(1), and A(7) output ports with a logical 1

    Place the TSP50C1x in external ROM mode

    Load the partial address

        Take R/$\overline{W}$ low

        Pulse STR low

    Put the TSP50C1x in internal ROM mode

    Maintain B(0) and A(7) configured as outputs in the high state.

2) Placing the TSP60C18 in a low-power mode by loading it with a complete address and maintaining R/$\overline{W}$ low and STR high, as shown below:

    Configure Port B and A(0,1,2,3,7) as outputs

    Load B(0), B(1), and A(7) output ports with a logical 1

    Load A(0), A(1), A(2), and A(3) output ports with a logical 0

    Place the TSP50C1x in external ROM mode

    Load the complete address

        Take R/$\overline{W}$ low

        Pulse STR low 4 times

    Wait a minimum of 16 instruction cycles

    Take R/$\overline{W}$ high

    Take STR low

    Put the TSP50C1x in internal ROM mode

    Maintain B(0) low and B(1) high.

To bring the TSP60C18 to an active condition, do an initialization as previously discussed.

NOTE: The SETOFF instruction places all outputs in a high-impedance state. If a SETOFF instruction is executed to place the TSP50C1x in a low-power state, then pullup or pull-down resistors should be provided to maintain the TSP60C18 control lines in the correct state after the SETOFF is executed.

6.7 Use of the GET Instruction

The GET instruction is used to retrieve a bit stream from RAM, internal ROM, or external ROM. It allows the program to unpack speech data in a time-efficient manner. As shown in Figure 6–8, it is implemented through the use of a parallel-to-serial shift register.

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, external 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.

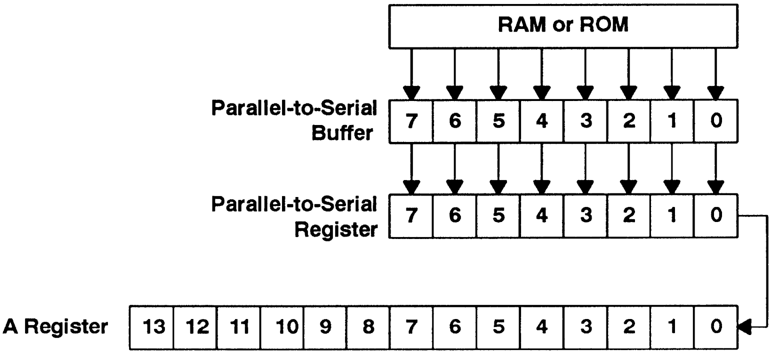


Figure 6–8. Register Connections for GET Instruction

If 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 on the fly 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. If the buffer did not need to be reloaded, the status bit is cleared.

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 6–9. 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 B7$_{16}$, and the A register contains all zeros. After the instruction, the least significant five bits of the P/S register are shifted into the A register. Because of the bit flip, the A register contains 1D$_{16}$ after the shift operation. The P/S register has only three valid bits left after the operation. If more than three bits are requested in the next GET operation, the P/S register is reloaded from the P/S buffer.

The source for the data is controlled by the EXTROM and RAMROM bits in the mode register as shown in Table 6–11.

Table 6–11. Mode Register Control of GET Data Source

MODE REGISTER BITS		DATA SOURCE
RAMROM	EXTROM	
0	0	Internal ROM
0	1	External ROM
1	0	Internal RAM
1	1	Internal RAM

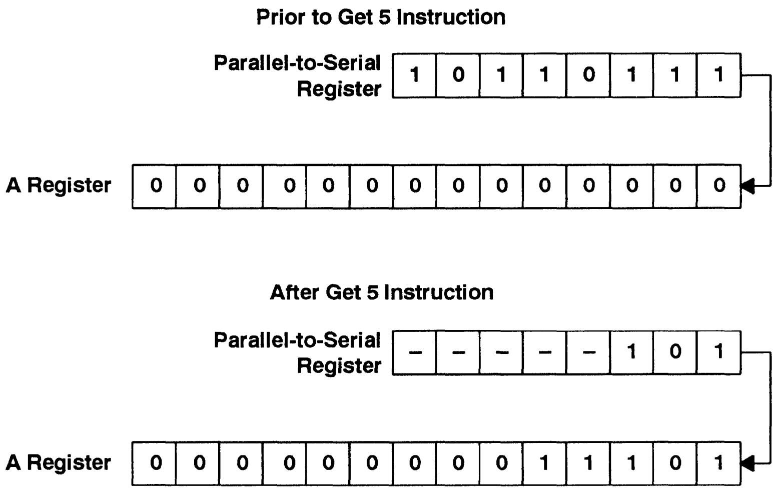


Figure 6–9. Parallel-to-Serial Operation for GET 5 Instruction

NOTE: Because of timing problems that may 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 LPC bit of the mode register is set.

Specifically, if the LPC bit is set and the first GET instruction is a GET 4 from external ROM or 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.

6.7.1 GET From Internal ROM

If both the RAMROM and EXTROM bits of the mode register are zero, the data source for GET instructions is the internal ROM. As detailed in Section 6.7, 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 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. If a nonsequential address is desired for a GET, repeat steps 1 and 2 for the new address.

6.7.2 GET From External ROM

If the RAMROM bit is cleared and the EXTROM bit is set in the mode register, the data source for GET instructions is the external ROM. As detailed in Section 6.7, the data is read into the A register in a byte-flipped form from the value stored in ROM, meaning that the LSB of the ROM data byte is shifted into the A register first. Because the external ROM needs to be initialized in addition to the TSP50C1x, the procedure is somewhat more complicated than that for the internal ROM case. See Section 6.6 for details on interfacing a TSP60C18 to the TSP50C1x.

The recommended sequence for preparing to GET from external ROM is as follows:

1. Configure all control lines as outputs
2. Place the TSP50C1x in external ROM mode
3. Execute LUAPS to initialize the counters in the TSP50C1x. When preparing to execute a GET from external ROM, the value in the A register during the LUAPS is unimportant.
4. Initialize the external ROM. The processor is now ready to execute a GET instruction starting at the address loaded to the ROM in this step.
5. If a nonsequential address is desired for a GET, repeat steps 3 and 4 for the new address.
6. Be very careful not to disturb the value on the control lines when not doing GET instructions.

NOTE: When in external ROM mode, only four or fewer bits may be fetched at a time. While GET 5, GET 6, GET 7, and GET 8 may work, the results are not guaranteed to be accurate because only four bits are fetched from the ROM at a time. If more than four bits are required, the preferred solution is to execute multiple GET instructions.

6.7.3 GET From Internal RAM

If the RAMROM bit is set in the mode register, the data source for GET instructions is the internal RAM. As detailed in Section 6.7, 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.

If 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 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 TSP50C1x 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 TSP50C1x and to load the first byte from RAM into the P/S register. When preparing to GET from RAM, the value in the A register during the LUAPS is unimportant. After this, 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. If status is set, then 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:

```
*         SAMPLE PROGRAM USING RAM GET
DTA       EQU       #10
*
          TCX       DTA       SET X REG TO POINT TO DTA
          TCA       #020      SET TO RAM MODE
          TAMODE
          LUAPS               SET UP PARALLEL TO SERIAL REG
          GET       8         DUMMY GET
          CALL      UPX
          .
          .
          .
          BR        LOOP
*
UPX       IXC
          RETN
```

6.8 External ROM Interface

The external ROM mode is designed to optimize the interface to the TSP60C18, although it is possible to use other devices for external memory. This section provides the information needed to interface to other devices. Appendix C contains an external ROM initialization routine.

When the TSP50C1x is placed in external ROM mode, the B(0) and A(7) pins assume a specialized function. The B(0) pin becomes the strobe output and A(7) becomes a clock signal running at one-fourth the master clock rate. A(0–3) are used to latch data in from the ROM. Other control signals needed by the ROM may be assigned to the other output ports at the designer's discretion. These other signals should be software-controlled.

The signal on the B(0) pin is the logical AND of the value written to the data output latch of B(0) and a hardware-generated strobe. If the P/S buffer is found to be empty during a GET instruction, the hardware-generated strobe pulses low, and the nibble present on A(0–3) is loaded into the P/S buffer.

The signal on the A(7) pin is the logical AND of the value written to the data output latch of A(7) and a free-running 2.4-MHz clock (if the master clock is 9.6 MHz).

6.9 Generating Tones Using PCM

The TSP50C1x can generate speech and tones using pulse code modulation (PCM) as well as LPC. When using PCM, a periodically sampled waveform may 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.

PCM mode is enabled by setting the PCM bit in the mode register high and the LPC bit in the mode register low. Once PCM mode is enabled, the software must load the DAC with a value every 30 or 60 instruction cycles using the TASYN instruction.

6.9.1 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 6–10. TASYN transfers the data in the A register to a temporary buffer register whose contents are periodically transferred to the DAC once every 30 instruction cycles.

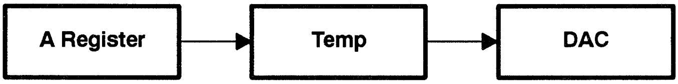


Figure 6–10. Operation of TASYN in PCM Mode

The data in the A register should be in a modified two's complement format, described as follows:

The A register is 14 bits long. When the contents of the A register is transferred to the DAC, the bits are interpreted as shown in Figure 6–11. The least significant bits (bits 0 and 1) are ignored. They are normally set to zero. The two most significant bits (bits 12 and 13) are the sign bits. If they are 1, then the value being loaded to the DAC is negative. The remaining 10 bits of the A register (bits 2 – 11) contain magnitude data. The greatest magnitude is ± 480. Any greater magnitude is clipped. The relative weights of the magnitude bits are listed in Table 6–12.

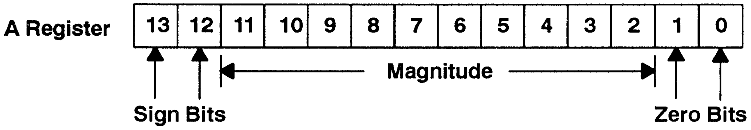


Figure 6–11. Format of Data in A Register Before TASYN

Table 6–12. Relative Weights of DAC Magnitude Bits

Bit Position	11	10	9	8	7	6	5	4	3	2
Relative Weight	256	128	64	32	16	8	4	2	1	1

6.9.2 Timing Considerations in PCM Mode

While in PCM mode, the contents of the DAC are refreshed every 30 instruction cycles. The new data must be loaded with TASYN instructions at an integer multiple of this rate. If the new data is not synchronous with the 30-cycle refresh rate, samples may be missed or doubled, thereby resulting in tone deterioration.

There are two approaches to keeping the TASYN instruction synchronous with the DAC refresh. The first (and normally preferred) approach is to use the level-1 interrupt to synchronize the program. When the mode register is set with the PCM bit high, the LPC bit low, and the INT1 bit high, a level-1 interrupt is generated every 30 instruction cycles. If the interrupt-service routine is longer than 30 instruction cycles, the interrupt is generated every 60 instruction cycles. The second approach is to program a tight loop using exactly 30 or 60 instruction cycles per loop. This method will work and avoids the instruction-cycle overhead associated with the interrupt but is more difficult to program reliably.

6.9.3 DTMF Program Walk-Through

This section contains a walk-through of the DTMF (dual-tone multifrequency or touch-tone) program found in Appendix D. The program generates a series of DTMF tones triggered by port A(0) going high and terminated by A(0) going low.

Following are the RAM locations used in the program. For each of the two sine waves that are added together to make the DTMF tone, a register that contains the angular difference between each data point (PERIOD1 and PERIOD2) and a register that contains the current angle for each frequency (TIME1 and TIME2) are required. Additionally, a temporary buffer is required to hold the intermediate result (PCMBUF).

In this application, each of these registers must be twelve bits long to maintain sufficient accuracy, which means that they must be in the lower 16 locations of RAM. These are the same registers that are used in the LPC routines, which is acceptable as long as LPC is not being executeed at the same time as PCM.

```
      *
      *    PCM register variables
      *
0000  PERIOD1     EQU     #00       -Period of 1st Wave
0001  TIME1       EQU     #01       -Cumulative angle of 1st wave
0002  PERIOD2     EQU     #02       -Period of 2nd Wave
0003  TIME2       EQU     #03       -Cumulative angle of 2nd wave
0004  PCMBUF      EQU     #04       -Intermediate data buffer
      *
      *
      *    LPC status variable locations
      *
0010  MODE_BUF    EQU     #10          ;Mode register buffer
      *
      *   Device Constants
      *
007F  MAX_RAM     EQU     #7F       -Highest RAM location
      *
      *   MODE Register Bit Definitions
      *
0001  ENA1        EQU     #01       -Enable Level 1 interrupt
0002  LPC         EQU     #02       -Enable LPC synthesis
0004  PCM         EQU     #04       -Enable PCM synthesis
0008  ENA2        EQU     #08       -Enable Level 2 interrupt
0010  EXTROM      EQU     #10       -Set external ROM mode
0020  RAMROM      EQU     #20       -Enable GETs from RAM
0040  MASTER      EQU     #40       -Master/Slave Toggle
0080  UNV         EQU     #80       -Enable Unvoiced excitation
```

Next is the DTMF frequency definition table and the sine wave look-up table. Each line in the DTMF frequency definition table contains four bytes, two bytes for each of the two frequencies that make up a DTMF tone. These two-byte numbers represent the angular interval by which the sine wave must be incremented between samples. For example, if the sample rate is 10,000 samples / second, the sine wave table must be accessed at intervals of 25.092 degrees in order to produce a 697-Hz sine wave.

$$\frac{697 \text{ cycles/second} \times 360 \text{ degrees/cycle}}{10,000 \text{ samples/second}} = 25.092 \text{ degrees/sample}$$

Note that the 10,000 samples / second figure in this equation assumes a 9.6-MHz crystal and a level-1 interrupt code length of between 30 and 60 instruction cycles. The following table contains sample rates based on different assumptions:

Table 6–13. Sample Rates

CRYSTAL	LEVEL-1 INTERRUPT CODE LENGTH	
	< 30 INSTRUCTION CYCLES	< 60 INSTRUCTION CYCLES
7.68 MHz	16,000 samples/second	8,000 samples/second
9.6 MHz	20,000 samples/second	10,000 samples/second

The sine wave table contains information for 32 points of a sine wave, spaced 11.25 degrees apart. Therefore, the number that was calculated above is divided by 11.25 degrees to determine the number of sine wave table entries to skip between samples.

$$\frac{25.092 \text{ degrees/sample}}{11.25 \text{ degrees/entry}} = 2.230 \text{ entries/sample}$$

Finally, the number is normalized, truncated, and converted to a two-byte hexadecimal value before placing it in the DTMF frequency definition table.

$$\text{TRUNC } (2.230 \times 128) = 285_{10} = 011D_{16}$$

The first byte on each line of the sine wave table is an amplitude and the second byte is an amplitude offset. The offset byte is multiplied by a fractional value and added to the amplitude byte to allow interpolation of the sine wave values. Because of the way the interpolation is performed, the fractional value for odd-numbered table entries will be negative and the fractional value for even-numbered table entries will be positive. Therefore, the first line of the table will have a positive fractional value that is multiplied by the offset byte and then added to the amplitude byte to allow sin (0°) up through sin (11.25°) to be represented. The second line of the table will have a negative fractional value that is multiplied by the offset byte and then added to the amplitude byte to allow sin (22.5°) down through sin (11.25°) to be represented.

```
0102 0024   80  DTMF        RBYTE   #01,#81,#02,#23   –zero = 941 Hz+1336 Hz
0103 0028   80              RBYTE   #01,#1D,#01,#EF   –One  = 697 Hz+1209 Hz
0104 002C   80              RBYTE   #01,#1D,#02,#23   –two  = 697 Hz+1336 Hz
0105 0030   80              RBYTE   #01,#1D,#02,#5D   –three= 697 Hz+1477 Hz
0106 0034   80              RBYTE   #01,#3B,#01,#EF   –four = 770 Hz+1209 Hz
0107 0038   80              RBYTE   #01,#3B,#02,#23   –five = 770 Hz+1336 Hz
0108 003C   80              RBYTE   #01,#3B,#02,#5D   –six  = 770 Hz+1477 Hz
0109 0040   80              RBYTE   #01,#5D,#01,#EF   –seven= 852 Hz+1209 Hz
0110 0044   80              RBYTE   #01,#5D,#02,#23   –eight= 852 Hz+1336 Hz
0111 0048   80              RBYTE   #01,#5D,#02,#5D   –nine = 852 Hz+1477 Hz
0112            *
0113            *     Digitized sine wave table
0114            *
0115 004C   00  SINEW       BYTE    #00,#19  0 degrees—>11.25 degrees
0116 004E   31              BYTE    #31,#18  11.25 degrees—>22.5  degrees
0117 0050   31              BYTE    #31,#16  22.5  degrees—>33.75 degrees
•
•
•
0144 0086   CF              BYTE    #CF,#16  326.25 degrees—>337.5  degrees
0145 0088   CF              BYTE    #CF,#18  337.5  degrees—>348.75 degrees
0146 008A   00              BYTE    #00,#19  348.75 degrees—>360    degrees
```

Following is the executable code. The code that is used to clear RAM and the mode register is not shown. After initializing the device, the program invokes the subroutine that generates the tone, passing the table index that defines the tone in the A register.

```
0150 008C   6E  GOGO        TCA     0          –Tone 'Zero'
     008D   00
0151 008E   00              CALL    DO_PCM
     008F   B5
0152            *
0153 0090   6E              TCA     1          –Tone 'One'
     0091   01
```

```
0092   00              CALL    DO_PCM
0093   B5
.
.
.
00AC   6E              TCA     8        –Tone 'Eight'
00AD   08
00AE   00              CALL    DO_PCM
00AF   B5
           *
00B0   6E              TCA     9        –Tone 'Nine'
00B1   09
00B2   00              CALL    DO_PCM
00B3   B5
           *
00B4   3F              SETOFF
```

The following code is used to wait until DTMF tone generation is requested. The program loops until A(0) goes high.

```
00B5   62  DO_PCM      TCX     #80      –Point to port A
00B6   80
00B7   66              TSTCM   #01      –Loop until A(0)
00B8   01
00B9   40              BR      GO_PCM      goes high
00BA   BD
00BB   40              BR      DO_PCM
00BC   B5
```

Since each table entry in the DTMF definition table is four bytes long, the value of the table index is quadrupled by left shifting it twice. Then the address of the start of the table is added, and a LUAPS is executed to point the speech address register to the desired table entry. The program uses two GET 8 instructions to fetch each number.

```
00BD   2E  GO_PCM      SALA             –Adjust value to
00BE   2E              SALA                table index
00BF   70              ACAAC   DTMF     –Add offset of table
00C0   24
00C1   6C              LUAPS            –Point to table entry

00C2   37              GET     8        –Get first frequency
00C3   37              GET     8           period
00C4   6A              TAMD    PERIOD1  –Store it away
00C5   00

00C6   37              GET     8        –Get second frequency
00C7   37              GET     8           period
00C8   6A              TAMD    PERIOD2  –Store it away
00C9   02
```

The program initializes other necessary RAM locations and sets the mode register to enable PCM and level-1 interrupt.

```
0210 00CA   2F             CLA                    —Clear cumulative data
0211 00CB   6A             TAMD     TIME1
     00CC   01
0212 00CD   6A             TAMD     TIME2
     00CE   03
0213
0214 00CF   62             TCX      MODE_BUF  —Turn on PCM and INT1
     00D0   10
0215 00D1   64             ORCM     PCM
     00D2   04
0216 00D3   64             ORCM     ENA1
     00D4   01
0217 00D5   11             TMA
0218 00D6   1D             TAMODE
```

The actual PCM code is in the interrupt-service routine. When the program is not executing PCM code, A(0) is continually polled. When A(0) goes low, the program disables PCM and returns for the next tone.

```
0220 00D7   62  L1         TCX      #80       —Loop until A(0)
     00D8   80
0221 00D9   66             TSTCM    #01          goes low
     00DA   01
0222 00DB   40             BR       L1
     00DC   D7
0223
0224 00DD   62             TCX      MODE_BUF  —Turn off PCM and INT1
     00DE   10
0225 00DF   65             ANDCM    ~PCM
     00E0   FB
0226 00E1   65             ANDCM    ~ENA1
     00E2   FE
0227 00E3   11             TMA
0228 00E4   1D             TAMODE
0229 00E5   3D             RETN
```

Following is the level-1 interrupt-service routine, INTPCM. This code performs the actual PCM calculations, which are done twice, once for each of the two sine waves. Then the results are summed together and transferred to the DAC buffer with the TASYN instruction. Because the interrupt-service routine is longer than 30 instruction cycles but less than 60 instruction cycles, it is invoked every 60 instruction cycles.

First the delta angle is added to the cumulative angle to generate a new cumulative angle.

```
0234 00E6   3B  INTPCM     INTGR
0235 00E7   20             CLX
0236
0237 00E8   14             TMAIX              —Add delta angle to
0238 00E9   28             AMAAC                 cumulative angle
```

```
0239
0240 00EA   16            TAM                -Save cumulative angle
0241 00EB   11            TMA                -Discard high bits of cum
```

The cumulative angle is shifted right seven bits in order to strip off its fractional part. The result is shifted left one bit to adjust for the two-byte size of each sine wave table entry. The address of the start of the table is then added to get the address of the desired table entry.

```
0242
0243 00EC   68            AXCA    01         -right shift 7 bits
     00ED   01
0244 00EE   2E            SALA               -Left 1 bit
0245 00EF   70            ACAAC   SINEW      -Add table offset
     00F0   4C
```

The sine-wave amplitude byte is put into the B register and the offset byte is put into the A register. The offset byte is multiplied by the fractional part of the cumulative angle and the result is added to the amplitude byte to interpolate between points. The SALA4 instruction correctly positions the value in the A register for transfer to the DAC buffer. This intermediate value is scaled for twist and then saved in PCMBUF before calculating the other wave.

```
0247 00F1   3C            EXTSG
0248 00F2   6D            LUAB               -get data point
0249 00F3   3A            IAC
0250 00F4   6B            LUAA               -get slope between points
0251 00F5   39            AXMA               -interpolate slope
0252 00F6   2C            ABAAC              -add interpolated slope
0253 00F7   1B            SALA4                 and scale for DAC
0254 00F8   68            AXCA    #78        -Scale value for twist
     00F9   78
0255
0256 00FA   6A            TAMD    PCMBUF     -Save intermediate data
     00FB   04
```

The only difference between the calculation of the first wave and the second wave is that the second wave is not scaled for twist. After both waves have been calculated, the result for the second wave is placed in the B register, the result for the first wave is retrieved to the A register, and the two values are added together. The result is divided by two to correctly scale it. TASYN is used to transfer the result to the DAC.

```
0279 010E   1A            TAB                -Store 2nd data point
0280
0281 010F   21            IXC                -Retrieve 1st data point
0282 0110   11            TMA
0283
0284 0111   2C            ABAAC              -Sum two waves together
0285 0112   15            SARA                  and normalize
0286 0113   1C            TASYN              -transfer data to D/A
0287 0114   3E            RETI
```

7 Customer Information

7.1 Development Cycle

The TSP50C1x development cycle is more complex than microprocessor development, because it adds speech development to the normal microprocessor development cycle. (Figure 7–1). The software design cycle is similar to that for other microprocessors. Speech development is discussed in Appendix A.

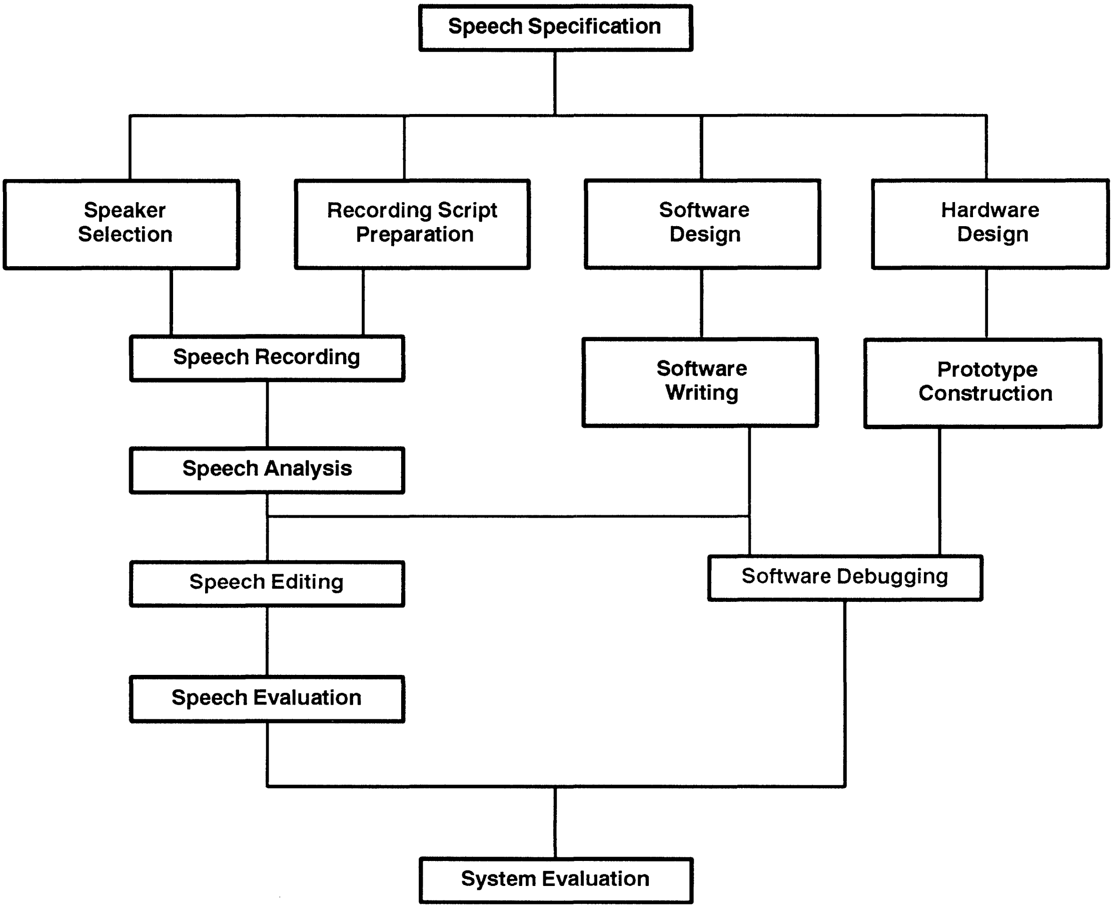


Figure 7–1. Speech Development Cycle

7.2 Summary of Speech Development/Production Sequence

The following is a summary of the speech development/production sequence:

1. For TI to accept a custom device program, the customer must submit a new product release form (NPRF) to TI. This form describes the custom features of the device (e.g., customer information, prototype and production qualities, symbolization, etc.). The NPRF will be completed by product engineering and product marketing personnel within TI. A copy of the NPRF can be found on pages 7–8 and 7–10.
2. TI generates the prototype photomask and processes, manufactures, and tests 25 prototype devices for shipment to the customer. Limited quantities in addition to the 25 prototypes may 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 25 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 25 prototype devices is incurred by the customer.
5. A minimum purchase might be required during the first year of production.

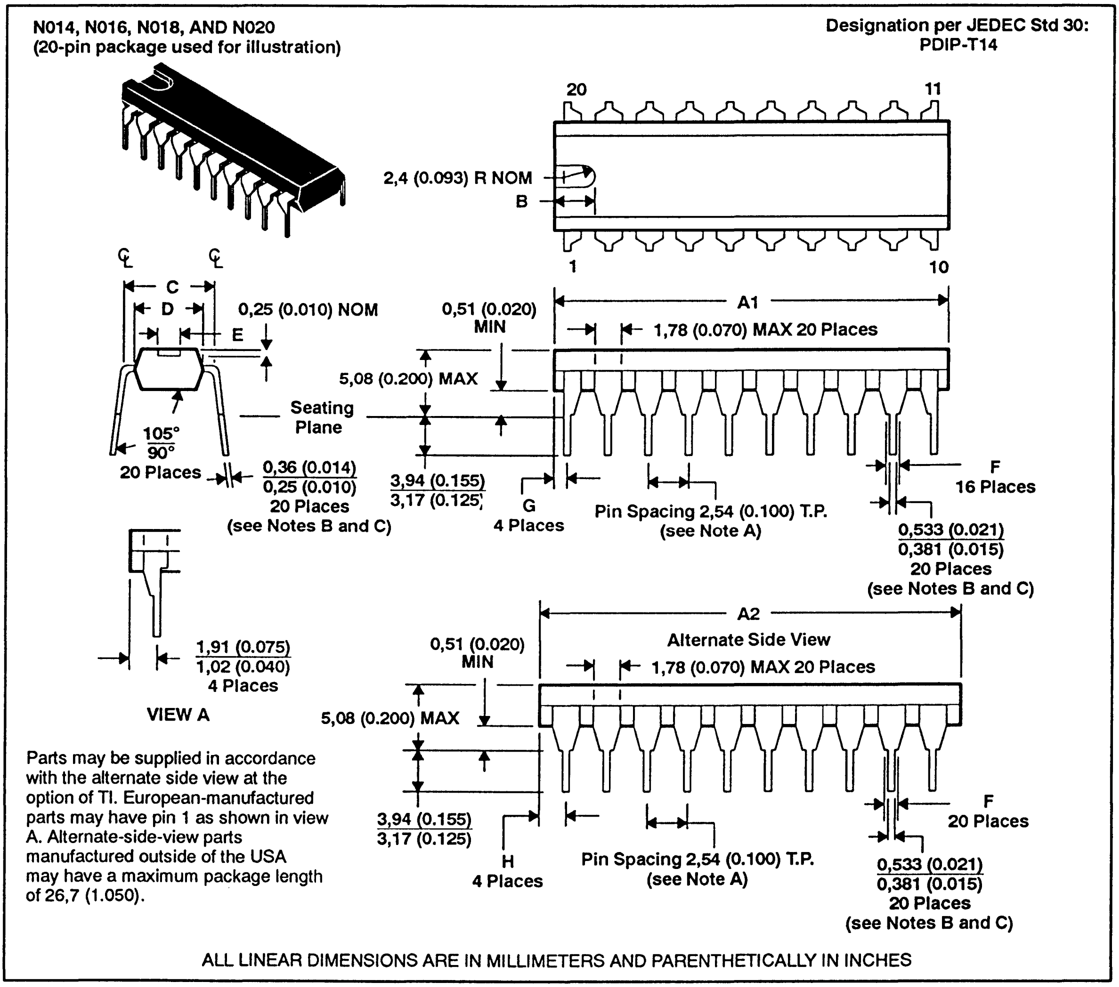
NOTE: Texas Instruments recommends that prototype devices not be used in production systems because their expected end-use failure rate is undefined but is predicted to be greater than standard qualified production.

7.3 N016 300-Mil Plastic Dual-In-Line Package

The dual-in-line package of the TSP50C10/11/14 (Figure 7-2) consists of a circuit mounted in a lead frame and encapsulated within an electrically nonconductive plastic compound. The compound will withstand soldering temperature with no deformation, and circuit performance characteristics will remain stable when operated in high-humidity conditions. The package is intended for insertion in mounting-hole rows on 7,62 (0.300) centers. 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 soldered assembly.

MECHANICAL DATA

N014, N016, N018, and N020
300-mil plastic dual-in-line package



N014, N016, N018, AND N020
(20-pin package used for illustration)

Designation per JEDEC Std 30:
PDIP-T14

Parts may be supplied in accordance with the alternate side view at the option of TI. European-manufactured parts may have pin 1 as shown in view A. Alternate-side-view parts manufactured outside of the USA may have a maximum package length of 26,7 (1.050).

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

NOTES: A. Each pin centerline is located within 0,25 (0.010) of its true longitudinal position.

B. This dimension does not apply for solder-dipped leads.

C. When solder-dipped leads are specified, dipped area of the lead extends from the lead tip to at least 0,51 (0.020) above seating plane.

Figure 7–2. TSP50C10/11/14 16-Pin N Package

N014, N016, N018, and N020
300-mil plastic dual-in-line package (continued)

DIM	PIN	14	16	18	20
A	MIN	18,0 (0.710)	(see Note A)	(see Note A)	23,22 (0.914)
	MAX	19,8 (0.780)	19,8 (0.780)	23,4 (0.920)	24,77 (0.975)
B	NOM	2,8 (0.110)	2,8 (0.110)	4,06 (0.160)	2,80 (0.110)
C	MIN	7,37 (0.290)	7,37 (0.290)	7,37 (0.290)	7,37 (0.290)
	MAX	7,87 (0.310)	7,87 (0.310)	7,87 (0.310)	7,87 (0.310)
D	MIN	6,10 (0.240)	6,10 (0.240)	(see Note A)	6,10 (0.240)
	MAX	6,60 (0.260)	6,60 (0.260)	6,99 (0.275)	7,11 (0.280)
E	NOM	2,0 (0.080)	2,0 (0.080)	2,03 (0.080)	2,0 (0.080)
F	MIN	0,84 (0.033)	0,84 (0.033)	0,89 (0.035)	0,84 (0.033)
G	MIN	(see Note B)	0,38 (0.015)	(See Note B)	1,68 (0.066)
	MAX	(see Note B)	1,65 (0.065)	(see Note B)	0,22 (0.009)
H	MIN	2,54 (0.100)	1,02 (0.040)	0,23 (0.009)	0,38 (0.015)
	MAX	1,52 (0.060)	2,41 (0.095)	1,91 (0.075)	1,27 (0.050)

NOTES: A. This packaging characteristic is not specified.

B. The 14-pin and 18-pin plastic dual-in-line package is only offered with the external pins shaped in their entirety.

7.4 FN068 68-Lead Plastic Leaded Chip Carrier (PLCC) Package

The 68-lead plastic chip carrier package, which is available only for the TSP50C12, consists of a circuit mounted on a lead frame and encapsulated within an electrically nonconductive plastic compound. The compound withstands soldering temperatures with no deformation, and circuit performance characteristics remain stable when the device is operated in high-humidity conditions. The package is intended for surface mounting on solder lands with 1,27 (0.050) centers. Leads require no additional cleaning or processing when used in soldered assembly.

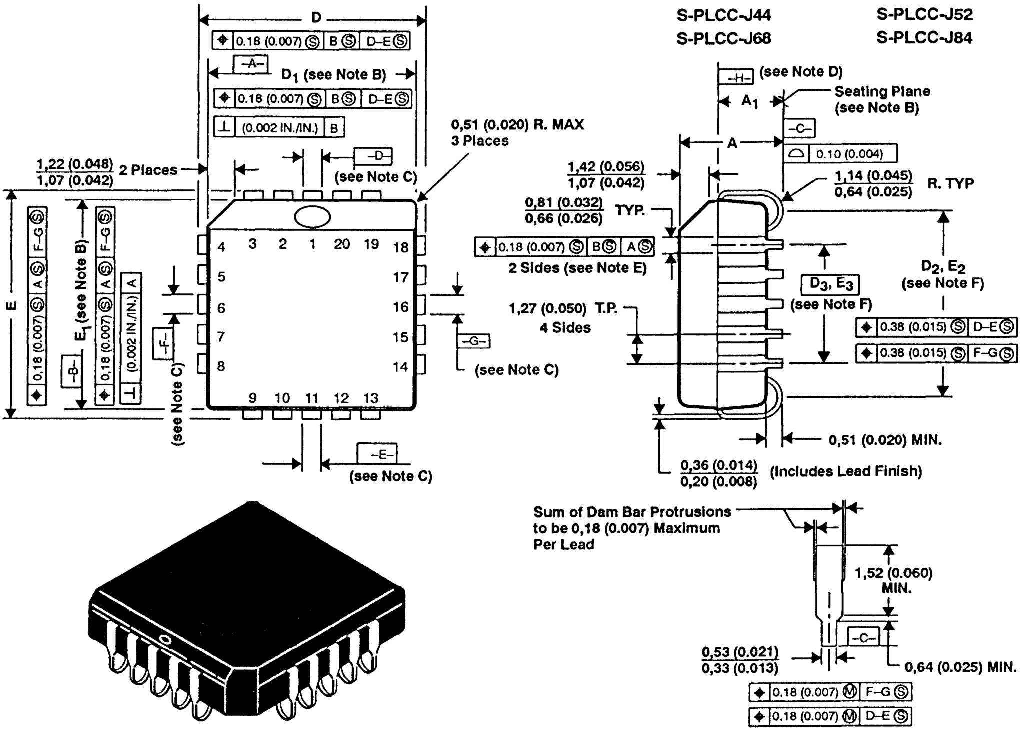
MECHANICAL DATA

FN020, FN028, FN044, FN052, FN068, and FN084 plastic J-leaded chip carrier

FN020, FN028, FN044, FN052, FN068, and FN084
(20-PIN package used for illustration)

Designation per JEDEC Std 30:

S-PLCC-J20	S-PLCC-J28
S-PLCC-J44	S-PLCC-J52
S-PLCC-J68	S-PLCC-J84



(see table on following page for additional dimensions)

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

NOTES:
- A. All dimensions conform to JEDEC Specification MO-047AA/AF. Dimensions and tolerancing are per ANSI Y14.5M – 1982.
- B. Dimensions D_1 and E_1 do not include mold flash protrusion. Protrusion shall not exceed 0,25 (0.010) on any side. Centerline of center pin each side is within 0,10 (0.004) of package centerline by dimension B. The lead contact points are planar within 0,10 (0.004).
- C. Datums D – E and F – G for center leads are determined at datum – H –.
- D. Datum – H – is located at top of leads where they exit plastic body.
- E. Location of datums – A – and – B – to be determined at datum – H –.
- F. Determined at seating plane – C –.

Figure 7–3. TSP50C12 68-Lead PLCC Package

WARNING:

When reflow soldering is required, refer to page 7–6 for special handling instructions.

FN020, FN028, FN044, FN052, FN068, and FN084 plastic J-leaded chip carrier (continued)

JEDEC	NO.OF	A		A_1		D, E		D_1, E_1		D_2, E_2		D_3, E_3
OUTLINE	PINS	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	BASIC
MO-047AA	20	4,19 (0.165)	4,57 (0.180)	2,29 (0.090)	3,05 (0.120)	9,78 (0.385)	10,03 (0.395)	8,89 (0.350)	9,04 (0.356)	7,37 (0.290)	8,38 (0.330)	5,08 (0.200)
MO-047AB	28	4,19 (0.165)	4,57 (0.180)	2,29 (0.090)	3,05 (0.120)	12,32 (0.485)	12,57 (0.495)	11,43 (0.450)	11,58 (0.456)	9,91 (0.390)	10,92 (0.430)	7,62 (0.300)
MO-047AC	44	4,19 (0.165)	4,57 (0.180)	2,29 (0.090)	3,05 (0.120)	17,40 (0.685)	17,65 (0.695)	16,51 (0.650)	16,66 (0.656)	14,99 (0.590)	16,00 (0.630)	12,70 (0.500)
MO-047AD	52	4,19 (0.165)	5,08 (0.200)	2,29 (0.090)	3,30 (0.130)	19,94 (0.785)	20,19 (0.795)	19,05 (0.750)	19,20 (0.756)	17,53 (0.690)	18,54 (0.730)	15,24 (0.600)
MO-047AE	68	4,19 (0.165)	5,08 (0.200)	2,29 (0.090)	3,30 (0.130)	25,02 (0.985)	25,27 (0.995)	24,13 (0.950)	24,33 (0.958)	22,61 (0.890)	23,62 (0.930)	20,32 (0.800)
MO-047AF	84	4,19 (0.165)	5,08 (0.200)	2,29 (0.090)	3,30 (0.130)	30,10 (1.185)	30,35 (1.195)	29,21 (1.150)	29,41 (1.141)	27,69 (1.090)	28,70 (1.130)	25,40 (1.000)

NOTES A: All dimensions conform to JEDEC Specification MO-047AA/AF. Dimensions and tolerancing are per ANSI Y14.5M – 1982.
F: Determined at seating plane $\boxed{-\mathrm{C}-}$.

TSP50C12 (PLCC) Reflow Soldering Precautions

Recent tests have identified an industry-wide problem experienced by surface mounted devices exposed to reflow soldering temperatures. The problem involves a package cracking phenomenon sometimes experienced by large (e.g., 68-lead) plastic leaded chip carrier (PLCC) packages during surface mount manufacturing. This phenomenon can occur if the TSP50C12 is exposed to uncontrolled levels of humidity prior to reflow solder. This moisture can flash to steam during solder reflow, causing sufficient stress to crack the package and compromise device integrity. If the TSP50C12 is being socketed, *no* special handling precautions are required. In addition, once the device is soldered into the board, *no* special handling precautions are required.

In order to minimize moisture absorption, TI ships the TSP50C12 in dry pack shipping bags with a RH indicator card and moisture absorbing desiccant. These moisture-barrier shipping bags will adequately block moisture transmission to allow shelf storage for 12 months from date of seal when stored at less than 60% relative humidity (RH) and less than 30°C. Devices may be stored outside the sealed bags indefinitely if stored at less than 25% RH and 30°C.

Once the bag seal is broken, the devices should be stored at less than 60% RH and 30°C as well as reflow-soldered within two days of removal. In the event that either of the above conditions is not met, TI recommends these devices be baked in a clean oven at 125°C and 10% maximum RH for 24 hours. This restores the devices to their dry packed moisture level.

NOTE: Shipping tubes will not withstand 125°C baking process. Devices should be transferred to a metal tray or tube before baking. Standard ESD precautions should be followed.

In addition, TI recommends that the reflow process not exceed two solder cycles and the temperature not exceed 220°C.

If you have any additional questions or concerns, please contact your local TI representative.

7.5 Ordering Information

Because the TSP50C1x are custom devices, they receive a distinct identification as follows:

CSM	1XXXX	X	X
Gate Code CSM – Custom Synthesizer With Memory	ROM Code	Revision Letter	Package or Die N – Plastic Dip Y – Die FN – PLCC

7.6 New Product Release Form (TSP50C1x)

The new product release form is simply a form that is used to track and document all the steps involved in implementing a new speech code onto one of the parent speech devices. Blank TSP50C1x forms are provided in Sections 7.6.1 through 7.6.3 (note that the addresses on these forms are subject to change). To initiate this implementation process, the customer must begin either by obtaining one of these new product release forms (NPRFs) from their local TI field sales office or by generating one of their own forms using the blank forms as a guide. The next step is to complete Section 1. As seen on the blank forms, Section 1 allows the customer to choose the parent device for their particular code, as well as the options pertinent to the parent device they wish to use. Section 1 also allows the customer to choose their own customer part number used for ordering their parts. If no customer part number is indicated, then TI defaults to the CSM1xxxxxx 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, the customer must submit the NPRF (along with their speech code) to the speech products group via their local TI field sales office.

Once the speech products group receives the speech code and the NPRF from the customer, they will take 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 the customer for recheck. This recheck ensures the confidence that the PG output file was generated correctly. Along with the reconstructed speech code, the NPRF is also returned to the customer with Section 2 completed by TI. In this section, TI assigns their own CSMxxxxxx part number and, in the case of packaged devices, TI also proposes a symbol format to the customer. If the customer wishes to deviate from the suggested symbol format, they must consult TI for requested changes.

After the customer verifies the reconstructed speech code and also accepts the proposed symbol format, they 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. The customer then needs to send or fax the NPRF to the speech products group via the local TI field sales office. TI should have the prototypes shipped to the customer approximately six weeks after receiving the NPRF with Section 3 signed. Once the customer receives these prototypes, they need to verify the functionality of the prototypes, sign Section 4, and send the NPRF (with Section 4 signed) back to TI. At this point, the customer can start ordering production units.

7.6.1 New Product Release Form for TSP50C10A and TSP50C11A

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.

Division:______________________________Company : ______________________________

Project Name : ____

Purchase Order # : ____

Management Contact : ______________________________ Phone : (   ) ____________

Technical Contact : ________________________________ Phone : (   ) ____________

Customer Part Number : ______________________________

Generic TI Part Number (Check one) :

_____ TSP50C10A

_____ TSP50C11A

D/A Output (Check one) :

_____ 2 pin push pull

_____ single pin single ended

_____ single pin double ended

Package Type (Check one) :

_____ N (plastic)

_____ 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

This section is to be completed by TI and approved by the customer. (Customer approval in Section 3)

TOP SIDE SYMBOLIZATION: (PACKAGED DEVICES ONLY)

```
 _______________________________________
| TI                                    |    YWW      = DATE CODE
|  LOGO         YWWLLLT                 |     LLL     = LOT TRACE CODE
>               {OPTIONAL 13 CHAR}      |      T      = ASSY SITE
|               {OPTIONAL 11 CHAR}      |   {FIRST LINE REQUIRED}
|                                       |
 _______________________________________
```

****

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 device 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 2B on all devices with the part number indicated in 2A.

By : ____ Title : ______________________

Date : ______________________

****

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: Bob Steel
P.O. Box 655303, M/S 8211
Dallas, TX 75265

or Fax to: (214) 997–3471
Attn. Bob Steel

7.6.2 New Product Release Form for TSP50C12

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.

Division:____________________________Company : ____________________________________
Project Name : ____
Purchase Order # : ____
Management Contact : ___________________________________ Phone : (   ) ______________
Technical Contact : ______________________________________ Phone : (   ) ______________

Customer Part Number : ___________________________________

D/A Output (Check one) :

_____ 2 pin push pull

_____ single pin double ended

LCD Drive :

_____ Type A, Fast

_____ Type B, Slow

Oscillator :

_____ RC (resistor/capacitor)

_____ CR (ceramic resonator)

Package : _____ Die Form

****

SECTION 2. ASSIGNMENT OF TI PRODUCTION PART NUMBER

This section is to be completed by TI.

TI Part Number: _____

****

SECTION 3. AUTHORIZATION TO GENERATE MASKS, PROTOTYPES, AND RISK UNITS

This section is to be signed by the customer and sent to TI.

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.

By : ____ Title : ______________________

Date : ______________________

****

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: Bob Steel
P.O. Box 655303, M/S 8211
Dallas, TX 75265

or Fax to: (214) 997–3471
Attn. Bob Steel

7.6.3 New Product Release Form for TSP50C14

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.

Division:______________________________Company : _______________________________
Project Name : ____
Purchase Order # : ____
Management Contact : _____________________________ Phone : (   ) ____________
Technical Contact   : _____________________________ Phone : (   ) ____________

Customer Part Number : ______________________________

D/A Output (Check one) :

_____ 2 pin push pull

_____ single pin single ended

Pulse-Width Modulated (Check one)

_____ PW1

_____ PW2

Internal RC Oscillator (Check one)

_____ 9.6 MHz

_____ 7.68 MHz

Package Type (Check one) :

_____ N (plastic)

_____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

This section is to be completed by TI and approved by the customer (customer approval in Section 3.)

TOP SIDE SYMBOLIZATION: (PACKAGED DEVICES ONLY)

```
 ________________________________________
| TI                                     |    YWW     = DATE CODE
|  LOGO          YWWLLLT                 |     LLL    = LOT TRACE CODE
>               {OPTIONAL 13 CHAR}       |      T     = ASSY SITE
|               {OPTIONAL 11 CHAR}       |    {FIRST LINE REQUIRED}
|                                        |
 ________________________________________
```

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 device 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 2B on all devices with the part number indicated in 2A.

By : ______________________________ Title : ______________

Date : ______________

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: Bob Steel
P.O. Box 655303, M/S 8211
Dallas, TX 75265

or Fax to: (214) 997–3471
Attn. Bob Steel

A Script Preparation and Speech Development Tools

Script preparation and speech development can be done either by the customer or TI. The following are major considerations during the process.

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 system 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 straightforward 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 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 esthetics 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 will analyze and synthesize properly. Certain consonants need to be emphasized and spoken more clearly than they are in normal speech. The TI SDS5000 development tool (see Figure A–2) provides immediate feedback for synthetic speech making, making the collecting 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 SDS5000 uses a TMS32020 digital signal processing chip to permit very rapid analysis. It consists of two boards that fit into an IBM PC®, software, and a documentation package. One of the boards contains the TMS32020 and related circuitry, and the other contains an analog-to-digital converter, a digital-to-analog converter, 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 but that requires low levels of phase distortion. For this reason, audio tape should not be used to collect speech. However, digital audio tape can be used.

A.1.3 LPC Editing

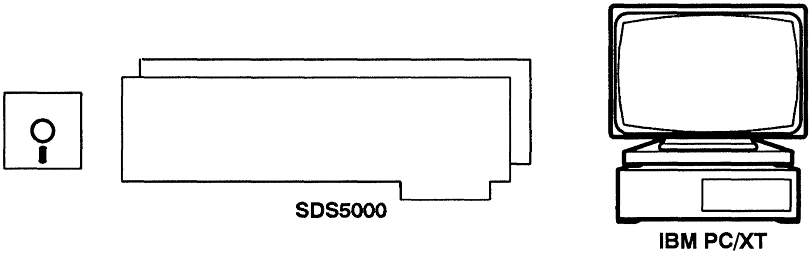
The speech often needs to be edited, both to define the boundaries of the words and to mask imperfections in the model, the analysis, and the speaker. Limited changes can be made to change inflection and emphasis, but the best quality is achieved by having the desired sound and inflection well-recorded. 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 SDS5000 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 SDS5000 supports quick prototyping.

LPC provides very-low-data-rate speech by virtue of its close modeling of the human vocal tract. Other sounds may or may not be modeled accurately by this model. The best way to find out is to try recording and analyzing the sound on the SDS5000.

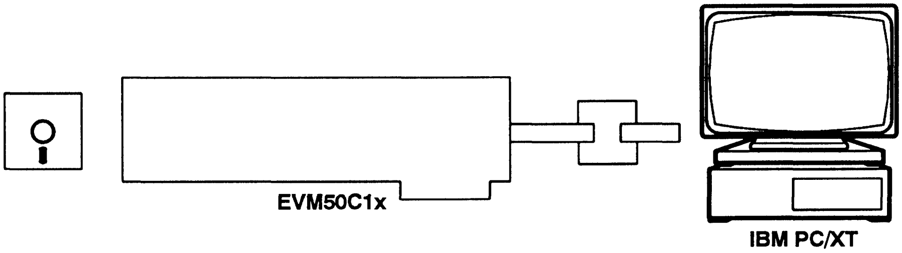
A.2 Speech Development Tools



- High-Speed Speech Analysis (2x Real Time)
- Graphical and Numerical Speech Editing
- Microphone and Line-Level Inputs
- Headphone Outputs
- Supports TSP5220, TSP50C4X Devices
- Requires IBM PC/XT, AT, or Compatible With CGA, EGA, or VGA Card
- Hard Disk and Tape Backup Strongly Suggested
- Uses TMS32020 Digital Signal Processor

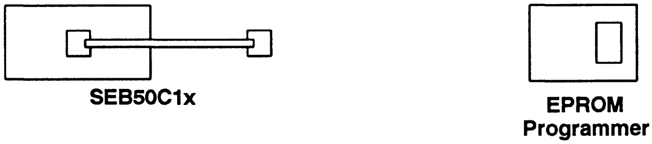
Figure A–1. SDS5000

IBM PC is a registered trademark of IBM Corporation.



- In-Circuit Emulation
- Hardware Breakpoints
- Single Step
- Examine/Modify Registers/Memory
- Includes Assembler
- Requires 1 Card Slot in IBM PC, PC/XT, PC/AT, and Compatibles

Figure A–2. EVM50C1X



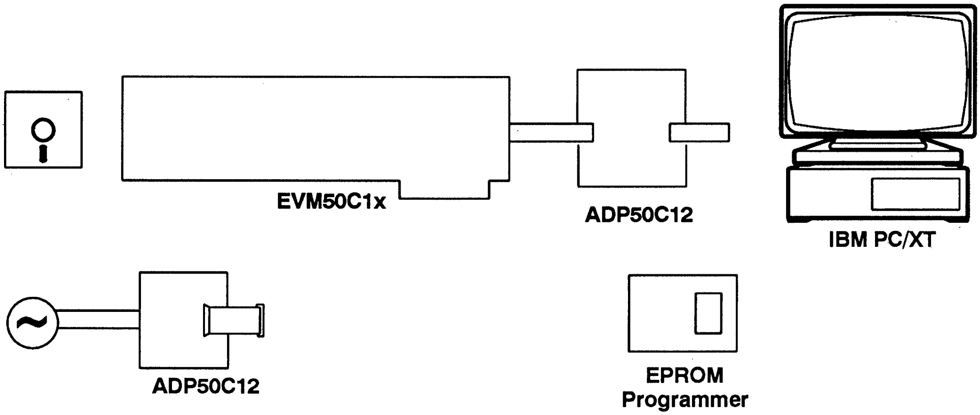
- In-Circuit Emulation
- Small Size, Low Power Consumption
- Ideal for Demonstration and Field Test
- Requires Industry-Standard EPROM (TMS27C256)

Figure A–3. SEB50C1X

SEB60Cxx
EPROM
Programmer

- In-Circuit Emulation of Up to Four TSP60CXXs
- Small Size, Low Power Consumption
- Ideal for Debugging, Demonstration, and Field Test
- Requires Industry-Standard EPROMs (TMS27C256)

Figure A–4. SEB60CXX



- Emulation of TSP50C12 for development purposes possible when using ADP50C12 and EVM50C1x
- Emulation of TSP50C12 for demonstration and field test purposes possible when using the ADP50C12 with an EPROM

Figure A–5. ADP50C12

B TSP50C1x Sample Synthesis Program

```
             OPTION  BUNLIST,DUNLIST,PAGEOF
*----------------------------------------------------*
*   TSP50C1x LPC SYNTHESIS PROGRAM                    *
*                                                     *
*   This is a sample speech synthesis program         *
*   which runs on the TSP50C1x family of speech       *
*   synthesis microprocessors.  It simply speaks the *
*   numbers from one to five.                         *
*                                                     *
*   This program uses the D6 Coding table format.     *
*                                                     *
*----------------------------------------------------*
*   COPYRIGHT 1989, 1992 TI - SPEECH PRODUCTS         *
*----------------------------------------------------*
*   RAM MAP                                           *
*+---------------------------------------------------*
*    +----+----+----+----+----+----+----+----+
*    | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
*    +----+----+----+----+----+----+----+----+
*    |    | EN | K12| K11| K10| K9 | K8 | K7 |
*    |    |    |    |    |    |    |    |    |
*    +----+----+----+----+----+----+----+----+
*    | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
*    +----+----+----+----+----+----+----+----+
*    | K6 | K5 | K4 | K3 | K2 | K1 | C1 | C2 |
*    |    |    |    |    |    |    |    |    |
*    +----+----+----+----+----+----+----+----+
*    | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
*    +----+----+----+----+----+----+----+----+
*    | EN | EN | PH      | PH      | K1      |
*    | V2 | V1 | V2      | V1      | V2      |
*    +----+----+----+----+----+----+----+----+
*    | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
*    +----+----+----+----+----+----+----+----+
*    | K1      | K2      | K2      | K3      |
*    | V1      | V2      | V1      | V2      |
*    +----+----+----+----+----+----+----+----+
*    | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
*    +----+----+----+----+----+----+----+----+
*    | K3      | K4      | K4      | K5 | K5 |
*    | V1      | V2      | V1      | V2 | V1 |
*    +----+----+----+----+----+----+----+----+
*    | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
*    +----+----+----+----+----+----+----+----+
*    | K6 | K6 | K7 | K7 | K8 | K8 | K9 | K9 |
*    | V2 | V1 | V2 | V1 | V2 | V1 | V2 | V1 |
*    +----+----+----+----+----+----+----+----+
*      | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
*      +----+----+----+----+----+----+----+----+
*      | K10| K10| K11| K11| K12| K12|TIMR|SCAL|
*      | V2 | V1 | V2 | V1 | V2 | V1 |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
*      +----+----+----+----+----+----+----+----+
*      |FLAG|FLAG|MODE|ADR |ADR |    |    |    |
*      |    | 1  |BUF |MSB |LSB |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
*      |    |    |    |    |    |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
*      |    |    |    |    |    |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
*      |    |    |    |    |    |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
*      |    |    |    |    |    |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
*      |    |    |    |    |    |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
*      |    |    |    |    |    |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
*      |    |    |    |    |    |    |    |    |
*      +----+----+----+----+----+----+----+----+
*      | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |
*      +----+----+----+----+----+----+----+----+
*      |    |    |    |    |    |    |    |    |
      *    |    |    |    |    |    |    |    |    |
      *    +----+----+----+----+----+----+----+----+
      *
      *----------------------------------------------------*
      *      ADDRESS LABELS FOR SYNTHESIS ROUTINE          *
      *----------------------------------------------------*
      ********************************************
      *      SYNTHESIZER RAM LOCATIONS
      ********************************************
      * NOTE - NEVER CHANGE LOCATIONS #01 TO #0F
      *
0001  EN          EQU     #01       -Energy working value
0002  K12         EQU     #02       -K12 Working Value
0003  K11         EQU     #03       -K11 Working Value
0004  K10         EQU     #04       -K10 Working Value
0005  K9          EQU     #05       -K9 Working Value
0006  K8          EQU     #06       -K8 Working Value
0007  K7          EQU     #07       -K7 Working Value
0008  K6          EQU     #08       -K6 Working Value
0009  K5          EQU     #09       -K5 Working Value
000A  K4          EQU     #0A       -K4 Working Value
000B  K3          EQU     #0B       -K3 Working Value
000C  K2          EQU     #0C       -K2 Working Value
000D  K1          EQU     #0D       -K1 Working Value
000E  C1          EQU     #0E       -C1 Parameter
000F  C2          EQU     #0F       -C2 Parameter
0010  ENV2        EQU     #10       -ENERGY New Value MSB
0011  ENV1        EQU     #11       -ENERGY Current Value MSB
0012  PHV2        EQU     #12       -PITCH New Value MSB
0014  PHV1        EQU     #14       -PITCH Current Value MSB
0016  K1V2        EQU     #16       -K1 New Value MSB
0018  K1V1        EQU     #18       -K1 Current Value MSB
001A  K2V2        EQU     #1A       -K2 New Value MSB
001C  K2V1        EQU     #1C       -K2 Current Value MSB
001E  K3V2        EQU     #1E       -K3 New Value MSB
0020  K3V1        EQU     #20       -K3 Current Value MSB
0022  K4V2        EQU     #22       -K4 New Value MSB
0024  K4V1        EQU     #24       -K4 Current Value MSB
0026  K5V2        EQU     #26       -K5 New Value
0027  K5V1        EQU     #27       -K5 Current Value
0028  K6V2        EQU     #28       -K6 New Value
0029  K6V1        EQU     #29       -K6 Current Value
002A  K7V2        EQU     #2A       -K7 New Value
002B  K7V1        EQU     #2B       -K7 Current Value
002C  K8V2        EQU     #2C       -K8 New Value
002D  K8V1        EQU     #2D       -K8 Current Value
002E  K9V2        EQU     #2E       -K9 New Value
002F  K9V1        EQU     #2F       -K9 Current Value
```

```
0030  K10V2       EQU     #30       –K10 New Value
0031  K10V1       EQU     #31       –K10 Current Value
0032  K11V2       EQU     #32       –K11 New Value
0033  K11V1       EQU     #33       –K11 Current Value
0034  K12V2       EQU     #34       –K12 New Value
0035  K12V1       EQU     #35       –K12 Current Value
      *
      *
      *   LPC status variable locations
      *
0036  TIMER       EQU     #36       –Stored Timer value for update
0037  SCALE       EQU     #37       –Interpolation factor INTP
0038  FLAGS       EQU     #38       –Flags used in LPC synthesis
0039  FLAG1       EQU     #39       –Flags used in LPC synthesis
003A  MODE_BUF    EQU     #3A       –Stored value of Mode register
003B  ADR_MSB     EQU     #3B       –MSB of address
003C  ADR_LSB     EQU     #3C       –LSB of address
      ************************************************************
      *   Constant Definitions
      ************************************************************
      *
      *   Bit Size of Speech parameters
      *
0004  EBITS       EQU     4         –Number of Energy Bits
0007  PBITS       EQU     7         –Number of Pitch Bits
0001  RBITS       EQU     1         –Number of Repeat Bits
0006  K1BITS      EQU     6         –Number of K1 Bits
0006  K2BITS      EQU     6         –Number of K2 Bits
0005  K3BITS      EQU     5         –Number of K3 Bits
0005  K4BITS      EQU     5         –Number of K4 Bits
0004  K5BITS      EQU     4         –Number of K5 Bits
0004  K6BITS      EQU     4         –Number of K6 Bits
0004  K7BITS      EQU     4         –Number of K7 Bits
0003  K8BITS      EQU     3         –Number of K8 Bits
0003  K9BITS      EQU     3         –Number of K9 Bits
0003  K10BITS     EQU     3         –Number of K10 Bits
0000  K11BITS     EQU     0         –Number of K11 Bits
0000  K12BITS     EQU     0         –Number of K12 Bits
      *
      *   Prescale Values
      *
      *   PSvalue = TRUNC(Samples * 2 * 30/256)
      *
      *   This comes from the fact that samples come every 30
      *   instruction cycles in LPC mode.  The factor of 2
      *   accounts for the cycle steal that happens in
      *   LPC mode.  When not in LPC mode, samples come
      *   every 60 instruction cycles, so it comes out the
      *   same.  The 256 divider is the full scale Timer
      *   register value.
      *
      *
00C8  SAMPLES     EQU     200       -Samples per frame
002E  PSVALUE     EQU     (SAMPLES*60/256) -Prescale Value
      *
      *   Device Constants
      *
0F61  C1_Value    EQU     #F61      -C1 Value
0B67  C2_Value    EQU     #B67      -C2 Value
007F  MAX_RAM     EQU     #7F       -Highest RAM location
      *
      *   Special Energy Values
      *
000F  ESTOP       EQU     15        -Stop code
0000  ESILENCE    EQU     0         -Silence Code
      *
      *   Special Pitch Value
      *
0000  PUnVoiced   EQU     0         -UnVoiced Frame Code
      *
      *
      *   End of sentence signal
      *
00FF  StopWord    EQU     #FF
      *
      *   FLAGS bit usage (and Set Masks)
      *
0001  STOPFLAG    EQU     #01       -Stop frame reached = 1
0002  R_FLAG      EQU     #02       -Repeat Frame = 1
0004  Update_Flg  EQU     #04       -Set high on update
0008  Sil_Flg1    EQU     #08       -New frame is silent = 1
0010  Unv_Flg1    EQU     #10       -New frame is unvoiced = 1
0020  Int_Inh     EQU     #20       -Inhibit interpolation = 1
0040  Sil_Flg2    EQU     #40       -Current frame silent = 1
0080  Unv_Flg2    EQU     #80       -Current frame unvoiced = 1
      *
      *   FLAG1 bit usage (and Set Masks)
      *
0001  Int_Off     EQU     #01       -Disable INTP routine = 1
      *
      *   MODE Register Bit Definitions
      *
0001  INT1        EQU     #01       -Enable Level 1 interrupt
0002  LPC         EQU     #02       -Enable LPC synthesis
0004  PCM         EQU     #04       -Enable PCM synthesis
0008  INT2        EQU     #08       -Enable Level 2 interrupt
```

```
0240      0010  EXTROM     EQU     #10        –Set external ROM mode
0241      0020  RAMROM     EQU     #20        –Enable GETs from RAM
0242      0040  MASTER     EQU     #40        –Master/Slave Toggle
0243      0080  UNV        EQU     #80        –Enable Unvoiced excitation
0244            ***********************************************************
0245            *   Start of program
0246            ***********************************************************
0247 0000                  AORG    #0000
0248 0000   69             TMAD    0
     0001   00
0249
0250            *-----------Initialize mode register-----------------------*
0251
0252 0002   2F             CLA
0253 0003   1D             TAMODE
0254
0255            *-----------Clear all ram to zero--------------------------*
0256
0257 0004   20             CLX                 –Start at bottom of RAM
0258 0005   13  RAM_LOOP   TAMIX               –Clear RAM, increment pointer
0259 0006   61             XGEC    MAX_RAM+1 –Finished all RAM?
     0007   80
0260 0008   40             BR      GO              yes, skip vector tables
     0009   24
0261 000A   40             BR      RAM_LOOP        no, loop back
     000B   05
0262            *
0263            ***********************************************************
0264            *   Interrupt vectors
0265            ***********************************************************
0266 0010                  AORG    #0010
0267 0010   A2             SBR     INT2_01    –Timer Underflow, PCM=0, LPC=1
0268 0011   A2             SBR     INT2_01    –Timer Underflow, PCM=0, LPC=1
0269 0012   A2             SBR     INT2_00    –Timer Underflow, PCM=0, LPC=0
0270 0013   A2             SBR     INT2_00    –Timer Underflow, PCM=0, LPC=0
0271 0014   A2             SBR     INT2_11    –Timer Underflow, PCM=1, LPC=1
0272 0015   A2             SBR     INT2_11    –Timer Underflow, PCM=1, LPC=1
0273 0016   A2             SBR     INT2_10    –Timer Underflow, PCM=1, LPC=0
0274 0017   A2             SBR     INT2_10    –Timer Underflow, PCM=1, LPC=0
0275 0018   A0             SBR     INT1_01    –PPC < 200 hex interrupt
0276 0019   A0             SBR     INT1_01    –PPC < 200 hex interrupt
0277 001A   A2             SBR     INT1_00    –Pin (B1) goes low interrupt
0278 001B   A2             SBR     INT1_00    –Pin (B1) goes low interrupt
0279 001C   A2             SBR     INT1_11    –10 kHz Clock interrupt
0280 001D   A2             SBR     INT1_11    –10 kHz Clock interrupt
0281 001E   A2             SBR     INT1_10    –20 kHz Clock interrupt
0282 001F   A2             SBR     INT1_10    –20 kHz Clock interrupt
0283            *
```

```
0020   40  INT1_01     BR      INTP      –PPC < 200 hex interrupt
0021   B4
           *
     0022  INT2_00
     0022  INT2_01
     0022  INT2_10
     0022  INT2_11
     0022  INT1_00
     0022  INT1_10
0022   2F  INT1_11     CLA
0023   3E              RETI
           ************************************************************
           *   Speak phrases
           ************************************************************
0024   6E  GO          TCA     0         –Speak 1st phrase
0025   00
0026   00              CALL    SPEAK
0027   31
           *
0028   6E              TCA     1         –Speak 2nd phrase
0029   01
002A   00              CALL    SPEAK
002B   31
           *
002C   6E              TCA     2         –Speak 3rd phrase
002D   02
002E   00              CALL    SPEAK
002F   31
           *
0030   3F              SETOFF            –Quit program
           ************************************************************
           *   Speak Utterance – Phrase number in A register
           ************************************************************
0031   3B  SPEAK       INTGR
0032   2E              SALA              –Double index to get offset
0033   75              ACAAC   SENTENCE  –Add base of table
0034   BF
0035   6D              LUAB              –get address MSB
0036   3A              IAC
0037   6B              LUAA              –Get address LSB
0038   12              XBA
0039   1B              SALA4             –Combine MSB and LSB
003A   1B              SALA4
003B   2C              ABAAC

003C   1A              TAB               –Save address
003D   6A              TAMD   ADR_LSB    –Save LSB of address
003E   3C
```

```
0323
0324 003F   68              AXCA    1          —Shift address right
     0040   01
0325 0041   15              SARA                  by 8 bits
0326
0327 0042   6A              TAMD    ADR_MSB    —Save MSB of address
     0043   3B
0328 0044   12              XBA
0329 0045   40              BR      SPEAK2
     0046   59
0330
0331 0047   69  SPEAK1      TMAD    ADR_LSB    —Fetch and combine
     0048   3C
0332 0049   1A              TAB                   address
0333 004A   69              TMAD    ADR_MSB
     004B   3B
0334 004C   1B              SALA4
0335 004D   1B              SALA4
0336 004E   2C              ABAAC
0337
0338 004F   3A              IAC                —Increment address
0339
0340 0050   1A              TAB                —Save new address
0341 0051   6A              TAMD    ADR_LSB    —Save LSB of address
     0052   3C
0342
0343 0053   68              AXCA    1          —Shift address right
     0054   01
0344 0055   15              SARA                  by 8 bits
0345
0346 0056   6A              TAMD    ADR_MSB    —Save MSB of address
     0057   3B
0347 0058   12              XBA
0348
0349 0059   6B  SPEAK2      LUAA               —Get word number
0350 005A   60              ANEC    StopWord   —End of phrase?
     005B   FF
0351 005C   40              BR      SPEAK3        no, continue
     005D   5F
0352 005E   3D              RETN                  yes, exit loop
0353
0354 005F   2E  SPEAK3      SALA               —Double index to get offset
0355 0060   75              ACAAC   SPEECH     —Add base of table
     0061   D5
0356 0062   6D              LUAB               —Get address MSB
0357 0063   3A              IAC
0358 0064   6B              LUAA               —Get address LSB
0359 0065   12              XBA
```

```
0066   1B              SALA4             –Combine LSB and MSB
0067   1B              SALA4
0068   2C              ABAAC

0069   6C              LUAPS             –Load Speech Address Register

006A   2F              CLA               –Kill K11 and K12 parameters
006B   6A              TAMD    K11
006C   03
006D   6A              TAMD    K12
006E   02

006F   6A              TAMD    FLAGS     –Init flags for speech
0070   38

0071   2F              CLA               –Load C2 parameter
0072   7B              ACAAC   C2_Value     (a device constant)
0073   67
0074   6A              TAMD    C2
0075   0F

0076   2F              CLA               –Load C1 parameter
0077   7F              ACAAC   C1_Value     (a device constant)
0078   61
0079   6A              TAMD    C1
007A   0E
           *
           * Now we give an initial value to the Pitch in case the
           * utterance starts with a silent frame.
           *
007B   70              ACAAC   #0C
007C   0C
007D   6A              TAMD    PHV1
007E   14
007F   6A              TAMD    PHV2
0080   12
           *
           * Now we preload the first two frames.
           *
0081   01              CALL    UPDATE    –Load first frame
0082   B5
0083   01              CALL    UPDATE    –Load 2nd frame
0084   B5
           *
           * Now we give some values to the Timer and Prescaler so
           * that we can do a valid interpolation on the first call to
           * INTP.  Then we do the first call to INTP to preload the
           * first valid interpolation.
           *
0085   6E              TCA     PSVALUE   -Initialize prescale
0086   2E
0087   19              TAPSC
0088   6E              TCA     #7F       -Pretend there was a previous
0089   7F
008A   6A              TAMD    TIMER        update
008B   36
008C   6E              TCA     #FF       -Set timer to max value to
008D   FF
008E   1E              TATM                 disable interpolation
008F   00              CALL    INTP      -Do first interpolation
0090   B4

           *
           * Now we enable the synthesizer for speech
           *
           * We do this in two stages so that we can reset the
           * interrupt pending latch without it being immediately
           * set again by the B1(low) interrupt.
           *
0091   62              TCX     MODE_BUF  -Turn on LPC synthesizer
0092   3A
0093   64              ORCM    LPC
0094   02
0095   11              TMA
0096   1D              TAMODE

0097   3E              RETI              -Reset interrupt pending latch

0098   64              ORCM    INT1      -Enable interrupt
0099   01
009A   11              TMA
009B   1D              TAMODE

           *
           * Now we loop until the utterance is complete. When the
           * utterance is finished, the routine UPDATE will execute a
           * RETN instruction which will exit this routine.  In the
           * meantime, this loop will poll the Timer register and
           * update the frame whenever it underflows.
           *
     009C  SPEAK_LP
009C   62              TCX     FLAGS
009D   38
009E   66              TSTCM   Update_Flg -Is Update already done?
009F   04
00A0   40              BR      SPEAK_LP     yes, loop
```

```
00A1   9C

00A2   62           TCX     TIMER     –Get old timer
00A3   36
00A4   11           TMA                  register value
00A5   1A           TAB                  into B register

00A6   17           TTMA              –Get new timer register
00A7   15           SARA                 value and scale it.

00A8   16           TAM               –Store new value
00A9   12           XBA               –Exchange new and old values
00AA   2D           SBAAN             –Subtract new from old
00AB   41           BR      UPDATE    –If underflowed, do an update
00AC   B5

00AD   11           TMA               –Get new timer value again.
00AE   60           ANEC    0         –Is it about to underflow?
00AF   00
00B0   40           BR      SPEAK_LP     no, loop again
00B1   9C
00B2   41           BR      UPDATE       yes, do update now
00B3   B5
           * * * * *
           * INTERPOLATION ROUTINE
           * * * * *
           * First we need to get the current value of the timer
           * register and store it away.  It will be divided by two
           * with the SARA instruction so that the most significant
           * bit is guaranteed to be zero so that it will always be
           * interpreted as a positive number during the
           * interpolation.
           * * * * *
00B4   3B  INTP         INTGR             –Ensure we are in integer mode
00B5   17               TTMA              –Get timer register contents
00B6   15               SARA                 shift to make positive
00B7   6A               TAMD    SCALE        and store it
00B8   37
           * * * * *
           * See if this routine is enabled.  If it is not, exit
           * the routine.
           * * * * *
00B9   62               TCX     FLAG1     –Point to flag
00BA   39
00BB   66               TSTCM   Int_Off   –If routine disabled...
00BC   01
00BD   41               BR      IRETI        ...branch to exit point
00BE   B3
```

```
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 00BF   62  TINTP       TCX     FLAGS     -Point to status flags
     00C0   38
0480 00C1   66              TSTCM   Int_Inh   -Is interpolation inhibited?
     00C2   20
0481 00C3   40              BR      NOINT        yes, inhibit interpolation
     00C4   C7
0482 00C5   40              BR      INTPCH       no, interpolate
     00C6   E4
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 00C7   6E  NOINT       TCA     #7F       -Set Scale factor to
     00C8   7F
0491 00C9   6A              TAMD    SCALE        highest value
     00CA   37
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 00CB   62              TCX     FLAGS
     00CC   38
0502 00CD   66              TSTCM   Unv_Flg2  -Is current frame unvoiced?
     00CE   80
0503 00CF   40              BR      Uv           yes, go to unvoiced branch
     00D0   D9
0504
0505 00D1   62              TCX     Mode_Buf  -Current frame is voiced
     00D2   3A
0506 00D3   66              TSTCM   UNV       -Has mode changed to unvoiced?
     00D4   80
0507 00D5   40              BR      ClrEN        yes, clear the energy
     00D6   DF
```

```
0508 00D7   40                BR      INTPCH       no, no action required
     00D8   E4
0509
0510 00D9   62  Uv            TCX     Mode_Buf  -New frame is unvoiced
     00DA   3A
0511 00DB   66                TSTCM   UNV       -Has voicing mode changed?
     00DC   80
0512 00DD   40                BR      INTPCH       no, no action required
     00DE   E4
0513
0514 00DF   2F  ClrEN         CLA               -Zero Energy during update
0515 00E0   6A                TAMD    EN
     00E1   01
0516 00E2   40                BR      INTPCH
     00E3   E4
0517
0518            *
0519            * Interpolate Pitch and write the result to the pitch
0520            * register
0521            *
0522 00E4   62  INTPCH        TCX     PHV2      -Combine new pitch and new
     00E5   12
0523 00E6   14                TMAIX                fractional pitch and
0524 00E7   1B                SALA4                leave in the B register
0525 00E8   28                AMAAC
0526 00E9   21                IXC
0527 00EA   1A                TAB
0528 00EB   14                TMAIX             -Combine current pitch and
0529 00EC   1B                SALA4                current fractional pitch
0530 00ED   28                AMAAC                and leave in A register
0531
0532 00EE   2D                SBAAN             -(Pcurrent - Pnew)
0533 00EF   62                TCX     SCALE
     00F0   37
0534 00F1   39                AXMA              -(Pcurrent-Pnew)*Timer
0535 00F2   2C                ABAAC             -Pnew+(Pcurrent-Pnew)*Timer
0536 00F3   2E                SALA              -Adjust for 2 byte excitation
0537 00F4   1C                TASYN             -Write to pitch register
0538            *
0539            * Interpolate K1 and store the result in the working K1
0540            * register
0541            *
0542 00F5   3C                EXTSG             -Allow negative K parameters
0543 00F6   62                TCX     K1V2      -Combine New K1 and New
     00F7   16
0544 00F8   14                TMAIX                fractional K1 and
0545 00F9   1B                SALA4                leave in the B register
0546 00FA   28                AMAAC
```

```
0547 00FB   21              IXC
0548 00FC   1A              TAB
0549
0550 00FD   14              TMAIX             -Combine current K1 and
0551 00FE   1B              SALA4                current fractional K1 and
0552 00FF   28              AMAAC                leave in the A register
0553
0554 0100   2D              SBAAN             -(K1current - K1new)
0555 0101   62              TCX     SCALE
     0102   37
0556 0103   39              AXMA              -(K1current - K1new) * Timer
0557 0104   2C              ABAAC             -K1new+(K1current-K1new)*Timer
0558 0105   6A              TAMD    K1        -Load interpolated K1 value
     0106   0D
0559            *
0560            * Interpolate K2 and store the result in the
0561            * working K2 register
0562            *
0563 0107   62              TCX     K2V2      -Combine New K2 and New
     0108   1A
0564 0109   14              TMAIX                fractional K2 and
0565 010A   1B              SALA4                leave in the B register
0566 010B   28              AMAAC
0567 010C   21              IXC
0568 010D   1A              TAB
0569
0570 010E   14              TMAIX             -Combine current K2 and
0571 010F   1B              SALA4                current fractional K2 and
0572 0110   28              AMAAC                leave in the A register
0573
0574 0111   2D              SBAAN             -(K2current - K2new)
0575 0112   62              TCX     SCALE
     0113   37
0576 0114   39              AXMA              -(K2current - K2new) * Timer
0577 0115   2C              ABAAC             -K2new+(K2current-K2new)*Timer
0578 0116   6A              TAMD    K2        -Load interpolated K2 value
     0117   0C
0579            *
0580            * Interpolate K3 and store the result in the working K3
0581            * register
0582            *
0583 0118   62              TCX     K3V2      -Combine New K3 and New
     0119   1E
0584 011A   14              TMAIX                fractional K3 and
0585 011B   1B              SALA4                leave in the B register
0586 011C   28              AMAAC
0587 011D   21              IXC
0588 011E   1A              TAB
```

```

011F   14              TMAIX            -Combine current K3 and
0120   1B              SALA4               current fractional K3 and
0121   28              AMAAC               leave in the A register

0122   2D              SBAAN            -(K3current - K3new)
0123   62              TCX     SCALE
0124   37
0125   39              AXMA             -(K3current - K3new) * Timer
0126   2C              ABAAC            -K3new+(K3current-K3new)*Timer
0127   6A              TAMD    K3       -Load interpolated K3 value
0128   0B
           *
           * Interpolate K4 and store the result in the working K4
           * register
           *
0129   62              TCX     K4V2     -Combine New K4 and New
012A   22
012B   14              TMAIX               fractional K4 and
012C   1B              SALA4               leave in the B register
012D   28              AMAAC
012E   21              IXC
012F   1A              TAB

0130   14              TMAIX            -Combine current K4 and
0131   1B              SALA4               current fractional K4 and
0132   28              AMAAC               leave in the A register

0133   2D              SBAAN            -(K4current - K4new)
0134   62              TCX     SCALE
0135   37
0136   39              AXMA             -(K4current - K4new) * Timer
0137   2C              ABAAC            -K4new+(K4current-K4new)*Timer
0138   6A              TAMD    K4       -Load interpolated K4 value
0139   0A
           *
           * Interpolate K5 and store the result in the working K5
           * register
           *
013A   62              TCX     K5V2     -Put New K5 (adjusted to
013B   26
013C   14              TMAIX               12 bits) in B register
013D   1B              SALA4
013E   1A              TAB
013F   14              TMAIX            -Put Current K5 (adjusted to
0140   1B              SALA4               12 bits) in A register

0141   2D              SBAAN            -(K5current - K5new)
```

```
0142   62               TCX     SCALE
0143   37
0144   39               AXMA               -(K5current - K5new) * Timer
0145   2C               ABAAC              -K5new+(K5current-K5new)*Timer
0146   6A               TAMD    K5         -Load interpolated K5 value
0147   09
             *
             * Interpolate K6 and store the result in the working K6
             * register
             *
0148   62               TCX     K6V2       -Put New K6 (adjusted to
0149   28
014A   14               TMAIX                 12 bits) in B register
014B   1B               SALA4
014C   1A               TAB
014D   14               TMAIX              -Put Current K6 (adjusted to
014E   1B               SALA4                 12 bits) in A register

014F   2D               SBAAN              -(K6current - K6new)
0150   62               TCX     SCALE
0151   37
0152   39               AXMA               -(K6current - K6new) * Timer
0153   2C               ABAAC              -K6new+(K6current-K6new)*Timer
0154   6A               TAMD    K6         -Load interpolated K6 value
0155   08
             *
             * Interpolate K7 and store the result in the working K7
             * register
             *
0156   62               TCX     K7V2       -Put New K7 (adjusted to
0157   2A
0158   14               TMAIX                 12 bits) in B register
0159   1B               SALA4
015A   1A               TAB
015B   14               TMAIX              -Put Current K7 (adjusted to
015C   1B               SALA4                 12 bits) in A register

015D   2D               SBAAN              -(K7current - K7new)
015E   62               TCX     SCALE
015F   37
0160   39               AXMA               -(K7current - K7new) * Timer
0161   2C               ABAAC              -K7new+(K7current-K7new)*Timer
0162   6A               TAMD    K7         -Load interpolated K7 value
0163   07
             *
             * Interpolate K8 and store the result in the working K8
             * register
             *
```

```
0164   62              TCX     K8V2      –Put New K8 (adjusted to
0165   2C
0166   14              TMAIX                12 bits) in B register
0167   1B              SALA4
0168   1A              TAB

0169   14              TMAIX             –Put Current K8 (adjusted to
016A   1B              SALA4                12 bits) in A register

016B   2D              SBAAN             –(K8current – K8new)
016C   62              TCX     SCALE
016D   37
016E   39              AXMA              –(K8current – K8new) * Timer
016F   2C              ABAAC             –K8new+(K8current–K8new)*Timer
0170   6A              TAMD    K8        –Load interpolated K8 value
0171   06
           *
           * Interpolate K9 and store the result in the working K9
           * register
           *
0172   62              TCX     K9V2      –Put New K9 (adjusted to
0173   2E
0174   14              TMAIX                12 bits) in B register
0175   1B              SALA4
0176   1A              TAB

0177   14              TMAIX             –Put Current K9 (adjusted to
0178   1B              SALA4                12 bits) in A register

0179   2D              SBAAN             –(K9current – K9new)
017A   62              TCX     SCALE
017B   37
017C   39              AXMA              –(K9current – K9new) * Timer
017D   2C              ABAAC             –K9new+(K9current–K9new)*Timer
017E   6A              TAMD    K9        –Load interpolated K9 value
017F   05
           *
           * Interpolate K10 and store the result in the working K10
           * register
           *
0180   62              TCX     K10V2     –Put New K10 (adjusted to
0181   30
0182   14              TMAIX                12 bits) in B register
0183   1B              SALA4
0184   1A              TAB

0185   14              TMAIX             –Put Current K10 (adjusted to
0186   1B              SALA4                12 bits) in A register
```

```

0187   2D              SBAAN            -(K10current - K10new)
0188   62              TCX     SCALE
0189   37
018A   39              AXMA         -(K10current - K10new) * Timer
018B   2C              ABAAC        -K10new+(K10current-K10new)*Timer
018C   6A              TAMD    K10  -Load interpolated K10 value
018D   04

           *
           * K11 and K12 are not needed for LPC 10, so they have been
           * commented out.
           *
           * Interpolate K11 and store the result in the working K11
           * register
           *
           *           TCX     K11V2    -Put New K11 (adjusted to
           *           TMAIX               12 bits) in B register
           *           SALA4
           *           TAB

           *           TMAIX            -Put Current K11 (adjusted to
           *           SALA4               12 bits) in A register

           *           SBAAN            -(K11current - K11new)
           *           TCX SCALE
           *           AXMA         -(K11current - K11new) * Timer
           *           ABAAC        -K11new+(K11current-K11new)*Timer
           *           TAMD K11     -Load interpolated K11 value
           *
           * Interpolate K12 and store the result in the working
           * K12 register
           *
           *           TCX K12V2        -Put New K12 (adjusted to
           *           TMAIX               12 bits) in B register
           *           SALA4
           *           TAB

           *           TMAIX            -Put Current K12 (adjusted to
           *           SALA4               12 bits) in A register

           *           SBAAN            -(K12current - K12new)
           *           TCX SCALE
           *           AXMA         -(K12current - K12new) * Timer
           *           ABAAC        -K12new+(K12current-K12new)*Timer
           *           TAMD K12     -Load interpolated K12 value
           *
           *
```

```
0758             * Interpolate Energy
0759             *
0760             *
0761 018E   3B             INTGR               -Back to integer mode for energy
0762 018F   62             TCX     ENV2        -Combine new energy and
     0190   10
0763 0191   14             TMAIX                  fractional energy and
0764 0192   1B             SALA4                  leave in the B register
0765 0193   1A             TAB
0766 0194   14             TMAIX               -Combine current energy and
0767 0195   1B             SALA4                  current fractional energy
0768 0196   2D             SBAAN               -(Ecurrent - Enew)
0769 0197   62             TCX     SCALE
     0198   37
0770 0199   39             AXMA                -(Ecurrent - Enew) * Timer
0771 019A   2C             ABAAC               -Enew+(Ecurrent-Enew)*Timer
0772 019B   6A             XBA                 -Save energy
0773             *
0774             * Set voiced/unvoiced mode according to current frame type.
0775             * This is done in a two step fashion:  first the value in
0776             * the MODE_BUF register is adjusted with an AND or OR
0777             * operation, then the result is written to the synthesizer
0778             * with a TAMODE operation.  We do it this way to keep a copy
0779             * of the current status of the synthesizer mode at all time.
0780             *
0781 019C   62  STMODE     TCX     FLAGS
     019E   38
0782 019E   65             ANDCM   ~Update_Flg -Signal that interp done
     019F   FB
0783 01A0   66             TSTCM   Unv_Flg2   -Is current frame unvoiced?
     01A1   80
0784 01A2   41             BR      SETUV      -yes, set mode to unvoiced
     01A3   AA
0785 01A4   62             TCX     MODE_BUF      no, ...
     01A5   3A
0786 01A6   65             ANDCM   ~UNV          ...set mode to voiced
     01A7   7F
0787 01A8   41             BR      WRITEMODE
     01A9   AE
0788
0789 01AA   62  SETUV      TCX     MODE_BUF   -Current frame is unvoiced, so
     01AB   3A
0790 01AC   64             ORCM    UNV        -set mode to unvoiced.
     01AD   80
0791
0792 01B1   11  WRITEMODE  TMA                -Write mode information
0793 01AF   1D             TAMODE              to mode register
0794
```

```
0795 01B0   12             XBA               -Write energy
0796 01B1   62             TAMD     EN        to filter
     01B2   01
0797
0798 01B3   3E  IRETI       RETI              -Return from interrupt
0799 01B4   3D              RETN              -Return from first call
0800            * Update the parameters for a new frame
0801            *
0802            * First we inhibit the operation of the interpolation
0803            * routine.
0804            *
0805 01B5   62  UPDATE      TCX     MODE_BUF
     01B6   3A
0806 01B7   65              ANDCM   ~INT1
     01B8   FE
0807 01B9   11              TMA
0808 01BA   1D              TAMODE
0809            *
0810            * To prevent double updates, if the stored value of the
0811            * timer register is zero, then we need to change it to #7F.
0812            * If we do not do this, then the polling routine will
0813            * discover an underflow and call Update a second time.
0814            *
0815 01BB   62              TCX     TIMER    -Get stored value
     01BC   36
0816 01BD   11              TMA                 of Timer into A
0817
0818 01BE   60              ANEC    0        -Is it zero?
     01BF   00
0819 01C0   41              BR      UPDT00      no, do nothing
     01C1   C5
0820 01C2   6E              TCA     #7F         yes, replace value
     01C3   7F
0821 01C4   16              TAM
0822            *
0823            * First we need to test to see if a stop frame was
0824            * encountered on the last pass through the routine.   If the
0825            * previous frame was a stop frame, we need to turn off the
0826            * synthesizer and stop speaking.
0827            *
0828 01C5   62  UPDT00      TCX     FLAGS
     01C6   38
0829 01C7   66              TSTCM   STOPFLAG -Was stop frame encountered
     01C8   01
0830 01C9   42              BR      STOP        yes, stop speaking
     01CA   EF
0831            *
0832            * Transfer the state of the previous frame to the Unvoiced
```

```
           * flag (Current).
           *
01CB  66               TSTCM   Unv_Flg1  -Was previous frame unvoiced?
01CC  10
01CD  41               BR      SUNVL        yes, current frame=unvoiced
01CE  D3
01CF  65               ANDCM   ~Unv_Flg2    no, current frame=voiced
01D0  7F
01D1  41               BR      TSIL         and continue
01D2  D5

01D3  64   SUNVL       ORCM    Unv_Flg2  -Set current frame unvoiced.
01D4  80
           *
           * Transfer the state of the previous frame to the
           * Silence flag (Current).
           *
01D5  66   TSIL        TSTCM   Sil_Flg1  -Was previous frame silent?
01D6  08
01D7  41               BR      SSIL         yes, current frame silent
01D8  DD
01D9  65               ANDCM   ~Sil_Flg2    no, current frame not sil.
01DA  BF
01DB  41               BR      ZROFLG       and continue
01DC  DF

01DD  64   SSIL        ORCM    Sil_Flg2  -Set current frame silent
01DE  40
           *
           * Reset the Repeat Flag, new Silence Flag, new Unvoiced
           * Flag, and Interpolation Inhibit flag so that new
           * values can be loaded in this routine.
           *
01DF  62   ZROFLG      TCX     FLAGS
01E0  38
01E1  65               ANDCM   #C5
01E2  C5
           *
           * Transfer the new frame parameters into the
           * storage location used for the current frame parameters.
           *
01E3  62               TCX     ENV2      -Transfer new frame energy
01E4  10
01E5  14               TMAIX                from new frame location
01E6  13               TAMIX                to current frame location
           *------PITCH------
01E7  14               TMAIX             -Transfer new frame pitch
01E8  6A               TAMD    PHV1         to current frame location
```

```
     01E9   14
0868
0869 01EA   14                TMAIX             –Transfer new fractional pitch
0870 01EB   21                IXC                 to current frame location
0871 01EC   13                TAMIX
0872              *———K1———
0873 01ED   14                TMAIX             –Transfer new frame K1 param.
0874 01EE   6A                TAMD     K1V1       to current frame location
     01EF   18
0875 01F0   14                TMAIX             –Transfer new fractional K1
0876 01F1   21                IXC                 to current frame location
0877 01F2   13                TAMIX
0878              *———K2———
0879 01F3   14                TMAIX             –Transfer new frame K2 param.
0880 01F4   6A                TAMD     K2V1       to current frame location
     01F5   1C
0881 01F6   14                TMAIX             –Transfer new fractional K2
0882 01F7   21                IXC                 to current frame location
0883 01F8   13                TAMIX
0884              *———K3———
0885 01F9   14                TMAIX             –Transfer new frame K3 param.
0886 01FA   6A                TAMD     K3V1       to current frame location
     01FB   20
0887 01FC   14                TMAIX             –Transfer new fractional K3
0888 01FD   21                IXC                 to current frame location
0889 01FE   13                TAMIX
0890              *———K4———
0891 01FF   14                TMAIX             –Transfer new frame K4 param.
0892 0200   6A                TAMD     K4V1       to current frame location
     0201   24
0893 0202   14                TMAIX             –Transfer new fractional K4
0894 0203   21                IXC                 to current frame location
0895 0204   13                TAMIX
0896              *———K5———
0897 0205   14                TMAIX             –Transfer new frame K5 param.
0898 0206   13                TAMIX               to current frame location
0899              *———K6———
0900 0207   14                TMAIX             –Transfer new frame K6 param.
0901 0208   13                TAMIX               to current frame location
0902              *———K7———
0903 0209   14                TMAIX             –Transfer new frame K7 param.
0904 020A   13                TAMIX               to current frame location
0905              *———K8———
0906 020B   14                TMAIX             –Transfer new frame K8 param.
0907 020C   13                TAMIX               to current frame location
0908              *———K9———
0909 020D   14                TMAIX             –Transfer new frame K9 param.
0910 020E   13                TAMIX               to current frame location
```

```
           *———K10———
020F   14              TMAIX             –Transfer new frame K10 param.
0210   13              TAMIX                to current frame location
           *
           * K11 and K12 are not used in LPC 10 synthesis.  The code
           * has been commented out.
           *
           *———K11———
           *           TMAIX             –Transfer new frame K11 param.
           *           TAMIX                 to current frame location
           *———K12———
           *           TMAIX             –Transfer new frame K12 param.
           *           TAMIX                 to current frame location
           *———
           *
           * We have now discarded the "current" values by replacing
           * them with the "new" values.  We now need to read in
           * another frame of speech data and use them as the
           * new "new" values.
           * * * * *
           *——— ENERGY ———
0211   2F              CLA
0212   62              TCX     FLAGS
0213   38
0214   33              GET     EBITS     –Get coded energy
0215   60              ANEC    ESILENCE  –Is it a silent frame?
0216   00
0217   42              BR      UPDT0        No, continue
0218   1D
0219   64              ORCM    Sil_Flg1+Int_Inh   Yes, set silence flag
021A   28
021B   42              BR      ZeroKs       and zero K params
021C   CD
           *
021D   60  UPDT0       ANEC    ESTOP     –Is it a stop frame?
021E   0F
021F   42              BR      UPDT1        no, continue
0220   25
0221   64              ORCM    STOPFLAG+Sil_Flg1+Int_Inh yes, set flags
0222   29
0223   42              BR      ZeroKs       and zero Ks
0224   CD
           *
0225   73  UPDT1       ACAAC   TBLEN     –Add table offset to energy
0226   27
0227   6B              LUAA              –Get decoded energy
0228   6A              TAMD    ENV2      –Store the Energy in RAM
0229   10
```

```
           *
           * If this is a silent frame, we are done with the update If
           * the previous frame was silent, the new frame should be
           * spoken immediately with no ramp up due to interpolation
           *
022A   62              TCX     FLAGS
022B   38
022C   66              TSTCM   Sil_Flg1  –Is this a silent frame?
022D   08
022E   43              BR      RTN          yes, exit
022F   0C
           *
           * A repeat frame will use the K parameter from the previous
           * frame.  If it is a repeat frame, we need to set a flag.
           *
0230   30  UPDT2       GET     RBITS     –Get the Repeat bit
0231   67              TSTCA   #01       –Is this a repeat frame?
0232   01
0233   42              BR      SFLG1        yes, set repeat flag
0234   37
0235   42              BR      UPDT3
0236   39

0237   64  SFLG1       ORCM    R_FLAG    –Set repeat flag
0238   02

           *——— PITCH ———

0239   2F  UPDT3       CLA
023A   33              GET     4         –Get coded pitch
023B   32              GET     3         –Get coded pitch
023C   60              ANEC    PUnVoiced –Is the frame unvoiced?
023D   00
023E   C1              SBR     UPDT3A       no, continue
023F   64              ORCM    Unv_Flg1     yes, set unvoiced flag
0240   10

0241   2E  UPDT3A      SALA              –Double coded pitch and
0242   73              ACAAC   TBLPH        add table offset to point
0243   37

0244   6D              LUAB              –Get decoded pitch
0245   3A              IAC
0246   6B              LUAA              –Get decoded fractional pitch

0247   62              TCX     PHV2      –Store the pitch and
0248   12
0249   2A              TBM                  fractional pitch in RAM
```

```
024A   21             IXC
024B   16             TAM
           *
           * If the voicing has changed with the new frame, then we
           * need to change the voicing in the mode register.
           *
024C   62             TCX     FLAGS
024D   38
024E   66             TSTCM   Unv_Flg1  –Is the new frame unvoiced?
024F   10
0250   D3             SBR     UPDT3B       yes, continue
0251   42             BR      VOICE        no, go to voiced code
0252   5D
           *
           * The following code is reached if the new frame is
           * unvoiced. We inspect the flags to see if the previous
           * frame was either silent or voiced. If either condition
           * applies, then we branch to code which inhibits
           * interpolation.
           *
0253   66  UPDT3B      TSTCM   Sil_Flg2  –Was the last frame silent?
0254   40
0255   42             BR      UPDT5        yes, inhibit interpolation
0256   63

0257   66             TSTCM   Unv_Flg2  –Was the last frame unvoiced
0258   80
0259   42             BR      UPDT4        yes, don't change anything
025A   65
025B   42             BR      UPDT5        no, inhibit interpolation
025C   63
           *
           * The following code is reached if the new frame is
           * voiced. We inspect the flags to see if the previous
           * frame was also voiced. If it was not, we need to inhibit
           * interpolation.
           *
025D   66  VOICE       TSTCM   Unv_Flg2  –Was the last frame voiced?
025E   80
025F   42             BR      UPDT5        no, disable interpolation
0260   63
0261   42             BR      UPDT4        yes, continue
0262   65

0263   64  UPDT5       ORCM    Int_Inh   –Inhibit interpolation
0264   20
           *
           * Now we test the repeat flag. If the new frame is a repeat
```

```
            * frame, then the current values are used for the K factors,
            * so new values do not need to be loaded and we can exit the
            * routine now.
            *
0265   66   UPDT4       TSTCM   R_FLAG     –Is repeat flag set?
0266   02
0267   43               BR      RTN          yes, exit routine
0268   0C
            *
            * Now we need to load the "new" K factors (K1 through K10).
            * The first four K factors are 12 bit values which will be
            * stored in two bytes. The most significant 8 bits in the
            * first byte, and the least significant 4 bits (called the
            * fractional value) in the second byte. For K5 through K12,
            * the fractional part is assumed to be zero. K11 and K12 are
            * not used in LPC10 synthesis, and the code loading them is
            * commented out. A coded factor is read into the A
            * register. It is then converted to a pointer to a table
            * element which contains the uncoded factor. Since the K1
            * through K4 table elements consist of two bytes, the
            * conversion consists of doubling the coded factor and adding
            * the result to the start of the table.  Since the K5 through
            * K10 table elements consist of one byte, the coded factor is
            * added directly to the start of the table.  Once the pointer
            * has been set up, the uncoded factor is fetched and stored
            * into RAM.
            *
            *———K1———
0269   2F               CLA
026A   33               GET     4          –Get coded K1
026B   31               GET     2          –Get coded K1
026C   2E               SALA               –Convert it to a
026D   74               ACAAC   TBLK1        pointer to table element
026E   37
026F   6D               LUAB               –Fetch MSB of uncoded K1
0270   3A               IAC
0271   6B               LUAA               –Fetch fractional K1
0272   62               TCX     K1V2
0273   16
0274   2A               TBM                –Store uncoded K1
0275   21               IXC
0276   16               TAM                –Store fractional K1
            *———K2———
0277   2F               CLA
0278   33               GET     4          –Get coded K2
0279   31               GET     2          –Get coded K2

027A   2E               SALA               –Convert it to a
```

```
027B   74                    ACAAC   TBLK2           pointer to table element
027C   B7

027D   6D                    LUAB                 -Fetch MSB of uncoded K2
027E   3A                    IAC
027F   6B                    LUAA                 -Fetch fractional K2
0280   62                    TCX     K2V2
0281   1A
0282   2A                    TBM                  -Store uncoded K2
0283   21                    IXC
0284   16                    TAM                  -Store fractional K2
               *------K3------
0285   2F                    CLA
0286   33                    GET     4            -Get Index into K3 table
0287   30                    GET     1            -Get Index into K3 table
0288   75                    ACAAC   TBLK3           and add offset of table
0289   37

028A   6B                    LUAA                 -Get uncoded K3
028B   6A                    TAMD    K3V2         -and store it in RAM
028C   1E
028D   2F                    CLA
028E   6A                    TAMD    K3V2+1
028F   1F
               *------K4------
0290   2F                    CLA
0291   33                    GET     4            -Get Index into K4 table
0292   30                    GET     1            -Get Index into K4 table
0293   75                    ACAAC   TBLK4           and add offset of table
0294   57
0295   6B                    LUAA                 -Get uncoded K4
0296   6A                    TAMD    K4V2         -and store it in RAM
0297   22
0298   2F                    CLA
0299   6A                    TAMD    K4V2+1
029A   23
               *
               * If this is an unvoiced frame, we only use four K factors,
               * so we load zeroes to the rest of the K factors.  If this
               * is a voiced frame, load the rest of the uncoded factors.
               *
029B   62                    TCX     FLAGS
029C   38
029D   66                    TSTCM   Unv_Flg1     -Is this an unvoiced frame?
029E   10
029F   42                    BR      UNVC            Yes, zero rest of factors
02A0   E0
               *
```

```
                * The following code is executed if the new frame is
                * voiced.  Since we assume that the fractional parameter is
                * zero for the remaining K factors, the table elements are
                * only one byte long. The conversion to
                * a table pointer consists of adding the coded factor to the
                * start of the table.
                *
                *------K5------
02A1     2F                 CLA
02A2     33                 GET      K5BITS    -Get Index into K5 table
02A3     75                 ACAAC    TBLK5        and add offset of table
02A4     77

02A5     6B                 LUAA               -Get uncoded K5
02A6     6A                 TAMD     K5V2         and store it in RAM
02A7     26
                *------K6------
02A8     2F                 CLA
02A9     33                 GET      K6BITS    -Get Index into K6 table
02AA     75                 ACAAC    TBLK6        and add offset of table
02AB     87
02AC     6B                 LUAA               -Get uncoded K6
02AD     6A                 TAMD     K6V2         and store it in RAM
02AE     28
                *------K7------
02AF     2F                 CLA
02B0     33                 GET      K7BITS    -Get Index into K7 table
02B1     75                 ACAAC    TBLK7        and add offset of table
02B2     97
02B3     6B                 LUAA               -Get uncoded K7
02B4     6A                 TAMD     K7V2         and store it in RAM
02B5     2A
                *------K8------
02B6     2F                 CLA
02B7     32                 GET      K8BITS    -Get Index into K8 table
02B8     75                 ACAAC    TBLK8        and add offset of table
02B9     A7
02BA     6B                 LUAA               -Get uncoded K8
02BB     6A                 TAMD     K8V2         and store it in RAM
02BC     2C
                *------K9------
02BD     2F                 CLA
02BE     32                 GET      K9BITS    -Get Index into K9 table
02BF     75                 ACAAC    TBLK9        and add offset of table
02C0     AF
02C1     6B                 LUAA               -Get uncoded K9
02C2     6A                 TAMD     K9V2         and store it in RAM
02C3     2E
```

```
                *------K10------
02C4    2F                CLA
02C5    32                GET      K10BITS   –Get Index into K10 table
02C6    75                ACAAC    TBLK10       and add offset of table
02C7    B7
02C8    6B                LUAA               –Get uncoded K10
02C9    6A                TAMD     K10V2        and store it in RAM
02CA    30
                *
                * Since K11 and K12 are not used in LPC 10, the K11 and K12
                * code is commented out.
                *
                *------K11------
                *         CLA
                *         GET      K11BITS   –Get Index into K11 table
                *         ACAAC    TBLK11       and add offset of table
                *         LUAA               –Get uncoded K11
                *         TAMD     K11V2        and store it in RAM
                *------K12------
                *         CLA
                *         GET      K12BITS   –Get Index into K12 table
                *         ACAAC    TBLK12       and add offset of table
                *         LUAA               –Get uncoded K12
                *         TAMD     K12V2        and store it in RAM
                *------
02CB    43                BR       RTN
02CC    0C
                *
                * The following code is executed if the K parameters need to
                * be zeroed out.  If  the new frame is a stop frame or a
                * silent frame, we zero out all K parameters and set the
                * energy to zero.  If the new frame is an unvoiced frame,
                * then we need to zero out the unused upper K parameters.
                *
                *
02CD    2F      ZeroKs    CLA
02CE    6A                TAMD     ENV2      –Kill Energy
02CF    10
02D0    6A                TAMD     K1V2      –Kill K1
02D1    16
02D2    6A                TAMD     K1V2+1
02D3    17
02D4    6A                TAMD     K2V2      –Kill K2
02D5    1A
02D6    6A                TAMD     K2V2+1
02D7    1B
02D8    6A                TAMD     K3V2      –Kill K3
02D9    1E
```

```
02DA   6A            TAMD    K3V2+1
02DB   1F
02DC   6A            TAMD    K4V2      –Kill K4
02DD   22
02DE   6A            TAMD    K4V2+1
02DF   23
02E0   2F  UNVC      CLA
02E1   6A            TAMD    K5V2      –Kill K5
02E2   26
02E3   6A            TAMD    K6V2      –Kill K6
02E4   28
02E5   6A            TAMD    K7V2      –Kill K7
02E6   2A
02E7   6A            TAMD    K8V2      –Kill K8
02E8   2C
02E9   6A            TAMD    K9V2      –Kill K9
02EA   2E
02EB   6A            TAMD    K10V2     –Kill K10
02EC   30
           *         TAMD    K11V2     –Kill K11
           *         TAMD    K12V2     –Kill K12
02ED   43            BR      RTN
02EE   0C
           *
           * STOP AND RETURN
           *
           * The following code has two entry points. STOP is reached
           * if the stop flag has been set. It turns off
           * synthesis and returns to the program.  RTN is the general
           * exit point for the UPDATE routine, it sets the Update flag
           * and leaves the routine.
           *
02EF   62  STOP      TCX     MODE_BUF
02F0   3A
02F1   65            ANDCM   ~LPC      –Turn off synthesis
02F2   FD
02F3   65            ANDCM   ~INT1     –Disable interrupt
02F4   FE
02F5   65            ANDCM   ~UNV      –Back to voiced for next word
02F6   7F
02F7   64            ORCM    PCM       –Enable PCM mode
02F8   04
02F9   11            TMA
02FA   1D            TAMODE            –Set mode per above setting
02FB   2F            CLA
02FC   1C            TASYN             –Write a zero to the DAC
02FD   6E            TCA     #FA
02FE   FA
```

```
02FF   3A  BACK        IAC                -Wait for minimum of 30
0300   43              BR      out           instruction cycles
0301   04
0302   42              BR      back
0303   FF
0304   62  OUT         TCX     MODE_BUF   -Disable PCM
0305   3A
0306   65              ANDCM   ~PCM
0307   FB
0308   11              TMA
0309   1D              TAMODE             -Set mode per above setting
030A   40              BR      SPEAK1     -Go back for next word
030B   47

030C   62  RTN         TCX     FLAGS      -Set a flag indicating that
030D   38
030E   64              ORCM    Update_Flg    the parameters are updated
030F   04

0310   62              TCX     MODE_BUF   -Get mode
0311   3A
0312   66              TSTCM   LPC        -Are we speaking yet?
0313   02
0314   43              BR      RTN1          yes, reenable interrupt
0315   17
0316   3D              RETN                  no, return for more data

0317   62  RTN1        TCX     FLAG1      -Inhibit any pending
0318   39
0319   64              ORCM    Int_Off       interpolation interrupt
031A   01

031B   62              TCX     MODE_BUF   -Reenable the interrupt
031C   3A
031D   64              ORCM    INT1
031E   01
031F   11              TMA
0320   1D              TAMODE

0321   62              TCX     FLAG1      -Reenable execution
0322   39
0323   65              ANDCM   ~Int_Off     of the interpolation routine
0324   FE
0325   40              BR      SPEAK_LP   -Go back to loop
0326   9C
           *
           * D6 SPEECH DECODING TABLES.
           *
           * Energy decoding table
           *
0327   00  TBLEN       BYTE    #00,#01,#02,#03,#04,#05,#07,#0B
032F   11              BYTE    #11,#1A,#29,#3F,#55,#70,#7F,#00

           *
           * Pitch period decoding table
           *
0337   0C  TBLPH       BYTE    #0C,#00
0339   10              BYTE    #10,#00
033B   10              BYTE    #10,#04
033D   10              BYTE    #10,#08
033F   11              BYTE    #11,#00
0341   11              BYTE    #11,#04
0343   11              BYTE    #11,#08
0345   11              BYTE    #11,#0C
0347   12              BYTE    #12,#04
0349   12              BYTE    #12,#08
034B   12              BYTE    #12,#0C
034D   13              BYTE    #13,#04
034F   13              BYTE    #13,#08
0351   14              BYTE    #14,#00
0353   14              BYTE    #14,#04
0355   14              BYTE    #14,#0C
0357   15              BYTE    #15,#00
0359   15              BYTE    #15,#08
035B   15              BYTE    #15,#0C
035D   16              BYTE    #16,#04
035F   16              BYTE    #16,#0C
0361   17              BYTE    #17,#00
0363   17              BYTE    #17,#08
0365   18              BYTE    #18,#00
0367   18              BYTE    #18,#04
0369   18              BYTE    #18,#0C
036B   19              BYTE    #19,#04
036D   19              BYTE    #19,#0C
036F   1A              BYTE    #1A,#04
0371   1A              BYTE    #1A,#0C
0373   1B              BYTE    #1B,#04
0375   1B              BYTE    #1B,#0C
0377   1C              BYTE    #1C,#04
0379   1C              BYTE    #1C,#0C
037B   1D              BYTE    #1D,#04
037D   1D              BYTE    #1D,#0C
037F   1E              BYTE    #1E,#04
0381   1F              BYTE    #1F,#00
0383   1F              BYTE    #1F,#08
0385   20              BYTE    #20,#00
```

```
1290 0387   20                 BYTE     #20,#0C
1291 0389   21                 BYTE     #21,#04
1292 038B   21                 BYTE     #21,#0C
1293 038D   22                 BYTE     #22,#08
1294 038F   23                 BYTE     #23,#00
1295 0391   23                 BYTE     #23,#0C
1296 0393   24                 BYTE     #24,#08
1297 0395   25                 BYTE     #25,#00
1298 0397   25                 BYTE     #25,#0C
1299 0399   26                 BYTE     #26,#08
1300 039B   27                 BYTE     #27,#04
1301 039D   28                 BYTE     #28,#00
1302 039F   28                 BYTE     #28,#0C
1303 03A1   29                 BYTE     #29,#08
1304 03A3   2A                 BYTE     #2A,#04
1305 03A5   2B                 BYTE     #2B,#00
1306 03A7   2B                 BYTE     #2B,#0C
1307 03A9   2C                 BYTE     #2C,#08
1308 03AB   2D                 BYTE     #2D,#04
1309 03AD   2E                 BYTE     #2E,#04
1310 03AF   2F                 BYTE     #2F,#00
1311 03B1   30                 BYTE     #30,#00
1312 03B3   30                 BYTE     #30,#0C
1313 03B5   31                 BYTE     #31,#0C
1314 03B7   32                 BYTE     #32,#08
1315 03B9   33                 BYTE     #33,#08
1316 03BB   34                 BYTE     #34,#08
1317 03BD   35                 BYTE     #35,#08
1318 03BF   36                 BYTE     #36,#08
1319 03C1   37                 BYTE     #37,#08
1320 03C3   38                 BYTE     #38,#08
1321 03C5   39                 BYTE     #39,#08
1322 03C7   3A                 BYTE     #3A,#08
1323 03C9   3B                 BYTE     #3B,#0C
1324 03CB   3C                 BYTE     #3C,#0C
1325 03CD   3D                 BYTE     #3D,#0C
1326 03CF   3F                 BYTE     #3F,#00
1327 03D1   40                 BYTE     #40,#04
1328 03D3   41                 BYTE     #41,#04
1329 03D5   42                 BYTE     #42,#08
1330 03D7   43                 BYTE     #43,#0C
1331 03D9   45                 BYTE     #45,#00
1332 03DB   46                 BYTE     #46,#04
1333 03DD   47                 BYTE     #47,#08
1334 03DF   49                 BYTE     #49,#00
1335 03E1   4A                 BYTE     #4A,#04
1336 03E3   4B                 BYTE     #4B,#0C
1337 03E5   4D                 BYTE     #4D,#00
```

```
03E7   4E                BYTE    #4E,#08
03E9   50                BYTE    #50,#00
03EB   51                BYTE    #51,#04
03ED   52                BYTE    #52,#0C
03EF   54                BYTE    #54,#08
03F1   56                BYTE    #56,#00
03F3   57                BYTE    #57,#08
03F5   59                BYTE    #59,#04
03F7   5A                BYTE    #5A,#0C
03F9   5C                BYTE    #5C,#08
03FB   5E                BYTE    #5E,#04
03FD   60                BYTE    #60,#00
03FF   61                BYTE    #61,#0C
0401   63                BYTE    #63,#08
0403   65                BYTE    #65,#04
0405   67                BYTE    #67,#04
0407   69                BYTE    #69,#00
0409   6B                BYTE    #6B,#00
040B   6D                BYTE    #6D,#00
040D   6F                BYTE    #6F,#00
040F   71                BYTE    #71,#00
0411   73                BYTE    #73,#04
0413   75                BYTE    #75,#04
0415   77                BYTE    #77,#08
0417   79                BYTE    #79,#0C
0419   7C                BYTE    #7C,#00
041B   7E                BYTE    #7E,#04
041D   80                BYTE    #80,#08
041F   82                BYTE    #82,#0C
0421   85                BYTE    #85,#04
0423   87                BYTE    #87,#0C
0425   8A                BYTE    #8A,#04
0427   8C                BYTE    #8C,#0C
0429   8F                BYTE    #8F,#08
042B   92                BYTE    #92,#00
042D   94                BYTE    #94,#0C
042F   97                BYTE    #97,#08
0431   9A                BYTE    #9A,#04
0433   9D                BYTE    #9D,#00
0435   A0                BYTE    #A0,#00

           *
           * K1 parameter decoding table
           *
0437   81  TBLK1         BYTE    #81,#00
0439   82                BYTE    #82,#04
043B   83                BYTE    #83,#04
043D   84                BYTE    #84,#08
```

```
1386 043F   85                  BYTE     #85,#0C
1387 0441   87                  BYTE     #87,#00
1388 0443   88                  BYTE     #88,#04
1389 0445   89                  BYTE     #89,#0C
1390 0447   8B                  BYTE     #8B,#04
1391 0449   8C                  BYTE     #8C,#0C
1392 044B   8E                  BYTE     #8E,#04
1393 044D   90                  BYTE     #90,#00
1394 044F   91                  BYTE     #91,#0C
1395 0451   93                  BYTE     #93,#08
1396 0453   95                  BYTE     #95,#08
1397 0455   97                  BYTE     #97,#04
1398 0457   99                  BYTE     #99,#08
1399 0459   9B                  BYTE     #9B,#08
1400 045B   9D                  BYTE     #9D,#08
1401 045D   9F                  BYTE     #9F,#0C
1402 045F   A2                  BYTE     #A2,#00
1403 0461   A4                  BYTE     #A4,#04
1404 0463   A6                  BYTE     #A6,#0C
1405 0465   A9                  BYTE     #A9,#04
1406 0467   AB                  BYTE     #AB,#08
1407 0469   AE                  BYTE     #AE,#00
1408 046B   B0                  BYTE     #B0,#0C
1409 046D   B3                  BYTE     #B3,#08
1410 046F   B6                  BYTE     #B6,#04
1411 0471   B9                  BYTE     #B9,#00
1412 0473   BC                  BYTE     #BC,#00
1413 0475   BF                  BYTE     #BF,#04
1414 0477   C2                  BYTE     #C2,#04
1415 0479   C5                  BYTE     #C5,#08
1416 047B   C8                  BYTE     #C8,#0C
1417 047D   CC                  BYTE     #CC,#04
1418 047F   CF                  BYTE     #CF,#0C
1419 0481   D3                  BYTE     #D3,#08
1420 0483   D7                  BYTE     #D7,#08
1421 0485   DB                  BYTE     #DB,#04
1422 0487   DF                  BYTE     #DF,#04
1423 0489   E3                  BYTE     #E3,#08
1424 048B   E7                  BYTE     #E7,#0C
1425 048D   EC                  BYTE     #EC,#00
1426 048F   F0                  BYTE     #F0,#04
1427 0491   F4                  BYTE     #F4,#0C
1428 0493   F9                  BYTE     #F9,#0C
1429 0495   FE                  BYTE     #FE,#0C
1430 0497   04                  BYTE     #04,#04
1431 0499   09                  BYTE     #09,#0C
1432 049B   0F                  BYTE     #0F,#04
1433 049D   15                  BYTE     #15,#08
```

```
049F   1C                  BYTE    #1C,#08
04A1   23                  BYTE    #23,#08
04A3   2A                  BYTE    #2A,#0C
04A5   32                  BYTE    #32,#08
04A7   3A                  BYTE    #3A,#08
04A9   42                  BYTE    #42,#0C
04AB   4B                  BYTE    #4B,#08
04AD   54                  BYTE    #54,#00
04AF   5C                  BYTE    #5C,#04
04B1   65                  BYTE    #65,#00
04B3   6E                  BYTE    #6E,#00
04B5   78                  BYTE    #78,#08

              *
              * K2 parameter decoding table
              *
04B7   8A   TBLK2          BYTE    #8A,#00
04B9   98                  BYTE    #98,#00
04BB   A3                  BYTE    #A3,#0C
04BD   AD                  BYTE    #AD,#0C
04BF   B4                  BYTE    #B4,#08
04C1   BA                  BYTE    #BA,#08
04C3   C0                  BYTE    #C0,#00
04C5   C5                  BYTE    #C5,#00
04C7   C9                  BYTE    #C9,#0C
04C9   CE                  BYTE    #CE,#04
04CB   D2                  BYTE    #D2,#0C
04CD   D6                  BYTE    #D6,#0C
04CF   DA                  BYTE    #DA,#0C
04D1   DE                  BYTE    #DE,#08
04D3   E2                  BYTE    #E2,#00
04D5   E5                  BYTE    #E5,#0C
04D7   E9                  BYTE    #E9,#04
04D9   EC                  BYTE    #EC,#0C
04DB   F0                  BYTE    #F0,#00
04DD   F3                  BYTE    #F3,#04
04DF   F6                  BYTE    #F6,#08
04E1   F9                  BYTE    #F9,#0C
04E3   FD                  BYTE    #FD,#00
04E5   00                  BYTE    #00,#00
04E7   03                  BYTE    #03,#04
04E9   06                  BYTE    #06,#04
04EB   09                  BYTE    #09,#04
04ED   0C                  BYTE    #0C,#04
04EF   0F                  BYTE    #0F,#04
04F1   12                  BYTE    #12,#08
04F3   15                  BYTE    #15,#08
04F5   18                  BYTE    #18,#08
```

```
04F7   1B                 BYTE    #1B,#08
04F9   1E                 BYTE    #1E,#08
04FB   21                 BYTE    #21,#08
04FD   24                 BYTE    #24,#0C
04FF   27                 BYTE    #27,#0C
0501   2A                 BYTE    #2A,#0C
0503   2D                 BYTE    #2D,#0C
0505   30                 BYTE    #30,#0C
0507   34                 BYTE    #34,#00
0509   37                 BYTE    #37,#00
050B   3A                 BYTE    #3A,#04
050D   3D                 BYTE    #3D,#00
050F   40                 BYTE    #40,#00
0511   43                 BYTE    #43,#00
0513   46                 BYTE    #46,#00
0515   49                 BYTE    #49,#00
0517   4C                 BYTE    #4C,#00
0519   4F                 BYTE    #4F,#04
051B   52                 BYTE    #52,#04
051D   55                 BYTE    #55,#04
051F   58                 BYTE    #58,#04
0521   5B                 BYTE    #5B,#04
0523   5E                 BYTE    #5E,#00
0525   61                 BYTE    #61,#00
0527   63                 BYTE    #63,#0C
0529   66                 BYTE    #66,#08
052B   69                 BYTE    #69,#04
052D   6C                 BYTE    #6C,#00
052F   6F                 BYTE    #6F,#00
0531   72                 BYTE    #72,#00
0533   76                 BYTE    #76,#04
0535   7C                 BYTE    #7C,#00

          *
          * K3 parameter decoding table
          *
0537   8B TBLK3           BYTE    #8B,#9A,#A2,#A9,#AF,#B5,#BB,#C0
053F   C5                 BYTE    #C5,#CA,#CF,#D4,#D9,#DE,#E2,#E7
0547   EC                 BYTE    #EC,#F1,#F6,#FB,#01,#07,#0D,#14
054F   1A                 BYTE    #1A,#22,#29,#32,#3B,#45,#53,#6D

          *
          * K4 parameter decoding table
          *
0557   94 TBLK4           BYTE    #94,#B0,#C2,#CB,#D3,#D9,#DF,#E5
055F   EA                 BYTE    #EA,#EF,#F4,#F9,#FE,#03,#07,#0C
0567   11                 BYTE    #11,#15,#1A,#1F,#24,#29,#2E,#33
056F   38                 BYTE    #38,#3E,#44,#4B,#53,#5A,#64,#74
```

```

           *
           * K5 parameter decoding table
           *
0577   A3  TBLK5       BYTE    #A3,#C5,#D4,#E0,#EA,#F3,#FC,#04
057F   0C              BYTE    #0C,#15,#1E,#27,#31,#3D,#4C,#66

           *
           * K6 parameter decoding table
           *
0587   AA  TBLK6       BYTE    #AA,#D7,#E7,#F2,#FC,#05,#0D,#14
058F   1C              BYTE    #1C,#24,#2D,#36,#40,#4A,#55,#6A

           *
           * K7 parameter decoding table
           *
0597   A3  TBLK7       BYTE    #A3,#C8,#D7,#E3,#ED,#F5,#FD,#05
059F   0D              BYTE    #0D,#14,#1D,#26,#31,#3C,#4B,#67

           *
           * K8 parameter decoding table
           *
05A7   C5  TBLK8       BYTE    #C5,#E4,#F6,#05,#14,#27,#3E,#58

           *
           * K9 parameter decoding table
           *
05AF   B9  TBLK9       BYTE    #B9,#DC,#EC,#F9,#04,#10,#1F,#45

           *
           * K10 parameter decoding table
           *
05B7   C3  TBLK10      BYTE    #C3,#E6,#F3,#FD,#06,#11,#1E,#43


           ***********************************************************
           *                                                         *
           *    This is the lookup table giving the starting address *
           *    of each concatenation list.                          *
           *                                                         *
           ***********************************************************
05BF 05C5' SENTENCE    DATA    PHRASE0
05C1 05CA'             DATA    PHRASE1
05C3 05CF'             DATA    PHRASE2
           ***********************************************************
           *                                                         *
           *    This is the concatenation table giving the lists     *
           *    of word numbers that define each phrase.  Each       *
           *    list is terminated by an #FF.                        *
           *                                                          *
           ************************************************************
05C5   01  PHRASE0     BYTE    1,2,3,4,#FF
05CA   04  PHRASE1     BYTE    4,3,2,1,#FF
05CF   05  PHRASE2     BYTE    5,4,3,2,1,#FF
           ************************************************************
           *                                                          *
           *    This is the lookup table for the speech stored at     *
           *    voc.                                                  *
           *                                                          *
           ************************************************************
05D5 0000' SPEECH      DATA    #0000
05D7 05E3'             DATA    #0000+VOC     Word 1        “One”
05D9 0667'             DATA    #0084+VOC     Word 2        “Two”
05DB 06D9'             DATA    #00F6+VOC     Word 3        “Three”
05DD 075D'             DATA    #017A+VOC     Word 4        “Four”
05DF 07C3'             DATA    #01E0+VOC     Word 5        “Five”
05E1 086F'             DATA    #028C+VOC     Word 6        “Six”
           ************************************************************
           *                                                          *
           *    This is the DTS speech coded with the D6 coding       *
           *    table                                                 *
           *                                                          *
           ************************************************************
     05E3  VOC
05E3   68              BYTE    #68,#89,#84,#FB,#1A,#53,#64,#B2
05EB   84              BYTE    #84,#87,#33,#C9,#35,#28,#9B,#A1
05F3   D1              BYTE    #D1,#BA,#22,#3A,#94,#8D,#08,#BD
05FB   BE              BYTE    #BE,#40,#1C,#6D,#BA,#BC,#14,#7E
0603   33              BYTE    #33,#CE,#4E,#75,#8D,#EE,#2F,#03
060B   BB              BYTE    #BB,#96,#4A,#46,#D7,#CF,#4A,#DD
0613   4A              BYTE    #4A,#23,#54,#CE,#26,#B7,#74,#A5
061B   9B              BYTE    #9B,#49,#7B,#62,#44,#B7,#32,#2D
0623   95              BYTE    #95,#D9,#C8,#B4,#5B,#9A,#35,#5A
062B   8D              BYTE    #8D,#C2,#DC,#2C,#CC,#5A,#CC,#0A
0633   2B              BYTE    #2B,#6E,#EE,#66,#19,#69,#98,#27
063B   75              BYTE    #75,#33,#CB,#80,#36,#AC,#94,#E6
0643   A9              BYTE    #A9,#85,#CE,#4B,#1B,#EC,#CD,#D4
064B   2C              BYTE    #2C,#50,#71,#52,#F5,#76,#AA,#1B
0653   9B              BYTE    #9B,#38,#98,#58,#33,#56,#B6,#35
065B   D2              BYTE    #D2,#58,#A3,#99,#C8,#7B,#AE,#D5
0663   A8              BYTE    #A8,#5E,#FB,#01,#04,#B0,#78,#BA
066B   2B              BYTE    #2B,#C0,#5D,#1B,#6D,#00,#F7,#65
0673   BA              BYTE    #BA,#01,#64,#BA,#13,#29,#B7,#06
067B   36              BYTE    #36,#81,#C9,#FE,#92,#DB,#5C,#15
0683   20              BYTE    #20,#B8,#7F,#29,#AF,#8A,#CA,#10
068B   DC              BYTE    #DC,#3F,#35,#12,#56,#47,#2A,#FA
```

```
1626 0693   9F                  BYTE     #9F,#FA,#26,#61,#97,#0C,#ED,#77
1627 069B   43                  BYTE     #43,#9A,#6E,#97,#9A,#F7,#8A,#01
1628 06A3   2E                  BYTE     #2E,#CE,#8D,#29,#7B,#48,#17,#B1
1629 06AB   CF                  BYTE     #CF,#86,#B4,#4E,#64,#04,#47,#77
1630 06B3   A1                  BYTE     #A1,#4B,#26,#32,#83,#9B,#13,#31
1631 06BB   AD                  BYTE     #AD,#23,#59,#E3,#DA,#5E,#90,#B2
1632 06C3   85                  BYTE     #85,#AC,#68,#65,#0D,#70,#E9,#4D
1633 06CB   36                  BYTE     #36,#44,#38,#13,#87,#74,#12,#BB
1634 06D3   8D                  BYTE     #8D,#52,#59,#90,#E4,#3D,#08,#60
1635 06DB   CA                  BYTE     #CA,#86,#13,#40,#66,#1A,#46,#00
1636 06E3   B9                  BYTE     #B9,#EC,#8B,#00,#14,#59,#B7,#0A
1637 06EB   90                  BYTE     #90,#5A,#35,#9A,#EC,#1E,#D9,#86
1638 06F3   A4                  BYTE     #A4,#EA,#5C,#41,#69,#85,#B2,#A6
1639 06FB   EE                  BYTE     #EE,#21,#AF,#CC,#24,#46,#63,#F7
1640 0703   94                  BYTE     #94,#53,#26,#E1,#65,#B1,#7B,#C9
1641 070B   3B                  BYTE     #3B,#A5,#77,#B8,#92,#3E,#E5,#9B
1642 0713   B4                  BYTE     #B4,#7B,#18,#EE,#9F,#0A,#5B,#52
1643 071B   02                  BYTE     #02,#B4,#EE,#4F,#8D,#23,#CF,#06
1644 0723   2A                  BYTE     #2A,#B7,#A7,#FE,#96,#04,#0A,#DD
1645 072B   DF                  BYTE     #DF,#D2,#70,#B6,#24,#C6,#9D,#25
1646 0733   61                  BYTE     #61,#3C,#F0,#1C,#F3,#ED,#A4,#30
1647 073B   59                  BYTE     #59,#74,#8E,#70,#E7,#96,#9B,#4C
1648 0743   0A                  BYTE     #0A,#47,#74,#3B,#D1,#CC,#07,#95
1649 074B   21                  BYTE     #21,#BE,#19,#65,#A6,#B3,#27,#20
1650 0753   CE                  BYTE     #CE,#4C,#62,#93,#58,#41,#B4,#77
1651 075B   0A                  BYTE     #0A,#3E,#80,#00,#A6,#6A,#03,#01
1652 0763   54                  BYTE     #54,#A6,#4F,#0C,#10,#C6,#D1,#0B
1653 076B   80                  BYTE     #80,#97,#D4,#E0,#12,#2A,#D7,#37
1654 0773   87                  BYTE     #87,#58,#09,#E9,#18,#B7,#3F,#0D
1655 077B   BD                  BYTE     #BD,#87,#74,#8A,#99,#9F,#86,#DE
1656 0783   43                  BYTE     #43,#D9,#26,#EA,#37,#C5,#EC,#A1
1657 078B   A9                  BYTE     #A9,#B0,#F3,#91,#71,#FE,#30,#60
1658 0793   83                  BYTE     #83,#B3,#B1,#C4,#7F,#1A,#B3,#ED
1659 079B   8E                  BYTE     #8E,#D4,#A2,#3F,#CC,#84,#AD,#4A
1660 07A3   1B                  BYTE     #1B,#E8,#1F,#D6,#EA,#38,#A4,#1C
1661 07AB   E6                  BYTE     #E6,#0F,#5B,#63,#49,#D4,#0F,#F3
1662 07B3   B9                  BYTE     #B9,#83,#B1,#7B,#E2,#87,#7B,#DD
1663 07BB   D5                  BYTE     #D5,#BA,#A8,#E8,#C5,#5D,#0F,#00
1664 07C3   08                  BYTE     #08,#90,#FB,#51,#23,#80,#AB,#19
1665 07CB   4A                  BYTE     #4A,#00,#B9,#97,#0D,#01,#34,#59
1666 07D3   49                  BYTE     #49,#0C,#D0,#A5,#29,#11,#80,#E5
1667 07DB   86                  BYTE     #86,#58,#EA,#BE,#32,#36,#27,#F5
1668 07E3   69                  BYTE     #69,#B5,#4C,#18,#CB,#9B,#DA,#B5
1669 07EB   7A                  BYTE     #7A,#AA,#EC,#61,#45,#6B,#4B,#33
1670 07F3   F0                  BYTE     #F0,#6F,#D1,#94,#25,#A5,#ED,#15
1671 07FB   37                  BYTE     #37,#68,#EA,#9C,#D4,#75,#BA,#ED
1672 0803   34                  BYTE     #34,#6D,#4E,#19,#7B,#CD,#76,#9A
1673 080B   7A                  BYTE     #7A,#BB,#CC,#A2,#F2,#18,#4D,#B9
```

```
1674 0813   59                BYTE    #59,#96,#59,#71,#B4,#A4,#3C,#2A
1675 081B   CB                BYTE    #CB,#BC,#5A,#5C,#52,#67,#A6,#4D
1676 0823   36                BYTE    #36,#36,#AA,#61,#17,#D3,#2E,#6F
1677 082B   22                BYTE    #22,#93,#F4,#05,#61,#1F,#56,#52
1678 0833   69                BYTE    #69,#E7,#41,#B3,#0F,#32,#E1,#AC
1679 083B   E2                BYTE    #E2,#B0,#D9,#EB,#95,#34,#5C,#7E
1680 0843   52                BYTE    #52,#EC,#E5,#44,#1B,#4A,#79,#C1
1681 084B   F6                BYTE    #F6,#3A,#6D,#1C,#9A,#76,#66,#BB
1682 0853   51                BYTE    #51,#32,#16,#89,#94,#99,#DD,#96
1683 085B   8F                BYTE    #8F,#69,#C9,#6A,#D5,#6E,#F2,#52
1684 0863   21                BYTE    #21,#62,#6A,#62,#37,#24,#2D,#22
1685 086B   11                BYTE    #11,#97,#07,#00,#04,#F0,#2A,#08
1686 0873   13                BYTE    #13,#C0,#BF,#F9,#44,#00,#FF,#EE
1687 087B   95                BYTE    #95,#00,#7C,#A5,#D3,#02,#F0,#B5
1688 0883   DA                BYTE    #DA,#94,#62,#C6,#17,#8D,#D9,#B7
1689 088B   4B                BYTE    #4B,#BE,#97,#8B,#25,#CB,#D7,#A5
1690 0893   5A                BYTE    #5A,#AA,#4D,#72,#F7,#DB,#D4,#2F
1691 089B   BD                BYTE    #BD,#4C,#75,#EA,#6B,#5A,#84,#15
1692 08A3   D1                BYTE    #D1,#DD,#BD,#11,#00,#80,#01,#1C
1693 08AB   6F                BYTE    #6F,#6B,#01,#78,#AC,#BE,#05,#E0
1694 08B3   5F                BYTE    #5F,#75,#62,#80,#7F,#D0,#9D,#01
1695 08BB   BE                BYTE    #BE,#8F,#7B,#02,#78,#3B,#5D,#1E
1696 08C3   08                BYTE    #08,#F0,#15,#3E,#13,#C0,#57,#F3
1697 08CB   4C                BYTE    #4C,#00,#7F,#CF,#38,#01,#FC,#81
1698 08D3   32                BYTE    #32,#0C,#F0,#5F,#C2,#85,#62,#C5
1699 08DB   0D                BYTE    #0D,#85,#59,#9B,#5A,#25,#D5,#87
1700 08E3   A4                BYTE    #A4,#AA,#67,#A5,#5A,#04,#5B,#62
1701 08EB   D7                BYTE    #D7,#DC,#52,#4B,#9A,#C9,#A9,#F2
1702 08F3   49                BYTE    #49,#E9,#46,#6D,#37,#94,#FE,#C4
1703 08FB   8C                BYTE    #8C,#75,#B3,#58,#52,#CB,#64,#A6
1704 0903   2C                BYTE    #2C,#53,#23,#47,#A6,#35,#6B,#DE
1705 090B   C8                BYTE    #C8,#9A,#23,#6B,#A5,#55,#E0,#36
1706 0913   C9                BYTE    #C9,#1A,#B7,#D2,#3E,#0E,#26,#67
1707 091B   8D                BYTE    #8D,#4B,#66,#AF,#26,#99,#BB,#D5
1708 0923   40                BYTE    #40,#B5,#97,#2D,#36,#95,#3A,#E6
1709 092B   03                BYTE    #03,#00,#A6,#2A,#5A,#BE,#D6,#45
1710 0933   E8                BYTE    #E8,#50,#C9,#5C,#A9,#EC,#7A,#76
1711 093B   A9                BYTE    #A9,#8C,#91,#65,#B8,#FD,#B6,#54
1712 0943   D6                BYTE    #D6,#3C,#52,#AC,#D9,#5A,#8A,#9B
1713 094B   E9                BYTE    #E9,#11,#6D,#3F,#2D,#E5,#29,#96
1714 0953   50                BYTE    #50,#AE,#E7,#A6,#FE,#92,#2B,#28
1715 095B   75                BYTE    #75,#AB,#DD,#A6,#8F,#29,#D4,#D9
1716 0963   59                BYTE    #59,#00,#0C,#B0,#08,#D4,#0A,#C0
1717 096B   13                BYTE    #13,#E6,#AE,#00,#6B,#7D,#9B,#02
1718 0973   8C                BYTE    #8C,#E5,#32,#0F,#A4,#25,#53,#73
1719 097B   57                BYTE    #57,#50,#53,#D1,#93,#C5,#3C,#5B
1720 0983   65                BYTE    #65,#99,#18,#CA,#7C,#99,#65,#BC
1721 098B   CE                BYTE    #CE,#8D,#65,#4A,#0F,#4D,#9D,#53
```

```
0993   C6                  BYTE     #C6,#9E,#D3,#1C,#65,#4E,#2C,#23
099B   3F                  BYTE     #3F,#3B,#52,#D2,#4F,#95,#9E,#9F
09A3   1D                  BYTE     #1D,#29,#E9,#A7,#4A,#37,#B7,#4F
09AB   A5                  BYTE     #A5,#B2,#35,#A5,#9B,#DB,#A7,#52
09B3   D9                  BYTE     #D9,#9A,#D2,#C9,#93,#93,#A8,#74
09BB   4D                  BYTE     #4D,#E9,#96,#D9,#F2,#54,#B2,#BA
09C3   8C                  BYTE     #8C,#F2,#BA,#09,#69,#9D,#59,#46
09CB   65                  BYTE     #65,#1E,#99,#96,#56,#4C,#A3,#38
09D3   54                  BYTE     #54,#CC,#5B,#0B,#B9,#91,#AD,#1B
09DB   9E                  BYTE     #9E,#C5,#45,#D9,#18,#73,#C2,#0C

               *EXCITATION FUNCTION

4000                       AORG     #4000
4000   00                  BYTE     #00,#A2,#00,#AF,#00,#BA,#00,#C2
4008   00                  BYTE     #00,#C7,#00,#C9,#00,#CA,#00,#C6
4010   00                  BYTE     #00,#C2,#00,#BC,#00,#B5,#00,#AD
4018   00                  BYTE     #00,#A5,#00,#9E,#00,#9A,#00,#95
4020   00                  BYTE     #00,#95,#00,#98,#00,#9F,#00,#A8
4028   00                  BYTE     #00,#B8,#00,#CA,#00,#E3,#00,#FE
4030   01                  BYTE     #01,#1F,#01,#41,#01,#69,#01,#91
4038   01                  BYTE     #01,#BD,#01,#E8,#02,#16,#02,#40
4040   02                  BYTE     #02,#6C,#02,#92,#02,#B9,#02,#D9
4048   02                  BYTE     #02,#F8,#03,#0F,#03,#25,#03,#32
4050   03                  BYTE     #03,#3F,#03,#43,#03,#47,#03,#45
4058   03                  BYTE     #03,#45,#03,#3F,#03,#3D,#03,#3A
4060   03                  BYTE     #03,#3D,#03,#41,#03,#4E,#03,#5F
4068   03                  BYTE     #03,#7B,#03,#A0,#03,#D2,#04,#0D
4070   04                  BYTE     #04,#57,#04,#AD,#05,#11,#05,#82
4078   06                  BYTE     #06,#00,#06,#8A,#07,#1F,#07,#BD
4080   08                  BYTE     #08,#64,#09,#11,#09,#C1,#0A,#74
4088   0B                  BYTE     #0B,#26,#0B,#D5,#0C,#7F,#0D,#20
4090   0D                  BYTE     #0D,#B7,#0E,#40,#0E,#BB,#0F,#24
4098   0F                  BYTE     #0F,#7A,#0F,#BC,#0F,#E9,#0F,#FF
40A0   0F                  BYTE     #0F,#FF,#0F,#E9,#0F,#BC,#0F,#7A
40A8   0F                  BYTE     #0F,#24,#0E,#BB,#0E,#40,#0D,#B7
40B0   0D                  BYTE     #0D,#20,#0C,#7F,#0B,#D5,#0B,#26
40B8   0A                  BYTE     #0A,#74,#09,#C1,#09,#11,#08,#64
40C0   07                  BYTE     #07,#BD,#07,#1F,#06,#8A,#06,#00
40C8   05                  BYTE     #05,#82,#05,#11,#04,#AD,#04,#57
40D0   04                  BYTE     #04,#0D,#03,#D2,#03,#A0,#03,#7B
40D8   03                  BYTE     #03,#5F,#03,#4E,#03,#41,#03,#3D
40E0   03                  BYTE     #03,#3A,#03,#3D,#03,#3F,#03,#45
40E8   03                  BYTE     #03,#45,#03,#47,#03,#43,#03,#3F
40F0   03                  BYTE     #03,#32,#03,#25,#03,#0F,#02,#F8
40F8   02                  BYTE     #02,#D9,#02,#B9,#02,#92,#02,#6C
4100   02                  BYTE     #02,#40,#02,#16,#01,#E8,#01,#BD
4108   01                  BYTE     #01,#91,#01,#69,#01,#41,#01,#1F
```

```
1770 4110    00                     BYTE     #00,#FE,#00,#E3,#00,#CA,#00,#B8
1771 4118    00                     BYTE     #00,#A8,#00,#9F,#00,#98,#00,#95
1772 4120    00                     BYTE     #00,#95,#00,#9A,#00,#9E,#00,#A5
1773 4128    00                     BYTE     #00,#AD,#00,#B5,#00,#BC,#00,#C2
1774 4130    00                     BYTE     #00,#C6,#00,#CA,#00,#C9,#00,#C7
1775 4138    00                     BYTE     #00,#C2,#00,#BA,#00,#AF,#00,#A2
1776 4140    05                     BYTE     #05,#80,#05,#80,#05,#80,#05,#80
1777 4148    05                     BYTE     #05,#80,#05,#80,#05,#80,#05,#80
1778 4150    05                     BYTE     #05,#80,#05,#80,#05,#80,#05,#80
1779 4158    05                     BYTE     #05,#80,#05,#80,#05,#80,#05,#80
1780 4160    3A                     BYTE     #3A,#80,#3A,#80,#3A,#80,#3A,#80
1781 4168    3A                     BYTE     #3A,#80,#3A,#80,#3A,#80,#3A,#80
1782 4170    3A                     BYTE     #3A,#80,#3A,#80,#3A,#80,#3A,#80
1783 4178    3A                     BYTE     #3A,#80,#3A,#80,#3A,#80,#3A,#80
1784 4180    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1785 4188    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1786 4190    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1787 4198    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1788 41A0    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1789 41A8    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1790 41B0    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1791 41B8    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1792 41C0    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
1793 41C8    FF                     BYTE     #FF,#FF,#FF,#FF,#FF,#FF,#FF,#FF
```

C External ROM Initialization

```
*****************************************************
* This is the TSP50C10/11 assembler source for the
* initialization routine for the TSP60C18 speech ROM.
* It assumes that the desired starting byte address
* is located at an arbitrary point in RAM. For
* purposes of checkout, this routine uses the memory
* location #10 for the most significant byte of the
* address and the memory location #11 for the least
* significant byte of the address.
*
* In actual use, the values given for Addr_MSB and
* Addr_LSB in the equate block should be replaced so
* as to point to the actual location in RAM used in
* the program.
*
* The following interconnections are assumed by the
* routine:
*
*    TSP50C10/11      TSP60C18
*
*       B(0) ------------ STR
*       B(1) ------------ R/W
*       A(0) ------------ C(0)
*       A(1) ------------ C(1)
*       A(2) ------------ C(2)
*       A(3) ------------ C(3)
*       A(7) ------------ SCLK
*
*   A0 and CEB on the TSP60C18 should be tied low.
*   HCLB on the TSP60C18 should be tied high.
*
* After calling this program, use the standard
* synthesis routine.
*
* To use internal speech afterwards, clear the
* EXTROM bit of the mode register.
*
```

```
* The strategy used in this routine is as follows:
*
*      Put the TSP50C10/11 in external ROM mode.
*      Do a dummy write.
*      Load the 16-bit starting ROM address four
*         bits at a time. For each nibble of the
*         address, present the nibble to the
*         TSP60C18 by outputting it on the TSP50C10/11
*         ports A(0) to A(3) and then pulsing B(0)
*         low.
*      Do a dummy read to ensure that the internal
*         pointers in the TSP60C18 are OK.
*      Burn eight instruction cycles.
*      Perform two GET 2 instructions.
*      If the 16-bit address is odd, then do a GET 8.
*
*
* Although the TSP60C18 is internally organized on
* word (16-bit) boundaries, the address that this
* subroutine uses is expressed in byte (8-bit)
* boundaries.  The address located at Addr_MSB and
* Addr_LSB is therefore right-shifted one bit before
* being loaded to the TSP60C18.  If the original
* address contains a one in the least significant bit
* position, a GET8 instruction is executed at the
* end of the program to move one byte further
* (halfway between word boundaries) in memory.
*
* When this routine is used to address an external
* ROM, both pins of Port B and A(0,1,2,3,7) are
* dedicated for use in addressing the ROM.  The Port
* B pins and A(7) need to be maintained as outputs
* in their initialized state when not in this routine
* or the address loaded in the ROM may be lost.
*
* This routine is reached by a
*
* CALL INIT
*
```

```
        * instruction.
        *
        *
        ******************************************************
        *
        *   EQUATE BLOCK
        *
        ******************************************************
        *
        *   Data Address
        *
0010    Addr_MSB  EQU       #10    Most significant byte of addr
0011    Addr_LSB  EQU       #11    Least significant byte of addr
        *
        * Mode Buffer - Because the contents of the mode
        * register cannot be read and because other bits in
        *  the mode register need to be maintained when a bit
        * is set or cleared, a copy of the mode register
        * is maintained in RAM.  The copy is first changed
        * and then the copy is written to the mode register.
        *
0012    Mode_buf  EQU       #12    Address of copy of mode register
        *
        * Temp - A scratch working register to
        * use for massaging the address.
        *
0013    Temp      EQU       #13    Temporary working register
        *
        * The following data is used to set or clear the
        * EXTROM bit in the MODE Register.
        *
0010    ExtRom    EQU       #10    Logically OR mode with this
00EF    IntRom    EQU       #EF    Logically AND mode with this
        *
        * Output port definitions
        *
0080    Input_A   EQU       #80    Read here for Port A input
0081    Special_A EQU       #81    Set to 1 for open drain
0082    IO_A      EQU       #82    Set to 0 for input, 1 for output
```

```
     0083 Output_A  EQU     #83    Write here for Port A output
     0084 Input_B   EQU     #84    Read here for Port B input
     0085 Special_B EQU     #85    Set to 1 for open drain
     0086 IO_B      EQU     #86    Set to 0 for input, 1 for output
     0087 Output_B  EQU     #87    Write here for Port B output
          *******************************************************
          *
          * Start Routine
          *
          *******************************************************
          *
          * In general, when A(0,1,2,3,7) are used as outputs,
          * the other Port A pins should not be disturbed, so
          * the required bits are masked.  An OR is performed
          * in the required high states.
          *
          *
0000 62   INIT      TCX     IO_A      Set up Port A7 to output
0001 82
0002 65             ANDCM   #7F
0003 7F
0004 64             ORCM    #80
0005 80
0006 62             TCX     Output_A  Set Port A7 to 1
0007 83
0008 65             ANDCM   #7F
0009 7F
000A 64             ORCM    #80
000B 80
          *
          *
          * The B port is simpler because all Port B pins
          * are changed.
          *
000C 6E             TCA     #03       Set Port B output data
000D 03
000E 6A             TAMD    Output_B  bits high
000F 87
0010 6A             TAMD    IO_B      Set Port B to output state
```

```
0011 86
          *
          * Set external ROM mode by ORing the correct bit in
          * the RAM location that is used to maintain a copy of
          * the current state of the mode register.  Then
          * write the result to the mode register.
          *
0012 62             TCX       Mode_buf  Point to local copy
0013 12
0014 64             ORCM      ExtRom    Set bit to local copy
0015 10
0016 11             TMA                 Copy local copy
0017 1D             TAMODE              to mode register
          * Do a dummy write
0018 62             TCX       Output_B        B1          B0
0019 87
001A  6             ANDCM     #FD          R/W_  = 0    STRB = 1
001B FD
001C 65             ANDCM     #FE          R/W_  = 0    STRB = 0
001D FE
001E 64             ORCM      #01          R/W_  = 0    STRB = 1
001F 01
0020 64             ORCM      #03          R/W_  = 1    STRB = 1
0021 03
          *
          * Intialize the internal registers of the TSP50C10/11
          * for new input by performing a LUAPS.
          *
0022 6C             LUAPS
          *
          * Set up A0-A3,A7 as output.
          *
0023 62             TCX       IO_A
0024 82
0025 64             ORCM      #8F
0026 8F
          *
          *
          * Present lower nibble of LSB of address to ROM.
```

```
0176                 *
0177   0027 69                TMAD      Addr_LSB  Get LSB
       0028 11
0178   0029 15                SARA                Divide address by 2
0179   002A 62                TCX       TEMP
       002B 13
0180   002C 16                TAM                 Move to working register
0181   002D 65                ANDCM     #0F       Mask off upper nibble
       002E 0F
0182   002F 64                ORCM      #80       Ensure A(7) high
       0030 80
0183   0031 11                TMA                 Move nibble through A reg
0184   0032 6A                TAMD      Output_A  to output
       0033 83
0185                 *
0186                 * Latch first nibble of address to ROM.
0187                 *
0188   0034 6E                TCA       #00       Str Low
       0035 00
0189   0036 6A                TAMD      Output_B
       0037 87
0190                 *
0191   0038 6E                TCA       #01       Str High
       0039 01
0192   003A 6A                TAMD      Output_B
       003B 87
0193                 *
0194                 * Present upper nibble of LSB of address to ROM.
0195                 *
0196   003C 69                TMAD      Addr_LSB  Get LSB
       003D 11
0197   003E 15                SARA                Position 2nd
0198   003F 15                SARA                nibble
0199   0040 15                SARA
0200   0041 15                SARA
0201   0042 15                SARA
0202   0043 16                TAM                 Move to working register
0203   0044 65                ANDCM     #0F       Mask off high bits
       0045 0F
```

```
0046 64              ORCM      #88        Ensure A(7), high bit of
0047 88                                   nibble set high
                *
                * If lower bit of MSB is low, then transfer
                * that to upper bit of upper LSB nibble.
                *
0048 62              TCX       Addr_MSB   Look at MSB of address
0049 10
004A 66              TSTCM     #01        Is lower bit high?
004B 01
004C 40              BR        INIT_1     yes, do nothing
004D 52
004E 62              TCX       TEMP       no, reset bit in working
004F 13
0050 65              ANDCM     #F7        register to same state
0051 F7
                *
0052 69         INIT_1 TMAD    TEMP       Move nibble through A reg
0053 13
0054 6A              TAMD      Output_A   to output
0055 83
                *
                * Latch second nibble of address to ROM.
                *
0056 6E              TCA       #00        Str Low
0057 00
0058 6A              TAMD      Output_B
0059 87
                *
005A 6E              TCA       #01        Str High
005B 01
005C 6A              TAMD      Output_B
005D 87
                *
                * Present lower nibble of MSB of address to ROM.
                *
005E 69              TMAD      Addr_MSB     Get MSB
005F 10
0060 62              TCX       Temp
```

```
0061 13
0062 15             SARA                Divide address by 2
0063 16             TAM                 Move to working register
0064 65             ANDCM     #0F       Mask off upper nibble
0065 0F
0066 64             ORCM      #80       Ensure A(7) high
0067 80
0068 11             TMA                 Move nibble through A reg
0069 6A             TAMD      Output_A  to output
006A 83
          *
          * Latch third nibble of address to ROM.
          *
006B 6E             TCA       #00       Str Low
006C 00
006D 6A             TAMD      Output_B
006E 87
          *
006F 6E             TCA       #01       Str High
0070 01
0071 6A             TAMD      Output_B
0072 87
          *
          * Present upper nibble of MSB of address to ROM.
          *
0073 69             TMAD      Addr_MSB  Get MSB
0074 10
0075 15             SARA                Position most significant
0076 15             SARA                nibble for
0077 15             SARA                output
0078 15             SARA
0079 15             SARA
007A 16             TAM                 Move to working register
007B 65             ANDCM     #0F       Mask off upper nibble
007C 0F
007D 64             ORCM      #80       Ensure A(7) high
007E 80
007F 11             TMA                 Move nibble through A reg
0080 6A             TAMD      Output_A  to output
```

```
      0081 83
0258              *
0259              * Latch fourth nibble of address to ROM.
0260              *
0261  0082 6E               TCA       #00       Str Low
      0083 00
0262  0084 6A               TAMD      Output_B
      0085 87
0263              *
0264  0086 6E               TCA       #01       Str High
      0087 01
0265  0088 6A               TAMD      Output_B
      0089 87
0266              *
0267              * Place Port A0-A3 in the high-impedance state.
0268              *
0269  008A 62               TCX       IO_A
      008B 82
0270  008C 65               ANDCM     #F0
      008D F0
0271              *
0272              * Set R/W_ to 1.
0273              *
0274  008E 62               TCX       Output_B
      008F 87
0275  0090 64               ORCM      #03
      0091 03
0276              *
0277              * Burn 8 instruction cycles.
0278              *
0279  0092 2F               CLA
0280  0093 2F               CLA
0281  0094 2F               CLA
0282  0095 2F               CLA
0283  0096 2F               CLA
0284  0097 2F               CLA
0285  0098 2F               CLA
                            CLA
0286              *
```

```
          * Do 2 GET2s.
          *
0099 31             GET       2
009A 31             GET       2
          *
          * As the above address was loaded, it was divided
          * to change it from the byte address that
          * was loaded in RAM into the word address that
          * the ROM expects.  If the original address
          * was odd, get 8 bits from the ROM to
          * move one byte further into the ROM to get to the
          * correct byte boundary.
          *
009B 62             TCX       Addr_LSB  Look at MSB of address
009C 11
009D 66             TSTCM     #01       Is address odd?
009E 01
009F 40             BR        INIT_2    yes, get another byte
00A0 A2
00A1 3D             RETN                no, do nothing
00A2 31   INIT_2    GET       2         Get a byte in 3 stages
00A3 31             GET       2
00A4 33             GET       4
00A5 3D             RETN
          *******************************************************
```

D DTMF Program

```
DTMF.asm       TSP50C10 Assembler  Version 1.08

0001                           OPTION  BUNLIST,DUNLIST,PAGEOF
0002               ***********************************************************
0003               *                                                         *
0004               * DTMF_GEN  - This is a sample program which generates    *
0005               *             a DTMF tone. In this sample, tones are      *
0006               *             generated in sequence, triggered by        *
0007               *             bit 0 of port A going high, and stopped     *
0008               *             by that bit going low.                      *
0009               *                                                         *
0010               ***********************************************************
0011 0000                      AORG    #0000
0012 0000    69    GO          TMAD    0
     0001    00
0013
0014 0002    2F                CLA              -Initialize mode register
0015 0003    1D                TAMODE
0016
0017 0004    20                CLX
0018 0005    13    RAM_LOOP    TAMIX            -Initialize All RAM to zeros
0019 0006    61                XGEC    MAX_RAM+1
     0007    80
0020 0008    40                BR      GOGO
     0009    8C
0021 000A    40                BR      RAM_LOOP
     000B    05
0022               ***********************************************************
0023               *   Interrupt vectors
0024               ***********************************************************
0025 0010                      AORG    #0010
0026
0027 0010    A2                SBR     INT2_01  -Timer Underflow, PCM=0, LPC=1
0028 0011    A2                SBR     INT2_01  -Timer Underflow, PCM=0, LPC=1
0029
0030 0012    A2                SBR     INT2_00  -Timer Underflow, PCM=0, LPC=0
0031 0013    A2                SBR     INT2_00  -Timer Underflow, PCM=0, LPC=0
0032
0033 0014    A2                SBR     INT2_11  -Timer Underflow, PCM=1, LPC=1
0034 0015    A2                SBR     INT2_11  -Timer Underflow, PCM=1, LPC=1
0035
0036 0016    A2                SBR     INT2_10  -Timer Underflow, PCM=1, LPC=0
0037 0017    A2                SBR     INT2_10  -Timer Underflow, PCM=1, LPC=0
0038
0039 0018    A2                SBR     INT1_01  -PPC < 16 samples interrupt
0040 0019    A2                SBR     INT1_01  -PPC < 16 samples interrupt
0041
0042 001A    A2                SBR     INT1_00  -Pin (B1) goes low interrupt
```

```
0043 001B   A2                SBR     INT1_00   -Pin (B1) goes low interrupt
0044
0045 001C   A2                SBR     INT1_11   -10 kHz Clock interrupt
0046 001D   A2                SBR     INT1_11   -10 kHz Clock interrupt
0047
0048 001E   A0                SBR     INT1_10   -20 kHz Clock interrupt
0049 001F   A0                SBR     INT1_10   -20 kHz Clock interrupt
0050            *
0051 0020   40  INT1_10       BR      INTPCM    -PCM service routine
     0021   E6
0052            *
0053      0022  INT1_01
0054      0022  INT2_00
0055      0022  INT2_01
0056      0022  INT2_10
0057      0022  INT2_11
0058      0022  INT1_00
0059 0022   2F  INT1_11       CLA
0060 0023   3E                RETI
0061            *
0062            *    PCM register variables
0063            *
0064      0000  PERIOD1       EQU     #00       -Period of 1st Wave
0065      0001  TIME1         EQU     #01       -Cumulative angle of 1st wave
0066      0002  PERIOD2       EQU     #02       -Period of 2nd Wave
0067      0003  TIME2         EQU     #03       -Cumulative angle of 2nd wave
0068      0004  PCMBUF        EQU     #04       -Intermediate data buffer
0069            *
0070            *
0071            *    LPC status variable locations
0072            *
0073      0010  MODE_BUF      EQU     #10           ;Mode register buffer
0074            *
0075            *   Device Constants
0076            *
0077      007F  MAX_RAM       EQU     #7F       -Highest RAM location
0078            *
0079            *   MODE Register Bit Definitions
0080            *
0081      0001  ENA1          EQU     #01       -Enable Level 1 interrupt
0082      0002  LPC           EQU     #02       -Enable LPC synthesis
0083      0004  PCM           EQU     #04       -Enable PCM synthesis
0084      0008  ENA2          EQU     #08       -Enable Level 2 interrupt
0085      0010  EXTROM        EQU     #10       -Set external ROM mode
0086      0020  RAMROM        EQU     #20       -Enable GETs from RAM
0087      0040  MASTER        EQU     #40       -Master/Slave Toggle
0088      0080  UNV           EQU     #80       -Enable Unvoiced excitation
0089            *
```

```
               *     DTMF tone definition table
               *
               *     Program assumes a 10kHz sampling frequency and
               *     has a spacing of 11.25 degrees between entries in
               *     the sine wave table, so value for a frequency is
               *
               *               (freq * 360 degrees)
               *     value =  ——————————————————————    * 128
               *              (10kHz * 11.25 degrees)

               *      The bottom 8 bits are fractional
               *
     0024  80  DTMF         RBYTE   #01,#81,#02,#23   -zero = 941 Hz+1336 Hz
     0028  80               RBYTE   #01,#1D,#01,#EF   -One  = 697 Hz+1209 Hz
     002C  80               RBYTE   #01,#1D,#02,#23   -two  = 697 Hz+1336 Hz
     0030  80               RBYTE   #01,#1D,#02,#5D   -three= 697 Hz+1477 Hz
     0034  80               RBYTE   #01,#3B,#01,#EF   -four = 770 Hz+1209 Hz
     0038  80               RBYTE   #01,#3B,#02,#23   -five = 770 Hz+1336 Hz
     003C  80               RBYTE   #01,#3B,#02,#5D   -six  = 770 Hz+1477 Hz
     0040  80               RBYTE   #01,#5D,#01,#EF   -seven= 852 Hz+1209 Hz
     0044  80               RBYTE   #01,#5D,#02,#23   -eight= 852 Hz+1336 Hz
     0048  80               RBYTE   #01,#5D,#02,#5D   -nine = 852 Hz+1477 Hz
               *
               *     Digitized sine wave table
               *
     004C  00  SINEW        BYTE    #00,#19  0 degrees—>11.25 degrees
     004E  31               BYTE    #31,#18  11.25 degrees—>22.5  degrees
     0050  31               BYTE    #31,#16  22.5  degrees—>33.75 degrees
     0052  5A               BYTE    #5A,#13  33.75 degrees—>45    degrees
     0054  5A               BYTE    #5A,#10  45.0  degrees—>56.25 degrees
     0056  75               BYTE    #75,#0B  56.25 degrees—>67.5  degrees
     0058  75               BYTE    #75,#08  67.5  degrees—>78.75 degrees
     005A  7F               BYTE    #7F,#02  78.75 degrees—>90    degrees
     005C  7F               BYTE    #7F,#FE  90.0  degrees—>101.25 degrees
     005E  75               BYTE    #75,#F8  101.25 degrees—>112.5  degrees
     0060  75               BYTE    #75,#F5  112.5  degrees—>123.75 degrees
     0062  5A               BYTE    #5A,#F0  123.75 degrees—>135.0  degrees
     0064  5A               BYTE    #5A,#ED  135.0  degrees—>146.25 degrees
     0066  31               BYTE    #31,#EA  146.25 degrees—>157.5  degrees
     0068  31               BYTE    #31,#E8  157.5  degrees—>168.75 degrees
     006A  00               BYTE    #00,#E7  168.75 degrees—>180.0  degrees
     006C  00               BYTE    #00,#E7  180.0  degrees—>191.25 degrees
     006E  CF               BYTE    #CF,#E8  191.25 degrees—>202.5  degrees
     0070  CF               BYTE    #CF,#EA  202.5  degrees—>213.75 degrees
     0072  A6               BYTE    #A6,#ED  213.75 degrees—>225.0  degrees
     0074  A6               BYTE    #A6,#F0  225.0  degrees—>236.25 degrees
     0076  8B               BYTE    #8B,#F5  236.25 degrees—>247.5  degrees
     0078  8B               BYTE    #8B,#F8  247.5  degrees—>258.75 degrees
```

```
0138 007A   81              BYTE    #81,#FE  258.75 degrees-->270.0  degrees
0139 007C   81              BYTE    #81,#02  270.0  degrees-->281.25 degrees
0140 007E   8B              BYTE    #8B,#08  281.25 degrees-->292.5  degrees
0141 0080   8B              BYTE    #8B,#0B  292.5  degrees-->303.75 degrees
0142 0082   A6              BYTE    #A6,#10  303.75 degrees-->315.0  degrees
0143 0084   A6              BYTE    #A6,#13  315.0  degrees-->326.25 degrees
0144 0086   CF              BYTE    #CF,#16  326.25 degrees-->337.5  degrees
0145 0088   CF              BYTE    #CF,#18  337.5  degrees-->348.75 degrees
0146 008A   00              BYTE    #00,#19  348.75 degrees-->360   degrees
0147             ***********************************************************
0148             *   Main body of program
0149             ***********************************************************
0150 008C   6E   GOGO       TCA     0          -Tone 'Zero'
     008D   00
0151 008E   00              CALL    DO_PCM
     008F   B5
0152             *
0153 0090   6E              TCA     1          -Tone 'One'
     0091   01
0154 0092   00              CALL    DO_PCM
     0093   B5
0155             *
0156 0094   6E              TCA     2          -Tone 'Two'
     0095   02
0157 0096   00              CALL    DO_PCM
     0097   B5
0158             *
0159 0098   6E              TCA     3          -Tone 'Three'
     0099   03
0160 009A   00              CALL    DO_PCM
     009B   B5
0161             *
0162 009C   6E              TCA     4          -Tone 'Four'
     009D   04
0163 009E   00              CALL    DO_PCM
     009F   B5
0164             *
0165 00A0   6E              TCA     5          -Tone 'Five'
     00A1   05
0166 00A2   00              CALL    DO_PCM
     00A3   B5
0167             *
0168 00A4   6E              TCA     6          -Tone 'Six'
     00A5   06
0169 00A6   00              CALL    DO_PCM
     00A7   B5
0170             *
0171 00A8   6E              TCA     7          -Tone 'Seven'
```

```
00A9   07
00AA   00              CALL    DO_PCM
00AB   B5
           *
00AC   6E              TCA     8        -Tone 'Eight'
00AD   08
00AE   00              CALL    DO_PCM
00AF   B5
           *
00B0   6E              TCA     9        -Tone 'Nine'
00B1   09
00B2   00              CALL    DO_PCM
00B3   B5
           *
00B4   3F              SETOFF
           *
           ***********************************************************
           *
           *   DO_PCM
           *
           *   This is the routine that sets up the DTMF tone.
           *   It waits for port PA0 to go high, then plays
           *   the DTMF tone specified by the contents of the
           *   A register until PA0 goes low.
           *
           ***********************************************************
00B5   62  DO_PCM      TCX     #80      -Point to port A
00B6   80
00B7   66              TSTCM   #01      -Loop until A(0)
00B8   01
00B9   40              BR      GO_PCM      goes high
00BA   BD
00BB   40              BR      DO_PCM
00BC   B5
           *
00BD   2E  GO_PCM      SALA             -Adjust value to
00BE   2E              SALA                table index
00BF   70              ACAAC   DTMF     -Add offset of table
00C0   24
00C1   6C              LUAPS            -Point to table entry

00C2   37              GET     8        -Get first frequency
00C3   37              GET     8           period
00C4   6A              TAMD    PERIOD1  -Store it away
00C5   00

00C6   37              GET     8        -Get second frequency
00C7   37              GET     8           period
```

```
00C8   6A                TAMD    PERIOD2   -Store it away
00C9   02

00CA   2F                CLA               -Clear cumulative data
00CB   6A                TAMD    TIME1
00CC   01
00CD   6A                TAMD    TIME2
00CE   03

00CF   62                TCX     MODE_BUF  -Turn on PCM and INT1
00D0   10
00D1   64                ORCM    PCM
00D2   04
00D3   64                ORCM    ENA1
00D4   01
00D5   11                TMA
00D6   1D                TAMODE

00D7   62   L1           TCX     #80       -Loop until A(0)
00D8   80
00D9   66                TSTCM   #01          goes low
00DA   01
00DB   40                BR      L1
00DC   D7

00DD   62                TCX     MODE_BUF  -Turn off PCM and INT1
00DE   10
00DF   65                ANDCM   ~PCM
00E0   FB
00E1   65                ANDCM   ~ENA1
00E2   FE
00E3   11                TMA
00E4   1D                TAMODE
00E5   3D                RETN

            ***********************************************************
            *   PCM interrupt service routine
            ***********************************************************
00E6   3B   INTPCM       INTGR
00E7   20                CLX

00E8   14                TMAIX             -Add delta angle to
00E9   28                AMAAC                cumulative angle

00EA   16                TAM               -Save cumulative angle
00EB   11                TMA               -Discard high bits of cum

00EC   68                AXCA    01        -right shift 7 bits
```

```
     00ED   01
0244 00EE   2E                 SALA                 -Left 1 bit
0245 00EF   70                 ACAAC    SINEW       -Add table offset
     00F0   4C
0246
0247 00F1   3C                 EXTSG
0248 00F2   6D                 LUAB                 -get data point
0249 00F3   3A                 IAC
0250 00F4   6B                 LUAA                 -get slope between points
0251 00F5   39                 AXMA                 -interpolate slope
0252 00F6   2C                 ABAAC                -add interpolated slope
0253 00F7   1B                 SALA4                   and scale for DAC
0254 00F8   68                 AXCA     #78         -Scale value for twist
     00F9   78
0255
0256 00FA   6A                 TAMD     PCMBUF      -Save intermediate data
     00FB   04
0257
0258 00FC   3B                 INTGR
0259 00FD   21                 IXC
0260 00FE   14                 TMAIX                -Add delta angle to
0261 00FF   28                 AMAAC                   cumulative angle
0262
0263 0100   16                 TAM                  -Save cumulative angle
0264 0101   11                 TMA                  -Discard high bits of angle
0265
0266 0102   68                 AXCA     01          -right shift 7 bits
     0103   01
0267 0104   2E                 SALA                 -Left 1 bit
0268 0105   70                 ACAAC    SINEW       -Add table offset
     0106   4C
0269
0270 0107   3C                 EXTSG
0271 0108   6D                 LUAB                 -get data point
0272 0109   3A                 IAC
0273 010A   6B                 LUAA                 -get slope between points
0274 010B   39                 AXMA                 -interpolate slope
0275
0276 010C   2C                 ABAAC                -add interpolated slope
0277 010D   1B                 SALA4                   and scale
0278
0279 010E   1A                 TAB                  -Store 2nd data point
0280
0281 010F   21                 IXC                  -Retrieve 1st data point
0282 0110   11                 TMA
0283
0284 0111   2C                 ABAAC                -Sum two waves together
0285 0112   15                 SARA                    and normalize
```

```
0286 0113   1C              TASYN              —transfer data to D/A
0287 0114   3E              RETI
```

E TSP50C10/11 Sample Music Program

```
                      OPTION  BUNLIST,DUNLIST,PAGEOF
           **********************************************************
           *
           *   MINUET.ASM
           *
           *   LPC can also be used to generate music.  In this
           *   program, the LPC filter is set to a narrow bandwidth
           *   filter that will only pass a single frequency.
           *   by appropriately varying the parameters, we play
           *   Minuet in G by Mozart.
           *
           **********************************************************
           *   RAM USAGE
           **********************************************************
     0001  EN         EQU     #01       -EN working value
     000C  K2         EQU     #0C       -K2 Working Value
     000D  K1         EQU     #0D       -K1 Working Value
     000E  C1         EQU     #0E       -C1 Parameter
     000F  C2         EQU     #0F       -C2 Parameter

     0010  TIME       EQU     #10       -Note Duration
     0011  ENERGY     EQU     #11       -Temp storage for energy
     0012  MODE_BUF   EQU     #12       -Mode register Buffer
     0013  EndSong    EQU     #13       -End of song flag
           *
           *   Device Constants
           *
     0F61  C1_Value   EQU     #F61      -C1 Value
     0B67  C2_Value   EQU     #B67      -C2 Value
     007F  MAX_RAM    EQU     #7F       -Highest RAM location
           *
           *   MODE Register Bit Definitions
           *
     0001  ENA1       EQU     #01       -Enable Level 1 interrupt
     0002  LPC        EQU     #02       -Enable LPC systhesis
     0004  PCM        EQU     #04       -Enable PCM synthesis
     0008  ENA2       EQU     #08       -Enable Level 2 interrupt
     0010  EXTROM     EQU     #10       -Set external ROM mode
     0020  RAMROM     EQU     #20       -Enable GETs from RAM
     0040  MASTER     EQU     #40       -Master/Slave Toggle
     0080  UNV        EQU     #80       -Enable Unvoiced excitation
           **********************************************************
           *  BEGINNING OF PROGRAM
           **********************************************************
0000                  AORG    #0000
0000    69            TMAD    0
0001    00
```

```

0002   2F              CLA             -Initialize mode register
0003   1D              TAMODE

0004   20              CLX
0005   13  RAM_LOOP    TAMIX           -Initialize All RAM to zeros
0006   61              XGEC    MAX_RAM+1
0008   40              BR      GOGO
0009   21
000A   40              BR      RAM_LOOP
000B   05
           ***********************************************************
           *   Interrupt vectors
           ***********************************************************
0010                   AORG    #0010
0010   A0              SBR     INT2_01  -Timer Underflow, PCM=0, LPC=1
0011   A0              SBR     INT2_01  -Timer Underflow, PCM=0, LPC=1
0012   A0              SBR     INT2_00  -Timer Underflow, PCM=0, LPC=0
0013   A0              SBR     INT2_00  -Timer Underflow, PCM=0, LPC=0
0014   A0              SBR     INT2_11  -Timer Underflow, PCM=1, LPC=1
0015   A0              SBR     INT2_11  -Timer Underflow, PCM=1, LPC=1
0016   A0              SBR     INT2_10  -Timer Underflow, PCM=1, LPC=0
0017   A0              SBR     INT2_10  -Timer Underflow, PCM=1, LPC=0
0018   A0              SBR     INT1_01  -PPC < 16 Samples interrupt
0019   A0              SBR     INT1_01  -PPC < 16 Samples interrupt
001A   A0              SBR     INT1_00  -Pin (B1) goes low interrupt
001B   A0              SBR     INT1_00  -Pin (B1) goes low interrupt
001C   A0              SBR     INT1_11  -10 kHz Clock interrupt
001D   A0              SBR     INT1_11  -10 kHz Clock interrupt
001E   A0              SBR     INT1_10  -20 kHz Clock interrupt
001F   A0              SBR     INT1_10  -20 kHz Clock interrupt
     0020  INT1_01
           *
     0020  INT2_00
     0020  INT2_01
     0020  INT2_10
     0020  INT2_11
     0020  INT1_00
     0020  INT1_10
0020   3E  INT1_11     RETI
           *
           ***********************************************************
           *   MAIN BODY OF PROGRAM
           ***********************************************************
0021   2F  GOGO        CLA              -Point to start of song
0022   70              ACAAC   NOTES
0023   84
0024   6C              LUAPS
```

```
0092
0093 0025   2F                CLA                  –Load C1 Value
0094 0026   7F                ACAAC    C1_VALUE
     0027   61
0095 0028   6A                TAMD     C1
     0029   0E
0096
0097 002A   2F                CLA                  –Load C2 Value
0098 002B   7B                ACAAC    C2_VALUE
     002C   67
0099 002D   6A                TAMD     C2
     002E   0F
0100
0101 002F   33                GET      4           –Get song tempo
0102 0030   33                GET      4
0103 0031   19                TAPSC
0104
0105 0032   00                CALL     LoadNote    –Load the first note data
     0033   56
0106
0107 0034   62                TCX      Mode_Buf    –Turn on LPC Mode
     0035   12
0108 0036   64                ORCM     LPC
     0037   02
0109 0038   11                TMA
0110 0039   1D                TAMODE
0111
0112 003A   6E                TCA      #ff         –Start countdown timer
     003B   FF
0113 003C   1E                TATM
0114
0115 003D   69  Loop          TMAD     EndSong     –Test end of song flag
     003E   13
0116 003F   63                AGEC     1           –Is song over?
     0040   01
0117 0041   40                BR       StopSong        yes, turn off LPC
     0042   4F
0118
0119 0043   17                TTMA                 –Get timer value
0120 0044   60                ANEC     0           –Time to decrement TIME?
     0045   00
0121 0046   40                BR       Loop            no, loop back
     0047   3D
0122
0123 0048   62                TCX      TIME        –Point to note duration
     0049   10
0124 004A   27                DECMN                –Is it time to get new note?
0125 004B   00                CALL     LoadNote        yes, get new note
```

```
004C   56
004D   40                BR      Loop         no, wait some more
004E   3D

004F   62   StopSong     TCX     Mode_Buf  –Turn off LPC Mode
0050   12
0051   65                ANDCM   ~LPC
0052   FD
0053   11                TMA
0054   1D                TAMODE

0055   3F                SETOFF            –Turn of device

            ************************************************************
            *   This subroutine loads in data for the next note
            ************************************************************
0056   2F   LoadNote     CLA               –Zero energy while we change
0057   6A                TAMD    EN           the filter parameters
0058   01

0059   33                GET     4         –Get the note duration
005A   33                GET     4
005B   6A                TAMD    TIME
005C   10

005D   60                ANEC    0         –End of song?
005E   00
005F   40                BR      Continue     no, continue
0060   67

0061   6E                TCA     1         –Signal...
0062   01
0063   6A                TAMD    EndSong      end of song...
0064   13
0065   40                BR      RelaxK2      and allow sound to die
0066   7E

0067   37   Continue     GET     8         –Get Note Energy
0068   6A                TAMD    ENERGY
0069   11

006A   37                GET     8         –Get pitch value
006B   37                GET     8
006C   1C                TASYN

006D   37                GET     8         –Get first filter parameter
006E   37                GET     8
006F   6A                TAMD    K1
```

```
     0070   0D
0162
0163 0071   2F              CLA                 -Get bandwidth
0164 0072   77              ACAAC   #7f8
     0073   F8
0165 0074   6A              TAMD    K2
     0075   0C
0166
0167 0076   69              TMAD    ENERGY      -Load energy to filter
     0077   11
0168 0078   6A              TAMD    EN
     0079   01
0169
0170 007A   60              ANEC    0           -Is note a rest?
     007B   00
0171 007C   40              BR      LoadNoteX       no, exit routine
     007D   83
0172
0173 007E   2F  RelaxK2     CLA                 -Note is a rest,
0174 007F   77              ACAAC   #780            relax filter bandwidth
     0080   80
0175 0081   6A              TAMD    K2              so sound can die down
     0082   0C
0176
0177 0083   3D  LoadNoteX   RETN
0178
0179
0180 0084   C8  NOTES       RBYTE   #13                         Tempo
0181 0085   02              RBYTE   #40,#06,#01,#B4,#08,#D6   note = 17, fre
0182 008B   04              RBYTE   #20,#06,#02,#8E,#08,#60   note = 10, fre
0183 0091   04              RBYTE   #20,#06,#02,#46,#08,#79   note = 12, fre
0184 0097   04              RBYTE   #20,#06,#02,#06,#08,#99   note = 14, fre
0185 009D   04              RBYTE   #20,#06,#01,#EA,#08,#AA   note = 15, fre
0186
0187 00A3   02              RBYTE   #40,#06,#01,#B4,#08,#D6   note = 17, fre
0188 00A9   1C              RBYTE   #38,#06,#02,#8E,#08,#60   note = 10, fre
0189 00AF   10              RBYTE   #08,#00,#02,#8E,#08,#60   REST
0190 00B5   1C              RBYTE   #38,#06,#02,#8E,#08,#60   note = 10, fre
0191 00BB   10              RBYTE   #08,#00,#02,#8E,#08,#60   REST
0192
0193 00C1   02              RBYTE   #40,#06,#01,#84,#09,#0D   note = 19, fre
0194 00C7   04              RBYTE   #20,#06,#01,#EA,#08,#AA   note = 15, fre
0195 00CD   04              RBYTE   #20,#06,#01,#B4,#08,#D6   note = 17, fre
0196 00D3   04              RBYTE   #20,#06,#01,#84,#09,#0D   note = 19, fre
0197 00D9   04              RBYTE   #20,#06,#01,#5A,#09,#51   note = 21, fre
0198
0199 00DF   02              RBYTE   #40,#06,#01,#46,#09,#7A   note = 22, fre
0200 00E5   1C              RBYTE   #38,#06,#02,#8E,#08,#60   note = 10, fre
```

```
0201 00EB   10                RBYTE  #08,#00,#02,#8E,#08,#60  REST
0202 00F1   1C                RBYTE  #38,#06,#02,#8E,#08,#60  note = 10, fre
0203 00F7   10                RBYTE  #08,#00,#02,#8E,#08,#60  REST
0204
0205 00FD   02                RBYTE  #40,#06,#01,#EA,#08,#AA  note = 15, fre
0206 0103   04                RBYTE  #20,#06,#01,#B4,#08,#D6  note = 17, fre
0207 0109   04                RBYTE  #20,#06,#01,#EA,#08,#AA  note = 15, fre
0208 010F   04                RBYTE  #20,#06,#02,#06,#08,#99  note = 14, fre
0209 0115   04                RBYTE  #20,#06,#02,#46,#08,#79  note = 12, fre
0210
0211 011B   02                RBYTE  #40,#06,#02,#06,#08,#99  note = 14, fre
0212 0121   04                RBYTE  #20,#06,#01,#EA,#08,#AA  note = 15, fre
0213 0127   04                RBYTE  #20,#06,#02,#06,#08,#99  note = 14, fre
0214 012D   04                RBYTE  #20,#06,#02,#46,#08,#79  note = 12, fre
0215 0133   04                RBYTE  #20,#06,#02,#8E,#08,#60  note = 10, fre
0216
0217 0139   02                RBYTE  #40,#06,#02,#B4,#08,#56  note = 9, freq
0218 013F   04                RBYTE  #20,#06,#02,#8E,#08,#60  note = 10, fre
0219 0145   04                RBYTE  #20,#06,#02,#46,#08,#79  note = 12, fre
0220 014B   04                RBYTE  #20,#06,#02,#06,#08,#99  note = 14, fre
0221 0151   04                RBYTE  #20,#06,#02,#8E,#08,#60  note = 10, fre
0222
0223              ***
0224
0225 0157   02                RBYTE  #40,#06,#02,#06,#08,#99  note = 14, fre
0226 015D   1C                RBYTE  #38,#06,#02,#46,#08,#79  note = 12, fre
0227 0163   10                RBYTE  #08,#00,#02,#46,#08,#79  REST
0228 0169   1C                RBYTE  #38,#06,#02,#46,#08,#79  note = 12, fre
0229 016F   10                RBYTE  #08,#00,#02,#46,#08,#79  REST
0230
0231 0175   02                RBYTE  #40,#06,#01,#B4,#08,#D6  note = 17, fre
0232 017B   04                RBYTE  #20,#06,#02,#8E,#08,#60  note = 10, fre
0233 0181   04                RBYTE  #20,#06,#02,#46,#08,#79  note = 12, fre
0234 0187   04                RBYTE  #20,#06,#02,#06,#08,#99  note = 14, fre
0235 018D   04                RBYTE  #20,#06,#01,#EA,#08,#AA  note = 15, fre
0236
0237 0193   02                RBYTE  #40,#06,#01,#B4,#08,#D6  note = 17, fre
0238 0199   1C                RBYTE  #38,#06,#02,#8E,#08,#60  note = 10, fre
0239 019F   10                RBYTE  #08,#00,#02,#8E,#08,#60  REST
0240 01A5   1C                RBYTE  #38,#06,#02,#8E,#08,#60  note = 10, fre
0241 01AB   10                RBYTE  #08,#00,#02,#8E,#08,#60  REST
0242
0243 01B1   02                RBYTE  #40,#06,#01,#84,#09,#0D  note = 19, fre
0244 01B7   04                RBYTE  #20,#06,#01,#EA,#08,#AA  note = 15, fre
0245 01BD   04                RBYTE  #20,#06,#01,#B4,#08,#D6  note = 17, fre
0246 01C3   04                RBYTE  #20,#06,#01,#84,#09,#0D  note = 19, fre
0247 01C9   04                RBYTE  #20,#06,#01,#5A,#09,#51  note = 21, fre
0248
```

TEXAS INSTRUMENTS

Printed in U.S.A.
0393–IP–5