TSP50C4X Family Speech Synthesizers Design Manual

TEXAS INSTRUMENTS

I

Contents

Contents (Continued)

Contents (Continued)

Contents (Continued)

Contents (Continued)

Contents (Concluded)

List of Illustrations

List of Tables

1 Introduction

The TSP50C4X family of speech synthesizers consists of the following four devices: TSP50C41, TSP50C42, TSP50C43, and TSP50C44. In each of these, an 8-bit microprocessor, a programmable speech synthesizer, and ROM are combined to provide a one-chip solution for many applications. The devices use Linear Predictive Coding (LPC) to generate speech at a low data rate. Mask options are also available to provide design flexibility.

This section consists of a brief overview of the TSP50C4X family. It begins with a summary of applications, key features, and a comparison of the devices, followed by a discussion on mask options and pin descriptions. Also included is an introduction to Linear Predictive Coding.

1.1 Applications

As illustrated in Figure 1-1, the TSP50C4X devices are versatile and can be used in many applications.

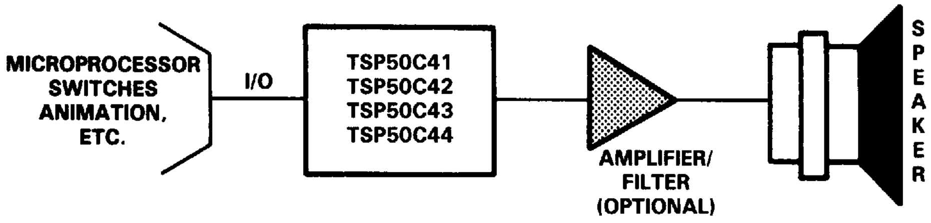


Figure 1-1. TSP50C4X Applications

Typical applications include:

Telecom	**Automotive**
PABX	Clock Systems
Telephone Management	Warning Systems
Security	**Consumer**
Home Monitors	Appliances
Navigation Aids	Mailboxes
Computer	Toys
Analyzers	**Medical**
Office Computers	Equipment for
Personal Computers	the Handicapped
Industrial	**Educational**
Inspection Controls	Learning Aids
Inventory Controls	Computer Aided
Machine Controls	Instructions
Warehouse Systems	

1.2 Description

The TSP50C4X device can be divided into several functional blocks (Figure 1-2). The two main blocks are the microcomputer and the speech synthesizer, which share RAM and timing circuits.

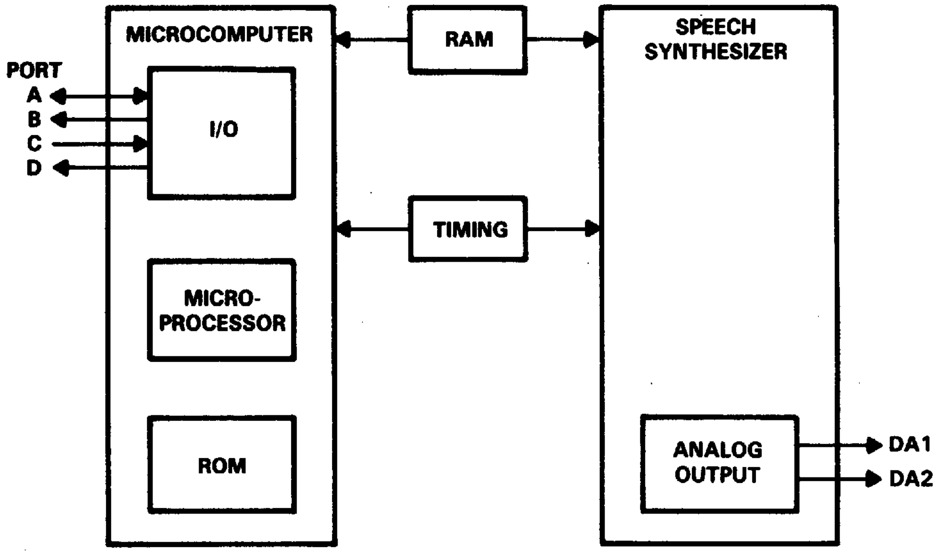


Figure 1-2. Block Diagram

These devices implement an LPC-10 speech synthesis algorithm using a 10-pole lattice filter. The internal microprocessor accesses speech data from the internal or external ROM (TSP60CXX), decodes the speech data and sends the decoded data to the synthesizer. The output of the synthesizer can be used to drive a small speaker directly or, with an external filter and amplifier, to drive a large speaker.

1.3 Features

- Programmable LPC-10 Speech Synthesizer
- 8-Bit Microprocessor with 61 Instructions
- 128 Bytes plus 16 Nibbles of RAM
- 4-V to 6-V CMOS Technology for Low Power Dissipation
- High-Efficiency Push-Pull Pulse-Width-Modulated Digital-to-Analog Output that can Drive a Speaker Directly
- 10-kHz or 8-kHz Speech Sample Rate
- 8K Byte or 16K Byte ROM, 21- or 33-pin I/O
- Mask Options
- External Event Counter/Internal Timer

1.4 Device Comparison

Table 1-1. TSP50C4X Device Comparison

	TSP50C41	TSP50C42	TSP50C43	TSP50C44
ROM (Bytes)	8K	8K	16K	16K
I/O pins*	21	33	21	33
8-bit ports	2 1/2	4	2 1/2	4
No. of pins	28	40	28	40

*I/O pins include the IRT pin.

1.5 Mask Options

The designer may choose from five basic mask options depending on the application. For instance, the master option is designed for single-chip applications in which the host is the internal microprocessor. The slave option is intended for use in multichip systems in which the host microprocessor is external as shown in Section 7. The mask options are as follows:

1. MASTER or SLAVE option
   a. MASTER option
      Port A (PA1-PA8) is a general purpose input/output port.
   b. SLAVE option
      Port A can be controlled by an external processor.
      Port C (PC0-PC3) pins are programmed to be interface control pins $\overline{\text{RDY}}$, $\overline{\text{ENA1}}$, $\overline{\text{ENA2}}$ and R/$\overline{\text{W}}$.

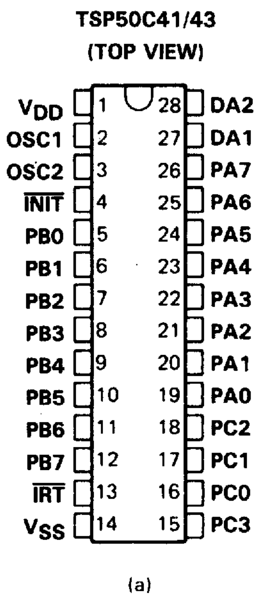
2. $\overline{\text{IRT}}$ INPUT or OUTPUT option
   a. $\overline{\text{IRT}}$ INPUT option
      $\overline{\text{IRT}}$ is an input that can be software selected by a TTMA command to be a clock signal for the timer prescale register. The $\overline{\text{IRT}}$ pin is unused if the internal clock is selected by a RSECT software command.
   b. $\overline{\text{IRT}}$ OUTPUT option
      $\overline{\text{IRT}}$ is an output that indicates that the data output on port A is stable.
3. KEYBOARD or NORMAL option
   a. KEYBOARD option
      Port A is split so that PA0-PA3 will be output pins and PA4-PA7 will be input pins. PC0 is not used and PC1-PC3 are tied low. This is referred to as the keyboard scan option since it is optimally configured for scanning a 4 × 4 keyboard.
   b. NORMAL option
      Port A is configured as an 8-bit I/O port.
4. ROM 8K or ROM 4K option
   a. ROM 8K option
      Allows the microprocessor software program to use 8K bytes of internal ROM for program instructions. Remaining ROM is available for other uses. Branches and calls must have even destination addresses (LSB = 0).
   b. ROM 4K option
      Limits the microprocessor program to the first 4K bytes of internal ROM for program instructions. Remaining ROM is available.
5. SETOFF DISABLED or ENABLED option
   a. SETOFF DISABLED
      Disables the software "Setoff" command and causes it to act as a "NOP".
   b. SETOFF ENABLED
      Enables the "Setoff" command. The microprocessor puts the TSP50C4X device in the low-power standby by executing the "Setoff" command. The external circuitry takes the chip out of the standby option by driving the $\overline{\text{INIT}}$ pin to a low state and then back to a high state.

When the master option is selected, the NORMAL and $\overline{\text{IRT}}$ input options are pre-selected. When the slave option is selected, all of the remaining options are available.

The TSP50C4X devices have additional I/O mask options to minimize the system parts count. Each pin on Ports A and C can be individually programmed to have a pull-up resistor. Ports B and D can be programmed in blocks of 4 to have open-drain outputs, that is, the pull-up device can be disabled. The blocks are B0-3, B4-7, D0-3, and D4-7.

1.6 Pin Assignment and Description

Figure 1-3 shows the pin assignments for the TSP50C41/43 and the TSP50C42/44. Tables 1-1 and 1-2 provide pin function descriptions.



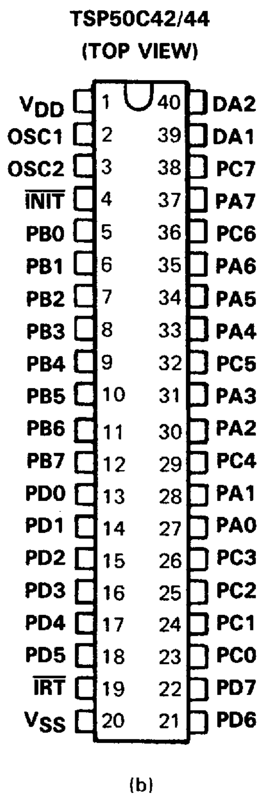


Figure 1-3. Pin Assignments

Table 1-2. Pin Function Description of Port A for Three Mask Options

PIN NAME	PIN NO. '50C41 '50C43	PIN NO. '50C42 '50C44	I/O	DESCRIPTION
[MASTER option]				Port A is a general purpose bi-directional port that is controlled by the internal microprocessor.
PA0 (LSB)	19	27	I/O	
PA1	20	28	I/O	
PA2	21	30	I/O	
PA3	22	31	I/O	
PA4	23	33	I/O	
PA5	24	34	I/O	
PA6	25	35	I/O	
PA7 (MSB)	26	37	I/O	
[SLAVE/NORMAL option]				Port A is an interface between the internal and external microprocessor. PC0-PC3 are configured as Ready. Enable and Read/$\overline{\text{Write}}$ control pins for interface.
PA0 (LSB)	19	27	I/O	
PA1	20	28	I/O	
PA2	21	30	I/O	
PA3	22	31	I/O	
PA4	23	33	I/O	
PA5	24	34	I/O	
PA6	25	35	I/O	
PA7 (MSB)	26	37	I/O	
[SLAVE/KEYBOARD opt]				Port A is configured so that PA0-PA3 are outputs and pins PA4-PA7 are inputs. This configuration is optimal for scanning a 4×4 keyboard. The $\overline{\text{RDY}}$ signal is not used. The $\overline{\text{ENA}}$1, $\overline{\text{ENA}}$2 and R/$\overline{\text{W}}$ should be tied low.
PA0 (LSB)	19	27	O	
PA1	20	28	O	
PA2	21	30	O	
PA3	22	31	O	
PA4	23	33	I	
PA5	24	34	I	
PA6	25	35	I	
PA7 (MSB)	26	37	I	

Table 1-3. Pin Function Description of Port B for External and Internal ROM Modes

PIN NAME	PIN NO. '50C41 '50C43	PIN NO. '50C42 '50C44	I/O	DESCRIPTION
[INTERNAL ROM mode]				The INTERNAL ROM mode is initiated by the INTRM software command. Port B is an output port controlled by the internal micro-processor. This port is put into the INTERNAL ROM mode on power-up and when the $\overline{\text{INIT}}$ pin is low. These two events also cause the port's outputs to latch low.
PB0 (LSB)	5	5	O	
PB1	6	6	O	
PB2	7	7	O	
PB3	8	8	O	
PB4	9	9	O	
PB5	10	10	O	
PB6	11	11	O	
PB7 (MSB)	12	12	O	
[EXTERNAL ROM mode]				The EXTERNAL ROM mode is initiated by a EXTRM software command. Port B is configured as an interface to a TSP60CXX vocabulary ROM.
M0	5	5	O	Vocabulary ROM mode control
M1	6	6	O	Vocabulary ROM mode control
ADD1	7	7	O	Vocabulary ROM address weight 1
ADD2	8	8	O	Vocabulary ROM address weight 2
ADD4	9	9	O	Vocabulary ROM address weight 4
ADD8	10	10	O	Vocabulary ROM address weight 8
ROMCLK	11	11	O	Clock output to the vocabulary ROM. Oscillator divided by 16.
RDIN	12	12	I	Vocabulary ROM data input

Table 1-4. Pin Function Description of Port C for Two Mask Options

PIN NAME	PIN NO.		I/O	DESCRIPTION
	'50C41 '50C43	'50C42 '50C44		
[SLAVE option] $\overline{\text{RDY}}$	15	23	O	When active (low), Port A is ready to receive data fron an external microprocessor. $\overline{\text{RDY}}$ is set high when the $\overline{\text{ENA}}$2 pin is pulled low. If the external processor is not holding $\overline{\text{ENA}}$2 low, then an RSRDY software command will reset $\overline{\text{RDY}}$ low. Status of the pin can be evaluated by the TPCA* instruction.
$\overline{\text{ENA}}$1	16	24	I	Enables the reading or writing of Port A data PA0-PA7
$\overline{\text{ENA}}$2	17	25	I	Read mode (R/$\overline{\text{W}}$ high) $\overline{\text{ENA}}$1: Most significant nibble of Port A latch is put on the bus PA4-PA7 while $\overline{\text{ENA}}$1 is low. When $\overline{\text{ENA}}$1 goes low, $\overline{\text{IRT}}$ goes high. $\overline{\text{ENA}}$2: Least significant nibble of Port A latch is put on the bus PA0-PA3 while $\overline{\text{ENA}}$2 is low. Write mode (R/$\overline{\text{W}}$ low) $\overline{\text{ENA}}$1: Most significant nibble on the data bus PA4-PA7 is strobed in the Port A latch when $\overline{\text{ENA}}$1 goes from low to high. $\overline{\text{ENA}}$2: Least significant nibble on the data bus PA0-PA3 is strobed in the Port A latch when $\overline{\text{ENA}}$2 goes from low to high.
R/$\overline{\text{W}}$	18	26	I	Determines the direction of the Port A data bus: R/$\overline{\text{W}}$ = high; data in the Port A latch is available to the external bus. R/$\overline{\text{W}}$ = low; data on the external bus is written into the Port A latch.
[MASTER option]				General-purpose input
PC0 (LSB)	16	23	I	
PC1	17	24	I	
PC2	18	25	I	
PC3	15	26	I	
[MASTER or SLAVE]				General-purpose input
PC4		29	I	
PC5		32	I	
PC6		36	I	
PC7 (MSB)		38	I	

Note: If an external driving circuit is used, it should not be allowed to go into high impedance.
*Refer to Table 5-1 for more information.

Table 1-5. Pin Function Description for Port D

PIN NAME	PIN NO.		I/O	DESCRIPTION
	'50C41 '50C43	'50C42 '50C44		
PD0 (LSB)		13	O	General-purpose output port
PD1		14	O	
PD2		15	O	
PD3		16	O	
PD4		17	O	
PD5		18	O	
PD6		21	O	
PD7 (MSB)		22	O	

Table 1-6. Pin Function Description of $\overline{\text{IRT}}$ (several options), $\overline{\text{INIT}}$, OSC, and DA

PIN NAME	PIN NO.		I/O	DESCRIPTION
	'50C41 '50C43	'50C42 '50C44		
[MASTER/$\overline{\text{IRT}}$ IN OPTION] $\overline{\text{IRT}}$	13	19	I	Interrupt input when programmed by a TTMA software command to be an input to the timer prescale register.
[SLAVE/$\overline{\text{IRT}}$ IN OPTION] $\overline{\text{IRT}}$	13	19	I	Interrupt input when programmed by a TTMA software command to be an input to the timer prescale register.
[SLAVE/$\overline{\text{IRT}}$ OUT OPTION] $\overline{\text{IRT}}$	13	19	O	Ready for data output. $\overline{\text{IRT}}$ goes high when $\overline{\text{ENA}}$1 is pulled low by external processor while pin R/$\overline{\text{W}}$ is high. $\overline{\text{IRT}}$ goes low when data are put into Port A with the TAPA* instruction. Software command TPCA can be used to read the data on the $\overline{\text{IRT}}$ pin.
DA1	27	39	O	Positive digital-to-analog converter output (PWM)
DA2	28	40	O	Negative digital-to-analog converter output (PWM)
$\overline{\text{INIT}}$	4	4	I	Initialize input; when low, device is initialized and goes into the low-power mode, Port B and Port D outputs are latched low. Port A is put into input mode. When $\overline{\text{INIT}}$ goes from low to high, the program counter is loaded with zeroes.
OSC1	2	2	I	Clock input. Crystal or ceramic resonator between OSC1 and OSC2: 3.07-MHz crystal/ceramic resonator for 8-kHz sampling rate 3.84-MHz crystal/ceramic resonator for 10-kHz sampling rate
OSC2	3	3	O	Clock return
V_{DD}	1	1	I	5-V nominal supply voltage
V_{SS}	14	20	I	Ground

*Refer to Table 5-1 for more information.

1.7 Introduction to LPC

The LPC-10 system uses a mathematical model of the human vocal tract to enable efficient digital storage and the recreation of realistic speech. To understand LPC (Linear Predictive Coding), it is essential to understand how the vocal tract works . This introduction, therefore, begins with a short description of the vocal tract. The LPC model and data compression techniques are then 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.

1.7.1 The Vocal Tract

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

The 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, higher pressure for louder speech.

As air flows through the vocal tract, it makes very 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 are produced by this type of restriction, for example, the ''A'' sound. This is called ''voiced'' speech.

A different type of restriction takes place in the mouth and causes a hissing sound called white noise. The ''S'' sound is a good example. This 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 which cause turbulence that produces white noise. This 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 really noise, not speech. The shape of the oral cavity changes noise into recognizable speech. The position of the tongue, the lips and the jaws change the resonance of the vocal tract, shaping the raw noise of restricted air flow into understandable sounds.

1.7.2 The LPC Model

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

Figure 1-4 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 TSP50C4X family, this filter has ten poles, so the synthesis is referred to as LPC-10.

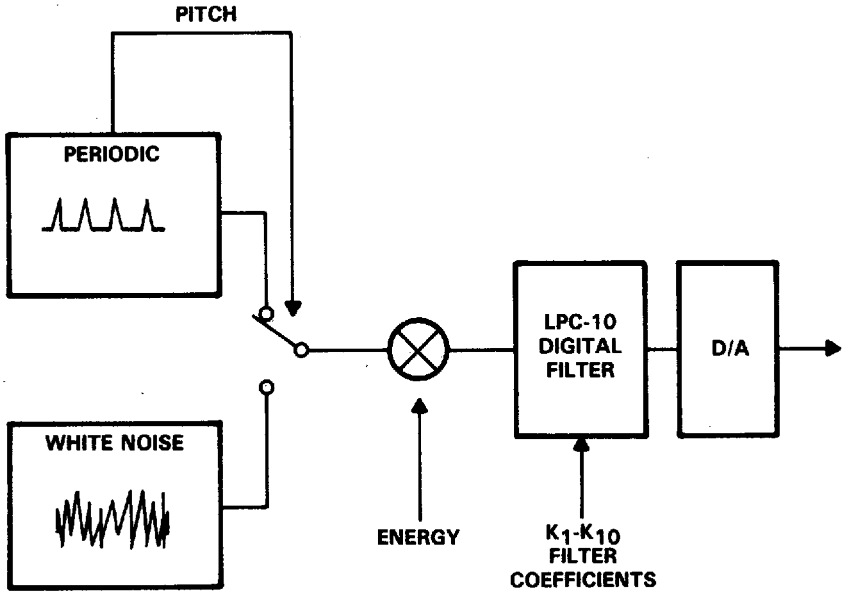


Figure 1-4. LPC-10 Vocal Tract Model

1.7.3 LPC Data Compression

The data compression for LPC-10 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 that are generally from 10 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 there isn't an abrupt transition at the end of each frame. Often speech changes even more slowly than the frame. TI's LPC model allows for a repeat frame, where 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 TSP50C4X Family Architecture

The major components of the TSP50C4X devices are a speech synthesizer, an 8-bit microprocessor, an internal 8K-byte (TSP50C41/42) or 16K-byte (TSP50C43/44) ROM and interface logic (I/O) as shown in Figure 2-1. Instructions are fetched by the microprocessor from the ROM approximately every 9 μs (oscillator frequency divided by 32) and are used to control the algorithm sequences. To generate speech, the processor accesses speech data from either the internal 8K-byte ROM or an external speech ROM. 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 I/O consists of one 8-bit bidirectional port (Port A), two 8-bit output ports (Port B and Port D), one 8-bit input port (Port C) and an $\overline{\text{IRT}}$ pin. These ports are under the control of the microprocessor and are configured by mask options.

The synthesizer shares access to the RAM and addresses the individual parameter locations as needed when generating speech. The speech synthesizer performs parameter smoothing and pitch period control as well as lattice filter computations.

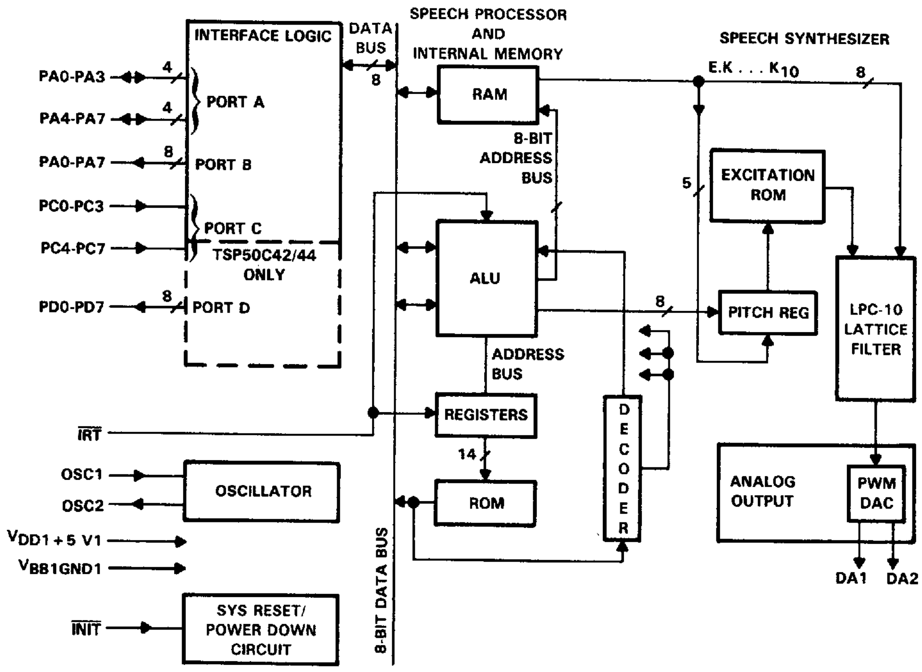
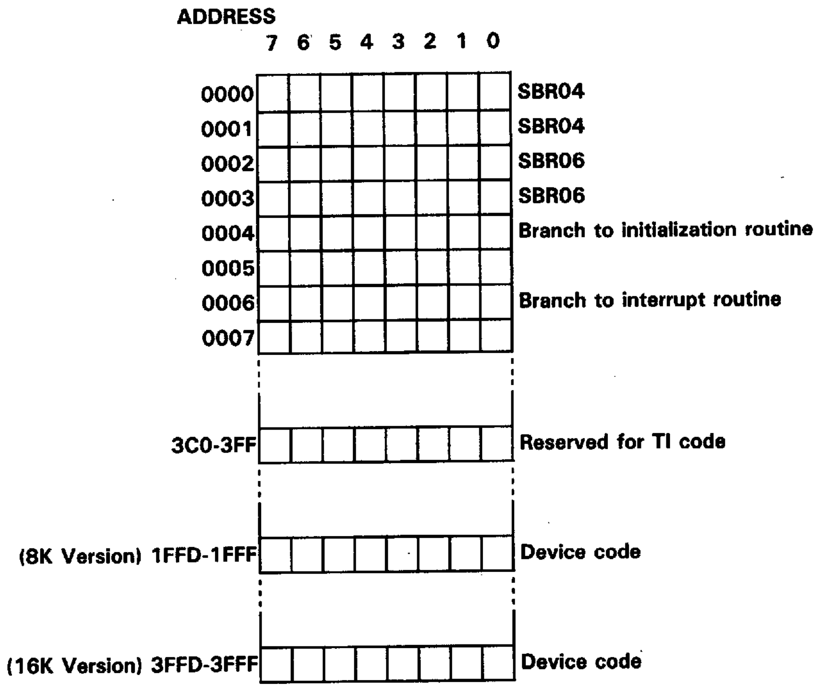


Figure 2-1. System Block Diagram

2.1 ROM

The ROM holds the control program, the speech data, and any other data required by the application. Certain locations in the ROM are reserved for specific purposes (Figure 2-2).



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

Figure 2-2. ROM Map

The ROM may be accessed in three 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 anywhere in ROM to the A register. The GET counter is initialized by the LUSPS 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 ROM locations 0-3FF into the A register.

*Refer to Table 5-1 for more information.

2.2 Program Counter

The TSP50C4X devices are available with a 13-bit (TSP50C41/42) or 14-bit (TSP50C43/44) program counter. The program counter points to the next instruction to be executed. After the instruction is executed, it is normally incremented to point to the next instruction. Several instructions are used to change the value of the program counter. These are:

BR — branch
SBR — short branch
CALL — call subroutine
RETN — return from subroutine
RETI — return from interrupts

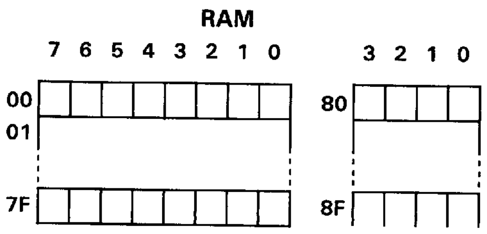
2.3 Program Counter Stack

The program counter stack has five levels. When a subroutine is called or an interrupt occurs, the contents of the program counter are pushed onto the stack. When a RETN (return from subroutine) or an RETI (return from interrupt) is executed, the contents of the top stack location are popped into the program counter. Certain instructions (LUSPS, GET, LUAA) push the contents of the program counter onto the stack and then pop it back during their execution. The POP* instruction may be used to pop the top stack location.

2.4 Random Access Memory (RAM)

The RAM has 128 bytes plus 16 nibbles. Addresses 0 to 7F refer to bytes, and addresses 80 to 8F refer to nibbles. RAM locations 0 to 18 and 80 to 8F are used for communication with the synthesizer when speech is being generated. When not executing speech, the entire memory may be used for algorithm data storage.

*Refer to Table 5-1 for more information.



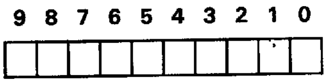
2.5 Arithmetic Logic Unit (ALU)

The ALU performs simple arithmetic, comparison, and logical functions for the central processor. The ALU is 10 bits in length and provides extra range for generating table look-up addresses. When transferring 8-bit data to the ALU, data is right justified. The input to the upper two bits may be either 0 (integer mode) or equal to the MSB of the 8-bit data (extended sign mode) depending on the set or reset condition of the mode latch (EXTSG and INTGR). All bit and comparison operations are performed on the lower 8 bits.

2.6 A Register

The A register or accumulator is the primary 10-bit register. Its contents can be transferred to or from ROM, RAM, and most of the other registers. It is used for arithmetic and logical operations. The contents are saved, in a dedicated storage register, during interrupts and restored by the RETI* instruction.

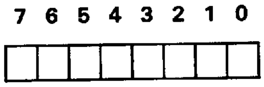
A Register



2.7 X Register

The X register is an 8-bit register used as a RAM index register. All RAM access instructions 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 interrupts.

X Register

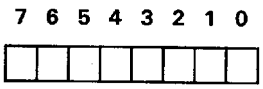


*Refer to Table 5-1 for more detail.

2.8 B Register

The 8-bit B register is used for temporary storage. It is especially helpful for storing a RAM address, since it can be exchanged with the X register using the XBX* instruction. The contents of the B register are not saved during interrupts.

B Register



2.9 Status Flag

The status flag is set or cleared by various instructions, depending on the result of the instruction. The BR, SBR, and CALL instructions are conditional. These instructions are executed only when the status flag is set. Refer to the individual descriptions of these instructions in Section 3 to find the status flag value.

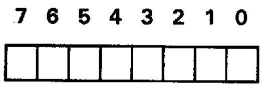
Status Flag

2.10 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 TXTM instruction and examined with the TTMA* instruction. When it decrements from #00 to #FF an interrupt request will be generated. If interrupts are enabled, an immediate interrupt will occur; if not, the interrupt request will remain pending until interrupts are enabled. The timer will not start counting down again until it is reloaded by the TXTM instruction.

The timer register must be loaded with a fixed #1F (hex) during synthesis. It is used to generate interrupts for the synthesis software and as a time value for parameter interpolation.

Timer Register



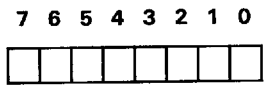
*Refer to Table 5-1 for more detail.

2.11 Timer Prescale Register

The 8-bit timer prescale register is used as a divider of the input to 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 the preset latch, and the counting starts again.

The timer prescale register clock comes from an internal clock or from an external source on the $\overline{\text{IRT}}$ pin. The internal clock runs at 1/48 the clock frequency of the chip. The TTMA* instruction makes the clock source external, and the RSECT selects the internal clock.

Timer Prescale Register

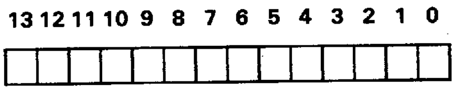


2.12 Pitch Register

The 8-bit pitch register is really a synthesizer register, but it is mentioned here because it is the only one loaded explicitly by the processor. When the START instruction is executed, the pitch register is loaded with the current value in the accumulator. After that, the pitch register is loaded from a RAM location. See Section 6 for a detailed explanation.

2.13 Speech Address Register

Speech Address Register



The speech address register is a 13-(TSP50C41/42) or 14-(TSP50C43/44) bit register that is used to point to data in the internal ROM. It is loaded with the TASH (Transfer Accumulator to Speech register High) and TASL (L is for Low) instructions. When a LUSPS or GET instruction is executed, the ROM value pointed to by the speech address register will be loaded into the parallel-to-serial register and the speech address register is incremented.

*Refer to Table 5-1 for more information.

2.14 Parallel-to-Serial Register

Parallel-to-Serial Register

7	6	5	4	3	2	1	0

The 8-bit parallel-to-serial register is used to unpack speech. It can be loaded with eight bits of data from an external TSP60CXX speech ROM or from the internal ROM pointed to by the speech address register. The LUSPS 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 will automatically be reloaded. The INTRM instruction selects the internal ROM as the source for the parallel-to-serial register, while EXTRM selects external ROM.

2.15 Interface Logic

The TSP50C4X interface consists of four 8-bit ports. Port A (PA) is a bidirectional port, Ports B and D (PB and PD) are output ports. Port B can also be used as an interface to an external TSP60CXX serial ROM. Port C (PC) is either a general 8-bit input port (master option) or is split into a 4-bit input port and a 4-bit control port for Port A (slave option). In addition, an interrupt ($\overline{\text{IRT}}$) pin and a hardware reset ($\overline{\text{INIT}}$) pin are provided. The remaining six pins are used for power supply, oscillator, and analog outputs. The choice of master or slave operation for Port A is a mask-generation option that depends on the type of product to which the device will be applied. The master option is designed for single-chip applications or for applications in which the host is the internal microprocessor. The slave option is intended for use in multichip systems in which the host microprocessor is external.

2.16 Port A (MASTER Option)

Port A is a bidirectional port. The direction (input or output) of the port is determined by software control. If a TAPA* instruction is executed, the contents of the lower eight bits of the accumulator are transfered to Port A, and it is used as an output port. If a TPAA* instruction is executed, Port A is used as an input port and its contents are transferred to the A register. The TPAM* instruction transfers the Port A values to the current RAM location.

If the port is switched from output to input mode with the TPAA or TPAM instructions, the data from the first transfer will be invalid. The instruction should be executed twice.

*Refer to Table 5-1 for more information.

2.17 Port A (SLAVE Option)

In the slave option, the transfer of data to and from the 8-bit Port A is controlled by an external host through four pins of Port C. In the slave mode, pins PC3-PC0 have the function of read/write control, high nibble strobe, low nibble strobe, and ready flag for handshake interfacing. The high and low nibble strobe arrangement permits simple interfacing to 4-bit as well as 8-bit microprocessors. The ready pin is set to a not ready by a low nibble write and reset by the RSRDY instruction to acknowledge that the data written to Port A has been read by the internal microprocessor.

2.18 Port B

Port B can be either a general 8-bit output port or a specialized external speech ROM port. The configuration of this port is controlled by the EXTRM and INTRM instructions. If the microprocessor executes the EXTRM command, then the port is configured as a ROM port and the data source for GET instructions will be Port B. If the microprocessor executes the INTRM instruction, then Port B will be configured as a general 8-bit output port and all speech data will source from the internal ROM memory.

If the TSP60CXX external ROM is enabled and the INTRM instruction is executed, there will be a bus conflict. To avoid this, access a nonexistent TSP60CXX device before going to the INTRM mode. This will turn off the TSP60CXX so that it will not conflict when Port B becomes all outputs.

At power-up, Port B is low.

2.19 Port C

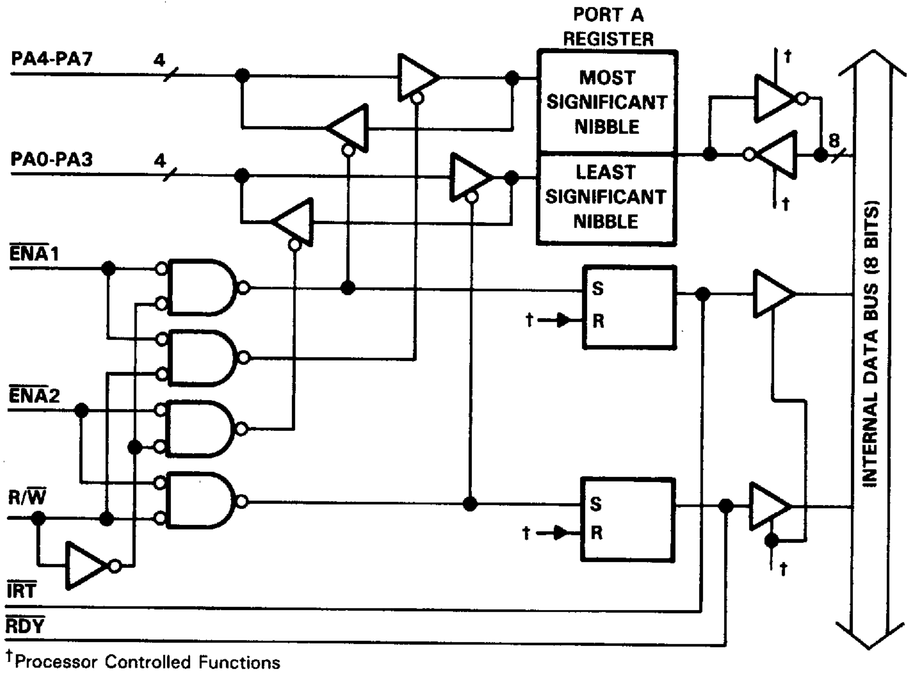
Port C has two possible configurations. If the master option is selected, Port C is a general input port. The data on the eight pins are transferred on command TPCA to the A Register.

In the slave option, the port is configured so that PC3-PC0 are used to control Port A through the functions ENA1, ENA2, R/W, and RDY. (See Applications, Section 6 for more details.)

2.20 Port D

Port D is a general output port. Data is transferred to this port from the internal microprocessor by executing the command TAPD. This is available only on the TSP50C42 and TSP50C44 devices.

At power-up, Port D is low.



†Processor Controlled Functions

Figure 2-3. I/O Data Bus (PA0-PA7)

2.21 $\overline{\text{IRT}}$ Pin

The interrupt pin is hardware configurable by mask option to be an event-counter input pin (IRT = input) or an interrupt output (slave option, IRT = output) . When selected as an event-counter input pin, a signal is used as the timer prescale register increment clock. The internal 80-kHz timing clock can be selected by executing RSECT. The external clock on the interrupt pin is selected by executing the command TTMA.

2.22 Speech Synthesizer

The task of generating synthetic speech is divided between the programmable microprocessor and the dedicated speech synthesizer.

The microprocessor controls speech synthesis by unpacking and decoding parameters as well as setting the update interval (frame rate). These aspects of speech tend to vary from application to application and are well suited to the microprocessor. The speech synthesizer, on the other hand, performs all of the synthetic speech functions that require intensive computations but do not change from application to application. These functions include the implementation of a 10-pole digital lattice filter, a pitch-controlled excitation generator, a parameter interpolator, 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 take place via a shared memory space in the microprocessor RAM. (Refer to the Applications section for more information.)

2.22.1 Use of RAM by the Synthesizer

The RAM consists of 1088 bits that are arranged as 128 8-bit words from address 00 to 7F and 16 4-bit words from 80-8F. The microprocessor can read or write to any word in the RAM. The synthesizer can only read from locations 00 to 17 and 80 to 8F, where the microprocessor stores the PRESENT and the NEW values for the frame parameters.

After the timer register generates an interrupt, the synthesizer will read only the PRESENT or both the PRESENT and NEW frame parameters. If interpolation is required, the INTE instruction is invoked for the current frame and the synthesizer uses both frame parameters. Otherwise, only the PRESENT parameter is used.

When interrupt occurs (see subsection 2.22.2), the context switch changes addresses for the PRESENT and NEW values. This is done so that the parameters put into NEW value RAM locations by the microprocessor become the PRESENT values for the current speech frame. This is a hardware function and it is transparent to the microprocessor.

If the INTE instruction is not invoked for the current frame, then interpolation will not be performed. The synthesizer will read the frame parameters for the PRESENT frame and put them into the LPC filter.

If the INTE instruction is invoked for the current frame, then the synthesizer will perform interpolation and the following sequence of events applies:

1. The interrupt will cause a context switch.
2. The microprocessor loads the next frame of data into the NEW value RAM location. The data for the current frame can be found in the PRESENT value RAM location.
3. The microprocessor invokes the INTE instruction, which will put the synthesizer into the interpolation mode.
4. Every pitch period the synthesizer will:
    a. Read the PRESENT and NEW value parameters.
    b. Read the timer register. This data defines the elapsed time from the start of the current frame (PRESENT data values) to the start of the next frame (NEW data values).
    c. The synthesizer uses the data from (a) and (b) to perform a straight line interpolation of the parameters for the current and next frame parameters.
    d. The computed parameters are put into the LPC filter.

ADDRESS		COMMENTS
00	NEW PITCH (11-4)	
01	OLD PITCH (11-4)	
02	NEW ENERGY (11-4)	
03	OLD ENERGY (11-4)	
04	NEW K1 (11-4)	
05	OLD K1 (11-4)	
06	NEW K2 (11-4)	8 MSBs (11-4) of both
07	OLD K2 (11-4)	"new" and "old"
08	NEW K3 (11-4)	speech parameters.
09	OLD K3 (11-4)	This area is reserved
0A	NEW K4 (11-4)	only during speech
•	•	generation. Context
•	•	address switch is
•	•	operative only for
14	NEW K9 (11-4)	speech.
15	OLD K9 (11-4)	
16	NEW K10 (11-4)	
17	OLD K10 (11-4)	
18		
•	•	General memory. No
•	•	context addressing mode.
7F		
80	NEW PITCH (3-0)	
81	OLD PITCH (3-0)	
82	NEW ENERGY (3-0)	
83	OLD ENERGY (3-0)	
84	NEW K1 (3-0)	4 LSBs of speech
85	OLD K1 (3-0)	parameters. K7-K10
86	NEW K2 (3-0)	do not have memory
87	OLD K2 (3-0)	assigned since 8 bit
88	NEW K3 (3-0)	values are sufficient.
89	OLD K3 (3-0)	When not generating
8A	NEW K4 (3-0)	speech the memory is
8B	OLD K4 (3-0)	available. Context
8C	NEW K5 (3-0)	addressing mode enabled
8D	OLD K5 (3-0)	during speech.
8E	NEW K6 (3-0)	
8F	OLD K6 (3-0)	
90	NOT AVAILABLE	Not available
•		
FF		

Figure 2-4. RAM Map During Speech Generation

2.22.2 Context Switch

The Context Switch is used to point to the parameter set just loaded as the NEW value and the previous set as the PRESENT value. Interpolation is then enabled between the two sets of parameter values. The INTE (Enable Timer Interrupt) instruction is used to control interpolation.

There are instances when interpolation should be disabled. The most common example is for voicing transitions or when going from zero to a nonzero value of energy. If no INTE instruction is executed, the Context Switch will change and interpolation will be disabled.

The context addressing mode is enabled for the dedicated speech data address space in RAM (addresses 00-17, 80-8F).

2.22.3 Interpolation

Interpolation takes place from the present values to the new values during the frame. If interpolation is not enabled, the present values are used for the entire frame. The programming task is made easier by the availability of the Context Switch.

2.22.4 Timing Requirements

Table 2-1. Initialization Timing

	CONDITION	MIN	MAX	UNIT
t_w	TSP50C4X in the standby mode due to a setoff command	10		ns
	$\overline{\text{INIT}}$ pulsed low while the TSP50C4X is active	*		

*One oscillator clock period.

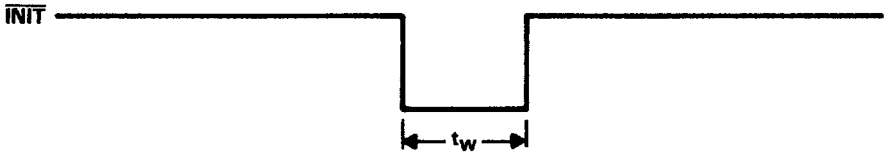


Figure 2-5. Initialization Timing

Table 2-2. Timing Requirements

	SAMPLE RATE		UNIT
	10 kHz NOM	8 kHz NOM	
Sample period	100	125	μs
ROM clock rate	240	192	kHz
ROM clock period	4.17	5.20	μs
Oscillator rate	3.84	3.07	MHz
Oscillator period	260	3.25	ns

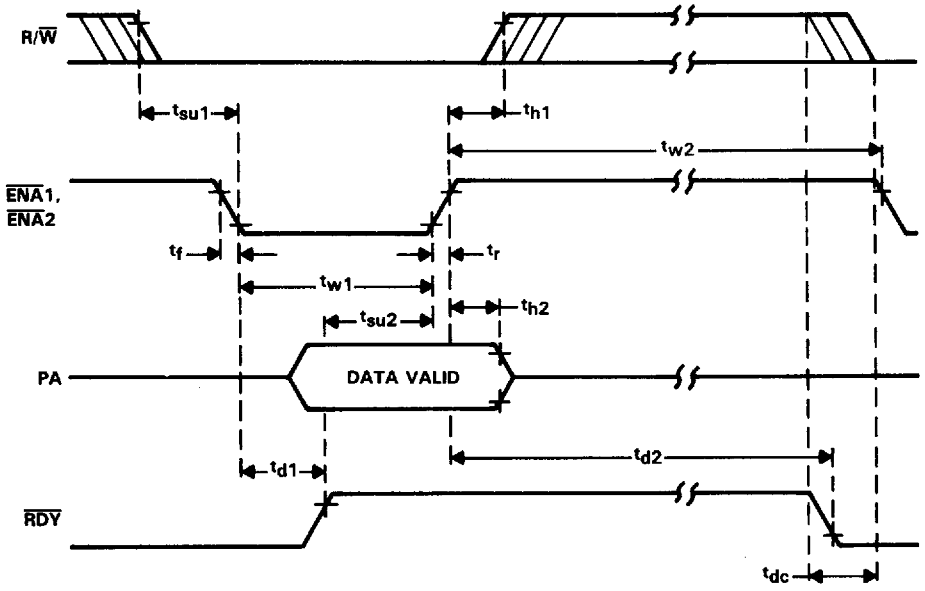


Figure 2-6. Write Timing Diagram

Table 2-3. Write Timing Requirements (see Figure 2-6)

PARAMETER		MIN	MAX	UNIT
t_{su1}	Setup time, R/W̄ before ENA1↓ or ENA2↓	80		ns
t_{su2}	Setup time, data valid before ENA1↑ or ENA2↑	100		ns
t_{h1}	Hold time, R/W̄ after ENA1↓ or ENA2↓	40		ns
t_{h2}	Hold time, data valid after ENA1↑ or ENA2↑	40		ns
t_{w1}	Pulse duration, ENA1 or ENA2 low	200		ns
t_{dc}	Cycle delay time	32		CLK cycles
t_r	Rise time, ENA1 or ENA2		50	ns
t_f	Fall time, ENA1 or ENA2		50	ns
t_{d1}	Delay time from ENA1 low or ENA2 low to RDY high		250	ns
t_{d2}	Delay time from ENA1 high or ENA2 high to RDY low	Program Dependent		

NOTE: ENA1 applies to PA4 through PA7, and ENA2 applies to PA0 through PA3.

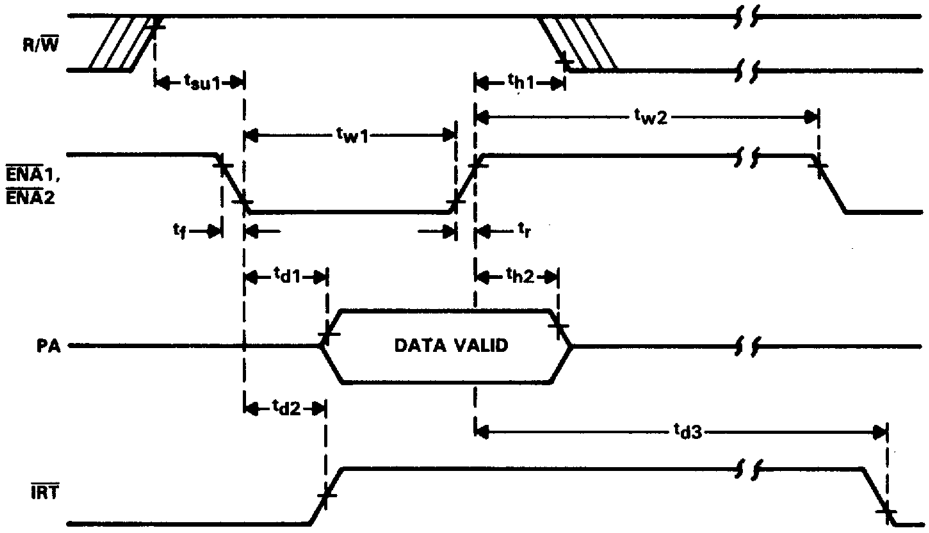


Figure 2-7. Read Timing Diagram

Table 2-4. Read Timing Requirements (see Figure 2-7)

PARAMETER		MIN	MAX	UNIT
t_{su1}	Setup time, R/W̄ before ENA1↓ or ENA2↓	80		ns
t_{h1}	Hold time, R/W̄ after ENA1↑ or ENA2↑	40		ns
t_{h2}	Hold time, data valid after ENA1↑ or ENA2↑	100		ns
t_{w1}	Pulse duration, ENA1 or ENA2 low	200		ns
t_{w2}	Pulse duration, ENA1 or ENA2 high	2		µs
t_r	Rise time, ENA1 or ENA2		50	ns
t_f	Fall time, ENA1 or ENA2		50	ns
t_{d1}	Delay time from ENA1 low or ENA2 low to data valid		250	ns
t_{d2}	Delay time from ENA1 low or ENA2 low to IRT high		250	ns
t_{d3}	Delay time from ENA1 high or ENA2 high to IRT low	Program Dependent		

NOTE: ENA1 applies to PA4 through PA7, and ENA2 applies to PA0 through PA3.

2.22.5 Voicing Control

Voicing transitions refer to the change in the excitation source from voiced to unvoiced or from unvoiced to voiced. (See section 1.7.1 for a definition of voiced speech). The voicing status of a frame is encoded into the speech data and must be decoded by the unpacking algorithm. The voicing status is conveyed to the synthesizer by executing the TAV instruction (Transfer A Register to Voicing Register). A "1" on the LSB of the A Register will cause voiced excitation to be used while a "0" will indicate unvoiced excitation. A change in value of the voicing register will take effect on the next frame boundary. The actual voicing change in the synthesizer is synchronized by both timer overflow (next frame boundary) and parameter interpolation. This synchronization is hardware-controlled and is transparent to software control.

2.22.6 Frame Length Control

All speech control algorithms must include some type of frame-length control. In order to obtain the proper frame length and also to insure proper operation of the parameter interpolation, the timer preset value is fixed to be 1F, and the prescale register preset value is variable and determines the actual frame length. The frame length in seconds is calculated by:

$$\text{TFL} = \frac{1536\ (\text{N}+1)}{\text{OSCILLATOR RATE}}$$

where N is the decimal value of the prescale preset value. The frame length in terms of speech samples is

$$\text{TFL} = 4 * (\text{N}+1).$$

It is important that one of the first statements of the speech interrupt routine is the timer register preset statement. For fixed-frame-length applications, the prescale register must be set only once at the beginning of speech. For variable-frame-length applications, the timer prescale register needs to be updated each frame as soon after the timer interrupt as possible.

2.22.7 Digital to Analog Converter and Output Buffer

The TSP50C4X devices contain an internal digital-to-analog converter (DAC) connected to the output of the synthesizer. The DAC has a pulse-width-modulated, push-pull output that drives the output buffers DA1 and DA2, which are capable of driving a low-power speaker directly (see Section 3).

3. Electrical Specifications

3.1 Absolute Maximum Ratings Over Operating Free-Air Temperature Range

Supply voltage, V_{DD}	−0.3 V to 7 V
Input voltage, V_I	−0.3 V to V_{DD} + 0.3 V
Output voltage, V_O	−0.3 V to V_{DD} + 0.3 V
Operating free-air temperature range	0 °C to 70 °C
Storage temperature range	−30 °C to 125 °C

All voltages are with respect to V_{SS}.

3.2 Recommended Operating Conditions — DC

PARAMETER	CONDITIONS		MIN	TYP[†]	MAX	UNIT
V_{DD}*			4	5	6	V
T_A	Operating free-air temperature		0		70	°C
V_{IH}	V_{DD} = 4 V		3		4	V
	V_{DD} = 5 V		3.8		5	
	V_{DD} = 6 V		4.5		6	
V_{IL}	V_{DD} = 4 V				1	V
	V_{DD} = 5 V				1.2	
	V_{DD} = 6 V				1.5	
V_L	V_{DD} = 5 V,	R_L = 50 Ω	1.9	2.8		V
	V_{DD} = 5 V,	R_L = 100 Ω	2.9	3.6		
	V_{DD} = 4 V,	R_L = 50 Ω	1.3	2		
	V_{DD} = 4 V,	R_L = 100 Ω	2	2.7		
Output power D/A	V_{DD} = 5 V,	R_L = 50 Ω	72	157		mW
	V_{DD} = 5 V,	R_L = 100 Ω	84	130		
	V_{DD} = 4 V,	R_L = 50 Ω	34	80		
	V_{DD} = 4 V,	R_L = 100 Ω	40	73		
Pullup Resistance	V_{DD} = 5 (when programmed)		25	50	100	kΩ

*Unless otherwise noted, all voltages are with respect to V_{SS}.

3.3 Recommended Operating Conditions — AC

PARAMETER	CONDITIONS	MIN	TYP[†]	MAX	UNIT
t_r	V_{DD} = 5 V, PA,B,D into 100 pF 10% to 90%			150	ns
t_f	V_{DD} = 5 V, PA,B,D into 100 pF 10% to 90%			100	ns
f_{osc}	Speech Sample Rate = 10 kHz		3.84		MHz
	Speech Sample Rate = 8 kHz		3.07		

[†]All typical values are at V_{DD} = 5 V and T_A = 25 °C.

3.4 Electrical Characteristics Over Recommended Operating Free-Air Temperature Range

PARAMETER	CONDITIONS	MIN	TYP[†]	MAX	UNIT
I_{CC}	V_{DD} = 5 V Standby mode = SETOFF executed or INIT high, no pullup resistor on INIT, all port pins are open.		10	50	μA
	V_{DD} = 5 V Operating mode = INIT high and SETOFF not executed, DA pins are open.		1.5	3	mA
V_{OH}	V_{DD} = 5 V, I_{OH} = 0.3 mA	4.7	4.85		V
	I_{OH} = 1.2 mA	4	4.5		
V_{OL}	V_{DD} = 5 V, I_{OL} = 1.7 mA		0.3	0.4	V
I_I	Input current			5.0	μA
I_{OH}	V_{DD} = 4 V, V_{OH} = 3.5 V	0.3	0.8		mA
	V_{DD} = 5 V, V_{OH} = 4.5 V	0.6	1.2		
	V_{DD} = 6 V, V_{OH} = 5.5 V	0.8	1.5		
I_{OL}	V_{DD} = 4 V, V_{OL} = 0.4 V	1.2	1.8		mA
	V_{DD} = 5 V, V_{OL} = 0.4 V	1.7	2.4		
	V_{DD} = 6 V, V_{OL} = 0.4 V	2	2.8		
r_o	V_{DD} = 5 V, DA1 and DA2 pins		50		Ω

[†]All typical values are at T_A = 25°C.

For details on Timing, see Section 2.

3.5 Oscillator

The oscillator pins OSC1 and OSC2 are provided for either a crystal or ceramic resonator connection in the typical phase-shift oscillator connection. The recommended value for circuit components C1 and C2 are shown.

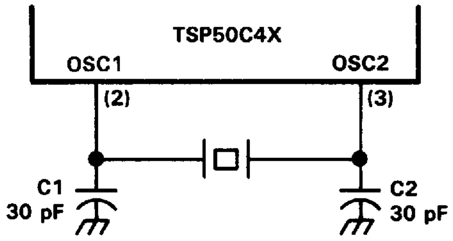


Figure 3-1. Typical Phase-Shift Oscillator Connection

3.6 Direct Speaker Driver

The analog buffers at DA1 and DA2 are designed to directly drive a 50- to 100-Ω speaker with approximately 120 to 150 mW of peak power. Average power is considerably below this figure. The reduction in power is caused by the nature of speech. The effective analog output impedance at 5 V is typically 50 Ω for output currents less than 60 mA. For output currents more than 60 mA, the DAC buffers act as current sources. The outputs can also be used to drive transistors or operational amplifiers.

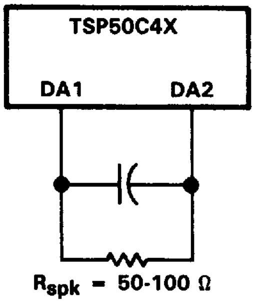


Figure 3-2. Typical Direct Speaker Drive Connection

4. TSP50C4X Assembler

TSP50C4X Assembly Language instructions are mnemonics that correspond directly to binary machine instruction codes. An assembly language program (source program) must be converted to a machine language program (object program) by a process called assembling before a computer can execute it. Assembling converts the mnemonics to binary values and associates those values with binary addresses, creating machine language instructions. Assembler directives control this process, place data in the object program, and assign values to the symbols used in the object program.

TSP50C4X directives are of four kinds:

- Directives that affect the location counters
- Directives that affect assembler output
- Directives that initialize constants
- Directives that copy source files and end programs.

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⟩]
```

4.1 Source Statement Format

An assembly language source program consists of statements contained in the assembly source file(s) that may contain assembler directives, machine instructions, or comments. Source statements may have 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, but the assembler will truncate the source line at 60 characters without warning. 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. It is printed in the assembly listing but has no other 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.

4.1.1 Label Field

The label field begins in character position 1 of the source line. If position 1 is a character other than 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 six alphabetic (a..z,A..Z), numeric (0..9) and special (@,$,_) characters. The first character should be alphabetic. The remaining five characters may be any of the others mentioned above.

4.1.2 Command Field

The command field begins after the blank that terminates the label field, or the first nonblank character after the first position (which is blank when the label is omitted). The command field is terminated by one or more blanks and may not extend past the character position 60. 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 identical to the assembler.

4.1.3 Operand Field

The operand field begins following the blank that terminates the command field and may not extend past character position 60. The operand may contain one or more of the constants or expressions described below. Terms in the operand field are separated by commas. The operand field is terminated by the first blank encountered.

4.1.4 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 characters including blanks.

4.2 Constants

The assembler recognizes the following five types of constants:

Decimal integer
Binary integer
Hexadecimal integer
Character
Assembly-time

4.2.1 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. Negative numbers are given their two's complement representation.

The following are valid decimal constants:

1000	Constant equal to 1000 or #03E8
−32768	Constant equal to −32768 or #8000
25	Constant equal to 25 or #0019

4.2.2 Binary Integer Constants

A binary integer constant is written as a string of up to sixteen binary digits preceded by a question mark ("?"). If less than sixteen digits are specified, the assembler will right justify 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

4.2.3 Hexadecimal Integer Constants

A hexadecimal integer constant is written as a string of up to four hexadecimal digits preceded by a pound sign '#' or a greater than sign '>'. If less than four hexadecimal digits are specified, the assembler will right justify 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)

4.2.4 Character Constants

A character constant is written as a string of one or two 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 will right justify 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

4.2.5 Assembly-Time Constants

An assembly-time constant is a symbol given a value that appears in the label field of a statement. The value of the symbol is determined at assembly time and may be assigned by expressions using any of the above constant types.

4.3 Symbols

Symbols are used in the label and the operand fields. A symbol is a string of alphanumeric characters: 'a' through 'z', 'A' through 'Z', '0' through '9', and special characters '@', '_', and '$'. Upper-case and lower-case characters are not distinguished from one another. The first character in a symbol must not be a number or a '$'. No character may be blank. When more than six 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 six 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 source by appearing in the label field of a statement.

The following are examples of valid symbols:

```
START
Start
strt_1
```

4.3.1 Predefined Symbol '$'

The dollar sign '$' is a predefined symbol given the value of the current location within the program. This 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.

4.3.2 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'''
```

4.4 Expressions

Expressions are used in the operand field 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 may 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.

4.4.1 Arithmetic Operators in Expressions

The arithmetic operators that can be used in an expression are as follows:

+	for addition
−	for subtraction
*	for multiplication
/	for 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 other than unary plus or unary minus (so that the expression 4+5*2 would be evaluated as 18, not 14).

4.4.2 Parentheses in Expressions

The assembler supports the use of parentheses in expressions to alter the order of evaluation of the expression. Nesting of pairs of parentheses within expressions is also supported. When parentheses are used, first the expression in the innermost pair is processed, then the expression within the next inner pair is evaluated, and so on. After the evaluation of the expressions within all the parentheses is finished, the rest is completed from left to right. Evaluation of the expressions within parentheses at the same nesting level is simultaneous. Parenthetical expressions may not be nested more than eight deep.

4.5 Invoking the Assembler

The assembler is invoked by typing:

```
ASM5C ^ [ –⟨options⟩] ^ ⟨source[.ext]⟩
```

where:

'Options' represents a list of assembler options (see Section 4.6). 'Source' stands for the name of the source file with the optional extension. If the extension is not given, then the default extension of '.ASM' is assumed.

For example:

```
ASM5C  –l PROGRAM
```

The assembler uses the source file PROGRAM.ASM and generates the output object file PROGRAM.MPO. No list file is generated.

4.5.1 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.

4.5.2 Assembly Source File

The assembly source file is specified in the command line. If no extension is given, then '.ASM' is assumed.

For example:

```
ASM5C PROGRAM.SRC
```

Uses the file PROGRAM.SRC as the Assembly source file.

```
ASM5C PROGRAM
```

Uses the file PROGRAM.ASM as the Assembly source file.

4.5.3 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 will be .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:

```
ASM5C PROGRAM.SRC
```

Uses file PROGRAM.SRC as the Assembly source file and the file PROGRAM.BIN as the binary object output file.

```
ASM5C –O PROGRAM.SRC
```

Uses the file PROGRAM.SRC as the Assembly source file and produces no object output.

4.5.4 Assembly Tagged Object File

If needed, the assembler can substitute an object file in 990 tagged object format for the binary format file. If produced, the object output is placed in a file with the same file name as the assembly source except that the extension will be '.MPO'.

For example:

```
ASM5C –9 PROGRAM.SRC
```

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

4.5.5 Assembly Listing File

The assembly process produces a listing file which contains the source instructions, the assembled code, and a cross-reference table (optional). The listing file will be placed in a file with the same file name as the assembly source except that the extension will be .LST.

For example:

```
ASM5C PROGRAM.SRC
```

Uses the file PROGRAM.SRC as the assembly source file and the file PROGRAM.LST as the assembly listing file.

4.6 Options and Switches

4.6.1 Command Line Options

Several options can be invoked from the command line. This is done by listing the option abbreviation prefixed by a minus sign.

For example:

```
ASM5C –lo PROGRAM.ASM
```

Assembles the program in file PROGRAM.ASM without generating either a listing file or an object file. Errors are written to the console. The following command line options are available (see Table 4-1).

4.6.1.1 BYTE Unlist Option

Placing a "b" or "B" in the command field causes the assembler to list only the first data byte in a BYTE or RBYTE statement. If a BYTE or RBYTE statement has n arguments, then n lines are used to list the resulting data in the object column. 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.

4.6.1.2 DATA Unlist Option

Placing a "d" or "D" in the command field causes the assembler to list only the first data byte in a DATA or RDATA statement. If a DATA or RDATA statement has n arguments, then n lines are used to list the resulting bytes in the object column. If the DATA unlist switch is set, then only the first line (which also contains the source line listing) will be written to the listing file.

4.6.1.3 XREF Unlist Option

Placing an "x" or "X" in the command field causes the assembler to add a cross-reference list at the end of the listing file.

4.6.1.4 TEXT Unlist Option

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

4.6.1.5 WARNING Unlist Option

Placing a "w" or "W" in the command field causes the assembler to suppress WARNING messages. However, warnings are counted and error messages are generated.

4.6.1.6 8K Assembly Mode Option

Placing an "8" in the command field puts the assembler in 8K mode. This has the effect of dividing the address generated by any branch by two and performing a check that any label addressed by a branch is on an even address.

4.6.2 Complete XREF Switch

Placing an "r" or "R" in the option field causes the assembler to create a reduced XREF listing if one is produced. All symbols (whether used or not) are listed. The 'r' option causes the assembler to omit from the XREF listing all symbols in the copy files that were never used.

4.6.3 Object Module Switch

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

4.6.4 Listing File Switch

Placing an "l" or "L" in the option field causes the assembler not to generate the listing file but to display error messages on the screen.

4.6.5 Page Eject Disable Switch

Placing a "p" or "P" in the option field causes the assembler to print the listing in a continual manner without division into separate pages. A form feed can be forced where desired using the PAGE command.

4.6.6 Error to Screen Switch

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

4.6.7 Binary Code File Disable Switch

Placing a "9" in the 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 object module in a file with a .BIN extension.

Table 4-1. Switches and Options

CHARACTER OR NUMBER	OPTION DESCRIPTION
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
L or l	Displays error messages without generating a listing
O or o	Generates object output module disable
P or p	Prints listing without page breaks
R or r	Produces a reduced XREF list
S or s	Writes no errors on screen unless no listing file is generated
T or t	Lists only the first data byte in a TEXT or RTEXT
W or w	Suppresses the warning message
X or x	Adds a cross-reference list at the end
8	Checks if any label addressed by a branch is on even boundary. Adjusts branch addresses for the 8K mask option.
9	Generates object module in tagged object format

4.7 Assembler Directives

Assembler directives are instructions that modify the assembler operation. They are invoked by placing the directive mnemonic in the command field and any changing operands in the operand field. The valid directives are described in the following paragraphs and are summarized in Table 4-2.

Table 4-2. Summary of Assembler Directives

DIRECTIVES THAT AFFECT THE LOCATION COUNTER		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
AORG	Absolute origin	[⟨label⟩] ^ AORG ^ ⟨expression⟩ ^ [⟨comment⟩]
BES	Block ending with symbol	[⟨label⟩] ^ BES ^ ⟨expression⟩ ^ [⟨comment⟩]
BSS	Block starting with symbol	[⟨label⟩] ^ BSS ^ ⟨expression⟩ ^ [⟨comment⟩]
EVEN	Even boundary	[⟨label⟩] ^ EVEN ^ [⟨comment⟩]
DIRECTIVES THAT AFFECT ASSEMBLER OUTPUT		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
IDT	Program identifier	[⟨label⟩] ^ IDTR ^ ⟨string⟩ ^ [⟨comment⟩]
LIST	Restart source listing	[⟨label⟩] ^ LIST ^ ⟨expression⟩ ^ [⟨comment⟩]
OPTION	Output options	[⟨label⟩] ^ OPTION ^ ⟨option list⟩ ^ [⟨comment⟩]
PAGE	Page eject	[⟨label⟩] ^ PAGE ^ [⟨comment⟩]
TITL	Page title	[⟨label⟩] ^ TITL ^ ⟨string⟩ ^ [⟨comment⟩]
UNL	Stop source listing	[⟨label⟩] ^ UNL ^ [⟨comment⟩]
DIRECTIVES THAT INITIALIZE CONSTANTS		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
BYTE	Initialize byte	[⟨label⟩] ^ BYTE ^ ⟨expr-1⟩ ^ [,⟨expr-2,⟩. . ., ⟨expr-n⟩] ^ [⟨comment⟩]
RBYTE	Reverse bit initialization of byte	[⟨label⟩] ^ BYTE ^ ⟨expr-1⟩ ^ [,⟨expr-2,⟩. . ., ⟨expr-n⟩] ^ [⟨comment⟩]
DATA	Initialize word	[⟨label⟩] ^ DATA ^ ⟨expr-1⟩ ^ [,⟨expr-2,⟩. . ., ⟨expr-n⟩] ^ [⟨comment⟩]
RDATA	Reverse bit initialization of word	[⟨label⟩] ^ RDATA ^ ⟨expr-1⟩ ^ [,⟨expr-2,⟩. . ., ⟨expr-n⟩] ^ [⟨comment⟩]
EQU	Define assembly-time	[⟨label⟩] ^ EQU ^ [⟨comment⟩]
TEXT	Initialize text	[⟨label⟩] ^ TEXT ^ [−]'⟨string⟩' ^ [⟨comment⟩]
RTEXT	Reverse bit initialization of text	[⟨label⟩] ^ RTEXT ^ [−]'⟨string⟩' ^ [⟨comment⟩]
MISCELLANEOUS DIRECTIVES		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
COPY	Copy source file	[⟨label⟩] ^ COPY ^ ⟨filename⟩ ^ [⟨comment⟩]
END	Program end	[⟨label⟩] ^ END ^ ⟨symbol⟩ ^ [⟨comment⟩]

4.7.1 AORG Directive

The AORG directive places the value in the operand field into the location counter. Subsequent instructions will 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+OFSET
```

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

4.7.2 BYTE Directive

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

4.7.3 BES Directive

The BES directive is used to reserve a block of memory. It advances the location counter by the value in the expression field. The label field may be used to assign the value of the memory location following the reserved block. The command field contains BES. The operand field contains a well-defined expression that represents a positive integer that gives the number of words to be added to the location counter. A well-defined expression is one that includes no symbols that are defined later in the source program.

The syntax of the BES directive is as follows:

```
[⟨label⟩] ^ BES ^ ⟨expression⟩ ^ [⟨comment⟩]
```

EXAMPLE

```
BES #20
```

The example increments the location counter by 32.

4.7.4 BSS Directive

The BSS directive is used to reserve a block of memory. It advances the location counter by the value in the expression field. The use of the label field is optional. When used, a label is assigned the value of the location of the first word in the block. The command field contains BSS. The operand field contains a well-defined expression that represents a positive integer that gives the number of words to be added to the location counter.

The syntax of the BSS directive is as follows:

```
    [⟨label⟩] ^ BSS ^ ⟨expression⟩ ^ [⟨comment⟩]
```

EXAMPLE

```
    BSS 20
```

The example increments the location counter by 20.

4.7.5 COPY Directive

The COPY directive causes the assembler to read source statements from a different file. The assembler will get subsequent statements from the copy file until either the end of file marker 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 read.

The syntax of the COPY directive is as follows:

```
    [⟨label⟩] ^ COPY ^ ⟨filename⟩ ^ [⟨comment⟩]
```

EXAMPLE

```
    COPY COPY.FIL
```

The directive in the example causes the assembler to take its source statements from a file called COPY.FIL. Until the end of file marker or an END directive is reached in COPY.FIL, the assembler continues processing source statements from the original source file.

4.7.6 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 65535. The command field contains DATA. The operand field contains a series of one or more expressions 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

4.7.7 EQU Directive

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

The syntax of the EQU directive is as follows:

```
[⟨label⟩] ^ EQU ^ ⟨expression⟩ ^ [⟨comment⟩]
```

EXAMPLE

```
OFSET EQU #100
```

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

4.7.8 EVEN Directive

The EVEN directive forces the following instruction to start at an even address. The directive tests whether the following instruction is even. If it is at an even address, then nothing is done; otherwise, a short branch to the next location is inserted in the code.

The syntax of the EVEN directive is as follows:

```
[⟨label⟩] ^ EVEN ^ [⟨comment⟩]
```

EXAMPLE

```
        EVEN
BR1     CLA
```

The example forces the CLA instruction to an even address. In the process, the value of the label is made even. The EVEN directive should be used with the 8K mask option to ensure that all long branch destinations fall on even addresses.

NOTE: Since the EVEN directive produces an even alignment by using a short branch, the status flag is affected. No command that depends on the condition of the status flag for its function should immediately follow an EVEN directive.

4.7.9 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

```
ACAA 1
END
CLA
```

In the example, the ACAA 1 instruction is assembled, but the CLA and any subsequent instructions are ignored. The END directive is not required, the end of the file serves the same purpose.

4.7.10 IDT Directive

The IDT directive assigns a name to the object module produced. If a label is used, it assumes the current value of the location counter. The command field contains IDT. The operand field contains the module name, which is 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).

4.7.11 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. This directive is not printed in the source listing, but the line counter is increased.

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.

4.7.12 OPTION Directive

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

The syntax of the OPTION directive is as follows:

```
[⟨label⟩] ^ OPTION ^ ⟨option-list⟩ ^ [⟨comment⟩]
```

EXAMPLE

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

The two examples above have an identical effect. The binary object file is replaced by one in tagged object format. The cross-reference list is produced, and the error messages are not sent to the screen (unless no source listing file is being produced). The options that are available are listed in the paragraphs below.

4.7.12.1 BUNLST — Byte Unlist Option

This option limits the listing of BYTE or RBYTE directives to one line. If a BYTE or RBYTE directive has more than one operand, the resulting object code is listed in a column in the object column of the source listing. If the directive has ten operands, then ten lines are required in the source listing. BUNLST is used to avoid this.

4.7.12.2 DUNLST — Data Unlist Option

This option limits the listing of DATA or RDATA directives to one line. If a DATA or RDATA directive has more than one operand, the resulting object code is listed in the object column of the source listing. If the directive has ten operands, then ten lines are required in the source listing to list it. DUNLST is used to avoid this.

4.7.12.3 FUNLST — Byte, Data, and Text Unlist Option

This option 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 the TUNLST directives at the same time.

4.7.12.4 LSTUNL — Listing Unlist Option

This option inhibits the listing file from being produced. It takes precedence over the LIST directive.

4.7.12.5 OBJUNL — Object File Unlist Option

This option inhibits either of the object output files from being produced.

4.7.12.6 PAGEOF — Page Break Inhibit Option

This option causes the listing file to be printed in a continous stream without page breaks.

4.7.12.7 RXREF — Reduced XREF Option

This option causes symbols that were found in copy files but never used to be omitted from the cross-reference list (if produced).

4.7.12.8 SCRNOF — Screen Error Message Unlist Option

This option causes the error messages not to be listed to the screen unless the listing file is not being produced.

4.7.12.9 TUNLST — Text Unlist Option

This option limits the listing of TEXT or RTEXT directives to one line. A TEXT or RTEXT directive takes as many lines to list as there are characters in the operand. TUNLST causes only the first line of the directive listing to be produced.

4.7.12.10 WARNOF — Warning Message Unlist Option

This option inhibits the listing of warning diagnostics. However, warnings are counted and the total is printed out at the end of the source listing.

4.7.12.11 XREF — Cross-Reference Listing Enable

This option causes a cross-reference list to be produced at the end of the source listing.

4.7.12.12 8KASM — 8K Assembler Mode Switch

This option causes the assembler to operate in 8K mode. This has the effect of dividing long branch destination values and checking if the long branch is to an even address. If a long branch is to an odd address, an error message is produced.

4.7.12.13 990 — Tagged Object Output Switch

This option causes the assembler to omit the binary coded object module (normally produced in a.bin file) and to produce instead a tagged object module in a.MPO file.

4.7.13 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 is increased. 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
T1   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, although the label T1 is entered into the symbol table and appears in the cross-reference list. The line in which it is assigned a value does not appear in the listing file.

4.7.14 RBYTE Directive

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

The syntax of the BYTE 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. The value of "data" must be known to the assembler.

4.7.15 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 65535. 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 BYTE directive is as follows:

> [⟨label⟩] ^ RDATA ⟨expr__1⟩ [,⟨expr__2⟩, . . .,⟨expr__n⟩] ^ [⟨comment⟩]

EXAMPLE

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

The example places the following bytes into successive locations in program memory: #00, #07, #82, #24

4.7.16 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 of the string to be written (which is the first character of the string as given) is done with its most significant bit set high. When used, the label assumes the current value of the location counter. The command field contains TITL. The operand field contains a character string of up to 52 characters 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 will write 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)].

4.7.17 TEXT Directive

The TEXT directive writes an ASCII string to the object file. If the string is preceded by a minus sign, then the last character in the string is written with its most significant bit set high. When used, the label assumes the current value of the location counter. The command field contains TITL. The operand field contains a character string of up to 52 characters 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 will write the final 't' in the word test with its most significant bit set high (that is, as a #F4 instead of a #74).

4.7.18 TITL Directive

The TITL directive supplies a title to be printed in the heading of each page of the source listing. If a title is desired, 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 will increment. When used, a label field assumes the current value of the location counter. The command field contains TITL. The operand field contains the title, a character string of up to 50 characters enclosed in single quotes. When more than 50 characters are entered, the assembler retains the first 50 characters as the title and prints a syntax error message. The comment field is optional.

The syntax of the TITL directive is as follows:

```
[⟨label⟩] ^ TITL '⟨string⟩' ^ [⟨comment⟩]
```

EXAMPLE

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

The example causes the title 'Sample Program' to be printed as 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. This line is not printed to the listing file.

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

5 Instruction Set

There are 61 different TSP50C4X instructions. Most of them require only one instruction cycle to execute, although a few require two or three. Instruction cycles require 32 clock cycles each. For example, if the clock speed is 3.84 MHz., that translates to 120,000 instruction cycles per second.

TABLE 5-1. TSP50C4X Instruction Set

Mnemonic	Operand size (bits)	Instruction cycles required	Status (1 always set, C conditional)	Number of bytes required	Description
ACAA	10	2	1	2	Add constant to A
AMAAC		1	C	1	Add memory to A
ANEC	8	2	C	2	A not equal to constant
BR	12	2	1	2	Branch if status is set
CALL	12	2	1	2	Call if status is set
CLA		1	1	1	Clear A
CLB		1	1	1	Clear B
CLX		1	1	1	Clear X
DECMC		1	C	1	Decrement memory
EXTRM		1	1	1	External data ROM mode
EXTSG		1	1	1	Extended sign mode
GET	3	*	1	1	Get bits from data
IBC		1	C	1	Increment B
INCMC		1	C	1	Increment memory
INTD		1	1	1	Interrupt disable
INTE		1	1	1	Interrupt enable
INTGR		1	1	1	Integer mode
INTRM		1	1	1	Internal data ROM mode
IXC		1	C	1	Increment X
LUAA		2	1	1	Lookup accumulator
LUSPS		2	1	1	Lookup PS register
POP		1	1	1	Pop stack
RBITM	3	1	1	1	Reset bit in memory
RETI		1	C	1	Return from interrupt
RETN		1	1	1	Return from subroutine
RSECT		1	1	1	Reset timer source

*The GET instruction requires three instruction cycles to execute if the parallel-to-serial register must be reloaded during the instruction. Otherwise it takes only two.

TABLE 5-1. TSP50C4X Instruction Set (Continued)

Mnemonic	Operand size (bits)	Instruction cycles required	Status (1 always set, C conditional)	Number of bytes required	Description
RSRDY		1	1	1	Reset RDY latch
SALA		1	1	1	Shift A left
SARA		1	1	1	Shift A right
SBITM		1	1	1	Set bit in RAM
SBR	7	1	1	1	Short branch
SETOFF		1	1	1	Turn processor off
SMAAN		1	C	1	Subtract memory from A
START		1	1	1	Start synthesis
STOP		1	1	1	Stop synthesis
TAPA		1	1	1	Transfer A to PA
TAPB		1	1	1	Transfer A to PB
TAPD		1	1	1	Transfer A to PD
TAM		1	1	1	Transfer A to memory
TAPRF		1	1	1	Transfer A to PF
TAPSC		1	1	1	Transfer A to Prescale
TASH		1	1	1	Transfer A to SA high
TASL		1	1	1	Transfer A to SA low
TAV		1	1	1	Transfer A to V latch
TAX		1	1	1	Transfer A to X
TBA		1	1	1	Transfer B to A
TBITA	3	1	C	1	Test bit in A
TBITM	3	1	C	1	Test bit in memory
TCX	8	2	1	2	Transfer constant to X
TMA		1	1	1	Transfer memory to A
TMAIX		1	1	1	Transfer memory to A, IXC
TMEDA		1	1	1	Transfer memory to DAC
TPAA		1	1	1	Transfer PA to A
TPAM		1	1	1	Transfer PA to memory
TPCA		1	1	1	Transfer PC to A
TTMA		1	1	1	Transfer Timer to A
TXA		1	1	1	Transfer X to A
TXPA		1	1	1	Transfer X to PA
TXTM		1	1	1	Transfer X to Timer
XBX		1	1	1	Exchange B and X
XGEC	8	2	C	2	X greater than or equal to constant

5.1 Instruction Format

The source code instruction format or syntax can be generally described as follows:

[⟨LABEL⟩] ∧ ⟨opcode mnemonic⟩ ∧ [⟨operand⟩] ∧ ... [⟨COMMENT⟩]

The fields are:

a 6-character optional label field,
a 6-character opcode field,
an opcode dependent operand field,
and a comment field.

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

5.2 ACAA – Add Constant to A

ACTION: Adds a 10-bit specified constant in the operand field to the contents of the A register and stores the result back in the A register.

OPCODE: 50 – 53

SOURCE CODE: [⟨LABEL⟩] ^ ACAA ^ ⟨CONST10⟩ ^ ... [⟨COMMENT⟩]

OBJECT CODE:

	7	6	5	4	3	2	1	0	
INSTRUCTION	0	1	0	1	0	0			← 2 most significant bits of + constant
CONSTANT	CONST10								← 8 least significant bits of + constant

EXECUTION RESULTS: (A) + CONST10 → (A)

STATUS FLAG: Always set to 1 after execution.

NOTE: The addition is performed independent of the arithmetic mode (EXTSG/INTGR) as an unsigned addition of all 10 bits of the constant and the A register. See subsection 6.2 for further information on arithmetic instructions.

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 address and a lookup can be completed to fetch the desired table element.

```
TABLE
        TPAA              Bring phrase number in from Port A
        ACAA     TABLE    Add start of phrase pointer table
        LUAA              Bring pointer value into A
```

5.3 AMAAC – Add Memory to A

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

OPCODE: 16

SOURCE CODE: [⟨LABEL⟩] ^ AMAAC ^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A) + (*X) → (A)

STATUS FLAG: Conditionally set to 1 if, as a result of the arithmetic operations, there is a carry into bit eight of the ALU. Else set to 0.

NOTE: The results of the addition are dependent on the arithmetic mode (EXTSG/INTGR) when the most significant bit of the memory being used is set. A carry into bit eight sets the status flag in all cases. See subsection 6.2 for further information on arithmetic instructions.

This instruction should be used when the sum of two variables is desired. It always adds the contents of the memory indexed by the X register to the A register.

```
TCX      VALU1   Point at VALU1 in RAM
TMA              Load VALU1 into A
TCX      VALU2   Point at VALU2 in RAM
AMAAC            Add VALU2 to VALU1
TCX      VALU3   Point at VALU3 in RAM
TAM              Store result of addition in VALU3
```

5.4 ANEC – A Not Equal to Constant

ACTION: Compares the lower eight bits of the A register to the constant specified and sets the status flag if they are not equal.

OPCODE: 54

SOURCE CODE: [⟨LABEL⟩] ^ ANEC ^^ ⟨CONST8⟩^... [⟨COMMENT⟩]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION	0	1	0	1	0	1	0	0
CONSTANT	CONST8							

EXECUTION RESULTS: if (A) ⟨⟩ CONST8 then 1 → SF
if (A) = CONST8 then 0 → SF

STATUS FLAG: Set to 1 if the lower eight bits of the A register are not equal to the 8-bit constant. Set to 0 if the two 8-bit values are equal.

NOTE: Only the lower eight bits of the A register are compared to the 8-bit constant value. The upper two bits of the A register are not considered, and as a result, the operation is independent of the arithmetic mode INTGR or EXTSG.

```
        ANEC    TESTY   Is A equal to TESTY
        SBR     NOTEG   Branch if not
EQUAL
NOTEG
```

5.5 BR — Branch

ACTION: If the status flag is set to 1, the program counter is loaded with the address specified and execution proceeds from that address. If the 8K mask option is selected, the address will be loaded into bits one to twelve of the program counter and bit zero will be a 0. In the 4K mode, bits zero to eleven will be loaded. Otherwise, the instruction following the BR instruction executes.

OPCODE: 60 — 6F

SOURCE CODE: [⟨LABEL⟩] ^ BR ^^^^ ⟨ADDR12⟩... [⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: if SF = 1 then ADDR12 → Program Counter
if SF = 0 then Program Counter → Program Counter

STATUS FLAG: Always set to 1 after execution.

NOTE: The branch instruction is a conditional instruction. When a branch is used following an instruction, which always leaves the status flag set high, the branch can be viewed as unconditional. See the Applications section for further information on branching/programming flow modification.

5.6 CALL — Call Subroutine

ACTION: If the status flag is set to 1, the contents of the program counter are pushed onto the stack and the program counter loaded with the address specified. Execution proceeds from that address. If the 8K mask option is selected, the address will be loaded into bits one to twelve of the program counter and bit zero will be a 0. In the 4K mode, bits zero to eleven will be loaded. Otherwise, the instruction following the CALL instruction executes.

OPCODE: 70 — 7F

SOURCE CODE: [⟨LABEL⟩] ^ CALL ^^ ⟨ADDR12⟩ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: if SF = 1 then Program Counter → STACK and ADDR12 → Program Counter

STATUS FLAG: Always set to 1 after execution.

NOTE: The program counter stack is capable of storing addresses up to five levels deep. An address is pushed onto the STACK whenever a timer interrupt occurs or when the CALL, GET, LUAA, or LUSPS instructions are executed. For GET, LUAA, and LUSPS, the address is popped from the stack before the instruction is complete.

The call instruction is a conditional instruction. When a call is used following an instruction, which always leaves STATUS high, it can be viewed as unconditional. See the Applications section for further information on branching/programming flow modification.

5.7 CLA — Clear A Register

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

OPCODE: 00

SOURCE CODE: [⟨LABEL⟩] ^ CLA ^^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $0 \rightarrow (A)$

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction is used to initialize the A register prior to loading it with a constant, using ACAA or shifting some number of bits from the parallel-to-serial register into the A register.

```
CLA
ACAA    CONST   Add constant to A register

CLA
GET     3       Get bits from data ROM
```

5.8 CLB — Clear B Register

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

OPCODE: 12

SOURCE CODE: [⟨LABEL⟩] ^ CLB ^^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: 0 → (B)

STATUS FLAG: Always set to 1 after execution.

5.9 CLX — Clear X Register

ACTION: The contents of the X register are set to 0.

OPCODE: 11

SOURCE CODE: [⟨LABEL⟩] ^ CLX ^^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: 0 → (X)

STATUS FLAG: Always set to 1 after execution.

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

5.10 DECMC — Decrement Memory

ACTION: Decrements the contents of the RAM location pointed to by the X register. If the memory location contains 00, it is set to FF and the status flag is set. Otherwise, the memory is decremented and the status flag is cleared.

OPCODE: 5F

SOURCE CODE: [⟨LABEL⟩] ∧ DECMC ∧...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $(*X) - 1 \rightarrow (*X)$

STATUS FLAG: Set if memory went from 0 to FF during instruction; otherwise cleared.

5.11 EXTRM – External ROM Mode

ACTION: Changes the path to the parallel-to-serial register from the internal ROM to the external ROM input data buffer.

OPCODE: 2B

SOURCE CODE: [⟨LABEL⟩] ∧ EXTRM ∧ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS:

1. Access path for data into the parallel-to-serial register changed from internal ROM to the external ROM input data buffer.
2. Output a (OSC/16, 50% duty cycle) clock to the PB6 pin.
3. Change PB7 output to RDIN input from the external ROM to the internal ROM buffer.
4. If a 0 is written to Port B, then the output clock is generated for the TSP60CXX by the GET instruction.

STATUS FLAG: Always set to 1 after execution.

NOTE: This mode is used to enable the interface to an external TSP60CXX.

This instruction changes the source for ROM data acquisition but does not initialize either the internal registers or the external ROM. Refer to Section 6 for more information on TSP60CXX interface.

5.12 EXTSG — Sign Mode

ACTION: Changes ALU to extended sign mode.

OPCODE: 29

SOURCE CODE: [⟨LABEL⟩] ^ EXTSG ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: The upper two bits of the ALU are filled with the value from bit seven for future arithmetic operations and data transfers to the A register.

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction affects all data from RAM, ports, B register, X register, or timer register that is being transferred to the A register or being added to or subtracted from the current A register value. See Section 6 for more information.

5.13 GET — Get Data from ROM

ACTION: Transfers N bits of data from ROM to the A register via the parallel-to-serial register.

OPCODE: 20 – 27

SOURCE CODE: [⟨LABEL⟩] ^ GET^^^ ⟨N⟩ ^...[⟨COMMENT⟩]

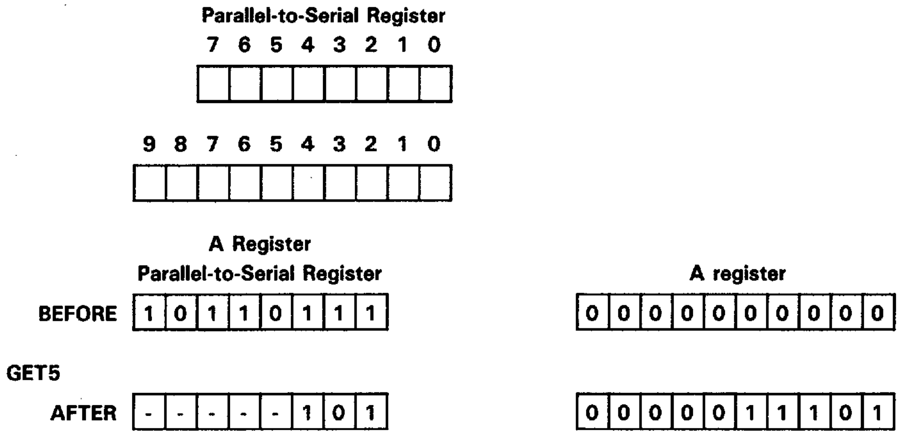
OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION AND NUMBER OF BITS	0	0	1	0	0	N−1		

EXECUTION RESULTS: N bits of data are transferred from the LSB of the parallel-to-serial register to 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 the ROM.

STATUS FLAG: Always set to 1 after execution.

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, a third instruction cycle is used to load the parallel-to-serial register with the next ROM data byte and transfer the remaining bits to the A register to satisfy the request.

Prior to the first use of the GET instruction, the GET counter and the parallel-to-serial register must be initialized. This initialization is accomplished by the LUSPS instruction, independent of the ROM source mode.

The EXTRM instruction causes the GET instruction to fetch data from an external ROM. The INTRM instruction causes GET to work with the internal TSP50C4X ROM.

During execution, the GET instruction pushes the program counter onto the stack and then pops it again. This uses up one stack level that cannot be used for other purposes. If there are five values on the stack and a GET instruction is executed, the bottom value will be lost.

5.14 IBC — Increment B Register

ACTION: Increments the contents of the B register by 1.

OPCODE: 13

SOURCE CODE: [⟨LABEL⟩] ^ IBC ^^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (B) + 1 → (B)

STATUS FLAG: Conditionally set to 1 when B goes from FF to 0 as a result of the arithmetic operation; otherwise set to 0.

NOTE: The status flag will only be set when the B register contains the value FF prior to the execution of the IBC instruction. In this case, the status flag will be set and the B register value will be 0. See subsection 6.2 for further information on arithmetic instructions.

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

5.15 INCMC — Increment Memory

ACTION: Increments the contents of the RAM location pointed to by the X register. If the memory location contains FF, sets it to 00 . Also sets the status flag. Otherwise, increments the memory and clears the status flag.

OPCODE: 5E

SOURCE CODE: [⟨LABEL⟩] ^ INCMC ^,..[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $(*X) + 1 \rightarrow (*X)$

STATUS FLAG: Set if memory went from FF to 0 during instruction; otherwise cleared.

5.16 INTD — Interrupt Disable

ACTION: Disables the timer from interrupting the present process flow.

OPCODE: 1D

SOURCE CODE: [⟨LABEL⟩] ^ INTD ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: The timer interrupt is disabled.

STATUS FLAG: Always set to 1 after execution.

NOTE: Only the timer interrupt is disabled. The timer register continues to decrement until an interrupt state is reached. When the interrupt state is reached, the timer interrupt is set and the timer register value is left at the FF value. If the interrupt is disabled as a result of this instruction, it becomes a pending interrupt until the processor reset or until the INTE instruction is executed.

5.17 INTE — Interrupt Enable

ACTION: Permits timer interrupts to occur and enables interpolation for one frame.

OPCODE: 1E

SOURCE CODE: [⟨LABEL⟩] ^ INTE ^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: Enable the timer to interrupt the present process flow when the timer register decrements past zero.

STATUS FLAG: Always set to 1 after execution.

NOTE: If the timer register has been reset, it will be decremented until an interrupt state is reached. When the interrupt state is reached, the timer register value is left at the FF value. If the interrupt is disabled as a result of the INTD instruction, it becomes a pending interrupt until it is reset or until this instruction is executed.

5.18 INTGR — Integer Mode

ACTION: Changes the state of the ALU to integer mode.

OPCODE: 2A

SOURCE CODE: [⟨LABEL⟩] ^ INTGR ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: The upper two bits of the ALU are filled with zeros for future arithmetic operations.

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction affects all data from RAM, ports, B register, X register, or timer register that is being transferred to the A register or being added to or subtracted from the current A register value.

5.19 INTRM — Internal ROM Mode

ACTION: Makes the internal ROM the source for GET instruction data transfers.

OPCODE: 2C

SOURCE CODE: [⟨LABEL⟩] ^ INTRM ^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS:

1. Access path for data into the parallel-to-serial register changed to internal ROM from the external ROM input data buffer.
2. Change PB6 to a data output pin.
3. Change PB7 to an output pin.
4. Change PB0 to output only from Port B.

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction changes the source for ROM data acquisition but does not initialize any internal registers. Port B becomes an 8-bit output port as a result of this instruction.

WARNING! This instruction makes the PB7 pin an output. If the system uses a TSP60CXX for external memory, its output must be disabled before the INTRM instruction is executed. Otherwise, bus contention and high power consumption may occur.

5.20 IXC — Increment X Register

ACTION: Increments the contents of the X register by 1.

OPCODE: 0F

SOURCE CODE: [〈LABEL〉] ^ IXC ^... [〈COMMENT〉]

OBJECT CODE:

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

EXECUTION RESULTS: (X) + 1 → (X)

STATUS FLAG: Set to 1 if X increments from FF to 0, otherwise set to 0.

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

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

5.21 LUAA — Lookup with A Register

ACTION: Replaces the contents of the A register by the contents of the ROM addressed by the A register.

OPCODE: 58

SOURCE CODE: [⟨LABEL⟩] ^ LUAA ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (*A) → (A)

STATUS FLAG: Always set to 1 after execution.

NOTE: The program counter stack is capable of storing addresses up to five levels deep. An address is pushed onto the stack whenever a timer interrupt occurs or when the CALL, GET, LUAA, or LUSPS instructions are executed. For GET, LUAA, and LUSPS, the address is popped from the stack before the instruction is complete.

Since the A register is a 10-bit register, this instruction allows lookup access to the first 1K bytes of ROM.

```
TABLE
          TPAA               Bring phrase number in from Port A
          ACAA     TABLE     Add start of phase pointer table
          LUAA               Bring pointer value into A
```

5.22 LUSPS — Lookup with Speech Address Register

ACTION: Replaces the contents of the parallel-to-serial register with the contents of the internal ROM addressed by the speech address register, increments the speech address register, sets the parallel-to-serial register counter to eight bits to indicate that the parallel-to-serial register is full.

OPCODE: 59

SOURCE CODE: [⟨LABEL⟩] ^ LUSPS ^ [⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $(*SA) \rightarrow (PS)$
$(SA) + 1 \rightarrow (SA)$

STATUS FLAG: Always set to 1 after execution.

NOTE: The program counter stack is capable of storing addresses up to five levels deep. An address is pushed onto the STACK whenever a timer interrupt occurs or when the CALL, GET, LUAA, or LUSPS instructions are executed. For GET, LUAA, and LUSPS, the address is popped from the stack before the instruction is complete.

Since the speech address register is a 14-bit register, this instruction allows lookup into any of the 16K bytes of internal ROM on the TSP50C43 and TSP50C44. On the TSP50C41 and TSP50C42, there are 8K bytes available.

TCX	ADDR	Point to RAM location with the high five bits of the speech counter
TMAIX		Speech pointer to A, point at low byte in RAM
TASH		Speech pointer to SAR
TMA		Low byte of speech pointer into A
TASL		Low byte of speech pointer into SAR
LUSPS		Initialize parallel-to-serial register.

5.23 POP — Pop Top Stack Location

ACTION: Pops and discards the top location on the stack. Moves all other stack values up by one.

OPCODE: 57

SOURCE CODE: [⟨LABEL⟩] ^ POP ^ ⟨N⟩ ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: Top of stack popped and discarded

STATUS FLAG: Always set to 1 after execution.

5.24 RBITM — Reset Bit in Memory

ACTION: Addresses Bit N of the RAM with the X register and resets it to 0. Where N = 1 through 8.

OPCODE: 48 – 4F

SOURCE CODE: [⟨LABEL⟩] ^ RBITM ^ ⟨N⟩ ^...[⟨COMMENT⟩]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION AND BIT NUMBER	0	1	0	0	1	N−1		

EXECUTION RESULTS: 0 → (RAM(*X register, *N − 1))

STATUS FLAG: Always set to 1 after execution.

NOTE: Any bit in the internal RAM can be reset as a result of this instruction.

5.25 RETI — Return from Timer Interrupt

ACTION: Retrieves the old contents of the A register, status flag, 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.

OPCODE: 2F

SOURCE CODE: [⟨LABEL⟩] ^ RETI ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A′) → (A)
(X′) → (X)
(SF′) → (SF)
(top of Program Counter Stack) → (Program Counter)

STATUS FLAG: Restored to value before interrupt

NOTE: The contents of the B register are not saved during interrupt.

5.26 RETN — Return from Subroutine

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

OPCODE: 1F

SOURCE CODE: [⟨LABEL⟩] ^ RETIN^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: top of stack →(Program Counter)

STATUS FLAG: Always set to 1 after execution.

5.27 RSECT – Reset Prescale Clock Source to Internal Mode

ACTION: Resets prescale clock source as the internal clock.

OPCODE: 2E

SOURCE CODE: [⟨LABEL⟩] ∧ RSECT ∧ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: Internal clock gated to decrement the timer prescale register.

STATUS FLAG: Always set to 1 after execution.

5.28 RSRDY — Reset $\overline{\text{RDY}}$ Pin

ACTION: Resets the $\overline{\text{RDY}}$ pin to a low logic level.

OPCODE: 1C

SOURCE CODE: [⟨LABEL⟩] ∧ RSRDY ∧...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $0 \rightarrow \overline{\text{RDY}}$

STATUS FLAG: Always set to 1 after execution.

NOTE: The $\overline{\text{RDY}}$ pin is set high when the $\overline{\text{ENA2}}$ pin is pulled low by the external system. The $\overline{\text{RDY}}$ pin will go low only if the external system is not actively holding the $\overline{\text{ENA2}}$ pin low. This instruction is used if the TSP50C4X is masked for the slave option.

5.29 SALA – Shift A Register Left Arithmetic

ACTION: Shifts the contents of the A register to the left towards MSB by one bit and fills the LSB with a 0.

OPCODE: 19

SOURCE CODE: [⟨LABEL⟩] ^ SALA ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $(A)i \rightarrow (A)i+1$
$0 \rightarrow (A)1$

STATUS FLAG: Always set to 1 after execution.

NOTE: Only data shifted out of bit 9 of the A register is lost. The results do not depend on the arithmetic mode (EXTSG/INTGR).

5.30 SARA – Shift A Register Right Arithmetic

ACTION: Shifts the contents of the A register to the right toward LSB by one bit and fills the MSB with its old value.

OPCODE: 18

SOURCE CODE: [⟨LABEL⟩] ^ SARA ^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $(A)_i \rightarrow (A)_{i-1}$
$(A)_9 \rightarrow (A)_9$

STATUS FLAG: Always set to 1 after execution.

NOTE: The execution of this instruction is independent of the arithmetic mode (EXTSG/INTGR).

5.31 SBITM — Set Bit in Memory

ACTION: Bit N of the RAM memory addressed by the X register is set to 1. Where N = 1 through 8

OPCODE: 38 — 3F

SOURCE CODE: [⟨LABEL⟩] ^ SBITM ^ ⟨N⟩ ^...[⟨COMMENT⟩]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION AND BIT NUMBER	0	0	1	1	1	N−1		

EXECUTION RESULTS: 1 →(RAM(∗X,∗N − 1))

STATUS FLAG: Always set to 1 after execution.

NOTE: Any bit in the internal RAM can be set as a result of this instruction.

5.32 SBR – Short Branch

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

SOURCE CODE: [⟨LABEL⟩] ∧ SBR ∧ ⟨ADDR7⟩ ∧ ..[⟨COMMENT⟩]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION AND ADDRESS	1	ADDR7						

EXECUTION RESULTS:

if SF = 1 then ADDR7 + Program Counter PAGE → Program Counter
if SF = 0 then Program Counter → Program Counter

STATUS FLAG: Always set to 1 after execution.

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. See the Applications section for further information on branching/programming flow modification.

The program counter is incremented when the instruction is fetched. Placing a SBR at address 07F relative to the current page will result in a branch to the next page. This occurs because the program counter value will be 000 relative to the following page at the time of execution.

5.33 SETOFF — Set Processor to Off Mode

ACTION: The processor is placed in a low-power mode. The clock is stopped. Bidirectional ports are made inputs and all outputs are cleared. RAM memory is retained.

OPCODE: 5A

SOURCE CODE: [⟨LABEL⟩] ∧ SETOFF ∧ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: Processor powered down

STATUS FLAG: State at power up not guaranteed.

5.34 SMAAN – Subtract Memory from A Register

ACTION: The contents of the RAM memory addressed by the X register are subtracted from the lower eight bits of the A register and the result is stored back in the A register.

OPCODE: 17

SOURCE CODE: [⟨LABEL⟩] ^ SMAAN ^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A) – (*X) → (A)

STATUS FLAG: Conditionally set to 1 when the memory is equal to or less than the A register; otherwise set to 0.

NOTE: The subtraction results are dependent on the arithmetic mode (EXTSG/INTGR) when the most significant bit of the memory being used is set. A carry into bit 8 sets the status flag in all cases. See subsection 6.2 for further information on arithmetic instructions.

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

```
TCX      VALU1    Point at VALU1 in RAM
TMA               Load VALU1 into A register
TCX      VALU2    Point at VALU2 in RAM
SMAAN             Subtract VALU2 from VALU1
TCX      VALU3    Point at VALU3 in RAM
TAM               Store result of subtraction in VALU3
```

5.35 START — Start Synthesizer

ACTION: Starts the filter clock, enables the LPC-10 lattice filter, and loads the pitch register from the A register.

OPCODE: 1A

SOURCE CODE: [⟨LABEL⟩] ^ START ^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: Enable the lattice filter to output speech using the parameters stored in RAM.

(A) → (pitch)

STATUS FLAG: Always set to 1 after execution.

NOTE: Since the START instruction loads the pitch register in addition to starting the filter clock, make sure that the correct value is in the A register when it is executed. After the START, the filter will get pitch information from the specified RAM location.

5.36 STOP — Stop Synthesizer

ACTION: Stops the filter clock, disables the LPC-10 lattice filter.

OPCODE: 1B

SOURCE CODE: [⟨LABEL⟩] ^ STOP ^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: Disable the lattice filter and stop speech output.

STATUS FLAG: Always set to 1 after execution.

NOTE: The STOP instruction resets the filter data acquisition mode from the direct digital-to-analog mode established by the TMEDA instruction. It also disables the context switch and sets it to its default condition.

5.37 TAPA — Transfer A to Port A

ACTION: Transfers the contents of the lower eight bits of the A register to the Port A latch. If the TSP50C4X is masked for the master option, Port A becomes an output port. If the TSP50C4X is masked for the slave option and the $\overline{\text{IRT}}$ pin is masked as an output, the $\overline{\text{IRT}}$ pin is driven low.

OPCODE: 08

SOURCE CODE: [⟨LABEL⟩] ^ TAPA ^^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (PA)
if processor masked for slave option and $\overline{\text{IRT}}$ output 0 $\overline{\text{IRT}}$
if processor masked for master option, Port A becomes an output port.

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction is used to send commands or status to the external system. When the TSP50C4X device is masked for the slave option, $\overline{\text{IRT}}$ going low indicates to the host that new data are available. Port A will remain in a high-impedance state until the external system activates it by pulling one or both enable lines active low. This will cause $\overline{\text{IRT}}$ to go to a 1. See subsection 6.3 for a complete discussion of slave option.

In the master option, Port A becomes an output port and the data appear on the pins immediately.

5.38 TAPB — Transfer A Register to Port B

ACTION: Transfers the contents of the lower eight bits of the A register to Port B.

OPCODE: 15

SOURCE CODE: [⟨LABEL⟩] ^ TAPB ^^ ...[⟨COMMENT⟩]

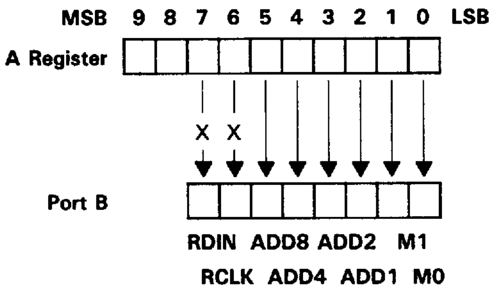
OBJECT CODE:

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

EXECUTION RESULTS: (A) → (PB)

STATUS FLAG: Always set to 1 after execution.

NOTE: Port B is configured as an 8-bit output port for the internal ROM mode (see INTRM instruction). For the external ROM mode (see EXTRM) only the lower six bits of the A register are transferred to Port B (a clock is presented on bit 6 and bit 7 becomes an input).



5.39 TAPD — Transfer A Register to Port D

ACTION: Transfers the contents of the lower eight bits of the A register to Port D.

OPCODE: 5C

SOURCE CODE: [⟨LABEL⟩] ^ TAPD ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (PD)

STATUS FLAG: Always set to 1 after execution.

5.40 TAM — Transfer A Register to Memory

ACTION: Transfers the contents of the lower eight bits of the A register to the RAM addressed by the X register.

OPCODE: 09

SOURCE CODE: [⟨LABEL⟩] ^ TAM ^^^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (∗X)

STATUS FLAG: Always set to 1 after execution.

5.41 TAPRF — Transfer A Register to Pitch Fractional Register

ACTION: Transfers bits 4-7 of the A register to the fractional part of the pitch register.

OPCODE: 0C

SOURCE CODE: [⟨LABEL⟩] ^ TAPRF ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A)7-4 → (PF)

STATUS FLAG: Always set to 1 after execution.

NOTE: The TAPRF instruction is included mainly for compatibility with the TSP50C4X devices. It is only useful on the first frame of synthesis because the synthesizer gets its fractional pitch from a RAM location after that.

5.42 TAPSC — Transfer A Register to Prescale Register

ACTION: Transfers the lower eight bits of the A register to the prescale register.

OPCODE: 5D

SOURCE CODE: [⟨LABEL⟩] ^ TAPSC ^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A)7-0 → (prescale register)

STATUS FLAG: Always set to 1 after execution.

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.

5.43 TASH — Transfer A Register to Speech Address Register High

ACTION: Transfers the lower six bits of the A register to the upper six bits of the speech address register.

OPCODE: 0B

SOURCE CODE: [⟨LABEL⟩] ^ TASH ^^...[⟨COMMENT⟩]

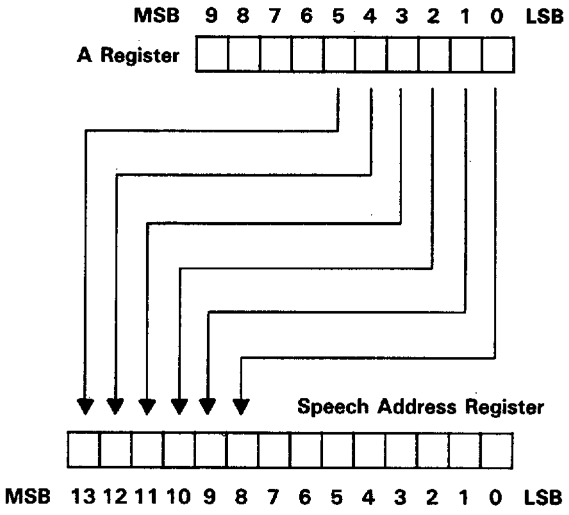
OBJECT CODE:

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

EXECUTION RESULTS: (A)5-0 → (SA)13-8

STATUS FLAG: Always set to 1 after execution.

NOTE:


MSB 9 8 7 6 5 4 3 2 1 0 LSB
A Register
Speech Address Register
MSB 13 12 11 10 9 8 7 6 5 4 3 2 1 0 LSB

5.44 TASL – Transfer A Register to Speech Address Register Low

ACTION: Transfers the lower eight bits of the A register to the lower eight bits of the speech address register.

OPCODE: 0A

SOURCE CODE: [⟨LABEL⟩] ^ TASL ^^ ...[⟨COMMENT⟩]

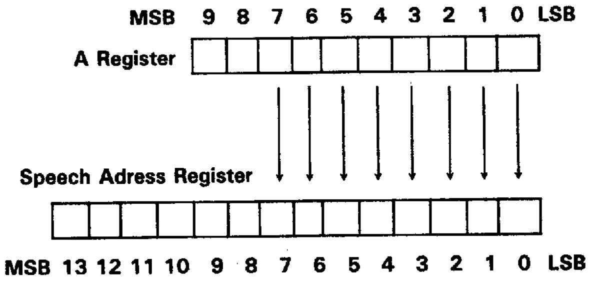
OBJECT CODE:

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

EXECUTION RESULTS: (A)7-0 → (SA)7-0

STATUS FLAG: Always set to 1 after execution.

NOTE:


MSB 9 8 7 6 5 4 3 2 1 0 LSB
A Register
Speech Adress Register
MSB 13 12 11 10 9 8 7 6 5 4 3 2 1 0 LSB

5.45 TAV — Transfer A Register to Voicing Latch

ACTION: Transfers bit 0 of the A register to the voicing latch.

OPCODE: 5B

SOURCE CODE: [⟨LABEL⟩] ^ TAV ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A)0 → (voicing latch)

STATUS FLAG: Always set to 1 after execution.

NOTE: The voicing latch selects either pitch excitation (1) or unvoiced white noise excitation (0).

5.46 TAX — Transfer A Register to X Register

ACTION: Transfers the lower eight bits of the A register to the X register.

OPCODE: 07

SOURCE CODE: [⟨LABEL⟩] ^ TAX ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A) → (X)

STATUS FLAG: Always set to 1 after execution.

5.47 TBA — Transfer B Register to A Register

ACTION: Transfers the contents of the B register to the lower eight bits of the A register.

OPCODE: 02

SOURCE CODE: [⟨LABEL⟩] ^ TBA ^^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS:

sign extended mode	(B)	→	(A)7-0
	(B)7	→	(A)9-8
integer mode	(B)	→	(A)7-0
	0	→	(A)9-8

STATUS FLAG: Always set to 1 after execution.

5.48 TBITA — Test Bit in A Register

ACTION: Tests bit N of the A register and sets the status flag accordingly. Where N = 1 through 8.

OPCODE: 30-37

SOURCE CODE: [⟨LABEL⟩] ^ TBITA ^ ⟨N⟩ ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (A)N − 1 → STATUS FLAG

STATUS FLAG: Set according to tested bit.

NOTE: This instruction is used to test a bit in the A register. The two upper bits of the A register cannot be tested directly by this instruction. To test the two upper bits, a shift must first be executed to shift bits 9 and 8 into bits 7 and 6.

5.49 TBITM — Test Bit in Memory

ACTION: Tests bit N of the RAM addressed by the X register and sets the status flag accordingly. Where N = 1 through 8.

OPCODE: 40-47

SOURCE CODE: [⟨LABEL⟩] ^ TBIT ^ ⟨N⟩ ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (*X)N − 1 → STATUS FLAG

STATUS FLAG: Set according to tested bit.

NOTE: Any bit of the internal RAM can be tested with this instruction.

5.50 TCX — Transfer Constant to X Register

ACTION: Loads the X register with the 8-bit constant specified in the operand field.

OPCODE: 56

SOURCE CODE: [⟨LABEL⟩] ^ TCX ^^^ ⟨CONST8⟩^...[⟨COMMENT⟩]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION	0	1	0	1	0	1	1	0
CONSTANT	CONST8							

EXECUTION RESULTS: CONST8 → (X)

STATUS FLAG: Always set to 1 after execution.

5.51 TMA — Transfer Memory to A Register

ACTION: Transfers the contents of the RAM addressed by the X register to the lower eight bits of the A register.

OPCODE: 04

SOURCE CODE: [〈LABEL〉] ^ TMA ^^^...[〈COMMENT〉]

OBJECT CODE:

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

EXECUTION RESULTS:

sign extended mode	(*X)	→ (A)7-0
	(*X)7	→ (A)9-8
integer mode	(*X)	→ (A)7-0
	0	→ (A)9-8

STATUS FLAG: Always set to 1 after execution.

5.52 TMAIX — Transfer Memory to A Register and Increment X Register

ACTION: Transfers the contents of the RAM memory addressed by the X register to the lower eight bits of the A register and increments the X register.

OPCODE: 05

SOURCE CODE: [⟨LABEL⟩] ^ TMAIX ^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: $(*X) \rightarrow (A)7\text{-}0$

sign extended mode $(*X)7 \rightarrow (A)9\text{-}8$

$(*X) \rightarrow (A)7\text{-}0$

integer mode $0 \rightarrow (A)9\text{-}8$

$(X) + 1 \rightarrow (X)$

STATUS FLAG: Always set to 1 after execution.

5.53 TMEDA — Transfer Memory to DAC

ACTION: Clears all feedback data calculated in the filter so that data is transferred directly to the DAC from RAM location 00.

OPCODE: 28

SOURCE CODE: [⟨LABEL⟩] ^ TMEDA ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: Set filter control to clear filter feedback data and load DAC directly with data stored at RAM address >00.

STATUS FLAG: Always set to 1 after execution.

NOTE: The data is taken from RAM in two's complement format. It is taken every 12 instruction cycles whether it is changed or not. It is necessary to execute the START instruction also to enable the data path to the DAC. Executing the STOP instruction turns the DAC off and disables the TMEDA function. Refer subsection 6.6 for more detail.

5.54 TPAA — Transfer Port A to A Register

ACTION: Transfers the contents of Port A to the lower eight bits of the A register. Makes Port A an input port if masked in the master option.

OPCODE: 03

SOURCE CODE: [⟨LABEL⟩] ^ TPAA ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS:

sign extended mode	(PA) → (A)7-0
	(PA)7→ (A)9-8
integer mode	(PA) → (A)7-0
	0→ (A)9-8

STATUS FLAG: Always set to 1 after execution.

NOTE: If switching Port A from output mode to input mode with this instruction, disregard the data brought in by the first instruction. Execute it once to make it an input port and then execute a second TPAA to bring in valid data.

5.55 TPAM — Transfer Port A to Memory

ACTION: Transfers the contents of Port A to RAM addressed by the X register. Makes Port A an input port if the TSP50C4X devices are masked for the master option.

OPCODE: 14

SOURCE CODE: [⟨LABEL⟩] ^ TPAM ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (PA) → (*X)

STATUS FLAG: Always set to 1 after execution.

NOTE: If switching Port A from output mode to input mode with this instruction, disregard the data brought in by the first instruction. Execute it once to make it an input port and then execute a second TPAM to bring in valid data.

5.56 TPCA – Transfer Port C to A Register

ACTION: Master Option: Transfers the data on Port C to the lower eight bits of the A register.

Slave Option: Transfers the $\overline{\text{RDY}}$ and $\overline{\text{IRT}}$ status flags to bit 0 and bit 1 of the register and bits 4 to 7 of Port C to bits 4 to 7 of the A register.

OPCODE: 06

SOURCE CODE: [⟨LABEL⟩] ^ TPCA ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS:

Master option:

sign extended mode	(PC) → (A)7-0
	(PC)7 → (A)9-8
integer mode	(PC) → (A)7-0
	0 → (A)9-8

Slave option:

(RDY) → (A)0
(IRT) → (A)1
(PC)4 → (A)4
(PC)5 → (A)5)
(PC)6 → (A)6
(PC)7 → (A)7

STATUS FLAG: Always set to 1 after execution.

NOTE: For the TSP50C41 and TSP50C43, bits 4-7 will always be high.

5.57 TTMA — Transfer Timer to A Register

ACTION: Transfers the contents of the timer register to the lower eight bits of the A register and makes the $\overline{\text{IRT}}$ pin an input to the prescale clock.

OPCODE: 2D

SOURCE CODE: [⟨LABEL⟩] ^ TTMA ^^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS:

sign extended mode	(timer) → (A)7-0
	(timer)7→ (A)9-8
integer mode	(timer) → (A)7-0
	0→ (A)9-8

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction is the only way to make the $\overline{\text{IRT}}$ pin the prescale clock source.

5.58 TXA — Transfer X Register to A Register

ACTION: Transfers the contents of the X register to the lower eight bits of the A register.

OPCODE: 01

SOURCE CODE: [⟨LABEL⟩] ^ TXA ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS:

	(X) → (A)7-0
sign extended mode	(X)7→ (A)9-8
	(X) → (A)7-0
integer mode	0→ (A)9-8

STATUS FLAG: Always set to 1 after execution.

8961724 0091832 604

5.59 TXPA — Transfer X Register to Port A

ACTION: Transfers the contents of the X register to Port A. Makes Port A an output port if masked for master option.

OPCODE: 0D

SOURCE CODE: [⟨LABEL⟩] ^ TXD ^ ...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (X) → (PA)
if processor masked for slave mode and $\overline{\text{IRT}}$ output 0 → $\overline{\text{IRT}}$
if processor masked for master mode, PA becomes an output port.

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction is used to send commands or status to the external system. When the TSP50C4X devices are masked for the slave option, $\overline{\text{IRT}}$ going low indicates to the host that new data are available. Port A will remain in a high impedance state until the external system activates it by pulling one or both enable lines active low. This will cause $\overline{\text{IRT}}$ to go to a 1. See subsection 6.3 for a complete discussion of the slave option.

In the master option, Port A is an output port and the data appear on the pins immediately.

5.60 TXTM — Transfer X Register to Timer

ACTION: Transfers the contents of the X register to the timer register.

OPCODE: 10

SOURCE CODE: [⟨LABEL⟩] ^ TXTM ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (X) → (timer)

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction is used to load the timer register with an initial value. Since the timer interrupt occurs when the timer decrements from 0, the value loaded into the timer register should be 1 less than the number of timer cycles desired.

5.61 XBX — Exchange B Register with X Register

ACTION: Exchanges the contents of the B register with the contents of the X register.

OPCODE: 0E

SOURCE CODE: [⟨LABEL⟩] ^ XBX ^...[⟨COMMENT⟩]

OBJECT CODE:

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

EXECUTION RESULTS: (B) $\longleftrightarrow$ (X)

STATUS FLAG: Always set to 1 after execution.

NOTE: This instruction can be used to keep and exchange two index pointers, swap a loop value and index pointer for testing, or load a constant value into the B register.

5.62 XGEC - X Register Greater Than or Equal to Constant

ACTION: Compares the contents of the X register and the constant specified and sets the status flag accordingly.

OPCODE: 55

SOURCE CODE: [⟨LABEL⟩]^ XGEC ^ ⟨CONST8⟩^...[⟨COMMENT⟩]

OBJECT CODE:

	7	6	5	4	3	2	1	0
INSTRUCTION	0	1	0	1	0	1	0	1
CONSTANT	CONST8							

EXECUTION RESULTS: SF = (X) : CONST8

STATUS FLAG: Set to 1 if the value of the X register is greater than or equal to the 8-bit constant. Set to 0 if the value of the X register is lower than the 8-bit constant.

NOTE:

```
        XGEC     TESTV    Is X > = TESTV
        SBR      GTE      Branch if so
LESS
GTE
```

6 Applications

This section presents programming techniques for specific parts of a TSP50C4X device:

6.1 Synthesizer Control

In this section, a sample program demonstrates how to control the synthesizer in the TSP50C4X devices. This program causes the devices to synthesize speech from data stored in the TSP5220 format. It is described here in several steps:

6.1.1 Speech Coding and Decoding

The TSP50C4X devices support linear predictive coding with ten K parameters For more information see, "Introduction to LPC", subsection 1.7.
The LPC model requires three types of information:

1. Pitch
2. Energy
3. 10 K parameters

Pitch parameters control the input into the LPC system by providing one of two excitation signals. If the 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 pitch code is 0, a white noise source similar to the turbulence generated by constricted airflow in the mouth is used. An example of this is the "F" sound. The LPC-10 model is entirely digital; thus the excitation function is a series of digital data samples.

The excitation function specified by the pitch code is then multiplied by the "energy" parameter. The output of the multiplication is put into a filter whose resonance is determined by ten K parameters. To model the resonance of the vocal tract, the output of the LPC-10 model is a series of digital samples, typically at an 8- or 10-kHz rate, that are then put into the DAC.

The sample program decodes the parameters and puts them into the right RAM locations at the proper time. It also starts, stops, and controls the synthesizer.

The parameters are stored in a coded form. Each parameter is given a specific number of bits. In the TSP5220 format, the parameters are calculated and coded every 200 samples. For a 10-kHz sampling rate, this corresponds to 20 milliseconds per frame. Different frame types are used for different circumstances. The TSP5220 decoding scheme is shown in Figure 6.2.

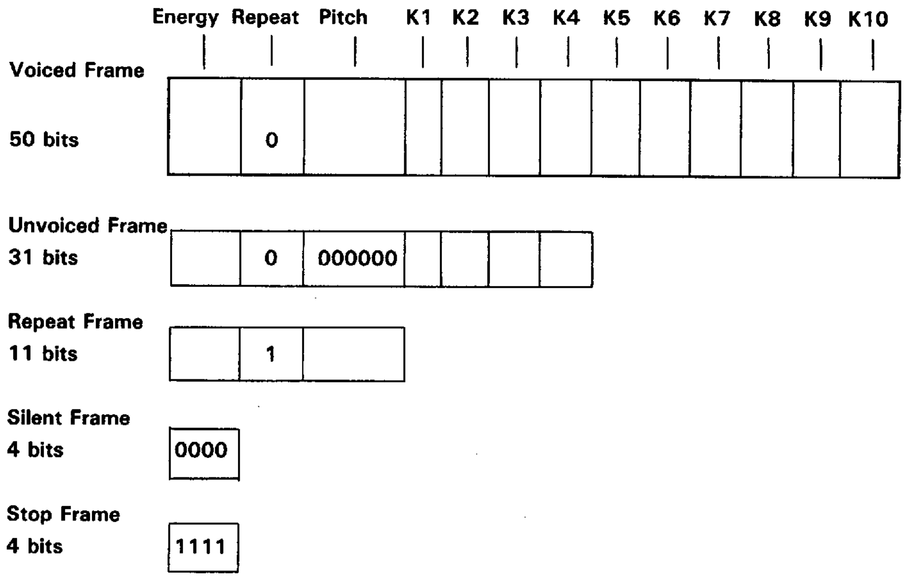


Figure 6-1. TSP5220 Frame Decoding

The longest frame type is the voiced frame. This provides all LPC-10 parameters. For unvoiced frames, which are indicated by a pitch of 0, only the pitch, energy, and first four K parameters are provided. The repeat frame is indicated by a repeat bit value of 1 and provides only pitch and energy. It is used in situations in which the vocal tract resonance is changing very slowly and pitch and energy are varying. Long vowels are a good example of this.

The silent frame is indicated by an energy value of 0 and is used for the parts of speech that are silent.

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.

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.

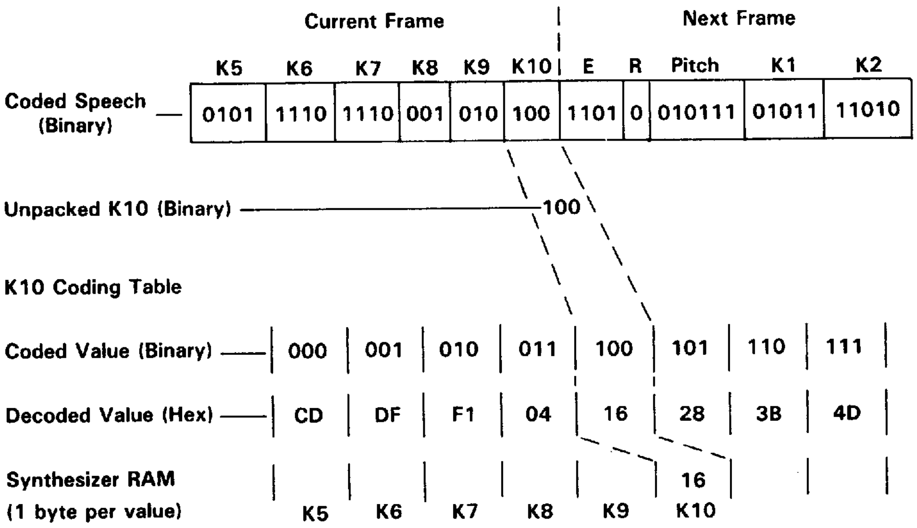


Figure 6-2. Speech Parameter Unpacking and Decoding

To decode speech, the processor must do the following three things:

1. Determine the frame type.
2. Unpack each parameter.
3. Decode each parameter using that parameter's coding table.

The specific details of these operations are given in subsection 6.1.4 — Program Overview. The processor is also required to decide if each frame should be interpolated. Interpolation is used to smooth out the transitions between frames. Most of the time, speech changes smoothly. If 20-ms frames are used without interpolation, changes occur abruptly and the speech sounds rough. The TSP50C4X devices provide automatic interpolation of parameters.

This means that the energy, pitch, and ten K parameters will change several times during the frame so that there is a smooth transition from the previous frame to the current one. Refer to subsection 2.22.3 for more information on this process.

Sometimes, speech changes quickly. For instance, in the case of voicing transitions, speech changes rapidly from voiced to unvoiced or vice versa. The sample program disables interpolation at voicing transitions.

6.1.2 RAM Usage

The sample program uses 8-bit-wide RAM locations 0 to 28 (hex), as well as 4-bit-wide locations from 80 to 87 (hex).

Some RAM locations used in the program are fixed by the architecture of the TSP50C4X, such as present and new values of energy, pitch, and K parameters. The most significant eight bits of the various parameters are put in value locations. In addition, there are fractional value locations that will take four bits more for energy, pitch, and K1 through K6. This permits these parameters to be entered with more precision if desired. Table 6-1 lists the names and addresses for the synthesizer RAM locations. All addresses are in hexadecimal, and the names in parentheses are the labels used in the sample program.

Table 6-1. Synthesizer RAM Addresses

00	Pitch new value (PNV)	01	Pitch present value (PPV)
02	Energy new value (ENV)	03	Energy present value (EPV)
04	K1 new value (K1NV)	05	K1 present value (K1PV)
06	K2 new value (K2NV)	07	K2 present value (K2PV)
08	K3 new value (K3NV)	09	K3 present value (K3PV)
0A	K4 new value (K4NV)	0B	K4 present value (K4PV)
0C	K5 new value (K5NV)	0D	K5 present value (K5PV)
0E	K6 new value (K6NV)	0F	K6 present value (K6PV)
10	K7 new value (K7NV)	11	K7 present value (K7PV)
12	K8 new value (K8NV)	13	K8 present value (K8PV)
14	K9 new value (K9NV)	15	K9 present value (K9PV)
16	K10 new value (K10NV)	17	K10 present value (K10PV)

Fractional values (lower four bits of the RAM location only)

80	Pitch new fraction (FPNV)	81	Pitch present fraction (FPPV)
82	Energy new fraction (FENV)	83	Energy present fraction (FEPV)
84	K1 new fraction (FK1NV)	85	K1 present fraction (FK1PV)
86	K2 new fraction (FK2NV)	87	K2 present fraction (FK2PV)
88	K3 new fraction (FK3NV)	89	K3 present fraction (FK3PV)
8A	K4 new fraction (FK4NV)	8B	K4 present fraction (FK4PV)
8C	K5 new fraction (FK5NV)	8D	K5 present fraction (FK5PV)
8E	K6 new fraction (FK6NV)	8F	K6 present fraction (FK6PV)

Note: This sample program does not use fractions for pitch and K3 to K6.

If interpolation is enabled, the TSP50C4X devices interpolate from the present values to the new values during the frame. If interpolation is not enabled, the present values are used for the entire frame. When the synthesizer is active, each interrupt causes an automatic context switch. The present values and the new values exchange positions with each interrupt. As will be seen in subsection 6.1.4, this eases the programming task considerably. The new values automatically become the present values. This way, each frame can be put into the new values location as it comes up.

The synthesis program also requires RAM for buffering and control flags. These can be stored anywhere in RAM other than the locations mentioned above. In the sample program given here, all the parameters for a single frame are buffered in RAM, along with fractional data for energy, K1, and K2. In addition, one byte of control and status flags and one temporary storage location are used. Table 6-2 shows the addresses and describes the flags.

Table 6-2. Buffer and Control RAM Usage

18	Pitch Buffer (PBF)	19	Energy Buffer (EBF)
1A	K1 Buffer (K1BF)	1B	K2 Buffer (K2BF)
1C	K3 Buffer (K3BF)	1D	K4 Buffer (K4BF)
1E	K5 Buffer (K5BF)	1F	K6 Buffer (K6BF)
20	K7 Buffer (K7BF)	21	K8 Buffer (K8BF)
22	K9 Buffer (K9BF)	23	K10 Buffer (K10BF)
24	Fractional Energy Buffer (FEBF)	25	Fractional K1 Buffer (FK1BF)
26	Fractional K2 Buffer (FK2BF)		
27	Temporary Storage (TEMP)		
28	Status and Control Flags (FLAGS)		

Flag Descriptions

Bit	Description
0	Stop frame detected (STP)
1	Interrupt – set by interrupt routine, cleared by decoding routine (INT)
2	No Interpolation – set if interpolation is not desired for frame (NINTP)
3	Unvoiced – set if frame is unvoiced (UNVO)
4	Start – set while program is decoding the first two frames (STRT)
5	Start 1 – set while the second frame is being decoded (STRT1)
6	Repeat – set if current frame is a repeat frame (RPT)
7	Stop 1 – set for second frame of stop sequence (STP1)

The uses of the buffer and flags will be described in detail in subsection 6.1.4. Note that locations 29 to 7F (hex) are available for user programs while the TSP50C4X devices are synthesizing speech. This is a total of 87 (decimal) bytes for user programming. When the TSP50C4X devices are not synthesizing speech, all 128 bytes, plus the 16 nibbles from 80 to 8F, are available for user programming.

6.1.3 ROM Usage

The sample program uses ROM locations 0 to 344 (hex), leaving the space from 345 to 1FFF or 3FFF (hex) available for user programs, speech, and data. Table 6-3 gives a breakdown of the ROM usage.

Table 6-3. ROM Usage

0000-0001	Power-up short branches
0002-0003	Interrupt short branches
0004-0005	Power-up branch
0006-0007	Interrupt branch
0008-014E	Speech decoding tables
014F-0158	Processor initialization subroutine ($\overline{\text{INIT}}$)
0159-0161	RAM clear subroutine (RAM0)
0162-0172	Initialization of processor, speech address register and parallel-to-serial register
0173-02CB	Speech synthesis subroutine
02CC-0344	Speech synthesis interrupt routine

Programs can be moved anywhere in program memory, but the speech decoding tables must remain below address 400 (hex) so that they can be addressed by the ten bits of the A register using the LUAA instruction.

6.1.4 Program Overview

The sample synthesis program is reproduced in its entirety in Appendix B. Parts of it are used in this section for explanatory purposes. An outline of the program flow follows:

```
Initialize processor
Initialize speech address register and parallel-to-serial register
Decode first frame of speech
Place first frame in present value RAM locations
Decode second frame of speech
Place second frame in new value RAM locations
Start synthesizer
Until stop frame reached.
Decode each frame
When interrupt occurs, copy frame to new value RAM locations
Clear all new value locations, except pitch
Wait two frames, then stop synthesizer
Return to calling routine.
```

6.1.5 Calling the Synthesis Program

The following sample program can be used to invoke the synthesis program. The program starts at ROM location 0 on power-up.

```
                *---------------------------------------------
                * BEGINNING OF PROGRAM
                *---------------------------------------------
0000                    AORG    #0000
0000  84                SBR     GOGO     ;POWER UP VECTOR
0001  84        SBR     GOGO
0002  86                SBR     INTO     ;INTERRUPT VECTOR
0003  86                SBR     INTO
0004  61        GOGO    BR      GO       ;BRANCH TO SPEECH ROUTINE
0005  5F
0006  62        INTO    BR      INT1     ;BRANCH TO INTERUPT ROUTINE
0007  C3
```

There are two short branches at the beginning because the condition of the status bit is unknown. Two branches ensure that at least one will be taken. Both branches go to label GOGO, which has a branch to the start of the actual program. The other branches in this code fragment are for the interrupt routine.

Next, the program goes to the initialization section:

```
                * START OF SYNTHESIS PROGRAM
                *---------------------------------------------
015F  71        GO      CALL    INIT     ;INITIALIZE PROCESSOR
0160  4F
0161  71                CALL    RAMO     ;CLEAR INTERNAL RAM
0162  56
0163  2E                RSECT            ;MAKE TIMER INPUT INTERNAL
```

First the INIT routine is called:

```
                * INIT: INITIALIZE PROCESSOR
                *+-------------------------------------------+
014F  1D        INIT    INTD             ;DISABLE INTERRUPTS
0150  1B                STOP             ;STOP SYNTHESIZER
0151  00                CLA
0152  50                ACAA    49       ;200 SAMPLES/FR (200/4) - 1
0153  31
0154  5D                TAPSC            ;INTO PRESCALE REGISTER
0155  1F                RETN
```

INIT disables interrupts, stops the synthesizer and sets the frame rate. It is important to disable interrupts because the initialization code is not designed to be interrupted. The STOP instruction is not used to stop the synthesizer. It is used to initialize the context switching bit. For more information, see subsection 6-7. The prescale register determines the length of the speech frame, which is 200 samples in this case. The formula for the prescale register value is (samples per frame/4) − 1.

The speech synthesis routine then calls the RAMO subroutine.

```
               *       RAM CLEAR
               *
               *       SUBROUTINE NAME : RAMO
               *       USES : A REGISTER, X REGISTER, ALL OF RAM
               *       DESCRIPTION : FILLS RAM WITH ZEROES
               *
               *-----------------------------------------------
0156  00  RAMO     CLA                ;CLEAR ACCUMULATOR
0157  11           CLX                ;POINT TO FIRST RAM LOCATION
0158  09  RC1      TAM                ;CLEAR RAM LOCATION
0159  0F           IXC                ;POINT TO NEXT RAM LOCATION
015A  55           XGEC     #90       ;AT END OF RAM?
015B  90
015C  DE           SBR      RC2       ;BRANCH IF SO
015D  D8           SBR      RC1
015E  1F  RC2      RETN
```

RAMO clears all memory locations. It should not be utilized in speech synthesis routine because it will clear any nonspeech data also. It should be used at power-up for initialization, since the RAM in the TSP50C4X devices powers up in a random state. For this synthesis program to function properly, the following locations need to be cleared; FPNV, FPPV, FK3NV, FK3PV, FK4NV, FK4PV, FK5NV, FK5PV, FK6NV, FK6PV.

Next, the program initializes the speech data paths and pointers:

```
0421 0163 2E          RSECT             ;MAKE TIMER INPUT INTERNAL
0422            *
0423 0164 00          CLA
0424 0165 50          ACAA     #08      ;HIGH 5 BITS OF SPEECH ADDRESS
     0166 08
0425 0167 0B          TASH              ;INTO HIGH BITS OF SAR
0426 0168 00          CLA               ;LOW BYTE OF ADDRESS = 0
0427 0169 0A          TASL              ;INTO LOW BYTE OF SAR
0428            *
0429 016A 2C          INTRM             ;USE INTERNAL ROM
0430 016B 59          LUSPS             ;INITIALIZE PARALLEL TO SERIAL
                                        ;REG
0431 016C 71          CALL     SPSTR    ;SPEAK
     016D 70
0432            *
0433 016E 61    LOOP  BR       LOOP     ;LOOP FOREVER
     016F 6E
```

The RSECT instruction tells the processor that the timer clock will be taken from the internal ROM. The CLA, ACAA 8, and TASH instructions initialize the high bits of the speech address register to 8. The CLA and TASL instructions put a 0 in the low byte of the speech address register. This prepares the program to start accessing speech from internal ROM at location 0800 (hex). INTRM instruction sets the data path to internal ROM rather than TSP60CXX external ROM interface and LUSPS initializes the parallel-to-serial register.

Finally, the speech subroutine SPSTR is called. Generally, user programs have a more complicated method of looking up speech start addresses. Often, there are three levels of pointers:

1. Sentence pointers that point to the start addresses.
2. Lists of word numbers that make up a sentence. The word numbers are the pointers.
3. Start addresses of speech data for each word.

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 or LUAA instructions. The structure is dependent on the specific application.

6.1.6 Synthesis Program Walkthrough

There is a short initialization section at the beginning of SPSTR:

```
0434                *
0435                *-----------------------------------------------
0436                * SPEECH
0437                *
0438                * SUBROUTINE NAME : SPSTR
0439                * USES : A REGISTER, X REGISTER,
                    *                         B REGISTER, RAM FROM 0 TO #29
0440                * RAM FROM #80 TO #8F
0441                * 2 STACK LEVELS
0442                * DESCRIPTION : SYNTHESIZES SPEECH
0443                *
0444                *-----------------------------------------------
0445 0170 56              SPSTR   TCX     PBF
     0171 18
0446 0172 00              CLA
0447 0173 50              ACAA    #0C
     0174 0C
0448 0175 09              TAM             ;SOME PITCH TO START OUT
                                          ;WITH, IN CASE WE
0449                *                     ;GET A SILENT FRAME
0450 0176 56              TCX     FLAGS   ;
     0177 28
0451 0178 00              CLA             ;CLEAR FLAGS
0452 0179 09              TAM             ;
0453 017A 3C              SBITM   STRT    ;FLAG START OF SPEECH
0454 017B 61              BR      SPDE2   ;BRANCH AROUND INTERRUPT
     017C 8B                              ;CHECK
```

Lines 0445 to 0455 put a default value in the pitch buffer and set the flags to indicate that the first frame is being processed. The pitch buffer must be loaded with a default value because values from 0 to B will cause the synthesizer to lock up and not speak. Usually, the pitch is loaded from the first frame, but if the first frame is silent, there is no pitch value to load. The last instruction in this section is a branch around the section of code that waits for an interrupt to occur. Interrupts are not enabled yet. The code now enters the speech decoding section. The same decoding section is used for all frames, for initialization and for continuous speaking.

```
0473 018B 00  SPDE2   CLA              ;CLEAR A
0474 018C 23          GET     NRGNB    ;GET ENERGY
0475 018D 54          ANEC    ESTOP    ;IS IT THE STOP CODE?
     018E 0F
0476 018F 61          BR      NOSTP    ;BRANCH IF NOT
     0190 99
0477 0191 56          TCX     FLAGS
     0192 28
0478              *
0479 0193 3F          SBITM   STP1     ;FLAG STOP1
0480              *
0481 0194 62          BR      CLRPR    ;GO CLEAR PARAMETERS
     0195 5A
```

Lines 0473 to 0482 get the coded energy (GET NRGNB) and check for a stop frame (ANEC ESTOP). If it is a stop frame, a flag is set to branch to clear all parameters except pitch. If it isn't a stop frame, branch is to NOSTP:

```
0486 0199 54  NOSTP   ANEC    ESILE    IS IT A SILENT FRAME?
     019A 00
0487 019B 61          BR      NOSIL    ;BRANCH IF NOT
     019C 9F
0488 019D 62          BR      CLRPR    ;IF SO, GO CLEAR PARAMETERS
     019E 5A
```

At NOSTP the code checks for a silent frame and branches either to NOSIL to process a nonsilent frame, or to CLRPR to clear the parameters for a silent frame. At NOSIL the energy is decoded:

```
019F 56   NOSIL   TCX     TEMP
01A0 27
01A1 09           TAM               ;SAVE ENERGY CODE
01A2 50           ACAA    TABEN     ;START OF ENERGY TABLE
01A3 08
01A4 58           LUAA              ;DECODE ENERGY
01A5 56   TCX     EBF
01A6 19
01A7 09           TAM               ;PUT IT AWAY
01A8 56           TCX     TEMP
01A9 27
01AA 04           TMA               ;GET ENERGY CODE BACK
01AB 50           ACAA    TABEF     ;START OF FRACTIONAL ENERGY
                                    ;TABLE
01AC FF
01AD 58           LUAA              ;DECODE ENERGY
01AE 56           TCX     FEBF
01AF 24
01B0 09           TAM               ;PUT IT AWAY
```

This code fragment illustrates the decoding process for all the parameters. The coded value is saved in temporary storage and retained in the accumulator. The start of the energy decoding table is then added to the coded value, yielding a pointer to the location of the decoded value. The LUAA instruction is then used to transfer the decoded value to the A register and the value is placed in the energy buffer. Then the process is repeated with the fractional decoding table and buffer. If fractional pitch or additional fractional K parameters are desired, the same process is used.

The repeat bit follows the energy parameter:

```
          * REPEAT
          *
01B1 00         CLA              ;CLEAR A
01B2 20         GET     RPTNB    ;GET REPEAT BIT
01B3 56         TCX     FLAGS
01B4 28
01B5 4E         RBITM   RPT      ;SET BIT REPEAT
01B6 54         ANEC    REPT     ;IS IT REPEAT?
01B7 01
01B8 BA         SBR     LABPI    ;GO TO PITCH
01B9 3E         SBITM   RPT      ;IF NOT RESET BIT REPEAT
```

This section gets the repeat bit and sets a repeat (RPT) flag. The program has three more decision points for decoding.

First, it gets the pitch and decodes it and sets a flag to indicate whether pitch is voiced or unvoiced. It also checks the voicing for the previous frame and sets a bit disabling interpolation if the voicing has changed.

Next, it checks the RPT bit set earlier to see if this is a repeat frame. If it is, it branches around the K parameter decoding section. If it is not a repeat frame, it decodes the first of the four K parameters.

Finally, if the frame is unvoiced, it branches to a routine that clears the last six K parameters. If the frame is voiced, it decodes and stores the last six K parameters.

Then there are some decision points that affect initialization:

```
0671 0286 56  SPCEX   TCX     FLAGS
     0287 28
0672 0288 44          TBITM   STRT     ;FIRST TWO FRAMES?
0673 0289 8C          SBR     SPCE1    ;BRANCH IF SO
0674 028A 61          BR      SPDEC    ;GO DO IT ALL OVER AGAIN
     028B 7D
0675 028C 45  SPCE1   TBITM   STRT1    ;SECOND FRAME?
0676 028D 62          BR      SPCE2
     028E B3
0677 028F 3D          SBITM   STRT1    ;NEXT ONE IS SECOND FRAME
```

This section of code checks the start flags and goes to SPDEC if the initialization was already done, to SPCE2 if the second frame is done, or falls through if the first frame has just been decoded and buffered.

If the first frame is waiting in the buffer, it is put into the RAM location of the present value. This is the only frame put in these locations. The TSP50C4X devices usually interpolate from the present values to the new values for each frame. On startup, the synthesizer will interpolate from the first frame to the second frame. Then the second frame will be automatically switched to the present value locations and the program will put the third frame into the new values location and so on.

After the first frame is copied into the present values location, the program branches back to the start and the second frame is decoded. The code section above is re-entered and the program goes to SPCE2. From there, the program enters the interrupt routine at a second entry point and leaves it at a special exit point. Even though this is not good structured programming, it is done to save ROM and stack space. The interrupt routine body does what needs to be done; it copies the buffer to the new values.

After the second frame has been copied to the new values, interrupts are started:

```
0716              *
0717              *NOW START IT UP
0718              *
0719 02B5 56  SPCE3   TCX     FLAGS   ;BACK HERE FROM MOVE
     02B6 28
0720 02B7 4C          RBITM   STRT    ;CLEAR STARTUP FLAGS
0721 02B8 4D          RBITM   STRT1
0722 02B9 56          TCX     PBF     ;GET PITCH
     02BA 18
0723 02BB 04          TMA             ;INTO A REGISTER
0724 02BC 56          TCX     TMVAL   ;TIME INTO X REGISTER
     02BD 1F
0725 02BE 1A          START
0726 02BF 10          TXTM            ;START THINGS GOING
0727 02C0 1E          INTE            ;ENABLE THOSE INTERRUPTS
0728 02C1 61          BR      SPDE2   ;GO FILL BUFFER AGAIN
     02C2 8B
```

Lines 0716 to 0729 clear the startup flags, load the pitch into the A register, and start the synthesizer. The timer register is then loaded and interrupts enabled. A 1F must always be loaded into the timer register for speech interrupts. It is hard-wired into the synthesizer logic that will be used by this value.

After this, the program branches to SPDE2, which will decode the third frame and then go back to the beginning to wait for indications that an interrupt has occured. Then it will buffer up another frame and another, and another, and so on, until a stop frame is encountered. Before looking at stop frame handling, it is important to examine the interrupt routine in more detail.

```
0744 02C3 56  INT1    TCX     FLAGS
     02C4 28
0745 02C5 39          SBITM   INT      ;SET INTERRUPT FLAG
0746 02C6 40          TBITM   STP      ;LAST FRAME?
0747 02C7 63          BR      NOINT    ;BRANCH IF SO.
     02C8 37
0748 02C9 56          TCX     TMVAL    ;LOAD TIMER INTERRUPT VALUE
     02CA 1F
0749 02CB 10          TXTM
0750 02CC 56          TCX     FLAGS
```

The first section of the interrupt program tests the STP flag to see if this is the last frame of synthesis. If it is, the timer register is not reloaded. This way, there will be no interrupt pending when the speech synthesis program is restarted for the next speech. If speech is ongoing, the timer register is reloaded to start timing for the next interrupt. The INT flag is set to indicate that the interrupt has occured.

Then voicing is handled:

```
0750 02CC 56          TCX     FLAGS
     02CD 28
0751          *
0752          *MOVE NEW VALUES INTO PRESENT VALUES
0753          *
0754          *VOICING
0755          *
0756 02CE 00  T$IR1   CLA
0757 02CF 43          TBITM   UNVO     ;CHECK VOICING
0758 02D0 62          BR      SSSSS    ;BRANCH IF NOT VOICED
     02D1 D4
0759 02D2 50          ACAA    1        ;SET VOICING BIT
     02D3 01
0760 02D4 5B          SSSSS   TAV      ;PUT IT AWAY
```

This section tests the voicing bit and does a TAV instruction with the least significant bit of the A Register set or reset accordingly. Next, all the parameters are copied from the buffer to the new values. The code for pitch is sufficient to illustrate this:

```
02D5 56          TCX      PBF      ;GET PITCH BUFFER
02D6 18
02D7 04          TMA
02D8 56          TCX      PNV      ;TELL CHIP ABOUT IT
02D9 00
           *
02DA 09          TAM
```

The code to copy parameters from buffer to new value is in a straight line, while the code used for initialization to code buffers to present values is in a loop. This is because the priorities are different. The initialization is done before speech starts; thus ROM space is the overriding criterion. The interrupt routine occurs during speech synthesis; therefore, time is the most important factor.

During synthesis, the synthesizer interpolates from the present values to the new values. When interrupt occurs, interpolation to the new values has already taken place. The context switch toggles and the new values and present values change positions. The synthesizer stops interpolating and uses the parameters that are now in the present value location (the old new values). The speech parameters stay constant until the program loads the values for the next frame into the new values locations and tells the synthesizer to start interpolation again. For this reason, it is important to load the new values as fast as possible. Any slowdown would have an adverse effect on the quality of speech.

Interpolation is enabled by the INTE instruction. The program uses the NINTP flag to decide whether interpolation is required or not. The program fragment is shown below:

```
           *INTERPOLATION
           *
032F 56          TCX      FLAGS
0330 28
0331 44          TBITM    STRT     ;START?
0332 62          BR       SPCE3    ;BRANCH IF SO
0333 B5
0334 42          TBITM    NINTP    ;INTERPOLATE?
0335 B7          SBR      NOINT    ;BRANCH IF NOT
0336 1E          INTE
0337 2F          NOINT    RETI
```

The first test above is for the STRT flag. This provides for the nonstructured exit from the interrupt routine for initialization.

The only section requiring more explanation is the stop code handling. When the stop code is detected, all the parameters in the buffer are cleared except for the pitch, which is left the same. The STP1 flag is also set. On the next interrupt, the program will put zeroes into the new values locations. The synthesizer will then interpolate down to all zero values for the next frame. The STP flag will be set so that the final interrupt will not reload the interrupt timer causing interrupts to cease. The synthesis program will go to its exit point:

```
0843              *HERE FOR A STOP CODE
0844              *
0845  0338  1D    STPIT    INTD
0846  0339  1B             STOP
0847  033A  1F             RETN
```

It will disable interrupts, stop the synthesizer, and return control to the calling routine.

6.2 Arithmetic Modes

The instructions transferring data to A register, AMAAC and SMAAN, are the only instructions affected by the ALU mode. The ALU mode can be set using the INTGR and EXTSG instructions.

Instructions transferring data to the A register clear bits eight and nine of the A register in the INTGR mode. In the EXTSG mode, bits eight and nine are set to the value of bit seven.

AMAAC and SMAAN set the Status Flag to the value of the carry from bit 7 of the ALU. The ALU accepts two 10-bit input values and returns a 10-bit output value. One 10-bit value comes from the A register, and the low eight bits of the other value come from the RAM location pointed to by the X register. The ALU mode affects what goes into the two high bits of that value. If the INTGR instruction has been executed last, the two bits are always filled by zeroes. The EXTSG instruction causes the high bit from the RAM to be put into the two high bits. This means that the only difference between the two modes occurs when the RAM most significant bit is a 1.

The INTGR mode is intended to be used for unsigned numbers, while the EXTSG mode allows two's complement numbers to be used. The example in Figure 6-5 illustrates the difference. Note that in integer mode, the FF (hex) acts as a positive number and FF is added to the value in the A register. In extended sign mode, the FF is treated as a two's complement number, giving it an effective value of −1. The example is for the AMAAC instruction, but the same rules hold true for SMAAN.

```
CARRY             1 1111 11          INTEGER MODE
A-reg      202   10 0000 0010
*X-reg      FF   00 1111 1111
                 ------------
RESULT     301   11 0000 0001     1 → SF

CARRY            11 1111 11          EXTENDED SIGN
                                     MODE
A-reg      202   10 0000 0010
*X-reg      FF   11 1111 1111
                 ------------
RESULT     201   10 0000 0001     1 → SF
```

Figure 6-3. ALU Modes

6.3 Standby Mode

Several design details must be taken care of to achieve minimum power consumption in the standby mode, whether it is reached with the SETOFF instruction or by pulling the INIT line low.

1. Any of the pull-up resistors, internal or external, will continue to draw power when the TSP50C4X devices are in standby mode. The design must make sure that there is no path to ground when the chips are powered down.
2. Port C should be tied to either a low or a high level. If its inputs go to an intermediate state, internal gates could be energized, causing additional power consumption.
3. Attention must be paid to the levels to which signals go when the TSP50C4X devices are in the standby mode.

Port A	— high impedance
Port B	— output mode, low-level
Port C	— high impedance
Port D	— outputs, all low-level
$\overline{\text{IRT}}$	— When masked as output — low, otherwise high-impedance.

6.4 Slave Option

The slave option is a specialized mask-selectable mode of the TSP50C4X devices that is used for applications in which the TSP50C4X devices need to be controlled by a master microprocessor. Port A changes to a latched port that can be used as an I/O or memory address by the master processor. Several lines from Port C become control and handshake lines for this port. The slave option is designed for use with 4- and 8-bit data widths.

The lines involved in the slave option are:

PA0 to PA7	— I/O port
$\overline{\text{ENA1}}$	— chip enable input for reading and writing to the high four bits of PA
$\overline{\text{ENA2}}$	— chip enable input for reading and writing to the low four bits of PA
R/$\overline{\text{W}}$	— input that controls direction of data flow for PA
$\overline{\text{RDY}}$	— output that indicates whether PA is ready to be written to.
$\overline{\text{IRT}}$	— output that indicates that there is data on PA to be read.

Here is a typical sequence for an 8-bit read operation (see Figure 6-4):

At the beginning of the operation, the master is holding $\overline{\text{ENA1}}$ and $\overline{\text{ENA2}}$ high, which causes Port A to remain in a high-impedance state. The TSP50C4X device is holding $\overline{\text{IRT}}$ high. The other lines do not matter yet.

1. The TSP50C4X device puts data in the Port A latch with a TAPA instruction. This automatically causes the $\overline{\text{IRT}}$ line to go low.
2. The master either polls or is interrupted by the $\overline{\text{IRT}}$ line.
3. The master raises the R/$\overline{\text{W}}$ line.
4. The master lowers $\overline{\text{ENA1}}$ and $\overline{\text{ENA2}}$. This causes step 5 to occur.
5. The contents of the Port A latch appear on the outputs of the port and are read in by the master. The $\overline{\text{IRT}}$ line goes high.
6. The master raises $\overline{\text{ENA1}}$ and $\overline{\text{ENA2}}$. This causes Port A to return to a high-impedance state. To ensure that $\overline{\text{IRT}}$ will stay high, this event must occur at least 20 clock cycles after the falling edge of $\overline{\text{IRT}}$.
7. The TSP50C4X device polls the status of IRT with the TPCA instruction. When $\overline{\text{IRT}}$ goes high, it gets another byte of data and puts it into the Port A latch, starting the cycle over again.

Note that $\overline{\text{IRT}}$ goes high after the falling edge of $\overline{\text{ENA1}}$. This means that if the TSP50C4X device polls $\overline{\text{IRT}}$ while the master is still reading, it will see a high and may overwrite the Port A latch, which could lose the existing data. If this is a possiblity, ensure that there is enough delay between polling $\overline{\text{IRT}}$ and writing to the latch to prevent this from occurring.

For a 4-bit master, follow the same sequence, reading the low nibble first, followed by the high nibble. $\overline{\text{ENA}}$2, which controls the low nibble, has no effect on $\overline{\text{IRT}}$, so it will not be set until the high nibble is read with $\overline{\text{ENA}}$2.

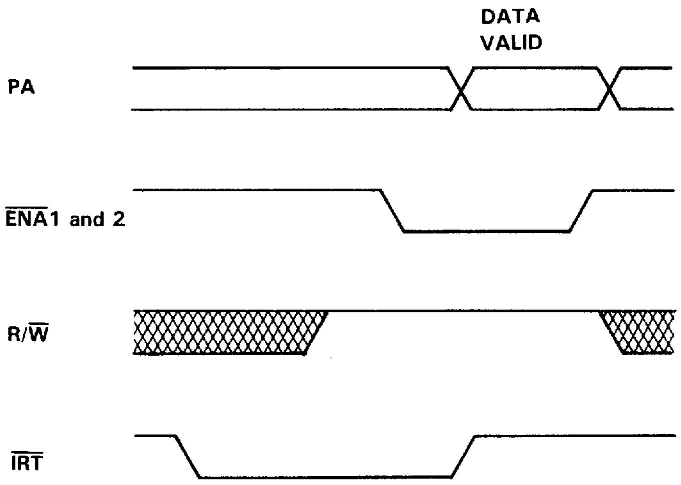


Figure 6-4. Read Operation

A write operation is similar except that RDY is used and R/$\overline{\text{W}}$ is in the opposite state. Here is the sequence for a write (see Figure 6-5):

At the beginning of the operation, the master holds $\overline{\text{ENA}}$1 and $\overline{\text{ENA}}$2 high, which causes Port A to remain in a high-impedance state. The TSP50C4X device holds $\overline{\text{RDY}}$ high. The other lines do not matter yet.

1. The TSP50C4X device completes processing any previous data it has and then executes a RSRDY instruction, which causes the $\overline{\text{RDY}}$ line to go low.
2. The master either polls or is interrupted by the $\overline{\text{RDY}}$ line.
3. The master lowers the R/$\overline{\text{W}}$ line.
4. The master lowers $\overline{\text{ENA}}$1 and $\overline{\text{ENA}}$2. This causes step 5 to occur.
5. The data on the inputs of Port A are placed into the Port A latch. The $\overline{\text{RDY}}$ line goes high.

6. The master raises $\overline{\text{ENA1}}$ and $\overline{\text{ENA2}}$. This causes the data on Port A to be latched into the Port A latch. To ensure that $\overline{\text{RDY}}$ will stay high, this event must occur at least 20 clock cycles after the falling edge of $\overline{\text{RDY}}$.

7. The TSP50C4X device polls the status of $\overline{\text{RDY}}$ with the TPCA instruction. When $\overline{\text{RDY}}$ goes high, it gets the byte of data from the Port A latch and then issues an RSRDY instruction, starting the cycle over again.

Note that $\overline{\text{RDY}}$ goes high after the falling edge of $\overline{\text{ENA2}}$. This means that if the TSP50C4X device polls $\overline{\text{RDY}}$ while the master is still writing, it will see a high and may read from the Port A latch. This could lead to false data. If this is a possibilty, ensure that there is enough delay between polling RDY and reading from the latch to prevent this from occuring.

For a 4-bit master, follow the same sequence, writing the high nibble first, followed by the low nibble. $\overline{\text{ENA1}}$, which controls the high nibble, has no effect on RDY, so it will not be set until the low nibble is read with $\overline{\text{ENA2}}$.

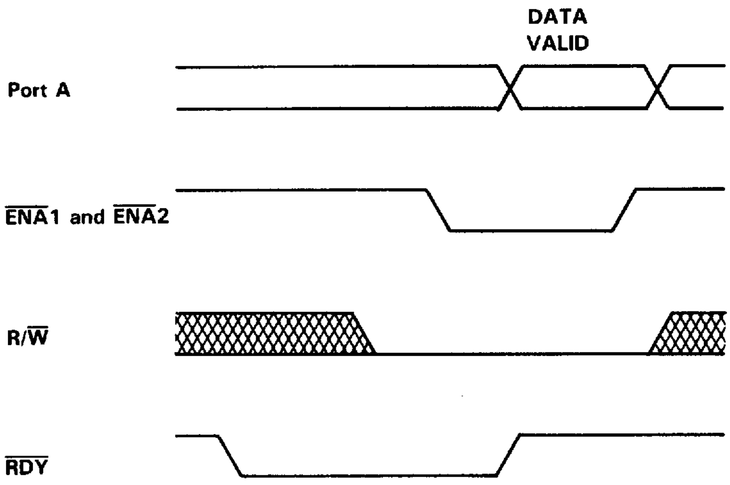


Figure 6-5. Write Operation

6.5 TSP60CXX Interface

If additional space for speech or other data is desired, the TSP50C4X can easily be interfaced to the TSP60C19, the TSP60C20, and the TSP60C80. Port B has a special mode activated by the EXTRM instruction that enables the GET instruction to be used with external memory just as it is with internal data. However, the initialization is completely different, and that is what is covered in this section. There are special precautions that must be taken when using internal (TSP50C4X) data after using the external (TSP60CXX) data. It is also possible to put the TSP60CXX devices into a low-power mode from the TSP50C4X devices.

The TSP60CXX devices supports an eight-line interface. There are two mode pins, M0 and M1. The data address is entered 4 bits at a time on pins ADD1, ADD2, ADD4, and ADD8. The data emerge serially on SRDTD (SeRial DaTa Delayed), which should be connected to the RDIN pin of the TSP50C4X devices, clocked by ROMCLK. See below.

6.5.1 Initialization

The TSP50C4X to TSP60CXX interface program needs to initialize the interface sections of both devices and to load the proper address into the TSP60CXX address register. After that, the actual data transfer is completely

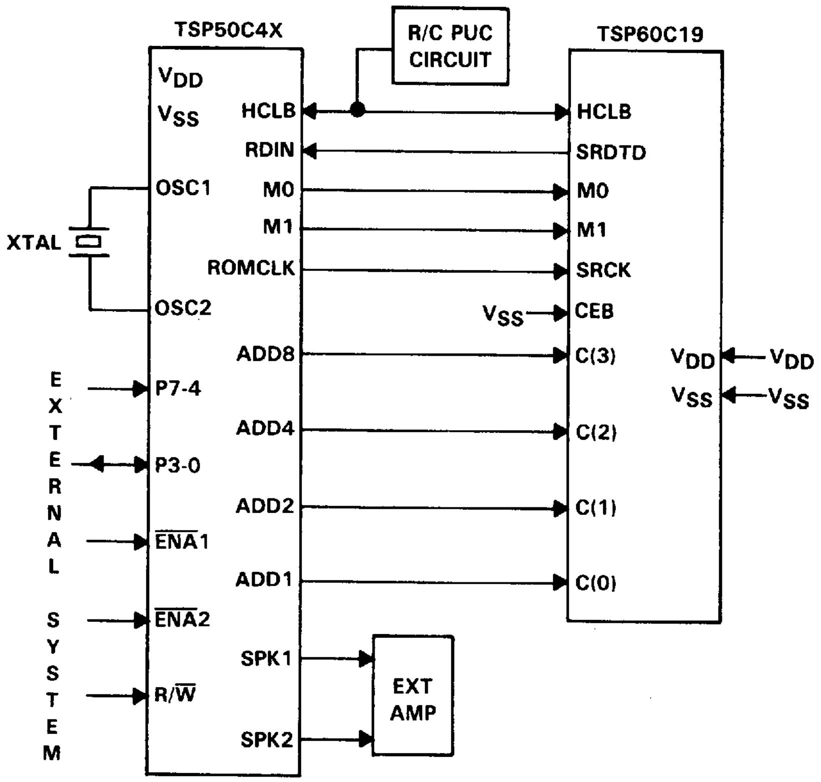


Figure 6-6. TSP0C4X and TSP60C19 Interface

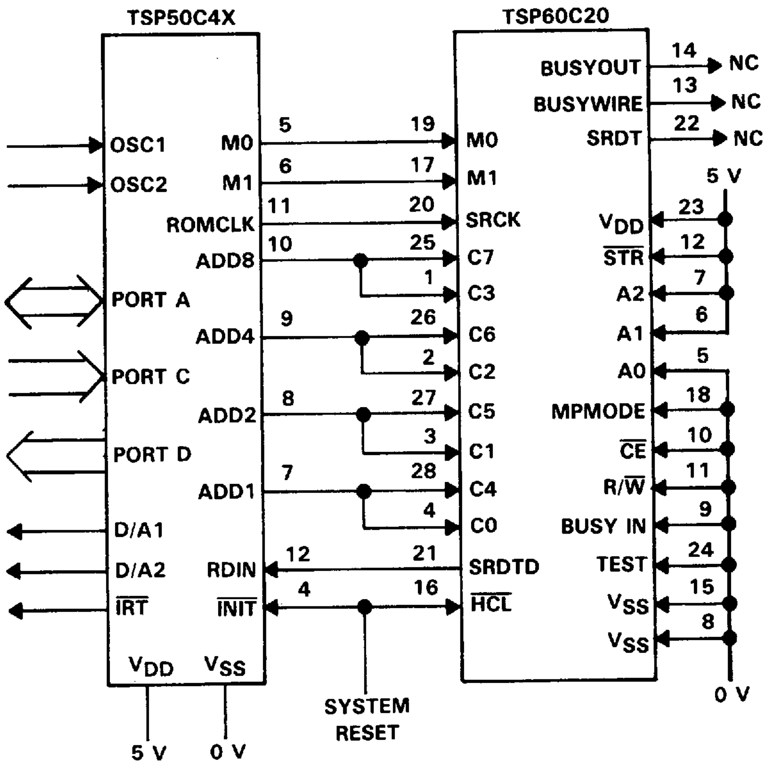


Figure 6-7. TSP50C4X/TSP60C20 Interface with TSP50C4X in the External ROM Mode

automatic and the standard speech synthesis routine can be used. The only restriction that applies is that each GET instruction must be at least eight instruction cycles after the previous GET because of the speed of the interface.

The address counter on the TSP60CXX devices has 16 bits and addresses the data on 16-bit boundaries. Therefore, it can address 65536 16-bit words. Each TSP60C19 or TSP60C20 contains only 16384 words, which can be addressed with 14 bits. The highest two bits are mask programmable as chip select bits so that up to four TSP60C19s or TSP60C20s can be used on the same eight-line interface. The TSP60C80 has four times as much data as the other TSP60CXX chips, so only one can be used with eight lines. For more information, see the TSP60CXX Data Manual.

Even though the TSP60CXX devices address data on 16-bit boundaries, speech data is customarily packed on 8-bit boundaries in order to get more data compression. To access data starting in the middle of the 16-bit TSP60CXX word, simply address the word and then do a GET 8 to get to the second byte. This approach is used in the program shown here.

The software accepts a 16-bit address of data divided up into 8-bit bytes. The high 15 bits of the address are used to address the TSP60CXX device, while the least significant bit is used to determine the need for a GET 8 as described above. Since only 15 bits are used for TSP60CXX address, the program as it is written can only be used to access two TSP60C19s. It can, however, be easily modified to incorporate the additional bit needed to access all four TSP60C19s or an entire TSP60C80.

The initialization sequence is as follows:

1. Pulse M1 high.
2. Pulse M0 high – this sequence resets the TSP60CXX.
3. Load the address.
4. Delay 16 instruction cycles so the TSP60CXX can fetch the data.
5. Read 16 bits from the TSP60Cxx to bring the new data to the output.
6. Read an additional eight bits if necessary.

The program to accomplish this is Appendix C.

6.5.2 Using Internal and External Data Alternately

Port B can be used in the TSP60CXX mode (EXTRM) or in the internal data mode (INTRM). In the TSP60CXX mode, PB7 becomes an input and the TSP60CXX device is activated to transmit data to it. When internal speech is needed, the INTRM instruction must be executed to direct the GET instruction to fetch data from internal ROM. This instruction also makes Port B into standard output port, making PB7 an output. If the TSP60CXX device is not deselected before the INTRM instruction is executed, there will be a possible conflict on PB7 that could damage either device and that will cause a large rise in power consumption.

To prevent this, deselect the TSP60CXX before using internal data. There are two ways to do this. There are several pins on the TSP60CXX that can be used to deselect the device or to put it into a low-power mode. Refer to the TSP60CXX data manual for more information. The other option involves using fewer than four TSP60C19s. If this is done, simply put out the address of the TSP60C19 that is missing. This will cause the other TSP60C19s to make their output pins high-impedance. If a TSP60C80 is being used, it will be necessary to use its chip select line to disable it before executing the INTRM instruction.

6.5.3 TSP60CXX Power Down

A simple sequence can also be used to put the TSP60C19 into a guaranteed low-power state. It is used when powering off the TSP50C4X devices with the SETOFF instruction to ensure that the TSP60C19 will not consume power. Method one involves simply keeping the TSP50C4X device running for 32 instruction cycles after the last GET instruction. Pulsing M1 high and then low will also put the TSP60C19 into a guaranteed power-down state.

6.6 Use of the TMEDA Instruction

The TMEDA instruction permits direct access to the TSP50C4X DAC. The synthesizer must be started in the usual way with the START instruction following the TMEDA instruction. The first value that is put out 11 cycles after the START instruction is executed is always 0. After that, RAM location 0 is put out on the DAC every 12 instruction cycles. For best results, the program that uses TMEDA should put out a value at 12 instruction cycle intervals. The STOP, INTE, and START instructions need to be executed to get the synthesizer into the proper configuration for the TMEDA to work. The program below will generate a 500-Hz sine wave if the clock frequency of the TSP50C4X device is 3.84 MHz.

```
*
*19 STEP SINE WAVE (ONE STEP HANDLED BY ROUTINE)
*(This section must go in the lower 1K of ROM)
*
SINE     BYTE     39
         BYTE     75
         BYTE     103
         BYTE     122
         BYTE     127
         BYTE     122
         BYTE     103
         BYTE     75
         BYTE     39
         BYTE     0
         BYTE     #D9
         BYTE     #B5
         BYTE     #99
         BYTE     #86
         BYTE     #81
         BYTE     #86
         BYTE     #99
         BYTE     #B5
SIEND    BYTE     #D9
*
*Start of program
```

```
*This section can go anywhere in the program portion of
*the ROM
*
BEGIN      STOP
           INTE                   INITIALIZE SYNTHESIZER
           TMEDA
           START                  START SYNTHESIZER IN TMEDA MODE
           SBR        TONE1
*
TONE       TBA                    TRANSFER TABLE POINTER TO A
           ANEC       SIEND+1     AT END OF TABLE?
           SBR        TONE2       BRANCH IF NOT
*
TONE1      TCX        SINE        BACK TO START OF TABLE
           XBX                    POINTER IN B
           CLX                    point at energy location
           CLX                    DELAY
           CLA                    AT END, PUT 0 OUT
           TAM
           SBR        TONE        GO DO IT AGAIN
*
TONE2      LUAA                   GET TABLE VALUE INTO A
           TAM                    AND INTO MEMORY
           TAM                    DELAY
           IBC                    INCREMENT TABLE POINTER
           LUAA                   DELAY
           SBR        TONE
```

6.7 Use of the Timer, Prescaler, Interrupt, and IRT Pin

The timer, prescaler, interrupt, and $\overline{\text{IRT}}$ pin all work together. The only interrupt in the TSP50C4X device is caused by timer underflow. The timer is decremented when the prescaler underflows, and the prescaler is decremented either by an internal clock equal to 1/48 the clock frequency or by an external clock on the $\overline{\text{IRT}}$ pin. The $\overline{\text{IRT}}$ pin can also be used as an output (see subsection 6.3, slave option).

The clock source is selected by the TTMA and RSECT instructions. TTMA selects the $\overline{\text{IRT}}$ pin as the clock source, while RSECT selects the internal source.

The prescale value is set by the TAPSC instruction. The prescale register is decremented once for each clock cycle and then automatically reloaded on an underflow with the value from the last TAPSC. When the prescale register underflows, the timer register is decremented. Since the prescale register is reloaded on underflow, a 0 value in the prescale requires one clock input for each clock output, a one requires two clocks and so on, up to hex FF, which requires 256 clocks.

The timer register is loaded with the TXTM instruction and must be reloaded with TXTM each time a countdown is desired. The timer is decremented by the clock from the prescale register until it underflows, and then it stops and sets an interrupt bit. If interrupts are enabled, the TSP50C4X device immediately vectors to the interrupt routine, starting at ROM address 6. If interrupts are not currently enabled, the bit will remain set until interrupts are enabled or until the TSP50C4X device is reinitialized with the $\overline{\text{INIT}}$ pin.

Interrupt routines should be written with the TXTM very close to the start of the routine to preserve time-keeping accuracy. If the interrupt routine is to be stopped, ensure that the last interrupt executed does not reload the timer with a TXTM; otherwise, there will be an interrupt pending when interrupts are re-enabled. This is especially important during speech synthesis.

When the interrupt occurs, the A register, the X register, and the status flag are all saved in special interrupt storage areas. The B register is not saved. The contents of the program counter is pushed onto the stack in the same manner as a subroutine call. Interrupts are disabled while the interrupt routine is running. When the RETI instruction is executed, the registers and status flags are restored to the values they had before the interrupt and the old program address is popped from the stack. Interrupts are re-enabled.

Warning: The RAM context switch is not enabled when the synthesizer is not on. However, the context bit still toggles every time an interrupt occurs. A stop instruction must be executed to put the context bit in the proper state before starting synthesis.

6.8 Use of the Stack

The TSP50C4X devices have a five-level push-down stack. The parallel-to-serial register is pushed on the stack by the CALL instruction and by an interrupt. The program counter can be loaded from the top of the stack by the RETI and RETN instructions. The POP instruction can be used to discard the value on the top of the stack. The GET, LUSPS, and LUAA instructions use one level of stack in their execution but push the value back when they are done.

Values can be pushed indefinitely onto the stack. If more than five values are pushed, the oldest value is lost. No more than five values can be popped from the stack. If an attempt is made to return from more than five levels down, the RETN instruction will not pop a value from the stack and program execution will continue at the next instruction. If this occurs during debugging, it is a useful indication that stack overflow has occurred.

7 Customer Information

7.1 Production Flow

The TSP50C4X devices are programmable and have five basic mask options. The semicustom nature of these devices requires a standard defined interface between the customer and Texas Instruments Incorporated. Figure 7-1 shows the standard prototype/production flow for customer TSP50C4X programs initiated through the TI Regional Technology Center (RTC). A list of RTCs is given on the back page.

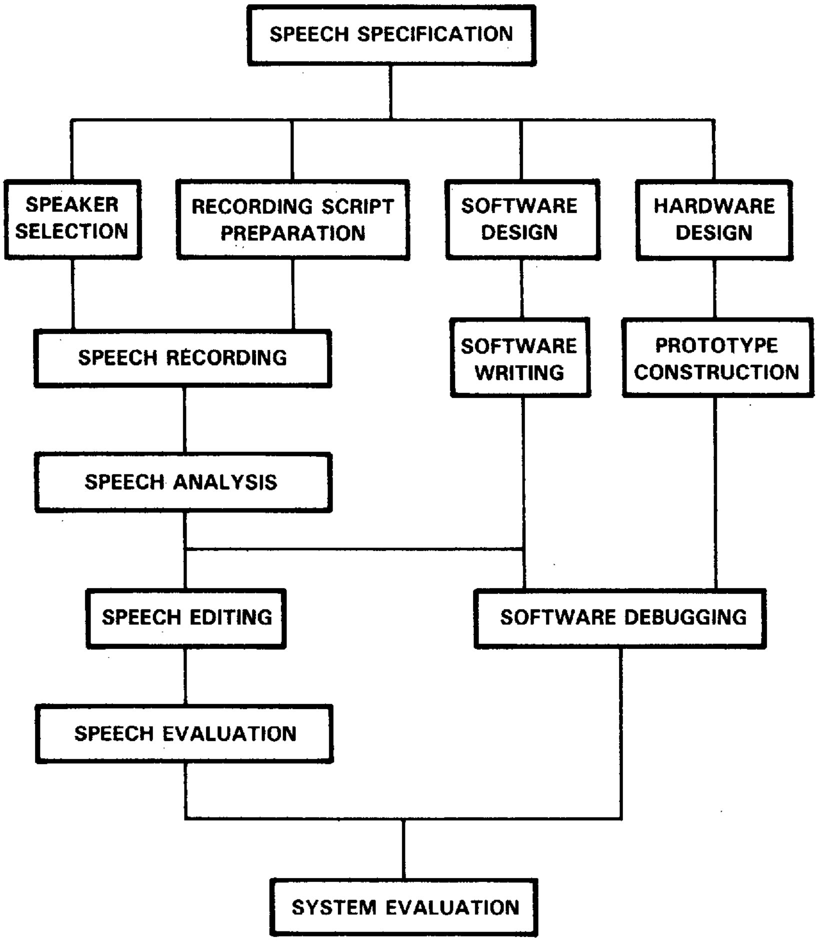


Figure 7-1. Speech Development Cycle

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 (NRPF) 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 thru 7-11.

2. TI generates the prototype photomask, processes, manufactures, and tests 50 prototype devices for shipment to the customer. Limited quantities in addition to the 50 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 initital 50 prototype devices (and any additional prototype devices purchased) were assembled on a prototype (i.e., non-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 50 prototype devices is incurred by the customer

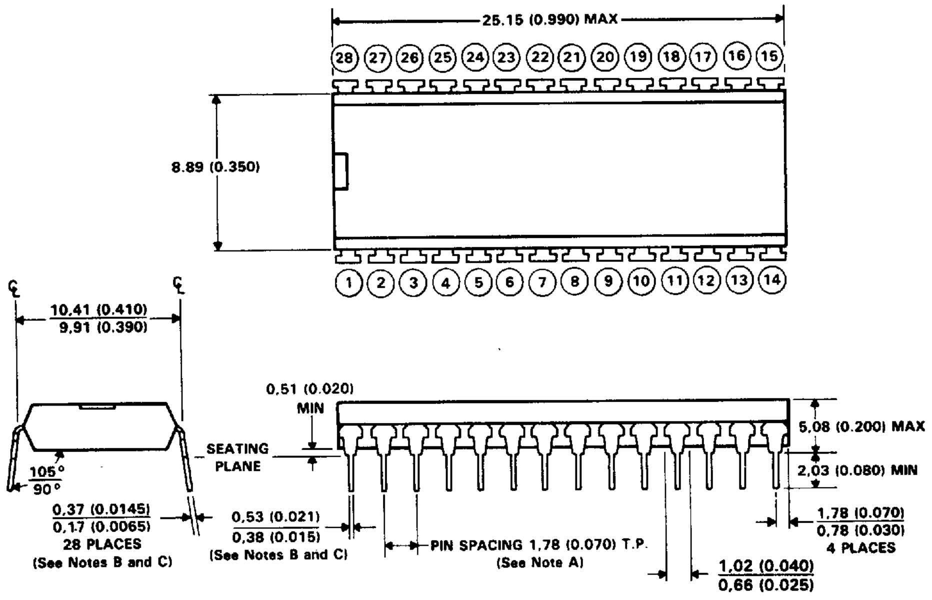
5. A minimum purchase might be required during the first year of production.

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

*New Product Release Form.

7.3 Mechanical Data

This dual-in-line package consists of a circuit mounted on a 28-pin lead frame and encapsulated within a 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 10,16 (0.400) centers. Pin spacing within the rows is 1,78 (0.070). Once the leads are compressed and inserted, sufficient tension is provided to secure the package in the board during soldering. Solder-plated leads require no addtional cleaning or processing when used in soldered assembly.

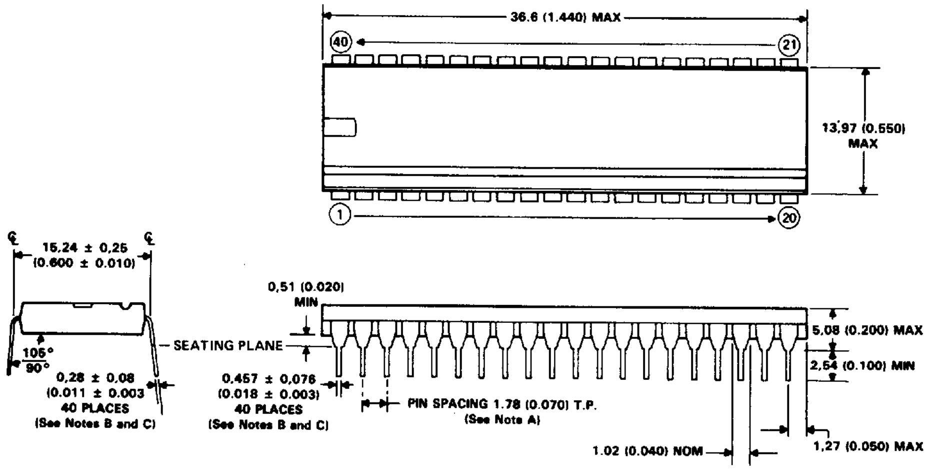


ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

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. TSP50C41 and TSP50C43 28-Pin NF† Plastic Package 0.070" Pin Center Spacing, 0.400" Pin Row Spacing

†The NF package has also been designated N2.



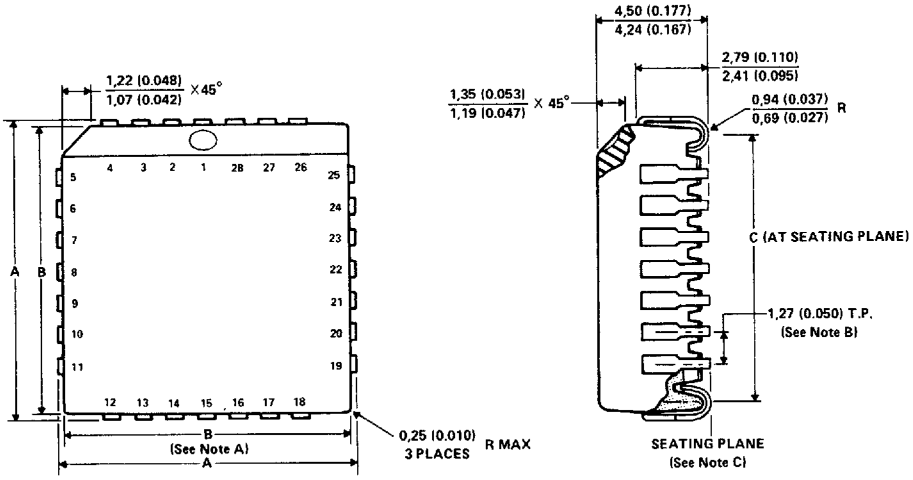
ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

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 7-3. TSP50C42 and TSP50C44 40-Pin NJ Plastic Package 0.070" Pin Center Spacing, 0.600" Pin Row Spacing

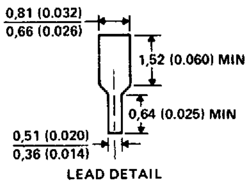
7.3.1 TSP50C43 FN plastic chip carrier package

Each of these chip carrier packages 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 devices are operated in high-humidity conditions. The packages are intended for surface mounting on solder lands on 1,27 (0.050) centers. Leads require no additional cleaning or processing when used in soldered assembly.



1,22 (0.048) / 1,07 (0.042) X45°
1,35 (0.053) / 1,19 (0.047) X 45°
4,50 (0.177) / 4,24 (0.167)
2,79 (0.110) / 2,41 (0.095)
0,94 (0.037) / 0,69 (0.027) R
C (AT SEATING PLANE)
1,27 (0.050) T.P. (See Note B)
A
B
4 3 2 1 28 27 26
5 6 7 8 9 10 11
25 24 23 22 21 20 19
12 13 14 15 16 17 18
B (See Note A)
A
0,25 (0.010) 3 PLACES R MAX
SEATING PLANE (See Note C)

A		B		C	
MIN	MAX	MIN	MAX	MIN	MAX
12,32 (0.485)	12,57 (0.495)	11,43 (0.450)	11,58 (0.456)	10.41 (0.410)	10.92 (0.430)



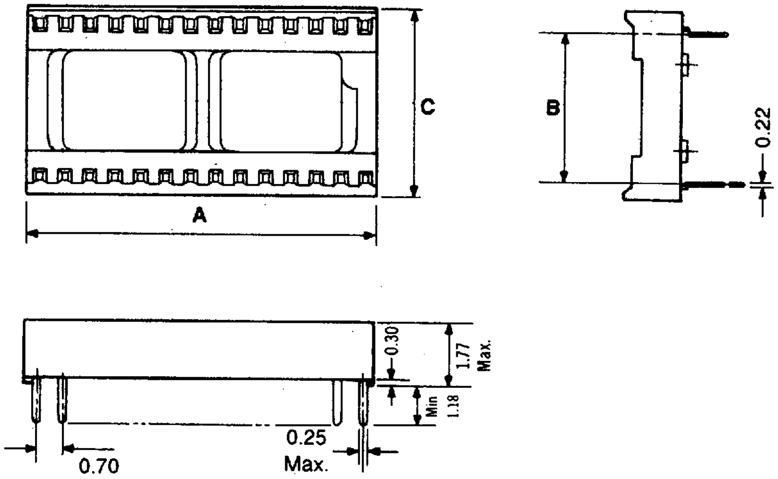
ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

NOTES:
A. Centerline of center lead on each side is within 0,10 (0.04) of package centerline as determined by dimension B.
B. Location of each lead within 0,127 (0.005) of true position with respect to center lead on each side.
C. The lead contact points are planar within 0,10 (0.004).

Figure 7-4. FN Plastic Chip Carrier

7.4 IC Sockets

Texas Instruments lines of off-the-shelf interconnection products are designed specifically to meet the performance needs of volume commercial applications. They provide both the economy of a standard product line and performance features developed after many years experience with custom designs.



POSITIONS	A MAX LENGTH	B ±.005 ROW TO ROW	C MAX WIDTH	PART NUMBER
28	.985 (25,02)	.400 (10,16)	.512 (13,00)	C4S28-02
40	1.405 (35,69)	.600 (15,24)	.708 (17,98)	C4S40-02

Dimensions in parentheses are metric.

Figure 7-5. Shrink Package C4S Series 28 and 40 Positions

Additional information including pricing and delivery quotations may be obtained from your nearest TI Distributor, TI Representative, or:

Texas Instruments Incorporated
Connector Systems Department
MS 14-3
Attleboro, Massachusetts 02703
Telephone: (617) 699-3800
TELEX: ABORA927708

7.5 Ordering Information

Since the TSP50C4X are custom devices, they receive a distinct identification as follows:

CSM or CSS	4XXXX	N2 FN
Gate Code (CS Custom Synthesizer with Memory. M...Master Option S...Slave Option)	ROM Code	Package N2 Plastic DIP FN Plastic Chip Carrier

7.6 New Product Release Form (TSP50C41)

This document describes completely the functional details of the TSP50C41 device number CSM41XXX being released to prototype tooling. The EPROM returned is programmed from the data stored at TI for making prototypes. Please review both and contact TI immediately if there are any questions. Sign and return one copy of this document when approved.

Return to: Texas Instruments Incorporated
P.O. Box 655303 MS 8211
Dallas, TX 75265

Company : ____________________
Division : ____________________
Address : ____________________
Telephone No. : ____________________
Approved By : ____________________
Title : ____________________
Date : ____________________

Options

Master [ ]	or	Slave [ ]
Master Option Requirements Normal [X] $\overline{\text{IRT}}$ in [X]		**Slave Option Selections Only** Normal [ ] or Keyscan [ ] $\overline{\text{IRT}}$ in [ ] or $\overline{\text{IRT}}$ out [ ]
4K Program Space [ ]	or	8K Program Space [ ]
Setoff enabled [ ]	or	Setoff disabled [ ]

Pull-Up Resistors

PA0[ ] PA1[ ] PA2[ ] PA3[ ] PA4[ ] PA5[ ] PA6[ ] PA7[ ]

PC0[ ] PC1[ ] PC2[ ] PC3[ ]

IRT[ ] INIT[ ]

Note: $\overline{\text{IRT}}$ out option requires $\overline{\text{IRT}}$ pull-up resistor.

Open-Drain Outputs: PB0 thru PB3[ ] PB4[ ] thru PB7[ ]

Package Type: N2[ ]

Leads on 0.070" Centers, 0.400" wide

7.7 New Product Release Form (TSP50C42)

This document describes completely the functional details of the TSP50C42 device number CSM42XXX being released to prototype tooling. The EPROM returned is programmed from the data stored at TI for making prototypes. Please review both and contact TI immediately if there are any questions. Sign and return one copy of this document when approved.

Return to:	Texas Instruments Incorporated	Company : ____________
	P.O. Box 655303 MS 8211	Division : ____________
	Dallas, TX 75265	Address : ____________
		Telephone No. : ____________
		Approved By : ____________
		Title : ____________
		Date : ____________

Options

Master [ ] or Slave [ ]

Master Option Requirements	Slave Option Selections Only
Normal [X]	Normal [ ] or Keyscan [ ]
$\overline{\text{IRT}}$ in [X]	$\overline{\text{IRT}}$ in [ ] or $\overline{\text{IRT}}$ out [ ]

4K Program Space [ ] or 8K Program Space [ ]

Setoff enabled [ ] or Setoff disabled [ ]

Pull-Up Resistors

PA0[ ] PA1[ ] PA2[ ] PA3[ ] PA4[ ] PA5[ ] PA6[ ] PA7[ ]

PC0[ ] PC1[ ] PC2[ ] PC3[ ] PC4[ ] PC5[ ] PC6[ ] PC7[ ]

IRT[ ] INIT[ ]

Note: $\overline{\text{IRT}}$ out option requires $\overline{\text{IRT}}$ pull-up resistor.

Open-Drain Outputs: PB0 thru PB3[ ] PB4[ ] thru PB7[ ]

PD0 thru PD3[ ] PD3[ ] thru PD7[ ]

Package Type: N2[ ]

Leads on 0.070" Centers, 0.600" wide

7.8 New Product Release Form (TSP50C43)

This document describes completely the functional details of the TSP50C43 device number CSM43XXX being released to prototype tooling. The EPROM returned is programmed from the data stored at TI for making prototypes. Please review both and contact TI immediately if there are any questions. Sign and return one copy of this document when approved.

Return to:	Texas Instruments Incorporated	Company :	________________
	P.O. Box 655303 MS 8211	Division :	________________
	Dallas, TX 75265	Address :	________________
		Telephone No. :	________________
		Approved By :	________________
		Title :	________________
		Date :	________________

Options

Master [ ]	or	Slave [ ]
Master Option Requirements		**Slave Option Selections Only**
Normal [X]		Normal [ ] or Keyscan [ ]
$\overline{\text{IRT}}$ in [X]		$\overline{\text{IRT}}$ in [ ] or $\overline{\text{IRT}}$ out [ ]
4K Program Space [ ]	or	8K Program Space [ ]
Setoff enabled [ ]	or	Setoff disabled [ ]

Pull-Up Resistors

PA0[ ] PA1[ ] PA2[ ] PA3[ ] PA4[ ] PA5[ ] PA6[ ] PA7[ ]

PC0[ ] PC1[ ] PC2[ ] PC3[ ]

IRT[ ] INIT[ ]

Note: $\overline{\text{IRT}}$ out option requires $\overline{\text{IRT}}$ pull-up resistor.

Open-Drain Outputs: PB0 thru PB3[ ] PB4[ ] thru PB7[ ]

Package Type: FN[ ]	or	N2[ ]
Leads on 0.050" Centers, 0.450" wide PLCC		Leads on 0.070" Centers, 0.400"wide DIP

7.9 New Product Release Form (TSP50C44)

This document describes completely the functional details of the TSP50C44 device number CSM44XXX being released to prototype tooling. The EPROM returned is programmed from the data stored at TI for making prototypes. Please review both and contact TI immediately if there are any questions. Sign and return one copy of this document when approved.

Return to: Texas Instruments Incorporated
P.O. Box 655303 MS 8211
Dallas, TX 75265

Company : ____________________
Division : ____________________
Address : ____________________
Telephone No. : ____________________
Approved By : ____________________
Title : ____________________
Date : ____________________

Options

Master [ ]	or	Slave [ ]
Master Option Requirements Normal [X] $\overline{\text{IRT}}$ in [X]		Slave Option Selections Only Normal [ ] or Keyscan [ ] $\overline{\text{IRT}}$ in [ ] or $\overline{\text{IRT}}$ out [ ]
4K Program Space [ ]	or	8K Program Space [ ]
Setoff enabled [ ]	or	Setoff disabled [ ]

Pull-Up Resistors
PA0[ ] PA1[ ] PA2[ ] PA3[ ] PA4[ ] PA5[ ] PA6[ ] PA7[ ]
PC0[ ] PC1[ ] PC2[ ] PC3[ ] PC4[ ] PC5[ ] PC6[ ] PC7[ ]
IRT[ ] INIT[ ]
Note: $\overline{\text{IRT}}$ out option requires $\overline{\text{IRT}}$ pull-up resistor.

Open-Drain Outputs: PB0 thru PB3[ ] PB4[ ] thru PB7[ ]
PD0 thru PD3[ ] PD4 thru PD7[ ]

Package Type: N2[ ]
Leads on 0.070″ Centers, 0.600″ wide

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 Recording Script Generation

The first step in designing a system using LPC is the generation of a system specification, including a script. A coding table needs to be selected that yields the best data rate for the voice selected at the level of quality required. 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 have their 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. Data rates for the TSP50C4X range from 1000 to 3000 bits per second and words average 0.6 second each. These are very rough rules of thumb and each application is different.

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 the words. This yields good intelligibility at low data rates, but it does not provide a 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, a statement, or an order. In addition, all the words are run together with each word 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 often memory constraints force compromises. An expert speech editor can look at a script that lists each word in each context where it occurs and determine what contexts are similar enough to permit reuse.

Once the application is designed 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 much more straightforward, with a word list or a phrase list being all that is required.

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. In addition, the strain of speaking for long periods of time in a controlled manner is considerable. Nonprofessional speakers are best used only for prototyping.

During the session, it may be necessary to experiment with inflection and expression to find the best approach. Ideally, the person making the final decision on product content and aesthetics should be at the recording session. Leaving this task to others leads to repeat visits to the studio.

There are various techniques that can be used to ensure that the speech will analyze and synthesize properly. Certain consonants need to be emphasized more and spoken more clearly than they are in normal speech. The TI SDS5000 development tool provides immediate feedback of synthetic speech making the collecting process much easier for inexperienced users.

The actual collection process is fairly simple. The speech is converted into a digital form and then analyzed with a very computation intensive algorithim. The SDS5000 uses a TMS32020 digital signal processing chip to permit very rapid analysis. It consists of two boards designed to 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 algorithim that provides very high quality but 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.

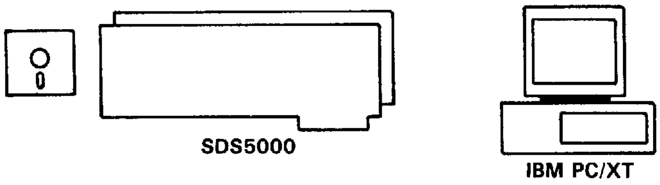
TI offers the SDS5000 Speech Development System, which eases many of these tasks by analyzing the speech immediately to provide quick feedback and to permit re-recording if the synthetic speech does not offer the desired quality. The TSP50C4X devices offer a variety of coding tables, permitting the use of higher data rates to achieve high quality with less editing, along with the flexibility of lower data rates when memory cost constraints outweigh the costs of editing.

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.

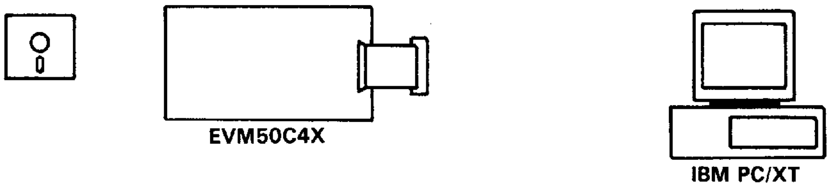
LPC provides very low data rate speech by virtue of its close modeling of the human vocal tract. Other sounds may or may not be modeled accurately by the model. The best way to find out is to try recording and analyzing the sound on the SDS5000. Applications assistance is available from TI for commonly used sounds such as musical tones and chimes. In addition, we have experience with a wide variety of other sounds. On the TSP50C4X devices, it is also possible to get direct access to the D-to-A output, so all sounds can be modeled, although at a considerable penalty in data rate.

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
- REQUIRES IBM PC/XT, AT, OR COMPATIBLE WITH CGA CARD
- HARD DISK AND TAPE BACKUP STRONGLY SUGGESTED
- USES TMS32020 DIGITAL SIGNAL PROCESSOR

Figure A-1. SDS5000



- IN CIRCUIT EMULATION
- HARDWARE BREAKPOINTS
- SINGLE STEP
- EXAMINE/MODIFY REGISTERS/MEMORY
- INCLUDES ASSEMBLER
- WORKS WITH IBM PC, PC/XT, PC/AT, AND COMPATIBLES
- REQUIRES EXTERNAL 5-, 12-, −12-VOLT POWER SUPPLY

Figure A-2. EVM50C4X

- IN CIRCUIT EMULATION
- SMALL SIZE, LOW POWER CONSUMPTION
- IDEAL FOR DEMONSTRATION AND FIELD TEST
- REQUIRES INDUSTRY STANDARD EPROM (TMS27C128)

Figure A-3. SEB50C4X

- IN CIRCUIT EMULATION OF UP TO FOUR TSP60C20S
- SMALL SIZE, LOW POWER CONSUMPTION
- IDEAL FOR DEBUGGING, DEMONSTRATION, AND FIELD TEST
- REQUIRES INDUSTRY STANDARD EPROMS (TMS27C256)

Figure A-4. SEB60C20

B TSP50C4X Synthesis Program

This program speaks a single phrase that is stored in the internal ROM, starting at address #0800.

```
*+------------------------------------------+
* TSP50C4X SYNTHESIS PROGRAM
*+------------------------------------------+
* COPYRIGHT 1987 TI - SPEECH PRODUCTS
*+------------------------------------------+
* RAM MAP
*+------------------------------------------+
*
```

00	01	02	03	04	05	06	07
PITCH		ENERGY		K1		K2	
NV	PV	NV	PV	NV	PV	NV	PV
08	09	0A	0B	0C	0D	0E	0F
K3		K4		K5		K6	
NV	PV	NV	PV	NV	PV	NV	PV
10	11	12	13	14	15	16	17
K7		K8		K9		K10	
NV	PV	NV	PV	NV	PV	NV	PV
18	19	1A	1B	1C	1D	1E	1F
PBF	EBF	K1BF	K2BF	K3BF	K4BF	K5BF	K6BF
20	21	22	23	24	25	26	27
K7BF	K8BF	K9BF	K10-BF	FEBF	FK1-BF	FK2-BF	TEMP
28	29	2A	2B	2C	2D	2E	2F
FLA-GS							
30	31	32	33	34	35	36	37

```
0040   *
0041   *
0042   *
0043   *
0044   *
0045   *
0046   *
0047   *
0048   *
0049   *
0050   *
0051   *
0052   *
0053   *
0054   *
0055   *
0056   *
0057   *
0058   *
0059   *
0060   *
0061   *
0062   *
0063   *
0064   *
0065   *
0066   *
0067   *
0068   *
0069   *
0070   *
0071   *
0072   *
0073   *
0074   *
0075   *
0076   *
0077   *
0078   *
0079   *
0080   *
0081   *
0082   *
0083   *
0084   *
0085   *
```

38	39	3A	3B	3C	3D	3E	3F
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

```
          *
          *
          *
          *   4 BITS WIDE BELOW (FRACTIONAL VALUES)
          *
          *   80    81    82       83    84    85    86    87
          *
          *   FPITCH      FENERGY        FK1         FK2
          *   NV    PV    NV       PV    NV    PV    NV    PV
          *
          *   88    89    8A       8B    8C    8D    8E    8F
          *
          *   FK3         FK4            FK5         FK6
          *   NV    PV    NV       PV    NV    PV    NV    PV


          *+----------------------------------------+
          * ADDRESS LABELS FOR SYNTHESIS ROUTINE
          *+----------------------------------------+
          *
          *+----------------------------------------+
          * SYNTHESIZER RAM LOCATIONS
          *+----------------------------------------+
     0000 PNV     EQU    #00    ;PITCH NEW VALUE
     0001 PPV     EQU    #01    ;PITCH PRESENT VALUE
     0002 ENV     EQU    #02    ;ENERGY NEW VALUE ADDR
     0003 EPV     EQU    #03    ;ENERGY PRESENT VALUE
     0004 K1NV    EQU    #04    ;K1 NEW VALUE
     0005 K1PV    EQU    #05    ;K1 PRESENT VALUE
     0006 K2NV    EQU    #06    ;K2 NEW VALUE
     0007 K2PV    EQU    #07    ;K2 PRESENT VALUE
     0008 K3NV    EQU    #08    ;K3 NEW VALUE
     0009 K3PV    EQU    #09    ;K3 PRESENT VALUE
     000A K4NV    EQU    #0A    ;K4 NEW VALUE
     000B K4PV    EQU    #0B    ;K4 PRESENT VALUE
     000C K5NV    EQU    #0C    ;K5 NEW VALUE
     000D K5PV    EQU    #0D    ;K5 PRESENT VALUE
     000E K6NV    EQU    #0E    ;K6 NEW VALUE
     000F K6PV    EQU    #0F    ;K6 PRESENT VALUE
     0010 K7NV    EQU    #10    ;K7 NEW VALUE
     0011 K7PV    EQU    #11    ;K7 PRESENT VALUE
     0012 K8NV    EQU    #12    ;K8 NEW VALUE
     0013 K8PV    EQU    #13    ;K8 PRESENT VALUE
     0014 K9NV    EQU    #14    ;K9 NEW VALUE
```

```
0015 K9PV    EQU    #15    ;K9 PRESENT VALUE
0016 K10NV   EQU    #16    ;K10 NEW VALUE
0017 K10PV   EQU    #17    ;K10 PRESENT VALUE
     *----------------------------------------------
     * FRACTIONAL VALUES
     *----------------------------------------------
0080 FPNV    EQU    #80    ;PITCH NEW VALUE
0081 FPPV    EQU    #81    ;PITCH PRESENT VALUE
0082 FENV    EQU    #82    ;ENERGY PRESENT VALUE ADDR
0083 FEPV    EQU    #83    ;ENERGY NEW VALUE
0084 FK1NV   EQU    #84    ;K1 NEW VALUE
0085 FK1PV   EQU    #85    ;K1 PRESENT VALUE
0086 FK2NV   EQU    #86    ;K2 NEW VALUE
0087 FK2PV   EQU    #87    ;K2 PRESENT VALUE
0088 FK3NV   EQU    #88    ;K3 NEW VALUE
0089 FK3PV   EQU    #89    ;K3 PRESENT VALUE
008A FK4NV   EQU    #8A    ;K4 NEW VALUE
008B FK4PV   EQU    #8B    ;K4 PRESENT VALUE
008C FK5NV   EQU    #8C    ;K5 NEW VALUE
008D FK5PV   EQU    #8D    ;K5 PRESENT VALUE
008E FK6NV   EQU    #8E    ;K6 NEW VALUE
008F FK6PV   EQU    #8F    ;K6 PRESENT VALUE
     *+---------------------------------------------+
     * BUFFER RAM LOCATIONS
     *+---------------------------------------------+
0018 PBF     EQU    #18    ;PITCH BUFFER
0019 EBF     EQU    #19    ;ENERGY BUFFER
001A K1BF    EQU    #1A    ;K1 BUFFER
001B K2BF    EQU    #1B    ;K2 BUFFER
001C K3BF    EQU    #1C    ;K3 BUFFER
001D K4BF    EQU    #1D    ;K4 BUFFER
001E K5BF    EQU    #1E    ;K5 BUFFER
001F K6BF    EQU    #1F    ;K6 BUFFER
0020 K7BF    EQU    #20    ;K7 BUFFER
0021 K8BF    EQU    #21    ;K8 BUFFER
0022 K9BF    EQU    #22    ;K9 BUFFER
0023 K10BF   EQU    #23    ;K10 BUFFER
     *----------------------------------------------
     * FRACTIONAL BUFFER RAM LOCATIONS
     *----------------------------------------------
0024 FEBF    EQU    #24    ;FRACTIONAL ENERGY BUFFER
0025 FK1BF   EQU    #25    ;FRACTIONAL K1 BUFFER
0026 FK2BF   EQU    #26    ;FRACTIONAL K2 BUFFER
```

```
     *+----------------------------------------+
     * CONTROL RAM LOCATIONS
     *+----------------------------------------- +
0027 TEMP    EQU    #27     ;TEMPORARY STORAGE
0028 FLAGS   EQU    #28     ;FLAGS FOR SPEECH
     *
     *------------------------------------------
     * BIT DEFINITIONS IN RAM LOCATION FLAGS
     *------------------------------------------
     *
0001 STP     EQU    1       ;STOP DETECTED
0002 INT     EQU    2       ;INTERUPT FLAG
     *                      ;SET BY INTERRUPT ROUTINE,
     *                      ;CLEARED BY CONVERSION ROUTINE.
0003 NINTP   EQU    3       ;SET FOR NO INTERPOLATION
0004 UNVO    EQU    4       ;SET FOR TARGET FRAME UNVOICED
0005 STRT    EQU    5       ;SET FOR FIRST TWO FRAMES
0006 STRT1   EQU    6       ;SET WHILE SECOND FRAME IS COMING
                            IN
0007 RPT     EQU    7       ;SET IF REPEAT DETECTED
0008 STP1    EQU    8       ;SECOND STOP FRAME
     *------------------------------------------
     * SPEECH DECODING CONSTANTS
     *------------------------------------------
000F ESTOP   EQU    #0F     ;ENERGY STOP CODE
0000 ESILE   EQU    #00     ;SILENT CODE
     *
0001 REPT    EQU    #01     ;REPEAT CODE
     *
0000 PUNVO   EQU    #00     ;PITCH UNVOICED CODE
     *
001F TMVAL   EQU    #1F     ;VALUE FOR TIMER REGISTER
     *------------------------------------------
     * NUMBER OF BITS FOR TSP5220 CODING TABLE
     *------------------------------------------
     *
0004 NRGNB   EQU    4       ;ENERGY
0006 PITNB   EQU    6       ;PITCH
0001 RPTNB   EQU    1       ;REPEAT
0005 K1NB    EQU    5       ;K1
0005 K2NB    EQU    5       ;K2
0004 K3NB    EQU    4       ;K3
0004 K4NB    EQU    4       ;K4
0004 K5NB    EQU    4       ;K5
0004 K6NB    EQU    4       ;K6
```

```
     0004 K7NB    EQU     4       ;K7
     0003 K8NB    EQU     3       ;K8
     0003 K9NB    EQU     3       ;K9
     0003 K10NB   EQU     3       ;K10
          *----------------------------------------
          * BEGINNING OF PROGRAM
          *----------------------------------------
0000              AORG    #0000
0000 84           SBR     GOGO    ;POWER UP VECTOR
0001 84           SBR     GOGO
0002 86           SBR     INTO    ;INTERRUPT VECTOR
0003 86           SBR     INTO
0004 61   GOGO    BR      GO      ;BRANCH TO SPEECH ROUTINE
0005 5F
0006 62   INTO    BR      INT1    ;BRANCH TO INTERUPT ROUTINE
0007 C3
          *----------------------------------------
          * 5220 SPEECH DECODING TABLES.
          *----------------------------------------
          *
          *+----------------------------------------+
          * ENERGY DECODING TABLE
          *+----------------------------------------+
0008 00   TABEN   BYTE    #00     ;CODED AS 0
0009 00           BYTE    #00     ;ENERGY CODE 1
000A 01           BYTE    #01     ;ENERGY CODE 2
000B 01           BYTE    #01     ;ENERGY CODE 3
000C 02           BYTE    #02     ;ENERGY CODE 4
000D 03           BYTE    #03     ; 5
000E 05           BYTE    #05     ; 6
000F 07           BYTE    #07     ; 7
0010 0A           BYTE    #0A     ; 8
0011 0E           BYTE    #0E     ; 9
0012 14           BYTE    #14     ; 10
0013 1C           BYTE    #1C     ; 11
0014 28           BYTE    #28     ; 12
0015 38           BYTE    #38     ; 13
0016 50           BYTE    #50     ; 14
          *+----------------------------------------+
          * PITCH DECODING TABLE
          *+----------------------------------------+
0017 0C   TABPI   BYTE    #0C     UNVOICED
0018 10           BYTE    #10     625 HZ
0019 11           BYTE    #11     588
001A 12           BYTE    #12     555
```

```
0259  001B  13              BYTE      #13       526
0260  001C  14              BYTE      #14       500
0261  001D  15              BYTE      #15       476
0262  001E  16              BYTE      #16       454
0263  001F  17              BYTE      #17       435
0264  0020  18              BYTE      #18       416
0265  0021  19              BYTE      #19       400
0266  0022  1A              BYTE      #1A       384
0267  0023  1B              BYTE      #1B       370
0268  0024  1C              BYTE      #1C       357
0269  0025  1D              BYTE      #1D       344
0270  0026  1E              BYTE      #1E       333
0271  0027  1F              BYTE      #1F       322
0272  0028  20              BYTE      #20       312
0273  0029  21              BYTE      #21       303
0274  002A  22              BYTE      #22       294
0275  002B  23              BYTE      #23       286
0276  002C  24              BYTE      #24       278
0277  002D  25              BYTE      #25       270
0278  002E  26              BYTE      #26       263
0279  002F  27              BYTE      #27       256
0280  0030  28              BYTE      #28       250
0281  0031  29              BYTE      #29       244
0282  0032  2A              BYTE      #2A       238
0283  0033  2B              BYTE      #2B       232
0284  0034  2D              BYTE      #2D       222
0285  0035  2F              BYTE      #2F       212
0286  0036  31              BYTE      #31       204
0287  0037  33              BYTE      #33       196
0288  0038  35              BYTE      #35       189
0289  0039  36              BYTE      #36       185
0290  003A  39              BYTE      #39       175
0291  003B  3B              BYTE      #3B       169
0292  003C  3D              BYTE      #3D       163
0293  003D  3F              BYTE      #3F       158
0294  003E  42              BYTE      #42       151
0295  003F  45              BYTE      #45       145
0296  0040  47              BYTE      #47       141
0297  0041  49              BYTE      #49       137
0298  0042  4D              BYTE      #4D       130
0299  0043  4F              BYTE      #4F       126
0300  0044  51              BYTE      #51       123
0301  0045  55              BYTE      #55       118
0302  0046  57              BYTE      #57       115
0303  0047  5C              BYTE      #5C       109
```

```
0048 5F           BYTE     #5F      105
0049 63           BYTE     #63      101
004A 66           BYTE     #66      98
004B 6A           BYTE     #6A      94
004C 6E           BYTE     #6E      91
004D 73           BYTE     #73      87
004E 77           BYTE     #77      84
004F 7B           BYTE     #7B      81
0050 80           BYTE     #80      78
0051 85           BYTE     #85      75
0052 8A           BYTE     #8A      72
0053 8F           BYTE     #8F      70
0054 95           BYTE     #95      67
0055 9A           BYTE     #9A      65
0056 A0           BYTE     #A0      62
          *+-------------------------------------------+
          * K1 DECODING TABLE
          *+-------------------------------------------+
0057 82   TABK1   BYTE     #82,#83,#83,#84,#84,#85,#86,#87,#88,
                           #89
0058 83
0059 83
005A 84
005B 84
005C 85
005D 86
005E 87
005F 88
0060 89
0061 8A           BYTE     #8A,#8C,#8D,#8F,#90,#92,#99,#A1,#AB,
                           #B8
0062 8C
0063 8D
0064 8F
0065 90
0066 92
0067 99
0068 A1
0069 AB
006A B8
006B C7           BYTE     #C7,#D8,#EB,#00,#14,#27,#38,#47,#54,
                           #5E
006C D8
006D EB
006E 00
```

```
006F 14
0070 27
0071 38
0072 47
0073 54
0074 5E
0075 67               BYTE    #67,#6D
0076 6D
                *+---------------------------------------+
                * K2 DECODING TABLE
                *+---------------------------------------+
0077 AE         TABK2 BYTE    #AE,#B4,#BB,#C3,#CB,#D4,#DD,#E7
0078 B4
0079 BB
007A C3
007B CB
007C D4
007D DD
007E E7
007F F1               BYTE    #F1,#FB,#06,#10,#1A,#24,#2D,#36
0080 FB
0081 06
0082 10
0083 1A
0084 24
0085 2D
0086 36
0087 3E               BYTE    #3E,#45,#4C,#53,#58,#5D,#62,#66
0088 45
0089 4C
008A 53
008B 58
008C 5D
008D 62
008E 66
008F 69               BYTE    #69,#6C,#6F,#71,#73,#75,#77,#7E
0090 6C
0091 6F
0092 71
0093 73
0094 75
0095 77
0096 7E
```

```
                *+-----------------------------------------+
                * K3 DECODING TABLE
                *+-----------------------------------------+
0097  92        TABK3   BYTE    #92,#9F,#AD,#BA,#C8,#D5,#E3,#F0
0098  9F
0099  AD
009A  BA
009B  C8
009C  D5
009D  E3
009E  F0
009F  FE                BYTE    #FE,#0B,#19,#26,#34,#41,#4F,#5C
00A0  0B
00A1  19
00A2  26
00A3  34
00A4  41
00A5  4F
00A6  5C
                *+-----------------------------------------+
                * K4 DECODING TABLE
                *+-----------------------------------------+
00A7  AE        TABK4   BYTE    #AE,#BC,#CA,#D8,#E6,#F4,#01,#0F
00A8  BC
00A9  CA
00AA  D8
00AB  E6
00AC  F4
00AD  01
00AE  0F
00AF  1D                BYTE    #1D,#2B,#39,#47,#55,#63,#71,#7E
00B0  2B
00B1  39
00B2  47
00B3  55
00B4  63
00B5  71
00B6  7E
                *+-----------------------------------------+
                * K5 DECODING TABLE
                *+-----------------------------------------+
00B7  AE        TABK5   BYTE    #AE,#BA,#C5,#D1,#DD,#E8,#F4,#FF
00B8  BA
00B9  C5
00BA  D1
```

```
     00BB  DD
     00BC  E8
     00BD  F4
     00BE  FF
0347 00BF  0B              BYTE    #0B,#17,#22,#2E,#39,#45,#51,#5C
     00C0  17
     00C1  22
     00C2  2E
     00C3  39
     00C4  45
     00C5  51
     00C6  5C
0348                *+----------------------------------------+
0349                * K6 DECODING TABLE
0350                *+----------------------------------------+
0351 00C7  C0       TABK6  BYTE    #C0,#CB,#D6,#E1,#EC,#F7,#03,#0E
     00C8  CB
     00C9  D6
     00CA  E1
     00CB  EC
     00CC  F7
     00CD  03
     00CE  0E
0352 00CF  19              BYTE    #19,#24,#2F,#3A,#45,#50,#5B,#66
     00D0  24
     00D1  2F
     00D2  3A
     00D3  45
     00D4  50
     00D5  5B
     00D6  66
0353                *+----------------------------------------+
0354                * K7 DECODING TABLE
0355                *+----------------------------------------+
0356 00D7  B3       TABK7  BYTE    #B3,#BF,#CB,#D7,#E3,#EF,#FB,#07
     00D8  BF
     00D9  CB
     00DA  D7
     00DB  E3
     00DC  EF
     00DD  FB
     00DE  07
0357 00DF  13       BYTE           #13,#1F,#2B,#37,#43,#4F,#5A,#66
     00E0  1F
     00E1  2B
```

```
00E2  37
00E3  43
00E4  4F
00E5  5A
00E6  66
                *+----------------------------------------+
                * K8 DECODING TABLE
                *+----------------------------------------+
00E7  C0        TABK8   BYTE    #C0,#D8,#F0,#07,#1F,#37,#4F,#66
00E8  D8
00E9  F0
00EA  07
00EB  1F
00EC  37
00ED  4F
00EE  66
                *+----------------------------------------+
                * K9 DECODING TABLE
                *+----------------------------------------+
00EF  C0        TABK9   BYTE    #C0,#D4,#E8,#FC,#10,#25,#39,#4D
00F0  D4
00F1  E8
00F2  FC
00F3  10
00F4  25
00F5  39
00F6  4D
                *+----------------------------------------+
                * K10 DECODING TABLE
                *+----------------------------------------+
00F7  CD        TAK10   BYTE    #CD,#DF,#F1,#04,#16,#28,#3B,#4D
00F8  DF
00F9  F1
00FA  04
00FB  16
00FC  28
00FD  3B
00FE  4D
                *+----------------------------------------+
                * FRACTIONAL ENERGY DECODING TABLE
                *+----------------------------------------+
00FF  00        TABEF   BYTE    #00,#0C,#04,#0C,#08,#08,#00,#04
0100  0C
0101  04
0102  0C
```

```
     0103  08
     0104  08
     0105  00
     0106  04
0374 0107  00          BYTE     #00,#04,#04,#08,#04,#0C,#04,#04
     0108  04
     0109  04
     010A  08
     010B  04
     010C  0C
     010D  04
     010E  04
0375             *+------------------------------------------+
0376             * FRACTIONAL K1 DECODING TABLE
0377             *+------------------------------------------+
0378 010F  0C    TAK1F BYTE     #0C,#08,#0C,#04,#0C,#04,#00,#08
     0110  08
     0111  0C
     0112  04
     0113  0C
     0114  04
     0115  00
     0116  08
0379 0117  08          BYTE     #08,#08,#0C,#00,#04,#00,#0C,#0C
     0118  08
     0119  0C
     011A  00
     011B  04
     011C  00
     011D  0C
     011E  0C
0380 011F  00          BYTE     #00,#04,#08,#04,#04,#0C,#0C,#00
     0120  04
     0121  08
     0122  04
     0123  04
     0124  0C
     0125  0C
     0126  00
0381 0127  04          BYTE     #04,#04,#0C,#0C,#08,#0C,#00,#04
     0128  04
     0129  0C
     012A  0C
     012B  08
     012C  0C
```

```
012D 00
012E 04
          *+-----------------------------------------+
          * FRACTIONAL K2 DECODING TABLE
          *+-----------------------------------------+
012F 00   TAK2F   BYTE    #00,#08,#08,#04,#08,#04,#0C,#08
0130 08
0131 08
0132 04
0133 08
0134 04
0135 0C
0136 08
0137 08           BYTE    #08,#0C,#00,#04,#04,#00,#04,#00
0138 0C
0139 00
013A 04
013B 04
013C 00
013D 04
013E 00
013F 04           BYTE    #04,#0C,#0C,#00,#08,#0C,#04,#04
0140 0C
0141 0C
0142 00
0143 08
0144 0C
0145 04
0146 04
0147 0C           BYTE    #0C,#0C,#08,#0C,#0C,#08,#00,#08
0148 0C
0149 08
014A 0C
014B 0C
014C 08
014D 00
014E 08
          *+-----------------------------------------+
          * INIT: INITIALIZE PROCESSOR
          *+-----------------------------------------+
014F 1D   INIT    INTD            ; DISABLE INTERRUPTS
0150 1B           STOP            ; STOP SYNTHESIZER
0151 00           CLA
0152 50           ACAA    49      ; 200 SAMPLES/FR (200/4) - 1
0153 31
```

```
0154 5D           TAPSC            ; INTO PRESCALE REGISTER
0155 1F           RETN
          *
          *----------------------------------------------
          * RAM CLEAR
          *
          * SUBROUTINE NAME : RAMO
          * USES : A, X, ALL OF RAM
          * DESCRIPTION : FILLS RAM WITH ZEROES
          *
          *----------------------------------------------
0156 00   RAMO    CLA              ;CLEAR ACCUMULATOR
0157 11           CLX              ;POINT TO FIRST RAM LOCATION
0158 09   RC1     TAM              ;CLEAR RAM LOCATION
0159 0F           IXC              ;POINT TO NEXT RAM LOCATION
015A 55           XGEC    #90      ;AT END OF RAM?
015B 90
015C DE           SBR     RC2      ;BRANCH IF SO
015D D8           SBR     RC1
015E 1F   RC2     RETN
          *
          *----------------------------------------------
          * START OF SYNTHESIS PROGRAM
          *----------------------------------------------
015F 71   GO      CALL    INIT     ;INITIALIZE PROCESSOR
0160 4F
0161 71           CALL    RAMO     ;CLEAR INTERNAL RAM
0162 56
0163 2E           RSECT            ;MAKE TIMER INPUT INTERNAL
          *
0164 00           CLA
0165 50           ACAA    #08      ;HIGH 5 BITS OF SPEECH ADDRESS
0166 08
0167 0B           TASH             ;INTO HIGH BITS OF SAR
0168 00           CLA              ;LOW BYTE OF ADDRESS = 0
0169 0A           TASL             ;INTO LOW BYTE OF SAR
          *
016A 59           LUSPS            ;INITIALIZE PARALLEL TO SERIAL REG
016B 2C           INTRM            ;USE INTERNAL ROM
016C 71           CALL    SPSTR    ;SPEAK
016D 70
          *
016E 61   LOOP    BR      LOOP     ;LOOP FOREVER
016F 6E
```

```
                *
                ********************************************
                * SPEECH
                *
                * SUBROUTINE NAME : SPSTR
                * USES : A X, B, RAM FROM 0 TO #29
                * RAM FROM #80 TO #8F
                * 2 STACK LEVELS
                * DESCRIPTION : SYNTHESIZES SPEECH
                *
                ********************************************
0170  56        SPSTR   TCX     PBF
0171  18
0172  00                CLA
0173  50                ACAA    #0C
0174  0C
0175  09                TAM             ;SOME PITCH TO START OUT WITH, IN
                                         CASE WE
                * FRAME
0176  56                TCX     FLAGS   ;
0177  28
0178  00                CLA             ;CLEAR FLAGS
0179  09                TAM             ;
017A  3C                SBITM   STRT    ;FLAG START OF SPEECH
017B  61                BR      SPDE2   ;BRANCH AROUND INTERRUPT CHECK
017C  8B
                *
                ********************************************
                * SPEECH DECODING PROGRAM DECODED VALUE IN BUFFER
                ********************************************
017D  56        SPDEC   TCX     FLAGS
017E  28
017F  41                TBITM   INT     ;HAS INTERRUPT OCURRED?
0180  83                SBR     SPDE1   ;IF SO, GO FOR IT
0181  61                BR      SPDEC   ;ELSE WAIT
0182  7D
                *
0183  49        SPDE1   RBITM   INT     ;RESET INTERRUPT FLAG
0184  4A                RBITM   NINTP   ;RESET INTERPOLATION INHIBIT
                *
0185  40                TBITM   STP     ;CHECK STOP FLAG
0186  63                BR      STPIT   ;BRANCH IF STOP IS APPROPRIATE
0187  38
                *
0188  47                TBITM   STP1    ;CHECK STOP1 FLAG
```

```
0189 61          BR      STPI1    ;BRANCH IF STOP IS APPROPRIATE
018A 96
            *
018B 00  SPDE2   CLA              ;CLEAR A
018C 23          GET     NRGNB    ;GET ENERGY
018D 54          ANEC    ESTOP    ;IS IT THE STOP CODE?
018E 0F
018F 61          BR      NOSTP    ;BRANCH IF NOT
0190 99
0191 56          TCX     FLAGS
0192 28
            *
0193 3F          SBITM   STP1     ;FLAG STOP1
            *
0194 62          BR      CLRPR    ;GO CLEAR PARAMETERS
0195 5A
            *
0196 38  STPI1   SBITM   STP
0197 61          BR      SPDEC
0198 7D
            *
0199 54  NOSTP   ANEC    ESILE    ;IS IT A SILENT FRAME?
019A 00
019B 61          BR      NOSIL    ;BRANCH IF NOT
019C 9F
019D 62          BR      CLRPR    ;IF SO, GO CLEAR PARAMETERS
019E 5A
019F 56  NOSIL   TCX     TEMP
01A0 27
01A1 09          TAM              ;SAVE ENERGY CODE
01A2 50          ACAA    TABEN    ;START OF ENERGY TABLE
01A3 08
01A4 58          LUAA             ;DECODE ENERGY
01A5 56          TCX     EBF
01A6 19
01A7 09          TAM              ;PUT IT AWAY
01A8 56          TCX     TEMP
01A9 27
01AA 04          TMA              ;GET ENERGY CODE BACK
01AB 50          ACAA    TABEF    ;START OF FRACTIONAL ENERGY TABLE
01AC FF
01AD 58          LUAA             ;DECODE ENERGY
01AE 56          TCX     FEBF
01AF 24
01B0 09          TAM              ;PUT IT AWAY
```

```
         *
         * REPEAT
         *
01B1 00           CLA              ;CLEAR A
01B2 20           GET     RPTNB    ;GET REPEAT BIT
01B3 56           TCX     FLAGS    ;
01B4 28
01B5 4E           RBITM   RPT      ;SET BIT REPEAT
01B6 54           ANEC    REPT     ;IS IT REPEAT?
01B7 01
01B8 BA           SBR     LABPI    ; GO TO PITCH
01B9 3E           SBITM   RPT      ;IF NOT RESET BIT REPEAT
         *
         * PITCH
         *
01BA 00  LABPI    CLA              ;CLEAR A
01BB 25           GET     PITNB    ;GET PITCH
01BC 54           ANEC    PUNVO    ;IS IT UNVOICED?
01BD 00
01BE 61           BR      VOICE    ;BRANCH IF NOT
01BF D6
01C0 56           TCX     FLAGS    ;LOOK AT FLAGS
01C1 28
01C2 43           TBITM   UNVO     ;WAS IT UNVOICED BEFORE?
01C3 C5           SBR     UNV1     ;BRANCH IF SO
01C4 3A           SBITM   NINTP    ;DISABLE INTERPOLATION IF NOT
01C5 3B  UNV1     SBITM   UNVO     ;SET PITCH UNVOICED
         *
01C6 56           TCX     PBF
01C7 18
01C8 00           CLA
01C9 50           ACAA    #0C
01CA 0C
01CB 09           TAM
         *
01CC 56           TCX     ENV      ;LOOK AT ENERGY OF PREVIOUS FRAME
01CD 02
01CE 04           TMA
01CF 54           ANEC    0        ;WAS IT 0 (SILENT FRAME)
01D0 00
01D1 E6           SBR     KPARM    ;BRANCH IF NOT
01D2 56           TCX     FLAGS
01D3 28
01D4 3A           SBITM   NINTP
01D5 E6           SBR     KPARM    ;BRANCH IF NOT
```

```
01D6 56  VOICE   TCX     TEMP
01D7 27
01D8 09          TAM             ;SAVE PITCH CODE
01D9 50          ACAA    TABPI   ;START OF PITCH TABLE
01DA 17
01DB 58          LUAA            ;DECODE PITCH
01DC 56          TCX     PBF
01DD 18
01DE 09          TAM             ;PUT IT IN PITCH BUFFER
01DF 56          TCX     FLAGS   ;TEST FLAG
01E0 28
01E1 43          TBITM   UNVO    ;WAS IT UNVOICED BEFORE?
01E2 E4          SBR     VOIC1   ;BRANCH IF NOT
01E3 E5          SBR     VOIC2   ;BRANCH IF SO
01E4 3A  VOIC1   SBITM   NINTP   ;DISABLE INTERPOLATION
01E5 4B  VOIC2   RBITM   UNVO    ;WE WANT VOICING HERE
         *
         * K PARAMETERS DECODING
         *
01E6 56  KPARM   TCX     FLAGS
01E7 28
01E8 46          TBITM   RPT     ;IF BIT REPEAT
01E9 62          BR      SPCEX   :EXIT SPEECH
01EA 86
01EB 00          CLA             ;CLEAR A
01EC 24          GET     K1NB    ;GET K1
01ED 56          TCX     TEMP    ;
01EE 27
01EF 09          TAM             ;SAVE K1 CODE
01F0 50          ACAA    TABK1   ;POINT AT K1 TABLE
01F1 57
01F2 58          LUAA            ;DECODE IT
01F3 56          TCX     K1BF
01F4 1A
01F5 09          TAM             ;PUT IT AWAY
01F6 56          TCX     TEMP
01F7 27
01F9 51          ACAA    TAK1F   ;POINT AT K1 FRACTION TABLE
01FA 0F
01FB 58          LUAA            ;DECODE IT
01FC 56          TCX     FK1BF
01FD 25
01FE 09          TAM             ;PUT IT AWAY
01FF 00          CLA             ;CLEAR A
0200 24          GET     K2NB    ;GET K2
```

```
0201 56          TCX     TEMP
0202 27
0203 09          TAM             ;SAVE K2 CODE
0204 50          ACAA    TABK2   ;POINT AT K2 TABLE
0205 77
0206 58          LUAA            ;DECODE IT
0207 56          TCX     K2BF
0208 1B
0209 09          TAM             ;PUT IT AWAY
020A 56          TCX     TEMP
020B 27
020C 04          TMA             ;GET K2 CODE
020D 51          ACAA    TAK2F   ;POINT AT K2 TABLE
020E 2F
020F 58          LUAA            ;DECODE IT
0210 56          TCX     FK2BF
0211 26
0212 09          TAM             ;PUT IT AWAY
0213 00          CLA             ;CLEAR A
0214 23          GET     K3NB    ;GET K3
0215 50          ACAA    TABK3   ;POINT AT K3 TABLE
0216 97
0217 58          LUAA            ;DECODE IT
0218 56          TCX     K3BF
0219 1C
021A 09          TAM             ;PUT IT AWAY
021B 00          CLA             ;CLEAR A
021C 23          GET     K4NB    ;GET K4
021D 50          ACAA    TABK4   ;POINT AT K4 TABLE
021E A7
021F 58          LUAA            ;DECODE IT
0220 56          TCX     K4BF
0221 1D
0222 09          TAM             ;PUT IT AWAY
0223 56          TCX     FLAGS
0224 28
0225 43          TBITM   UNVO    ;IF UNVOICED GO CLEAR THE REST
0226 62          BR      CLRP1   ;GO K5 K6 K7 K8 K9 K10
0227 73
             *
             * K5 K6 K7 K8 K9 K10
             *
0228 00          CLA             ;CLEAR A
0229 23          GET     K5NB    ;GET K5
022A 50          ACAA    TABK5   ;POINT AT K5 TABLE
```

```
022B B7
022C 58           LUAA             ;DECODE IT
022D 56           TCX      K5BF
022E 1E
022F 09           TAM              ;PUT IT AWAY
0230 00           CLA              ;CLEAR A
0231 23           GET      K6NB    ;GET K6
0232 50           ACAA     TABK6   ;POINT AT K6 TABLE
0233 C7
0234 58           LUAA             ;DECODE IT
0235 56           TCX      K6BF
0236 1F
0237 09           TAM              ;PUT IT AWAY
0238 00           CLA              ;CLEAR A
0239 23           GET      K7NB    ;GET K7
023A 50           ACAA     TABK7   ;POINT AT K7 TABLE
023B D7
023C 58           LUAA             ;DECODE IT
023D 56           TCX      K7BF
023E 20
023F 09           TAM              ;PUT IT AWAY
0240 00           CLA              ;CLEAR A
0241 22           GET      K8NB    ;GET K8
0242 50           ACAA     TABK8   ;POINT AT K8 TABLE
0243 E7
0244 58           LUAA             ;DECODE IT
0245 56           TCX      K8BF
0246 21
0247 09           TAM              ;PUT IT AWAY
0248 00           CLA              ;CLEAR A
0249 22           GET      K9NB    ;GET K9
024A 50           ACAA     TABK9   ;POINT AT K9 TABLE
024B EF
024C 58           LUAA             ;DECODE IT
024D 56           TCX      K9BF
024E 22
024F 09           TAM              ;PUT IT AWAY
0250 00           CLA              ;CLEAR A
0251 22           GET      K10NB   ;GET K10
0252 50           ACAA     TAK10   ;POINT AT K10 TABLE
0253 F7
0254 58           LUAA             ;DECODE IT
0255 56           TCX      K10BF
0256 23
0257 09           TAM              ;PUT IT AWAY
```

```
0258 62            BR      SPCEX
0259 86
                 *
                 * CLEAR ALL PARAMETERS
                 *
025A 00  CLRPR     CLA                 ;GET READY TO CLEAR NEW PARAMETERS
025B 56            TCX     EBF         ;ENERGY NEW VALUE
025C 19
025D 09            TAM                 ;CLEARED
025E 56            TCX     FEBF
025F 24
0260 09            TAM
0261 56            TCX     FK1BF
0262 25
0263 09            TAM
0264 56            TCX     FK2BF
0265 26
0266 09            TAM
0267 56            TCX     K1BF
0268 1A
0269 09            TAM
026A 56            TCX     K2BF
026B 1B
026C 09            TAM
026D 56            TCX     K3BF
026E 1C
026F 09            TAM
0270 56            TCX     K4BF
0271 1D
0272 09            TAM
0273 00            CLRP1   CLA
0274 56            TCX     K5BF
0275 1E
0276 09            TAM
0277 56            TCX     K6BF
0278 1F
0279 09            TAM
027A 56            TCX     K7BF
027B 20
027C 09            TAM
027D 56            TCX     K8BF
027E 21
027F 09            TAM
0280 56            TCX     K9BF
0281 22
```

```
0282 09           TAM
0283 56           TCX     K10BF
0284 23
0285 09           TAM
                *
0286 56 SPCEX     TCX     FLAGS
0287 28
0288 44           TBITM   STRT     ;FIRST TWO FRAMES?
0289 8C           SBR     SPCE1    ;BRANCH IF SO
028A 61           BR      SPDEC    ;GO DO IT ALL OVER AGAIN
028B 7D
028C 45 SPCE1     TBITM   STRT1    ;SECOND FRAME?
028D 62           BR      SPCE2
028E B3
028F 3D           SBITM   STRT1    ;NEXT ONE IS SECOND FRAME
        *
        * THIS SECTION COPIES THE BUFFER INTO PRESENT VALUES

        * PITCH
        *
0290 56           TCX     PPV      ;PITCH PRESENT VALUE
0291 01
0292 0E           XBX              ;POINTER IN B
0293 56           TCX     PBF      ;POINT AT PITCH BUFFER
0294 18
        *
0295 05 PCOPY     TMAIX            ;GET VALUE, POINT AT NEXT BUFFER
0296 0E           XBX              ;POINT AT PRESENT VALUE
0297 09           TAM              ;STORE AS PRESENT VALUE
0298 0F           IXC              ;POINT AT NEXT PRESENT VALUE LOCATION
0299 0F           IXC
029A 0E           XBX              BACK TO BUFFER POINTER
029B 55           XGEC    K10BF 1  DONE?
029C 24
029D 9F           SBR     FCOPY    BRANCH IF SO
029E 95           SBR     PCOPY
        *
029F 56 FCOPY     TCX     FEBF     ;POINT AT ENERGY FRACTIONAL BUFFER
02A0 24
02A1 04           TMA              ;GET VALUE
02A2 56           TCX     FEPV     ;ENERGY FRACTIONAL PRESENT VALUE
02A3 83
02A4 09           TAM
        *
        * K PARAMETERS
```

```
                *
02A5 56                 TCX     FK1BF   ;POINT AT K1 FRACTIONAL BUFFER
02A6 25
02A7 04                 TMA             ;GET VALUE
02A8 56                 TCX     FK1PV   ;K1 FRACTIONAL PRESENT VALUE
02A9 85
02AA 09                 TAM
02AB 56                 TCX     FK2BF   ;POINT AT K2 FRACTIONAL BUFFER
02AC 26
02AD 04                 TMA             ;GET VALUE
02AE 56                 TCX     FK2PV   ;K2 FRACTIONAL PRESENT VALUE
02AF 87
02B0 09                 TAM
02B1 61                 BR      SPDE2   ;GO FILL BUFFER AGAIN
02B2 8B
                *
                * HERE FOR SECOND FRAME
02B3 62         SPCE2   BR      T$IR1   ;GO MOVE BUFFER INTO NEW VALUES
02B4 CE
                *
                * NOW START IT UP
                *
02B5 56         SPCE3   TCX     FLAGS   ;BACK HERE FROM MOVE
02B6 28
02B7 4C                 RBITM   STRT    ;CLEAR STARTUP FLAGS
02B8 4D                 RBITM   STRT1
02B9 56                 TCX     PBF     ;GET PITCH
02BA 18
02BB 04                 TMA             ;INTO A
02BC 56                 TCX     TMVAL   ;TIME INTO X
02BD 1F
02BE 1A                 START
02BF 10                 TXTM            ;START THINGS GOING
02C0 1E                 INTE            ;ENABLE THOSE INTERRUPTS
02C1 61                 BR      SPDE2   ;GO FILL BUFFER AGAIN
02C2 8B
```

```
          *
          *
          *********************************************
          * TIMER INTERRUPT ROUTINE
          * INTERRUPT OCCURS ONCE EACH FRAME
          *
          * TRANSFERS NEW PITCH, ENERGY, AND K VALUES TO
          * PROPER LOCATIONS FROM RAM, READS NEW VALUES IN
          * TO RAM.
          *
          * BLASTS: PTR, B,
          * ALL PRESENT VALUES,
          * ALL NEW VALUES
          *
          *********************************************
02C3 56           INT1    TCX     FLAGS
02C4 28
02C5 39           SBITM   INT     ;SET INTERRUPT FLAG
02C6 40           TBITM   STP     ;LAST FRAME?
02C7 63           BR      NOINT   ;BRANCH IF SO.
02C8 37
02C9 56           TCX     TMVAL   ;LOAD TIMER INTERRUPT VALUE
02CA 1F
02CB 10           TXTM
02CC 56           TCX     FLAGS
02CD 28
          *
          * MOVE NEW VALUES INTO PRESENT VALUES
          *
          * VOICING
          *
02CE 00   T$IR1   CLA
02CF 43           TBITM   UNVO    ;CHECK VOICING
02D0 62           BR      SSSSS   ;BRANCH IF NOT VOICED
02D1 D4
02D2 50           ACAA    1       ;SET VOICING BIT
02D3 01
02D4 5B   SSSSS   TAV             ;PUT IT AWAY
          *
          * PITCH
          *
02D5 56           TCX     PBF     ;GET PITCH BUFFER
02D6 18
02D7 04           TMA
02D8 56           TCX     PNV     ;TELL CHIP ABOUT IT
```

```
02D9  00
                *
02DA  09              TAM               ;ZERO FOR ...
                *
                * ENERGY
                *
02DB  56              TCX      EBF      ;POINT AT ENERGY BUFFER
02DC  19
02DD  04              TMA               ;GET ENERGY NEW VALUE
02DE  56              TCX      ENV
02DF  02
02E0  09              TAM               ;EBF # ENV
                *
02E1  56              TCX      FEBF     ;POINT AT FRACTIONAL ENERGY BUFFER
02E2  24
02E3  04              TMA               ;GET FRACTINAL ENERGY NEW VALUE
02E4  56              TCX      FENV
02E5  82
02E6  09              TAM               ;FEBF # FENV
                *
                * K PARAMETERS
                *
02E7  56              TCX      K1BF     ;POINT AT K1 BUFFER
02E8  1A
02E9  04              TMA               ;GET K1 NEW VALUE
02EA  56              TCX      K1NV
02EB  04
02EC  09              TAM               ;K1BF # K1NV
02ED  56              TCX      FK1BF    ;POINT AT K1 FRACTIONAL BUFFER
02EE  25
02EF  04              TMA               ;GET K1 FRACTIONAL NEW VALUE
02F0  56              TCX      FK1NV
02F1  84
02F2  09              TAM               ;FK1BF # FK1NV
02F3  56              TCX      K2BF     ;POINT AT K2 BUFFER
02F4  1B
02F5  04              TMA               ;GET K2 NEW VALUE
02F6  56              TCX      K2NV
02F7  06
02F8  09              TAM               ;K2BF # K2NV
02F9  56              TCX      FK2BF    ;POINT AT K2 FRACTIONAL BUFFER
02FA  26
02FB  04              TMA               ;GET K2 FRACTIONAL NEW VALUE
02FC  56              TCX      FK2NV
02FD  86
```

```
0799 02FE 09          TAM             ;FK2BF # FK2NV
0800 02FF 56          TCX      K3BF   ;POINT AT K3 BUFFER
     0300 1C
0801 0301 04          TMA             ;GET K3 NEW VALUE
0802 0302 56          TCX      K3NV
     0303 08
0803 0304 09          TAM             ;K3BF # K3NV
0804 0305 56          TCX      K4BF   ;POINT AT K4 BUFFER
     0306 1D
0805 0307 04          TMA             ;GET K4 NEW VALUE
0806 0308 56          TCX      K4NV
     0309 0A
0807 030A 09          TAM             ;K4BF # K4NV
0808 030B 56          TCX      K5BF   ;POINT AT K5 BUFFER
     030C 1E
0809 030D 04          TMA             ;GET K5 NEW VALUE
0810 030E 56          TCX      K5NV
     030F 0C
0811 0310 09          TAM             ;K5BF # K5NV
0812 0311 56          TCX      K6BF   ;POINT AT K6 BUFFER
     0312 1F
0813 0313 04          TMA             ;GET K6 NEW VALUE
0814 0314 56          TCX      K6NV
     0315 0E
0815 0316 09          TAM             ;K6BF # K6NV
0816 0317 56          TCX      K7BF   ;POINT AT K7 BUFFER
     0318 20
0817 0319 04          TMA             ;GET K7 NEW VALUE
0818 031A 56          TCX      K7NV
     031B 10
0819 031C 09          TAM             ;K7BF # K7NV
0820 031D 56          TCX      K8BF   ;POINT AT K8 BUFFER
     031E 21
0821 031F 04          TMA             ;GET K8 NEW VALUE
0822 0320 56          TCX      K8NV
     0321 12
0823 0322 09          TAM             ;K8BF # K8NV
0824 0323 56          TCX      K9BF   ;POINT AT K9 BUFFER
     0324 22
0825 0325 04          TMA             ;GET K9 NEW VALUE
0826 0326 56          TCX      K9NV
     0327 14
0827 0328 09          TAM             ;K9BF # K9NV
0828 0329 56          TCX      K10BF  ;POINT AT K10 BUFFER
     032A 23
```

```
032B 04           TMA               ;GET K10 NEW VALUE
032C 56           TCX      K10NV
032D 16
032E 09           TAM               ;K10BF # K10NV
            *
            * INTERPOLATION
            *
032F 56           TCX      FLAGS
0330 28
0331 44           TBITM    STRT     ;START?
0332 62           BR       SPCE3    ;BRANCH IF SO
0333 B5
0334 42           TBITM    NINTP    ;INTERPOLATE?
0335 B7           SBR      NOINT    ;BRANCH IF NOT
0336 1E           INTE
0337 2F   NOINT   RETI
            *
            * HERE FOR A STOP CODE
            *
0338 1D   STPIT   INTD
0339 1B           STOP
033A 1F           RETN
```

C Program to Initialize the TSP60C20 Speech ROM

```
***********************************************************
* This is the Assembler Source for the                    *
* initialization routine for the                          *
* TSP60C20 speech ROM. It assumes that                    *
* the desired starting byte address is                    *
* located in an arbitrary point in                        *
* RAM. For the purposes of checkout                       *
* it is assumed to be at #10-#11 with                     *
* the most significant byte of the                        *
* address at #10 and the least                            *
* significant byte of the address in                      *
* #11.                                                    *
*                                                         *
* In actual use, the values given                         *
* for HADDR and LADDR in the Equate                       *
* block should be replaced so as to                       *
* point to the actual location in RAM                     *
* used in the program.                                    *
*                                                         *
* After calling this program, you                         *
* can use the standard synthesis                          *
* routine. Just make sure that there                      *
* is at least a 9 instruction cycle                       *
* gap between each GET instruction                        *
*                                                         *
* If you need to use internal speech                      *
* afterwards, you need to do an                           *
* INTRM (internal ROM) instruction                        *
* first.                                                  *
*                                                         *
* The strategy is as follows:                             *
* Pulse M1 High                                           *
* Pulse M0 High                                           *
* Load the 16-bit starting ROM                            *
* address; four bits at a time.                           *
* Pulsing M1 high for each                                *
* nibble. The address in                                  *
* RAM is right shifted one                                *
* bit so as to make the address                           *
* reflect word instead of byte                            *
* boundaries.                                             *
* Burn 16 instruction cycles                              *
* for ROM access cycle.                                   *
```

```
      * Get 8 bits                                            *
      * Burn 9 instruction Cycles                             *
      * If the address in RAM was odd,                        *
      * Get 8 bits to move to                                 *
      * correct byte Boundary.                                *
      * Get 8 bits                                            *
      *                                                       *
      * Although the TSP60C20 is addressed on                 *
      * word (16-bit) boundaries; the                         *
      * address that this subroutine uses                     *
      * is expressed in byte (8-bit)                          *
      * boundaries. The address located at                    *
      * HADDR and LADDR is therefore                          *
      * shifted right one bit before being                    *
      * loaded into the TSP60C20. If the                      *
      * original address located contains a                   *
      * one in the least significant bit                      *
      * position then a GET8 instruction is                   *
      * executed at the end to move one byte                  *
      * further (halfway between word                         *
      * boundaries) in memory.                                *
      *                                                       *
      * The address of the starting byte                      *
      * of ROM is placed in RAM with the                      *
      * most significant byte at HADDR and                    *
      * the least significant byte at                         *
      * LADDR.                                                *
      *                                                       *
      * This routine will be reached by a                     *
      *                                                       *
      * CALL INIT                                             *
      *                                                       *
      * instruction.                                          *
      *                                                       *
      *********************************************************
      *
      *
      *********************************************************
      *                                                       *
      * Equate Block                                          *
      *                                                       *
      *********************************************************
0010  HADDR    EQU    #10    -Most Significant RAM Address Byte
0011  LADDR    EQU    #11    -Least Significant RAM Address Byte
      *
```

```
           ************************************************************
           *                                                          *
           * Start Routine                                            *
           *                                                          *
           ************************************************************
0000 2B    INIT    EXTRM               -Set to External ROM Mode
           *
           * Pulse M1 High
           *
0001 00                    CLA     -Clear A Register
0002 50            ACAA    2       -Set A = 2
0003 02
0004 15            TAPB            -Xfer A to PB
0005 00            CLA             -Clear A
0006 15            TAPB            -Xfer A to PB
           *
           * Pulse M0 High
           *
0007 50            ACAA    1       -Set A = 1
0008 01
0009 15            TAPB            -Xfer A to PB
000A 00            CLA             -Clear A Register
000B 15            TAPB            -Xfer A to PB
           ************************************************************
           *                                                          *
           * Load the least significant byte                          *
           * of the ROM starting address to                           *
           * the A register; then Right shift                         *
           * it once to convert it to a word                          *
           * address; then left shift it to                           *
           * position the least significant                           *
           * nibble correctly in the register                         *
           * with the two least significant                           *
           * bits of the register set to 0.                           *
           * Then add 2 to A register to set                          *
           * M1 high and transfer the result                          *
           * to the PB.                                               *
           *                                                          *
           ************************************************************
000C 56            TCX     LADDR   -Point to LSB of Address
000D 11
000E 04            TMA             -Xfer 8 bit Address
```

```
000F 18          SARA            -Shift A register Right
0010 19          SALA            -Shift A register Left
0011 19          SALA            -Shift A register Left
0012 50          ACAA    2       -Set M1 High
0013 02
0014 15          TAPB            -Xfer A to PB
         ***********************************************************
         *                                                         *
         * Add hex >0FE to A register to set M1                    *
         * to 0 then transfer the result to                        *
         * PB.                                                     *
         *                                                         *
         ********************************************************* *
0015 50          ACAA    #0FE    -Set M1 Low
0016 FE
0017 15          TAPB            -Xfer A to PB
         *
         ***********************************************************
         *                                                         *
         * Load the least significant byte                         *
         * of the ROM starting address to                          *
         * the A register; then right shift                        *
         * it five times and then left shift                       *
         * it twice to position the upper                          *
         * three bits of the address byte                          *
         * correctly in the A register.                            *
         * Then add 2 to A register to set                         *
         * M1 high.                                                *
         *                                                         *
         * The least significant bit of the                        *
         * other byte of the address word                          *
         * will still need to be transferred                       *
         * to the most significant bit of this                     *
         * word before it can be transferred                       *
         * to the PB.                                              *
         *                                                         *
         ***********************************************************
0018 04          TMA             -Xfer 8 bit Address to A Register
0019 18          SARA            -Shift A register right
001A 18          SARA            -Shift A register right
001B 18          SARA            -Shift A register right
001C 18          SARA            -Shift A register right
001D 18          SARA            -Shift A register right
001E 19          SALA            -Shift A register Left
001F 19          SALA            -Shift A register Left
```

```
0020 56          TCX     HADDR   -Point to MSB of Address
0021 10
        ***********************************************************
        *                                                         *
        * This block of code is used to move                      *
        * last bit of the least significant                       *
        * byte of the ROM address to the                          *
        * to the most significant bit of the                      *
        * least significant byte of the ROM                       *
        * address. This is necessary as we                        *
        * right shift the two bytes across                        *
        * byte boundaries.                                        *
        *                                                         *
        ***********************************************************
0022 40          TBITM   1       -Is LSB of Byte=1?
0023 60          BR      ONE     -If Yes, Set Bit
0024 27
0025 60          BR      ZERO    -If No, No action
0026 29
0027 50  ONE     ACAA    32      -Set Bit if necesary
0028 20
0029 50  ZERO    ACAA    2       -Set M1 High
002A 02
002B 15          TAPB            -Xfer A to PB
        ***********************************************************
        *                                                         *
        * Add hex >OFE to A register to set M1                    *
        * to 0 then transfer the result to                        *
        * PB.                                                     *
        *                                                         *
        ***********************************************************
002C 50          ACAA    #OFE    -Set M1 Low
002D FE
002E 15          TAPB            -Xfer A to PB
        *
        ***********************************************************
        *                                                         *
        * Load the most significant byte                          *
        * of the ROM starting address to                          *
        * the A register; then right shift                        *
        * it once to convert it to a word                         *
        * address; then left shift it to                          *
        * position the least significant                          *
        * nibble correctly in the register                        *
        * with the two least significant                          *
```

```
           * bits of the register set to 0.                           *
           * Then add 2 to A register to set                          *
           * M1 high and transfer the result                          *
           * to the PB. *
           *                                                          *
           ************************************************************
002F 04            TMA             -Xfer 8 bit Address to A Register
0030 18            SARA            -Truncate Last Bit (Already Loaded)
0031 19            SALA            -Shift A register Left
0032 19            SALA            -Shift A register Left
0033 50            ACAA    2       -Set M1 High
0034 02
0035 15            TAPB            -Xfer A to PB
           ************************************************************
           *                                                          *
           * Add hex 0FE to A register to set M1                      *
           * to 0 then transfer the result to                         *
           * PB.                                                      *
           *                                                          *
           ************************************************************
           *
0036 50            ACAA    #0FE    -Set M1 Low
0037 FE
0038 15            TAPB            -Xfer A to PB
           ************************************************************
           *                                                          *
           * Load the most significant byte                           *
           * of the ROM starting address to                           *
           * the A register; then right shift                         *
           * it five times and then left shift                        *
           * it twice to position the upper                           *
           * three bits of the address byte                           *
           * correctly in the A register.                             *
           * Then add 2 to A register to set                          *
           * M1 high and transfer the result                          *
           * to the PB.                                               *
           *                                                          *
           ************************************************************
0039 04            TMA             -Xfer 8 bit Address to A Register
003A 18            SARA            -Shift A register right
003B 18            SARA            -Shift A register right
003C 18            SARA            -Shift A register right
003D 18            SARA            -Shift A register right
003E 18            SARA            -Shift A register right
```

```
003F 19           SALA            -Shift A register Left
0040 19           SALA            -Shift A register Left
0041 50           ACAA     2      -Set M1 High
0042 02
0043 15           TAPB            -Xfer A to PB
            *
            *************************************************************
            *                                                           *
            * Add hex >0FE to A register to set M1                      *
            * to 0 then transfer the result to                          *
            * PB.                                                       *
            *                                                           *
            *************************************************************
            *
0044 50           ACAA     #0FE   -Set M1 Low
0045 FE
0046 15           TAPB            -Xfer A to PB
            *
            *************************************************************
            *                                                           *
            * Burn 16 Instruction cycles for the                        *
            * TSP60C20 internal access cycle.                           *
            *                                                           *
            *************************************************************
0047 00           CLA             -Clear A register
0048 50     BRN16 ACAA     1      -Increment A register
0049 01
004A 54           ANEC     5      -Has Loop timed out?
004B 05
004C 60           BR       BRN16  -If not timed out, repeat
004D 48
            *
            *************************************************************
            *                                                           *
            * Load 8 Bits                                               *
            *                                                           *
            *************************************************************
004E 27           GET      8
            *
            *************************************************************
            *                                                           *
            * Burn 9 Instruction cycles                                 *
            * (Actually burns 10 cycles)                                *
            *                                                           *
            *************************************************************
```

```
          *
004F 00           CLA             -Clear A register
0050 50   LOOP    ACAA    1       -Increment A register
0051 01
0052 54           ANEC    3       -Has Loop timed out?
0053 03
0054 60           BR      LOOP    -If no, repeat
0055 50
          *
          ************************************************************
          *                                                          *
          *                                                          *
          * This Block of code inserts an                            *
          * extra Get8 instruction if the                            *
          * Address in ROM is odd so as                              *
          * to shift the data to the Byte
          * Boundary.                                                *
          *                                                          *
          ********************************************************** *
0056 56           TCX     LADDR   -Point to LSB of Address
0057 11
0058 40           TBITM   1       -Is LSB of Byte=1?
0059 60           BR      ODD     -If Yes, ODD
005A 5D
005B 60           BR      LEGAL   -Else LEGAL
005C 65
005D 27   ODD     GET     8       -Move to Odd Boundary
          ************************************************************
          *                                                          *
          * Burn 9 Instruction cycles                                *
          * (Actually burns 10 cycles)                               *
          *                                                          *
          ********************************************************** *
          *
005E 00           CLA             -Clear A register
005F 50   BRN09   ACAA    1       -Increment A register
0060 01
0061 54           ANEC    3       -Has Loop timed out?
0062 03
0063 60           BR      BRN09   -If no, repeat
0064 5F
          *
          * Load 8 Bits
          *
0065 27   LEGAL   GET     8       -Next GET8 will give Legal Data
```

```
                ************************************************************
                *                                                          *
                * Burn 9 Instruction cycles                                *
                * (Actually burns 10 cycles)                               *
                *                                                          *
                ************************************************************
                *
     0066 00            CLA                 -Clear A register
     0067 50    BRN10   ACAA      1         -Increment A register
     0068 01
     0069 54            ANEC      3         -Has Loop timed out?
     006A 03
     006B 60            BR        BRN10     -If no, repeat
     006C 67
                *
                *
                *
     006D 1F            RETN                -Exit Subroutine
```