MSP53C391 and MSP53C392 Speech Synthesizers User's Guide

May 2000

SPSU016A

Read This First

How to Use This Manual

This document contains the following chapters:

- ❑ Chapter 1 –Introduction to the MSP53C391 and MSP53C392 Speech Synthesizers
- ❑ Chapter 2 –MSP53C391 Hardware Description
- ❑ Chapter 3 –MSP53C392 Hardware Description
- ❑ Chapter 4 –MSP53C391 and MSP53C392 Software Description
- ❑ Appendix A –Editing Tools and Data Preparation
- ❑ Appendix B –FM Synthesis
- ❑ Appendix C –Listing of FMequM2.inc
- ❑ Appendix D –MSP53C391/392 Timing Considerations
- ❑ Appendix E –Listing of FM2INTR1.inc
- ❑ Appendix F –MSP53C391 and MSP53C392 Data Sheet

Related Documentation From Texas Instruments

MSP50x3x Mixed-Signal Processor User’s Guide
(Literature Number SLOU006B)

Contents

Figures

Tables

Chapter 1

Introduction to the MSP53C391 and MSP53C392 Speech Synthesizers

Topic | Page

1.1 Description

MSP53C391 and MSP53C392 are standard slave synthesizers from Texas Instruments that accept compressed speech data from another microprocessor and produce speech with that data. This allows the MSP53C391 and MSP53C392 to be used with a master microprocessor in the various speech products including electronic learning aids, games, and toys.

The TI MSP53C391 and MSP53C392 support several different speech synthesis algorithms to permit tradeoffs to meet the different price performance requirements of different markets. They also incorporate a two-channel FM synthesis routine for music generation.

Both the MSP53C391 and MSP53C392 are special programs that run on the MSP50C3x device. For more information about the MSP50C3x, please refer to the *MSP50x3x User’s Guide* (literature number: SLOU006B)

1.2 Features

❑ Wide ranges of algorithms are incorporated in one chip. This allows the user to choose from a low bit rate to high-quality synthesizing routines for their application. Algorithms included are:

  LPC 5220, LPC D6
  MELP v4.1
  CELP v3.4, 4.2 kbps, 4.8 kbps, 6.2 kbps, 8.6 kbps, 10.7 kbps
  8-bit PCM
  FM II

❑ Software selectable 8-kHz or 10-kHz speech sample rate

❑ Three different interface options to support different pin count requirements

❑ 32-Ω speaker direct drive capability

❑ Internally generated clock requires no external components

❑ Maximum 10-µA standby current in sleep mode

❑ Digital volume control

❑ Built-in, two general-purpose output pins for MSP53C391 pin expansion

1.3 MSP53C391 and MSP53C392 Comparison

The MSP53C391 is optimized to support a 4-bit wide data transfer protocol. The MSP53C392 is optimized to support an 8-bit wide data transfer protocol.

The use of the 4-bit wide protocol in the MSP53C391 frees up some I/O pins that can be used for other purposes. These pins (EOS and BUSY) can be used to simplify the interface by minimizing the need to periodically poll the MSP53C391 for its current status.

The use of the 8-bit wide protocol in the MSP53C392 provides a more efficient data transfer.

A detailed comparison of the two devices is listed in Table 1–1.

Table 1–1. MSP53C391 and MSP53C392 Comparison

	MSP53C391	**MSP53C392**
Number of Data Lines	4 bit	8 bit
Number of Control Lines	2 (Strobe & R/W)	2 (Strobe & R/W)
Data Request	Supported	N/A
Separate EOS Line for Detecting End-of-Speech	Supported	N/A
Pin Expansion	Two expansion pins	N/A

1.4 Pin Assignments and Description

Figure 1–1 shows the pin assignments for the MSP53C391. Table 1–2 provides pin functional descriptions. Figure 1–2 shows the pin assignments for the MSP53C392. Table 1–3 shows the pin functional descriptions.

Figure 1–1. MSP53C391 Pin Assignments

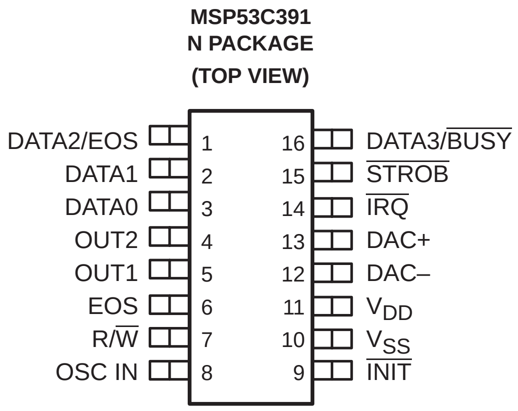


Table 1–2. MSP53C391 Terminal Functions

PIN NAME	PIN NO.	I/O	DESCRIPTION
DAC+	13	O	D/A output. This output pulses high for positive output values. It remains low when negative values are output.
DAC–	12	O	D/A output. This output pulses high for negative output values. It remains low when positive values are output.
DATA 0–3	3,2,1,16	I/O	Data lines
EOS	6	O	End of speech signal. Output high when end of speech is reached.
$\overline{INIT}$	9	I	Initialize input. When $\overline{INIT}$ goes low, the clock stops, the MSP53C391 goes into low-power mode, the program counter is set to zero, and the contents of the RAM are retained. An $\overline{INIT}$ pulse of 1 µs is sufficient to reset the processor.
$\overline{IRQ}$	14	O	Negative-edge trigger interrupt request line. Connect to the external interrupt of the master MCU for interrupt mode operation.
OUT 1–2	5,4	O	General-purpose output ports used for pin expansion
OSC IN	8	I	This signal should be connected to V_{SS}.
R/$\overline{W}$	7	I	Read/write select signal. Set high for read operations or cleared low for write operations by the master processor.
$\overline{STROB}$	15	I	Strobe signal for read and write operations. Pulsed low for read or write operations
V_{DD}	11	–	5-V nominal supply voltage
V_{SS}	10	–	Ground pin

Figure 1–2. MSP53C392 Pin Assignments

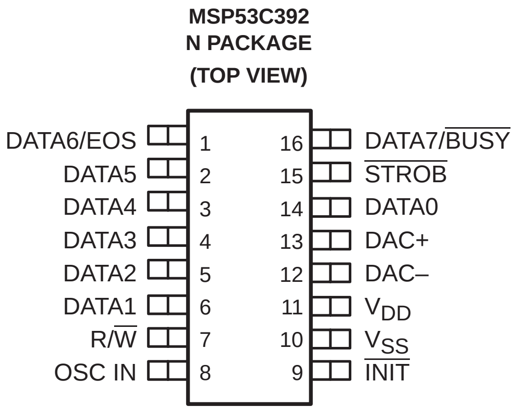


Table 1–3. MSP53C392 Terminal Functions

PIN NAME	PIN NO.	I/O	DESCRIPTION
DAC+	13	O	D/A output. This output pulses high for positive output values. It remains low when negative values are output.
DAC–	12	O	D/A output. This output pulses high for negative output values. It remains low when positive values are output.
DATA 0–7	14,6,5,4, 3,2,1,16	I/O	Data lines
$\overline{INIT}$	9	I	Initialize input. When $\overline{INIT}$ goes low, the clock stops, the MSP53C392 goes into low-power mode, the program counter is set to zero, and the contents of the RAM are retained. An $\overline{INIT}$ pulse of 1 µs is sufficient to reset the processor.
OSC IN	8	I	This signal should be connected to Vss.
R/$\overline{W}$	7	I	Read/write signal
$\overline{STROB}$	15	I	Strobe signal for read/write
V_{DD}	11	–	5-V nominal supply voltage
V_{SS}	10	–	Ground pin

1.5 D/A Information

Two-Pin Push Pull (Option 1) is selected in MSP53C391 and MSP53C392 that can directly drive a 32-Ω speaker. Please refer to the *MSP50x3x Mixed Signal Processor Users Guide* (literature number: SPSU006B) for more information on the D/A and amplifier circuit.

1.6 Algorithms Supported

❑ LPC: D6 and 5220 format. Data rates 1.5 to 3 kbps at an 8-kHz sample rate

❑ MELP: Data rates range form 2kbps ~ 3.5 kbps at an 8-kHz sample rate

❑ CELP: Data rates can be selected form 4.2 kbps ~ 10.7kbps at an 8-kHz sample rate

❑ PCM: 8 bit. Data rates is 64 kbps for 8 kHz sampling

❑ FM: Frequency modulation for two-channel musical instrument synthesis.

Chapter 2

MSP53C391 Hardware Description

Topic **Page**

2.1 MSP53C391 Interface Overview

The MSP53C391 accepts data from the master microprocessor across four data lines. The transfer of data is controlled by two control lines (R/$\overline{\text{W}}$ and $\overline{\text{STROB}}$). The data is loaded to an internal buffer and the synthesis process reads the data from the internal buffer as needed. The MSP53C391 signals that it is ***not*** ready to accept data when the buffer is full, or there is some other condition that would prevent the MSP53C391 from accepting new data. This signal is communicated to the master microprocessor using either the $\overline{\text{IRQ}}$ or $\overline{\text{BUSY}}$ signals.

Depending on the number of available pins on the master microprocessor, three different connection options are provided to connect the master microprocessor to the MSP53C391. Whichever method is used, two operations must be accomplished: 1) Determining if the MSP53C391 is ready to accept new data, and if it is ready, 2) writing new data to the MSP53C391.

Two control lines are provided to enable the master microprocessor to accomplish these two tasks, $\overline{\text{STROB}}$ and R/$\overline{\text{W}}$.

The R/$\overline{\text{W}}$ line determines whether a read or a write operation is done to the MSP53C391 when the $\overline{\text{STROB}}$ is pulsed low. If the R/$\overline{\text{W}}$ is high, then a read from the MSP53C391 is done when the $\overline{\text{STROB}}$ is pulsed low. If the R/$\overline{\text{W}}$ is low, then data is written to the MSP53C391 when the $\overline{\text{STROB}}$ is pulsed low.

Two signals are provided to determine if the MSP53C391 is ready or not ready to accept new data. The $\overline{\text{BUSY}}$ signal shares the same pin as the DATA3 signal. During a read operation, this signal goes high to signal that the MSP53C391 is ready for a write operation. If this signal is low during a read operation, then the MSP53C391 is ***not*** ready for a write operation.

An alternative to polling the $\overline{\text{BUSY}}$ signal is provided by the $\overline{\text{IRQ}}$ signal. This signal goes from high to low when the MSP53C391 is ready for a write operation.

The EOS signal indicates whether or not the end-of-speech has been reached by the synthesis process. It is set high by the MSP53C391 when the stop code in the data stream is reached. This signal is provided on two pins. It can be read directly on pin 6 (EOS), or during a read operation on pin 1 (DATA2/EOS).

Three methods are provided for interfacing the MSP53C391 to various microprocessors. This allows the designer to make trade-offs between the number of device pins being used and the algorithm complexity for the interface to the master microprocessor.

2.2 Signal Description

Table 2–1. MSP53C391 Signal Description

Pin		Description
Name	No.	
DAC+ DAC–	12 13	PDM-style DAC used for speech output.
DATA3/$\overline{\text{BUSY}}$	16	$\overline{\text{BUSY}}$ signal can be obtained on DATA 3 during read operation. A high signal indicates that the MSP53C391 is ***not*** busy and ready to accept data. A low signal indicates that the MSP53C391 is busy and the master should not write any command or data to MSP53C391.
DATA2/EOS	1	The EOS signal can be obtained on the DATA2 pin during a read operation. This signal is normally low, but goes high when the end-of-speech code is reached in the data stream.
$\overline{\text{STROB}}$	15	This is an active low strobe signal for the reading and writing operation from the master microprocessor. The data to be read is available when the strobe is active (low) for the read operation. The data on the data line is latched into the MSP53C391 on the raising edge of the strobe signal for the write operation.
R/$\overline{\text{W}}$	7	Read/write signal from master microprocessor. A high signal for a read operation and a low signal for a write operation.
$\overline{\text{IRQ}}$	14	When the data latched into the MSP53C391 is read and the MSP53C391 device is ready to accept more data, a negative edge interrupt signal is generated to interrupt the master. For proper operation of the interrupt function, a negative edge triggered external interrupt input pin is required on the master microprocessor.
EOS	6	This is an active high output signal that is asserted when end-of-speech is reached. It indicates that the speech synthesis process is finished. When a high is detected on the EOS line by the master microprocessor, dummy bytes are written to the MSP53C391 to reset the EOS. The next transfer can then be initiated after the EOS was de-asserted. EOS also appears on the DATA2 pin during a read operation for adopting different interfacing methods.
OUT1–2	5,4	General-purpose output port that can be controlled by the master microprocessor.
DATA 0–3	3,2,1,16	4-bit bidirectional data line
$\overline{\text{INIT}}$	9	Reset signal. A low pulse to reset the chip. It can also be used to stop the MSP53C391 operation during speech synthesis. Following the rising edge of the $\overline{\text{INIT}}$ pulse, a delay of up to 5 ms will be required to permit the MSP53C391 to completely initialize its internal condition.

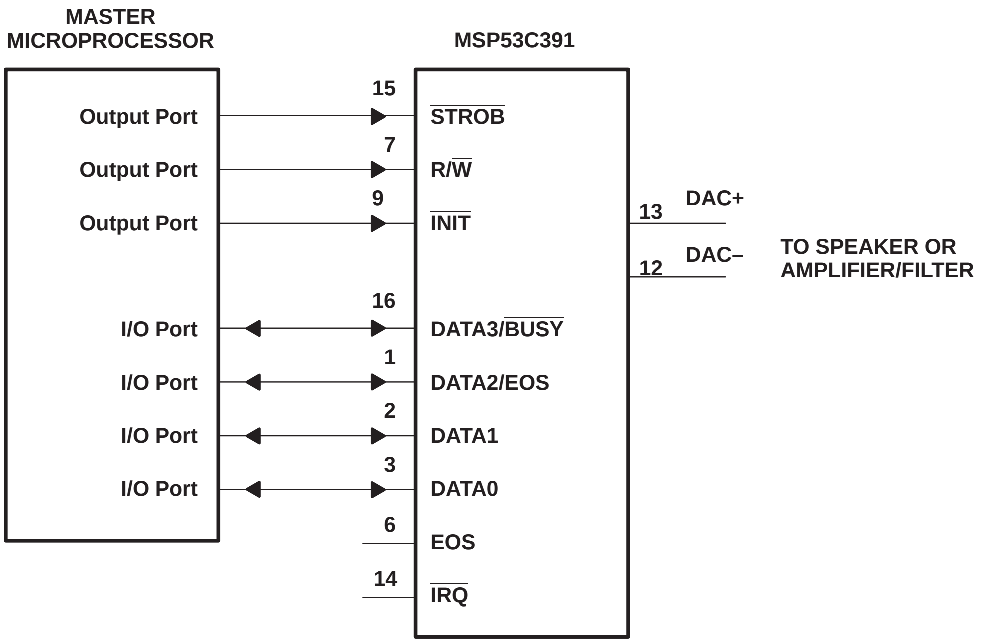
2.3 Master Microprocessor Interface Description

2.3.1 Method 1: Polling

This method is used when it is important to minimize the total number of interface pins between the master microprocessor and the MSP53C391. A total of three control lines and 4 data lines are required for this method. The two status bits can be read from the MSP53C391 by manipulating the R/$\overline{\text{W}}$ and $\overline{\text{STROB}}$ lines and reading the data lines.

The interfacing diagram is shown in Figure 2–1:

Figure 2–1. MSP53C391 Interfacing Diagram (Method 1: Polling)



NOTES: A.

$\overline{\text{STROB}}$:	Active low strobe signal
R/$\overline{\text{W}}$:	Read/write signal
DATA 0–3:	4-bit data line
$\overline{\text{BUSY}}$:	Active low busy signal form MSP53C391. A high signal indicates that the MSP53C391 is **not** busy and is ready to accept data.
EOS:	End-of-speech data. A high signal indicates end-of-speech. Two bytes of dummy data writen resets the EOS to low.
$\overline{\text{INIT}}$:	Active low reset signal. The master microprocessor should issue a reset signal to MSP53C391 after power up to properly initialize the MSP53C391 device.

B. When not being used, the EOS and IRQ pins should be left unconnected.

Read Operation

1) The master microprocessor sets R/$\overline{\text{W}}$ high to indicate a read operation.
2) The master microprocessor sets $\overline{\text{STROB}}$ to low and reads the state of $\overline{\text{BUSY}}$ and EOS.
3) The master microprocessor sets $\overline{\text{STROB}}$ high.

If the $\overline{\text{BUSY}}$ signal was high in step 2, the MSP53C391 is ***not*** busy and is ready to accept a write operation. If the $\overline{\text{BUSY}}$ signal was low in step 2, the MSP53C391 is ***not*** ready to accept a write operation and the read operation should be repeated until $\overline{\text{BUSY}}$ is asserted high.

If EOS was high in step 2, the synthesis process has reached the end of the speech data stream. In this case, the master microprocessor should stop trying to send data and reset the MSP53C391 to allow it to accept additional commands or synthesis data.

The frequency of the polling operation should be optimized to the data rate of the algorithm being used to synthesize speech. If the polling operation is too frequent, the MSP53C391 spends too much time servicing the polling operation and the quality of the synthetic speech may be affected. If the polling operation is too infrequent, the internal buffer may run out of data and the synthesis process can become corrupted. Normally, a polling frequency of four times the bit rates of the speech data provides optimal transfer characteristics.

Example:

For 6.2 kbps CELP the frequency of polling would be

$$\frac{6.2}{n} \times 4$$

n = the number of bits transfered at a time

Write Operation

1) The master processor should determine that the MSP53C391 is ready to accept data by reading the $\overline{\text{BUSY}}$ signal as described previously.
2) The master microprocessor clears R/$\overline{\text{W}}$ low to indicate a write operation
3) The master microprocessor presents valid data to the four data pins (DATA0 – DATA3).
4) The master microprocessor pulses the $\overline{\text{STROB}}$ signal low and then high to latch the data to the MSP53C391 input data latch.
5) The master microprocessor should do a read operation to determine that the MSP53C391 is ready to accept additional data before attempting to write more data.

If the EOS signal is asserted high during the read operation, the end-of-speech has been reached and a reset operation should be performed prior to sending new commands or speech data. The reset can be done in one of two ways: Pulsing the $\overline{\text{INIT}}$ pin low and then waiting for the MSP53C391 to re-initialize itself or by writing two dummy bytes as described in the following.

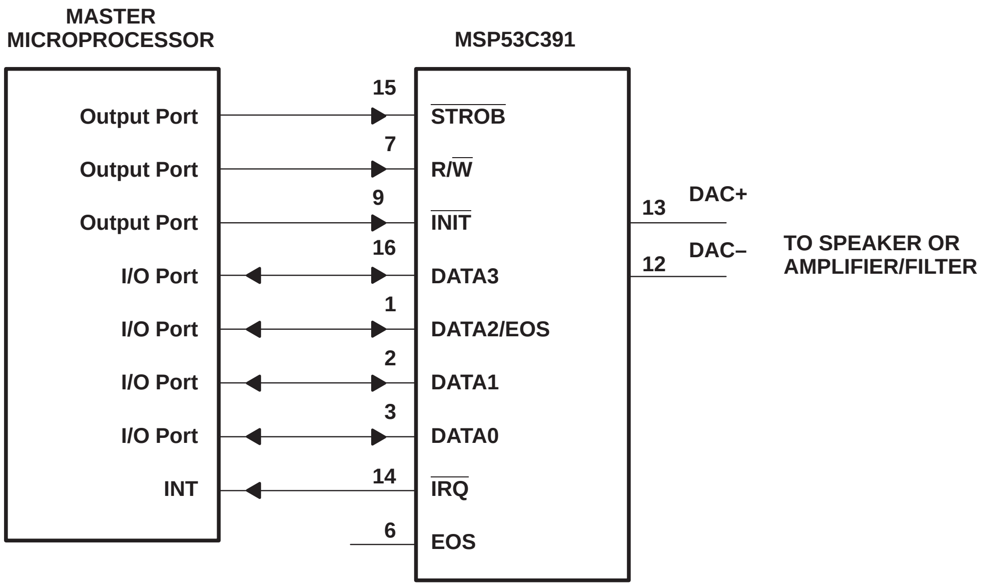
RESET Operation

1) Perform a read operation to determine that both the EOS and $\overline{\text{BUSY}}$ signals are high.
2) If both the EOS and $\overline{\text{BUSY}}$ signals are high, write 2 bytes of dummy data to the MSP53C391 by repeating the write operation four times as described previously.

2.3.2 Method 2: Interrupt 1

In this method, the $\overline{\text{IRQ}}$ pin of the MSP53C391 is connected to an external interrupt input pin of the master microprocessor. When the MSP53C391 is ***not*** busy and is ready to accept data, the $\overline{\text{IRQ}}$ signal goes low and provides a negative edge to trigger an interrupt in the master processor. This minimizes the need to constantly poll the MSP53C391 while waiting for it to become ready to accept new data.

Figure 2–2. MSP53C391 Interfacing Diagram (Method 2: Interrupt 1)



NOTE A: $\overline{\text{IRQ}}$: Negative edge interrupt to master microprocessor when MSP53C391 is not busy and ready to accept data.

Read Operation

1) The master microprocessor sets R/$\overline{\text{W}}$ high to indicate a read operation.

2) The master microprocessor sets $\overline{\text{STROB}}$ to low and reads the state of $\overline{\text{BUSY}}$ and EOS.

3) The master microprocessor sets $\overline{\text{STROB}}$ high.

The EOS is used to signal that the end-of-speech has been reached. In this case, the master microprocessor should stop trying to send data and act to reset the MSP53C391 so as to prepare it to accept additional commands or synthesis data.

In this method, the $\overline{\text{BUSY}}$ signal is not normally used. Instead, the $\overline{\text{IRQ}}$ signal is used to signal the need for new speech data. It pulses low then high to produce a negative edge signal to the master microprocessor when the MSP53C391 becomes ready to accept a new write operation. It will remain low until a new nibble is written. The master microprocessor should immediately initiate a write operation when the $\overline{\text{IRQ}}$ signal goes low. If the master microprocessor delays for too long a time before writing new data, it is possible that the buffer will empty and the synthesis process will be interrupted or the quality of speech will be degraded.

Write Operation

1) The master microprocessor should clear all pending interrupts and enable the external interrupt.

2) The master microprocessor writes the first nibble of data by presenting valid data on DATA0 – DATA3, setting R/$\overline{\text{W}}$ low to indicate a write operation and pulsing $\overline{\text{STROB}}$ low and high to latch the data into the MSP53C391 input latch.

Subsequent data is written following the falling edge of the $\overline{\text{IRQ}}$ signal.

3) The master microprocessor waits for a falling edge on the $\overline{\text{IRQ}}$ signal.

4) Master microprocessor sets R/$\overline{\text{W}}$ high and pulses the $\overline{\text{STROB}}$ to read the EOS signal.

5) The master microprocessor clears R/$\overline{\text{W}}$ low to indicate a write operation

6) The master microprocessor presents valid data (first nibble of the dummy data if EOS is high or nibble of speech data if EOS is low) to the four data pins (DATA0 – DATA3).

7) The master microprocessor pulses the $\overline{\text{STROB}}$ signal low and then high to latch the data to the MSP53C391 input data latch.

8) A read operation should be performed just before each write operation to ensure that the end-of-speech has not been reached.

If the EOS signal is asserted high during the read operation, the end-of-speech has been reached and a reset operation should be performed prior to sending new commands or speech data. The reset can be done in one of two ways: Pulsing the $\overline{\text{INIT}}$ pin low and then waiting for the MSP53C391 to re-initialize itself or by writing two dummy bytes as described the following.

RESET Operation

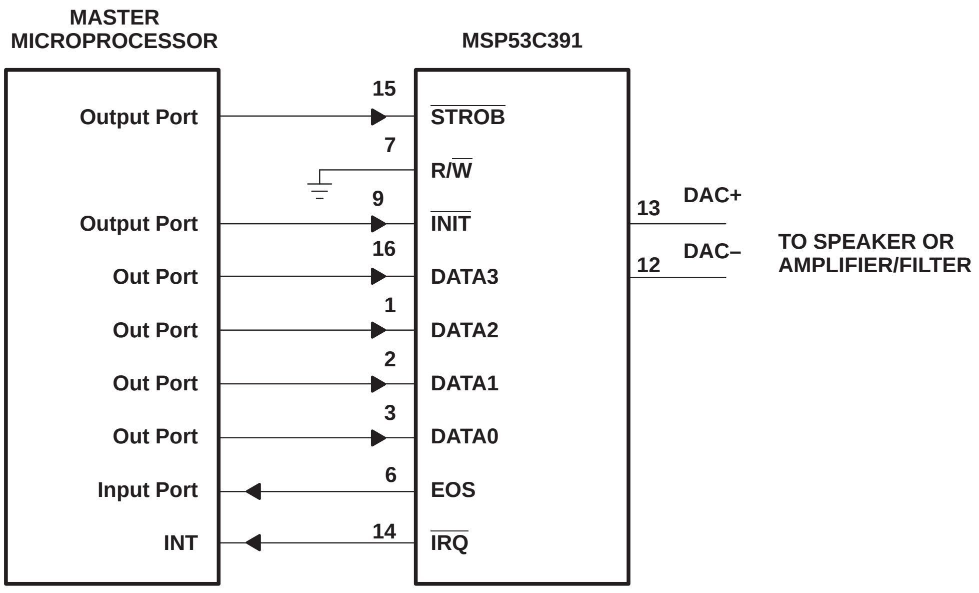
1) Perform a read operation to determine that the EOS signal is high.

2) If the EOS signal is high, write 2 bytes of dummy data to the MSP53C391 by repeating the write operation four times as described previously.

2.3.3 Method 3: Interrupt 2

This method is similar to method 2. The only difference is performing the read operation is not necessary because the EOS and $\overline{\text{IRQ}}$ are available for direct reads.

Figure 2–3. MSP53C391 Interfacing Diagram (Method 3: Interrupt 2)



Write Operation

1) The master microprocessor should clear all pending interrupts and enable the external interrupt.

2) The master microprocessor writes the first nibble of data by presenting valid data on DATA0 – DATA3, tying R/$\overline{\text{W}}$ to ground indicates a write operation and pulsing $\overline{\text{STROB}}$ low and high to latch the data into the MSP53C391 input latch.

Subsequent data is written following the falling edge of the $\overline{\text{IRQ}}$ signal.

3) The master microprocessor waits for a falling edge on the $\overline{\text{IRQ}}$ signal.

4) The master microprocessor checks the EOS signal to verify that the end-of-speech has not been reached. If the EOS is high, the end-of-speech has been reached and the master microprocessor should stop trying to send data and should reset the MSP53C391 as described in the following. If the EOS is low, the end-of-speech has not been reached and the write operation should continue with step 5.

5) Tie R/$\overline{\text{W}}$ to ground indicates a write operation

6) The master microprocessor presents valid data to the four data pins (DATA0 – DATA3).

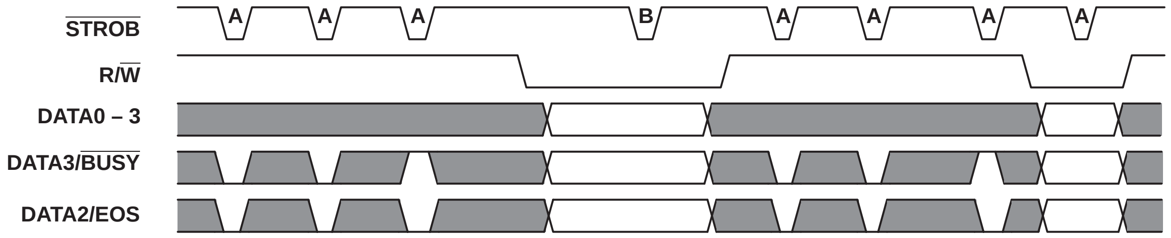
7) The master microprocessor pulses the $\overline{\text{STROB}}$ signal low and then high to latch the data to the MSP53C391 input data latch.

If the EOS signal is asserted high in step 4 (shown previously), the end-of-speech has been reached and a reset operation should be performed prior to sending new commands or speech data. The reset can be done in one of two ways: Pulsing the $\overline{\text{INIT}}$ pin low and then waiting for the MSP53C391 to re-initialize itself or by writing two dummy bytes to the MSP53C391.

2.4 Master Microprocessor Interface Timing

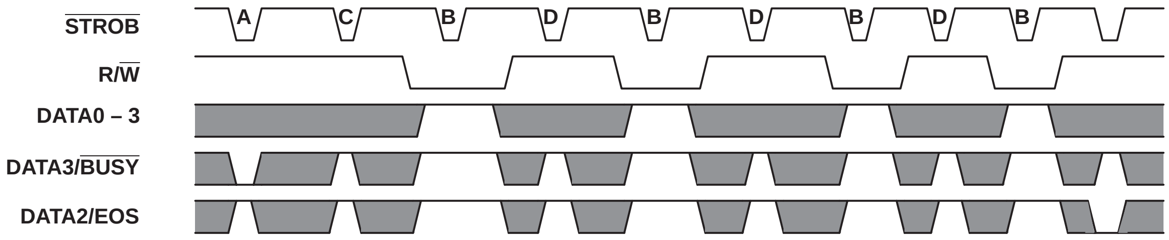
2.4.1 Timing Method 1: Polling

Data Transfer



NOTE A: State A: Polling the status by reading the $\overline{\text{BUSY}}$ and EOS
State B: Write operation

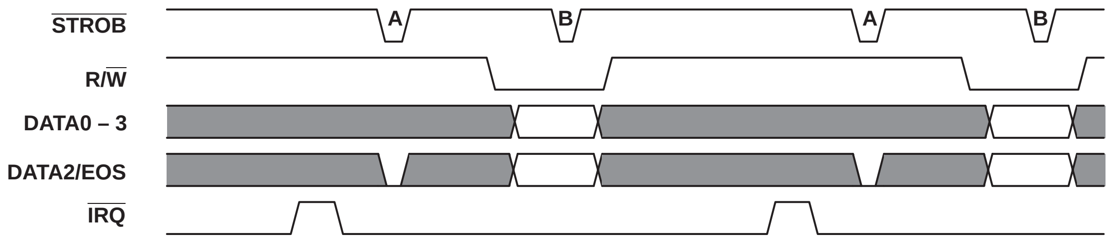
End-of-Speech



NOTE A: State A: EOS detected by reading DATA 2/EOS
State B: Dummy write. A 4-nibble dummy write resets the EOS for the next transfer.
State C: Wait until the part is ready to accept dummy data ($\overline{\text{BUSY}}$ high).
State D: Check to see if the device is busy or not.

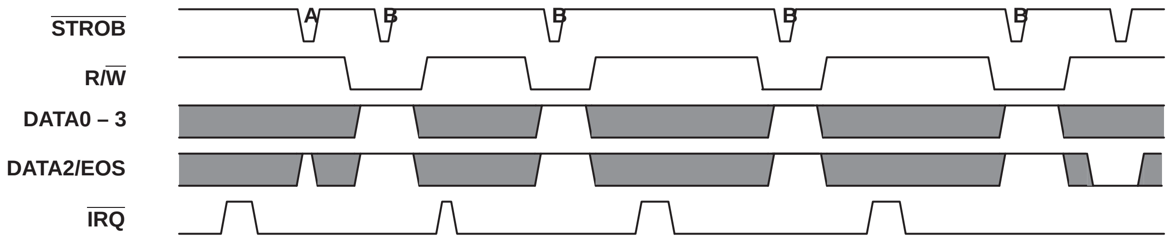
2.4.2 Timing Method 2: Interrupt 1

Data Transfer



NOTE A: State A: Read the EOS state
State B: Write operation

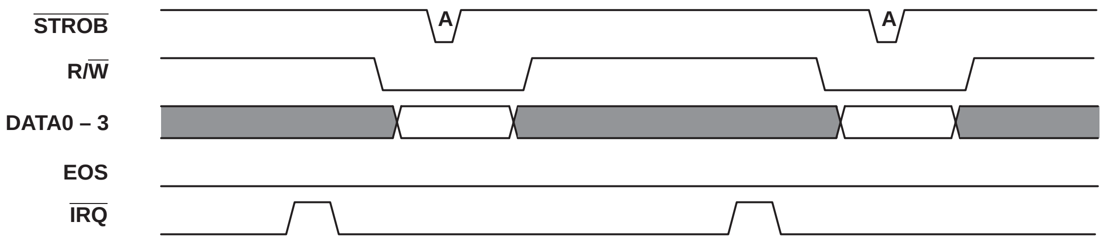
End-of-Speech



NOTE A: State A: EOS detected by read DATA2/EOS
State B: Dummy write. A 4-byte dummy write resets the EOS for the next transfer.

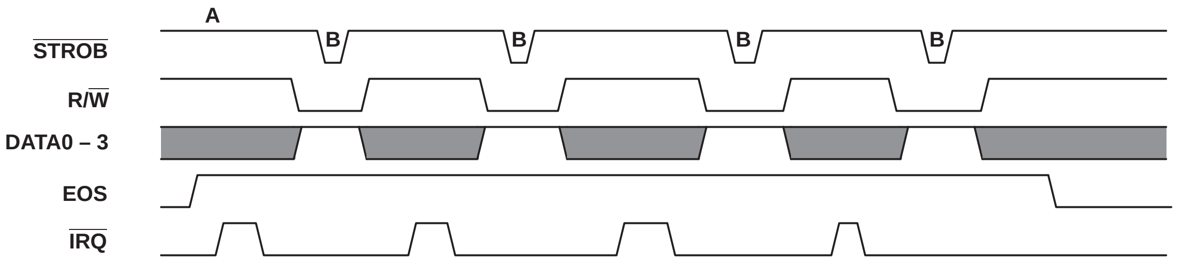
2.4.3 Timing Method 3: Interrupt 2

Data Transfer



NOTE A: State A: Write operation

End-of-Speech



NOTE A: State A: EOS detected by read on pin 6
State B: Dummy write. A 4-nibble dummy write resets the EOS for the next transfer.

2.5 MSP53C391 Device Initialization

For proper operation, the MSP53C391 device should be initialized by sending the following command sequence of bytes:

F,F,F,F,0,A,0,1,0,0,F,F,F,F,F,F

Following this command sequence, the normal command sequence options are available as described in Section 4.2 and onwards.

The function of this sequence is to properly initialize the synthesis engine by speaking a short selection of LPC prior to speaking selections using other synthesis algorithms.

This initialization needs to be performed:

1) After you apply power to the device, or

2) When you reset the part by toggling the INIT pin.

Chapter 3

MSP53C392 Hardware Description

3.1 Interface Overview

The MSP53C392 accepts data from the master microprocessor across the eight data lines. The transfer of data is controlled by two control lines (R/$\overline{\text{W}}$ and $\overline{\text{STROB}}$). The data is loaded to an internal buffer and the synthesis process reads the data from the internal buffer as needed. The MSP53C392 signals that it is ***not*** ready to accept data when the buffer is full, or there is some other condition that would prevent the MSP53C392 from accepting new data. This signal is communicated to the master microprocessor using $\overline{\text{BUSY}}$ signal.

The MSP53C392 accepts data across an 8-bit wide data connection that is controlled using two control lines (R/$\overline{\text{W}}$ and $\overline{\text{STROB}}$). Two operations must be accomplished: 1) Determining if the MSP53C392 is ready to accept new data, and if it is ready, 2) writing new data to the MSP53C392.

Two control lines are provided to enable the master microprocessor to accomplish these two tasks, $\overline{\text{STROB}}$ and R/$\overline{\text{W}}$.

The R/$\overline{\text{W}}$ line determines whether a read or a write operation is done to the MSP53C392 when the $\overline{\text{STROB}}$ is pulsed low. If the R/$\overline{\text{W}}$ is high, than a read from the MSP53C392 is done when the $\overline{\text{STROB}}$ is pulsed low. If the R/$\overline{\text{W}}$ is low, then data is written to the MSP53C392 when the $\overline{\text{STROB}}$ is pulsed low.

The $\overline{\text{BUSY}}$ signal shares the same pin as the DATA7 signal. During a read operation, this signal goes high to signal that the MSP53C392 is ready for a write operation. If this signal is low during a read operation, then the MSP53C392 is ***not*** ready for a write operation.

The EOS signal shares the same pin as the DATA6 signal. During a read operation, this signal normally goes low, but goes high to signal that the MSP53C392 has encountered an end-of-speech code in the data stream.

A negative going pulse on the $\overline{\text{INIT}}$ line can be used to reset the device. An $\overline{\text{INIT}}$ pulse of 1 µs is enough to reset the device. Following the rising edge of the $\overline{\text{INIT}}$ pulse, a delay of up to 5 ms will be required to permit the MSP53C391 and MSP53C392 to completely initialize its internal condition.

3.2 Signal Description

Table 3–1. MSP53C392 Signal Description

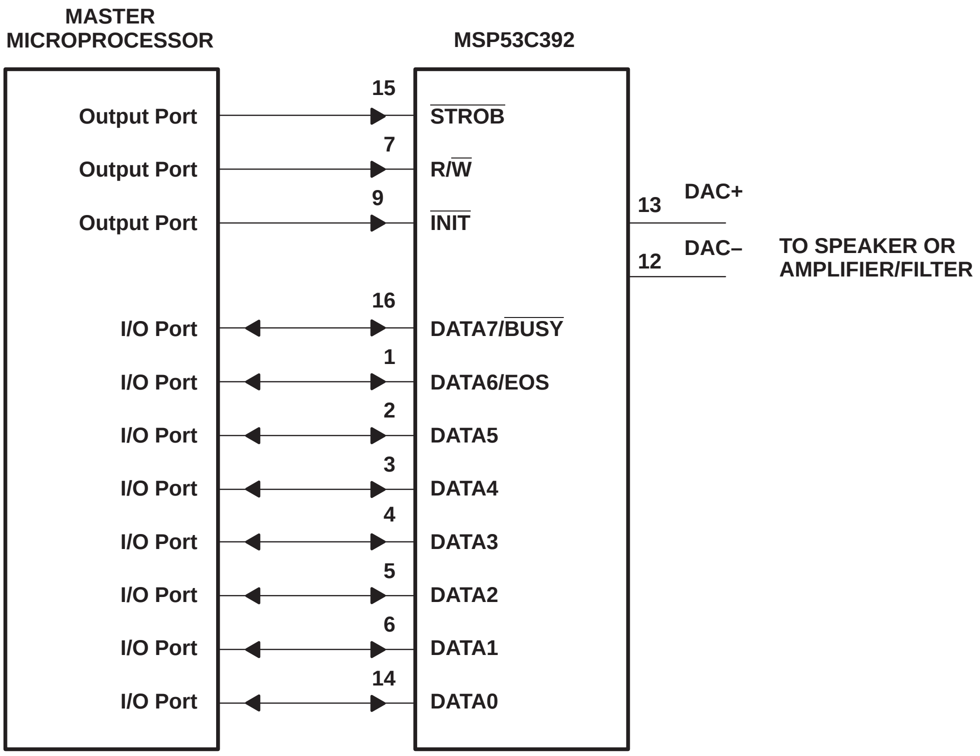
Pin		Description
Name	No.	
DAC+ DAC–	12 13	PDM-style DAC used for speech output.
DATA7/$\overline{\text{BUSY}}$	16	The $\overline{\text{BUSY}}$ signal can be obtained on DATA 7 during a read operation. A high signal indicates that the MSP53C392 is ***not*** BUSY and ready to accept data. A low signal indicates that the MSP53C392 is BUSY and master should not write any command or data to MSP53C392.
DATA6 / EOS	1	The EOS signal can be obtained on DATA 6 during a read operation. This is an active high signal that is asserted when end-of-speech is reached. It indicates that the speech synthesis is finished. When a high is detected on EOS by the master microprocessor, the MSP53C392 should be reset.
R/$\overline{\text{W}}$	7	Read/write signal from master microprocessor. A high signal for a read operation and a low signal for a write operation.
$\overline{\text{STROB}}$	15	This is an active low strobe signal for the reading and writing operation form master microprocessor. The data to be read is available when the strobe is active (low) for the read operation. The data on the data line is latched into the MSP53C392 on the rising edge of the strobe signal for the write operation.
DATA 0–7	14,6,5,4 3,2,1,16	8-bit bidirectional data line
INIT	9	Reset signal. A low pulse to reset the chip. It can also be used to stop the MSP53C392 operation during speech synthesis. Following the rising edge of the $\overline{\text{INIT}}$ pulse, a delay of up to 5 ms will be required to permit the MSP53C392 to completely initialize its internal condition.

3.3 Master Microprocessor Interface Description

3.3.1 Method 1: Polling

Three control lines and eight I/O data lines are used in this interface. Data is written to the MSP53C392 device and the status can be read back. Two status bits ($\overline{\text{BUSY}}$ and EOS) can be read back by the master microprocessor to signal that the MSP53C392 is busy and signal the end-of-speech has been reached.

The interfacing diagram is shown in Figure 3–1:



NOTE A:	$\overline{\text{STROB}}$:	Active low strobe signal
	R/$\overline{\text{W}}$:	Read/write signal
	DATA 0–7:	8-bit data line
	$\overline{\text{BUSY}}$:	Active low busy signal form MSP53C392. A high signal indicates that the MSP53C392 is **not** busy and is ready to accept data.
	EOS:	End-of-speech data. A high signal indicates end-of-speech. Two bytes of dummy data written resets the EOS to low.
	$\overline{\text{INIT}}$:	Active low reset signal. The master microprocessor should issue a reset signal to MSP53C392 after power up to properly initialize the MSP53C392 device.

Read Operation

1) The master microprocessor sets R/$\overline{\text{W}}$ high to indicate a read operation.
2) The master microprocessor sets $\overline{\text{STROB}}$ to low and reads the state of $\overline{\text{BUSY}}$ and EOS signals.
3) The master microprocessor sets $\overline{\text{STROB}}$ high.

If the $\overline{\text{BUSY}}$ signal was high in step 2, the MSP53C392 is ***not*** busy and is ready to accept a write operation. If the $\overline{\text{BUSY}}$ signal was low in step 2, the MSP53C392 is ***not*** ready to accept a write operation and the read operation should be repeated until $\overline{\text{BUSY}}$ is asserted high.

If EOS was high in step 2, the synthesis process has reached the end of the speech data stream. In this case, the master microprocessor should stop trying to send data and reset the MSP53C392 to allow it to accept additional commands or synthesis data.

The frequency of the polling operation should be optimized to the data rate of the algorithm being used to synthesize speech. If the polling operation is too frequent, the MSP53C392 spends too much time servicing the polling operation and the quality of the synthetic speech may be affected. If the polling operation is too infrequency, the internal buffer may run out of data and the synthesis process can become corrupted. Normally, a polling frequency of four times the bit rates of the speech data provides optimal transfer characteristics.

Example:

For 6.2 kbps CELP the frequency of polling would be

$$\frac{6.2}{n} \times 4$$

n = the number of bits transfered at a time

Write Operation

1) The master processor should determine that the MSP53C392 is ready to accept data by reading the $\overline{\text{BUSY}}$ signal as described previously.
2) The master microprocessor clears R/$\overline{\text{W}}$ low to indicate a write operation.
3) The master microprocessor presents valid data to the eight data pins (DATA0 – DATA7).
4) The master microprocessor pulses the $\overline{\text{STROB}}$ signal low and then high to latch the data to the MSP53C392 input data latch.
5) The master microprocessor should do a read operation to determine that the MSP53C392 is ready to accept additional data before attempting to write more data.

If the EOS signal is asserted high during the read operation, the end-of-speech has been reached and a reset operation should be performed prior to sending new commands or speech data. The reset can be done in one of two ways: Pulsing the $\overline{\text{INIT}}$ pin low and then waiting for the MSP53C392 to re-initialize itself or by writing four dummy bytes as described in the following.

RESET Operation

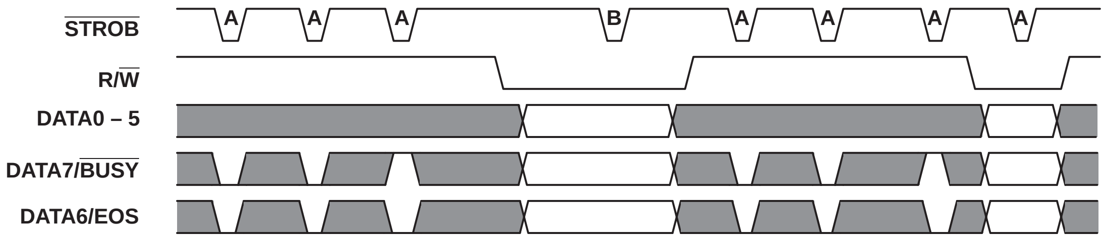
1) Perform a read operation to determine that both the EOS and $\overline{\text{BUSY}}$ signals are high.

2) If both the EOS and $\overline{\text{BUSY}}$ signals are high, write four bytes of dummy data to the MSP53C392 by repeating the write operation four times as described previously.

3.4 Master Microprocessor Interface Timing

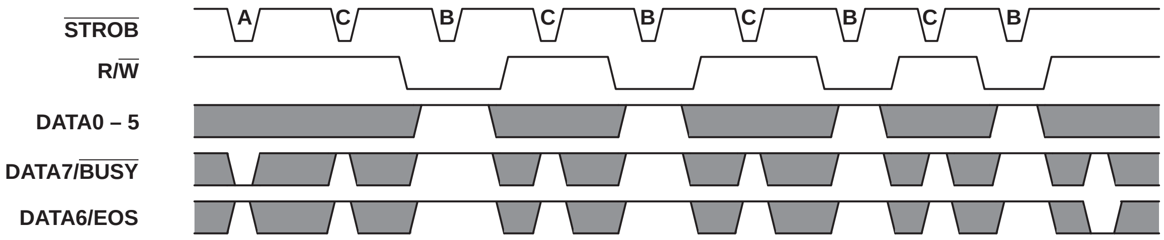
3.4.1 Timing Method 1: Polling

Data Transfer



NOTE A: State A: Polling the status by reading the BUSY and EOS
State B: Write operation

End-of-Speech



NOTE A: State A: EOS detected by reading DATA 6/EOS
State B: Dummy write. A 4-byte dummy write resets the EOS for the next transfer.
State C: Wait until the part is ready to accept dummy data (BUSY high).

3.5 MSP53C392 Device Initialization

For proper operation, the MSP53C392 device should be initialized by sending the following command sequence of bytes:

FF,FF,FF,FF,0A,01,00,FF,FF,FF,FF,FF

Following this command sequence, the normal command sequence options are available as described in Section 4.2 and onwards.

The function of this sequence is to properly initialize the synthesis engine by speaking a short selection of LPC prior to speaking selections using other synthesis algorithms.

This initialization needs to be performed:

1) After you apply power to the device, or

2) When you reset the part by toggling the INIT pin.

Chapter 4

MSP53C391 AND MSP53C392 Software Description

Topic **Page**

4.1 Software Overview

The MSP53C391 and the MSP53C392 are controlled using a formatted communication sequence that passes commands and data from the master microprocessor to the slave.

4.2 Command Sequence

There are two types of streams that can be sent to the slave.

- ❏ Data streams transmit speech synthesis data.
- ❏ Command streams control various features of the slave device such as volume, the state of the two expansion pins, and other special features.

Each stream consists of:

1) A command header whose purpose is to synchronize the data stream.
2) A command code that indicates which command should be executed or which synthesis algorithm is to be used to process the data stream.
3) The data stream or optional command parameters.
4) After the termination code in the data stream, four nibbles (MSP53C391) or four bytes (MSP53C392) of dummy data that resets the processor.

4.3 Command Header

The command header is used to synchronize the data stream between the master microprocessor and the MSP53C391 or MSP53C392. The command header is the same for both data streams and command streams, but the command header used for the MSP53C391 is different from the command header used for the MSP53C392 because of the different data bus widths.

The command header used for the MSP53C391 is a series of at least 5 nibbles with all bits set high followed by a 0x0, 0xA sequence. The complete command header sequence used for the MSP53C391 is therefore: 0xF, 0xF, 0xF, 0xF, 0xF, 0x0, 0xA.

The command header used for the MSP53C392 is a series of at least 5 bytes with all bits set high followed by a 0x0A sequence. The complete command header sequence used for the MSP53C392 is therefore: 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0FF, 0x0A.

4.4 Data Streams

To initiate speech, the master microprocessor transmits:

- ❑ The command header.
- ❑ The synthesis selection code. See Section 4.4.1, *Synthesis Selection Codes*, for valid selection codes.
- ❑ The synthesis data. This data must be matched to the command code that was sent. The synthesis data contains an imbedded code that identifies the end of the synthesis data. The synthesis process detects this imbedded code and responds by shutting down the synthesis process and toggling the EOS signal. See Appendix A for more information regarding the data preparation.

Table 4–1. Speech Initiation Data

Device	Command Header	Synthesis Selection Code	Speech Data
MSP53C391	F,F,F,F,F,0,A	XX	X,x,x,x…
MSP53C392	FF,FF,FF,FF,FF,0A	XX	X,x,x,x…

4.4.1 Synthesis Selection Codes

The synthesis selection code is a one-byte value that indicates the format and sampling rate of the synthetic speech data that follows it. The valid codes are shown in Table 4–2. The format of the data that follows ***must*** match the specified algorithm or the speech will not synthesize properly.

Table 4–2. Synthesis Selection Codes

Freq.	**LPC**		**MELP**	**CELP (Ver. 3.4) (kbps)**						**8-Bit PCM**	**FM II**[‡] **(Ver. 2.08)**
	5220	**D6**	**Ver. 4.1**	**4.2**	**4.8**	**5.8**[†]	**6.2**	**8.6**	**10.7**		
8 kHz	01	02	03	04	05	06	07	08	09	0D	0E
10 kHz	11	12	13	14	15	16	17	18	19	1D	1E

† CELP 5.8 kbps is **not** a standard CELP rate. It can only be encoded and edited by selecting the CELP parameter manually in SDS3000. The parameters for CELP 5.8 kbps are:
‡ Since the pitch table is based on a 10-kHz sampling rate, it is recommended the 10-kHz sampling rate option be used.

- ■ Subframe Size = 60
- ■ Subframes per Frame = 4
- ■ Pulses per Subframe = 4

Because the CELP 5.8 kbps is not a standard CELP rate, it is **not recommended** for less experienced users. The standard CELP rates (4.2 kbps, 4.8 kbps, 6.2 kbps, 8.6 kbps, and 10.7 kbps) should be used instead.

4.5 Command Sequences

A command sequence transmits controlling commands that instruct the MSP53C391 or MSP53C392 to modify its function in some way. Available commands are:

❑ Set the two general-purpose output pins either high or low (MSP53C391 only)

❑ Place the MSP53C391 or MSP53C392 into a low-power sleep state

❑ Adjust the output volume

❑ Initiate a test mode in which a 4-kHz or 5-kHz square wave is generated at the two general-purpose output pins (MSP53C391) or selected data pins (MSP53C392)

❑ Read the software version programmed into the MSP53C391 or MSP53C392.

To send a command sequence, the master microprocessor transmits:

❑ The command header

❑ A valid command code. See Section 4.5.1, *Command Codes*, for valid command codes.

❑ Optional parameters

Table 4–3. Command Sequence

Device	**Command Header**	**Command Code**	**Optional Parameters**
MSP53C391	F,F,F,F,F,0,A		X,x,x,x…
MSP53C392	FF,FF,FF,FF,FF,0A		X,x,x,x…

4.5.1 Command Codes

The valid command codes are shown in Table 4–4. Due to the absence of the OUT1 and OUT2 pins on the MSP53C392, not all of the functions are available on the MSP53C392.

Table 4–4. Command Codes

Command Codes	MSP53C391	MSP53C392
0x20	Program OUT1 and OUT2 both low	N/A
0x21	Program OUT1 high and OUT2 low	N/A
0x22	Program OUT1 low and OUT2 high	N/A
0x23	Program OUT1 and OUT2 both high	N/A
0x2E	Scale output volume	Scale output volume
0x2F	Place MSP53C391 into a low-power sleep state	Place MSP53C392 into a low-power sleep state
0xD1	Request version information	Request version information
0xE65D	Produce 5-kHz signal on the OUT2 pin	Produce 5-kHz signal on the DATA3 pin
0xE65C	Produce 4-kHz signal on the OUT1 pin	Produce 4-kHz signal on the DATA2 pin
0xE96D	Echo mode	N/A

4.5.2 Pin Expansion

The OUT1 and OUT2 pins are available on the MSP53C391 for pin expansion. To program these pins to the desired state, transmit one of the command codes as shown in Table 4–5:

Table 4–5. Pin Expansion Command Codes

Command Code	State of OUT1	State of OUT2
0x20	Low	Low
0x21	High	Low
0x22	Low	High
0x23	High	High

4.5.3 Volume Control

Transmit a command of 0x2E to scale the output volume of the synthesized data. At power up the default volume is set to the maximum value (0x80). This command can be transmitted to reduce the volume from this maximum. The minimum permitted volume is 0x20.

Table 4–6. Volume Control Commands

Device	Command Header	Command Code	Volume Control Code
MSP53C391	F,F,F,F,F,0,A	0x2E	xx
MSP53C392	FF,FF,FF,FF,FF,0A	0x2E	xx

Valid volume control codes range from a minimum of 0x20 to a maximum of 0x80 as shown in Table 4–7:

Table 4–7. Volume Control Code Ranges

Volume Control Code	Result
0x20	Minimum volume setting
0x30	Intermediate volume setting
0x40	Intermediate volume setting
0x60	Intermediate volume setting
0x80	Maximum volume (power up default)

4.5.4 Low-Power Sleep State

Sending a command code of 0x2F places the MSP53C391 or MSP53C392 into a low-power sleep state. The MSP53C391 or MSP53C392 can be restarted by resetting the device (pulsing the $\overline{\text{INIT}}$ pin low) and waiting 5 ms for the device to complete initialization.

4.5.5 Request Software Version

The master processor can request the version number of the software programmed into the MSP53C391 or MSP53C392 by sending the command code 0xD1. Following transmission of this command, the master processor should poll the BUSY bit to verify that it is high (indicating that the data is available to be read).

The version information is then available on DATA0, DATA1, and DATA2 on the MSP53C391. It can be read from these pins using the READ protocol described in Chapter 2. The version number read from the MSP53C391 is 1.

On the MSP53C392, the version information is then available on DATA4, DATA5, and DATA6. It can be read from these pins using the READ protocol described in Chapter 3. The version number read from the MSP53C392 is 2.

4.5.6 Generate Test Signal

The MSP53C391 has three test modes.

- Generate 4-kHz signal on OUT1
- Generate 5-kHz signal on OUT2
- Echo input data

  The MSP53C392 has two test modes. Echo input data is not available on the MSP53C392.

- Generate 4-kHz signal on DATA2
- Generate 5-kHz signal on DATA3

Sending a command code of 0xE65D to the MSP53C391 generates a 5-kHz signal on the OUT2 pin. This can be used to test the accuracy of the internal oscillator when it is programmed to 19.2 MHz. The only way to exit this test mode is to pulse the $\overline{\text{INIT}}$ pin low.

Sending a command code of 0xE65C to the MSP53C391 generates a 4-kHz signal on the OUT1 pin. This can be used to test the accuracy of the internal oscillator when it is programmed to 15.36 MHz. The only way to exit this test mode is to pulse the $\overline{\text{INIT}}$ pin low.

Sending a command code of 0xE65D to the MSP53C392 generates a 5-kHz signal on the DATA3 pin. Once the command code has been transmitted to the MSP53C392, the $\overline{\text{STROB}}$ must be set low and the R/$\overline{\text{W}}$ must be set high to enable the generation of the 5-kHz signal. This can be used to test the accuracy of the internal oscillator when it is programmed to 19.2 MHz. The only way to exit this test mode is to pulse the $\overline{\text{INIT}}$ pin low.

Sending a command code of 0xE65C to the MSP53C392 generates a 4-kHz signal on the DATA2 pin. Once the command code has been transmitted to the MSP53C392, the $\overline{\text{STROB}}$ must be set low and the R/$\overline{\text{W}}$ must be set high to enable the generation of the 4-kHz signal. This can be used to test the accuracy of the internal oscillator when it is programmed to 15.36 MHz. The only way to exit this test mode is to pulse the $\overline{\text{INIT}}$ pin low.

Sending a command code of 0xE96D to the MSP53C391 causes it to enter a special test mode in which the input data latched into the device is echoed out to the OUT2, OUT1, EOS, and $\overline{\text{IRQ}}$ pins. This mode is for debugging the communications interface between the master microprocessor and the MSP53C391. While in this mode, the MSP53C391 will not speak the voice data. The only way to exit this mode is to pulse the $\overline{\text{INIT}}$ pin low. This mode is **not** available on the MSP53C392.

Appendix A

Editing Tools and Data Preparation

Topic **Page**

A.1 Editing Tools

TI provides several tools to support speech editing. The WINSDS is a tool for LPC editing and the SDS3000 is a tool for MELP editing and CELP and MELP encoding.

A.1.1 WINSDS

WINSDS (Windows™ interface speech development station) is a powerful tool to produce high-quality LPC (linear predictive coding) speech and sound.

The Windows-based WINSDS is a successor to the SDS5000 speech development station designed to produce synthesized vocabulary. The WINSDS system requires a personal computer running Windows 95/98™ and requires at least one available ISA slot.

A.1.2 SDS3000

SDS3000 is an integrated tool that accepts input sound files (sampled at either 8-kHz or 10-kHz) in either a .WAV format or a 16-bit raw binary format and converts the files into the CELP or MELP data format. It requires a personal computer running Windows 95/98 with at least one available ISA slot. SDS300 does MELP and CELP encoding as well as MELP editing.

Windows, Windows 95, and Windows 98 are a trademarks Microsoft Corporation.

A.2 Data Preparation

The MSP53C391 and MSP53C392 slave synthesizers support several algorithms for speech synthesizing. The speech data sent to the slave device must match the format defined by TI or generated from a TI tool (SDS3000 for MELP or CELP and WINSDS for LPC). The data preparation for different algorithms for MSP53C391 and MSP53C392 is discussed in the following paragraphs.

A.2.1 LPC

LPC is processed and editing using the WINSDS station. Please refer to the *WINSDS User's Guide* (literature number: SPSU010) for details.

A.2.2 MELP and CELP

The SDS3000 is used to convert an input audio data file into the MELP or CELP data formats. The input data file can be either signed binary or .WAV format file. The audio data should be sampled at either 8 kHz or 10 kHz and should have a precision of 16 bits.

The sound files should start and stop at a level close to zero, otherwise errors may result. The volume of the sound files should also be adjusted to a level with a peak-to-peak around +0.5 to –0.5 (assume full scale is +1 to –1). When the sound file is too loud after the encoding, clipping will result. Additionally, if the wave file is sampled at a higher rate (CD quality sound file 44.1-kHz sampling or DAT in 48-kHz), resampling to 8 kHz or 10 kHz is necessary for the conversion. Filtering and renormalizing may be necessary during down-sampling to reduce aliasing and noise.

There are a number of software programs available to do this resampling function. Two commonly available examples are GoldWave and Cool Edit.

GoldWave is a shareware program that provides editing function on sound files in a Windows environment. This software can be downloaded from the web: **http://www.goldwave.com**

By using the GoldWave sound editor, the sound file can be cut to eliminate the leading and the following silence. The volume can also be adjusted.

To resample the original wave/sound file, we need to cut all the high-frequency portion first to eliminate the error in resampling. All frequencies above one half of the final sampling rate should be removed from the sound file. For example, before converting the sampling rate to 8 kHz, the data should be filtered to remove all frequency components above 4 kHz.

GoldWave is a product of Goldwave Corporation
Cool Edit is a product of Syntrillium Software Corporation

The procedures for a low pass filter and resampling with GoldWave are listed in the following:

1) Open the data file (44.1-kHz 16-bit mono signed).
2) In EFFECT MENU choose FILTER then LOW/HIGH PASS. Use the low pass filer to cut the signal above 4 kHz for 8-kHz sampling and 5 kHz for 10-kHz sampling.
3) In EFFECT MENU Choose RESAMPLE and then choose 8k/10k to resample the file.
4) Save the file in 16-bit monaural signed data format. This file can then be used for the MELP/CLEP encoding program SDS3000.

For the operation of SDS3000, please refer to the related documents.

A.2.3 PCM

The MSP53C391 and MSP53C392 can accept PCM data. The PCM data should be sampled at either 8 kHz or 10 kHz and should be signed 8-bit data. The data should be scaled so that the peak signal is close to the 8-bit maximum. As an example, to obtain a suitable PCM file from GOLDWAVE:

1) Open a .WAV file. The data in the .WAV file should be sampled at either 8 kHz or 10 kHz. The file should contain monaural data.
2) If the data in the file is not sampled at one of these two frequencies, it should be resampled to one of these two frequencies. First low pass filter the data to 4 kHz or 5 kHz to avoid sound degradation due to aliasing with the command EFFECTS – FILTER – LOW/HIGH PASS, then resample the data to the new sampling rate using the command EFFECTS –RESAMPLE.
3) Maximize the volume with the command EFFECTS – VOLUME – MAXIMIZE
4) Save the resulting file using the command FILE – SAVE AS, In the *Save as type:* field, select RAW. In the *File Attributes* field, select *8–bit, mono, signed*.
5) Since the process of converting a sound file to 8 bit can introduce a quantization error, it is recommended that resulting file be processed to reduce the noise. In EFFECT MENU choose NOISE REDUCTION. In PRESET SHAPES use the Hiss shape to reduce the noise.
6) It is necessary to append a termination code to the end of the PCM data to signal the end of the file to the MSP53C391 or MSP53C392. The proper end code is the two byte sequence: 0x7F, 0x80.

A.2.4 FM

Music can be coded manually or can be converted from MIDI (musical instrument digital interface) files. For manual coding, please refer to Appendix B for the data format of FM synthesis. If the song is composed in MIDI format (.mid), it can be converted to FM by a DOS executable routine MD2FM.EXE. There are several limitations on the MIDI files, which the MD2FM program processes.

1) The MSP53C391 and MSP53C392 support a maximum of two channels of FM synthesis music. The MD2FM can convert only one track or channel at a time. Two passes through the program are required to convert the two channels into two separate output file. The two files are combined later into a single file using the FM2MERGE program.

2) The timebase of the file should be 48.

3) The MD2FM program does not understand the instrument definition of the MIDI file. The instruments will need to be added to the output file in a separate step.

As an example, assume that a MIDI file named midi_t1.mid contains two tracks and each track contains a single channel. Execute the MD2FM program twice:

```
>MD2FM midi_t1 midi_t11 –c1 –t1
>MD2FM midi_t1 midi_t12 –c1 –t2
```

The first pass extracts channel 1 of track 1 from the file and stores it into the file midi_t11.inc. The second pass extracts channel 1 of track 2 from the file and stores it into the file midi_t12.inc.

The two files are combined into a single file using the program FM2MERGE as follows:

```
>FM2MERGE midi_t11.inc midi_t12.inc midi_t1.inc
```

The FM2MERGE program combines the midi_t11.inc and the midi_t12.inc files into a single output file called midi_t1.inc.

Appendix B

FM Synthesis

Topic **Page**

B.1 FM Synthesis Overview

FM synthesis is a technique for creating harmonically rich musical tones in a relatively simple manner. Generally speaking, the tones generated do not closely correspond to the tonal texture of conventional instruments; but can be used to generate interesting and pleasant music.

The MSP53C391 and MSP53C392 can generate two channels of FM synthesis. This means that a maximum of two notes can be played simultaneously. This appendix describes the command formats used to describe the music.

B.2 FM Synthesis Format and Commands

The song to be played is coded into a file in a specified format. This file is then assembled and the binary result is transmitted by the master microprocessor to the MSP53C391 and MSP53C392 to play the music.

The file contains a series of BYTE or DATA statements as described in the following to specify the notes, instruments and other details of the music.

Each command is of the general form:

```
BYTE   command, parameters,…
```

Where *command* indicates the action to be taken, followed by one or more modifying parameters. For example, to transpose a section of a song up by a semitone, the following command would be written:

```
BYTE    RTRNS,1
```

In this example, RTRNS is the command and 1 is the modifying parameter.

As another example, to play a note, the following command might be written:

```
BYTE   C1,n4,n4,127
```

This example commands the synthesizer to play a ‘C’ note for a quarter note duration at the maximum volume.

The formats and parameters for the different commands are described in the following sections.

The various commands are defined in the file FMEQUM2.INC. The contents of this file should either be copied into the start of the FM file or a copy command should be inserted at the start of the FM file to insert the values; for example:

```
COPY   ‘FMEQUM2.INC’
```

B.2.1 Musical Notes

Musical notes are defined as:

```
BYTE     Notevalue, TimeValue,Duration,Velocity
```

Where:

Notevalue defines the pitch of the musical note. The valid values for Notevalue are defined in Appendix C. In general, they range from a minimum of C1 to a maximum of C6. Within each octave, the sequence is: C, Cs or Db, D, Ds or Eb, E, F, Fs or Gb, G, Gs or Ab, A, As or Bb, B. For example, the C sharp in the first octave would be written Cs1. This is the same tone as Db1.

TimeValue defines the duration of the note that is played. The valid values for TimeValue are defined in Appendix C. The more common values are: N8 defines an eighth note. N4 defines a quarter note. N2 defines a half note. N1 defines a whole note.

Duration defines the sum of the TimeValue and any following rest. For example, if a quarter note is followed by a quarter note rest, then it would be coded with a TimeValue=n4 and a Duration=n2.

Velocity is the relative volume of the note. Values range from 0 to 127.

B.2.2 Tempo Control

The tempo of the music is defined as:

```
BYTE     TEMPO,BPM,TimeSig,EnvelopeLen
```

Where:

TEMPO is the command that indicates a tempo change is being defined.

BPM is the new tempo. The valid values for BPM are defined in Appendix C. An example is bpm62, which indicates 62 beats per minute.

TimeSig defines the time signature of the songs. TS44 sets the time signature to 4/4 time.

EnvelopeLen defines the length of the note envelope. ENVOK sets the envelope length to normal (i.e., lasting for a whole note).

Note:

The tempo should be set in the channel one stream only. The TEMPOSYNC command should be placed in the same position in the channel two stream.

B.2.3 Tempo Synchronization

The tempo of the two channels needs to be the same. If it changes, it needs to change at the same point in the music for both channels. This is accomplished by placing the tempo change information in the channel one data stream (using the TEMPO command) and by placing a synchronizing placeholder in the channel 2 data stream to ensure that the tempo change happens at the same point in the music for both channels. This placeholder is the TEMPOSYNC command. The TEMPOSYNC channel should be placed in the channel two data stream only.

```
BYTE      TEMPOSYNC
```

B.2.4 LOADTIMBRE Command

The LOADTIMBRE is used to change the tonal quality of one of the two channels. It is followed by a 21 byte stream of data that defines the new instrument.

```
BYTE      LOADTIMBRE,XX,XX.........,XX
```

Where *xx* denotes a series of 21 bytes of data that defines the new instrument sound. The data definitions are as follows:

The first three bytes define the frequencies of the modulator and carrier sinusoids. The first byte defines the carrier frequency.

The next two bytes define the initial amplitude of the carrier and the modulator sinusoids. The valid values range from 0 to 127.

The remaining 16 bytes represent the change in values that define the envelope of the carrier and multiplier during 8 uniform time slices. Each pair of values increments or decrements the carrier and modulator amplitude during the next time slice. The cumulative value of both the carrier and modulator amplitude is limited to a range of 0 to 127. The carrier amplitude should taper to zero at the end of the envelope.

For example:

```
BYTE  LOADTIMBRE     ;Load new instrument patch
BYTE  X2             ;Carrier   Fc = 2Fo
BYTE  X2             ;Modulator Fm = 2Fo
BYTE  16*4           ;Modulation Index Scaler
BYTE  2,1            ;CarAmp, FmAmp  Initial Values
BYTE  124,46         ;CarAmp=126, FmAmp=47
BYTE  –10,80         ;CarAmp=116, FmAmp=127
BYTE  –9,–8          ;CarAmp=107, FmAmp=119
BYTE  –8,–7          ;CarAmp=99,  FmAmp=112
BYTE  –7,–6          ;CarAmp=92,  FmAmp=106
BYTE  –6,–5          ;CarAmp=86,  FmAmp=101
BYTE  –5,–4          ;CarAmp=81,  FmAmp=97
BYTE  –81,–3         ;CarAmp=0,   FmAmp=94
```

B.2.5 Transposition

Two commands are available for transposing the music (i.e., uniformly shifting the notes to higher or lower frequencies). The ATRNS command adds a specific amount to the note value. The RTRNS command adds a relative amount to the note value.

BYTE ATRNS, NUM

Shifts the music NUM semitones from the value as written, for example:

BYTE ATRNS,12

Shifts the music one octave above the music as written.

The RTRNS command is used to shift the music by a cumulative amount, for example in the following sequence:

```
BYTE  RTRNS,12  ;This will shift the music up by one octave
BYTE  RTRNS,12  ;This will shift the music up a second octave
```

B.2.6 DETUNE

The DETUNE command shifts the frequency of notes on channel two with respect to the frequency of the notes on channel 1.

```
BYTE  DETUNE,5    ;Add 5 to the frequency of channel 5
```

B.2.7 Adjust Output Volume

The FADER command is used to scale the volume of the notes played.

BYTE FADER, InitialFaderValue,FaderInc

Where:

FADER is the signal that the command is to adjust the output volume

InitialFaderValue is the new sound volume

FaderInc allows a gradual transition to the new volume. It is a signed two-byte value that specifies the incremental amount to change the volume during each interval.

For example:

```
BYTE   FADER,f100p     ;Set volume to 100
DATA   NOFADER      ;Change is abrupt
```

B.2.8 Modulation Index Adjustment

It is frequently desirable to incrementally change the texture of the sound quality in the song. This can be done by changing the modulation index to get a more or less brighter tonal quality. This can be done by using the following commands:

```
BYTE   MIX1     ;Set the modulation index scale to 1
BYTE   MIX2     ; Set the modulation index scale to 2
BYTE   MIX3     ; Set the modulation index scale to 3
BYTE   MIX4     ; Set the modulation index scale to 4
BYTE   MIX5     ; Set the modulation index scale to 5
BYTE   MIX6     ; Set the modulation index scale to 6
BYTE   MIX7     ; Set the modulation index scale to 7
BYTE   MIX8     ; Set the modulation index scale to 8
BYTE   MIXUP    ; Increment the modulation index scale
BYTE   MIXDN    ;Decrement the modulation index scale
```

B.2.9 End of Song

The STOPSONG is used to signal the end of the song.

```
BYTE   STOPSONG    ;Signal the end of song
```

B.2.10 Command Summary

Table B–1 summarizes the several valid commands.

Table B–1. Command Summary

Command and Format	Description
Music Notes: Format: Note,TimeValue,Duration,Velocity Example: C1, n4, n4, 127	Note: Is the music note that can range form C0 to C6 TimeValue: Total length of the note. n4 is 1/4 note. Duration: Length of tone generate Velocity: Note volume from 0 to 127
Tempo control the speed of music: Format: TEMPO,BPM,TimeSig,EnvelopeLen Example: TEMPO,BPM116,TS44,ENVOK	TEMPO: Tempo command. Set the song tempo in channel 1 ONLY. BPM: Beats per minutes. TimeSig: Beats per measure. TS44 sets the time signature to 4/4 time. EnvelopLen: Envelop time. ENVOK sets the envelope length to normal.
Tempo control for channel 2: Format: TEMPOSYNC Example: TEMPOSYNC	TEMPOSYNC: Use in channel 2 **only**. It must be placed at the same bar # as the channel 1 TEMPO command. This is to ensure that the Tempo change is synchronous, with both channels changing at the same time.
Load Timbre into each channel: Format: LOADTIMBRE,XX,XX.........,XX Example: LOADTIMBRE, 21 bytes Parameters	LOADTIMBRE: Load new timbre (instrument) parameters. Parameters: Contains 21 bytes that define a musical instrument.
Transpose: Format: ATRNS, NUM Example: ATRNS,–12 (–12 = Transpose Down an Octave)	ATRANS: Transpose command. NUM: Set the channel's transpose to a signed offset.
Transpose: Format: RTRNS,NUM Example: RTRNS,7 (Add 7 Semitones to the channel's transpose offset)	RTRNS: Transpose command. NUM: Add a signed offset to the channel's transpose value.

Table B–1. Command Summary (Continued)

Command and Format	Description
Detune: Format: DETUNE,NUM Example: DETUNE,4 (add 4 to channel 2's Sine table index)	DETUNE: Allows detuning channel 2 NUM: A signed offset to the channel 2's frequency value.
Fade control: Format: FADER,InitialFaderValue,FaderInc Example: FADER,f100p, –32 (word)	FADER: Fader command InitalFaderValue: Set initial fader value from 0 to 63. F100p is defined in FmequM2.inc FadInc: Fader increment, which is a 16-bit value. Calculate as follows: $\frac{(\text{End Fader Value} - \text{Start Fader Value}) \times 16}{\#\ \text{of Events}}$
Mix control: Format: MixLevel Example: MIX8	MixLevel: Set the modulation index value by table lookup form MIX0 to MIX15
Mix control: Format: MIXUP Example: MIXUP	MIXUP: Increment the current modulation index as set by MIXn.
Mix control Format: MIXDN Example: MIXDN	MIXDN: Decrement the current modulation index as set by MIXn.
End of the song: Format: StopSong Example: StopSong	StopSong: Stop playing the song.

B.3 FM Synthesis Data Structure

As there are two channels data that are passed to the MSP53C391 or MSP53C392 through a single data path; the note information needs to be interleaved to provide the correct sequencing.

1) The channel one setup information and the first note of channel one are loaded first.

2) The channel 2 setup information and the first note of channel two will be loaded following the first note of channel 1.

3) Then the note duration of channel 1 and channel 2 are compared. If the total duration of the notes of channel 1 is less than or equal to channel 2, channel 1 data is loaded. If the total duration of the notes of channel 1 is larger than channel 2, channel 2 data is loaded. In this way, the channel data are interleaved according to the accumulative duration of the notes. The following example is in MSP50C3x syntax:

```
*
* Example of FM synthesis
*
* define the Channel 1 and load first note
* Channel 1
; BYTE means define a byte data in C3x syntax
; ”;” and ”*” is comments
    BYTE   TEMPO,BPM122,–48,ENVOK ; set the Tempo of channel 1
    BYTE   ATRNS,0                ; define the Transpose
    BYTE   MIX3                   ; set Mix Level
                                  ; DATA means define a word data
    BYTE   FADER,f100p            ; set No Fader
    DATA   NOFADER
                                  ; load Piano tone 1 in channel 1
    BYTE   LOADTIMBRE             ; Parameters of 21 bytes follows
    byte   X1
    byte   X1
    byte   38*4
    byte   126,108
    byte   –13,006
    byte   –25,013
    byte   –13,–19
```

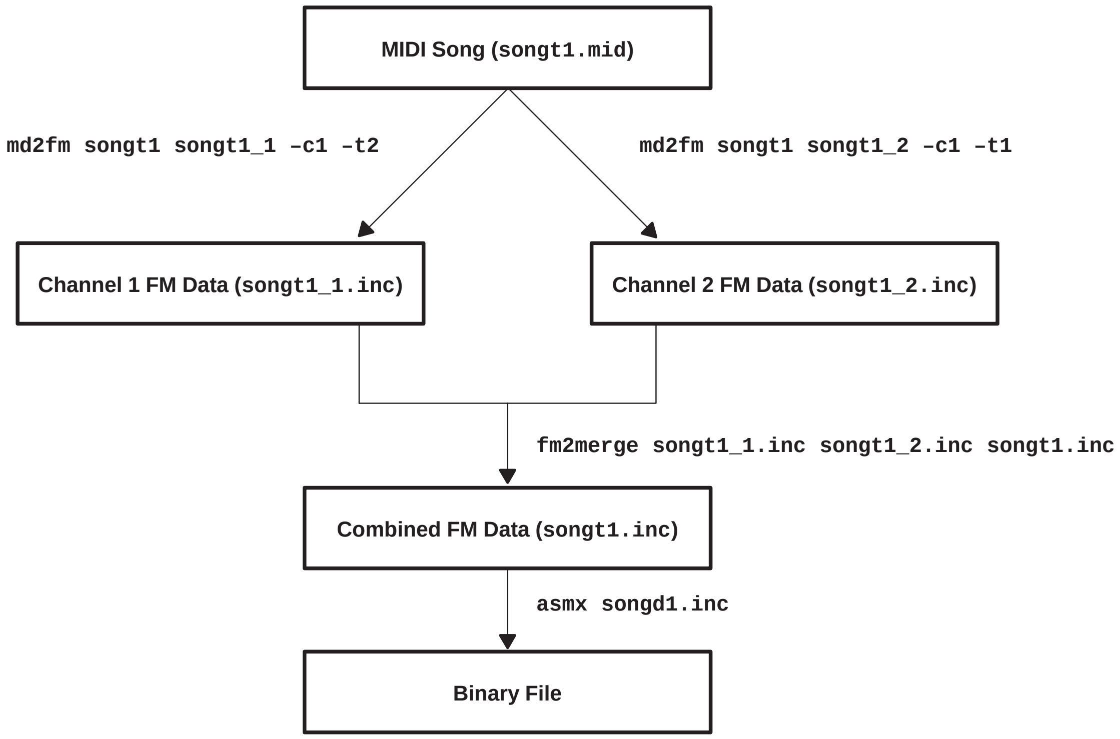
```
   byte   –25,–19
   byte   000,–06
   byte   000,–25
   byte   –38,–32
   byte   –10,–23
   BYTE   C4,12,12,127           ; load first note of channel 1
* define Channel 2 and loading the first note
* Channel 2
   BYTE   TEMPOSYNC              ; sync the Tempo of channel 2 with 1
   BYTE   ATRNS,0                ; set Transpose of channel 2
   BYTE   MIX3                   ; set Mix Level of channel 2
   BYTE   FADER,f100p            ; set No Fader of channel 2
   DATA   NOFADER
                                 ; load timbre Flute tone 1 in channel 2
   byte   LOADTIMBRE             ; Parameters of 21 bytes follows
   byte   X2
   byte   X2
   byte   16*4
   byte   2,1
   byte   124,46
   byte   –10,80
   byte   –9,–8
   byte   –8,–7
   byte   –7,–6
   byte   –6,–5
   byte   –5,–4
   byte   –81,–3
   BYTE   A4,12,12,64            ; first note of channel 2
* Continue to play the rest data of channel 1&2
                       ; interleaved the channel 1 and channel 2 data
                       ; according to the accumulative duration in each channel
* Channel 1
   BYTE   E4,12,12,127
* Channel 2
   BYTE   A4,12,12,64
* Channel 1
```

```
   BYTE   G4,12,12,127
* Channel 2
   BYTE   A4,12,12,64
* Channel 1
   BYTE   E4,12,12,127
* Channel 2
   BYTE   A4,12,12,64
* Channel 1
   BYTE   C5,12,12,127
* Channel 2
   BYTE   REST,48,2,OFF
* Channel 1
   BYTE   E5,12,12,127        ; the duration of channel 2 is still larger than 1
   BYTE   C5,12,12,127        ; by loading one note only
   BYTE   G4,12,12,127        ; thus, continue to load 1 until
   BYTE   REST,12,2,OFF       ; the duration of 1 is larger than 2
* Channel 2
   BYTE   REST,48,2,OFF
* Channel 1
   BYTE   G4,12,12,127
   BYTE   E4,24,12,127
   BYTE   E4,48,48,127
* Channel 2
   BYTE   REST,48,2,OFF
* Channel 1
   BYTE   REST,48,2,OFF
* Channel 2
   BYTE   REST,48,2,OFF
* Channel 1
   BYTE   StopSong
* Channel 2
   BYTE   StopSong
```

B.4 Data Preparation of FM Synthesis

According to the discussion on FM data format and structure, a song can be coded following the predefined command and formats. Alternatively, software utilities are available for converting a song from MIDI (musical instrument digital interface) formatted files to a format accepted by MSP53C391 AND MSP53C392. There are two utilities, MD2FM.exe and FM2MERGE.exe, for the conversion. The process for the conversion is shown in Figure B–1:

Figure B–1. FM Conversion Process


MIDI Song (songt1.mid)
md2fm songt1 songt1_1 –c1 –t2
md2fm songt1 songt1_2 –c1 –t1
Channel 1 FM Data (songt1_1.inc)
Channel 2 FM Data (songt1_2.inc)
fm2merge songt1_1.inc songt1_2.inc songt1.inc
Combined FM Data (songt1.inc)
asmx songd1.inc
Binary File

By using the MD2FM, the channel 1 and channel 2 data of a MIDI file (.mid) can be extracted and converted to FM format (.inc). Then, the two FM files need to be combined according to the duration of each note in channel 1 and 2 by the utility FM2MERGE. The combined file can then be sent to the master device and passed on to the slave for FM synthesis.

Once MD2FM has been used to merge the files, the merged file can be converted to a binary file using the ASMX or ASM10 assembly program.

B.4.1 MD2FM Software

MD2FM converts a MIDI format file to a FM data accepted by MSP50C391/2. With this routine, users can compose or translate music base on the MIDI format. This routine runs under the DOS environment and the syntax is as follows:

```
   md2fm songt1 songt1_1 –c1 –t1
input  : songt1.mid (MIDI format)
output : songt1_1.inc (FM format)
–c1: channel #1 to be decoded†
–t1: track #1 to be decoded†
```

† Specify the number of tracks and channels to be extracted and converted. Only one channel within one track can be converted on the MIDI file at any one time.

Assuming that the songt1.mid contains two tracks and each has one channel music data, the FM data can be extracted as follows:

```
   md2fm songt1 songt1_1 –c1 –t1
```

and

```
   md2fm songt1 songt1_2 –c1 –t2
```

The output file songt1_1.inc is the track 1 data and the songt1_2.inc is the track 2 data in FM format.

Since MD2FM does not support all the features of MIDI, the following must be noted:

1) Due to the limitation of conversion program, the TIMEBASE of the .mid file must be 48 and there must not be any TEMPO/METER changes after the initial settings. Also, it does not recognize channel pressure, control.

2) This routine creates a .inc file for a **single channel** of a **single track only** from a simple .mid file. To use this polyphonically, separate the MIDI file's chords into separate channels/tracks and run the md2fm program once per each channel/track to generate individual .inc files. Thus, it will need to separate the two overlapping notes into different channels/tracks and generate two .inc files by the converter. Then, a maximum of two tones can be generated simultaneously.

3) Since two files are generated for a song with two channels/tracks. The TEMPO command exists on each file, which is not valid for channel 2. The TEMPO command, which defines the tempo of the song, only applies to channel 1 and the TEMPOSYNC command should be used on channel 2. Thus, the TEMPO command must be modified to TEMPOSYNC for the file intended for channel 2.

Modify from:

```
    BYTE   TEMPO,BPM150,TS44,ENVOK
```

to:

```
    BYTE   TEMPOSYNC
```

on the channel 2 file.

4) For music with only one channel, a data stream with the following statement,

```
    BYTE   StopSong
```

can be used for the channel 2 and the single channel music can be placed on channel 1 for the synthesizing.

B.4.2 FM2MERGE Software

FM2MERGE is a routine run on the DOS environment. The function of the program is to merge two FM data stream into one data file for the MSP53C391 and MSP53C392 slave synthesis. The two data streams are combined according to the accumulative duration of the notes on channel 1 and 2. Please refer to the section B.2 for more information on the data structure of FM slave synthesis. The following is the syntax of the fm2merge:

```
fm2merge [input file1] [input file2] [output file]
input1 : channel 1 data file (songt1_1.inc)
input2 : channel 2 data file (songt1_2.inc)
output : output file (songt1.inc)
```

Assuming that there are two files, songt1_1.inc and songt1_2.inc, which is the song data for channel 1 and channel 2 in FM format. The combined file can be generated with the following command line:

```
fm2merge songt1_1.inc songt1_2.inc songt1.inc
```

The file songt1.inc can then be used for the slave FM synthesis.

The following points should be noted when using the FM2MERGE:

1) The instruments of FM II may not be compatible with the instruments selected in the MIDI file. It is also due to capability of downloading instruments in the slave device, it is necessary to replace the instrument define statement after the merge process from:

```
byte  PatchMT6i ;Use instrument Metallic tone 6i
```

to,

```
*PatchMT6i: Metallic tone 6i, hard metallic sound 1
byte  LOADTIMBRE
byte  X1
byte  X3
byte  28*4
byte  127,120
byte  –24,7
byte  –12,–20
byte  –6,–10
byte  –3,–5
byte  –2,–3
byte  –1,–1
byte  –12,–6
byte  –67,–12
```

which is the actual parameter for this instrument. All the PatchXXXX statements must be replaced with these parameters for the slave program to run properly. All the instruments available are listed in the file fm2intr1.inc. The instruments that match the applications can be selected.

2) For music with only one channel, a file with the following statement,

```
BYTE  StopSong
```

can be created for the channel 2 file. Then, go through the same process to create the FM II data for slave.

B.4.3 Assembler

Once the FM2MERGE program has been used to merge the two channels into one file, the file can be converted to a binary file by either the ASMX program or the ASM10 program.

```
ASMX  [input files]  [output file]
ASM10 [input files]  [output file]
```

Appendix C

Listing of FMequM2.inc

C.1 Listing of FMequM2.inc

FMequM2.inc contains all the FM command and note definitions that are used for preparation of FM data. The listings are shown in the following:

```
*                 FMEQU.INC
*                 Version     2.08

*     RAM Definitions
*     Constant Definitions

*                       DC,BA98:7654,3210
*                       --,----:----,----
BIT0        equ   #0001 ;00,0000:0000,0001
BIT1        equ   #0002 ;00,0000:0000,0010
BIT2        equ   #0004 ;00,0000:0000,0100
BIT3        equ   #0008 ;00,0000:0000,1000
BIT4        equ   #0010 ;00,0000:0001,0000
BIT5        equ   #0020 ;00,0000:0010,0000
BIT6        equ   #0040 ;00,0000:0100,0000
BIT7        equ   #0080 ;00,0000:1000,0000
BIT8        equ   #0100 ;00,0001:0000,0000
BIT9        equ   #0200 ;00,0010:0000,0000
BIT10       equ   #0400 ;00,0100:0000,0000
BIT11       equ   #0800 ;00,1000:0000,0000
BIT12       equ   #1000 ;01,0000:0000,0000
BIT13       equ   #2000 ;10,0000:0000,0000

NUMSONGS    equ   2     ;# of songs in list.

*     Initialization defaults
*     default gain value
MAXGAIN           equ   24    ;default value, also MAX. do not exceed!
*     default Master Modulation Index Scale values
DEFSCLMIX   equ   96    ;like a tone control...

*     Song interval delay values
ONESEC      equ   1*10
TWOSECS     equ   2*10
THREESECS   equ   3*10
FOURSECS    equ   4*10
FIVESECS    equ   5*10
TENSECS     equ   10*10

*     FM channel Automated Fader calculations
*     Coded as:
*     BYTE  FADER, CurrFader, ((DestFader–CurrFader) * 16) / #Events
*     Init Fader with Start Gain * 16 = 384
*     So, when each new event comes along, Add FaderInc to Fader
*     (EXTSG ON) and Update Fader.
*     When calculating Loudness, use Fader / 16 * Current Signal.

*     My standard fader values
f100p       equ   63
f94p        equ   60
f87p        equ   56
f75p        equ   48
f62p        equ   40
f50p        equ   32
f37p        equ   24
f25p        equ   16
f18p        equ   12
f12p        equ   8
f9p         equ   6
f6p         equ   4
f4p         equ   3
f3p         equ   2
f2p         equ   1
f0p         equ   0
OFF         equ   0     ;use for REST event, set velocity = 0

NOFADER           equ   0     ;Fader Increment = 0.

*     Musical note index definitions
C0          equ   0
Cs0         equ   1
Db0         equ   1
D0          equ   2
Ds0         equ   3
Eb0         equ   3
E0          equ   4
F0          equ   5
Fs0         equ   6
Gb0         equ   6
G0          equ   7
Gs0         equ   8
Ab0         equ   8
A0          equ   9
As0         equ   10
Bb0         equ   10
B0          equ   11
C1          equ   12
Cs1         equ   13
Db1         equ   13
D1          equ   14
Ds1         equ   15
Eb1         equ   15
E1          equ   16
```

```
F1          equ   17
Fs1         equ   18
Gb1         equ   18
G1          equ   19
Gs1         equ   20
Ab1         equ   20
A1          equ   21
As1         equ   22
Bb1         equ   22
B1          equ   23
C2          equ   24
Cs2         equ   25
Db2         equ   25
D2          equ   26
Ds2         equ   27
Eb2         equ   27
E2          equ   28
F2          equ   29
Fs2         equ   30
Gb2         equ   30
G2          equ   31
Gs2         equ   32
Ab2         equ   32
A2          equ   33
As2         equ   34
Bb2         equ   34
B2          equ   35
C3          equ   36
Cs3         equ   37
Db3         equ   37
D3          equ   38
Ds3         equ   39
Eb3         equ   39
E3          equ   40
F3          equ   41
Fs3         equ   42
Gb3         equ   42
G3          equ   43
Gs3         equ   44
Ab3         equ   44
A3          equ   45
As3         equ   46
Bb3         equ   46
B3          equ   47
C4          equ   48
Cs4         equ   49
Db4         equ   49
D4          equ   50
```

```
Ds4         equ   51
Eb4         equ   51
E4          equ   52
F4          equ   53
Fs4         equ   54
Gb4         equ   54
G4          equ   55
Gs4         equ   56
Ab4         equ   56
A4          equ   57
As4         equ   58
Bb4         equ   58
B4          equ   59
C5          equ   60
Cs5         equ   61
Db5         equ   61
D5          equ   62
Ds5         equ   63
Eb5         equ   63
E5          equ   64
F5          equ   65
Fs5         equ   66
Gb5         equ   66
G5          equ   67
Gs5         equ   68
Ab5         equ   68
A5          equ   69
As5         equ   70
Bb5         equ   70
B5          equ   71
C6          equ   72
RST         equ   73
REST        equ   120

*     Lookup values for Instrument Sound tables
PatchFLT1   equ   128+0       ;FM Flute tone 1
PatchBRS1   equ   128+1       ;FM Brass tone 1, Medium slow attack
PatchBRS2   equ   128+2       ;FM Brass tone 2, Fast attack
PatchBRS3   equ   128+3       ;FM Brass tone 3, Slow attack
PatchTRM1   equ   128+4       ;FM Brass tone Trombone 1, Slow attack
PatchTRM2   equ   128+5       ;FM Brass tone Trombone 2, Med slow attack
PatchCLR1   equ   128+6       ;FM Clarinet Tone 1
PatchCLR2   equ   128+7       ;FM Clarinet Tone 2, brighter than CLR1
PatchMT1a   equ   128+8       ;FM Metallic tone 1a
PatchMT1b   equ   128+9       ;FM Metallic tone 1b
PatchMT1c   equ   128+10      ;FM Metallic tone 1c
PatchMT2a   equ   128+11      ;FM Metallic tone 2a
PatchMT2b   equ   128+12      ;FM Metallic tone 2b
```

```
PatchMT2c   equ   128+13      ;FM Metallic tone 2c
PatchMT3a   equ   128+14      ;FM Metallic tone 3a
PatchMT3b   equ   128+15      ;FM Metallic tone 3b
PatchMT3c   equ   128+16      ;FM Metallic tone 3c
PatchMT4a   equ   128+17      ;FM Metallic tone 4a
PatchMT4b   equ   128+18      ;FM Metallic tone 4b
PatchMT4c   equ   128+19      ;FM Metallic tone 4c
PatchMT5a   equ   128+20      ;FM Metallic tone 5a
PatchMT5b   equ   128+21      ;FM Metallic tone 5b
PatchMT5c   equ   128+22      ;FM Metallic tone 5c
PatchMT6a   equ   128+23      ;FM Metallic tone 6a, good BASS sound 1
PatchMT6b   equ   128+24      ;FM Metallic tone 6b, good BASS sound 2
PatchMT6c   equ   128+25      ;FM Metallic tone 6c, good BASS sound 3
PatchMT6d   equ   128+26      ;FM Metallic tone 6d
PatchMT6e   equ   128+27      ;FM Metallic tone 6e
PatchMT6f   equ   128+28      ;FM Metallic tone 6f
PatchMT6g   equ   128+29      ;FM Metallic tone 6g
PatchMT6h   equ   128+30      ;FM Metallic tone 6h
PatchMT6i   equ   128+31      ;FM Metallic tone 6i, hard metallic sound 1
PatchMT6j   equ   128+32      ;FM Metallic tone 6j, hard metallic sound 2
PatchMT6k   equ   128+33      ;FM Metallic tone 6k, hard metallic sound 3
PatchMT6l   equ   128+34      ;FM Metallic tone 6l, plucked string 1
PatchMT6m   equ   128+35      ;FM Metallic tone 6m, plucked string 2
PatchMT6n   equ   128+36      ;FM Metallic tone 6n
PatchMT6o   equ   128+37      ;FM Metallic tone 6o, plucked string 3
PatchMT6p   equ   128+38      ;FM Metallic tone 6p, plucked string 4
PatchCHM1   equ   128+39      ;FM Chimes tone 1
PatchCHM2   equ   128+40      ;FM Chimes tone 2
PatchCHM3   equ   128+41      ;FM Chimes tone 3
*     Max # of patch codes is 64

CONTROL     equ   #80   ;< is NOTE data, >= are commands and controls

*     Codes #80 to #BF are reserved for Patch Codes (packed BYTE)
PATCH       equ   #80   ;usage: BYTE PATCH+patchcode, ie 80+MT6o = B7

*     Codes #C0 to #FF are reserved for Commands
CMDS        equ   #C0   ;192 to 255 are Commands

LOADTIMBRE  equ   #d3
StopSong    equ   #D4   ;End of Song
GOTO        equ   #D5   ;Control code for GOTO a Label
RETURN      equ   #D6   ;Control code for Return from a Subroutine
GOSUB       equ   #D7   ;Control code for Play a Subroutine
DETUNE      equ   #D8   ;Control code for Ch2 Detune, Signed offset
TEMPOSYNC   equ   #D9   ;Control code for Sync Ch2/Ch1 Tempo change
MIXDN       equ   #DA   ;Control code for Shift MIX value DOWN
MIXUP       equ   #DB   ;Control code for Shift MIX value UP
RTRNS       equ   #DC   ;Control code for RELATIVE Transpose
```

```
ATRNS        equ   #DD   ;Control code for ABSOLUTE Transpose
TEMPO        equ   #DE   ;Control code for Tempo
FADER        equ   #DF   ;Control code for Set Fader
MIX0         equ   #E0   ;Control code for Modix (Modulation Index)
MIX1         equ   #E1   ;isolate bits 0–3 for table lookup
MIX2         equ   #E2   ;
MIX3         equ   #E3   ;
MIX4         equ   #E4   ;
MIX5         equ   #E5   ;
MIX6         equ   #E6   ;
MIX7         equ   #E7   ;
MIX8         equ   #E8   ;
MIX9         equ   #E9   ;
MIX10        equ   #EA   ;
MIX11        equ   #EB   ;
MIX12        equ   #EC   ;
MIX13        equ   #ED   ;
MIX14        equ   #EE   ;
MIX15        equ   #EF   ;
ENDREP       equ   #F0   ;Control code for End Repeat
BEGREP2      equ   #F1   ;Control code for Start Repeat, Play 2X
BEGREP3      equ   #F2   ;Control code for Start Repeat, Play 3X
BEGREP4      equ   #F3   ;Control code for Start Repeat, Play 4X
BEGREP5      equ   #F4   ;Control code for Start Repeat, Play 5X
BEGREP6      equ   #F5   ;Control code for Start Repeat, Play 6X
BEGREP7      equ   #F6   ;Control code for Start Repeat, Play 7X
BEGREP8      equ   #F7   ;Control code for Start Repeat, Play 8X
ENDMULTI     equ   #F8   ;Control code for End MULTI Repeat
BEGMULTI2    equ   #F9   ;Control code for Start MULTI, Play 2X
BEGMULTI3    equ   #FA   ;Control code for Start MULTI, Play 3X
BEGMULTI4    equ   #FB   ;Control code for Start MULTI, Play 4X
BEGMULTI5    equ   #FC   ;Control code for Start MULTI, Play 5X
BEGMULTI6    equ   #FD   ;Control code for Start MULTI, Play 6X
BEGMULTI7    equ   #FE   ;Control code for Start MULTI, Play 7X
BEGMULTI8    equ   #FF   ;Control code for Start MULTI, Play 8X

*     Tempo values as 1/4 notes (Beats) Per Minute, 12 Clocks per Beat
*                  Index    MyPsc  Exact_BPM  Beat_Prd  Exact_Tmr  Timer
BPM61        equ   0     ;20   61.035156  0.983040  196.608000  197
BPM67        equ   1     ;22   67.138672  0.893673  178.734545  179
BPM73        equ   2     ;24   73.242188  0.819200  163.840000  164
BPM79        equ   3     ;26   79.345703  0.756185  151.236923  151
BPM85        equ   4     ;28   85.449219  0.702171  140.434286  140
BPM92        equ   5     ;30   91.552734  0.655360  131.072000  131
BPM98        equ   6     ;32   97.656250  0.614400  122.880000  123
BPM104       equ   7     ;34  103.759766  0.578259  115.651765  116
BPM110       equ   8     ;36  109.863281  0.546133  109.226667  109
BPM116       equ   9     ;38  115.966797  0.517389  103.477895  103
```

```
BPM122      equ   10    ;40  122.070313  0.491520   98.304000  98
BPM128      equ   11    ;42  128.173828  0.468114   93.622857  94
BPM134      equ   12    ;44  134.277344  0.446836   89.367273  89
BPM140      equ   13    ;46  140.380859  0.427409   85.481739  85
BPM146      equ   14    ;48  146.484375  0.409600   81.920000  82
BPM153      equ   15    ;50  152.587891  0.393216   78.643200  79
BPM159      equ   16    ;52  158.691406  0.378092   75.618462  76
BPM165      equ   17    ;54  164.794922  0.364089   72.817778  73
BPM171      equ   18    ;56  170.898438  0.351086   70.217143  70
BPM177      equ   19    ;58  177.001953  0.338979   67.795862  68
BPM183      equ   20    ;60  183.105469  0.327680   65.536000  66
BPM189      equ   21    ;62  189.208984  0.317110   63.421935  63
BPM195      equ   22    ;64  195.312500  0.307200   61.440000  61
BPM201      equ   23    ;66  201.416016  0.297891   59.578182  60
BPM208      equ   24    ;68  207.519531  0.289129   57.825882  58
BPM214      equ   25    ;70  213.623047  0.280869   56.173714  56
BPM220      equ   26    ;72  219.726563  0.273067   54.613333  55
BPM226      equ   27    ;74  225.830078  0.265686   53.137297  53
BPM232      equ   28    ;76  231.933594  0.258695   51.738947  52
BPM238      equ   29    ;78  238.037109  0.252062   50.412308  50
BPM244      equ   30    ;80  244.140625  0.245760   49.152000  49
BPM250      equ   31    ;82  250.244141  0.239766   47.953171  48
BPM256      equ   32    ;84  256.347656  0.234057   46.811429  47
BPM262      equ   33    ;86  262.451172  0.228614   45.722791  46
BPM269      equ   34    ;88  268.554688  0.223418   44.683636  45
BPM275      equ   35    ;90  274.658203  0.218453   43.690667  44
BPM281      equ   36    ;92  280.761719  0.213704   42.740870  43
BPM287      equ   37    ;94  286.865234  0.209157   41.831489  42
BPM293      equ   38    ;96  292.968750  0.204800   40.960000  41
BPM299      equ   39    ;98  299.072266  0.200620   40.124082  40
BPM305      equ   40    ;100 305.175781  0.196608   39.321600  39

*     Time Signature constants as Negative UP Counter values.
Qnote       equ   –12         ;–12 clocks per 1/4 note, 1/4 = 1 Beat
TS24        equ   2*Qnote     ;2 beats per Measure
TS34        equ   3*Qnote     ;3 beats per Measure
TS44        equ   4*Qnote     ;4 beats per Measure
TS54        equ   5*Qnote     ;5 beats per Measure
TS64        equ   6*Qnote     ;6 beats per Measure
TS74        equ   7*Qnote     ;7 beats per Measure

*     SIGNED Offsets added to MyPsc to Shorten/Lengthen Envelope times
ENVOK       equ   0           ;No Envelope Length adjust
ENVS1       equ   1           ;make Envelope length Shorter by 1
ENVS2       equ   2           ;make Envelope length Shorter by 2
ENVS3       equ   3           ;make Envelope length Shorter by 3
ENVS4       equ   4           ;make Envelope length Shorter by 4
ENVS5       equ   5           ;make Envelope length Shorter by 5
ENVS6       equ   6           ;make Envelope length Shorter by 6
```

```
ENVS7        equ   7     ;make Envelope length Shorter by 7
ENVS8        equ   8     ;make Envelope length Shorter by 8
ENVS10       equ   10    ;make Envelope length Shorter by 10
ENVS12       equ   12    ;make Envelope length Shorter by 12
ENVS14       equ   14    ;make Envelope length Shorter by 14
ENVS16       equ   16    ;make Envelope length Shorter by 16
ENVS18       equ   18    ;make Envelope length Shorter by 18
ENVS20       equ   20    ;make Envelope length Shorter by 20

ENVL1        equ   –1    ;make Envelope length Longer by 1
ENVL2        equ   –2    ;make Envelope length Longer by 2
ENVL3        equ   –3    ;make Envelope length Longer by 3
ENVL4        equ   –4    ;make Envelope length Longer by 4
ENVL5        equ   –5    ;make Envelope length Longer by 5
ENVL6        equ   –6    ;make Envelope length Longer by 6
ENVL7        equ   –7    ;make Envelope length Longer by 7
ENVL8        equ   –8    ;make Envelope length Longer by 8
ENVL10       equ   –10   ;make Envelope length Longer by 10
ENVL12       equ   –12   ;make Envelope length Longer by 12
ENVL14       equ   –14   ;make Envelope length Longer by 14
ENVL16       equ   –16   ;make Envelope length Longer by 16
ENVL18       equ   –18   ;make Envelope length Longer by 18
ENVL20       equ   –20   ;make Envelope length Longer by 20

*     Musical Note Time value definitions (FM)

ngrc         equ   1     ;Grace Note, use with n8m etc
n163         equ   2     ;1/16 note triplet
n16          equ   3     ;1/16 note
n83          equ   4     ;1/8  note triplet
n8m          equ   5     ;1/8  note off beat (use for Shuffle feel)
n8           equ   6     ;1/8  note
n8p          equ   7     ;1/8  note on beat  (use for Shuffle feel)
n43          equ   8     ;1/4  note triplet
nd8          equ   9     ;.1/8 note
n4           equ   12    ;1/4  note
n23          equ   16    ;1/2  note triplet
nd4          equ   18    ;.1/4 note
n2           equ   24    ;1/2  note
nd2          equ   36    ;.1/2 note
n1           equ   48    ;1/1  note
nd1          equ   72    ;.1/1 note
n21          equ   96    ;2/1  note
n31          equ   144   ;3/1  note
n41          equ   192   ;4/1  note
n51          equ   240   ;5/1  note

*     Carrier & Modulator frequency multipliers.
X05          equ   16    ;Multiply * 0.5
```

```
X0707       equ   22    ;Multiply * 0.707
Xd4th       equ   24    ;Multiply * 0.750 (down a Fourth)
X1          equ   32    ;Multiply * 1
X1414       equ   45    ;Multiply * 1.414
Xu5th       equ   48    ;Multiply * 1.500 (up a Fifth)
X2          equ   64    ;Multiply * 2
X3          equ   96    ;Multiply * 3

*     Carrier & Modulator frequency multipliers.
X0_33       equ   12    ;Multiply * 0.3333
X0_5        equ   16    ;Multiply * 0.5
X0_70       equ   22    ;Multiply * 0.707
X0_75       equ   24    ;Multiply * 0.750 (down a Fourth)
;X1         equ   32    ;Multiply * 1
X1_41       equ   45    ;Multiply * 1.414
X1_5        equ   48    ;Multiply * 1.500 (up a Fifth)
;X2         equ   64    ;Multiply * 2
;X3         equ   96    ;Multiply * 3
X4          equ   128   ;Multiply * 4
X5          equ   160   ;Multiply * 5
X6          equ   192   ;Multiply * 6
X7          equ   224   ;Multiply * 7
X7_8        equ   250   ;Multiply * 7.8125
X7_9        equ   255   ;Multiply * 7.96875 (8 bit number max)
X8          equ   256   ;Multiply*8 (NOTE: is this limited
                           to 8 bit number?)
X9          equ   288   ;Multiply*9 (NOTE: is this limited
                           to 8 bit number?)
X10         equ   320   ;Multiply*10 (NOTE:is this limited
                           to 8 bit number?)

*     End of FmequM2.INC
```

Appendix D

MSP53C391/392 Timing Considerations

Topic	Page

D.1 General Constraints

The sound quality of the speech produced by the MSP53C391 and MSP53C392 is sensitive to the timing of the waveforms which transfer speech data to the device from the master microprocessor. Depending upon the algorithm being used to synthesize the speech, the speech data is stored in a circular buffer either 16 or 32 bytes wide. If the data is written to the slave too infrequently, the buffer will empty and the synthesis process will stop or be corrupted. If the MSP53C391 or MSP53C392 is polled too frequently to determine whether or not it is ready to accept new data, then too many of the internal instruction cycles may be used servicing the polling process and the sound quality can be degraded.

Each of the synthesis algorithms tends to use the data in the buffer in *bursts* followed by a period of time in which the data is being utilized before more data is read from the buffer. Once the MSP53C391 or MSP53C392 has been polled and determined to be ready to accept new data, the data should be loaded quickly until the buffer is again full. The read pulses should subsequently be spaced more widely until it is again determined that the buffer is not full and the device is ready to accept new data.

The spacing of the read pulses while the buffer is an important determinant of the synthesized speech quality, but is difficult to specify precisely due to the different data throughput requirements of the different algorithms. In general, the optimal polling frequency will increase with the bit rate of the synthesis algorithm being used. In the sections below are some timing waveforms which we have found to work with the datasets that we have tested. In many cases it will be difficult for the system designer to exactly replicate the timing shown in these sections. The timing should be adjusted to optimize the sound quality for the specific system being designed.

The following general considerations should be observed:

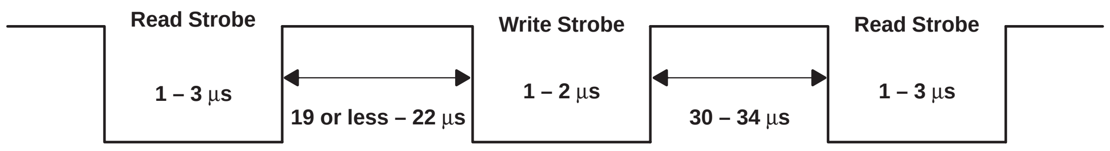
- Keep the STROBE pulses as short as possible, but they should not be shorter than that shown in the waveforms below.
- Once it is determined that the buffer is not full, load new data quickly.
- Once it is determined that the buffer is full, read the status of the BUSY signal periodically, but not as frequently as when refreshing the buffer with new data.

D.2 MSP53C391 Timing Waveforms

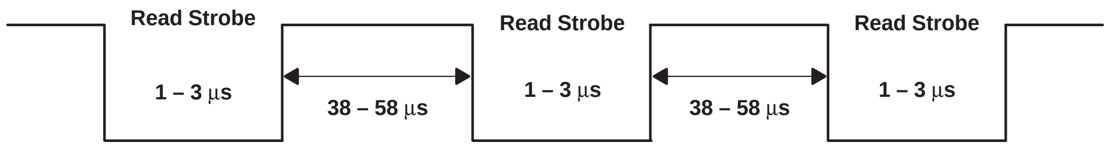
The following waveforms have been found to provide good synthesis results with a variety of data.

The master microprocessor transfers data to the MSP53C391 slave code through 4 bit interface. The master microprocessor also samples two of the data lines to determine the status of the slave. The detailed description about how to transfer data and read back the status of the slave is described in chapter 2. The typical range of the read strobe pulse is 1 µs – 3 µs and the same for write strobe pulse is 1 µs – 2 µs. It is advisable to restrict the strobe pulse widths within the recommended range. The typical range of timing between the rising edge of a read pulse and the falling edge of a write pulse if the slave is not busy is 19 µs – 22 µs. Since the slave expects data from the master *in bursts* to fill up the buffer, the lower limit of this part of the specification could be less than 19 µs but it is advisable not to stretch the higher limit. The typical range of timing between the rising edge of the write strobe pulse and the falling edge of the read strobe pulse if the slave is not busy is 30 µs – 34 µs. The typical range of timing between the rising edge of the read strobe pulse and the falling edge of the write strobe pulse if the slave is busy is 38 µs – 58 µs. This timing could be stretched depending upon the synthesis algorithm, but polling the slave too frequently if the slave is busy would unnecessarily waste time-critical instruction cycles in the MSP53C391 which might affect the quality of the speech. At the same time, polling the slave less frequently would lengthen the response time of the master when the slave needs speech data, which might eventually lead into the exhaustion of the buffer for the new and refreshed speech data.

If the slave is not busy



If the slave is busy



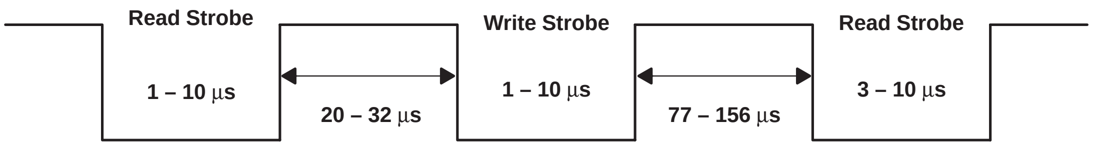
In both cases, it is advisable to follow the timing window for the width of the strobe pulse and also to provide speech data to the slave in response to the interrupt as soon as possible. It is also advisable to keep any interrupt service routine small so that a new Interrupt service request does not occur while the master is processing the previous request.

It is advisable to follow the typical timing window as much as possible for any synthesis algorithm; but if difficulties are found, you can try changing any of those timing windows (except the width of the read and write strobe pulses) to correct the problem for your particular system.

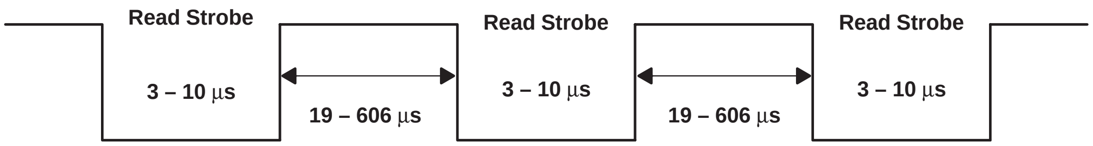
D.3 MSP53C392 Timing Waveforms

The master microprocessor transfers data to the MSP53C392 slave code through 8-bit interface. The master microprocessor also samples two of the data lines to determine the status of the slave. The detailed description about how to transfer data and read back the status of the slave is described in chapter 3. The typical range of the read strobe pulse is 3 µs – 10 µs and the same for write strobe pulse is 1 µs – 10 µs. It is advisable to restrict the strobe pulse widths within the recommended range. The typical range of timing between the rising edge of a read pulse and the falling edge of a write pulse if the slave is NOT BUSY is 20 µs – 32 µs. The typical range of timing between the rising edge of the write strobe pulse and the falling edge of the read strobe pulse if the slave is not busy is 77 µs – 156 µs. The typical range of timing between the rising edge of the read strobe pulse and the falling edge of the write strobe pulse if the slave is BUSY is 19 µs – 606 µs. This timing could be stretched depending upon the synthesis algorithm, but polling the slave too frequently if the slave is BUSY would unnecessarily waste time-critical MSP53C392 instruction cycles which might affect the quality of the speech. At the same time, polling the slave less frequently would lengthen the response time of the master when the slave needs speech data which might eventually lead into the exhaustion of the buffer for the new and refreshed speech data. The length of the timing between two read strobe pulses when the slave is busy is dependent on the length of the timing between the write and read strobe pulse to some extent. Lowering the lower limit of the timing between the strobe pulse between the write and read would lower the higher limit of the length of the timing window between the the two READ strobe pulses when the slave is busy.

If the slave is not busy



If the slave is busy



In both cases, it is advisable to follow the timing window for the width of the strobe pulse and also to provide speech data to the slave in response to the interrupt as soon as possible.

It is advisable to follow the typical timing window as much as possible for any synthesis algorithm; but if difficulties are found, you can try changing any of those timing window (except the width of the read and write strobe pulses) to correct the problem for your particular system.

Listing of FM2INTR1.inc

Topic **Page**

E.1 Listing of FM2INTR1.inc

FM2INTR1.inc contains ?????? The listings are shown in the following:

```
*     Patch tables define each instrument sound.
* PatchFLT1: Flute tone 1
      byte  LOADTIMBRE
      byte  X2                ;Carrier   Fc = 2Fo
      byte  X2                ;Modulator Fm = 2Fo
      byte  16*4              ;Modulation Index Scaler
      byte  2,1                     ;CarAmp, FmAmp    Initial Values
      byte  124,46            ;0    CarInc, FmInc     0  –  7
      byte  –10,80            ;1    CarInc, FmInc     8  –  15
      byte  –9,–8             ;2    CarInc, FmInc     16 –  23
      byte  –8,–7             ;3    CarInc, FmInc     24 –  31
      byte  –7,–6             ;4    CarInc, FmInc     32 –  39
      byte  –6,–5             ;5    CarInc, FmInc     40 –  47
      byte  –5,–4             ;6    CarInc, FmInc     48 –  55
      byte  –81,–3            ;7    CarInc, FmInc     56 –  63
* PatchBRS1: Brass tone 1, Medium slow attack
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1                ;Modulator Fm = Fo
      byte  36*4              ;Modulation Index Scaler
      byte  96,1                    ;CarAmp, FmAmp    Initial Values
      byte  31,24             ;0    CarInc, FmInc     0  –  7
      byte  –18,24                  ;1    CarInc, FmInc     8  –  15
      byte  –17,24            ;2    CarInc, FmInc     16 –  23
      byte  –16,24            ;3    CarInc, FmInc     24 –  31
      byte  –14,8             ;4    CarInc, FmInc     32 –  39
      byte  –13,8             ;5    CarInc, FmInc     40 –  47
      byte  –12,8             ;6    CarInc, FmInc     48 –  55
      byte  –37,6             ;7    CarInc, FmInc     56 –  63
* PatchBRS2: Brass tone 2, Fast attack
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1                ;Modulator Fm = Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,64                  ;CarAmp, FmAmp    Initial Values
      byte  –22,63            ;0    CarInc, FmInc     0  –  7
      byte  –14,–110          ;1    CarInc, FmInc     8  –  15
      byte  –13,13            ;2    CarInc, FmInc     16 –  23
      byte  –11,16            ;3    CarInc, FmInc     24 –  31
      byte  –11,16            ;4    CarInc, FmInc     32 –  39
      byte  –10,15            ;5    CarInc, FmInc     40 –  47
      byte  –7,16             ;6    CarInc, FmInc     48 –  55
      byte  –39,17            ;7    CarInc, FmInc     56 –  63
```

```
* PatchBRS3: Brass tone 3, Slow attack
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1                ;Modulator Fm = Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,69                  ;CarAmp, FmAmp    Initial Values
      byte  –22,58            ;0    CarInc, FmInc     0  –  7
      byte  –18,–50           ;1    CarInc, FmInc     8  –  15
      byte  –17,–44           ;2    CarInc, FmInc     16 –  23
      byte  –16,7             ;3    CarInc, FmInc     24 –  31
      byte  –14,8             ;4    CarInc, FmInc     32 –  39
      byte  –12,9             ;5    CarInc, FmInc     40 –  47
      byte  –10,10            ;6    CarInc, FmInc     48 –  55
      byte  –18,11            ;7    CarInc, FmInc     56 –  63
* PatchTRM1: Brass tone Trombone 1, Slow attack
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X05               ;Modulator Fm = 0.5Fo
      byte  24*4              ;Modulation Index Scaler
      byte  127,69                  ;CarAmp, FmAmp    Initial Values
      byte  –22,58            ;0    CarInc, FmInc     0  –  7
      byte  –18,–50           ;1    CarInc, FmInc     8  –  15
      byte  –17,–44           ;2    CarInc, FmInc     16 –  23
      byte  –16,7             ;3    CarInc, FmInc     24 –  31
      byte  –14,8             ;4    CarInc, FmInc     32 –  39
      byte  –12,9             ;5    CarInc, FmInc     40 –  47
      byte  –10,10            ;6    CarInc, FmInc     48 –  55
      byte  –18,11            ;7    CarInc, FmInc     56 –  63
* PatchTRM2: Brass tone Trombone 2, Med slow attack
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X05               ;Modulator Fm = 0.5Fo
      byte  26*4              ;Modulation Index Scaler
      byte  127,68                  ;CarAmp, FmAmp    Initial Values
      byte  –12,32            ;0    CarInc, FmInc     0  –  7
      byte  –10,27            ;1    CarInc, FmInc     8  –  15
      byte  –8,–64            ;2    CarInc, FmInc     16 –  23
      byte  –6,–48            ;3    CarInc, FmInc     24 –  31
      byte  –4,8              ;4    CarInc, FmInc     32 –  39
      byte  –2,10             ;5    CarInc, FmInc     40 –  47
      byte  –1,12             ;6    CarInc, FmInc     48 –  55
      byte  –84,14            ;7    CarInc, FmInc     56 –  63
* PatchCLR1: Clarinet Tone 1
      byte  LOADTIMBRE
      byte  X3                ;Carrier   Fc = 3Fo
      byte  X2                ;Modulator Fm = 2Fo
      byte  18*4              ;Modulation Index Scaler
      byte  2,1                     ;CarAmp, FmAmp    Initial Values
      byte  124,46            ;0    CarInc, FmInc     0  –  7
      byte  –10,80            ;1    CarInc, FmInc     8  –  15
      byte  –9,–8             ;2    CarInc, FmInc     16 –  23
      byte  –8,–7             ;3    CarInc, FmInc     24 –  31
      byte  –7,–6             ;4    CarInc, FmInc     32 –  39
      byte  –6,–5             ;5    CarInc, FmInc     40 –  47
      byte  –5,–4             ;6    CarInc, FmInc     48 –  55
      byte  –81,–3            ;7    CarInc, FmInc     56 –  63
* PatchCLR2: Clarinet Tone 2, brighter than CLR1
      byte  LOADTIMBRE
      byte  X3                ;Carrier   Fc = 3Fo
      byte  X2                ;Modulator Fm = 2Fo
      byte  24*4              ;Modulation Index Scaler
      byte  3,1                     ;CarAmp, FmAmp    Initial Values
      byte  124,63            ;0    CarInc, FmInc     0  –  7
      byte  –11,63            ;1    CarInc, FmInc     8  –  15
      byte  –7,–4             ;2    CarInc, FmInc     16 –  23
      byte  –6,–3             ;3    CarInc, FmInc     24 –  31
      byte  –8,–4             ;4    CarInc, FmInc     32 –  39
      byte  –6,–4             ;5    CarInc, FmInc     40 –  47
      byte  –5,–4             ;6    CarInc, FmInc     48 –  55
      byte  –81,–108          ;7    CarInc, FmInc     56 –  63
* PatchMT1a: Metallic tone 1a
      byte  LOADTIMBRE
      byte  X2                ;Carrier   Fc = 2Fo
      byte  X0707             ;Modulator Fm = 0.707Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –40,–20           ;0    CarInc, FmInc     0  –  7
      byte  –27,–8            ;1    CarInc, FmInc     8  –  15
      byte  –19,–8            ;2    CarInc, FmInc     16 –  23
      byte  –7,–8             ;3    CarInc, FmInc     24 –  31
      byte  –6,–8             ;4    CarInc, FmInc     32 –  39
      byte  –5,–19            ;5    CarInc, FmInc     40 –  47
      byte  –4,–19            ;6    CarInc, FmInc     48 –  55
      byte  –19,0             ;7    CarInc, FmInc     56 –  63
* PatchMT1b: Metallic tone 1b
      byte  LOADTIMBRE
      byte  X2                ;Carrier   Fc = 2Fo
      byte  X0707             ;Modulator Fm = 0.707Fo
      byte  36*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –16,–15           ;0    CarInc, FmInc     0  –  7
      byte  –16,–8            ;1    CarInc, FmInc     8  –  15
      byte  –16,–8            ;2    CarInc, FmInc     16 –  23
      byte  –16,–8            ;3    CarInc, FmInc     24 –  31
      byte  –16,–4            ;4    CarInc, FmInc     32 –  39
      byte  –16,–2            ;5    CarInc, FmInc     40 –  47
      byte  –16,–1            ;6    CarInc, FmInc     48 –  55
      byte  –15,0             ;7    CarInc, FmInc     56 –  63
* PatchMT1c: Metallic tone 1c
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = 2Fo
      byte  X0707             ;Modulator Fm = 0.707Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –40,–20           ;0    CarInc, FmInc     0  –  7
      byte  –27,–8            ;1    CarInc, FmInc     8  –  15
      byte  –19,–8            ;2    CarInc, FmInc     16 –  23
      byte  –7,–8             ;3    CarInc, FmInc     24 –  31
      byte  –6,–8             ;4    CarInc, FmInc     32 –  39
      byte  –5,–19            ;5    CarInc, FmInc     40 –  47
      byte  –4,–19            ;6    CarInc, FmInc     48 –  55
      byte  –19,0             ;7    CarInc, FmInc     56 –  63
* PatchMT2a: Metallic tone 2a
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1414             ;Modulator Fm = 1.414Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –40,–20           ;0    CarInc, FmInc     0  –  7
      byte  –27,–8            ;1    CarInc, FmInc     8  –  15
      byte  –19,–8            ;2    CarInc, FmInc     16 –  23
      byte  –7,–8             ;3    CarInc, FmInc     24 –  31
      byte  –6,–8             ;4    CarInc, FmInc     32 –  39
      byte  –5,–19            ;5    CarInc, FmInc     40 –  47
      byte  –4,–19            ;6    CarInc, FmInc     48 –  55
      byte  –19,0             ;7    CarInc, FmInc     56 –  63
* PatchMT2b: Metallic tone 2b
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1414             ;Modulator Fm = 1.414Fo
      byte  36*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –16,–15           ;0    CarInc, FmInc     0  –  7
      byte  –16,–8            ;1    CarInc, FmInc     8  –  15
      byte  –16,–8            ;2    CarInc, FmInc     16 –  23
      byte  –16,–8            ;3    CarInc, FmInc     24 –  31
      byte  –16,–4            ;4    CarInc, FmInc     32 –  39
      byte  –16,–2            ;5    CarInc, FmInc     40 –  47
      byte  –16,–1            ;6    CarInc, FmInc     48 –  55
      byte  –15,0             ;7    CarInc, FmInc     56 –  63
* PatchMT2c: Metallic tone 2c
      byte  LOADTIMBRE
      byte  X2                ;Carrier   Fc = 2Fo
      byte  X1414             ;Modulator Fm = 1.414Fo
      byte  32*4                ;Modulation Index Scaler
      byte  127,127                   ;CarAmp, FmAmp    Initial Values
      byte  –40,–20             ;0    CarInc, FmInc     0  –  7
      byte  –27,–8              ;1    CarInc, FmInc     8  –  15
      byte  –19,–8              ;2    CarInc, FmInc     16 –  23
      byte  –7,–8               ;3    CarInc, FmInc     24 –  31
      byte  –6,–8               ;4    CarInc, FmInc     32 –  39
      byte  –5,–19              ;5    CarInc, FmInc     40 –  47
      byte  –4,–19              ;6    CarInc, FmInc     48 –  55
      byte  –19,0               ;7    CarInc, FmInc     56 –  63
* PatchMT3a: Metallic tone 3a
      byte  LOADTIMBRE
      byte  X1                  ;Carrier   Fc = Fo
      byte  Xd4th               ;Modulator Fm = 0.750Fo
      byte  32*4                ;Modulation Index Scaler
      byte  127,127                   ;CarAmp, FmAmp    Initial Values
      byte  –40,–20             ;0    CarInc, FmInc     0  –  7
      byte  –27,–8              ;1    CarInc, FmInc     8  –  15
      byte  –19,–8              ;2    CarInc, FmInc     16 –  23
      byte  –7,–8               ;3    CarInc, FmInc     24 –  31
      byte  –6,–8               ;4    CarInc, FmInc     32 –  39
      byte  –5,–19              ;5    CarInc, FmInc     40 –  47
      byte  –4,–19              ;6    CarInc, FmInc     48 –  55
      byte  –19,0               ;7    CarInc, FmInc     56 –  63
* PatchMT3b: Metallic tone 3b
      byte  LOADTIMBRE
      byte  X1                  ;Carrier   Fc = Fo
      byte  Xd4th               ;Modulator Fm = 0.750Fo
      byte  32*4                ;Modulation Index Scaler
      byte  127,127                   ;CarAmp, FmAmp    Initial Values
      byte  –16,–15             ;0    CarInc, FmInc     0  –  7
      byte  –16,–8              ;1    CarInc, FmInc     8  –  15
      byte  –16,–8              ;2    CarInc, FmInc     16 –  23
      byte  –16,–8              ;3    CarInc, FmInc     24 –  31
      byte  –16,–4              ;4    CarInc, FmInc     32 –  39
      byte  –16,–2              ;5    CarInc, FmInc     40 –  47
      byte  –16,–1              ;6    CarInc, FmInc     48 –  55
      byte  –15,0               ;7    CarInc, FmInc     56 –  63
* PatchMT3c: Metallic tone 3c
      byte  LOADTIMBRE
      byte  X2                  ;Carrier   Fc = 2Fo
      byte  Xd4th               ;Modulator Fm = 0.750Fo
      byte  32*4                ;Modulation Index Scaler
      byte  127,127                   ;CarAmp, FmAmp    Initial Values
      byte  –40,–20             ;0    CarInc, FmInc     0  –  7
      byte  –27,–8              ;1    CarInc, FmInc     8  –  15
      byte  –19,–8              ;2    CarInc, FmInc     16 –  23
      byte  –7,–8               ;3    CarInc, FmInc     24 –  31
      byte  –6,–8             ;4    CarInc, FmInc     32 –  39
      byte  –5,–19            ;5    CarInc, FmInc     40 –  47
      byte  –4,–19            ;6    CarInc, FmInc     48 –  55
      byte  –19,0             ;7    CarInc, FmInc     56 –  63
* PatchMT4a: Metallic tone 4a
      byte  LOADTIMBRE
      byte  X2                ;Carrier   Fc = 2Fo
      byte  Xu5th             ;Modulator Fm = 1.500Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –40,–20           ;0    CarInc, FmInc     0  –  7
      byte  –27,–8            ;1    CarInc, FmInc     8  –  15
      byte  –19,–8            ;2    CarInc, FmInc     16 –  23
      byte  –7,–8             ;3    CarInc, FmInc     24 –  31
      byte  –6,–8             ;4    CarInc, FmInc     32 –  39
      byte  –5,–19            ;5    CarInc, FmInc     40 –  47
      byte  –4,–19            ;6    CarInc, FmInc     48 –  55
      byte  –19,0             ;7    CarInc, FmInc     56 –  63
* PatchMT4b: Metallic tone 4b
      byte  LOADTIMBRE
      byte  X2                ;Carrier   Fc = 2Fo
      byte  Xu5th             ;Modulator Fm = 1.500Fo
      byte  36*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –16,–15           ;0    CarInc, FmInc     0  –  7
      byte  –16,–8            ;1    CarInc, FmInc     8  –  15
      byte  –16,–8            ;2    CarInc, FmInc     16 –  23
      byte  –16,–8            ;3    CarInc, FmInc     24 –  31
      byte  –16,–4            ;4    CarInc, FmInc     32 –  39
      byte  –16,–2            ;5    CarInc, FmInc     40 –  47
      byte  –16,–1            ;6    CarInc, FmInc     48 –  55
      byte  –15,0             ;7    CarInc, FmInc     56 –  63
* PatchMT4c: Metallic tone 4c
      byte  LOADTIMBRE
      byte  X2                ;Carrier   Fc = 2Fo
      byte  Xu5th             ;Modulator Fm = 1.500Fo
      byte  28*4              ;Modulation Index Scaler
      byte  127,1                   ;CarAmp, FmAmp    Initial Values
      byte  –22,32            ;0    CarInc, FmInc     0  –  7
      byte  –14,32            ;1    CarInc, FmInc     8  –  15
      byte  –13,31            ;2    CarInc, FmInc     16 –  23
      byte  –11,31            ;3    CarInc, FmInc     24 –  31
      byte  –9,–32            ;4    CarInc, FmInc     32 –  39
      byte  –10,–32           ;5    CarInc, FmInc     40 –  47
      byte  –7,–32            ;6    CarInc, FmInc     48 –  55
      byte  –41,–31           ;7    CarInc, FmInc     56 –  63
* PatchMT5a: Metallic tone 5a
      byte  LOADTIMBRE
      byte  Xd4th               ;Carrier   Fc = 0.75Fo
      byte  X2                  ;Modulator Fm = 2Fo
      byte  32*4                ;Modulation Index Scaler
      byte  127,127                   ;CarAmp, FmAmp    Initial Values
      byte  –40,–20             ;0    CarInc, FmInc     0  –  7
      byte  –27,–8              ;1    CarInc, FmInc     8  –  15
      byte  –19,–8              ;2    CarInc, FmInc     16 –  23
      byte  –7,–8               ;3    CarInc, FmInc     24 –  31
      byte  –6,–8               ;4    CarInc, FmInc     32 –  39
      byte  –5,–19              ;5    CarInc, FmInc     40 –  47
      byte  –4,–19              ;6    CarInc, FmInc     48 –  55
      byte  –19,0               ;7    CarInc, FmInc     56 –  63
* PatchMT5b: Metallic tone 5b
      byte  LOADTIMBRE
      byte  Xd4th               ;Carrier   Fc = 0.75Fo
      byte  X2                        ;Modulator Fm = 2Fo
      byte  36*4          ;Modulation Index Scaler
      byte  127,127                   ;CarAmp, FmAmp    Initial Values
      byte  –16,–15             ;0    CarInc, FmInc     0  –  7
      byte  –16,–8              ;1    CarInc, FmInc     8  –  15
      byte  –16,–8              ;2    CarInc, FmInc     16 –  23
      byte  –16,–8              ;3    CarInc, FmInc     24 –  31
      byte  –16,–4              ;4    CarInc, FmInc     32 –  39
      byte  –16,–2              ;5    CarInc, FmInc     40 –  47
      byte  –16,–1              ;6    CarInc, FmInc     48 –  55
      byte  –15,0               ;7    CarInc, FmInc     56 –  63
* PatchMT5c: Metallic tone 5c
      byte  LOADTIMBRE
      byte  Xd4th               ;Carrier   Fc = 0.75Fo
      byte  X2                  ;Modulator Fm = 2Fo
      byte  24*4                ;Modulation Index Scaler
      byte  96,1                ;CarAmp, FmAmp    Initial Values
      byte  31,24               ;0    CarInc, FmInc     0  –  7
      byte  –18,24              ;1    CarInc, FmInc     8  –  15
      byte  –17,24              ;2    CarInc, FmInc     16 –  23
      byte  –16,24              ;3    CarInc, FmInc     24 –  31
      byte  –14,8               ;4    CarInc, FmInc     32 –  39
      byte  –13,8               ;5    CarInc, FmInc     40 –  47
      byte  –12,8               ;6    CarInc, FmInc     48 –  55
      byte  –37,6               ;7    CarInc, FmInc     56 –  63
* PatchMT6a: Metallic tone 6a, good BASS sound 1
      byte  LOADTIMBRE
      byte  X1                  ;Carrier   Fc = Fo
      byte  X05                 ;Modulator Fm = 0.5Fo
      byte  32*4                ;Modulation Index Scaler
      byte  127,127                   ;CarAmp, FmAmp    Initial Values
      byte  –16,–16             ;0    CarInc, FmInc     0  –  7
      byte  –10,–16             ;1    CarInc, FmInc     8  –  15
      byte  –6,–6             ;2    CarInc, FmInc     16 –  23
      byte  –4,–4             ;3    CarInc, FmInc     24 –  31
      byte  –2,–2             ;4    CarInc, FmInc     32 –  39
      byte  –1,–1             ;5    CarInc, FmInc     40 –  47
      byte  –10,–5            ;6    CarInc, FmInc     48 –  55
      byte  –78,–16           ;7    CarInc, FmInc     56 –  63
* PatchMT6b: Metallic tone 6b, good BASS sound 2
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X05               ;Modulator Fm = 0.5Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,120                 ;CarAmp, FmAmp    Initial Values
      byte  –24,7             ;0    CarInc, FmInc     0  –  7
      byte  –12,–20           ;1    CarInc, FmInc     8  –  15
      byte  –6,–10            ;2    CarInc, FmInc     16 –  23
      byte  –3,–5             ;3    CarInc, FmInc     24 –  31
      byte  –2,–3             ;4    CarInc, FmInc     32 –  39
      byte  –1,–1             ;5    CarInc, FmInc     40 –  47
      byte  –12,–6            ;6    CarInc, FmInc     48 –  55
      byte  –67,–12           ;7    CarInc, FmInc     56 –  63
* PatchMT6c: Metallic tone 6c, good BASS sound 3
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X05               ;Modulator Fm = 0.5Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –24,–20           ;0    CarInc, FmInc     0  –  7
      byte  –18,–15           ;1    CarInc, FmInc     8  –  15
      byte  –6,–12            ;2    CarInc, FmInc     16 –  23
      byte  –3,–10            ;3    CarInc, FmInc     24 –  31
      byte  –2,–8             ;4    CarInc, FmInc     32 –  39
      byte  –1,–5             ;5    CarInc, FmInc     40 –  47
      byte  –18,–3            ;6    CarInc, FmInc     48 –  55
      byte  –55,–1            ;7    CarInc, FmInc     56 –  63
* PatchMT6d: Metallic tone 6d
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X2                ;Modulator Fm = 2Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –16,–16           ;0    CarInc, FmInc     0  –  7
      byte  –10,–16           ;1    CarInc, FmInc     8  –  15
      byte  –6,–6             ;2    CarInc, FmInc     16 –  23
      byte  –4,–4             ;3    CarInc, FmInc     24 –  31
      byte  –2,–2             ;4    CarInc, FmInc     32 –  39
      byte  –1,–1             ;5    CarInc, FmInc     40 –  47
      byte  –10,–5            ;6    CarInc, FmInc     48 –  55
      byte  –78,–16           ;7    CarInc, FmInc     56 –  63
```

```
* PatchMT6e: Metallic tone 6e
      byte  LOADTIMBRE
      byte  X1                 ;Carrier   Fc = Fo
      byte  X2                 ;Modulator Fm = 2Fo
      byte  32*4               ;Modulation Index Scaler
      byte  127,120                  ;CarAmp, FmAmp    Initial Values
      byte  –24,7              ;0    CarInc, FmInc     0  –  7
      byte  –12,–20            ;1    CarInc, FmInc     8  –  15
      byte  –6,–10             ;2    CarInc, FmInc     16 –  23
      byte  –3,–5              ;3    CarInc, FmInc     24 –  31
      byte  –2,–3              ;4    CarInc, FmInc     32 –  39
      byte  –1,–1              ;5    CarInc, FmInc     40 –  47
      byte  –12,–6             ;6    CarInc, FmInc     48 –  55
      byte  –67,–12            ;7    CarInc, FmInc     56 –  63
* PatchMT6f: Metallic tone 6f
      byte  LOADTIMBRE
      byte  X1                 ;Carrier   Fc = Fo
      byte  X2                 ;Modulator Fm = 2Fo
      byte  32*4               ;Modulation Index Scaler
      byte  127,127                  ;CarAmp, FmAmp    Initial Values
      byte  –24,–20            ;0    CarInc, FmInc     0  –  7
      byte  –18,–15            ;1    CarInc, FmInc     8  –  15
      byte  –6,–12             ;2    CarInc, FmInc     16 –  23
      byte  –3,–10             ;3    CarInc, FmInc     24 –  31
      byte  –2,–8              ;4    CarInc, FmInc     32 –  39
      byte  –1,–5              ;5    CarInc, FmInc     40 –  47
      byte  –18,–3             ;6    CarInc, FmInc     48 –  55
      byte  –55,–1             ;7    CarInc, FmInc     56 –  63
* PatchMT6g: Metallic tone 6g
      byte  LOADTIMBRE
      byte  X1                 ;Carrier   Fc = Fo
      byte  X0707              ;Modulator Fm = 0.707Fo
      byte  36*4               ;Modulation Index Scaler
      byte  127,1              ;CarAmp, FmAmp    Initial Values
      byte  –22,32             ;0    CarInc, FmInc     0  –  7
      byte  –14,32             ;1    CarInc, FmInc     8  –  15
      byte  –13,31             ;2    CarInc, FmInc     16 –  23
      byte  –11,31             ;3    CarInc, FmInc     24 –  31
      byte  –9,–32             ;4    CarInc, FmInc     32 –  39
      byte  –10,–32            ;5    CarInc, FmInc     40 –  47
      byte  –7,–32             ;6    CarInc, FmInc     48 –  55
      byte  –41,–31            ;7    CarInc, FmInc     56 –  63
* PatchMT6h: Metallic tone 6h
      byte  LOADTIMBRE
      byte  X2                 ;Carrier   Fc = 2Fo
      byte  X1                 ;Modulator Fm = Fo
      byte  32*4               ;Modulation Index Scaler
      byte  127,127                  ;CarAmp, FmAmp    Initial Values
      byte  –24,–20           ;0    CarInc, FmInc     0  –  7
      byte  –18,–15           ;1    CarInc, FmInc     8  –  15
      byte  –6,–12            ;2    CarInc, FmInc     16 –  23
      byte  –3,–10            ;3    CarInc, FmInc     24 –  31
      byte  –2,–8             ;4    CarInc, FmInc     32 –  39
      byte  –1,–5             ;5    CarInc, FmInc     40 –  47
      byte  –18,–3            ;6    CarInc, FmInc     48 –  55
      byte  –55,–1            ;7    CarInc, FmInc     56 –  63
* PatchMT6i: Metallic tone 6i, hard metallic sound 1
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = 2Fo
      byte  X3                ;Modulator Fm = Fo
      byte  28*4              ;Modulation Index Scaler
      byte  127,120                 ;CarAmp, FmAmp    Initial Values
      byte  –24,7             ;0    CarInc, FmInc     0  –  7
      byte  –12,–20           ;1    CarInc, FmInc     8  –  15
      byte  –6,–10            ;2    CarInc, FmInc     16 –  23
      byte  –3,–5             ;3    CarInc, FmInc     24 –  31
      byte  –2,–3             ;4    CarInc, FmInc     32 –  39
      byte  –1,–1             ;5    CarInc, FmInc     40 –  47
      byte  –12,–6            ;6    CarInc, FmInc     48 –  55
      byte  –67,–12           ;7    CarInc, FmInc     56 –  63
* PatchMT6j: Metallic tone 6j, hard metallic sound 2
      byte  LOADTIMBRE
      byte  X3                ;Carrier   Fc = 2Fo
      byte  X1                ;Modulator Fm = Fo
      byte  20*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –24,–20           ;0    CarInc, FmInc     0  –  7
      byte  –18,–15           ;1    CarInc, FmInc     8  –  15
      byte  –6,–12            ;2    CarInc, FmInc     16 –  23
      byte  –3,–10            ;3    CarInc, FmInc     24 –  31
      byte  –2,–8             ;4    CarInc, FmInc     32 –  39
      byte  –1,–5             ;5    CarInc, FmInc     40 –  47
      byte  –18,–3            ;6    CarInc, FmInc     48 –  55
      byte  –55,–1            ;7    CarInc, FmInc     56 –  63
* PatchMT6k: Metallic tone 6k, hard metallic sound 3
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X3                ;Modulator Fm = 3Fo
      byte  28*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –16,–16           ;0    CarInc, FmInc     0  –  7
      byte  –10,–16           ;1    CarInc, FmInc     8  –  15
      byte  –6,–6             ;2    CarInc, FmInc     16 –  23
      byte  –4,–4             ;3    CarInc, FmInc     24 –  31
      byte  –2,–2             ;4    CarInc, FmInc     32 –  39
      byte  –1,–1             ;5    CarInc, FmInc     40 –  47
      byte  –10,–5            ;6    CarInc, FmInc     48 –  55
      byte  –78,–16           ;7    CarInc, FmInc     56 –  63
* PatchMT6l: Metallic tone 6l, plucked string 1
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = 2Fo
      byte  X2                ;Modulator Fm = Fo
      byte  28*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –16,–16           ;0    CarInc, FmInc     0  –  7
      byte  –10,–16           ;1    CarInc, FmInc     8  –  15
      byte  –6,–6             ;2    CarInc, FmInc     16 –  23
      byte  –4,–4             ;3    CarInc, FmInc     24 –  31
      byte  –2,–2             ;4    CarInc, FmInc     32 –  39
      byte  –1,–1             ;5    CarInc, FmInc     40 –  47
      byte  –10,–5            ;6    CarInc, FmInc     48 –  55
      byte  –78,–16           ;7    CarInc, FmInc     56 –  63
* PatchMT6m: Metallic tone 6m, plucked string 2
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = 2Fo
      byte  X2                ;Modulator Fm = Fo
      byte  32*4              ;Modulation Index Scaler
      byte  127,120                 ;CarAmp, FmAmp    Initial Values
      byte  –24,7             ;0    CarInc, FmInc     0  –  7
      byte  –12,–20           ;1    CarInc, FmInc     8  –  15
      byte  –6,–10            ;2    CarInc, FmInc     16 –  23
      byte  –3,–5             ;3    CarInc, FmInc     24 –  31
      byte  –2,–3             ;4    CarInc, FmInc     32 –  39
      byte  –1,–1             ;5    CarInc, FmInc     40 –  47
      byte  –12,–6            ;6    CarInc, FmInc     48 –  55
      byte  –67,–12           ;7    CarInc, FmInc     56 –  63
* PatchMT6n: Metallic tone 6n
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X05               ;Modulator Fm = 0.5Fo
      byte  36*4              ;Modulation Index Scaler
      byte  127,1             ;CarAmp, FmAmp    Initial Values
      byte  –22,32            ;0    CarInc, FmInc     0  –  7
      byte  –14,32            ;1    CarInc, FmInc     8  –  15
      byte  –13,31            ;2    CarInc, FmInc     16 –  23
      byte  –11,31            ;3    CarInc, FmInc     24 –  31
      byte  –9,–32            ;4    CarInc, FmInc     32 –  39
      byte  –10,–32           ;5    CarInc, FmInc     40 –  47
      byte  –7,–32            ;6    CarInc, FmInc     48 –  55
      byte  –41,–31           ;7    CarInc, FmInc     56 –  63
* PatchMT6o: Metallic tone 6o, plucked string 3
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = 2Fo
      byte  X2                ;Modulator Fm = Fo
      byte  28*4              ;Modulation Index Scaler
      byte  127,127                 ;CarAmp, FmAmp    Initial Values
      byte  –24,–20           ;0    CarInc, FmInc     0  –  7
      byte  –18,–15           ;1    CarInc, FmInc     8  –  15
      byte  –6,–12            ;2    CarInc, FmInc     16 –  23
      byte  –3,–10            ;3    CarInc, FmInc     24 –  31
      byte  –2,–8             ;4    CarInc, FmInc     32 –  39
      byte  –1,–5             ;5    CarInc, FmInc     40 –  47
      byte  –18,–3            ;6    CarInc, FmInc     48 –  55
      byte  –55,–1            ;7    CarInc, FmInc     56 –  63
* PatchMT6p: Metallic tone 6p, plucked string 4
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = 2Fo
      byte  X2                ;Modulator Fm = Fo
      byte  28*4              ;Modulation Index Scaler
      byte  127,64                  ;CarAmp, FmAmp    Initial Values
      byte  –20,63            ;0    CarInc, FmInc     0  –  7
      byte  –12,–15           ;1    CarInc, FmInc     8  –  15
      byte  –6,–12            ;2    CarInc, FmInc     16 –  23
      byte  –3,–10            ;3    CarInc, FmInc     24 –  31
      byte  –2,–8             ;4    CarInc, FmInc     32 –  39
      byte  –1,–5             ;5    CarInc, FmInc     40 –  47
      byte  –18,–3            ;6    CarInc, FmInc     48 –  55
      byte  –65,–1            ;7    CarInc, FmInc     56 –  63
* PatchCHM1: Chimes tone 1
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1414             ;Modulator Fm = 1.414Fo
      byte  24*4              ;Modulation Index Scaler
      byte  127,120                 ;CarAmp, FmAmp    Initial Values
      byte  –28,–13           ;0    CarInc, FmInc     0  –  7
      byte  –20,–11           ;1    CarInc, FmInc     8  –  15
      byte  –11,–10           ;2    CarInc, FmInc     16 –  23
      byte  –8,–9             ;3    CarInc, FmInc     24 –  31
      byte  –6,–8             ;4    CarInc, FmInc     32 –  39
      byte  –7,–5             ;5    CarInc, FmInc     40 –  47
      byte  –6,–3             ;6    CarInc, FmInc     48 –  55
      byte  –41,–1            ;7    CarInc, FmInc     56 –  63
* PatchCHM2: Chimes tone 2
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1414             ;Modulator Fm = 1.414Fo
      byte  24*4              ;Modulation Index Scaler
      byte  127,120                 ;CarAmp, FmAmp    Initial Values
      byte  –22,–20           ;0    CarInc, FmInc     0  –  7
      byte  –20,–22           ;1    CarInc, FmInc     8  –  15
      byte  –18,–15           ;2    CarInc, FmInc     16 –  23
      byte  –14,–11           ;3    CarInc, FmInc     24 –  31
      byte  –9,–8             ;4    CarInc, FmInc     32 –  39
      byte  –7,–6             ;5    CarInc, FmInc     40 –  47
      byte  –8,–5             ;6    CarInc, FmInc     48 –  55
      byte  –28,–3            ;7    CarInc, FmInc     56 –  63
* PatchCHM3: Chimes tone 3
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = Fo
      byte  X1414             ;Modulator Fm = 1.414Fo
      byte  24*4              ;Modulation Index Scaler
      byte  127,0                   ;CarAmp, FmAmp    Initial Values
      byte  –20,42            ;0    CarInc, FmInc     0  –  7
      byte  –12,42            ;1    CarInc, FmInc     8  –  15
      byte  –6,42             ;2    CarInc, FmInc     16 –  23
      byte  –3,–25            ;3    CarInc, FmInc     24 –  31
      byte  –2,–23            ;4    CarInc, FmInc     32 –  39
      byte  –1,–21            ;5    CarInc, FmInc     40 –  47
      byte  –18,–19           ;6    CarInc, FmInc     48 –  55
      byte  –65,–17           ;7    CarInc, FmInc     56 –  63
* PatchPNO1:  Piano tone 1
      byte  LOADTIMBRE
      byte  X1                ;Carrier   Fc = 1*Fo
      byte  X1                ;Modulator Fm = 1*Fo
      byte  38*4              ;MIX Scaler (higher#=more timbre chg.
                                 per progressive Velocity & MIXn )
      byte  126,108                 ;CarAmp, FmAmp    Initial Values
      byte  –13,006           ;0    CarInc, FmInc     0  –  7
      byte  –25,013           ;1    CarInc, FmInc     8  –  15
      byte  –13,–19           ;2    CarInc, FmInc     16 –  23
      byte  –25,–19           ;3    CarInc, FmInc     24 –  31
      byte  000,–06           ;4    CarInc, FmInc     32 –  39
      byte  000,–25           ;5    CarInc, FmInc     40 –  47
      byte  –38,–32           ;6    CarInc, FmInc     48 –  55
      byte  –10,–23           ;7    CarInc, FmInc     56 –  63
* PatchPNO2: Piano tone 2
      byte  LOADTIMBRE
   byte X1                    ;Carrier   Fc = 1*Fo
   byte X1                    ;Modulator Fm = 1*Fo
   byte 32*4                  ;MIX Scaler (higher#=more timbre chg.
                                 per progressive Velocity & MIXn )
   byte  127,127                    ;CarAmp, FmAmp    Initial Values
   byte  –13,–20              ;0    CarInc, FmInc     0  –  7
   byte  –25,–41              ;1    CarInc, FmInc     8  –  15
   byte  –25,–28              ;2    CarInc, FmInc     16 –  23
   byte  000,000              ;3    CarInc, FmInc     24 –  31
   byte  000,000              ;4    CarInc, FmInc     32 –  39
   byte  –19,000              ;5    CarInc, FmInc     40 –  47
   byte  –19,000              ;6    CarInc, FmInc     48 –  55
   byte  –23,–36              ;7    CarInc, FmInc     56 –  63
```

```
* PatchEP_1: Electric Piano tone 1
      byte  LOADTIMBRE
   byte X2              ;Carrier   Fc = 1*Fo
   byte X10             ;Modulator Fm = 10*Fo
   byte 30*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  127,127              ;CarAmp, FmAmp    Initial Values
   byte  –13,–20        ;0    CarInc, FmInc     0  –  7
   byte  –25,–41        ;1    CarInc, FmInc     8  –  15
   byte  –25,–28        ;2    CarInc, FmInc     16 –  23
   byte  000,000        ;3    CarInc, FmInc     24 –  31
   byte  000,000        ;4    CarInc, FmInc     32 –  39
   byte  –19,000        ;5    CarInc, FmInc     40 –  47
   byte  –19,000        ;6    CarInc, FmInc     48 –  55
   byte  –23,–36        ;7    CarInc, FmInc     56 –  63
* PatchEP_2: Electric Piano tone 2 (more aggressive)
      byte  LOADTIMBRE
   byte X2              ;Carrier   Fc = 1*Fo
   byte X10             ;Modulator Fm = 10*Fo
   byte 42*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  127,127              ;CarAmp, FmAmp    Initial Values
   byte  –13,–13        ;0    CarInc, FmInc     0  –  7
   byte  –25,–06        ;1    CarInc, FmInc     8  –  15
   byte  –25,–06        ;2    CarInc, FmInc     16 –  23
   byte  000,–13        ;3    CarInc, FmInc     24 –  31
   byte  000,–25        ;4    CarInc, FmInc     32 –  39
   byte  –19,–38        ;5    CarInc, FmInc     40 –  47
   byte  –19,–19        ;6    CarInc, FmInc     48 –  55
   byte  –23,–04        ;7    CarInc, FmInc     56 –  63
* PatchBNJ1: Banjo 1
      byte  LOADTIMBRE
   byte X3              ;Carrier   Fc = 3*Fo
   byte X1              ;Modulator Fm = 1*Fo
   byte 62*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  127,127              ;CarAmp, FmAmp    Initial Values
   byte  –25,–01        ;0    CarInc, FmInc     0  –  7
   byte  –25,–01        ;1    CarInc, FmInc     8  –  15
   byte  –25,–01        ;2    CarInc, FmInc     16 –  23
   byte  –25,–01        ;3    CarInc, FmInc     24 –  31
   byte  –25,–01        ;4    CarInc, FmInc     32 –  39
   byte  000,–01        ;5    CarInc, FmInc     40 –  47
   byte  000,–01        ;6    CarInc, FmInc     48 –  55
   byte  000,–04        ;7    CarInc, FmInc     56 –  63
* PatchGTR1: Guitar 1
      byte  LOADTIMBRE
   byte X1              ;Carrier   Fc = 1*Fo
   byte X5              ;Modulator Fm = 5*Fo
   byte 30*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  126,108           ;CarAmp, FmAmp    Initial Values
   byte  –13,006        ;0    CarInc, FmInc  0  –  7
   byte  –25,013        ;1    CarInc, FmInc  8  –  15
   byte  –13,–19        ;2    CarInc, FmInc  16 –  23
   byte  –25,–19        ;3    CarInc, FmInc  24 –  31
   byte  000,–06        ;4    CarInc, FmInc  32 –  39
   byte  000,–25        ;5    CarInc, FmInc  40 –  47
   byte  –38,–32        ;6    CarInc, FmInc  48 –  55
   byte  –10,–23        ;7    CarInc, FmInc  56 –  63
* PatchHRP1: Harp 1
      byte  LOADTIMBRE
   byte X1              ;Carrier   Fc = 1*Fo
   byte X2              ;Modulator Fm = 2*Fo
   byte 20*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  127,089              CarAmp, FmAmp     Initial Values
   byte  –38,038        ;0    CarInc, FmInc     0  –  7
   byte  –13,–25        ;1    CarInc, FmInc     8  –  15
   byte  000,–25        ;2    CarInc, FmInc     16 –  23
   byte  000,–25        ;3    CarInc, FmInc     24 –  31
   byte  000,–25        ;4    CarInc, FmInc     32 –  39
   byte  000,–25        ;5    CarInc, FmInc     40 –  47
   byte  –13,000        ;6    CarInc, FmInc     48 –  55
   byte  –57,000        ;7    CarInc, FmInc     56 –  63
* PatchEGT1: Electric Guitar 1
      byte  LOADTIMBRE
   byte X1              ;Carrier   Fc = 1*Fo
   byte X0_5            ;Modulator Fm = 0.5*Fo
   byte 50*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  102,127              CarAmp, FmInc     Initial Values
   byte  013,–06        ;0    CarInc, FmInc     0  –  7
   byte  000,–06        ;1    CarInc, FmInc     8  –  15
   byte  000,–06        ;2    CarInc, FmInc     16 –  23
   byte  –13,–06        ;3    CarInc, FmInc     24 –  31
   byte  –13,000        ;4    CarInc, FmInc     32 –  39
   byte  000,000        ;5    CarInc, FmInc     40 –  47
   byte  000,000        ;6    CarInc, FmInc     48 –  55
   byte  –89,000        ;7    CarInc, FmInc     56 –  63
* PatchXYL1: Xylophone
      byte  LOADTIMBRE
   byte X4              ;Carrier   Fc = 4*Fo
   byte X8              ;Modulator Fm = 8*Fo
   byte 30*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  127,127              CarAmp, FmInc     Initial Values
   byte  –25,–25        ;0    CarInc, FmInc     0  –  7
   byte  –25,–25        ;1    CarInc, FmInc     8  –  15
   byte  –25,–13        ;2    CarInc, FmInc     16 –  23
   byte  –25,–13        ;3    CarInc, FmInc     24 –  31
   byte  –13,–25        ;4    CarInc, FmInc     32 –  39
   byte  –06,–25        ;5    CarInc, FmInc     40 –  47
   byte  –06,000        ;6    CarInc, FmInc     48 –  55
   byte  000,000        ;7    CarInc, FmInc     56 –  63
* PatchVIB1: Vibraphone
      byte  LOADTIMBRE
   byte X1              ;Carrier   Fc = 1*Fo
   byte X5              ;Modulator Fm = 5*Fo
   byte 20*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  127,069              CarAmp, FmInc     Initial Values
   byte  000,–13        ;0    CarInc, FmInc     0  –  7
   byte  000,–13        ;1    CarInc, FmInc     8  –  15
   byte  –13,–13        ;2    CarInc, FmInc     16 –  23
   byte  –25,–13        ;3    CarInc, FmInc     24 –  31
   byte  –25,–01        ;4    CarInc, FmInc     32 –  39
   byte  –25,–03        ;5    CarInc, FmInc     40 –  47
   byte  –13,–01        ;6    CarInc, FmInc     48 –  55
   byte  –25,–10        ;7    CarInc, FmInc     56 –  63
* PatchFLT1: Flute Tone 1
      byte  LOADTIMBRE
   byte X2              ;Carrier   Fc = 2*Fo
   byte X2              ;Modulator Fm = 2*Fo
   byte 32*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  013,013              CarAmp, FmInc     Initial Values
   byte  089,047        ;0    CarInc, FmInc     0  –  7
   byte  025,038        ;1    CarInc, FmInc     8  –  15
   byte  000,000        ;2    CarInc, FmInc     16 –  23
   byte  000,000        ;3    CarInc, FmInc     24 –  31
   byte  000,000        ;4    CarInc, FmInc     32 –  39
   byte  –25,–38        ;5    CarInc, FmInc     40 –  47
   byte  –76,–38        ;6    CarInc, FmInc     48 –  55
   byte  –25,–22        ;7    CarInc, FmInc     56 –  63
* PatchCLR1: Clarinet Tone 1
      byte  LOADTIMBRE
   byte X1              ;Carrier   Fc = 1*Fo
   byte X2              ;Modulator Fm = 2*Fo
   byte 42*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  013,013              CarAmp, FmInc     Initial Values
   byte  089,083        ;0    CarInc, FmInc     0  –  7
   byte  025,–04        ;1    CarInc, FmInc     8  –  15
   byte  –13,–04        ;2    CarInc, FmInc     16 –  23
   byte  –13,–04        ;3    CarInc, FmInc     24 –  31
   byte  –13,–04        ;4    CarInc, FmInc     32 –  39
   byte  000,–25        ;5    CarInc, FmInc     40 –  47
   byte  000,–25        ;6    CarInc, FmInc     48 –  55
   byte  –89,–29        ;7    CarInc, FmInc     56 –  63
* PatchOBO1: Oboe 1
      byte  LOADTIMBRE
   byte X3              ;Carrier   Fc = 1*Fo
   byte X1              ;Modulator Fm = 1*Fo
   byte 20*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  013,099              CarAmp, FmInc     Initial Values
   byte  038,–14        ;0    CarInc, FmInc     0  –  7
   byte  051,–14        ;1    CarInc, FmInc     8  –  15
   byte  025,–14        ;2    CarInc, FmInc     16 –  23
   byte  –13,–14        ;3    CarInc, FmInc     24 –  31
   byte  –25,000        ;4    CarInc, FmInc     32 –  39
   byte  000,000        ;5    CarInc, FmInc     40 –  47
   byte  000,000        ;6    CarInc, FmInc     48 –  55
   byte  –89,–43        ;7    CarInc, FmInc     56 –  63
* PatchHRN1: French Horn 1
      byte  LOADTIMBRE
   byte X0_5            ;Carrier   Fc = 0.5*Fo
   byte X0_5            ;Modulator Fm = 0.5*Fo
   byte 20*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  038,048              CarAmp, FmInc     Initial Values
   byte  064,050        ;0    CarInc, FmInc     0  –  7
   byte  025,–06        ;1    CarInc, FmInc     8  –  15
   byte  000,–06        ;2    CarInc, FmInc     16 –  23
   byte  –25,–06        ;3    CarInc, FmInc     24 –  31
   byte  –25,–06        ;4    CarInc, FmInc     32 –  39
   byte  –25,–25        ;5    CarInc, FmInc     40 –  47
   byte  –25,–25        ;6    CarInc, FmInc     48 –  55
   byte  –25,–22        ;7    CarInc, FmInc     56 –  63
* PatchBSN1: Bassoon 1
      byte  LOADTIMBRE
   byte X2              ;Carrier   Fc = 2*Fo
   byte X0_5            ;Modulator Fm = 0.5*Fo
   byte 10*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  038,076              CarAmp, FmInc     Initial Values
   byte  064,051        ;0    CarInc, FmInc     0  –  7
   byte  025,–06        ;1    CarInc, FmInc     8  –  15
   byte  000,–06        ;2    CarInc, FmInc     16 –  23
   byte  000,–06        ;3    CarInc, FmInc     24 –  31
   byte  000,000        ;4    CarInc, FmInc     32 –  39
   byte  000,000        ;5    CarInc, FmInc     40 –  47
   byte  –64,000        ;6    CarInc, FmInc     48 –  55
   byte  –64,–108       ;7    CarInc, FmInc     56 –  63
* PatchTBA1: Tuba 1
      byte  LOADTIMBRE
   byte X0_5            ;Carrier   Fc = 0.5*Fo
   byte X0_5            ;Modulator Fm = 0.5*Fo
   byte 18*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  089,064              CarAmp, FmInc     Initial Values
   byte  038,064        ;0    CarInc, FmInc     0  –  7
   byte  –03,–25        ;1    CarInc, FmInc     8  –  15
   byte  –03,–25        ;2    CarInc, FmInc     16 –  23
   byte  –03,–25        ;3    CarInc, FmInc     24 –  31
   byte  –03,–25        ;4    CarInc, FmInc     32 –  39
   byte  –03,–25        ;5    CarInc, FmInc     40 –  47
   byte  –57,000        ;6    CarInc, FmInc     48 –  55
   byte  –57,000        ;7    CarInc, FmInc     56 –  63
* PatchTRB1: Trombone 1
      byte  LOADTIMBRE
   byte X0_5            ;Carrier   Fc = 0.5*Fo
   byte X0_5            ;Modulator Fm = 0.5*Fo
   byte 28*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  064,051              CarAmp, FmInc     Initial Values
   byte  064,051        ;0    CarInc, FmInc     0  –  7
   byte  000,–03        ;1    CarInc, FmInc     8  –  15
   byte  –13,–03        ;2    CarInc, FmInc     16 –  23
   byte  –13,–03        ;3    CarInc, FmInc     24 –  31
   byte  –13,–05        ;4    CarInc, FmInc     32 –  39
   byte  000,–29        ;5    CarInc, FmInc     40 –  47
   byte  000,–29        ;6    CarInc, FmInc     48 –  55
   byte  –89,–30        ;7    CarInc, FmInc     56 –  63
* PatchTPT1: Trumpet 1
      byte  LOADTIMBRE
   byte X1              ;Carrier   Fc = 1*Fo
   byte X1              ;Modulator Fm = 1*Fo
   byte 28*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  064,051              CarAmp, FmInc     Initial Values
   byte  064,051        ;0    CarInc, FmInc     0  –  7
   byte  000,–06        ;1    CarInc, FmInc     8  –  15
   byte  –13,–06        ;2    CarInc, FmInc     16 –  23
   byte  –13,–06        ;3    CarInc, FmInc     24 –  31
   byte  –13,–06        ;4    CarInc, FmInc     32 –  39
   byte  000,000        ;5    CarInc, FmInc     40 –  47
   byte  000,000        ;6    CarInc, FmInc     48 –  55
   byte  –89,–76        ;7    CarInc, FmInc     56 –  63
```

```
* PatchVLN1: Violin 1
      byte  LOADTIMBRE
   byte X2              ;Carrier   Fc = 2*Fo
   byte X1              ;Modulator Fm = 1*Fo
   byte 40*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  013,114              CarAmp, FmInc     Initial Values
   byte  038,000        ;0    CarInc, FmInc     0  –  7
   byte  038,000        ;1    CarInc, FmInc     8  –  15
   byte  038,000        ;2    CarInc, FmInc     16 –  23
   byte  000,000        ;3    CarInc, FmInc     24 –  31
   byte  000,000        ;4    CarInc, FmInc     32 –  39
   byte  000,000        ;5    CarInc, FmInc     40 –  47
   byte  000,000        ;6    CarInc, FmInc     48 –  55
   byte –127,–114       ;7    CarInc, FmInc     56 –  63
* PatchEBS1: Electric Bass 1
      byte  LOADTIMBRE
   byte X0_5            ;Carrier   Fc = 0.5*Fo
   byte X0_5            ;Modulator Fm = 0.5*Fo
   byte 28*4            ;MIX Scaler (higher#=more timbre chg.
                           per progressive Velocity & MIXn )
   byte  127,114              CarAmp, FmInc     Initial Values
   byte  –13,013        ;0    CarInc, FmInc     0  –  7
   byte  –13,–13        ;1    CarInc, FmInc     8  –  15
   byte  –13,–13        ;2    CarInc, FmInc     16 –  23
   byte  –13,–13        ;3    CarInc, FmInc     24 –  31
   byte  –13,000        ;4    CarInc, FmInc     32 –  39
   byte  000,000        ;5    CarInc, FmInc     40 –  47
   byte  000,000        ;6    CarInc, FmInc     48 –  55
   byte  –64,–89        ;7    CarInc, FmInc     56 –  63
```

Appendix F

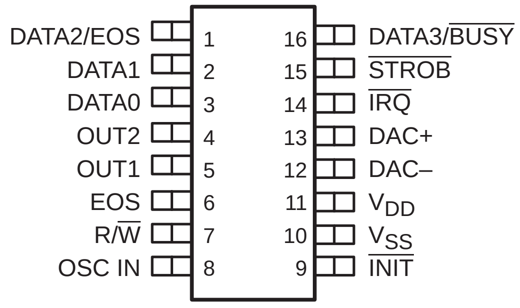
MSP53C391 and MSP53C392 Data Sheet

Topic **Page**

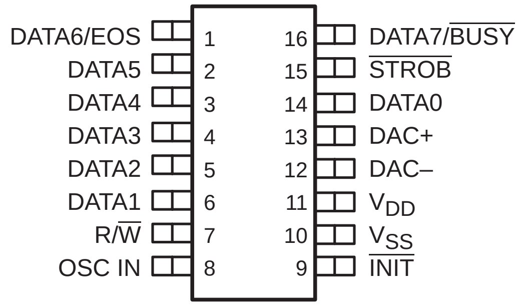
F.1 MSP53C31 and MSP53C32 Data Sheet

- **Slave Speech Synthesizers, LPC, MELP, CELP**
- **Two Channel FM Synthesis, PCM**
- **8-Bit Microprocessor With 61 instructions**
- **3.3V to 6.5V CMOS Technology for Low Power Dissipation**
- **Direct Speaker Drive Capability**
- **Internal Clock Generator That Requires No External Components**
- **Two Software-Selectable Clock Speeds**
- **10-kHz or 8-kHz Speech Sample Rate**

**MSP53C391
N PACKAGE
(TOP VIEW)**



**MSP53C392
N PACKAGE
(TOP VIEW)**



description

The MSP53C391 and MSP53C392 are catalog MSP50C3x codes which implements the functionality of a slave speech synthesizer. They communicate with a master microprocessor using two control lines (R/$\overline{W}$ and $\overline{STROBE}$) and either a 4-bit data bus (MSP53C391) or an 8-bit data bus (MSP53C392).

Either the MSP53C391 or the MSP53C392 can synthesize speech using several different compression algorithms; LPC, MELP, or CELP. They also can synthesize two-channel music using FM synthesis.

See the MSP50C3x User's Guide (literature number: SLOU006B) for more information about the MSP50C3x family.

Table 1. MSP53C39x Family

DEVICE	FEATURES
MSP53C391	4-bit data bus
MSP53C392	8-bit data bus

TEXAS INSTRUMENTS

absolute maximum ratings over operating free-air temperature range[†]

Supply voltage range, V_{DD} (see Note 1)	–0.3 V to 8 V
Supply current, I_{DD} or I_{SS} (see Note 2)	100 mA
Input voltage range, V_I (see Note 1)	–0.3 V to V_{DD} + 0.3 V
Output voltage range, V_O (see Note 1)	–0.3 V to V_{DD} + 0.3 V
Storage temperature range	–30°C to 125°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. All voltages are with respect to ground.
2. The total supply current includes the current out of all the I/O terminals and DAC terminals as well as the operating current of the device.

recommended operating conditions

			MAX	MAX	UNIT
V_{DD}	Supply voltage[†]		3.3	6.5	V
V_{IH}	High-level input voltage	V_{DD} = 3.3 V	2.5	3.3	V
		V_{DD} = 5 V	3.8	5	
		V_{DD} = 6 V	4.5	6	
V_{IL}	Low-level input voltage	V_{DD} = 3.3 V	0	0.65	V
		V_{DD} = 5 V	0	1	
		V_{DD} = 6 V	0	1.3	
T_A	Operating free-air temperature	Device functionality	0	70	°C
Rspeaker	Minimum speaker impedance	Direct speaker drive using 2 pin push-pull DAC option	32		Ω

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

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{T+}	Positive-going threshold voltage (INIT)	V_{DD} = 3.5 V			2		V
		V_{DD} = 6 V			3.4		
V_{T-}	Negative-going threshold voltage (INIT)	V_{DD} = 3.5 V			1.6		V
		V_{DD} = 6 V			2.3		
V_{hys}	Hysteresis (V_{T+} – V_{T-}) (INIT)	V_{DD} = 3.5 V			0.4		V
		V_{DD} = 6 V			1.1		
I_{lkg}	Input leakage current (except for OSC IN)					2	µA
$I_{standby}$	Standby current ($\overline{\text{INIT}}$ low, SETOFF)					10	µA
I_{DD}[†]	Supply current	V_{DD} = 3.3 V,	V_{OH} = 2.75 V		2.1		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V		3.1		
		V_{DD} = 6 V,	V_{OH} = 5.5 V		4.5		
I_{OH}	High-level output current (DATA0 – DATA7, OUT1, OUT2)	V_{DD} = 3.3 V,	V_{OH} = 2.75 V	–4	–12		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V	–5	–14		
		V_{DD} = 6 V,	V_{OH} = 5.5 V	–6	–15		
		V_{DD} = 3.3 V,	V_{OH} = 2.2 V	–8	–20		mA
		V_{DD} = 5 V,	V_{OH} = 3.33 V	–14	–40		
		V_{DD} = 6 V,	V_{OH} = 4 V	–20	–51		
I_{OL}	Low-level output current (DATA0 – DATA7, OUT1, OUT2)	V_{DD} = 3.3 V,	V_{OL} = 0.5 V	5	9		mA
		V_{DD} = 5 V,	V_{OL} = 0.5 V	5	9		
		V_{DD} = 6 V,	V_{OL} = 0.5 V	5	9		
		V_{DD} = 3.3 V,	V_{OL} = 1.1 V	10	19		mA
		V_{DD} = 5 V,	V_{OL} = 1.67 V	20	29		
		V_{DD} = 6 V,	V_{OL} = 2 V	25	35		
I_{OH}	High-level output current (DAC)	V_{DD} = 3.3 V,	V_{OH} = 2.75 V	–30	–50		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V	–35	–60		
		V_{DD} = 6 V,	V_{OH} = 5.5 V	–40	–65		
		V_{DD} = 3.3 V,	V_{OH} = 2.3 V	–50	–90		mA
		V_{DD} = 5 V,	V_{OH} = 4 V	–90	–140		
		V_{DD} = 6 V,	V_{OH} = 5 V	–100	–150		
I_{OL}	Low-level output current (DAC)	V_{DD} = 3.3 V,	V_{OL} = 0.5 V	50	80		mA
		V_{DD} = 5 V,	V_{OL} = 0.5 V	70	90		
		V_{DD} = 6 V,	V_{OL} = 0.5 V	80	110		
		V_{DD} = 3.3 V,	V_{OL} = 1 V	100	140		mA
		V_{DD} = 5 V,	V_{OL} = 1 V	140			
		V_{DD} = 6 V,	V_{OL} = 1 V	150			
$f_{osc(low)}$	Oscillator frequency[‡]	V_{DD} = 5 V, Target frequency = 15.36 MHz	T_A = 25°C,	14.89	15.36	15.86	MHz
$f_{osc(high)}$	Oscillator frequency[‡]	V_{DD} = 5 V, Target frequency = 19.2 MHz	T_A = 25°C,	18.62	19.2	19.7	MHz

† Operating current assumes all inputs are tied to either V_{SS} or V_{DD} with no input currents due to programmed pullup resistors. The DAC output and other outputs are open circuited.
‡ The frequency of the internal clock has a temperature coefficient of approximately –0.2 %/°C and a V_{DD} coefficient of approximately ±1%/V.

switching characteristics

PARAMETER		TEST CONDITIONS	MIN	NOM	MAX	UNIT
t_r	Rise time, DATA0 – DATA7, DAC	V_{DD} = 3.3 V, C_L = 100 pF, 10% to 90%		50		ns
t_f	Fall time, DATA0– DATA7, DAC	V_{DD} = 3.3 V, C_L = 100 pF, 10% to 90%		50		ns

timing requirements

		MIN	MAX	UNIT
Initialization				
t_{INIT}	$\overline{\text{INIT}}$ pulsed low while the MSP53C39x has power applied (see Figure 1)	1		µs
t_{SETUP}	Delay between rising edge of $\overline{\text{INIT}}$ and device initialization complete	5		ms
Writing (Slave Mode)				
$t_{su1(R/W)}$	Setup time, R/$\overline{\text{W}}$ low before $\overline{\text{STROB}}$ goes low (see Figure 2)	20		ns
$t_{su(d)}$	Setup time, data valid before $\overline{\text{STROB}}$ goes high (see Figure 2)	100		ns
$t_{h1(R/W)}$	Hold time, R/$\overline{\text{W}}$ low after $\overline{\text{STROB}}$ goes high (see Figure 2)	20		ns
$t_{h(d)}$	Hold time, data valid after $\overline{\text{STROB}}$ goes high (see Figure 2)	30		ns
t_w	Pulse duration, $\overline{\text{STROB}}$ low (see Figure 2)	100		ns
t_r	Rise time, $\overline{\text{STROB}}$ (see Figure 2)		50	ns
t_f	Fall time, $\overline{\text{STROB}}$ (see Figure 2)		50	ns
Reading (Slave Mode)				
$t_{su2(R/W)}$	Setup time, R/$\overline{\text{W}}$ before $\overline{\text{STROB}}$ goes low (see Figure 3)	20		ns
$t_{h2(R/W)}$	Hold time, R/$\overline{\text{W}}$ after $\overline{\text{STROB}}$ goes high (see Figure 3)	20		ns
t_{dis}	Output disable time, data valid after $\overline{\text{STROB}}$ goes high (see Figure 3)	0	30	ns
t_w	Pulse duration, $\overline{\text{STROB}}$ low (see Figure 3)	100		ns
t_r	Rise time, $\overline{\text{STROB}}$ (see Figure 3)		50	ns
t_f	Fall time, $\overline{\text{STROB}}$ (see Figure 3)		50	ns
t_d	Delay time for $\overline{\text{STROB}}$ low to data valid (see Figure 3)		50	ns

PARAMETER MEASUREMENT INFORMATION

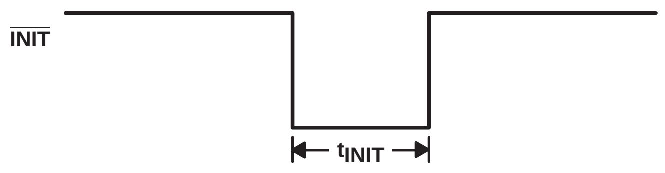


Figure 1. Initialization Timing Diagram

PARAMETER MEASUREMENT INFORMATION

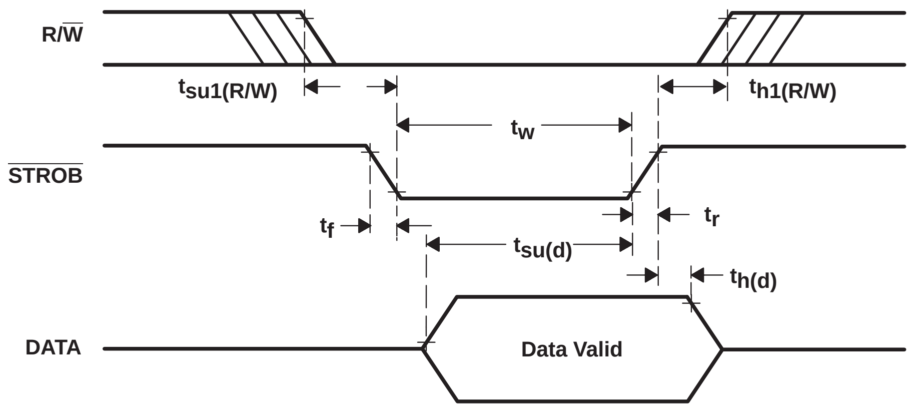


Figure 2. Write Timing Diagram (Slave Mode)

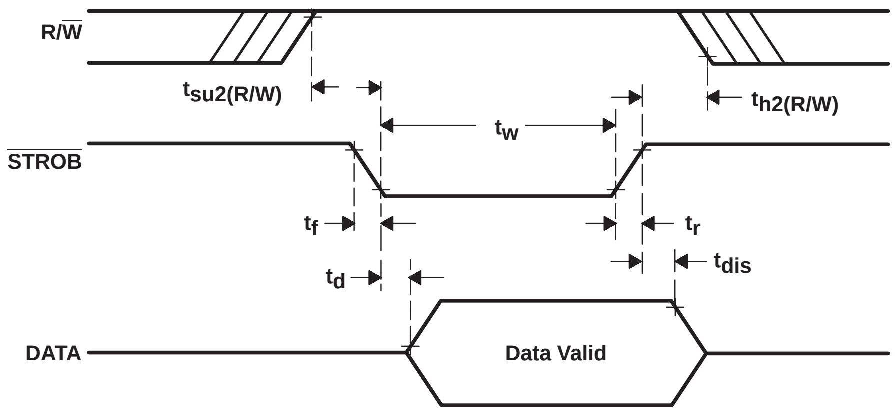


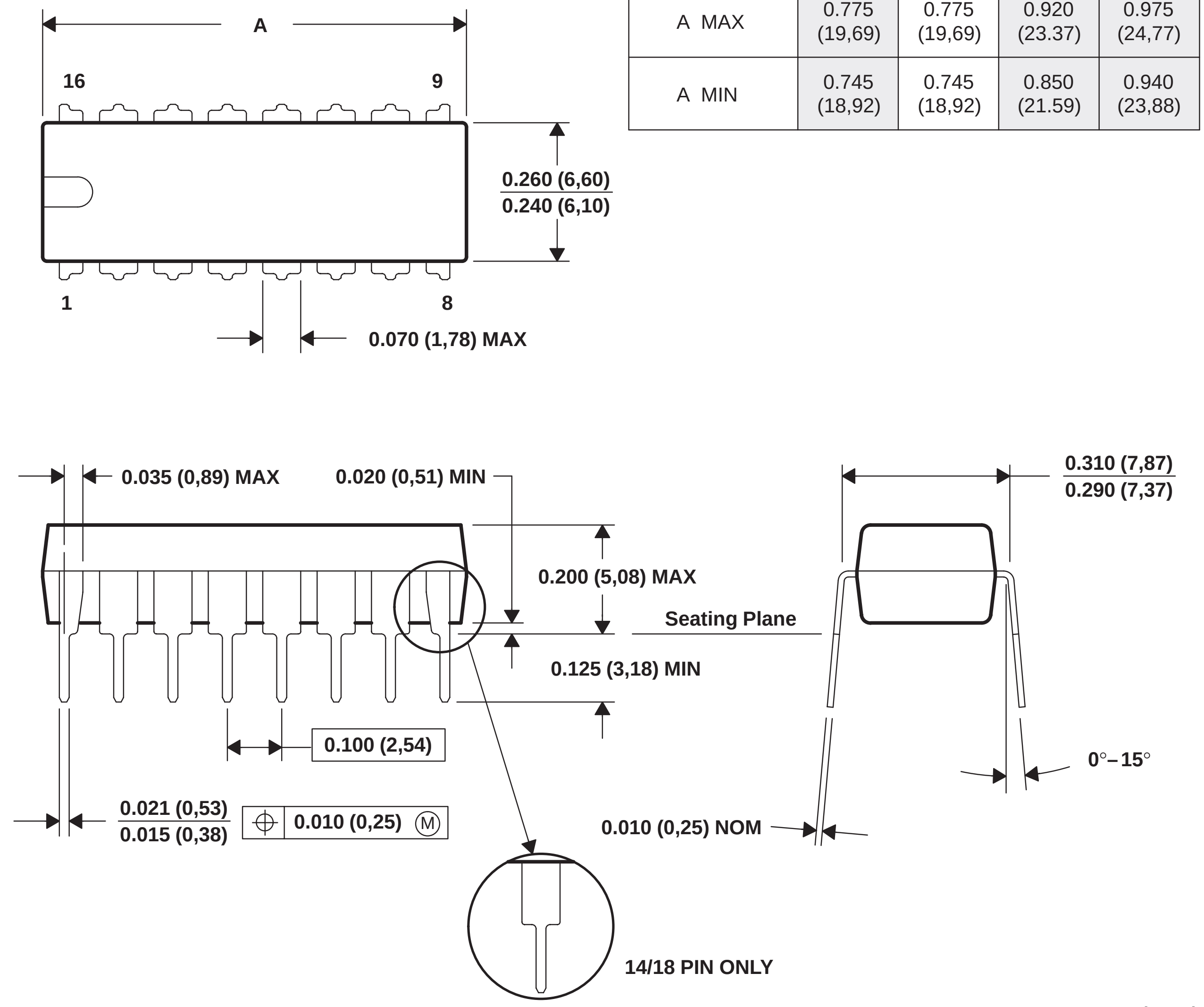
Figure 3. Read Timing Diagram (Slave Mode)

MECHANICAL DATA

N (R-PDIP-T)** **PLASTIC DUAL-IN-LINE PACKAGE**

16 PIN SHOWN

DIM \ PINS **	14	16	18	20
A MAX	0.775 (19,69)	0.775 (19,69)	0.920 (23.37)	0.975 (24,77)
A MIN	0.745 (18,92)	0.745 (18,92)	0.850 (21.59)	0.940 (23,88)



4040049/C 08/95

NOTES: A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-001 (20 pin package is shorter then MS-001.)

Index