TEXAS INSTRUMENTS

MSP50C30

Mixed-Signal Processor

User's Guide

2000 *Mixed-Signal Products*

SPSU012A

IMPORTANT NOTICE

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

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

Customers are responsible for their applications using TI components.

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

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

Preface

Read This First

About This Manual

This manual describes the MSP50C30 speech synthesizing device. The object of this user's guide is to provide the information needed to implement a speech synthesizer design using a MSP50C30 device.

How to Use This Manual

This document contains the following chapters:

Chapter 1 **Introduction to the MSP50C30**
This chapter describes the MSP50C30 features, D/A options, pin assignments and descriptions, and gives a brief introduction to linear predictive coding.

Chapter 2 **MSP50C30 Device Architecture**
This chapter describes the architecture of the MSP50C30 with a separate sections for speech synthesis, interrupts, power control, initialization, and clocks.

Chapter 3 **MSP50C30 Assembler**
This chapter contains a detailed description of the MSP50C30 assembler.

Chapter 4 **MSP50x3x Instruction Set**
This chapter provides the instruction set for the MSP50C30.

Chapter 5 **Applications**
This chapter describes various hints and useful advice for designing applications for the MSP50C30.

Chapter 6 **Customer Information**
This chapter describes customer information including development cycles structure, speech development/production sequence, mechanical information, and ordering information.

Appendix A **Script Preparation and Speech Development Tools**
This appendix describes script preparation and development tools for the MSP50C30.

Appendix B **MSP50C3x Versus TSP50C1x**
This appendix contains information about switching from a TSP50C1x family device to a MSP50C3x family device.

Appendix C **Quick Guide to Programming the MSP50C3x**
This appendix contains information about programming the MSP50C3x.

Appendix D **Using the PSA and SSA Routines**
This appendix contains the code for using of PSA and SSA routines when writing code for the MSP50C30.

Appendix E **Using the PSA and SSA With the TSEGA, LBR, CALL, and BRA Instructions**
This appendix contains code that demonstrates the use of the PSA and SSA routines with TSEGA, LBR, CALL, and BRA instructions.

Appendix F **Pseudo-CALL Instruction**
This appendix contains the code for one way of accessing a subroutine from different locations in the PSA address space. It also demonstrates a pseudo–CALL instruction operating in the SSA region.

Appendix G **Calling Divide from PSA**
This appendix contains the code for how to call one routine (divide) from different parts of the PSA space and demonstrates pseudo-CALL in SSA region.

Appendix H **The CSM30003 Catalog Device**
This appendix contains information on the CSM30003 catalog device.

Appendix I **MSP50C3x Family Data Sheet**
This appendix contains the data sheet for the MSP50C3x family of devices. This data sheet lists the absolute maximum ratings, recommended operation conditions, and the electrical characteristics for the MSP50C3x devices.

Notational Conventions

This document uses the following conventions.

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

  Here is a sample program listing:

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

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

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

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

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

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

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

Information About Cautions

This book may contain cautions.

This is an example of a caution statement.

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

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

Trademarks

IBM, PC, PC/XT, PC/AT are trademarks of IBM Corporation.

Contents

Figures

Tables

Notes and Cautions

Chapter 1

Introduction to The MSP50C30

The MSP50C30 is based off of the MSP50x3x family with some significant differences. The major difference is that the MSP50C30 is able to utilize an external ROM (up to 8 MB) for program and data storage.

Topic **Page**

1.1 MSP50C30 Device

The MSP50C30 device is part of the MSP50x3x family of speech synthesizer devices with some added features and capabilities.

Table 1–1. MSP50x3x Device Family

Device	Amount of ROM/PROM	Features
MSP50C30	4K bytes mask ROM, up to 8M bytes external ROM	28 I/O lines, 23 address lines, 8 data lines from external ROM
MSP50C32	16K bytes mask ROM	9/10 I/O lines
MSP50C33	32K bytes mask ROM	9/10 I/O lines
MSP50C34	64K bytes mask ROM	9/10 I/O lines in package, 24 I/O lines in die form
MSP50P34	64K bytes PROM	9/10 I/O lines
MSP50C37	16K bytes mask ROM	18 I/O lines, A/D converter/analog amplifier
MSP50P37	16K bytes PROM	18 I/O lines, A/D converter/analog amplifier

1.2 Applications

The MSP50C30, like the rest of the MSP50x3x family, is flexible and programmable, making it suitable for a wide variety of applications. Its even lower system cost opens up new applications for solid-state speech. These include:

- Learning aids
- Toys
- Games
- Navigation systems
- Fitness equipment
- Voice mailboxes
- Talking books
- Equipment for the handicapped

1.3 Description

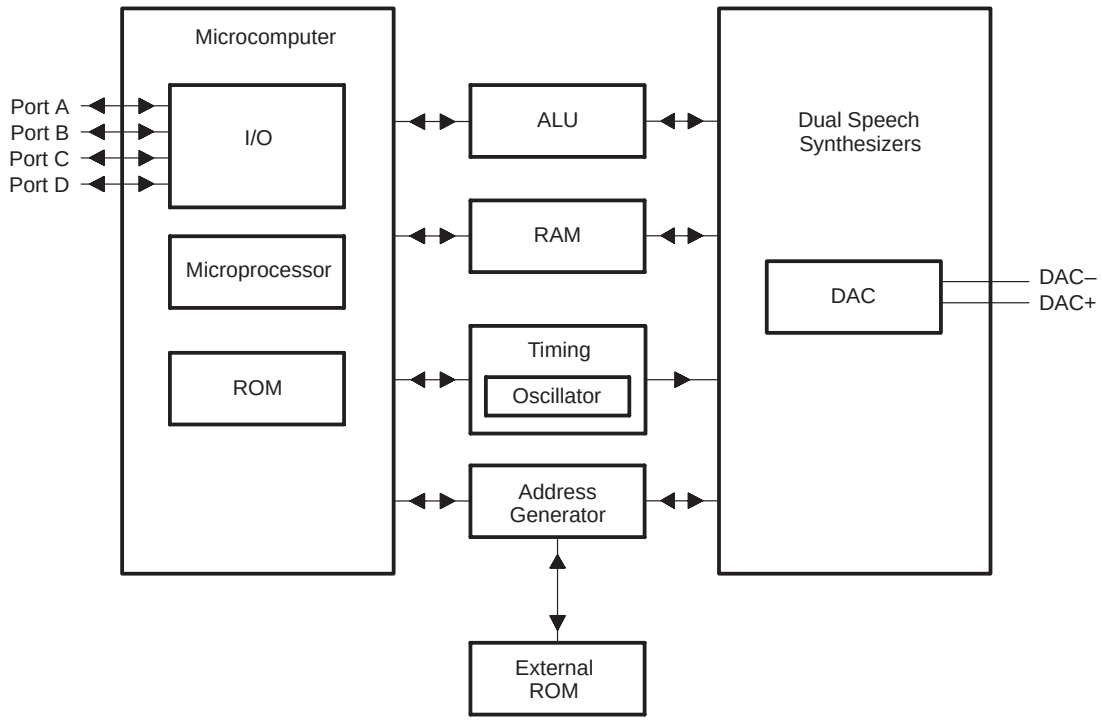
The MSP50C30 can be divided into several functional blocks (see Figure 1–1). The ALU and RAM are shared by the two speech synthesizers and the microcomputer.

The MSP50C30 implements an LPC-12 speech-synthesis algorithm using two 12-pole lattice filters. The internal microprocessor fetches speech data from the internal or external ROM, decodes the speech data, and sends the decoded data to the synthesizer. The microprocessor also interpolates (smoothes) the speech data between fetches. The microprocessor can calculate a PCM waveform which can be added to the output of one of the two lattice filters to create composite PCM+LPC waveforms.

The general-purpose microprocessor in the MSP50C30 is also capable of a variety of logical, arithmetic, and control functions and can be used for the non-synthesis tasks of the application as well.

The MSP50C30 incorporating a built-in oscillator and an external ROM address generator, has the capability to directly drive a 32-Ω speaker.

Figure 1–1. MSP50C30 Functional Block Diagram



1.4 Features

The MSP50C30 contains all of the same key features of the MSP50x3x family plus a few of its own.

- ❑ External ROM interface (up to 8M bytes)
- ❑ 32 12-bit words and 992 bytes of RAM (total 1K bytes RAM)
- ❑ 28 I/O Terminals

1.5 D/A Options

The MSP50C30 device offers two D/A (digital-to-analog) output options to match different applications just as the MSP50x3x family does. As shown in Table 1–2, Option 1 can directly drive a 32-Ω speaker.

Option 1 is the two-pin push pull option and option 2 is the single-pin double-ended option. Please refer to the MSP50x3x manual for more information on the D/A Options for the MSP50C30.

Table 1–2. MSP50C30 D/A Options

D/A Options Available	Speaker Drive
Option 1	32-Ω direct drive
Option 2	Drives an operational amplifier

1.6 Terminal Assignments and Signal Descriptions

The MSP50C30 is currently available in a 100-pin quad flatpack and in die form. Table 1–3 provides terminal function descriptions. Figure 1–2 shows the pinout of the MSP50C30 in the PJM package. Table 1–4 gives the pad locations on the MSP50C30 in die form.

Table 1–3. MSP50C30 100-Pin Package Terminal Functions

Terminal Name	Terminal Number	I/O	Signal Description
DAC+	67	O	D/A output. When D/A Option 1 is selected, DAC+ pulses high for positive output values. It remains low when negative values are output.
DAC–	68	O	D/A output. When D/A Options 1 is selected, DAC– pulses high for negative output values. It remains low when positive values are output. When D/A Option 2 is selected, this terminal is driven low.
ID7 – ID0	14 – 21	I	Data input
$\overline{\text{INIT}}$	22	I	Initialize input. When $\overline{\text{INIT}}$ goes low, the clock stops, the MSP50C30 goes into low-power mode, the program counter is set to 0, and the contents of the RAM are retained. An $\overline{\text{INIT}}$ pulse of 1 µs is sufficient to reset the processor.
OSC IN	49	I	Clock input. When not in use, OSC IN should be tied to V_{SS}.
OSC OUT	57	–	Clock return. When the internal clock option is selected, This terminal is the B1 I/O terminal.
OA0–OA22	74, 83–98, 7–12	O	23-bit output address
PA0–PA7	24, 32–37, 40	I/O	8-bit bidirectional I/O port
PB0–PB7	41–48	I/O	2-bit bidirectional I/O port. When the external clock option is selected, B1 is not available, since terminal 7 is used for the OSC OUT function.
PC0–PC7	58–65	I/O	8-bit bidirectional I/O port
PD0–PD3	70–73	I/O	4-bit bidirectional I/O port
V_{DD}	23, 69	–	5-V supply voltage
V_{SS}	13, 66	–	Ground

Figure 1–2. MSP50C30 Pinout

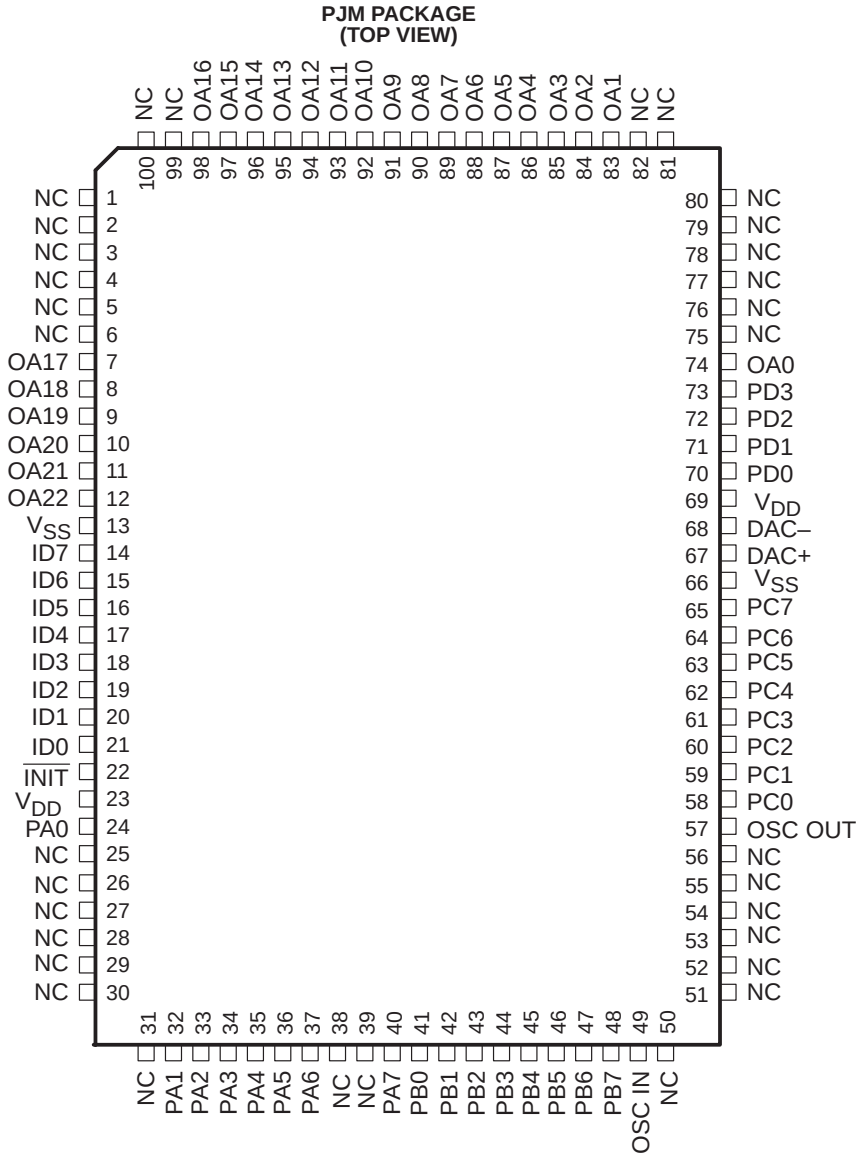


Table 1–4. Pad Locations on the MSP50C30 Die Form

Pad Name	Pad Pinout‡	Lower Left† X (microns)	Lower Left† Y (microns)	Upper Right† X (microns)	Upper Right† Y (microns)	Description
OA_18	1	123.00	4799.00	273.00	4949.00	Output Address, terminal 18
OA_19	2	120.75	4516.75	270.75	4666.75	Output Address, terminal 19
OA_20	3	120.75	4233.00	270.75	4383.00	Output Address, terminal 20
OA_21	4	120.75	3950.75	270.75	4100.75	Output Address, terminal 21
OA_22	5	120.75	3667.00	270.75	3817.00	Output Address, terminal 22
V_{SS}	6	49.00	3384.50	199.00	3534.50	Ground
ID_7	7	120.75	3105.00	270.75	3255.00	Input Data, terminal 7
ID_6	8	120.75	2822.00	270.75	2972.00	Input Data, terminal 6
ID_5	9	120.75	2539.00	270.75	2689.00	Input Data, terminal 5
ID_4	10	120.75	2256.00	270.75	2406.00	Input Data, terminal 4
ID_3	11	120.75	1973.00	270.75	2123.00	Input Data, terminal 3
ID_2	12	120.75	1690.00	270.75	1840.00	Input Data, terminal 2
ID_1	13	120.75	1407.00	270.75	1557.00	Input Data, terminal 1
ID_0	14	120.75	1124.00	270.75	1274.00	Input Data, terminal 0
$\overline{\text{INIT}}$	15	148.75	837.25	298.75	987.25	Initialization
V_{DD}	16	185.25	554.25	335.25	704.25	5-V Supply Voltage
PA0	17	188.25	120.75	338.25	270.75	A Port I/O, terminal 0
PA1	18	471.25	120.75	621.25	270.75	A Port I/O, terminal 1
PA2	19	754.25	120.75	904.25	270.75	A Port I/O, terminal 2
PA3	20	1037.25	120.75	1187.25	270.75	A Port I/O, terminal 3
PA4	21	1320.25	120.75	1470.25	270.75	A Port I/O, terminal 4
PA5	22	1603.25	120.75	1753.25	270.75	A Port I/O, terminal 5
PA6	23	1886.25	120.75	2036.25	270.75	A Port I/O, terminal 6
PA7	24	2169.25	120.75	2319.25	270.75	A Port I/O, terminal 7
PB0	25	2452.25	120.75	2602.25	270.75	B Port I/O, terminal 0
PB1	26	2735.25	120.75	2885.25	270.75	B Port I/O, terminal 1
PB2	27	3018.25	120.75	3168.25	270.75	B Port I/O, terminal 2
PB3	28	3301.25	120.75	3451.25	270.75	B Port I/O, terminal 3
PB4	29	3584.25	120.75	3734.25	270.75	B Port I/O, terminal 4

† Coordinates are in respect to a specific corner of the device.
‡ Pad pinout does not correspond to package pin number.

Table 1–4. Pad Locations on the MSP50C30 Die Form (Continued)

Pad Name	Pad Pinout‡	Lower Left† X (microns)	Lower Left† Y (microns)	Upper Right† X (microns)	Upper Right† Y (microns)	Description
PB5	30	3867.25	120.75	4017.25	270.75	B Port I/O, terminal 5
PB6	31	4150.25	120.75	4300.25	270.75	B Port I/O, terminal 6
PB7	32	4433.25	120.75	4583.25	270.75	B Port I/O, terminal 7
OSC_IN	33	4716.25	120.75	4866.25	270.75	Clock Input
OSC_OUT	34	5009.25	120.75	5159.25	270.75	Clock Output
PC0	35	5128.00	555.00	5278.00	705.00	C Port I/O, terminal 0
PC1	36	5128.00	838.00	5278.00	988.00	C Port I/O, terminal 1
PC2	37	5128.00	1121.00	5278.00	1271.00	C Port I/O, terminal 2
PC3	38	5128.00	1404.00	5278.00	1554.00	C Port I/O, terminal 3
PC4	39	5128.00	1687.00	5278.00	1837.00	C Port I/O, terminal 4
PC5	40	5128.00	1970.00	5278.00	2120.00	C Port I/O, terminal 5
PC6	41	5128.00	2253.00	5278.00	2403.00	C Port I/O, terminal 6
PC7	42	5128.00	2536.00	5278.00	2686.00	C Port I/O, terminal 7
VSS	43	5199.50	2817.75	5349.50	2967.75	Ground
DA–1	44	5126.25	3101.50	5276.25	3251.50	D/A Output
DA–2	45	5126.25	3384.50	5276.25	3534.50	D/A Output
V_{DD}	46	5063.50	3667.25	5213.50	3817.25	5-V Supply Voltage
PD0	47	5128.00	3951.00	5278.00	4101.00	D Port I/O, terminal 0
PD1	48	5128.00	4234.00	5278.00	4384.00	D Port I/O, terminal 1
PD2	49	5128.00	4517.00	5278.00	4667.00	D Port I/O, terminal 2
PD3	50	5128.00	4800.00	5278.00	4950.00	D Port I/O, terminal 3
OA_0	51	5037.50	5235.00	5187.50	5385.00	Output Address, terminal 0
OA_1	52	4753.75	5235.00	4903.75	5385.00	Output Address, terminal 1
OA_2	53	4471.50	5235.00	4621.50	5385.00	Output Address, terminal 2
OA_3	54	4187.75	5235.00	4337.75	5385.00	Output Address, terminal 3
OA_4	55	3905.50	5235.00	4055.50	5385.00	Output Address, terminal 4
OA_5	56	3621.75	5235.00	3771.75	5385.00	Output Address, terminal 5
OA_6	57	3339.50	5235.00	3489.50	5385.00	Output Address, terminal 6
OA_7	58	3055.75	5235.00	3205.75	5385.00	Output Address, terminal 7
OA_8	59	2773.50	5235.00	2923.50	5385.00	Output Address, terminal 8

† Coordinates are in respect to a specific corner of the device.
‡ Pad pinout does not correspond to package pin number.

Table 1–4. Pad Locations on the MSP50C30 Die Form (Continued)

Pad Name	Pad Pinout‡	Lower Left† X (microns)	Lower Left† Y (microns)	Upper Right† X (microns)	Upper Right† Y (microns)	Description
OA_9	60	2489.75	5235.00	2639.75	5385.00	Output Address, terminal 9
OA_10	61	2207.50	5235.00	2357.50	5385.00	Output Address, terminal 10
OA_11	62	1923.75	5235.00	2073.75	5385.00	Output Address, terminal 11
OA_12	63	1641.50	5235.00	1791.50	5385.00	Output Address, terminal 12
OA_13	64	1357.75	5235.00	1507.75	5385.00	Output Address, terminal 13
OA_14	65	1075.50	5235.00	1225.50	5385.00	Output Address, terminal 14
OA_15	66	791.75	5235.00	941.75	5385.00	Output Address, terminal 15
OA_16	67	509.50	5235.00	659.50	5385.00	Output Address, terminal 16
OA_17	68	225.75	5235.00	375.75	5385.00	Output Address, terminal 17

† Coordinates are in respect to a specific corner of the device.
‡ Pad pinout does not correspond to package pin number.

1.7 MSP50C30 Mask Options

The MSP50C30 can be configured to suit different applications with a variety of mask options.

1.7.1 Clock Select Option

The MSP50C30 has two mask-selectable clock options: an internal oscillator option and an external oscillator option.

The internal oscillator is recommended when the lowest-cost solution is required and the absolute accuracy of the oscillator is a secondary consideration. The internal clock is trimmed at probe to standard frequencies of 15.36 MHz and 19.2 MHz. The frequency of the internal clock can be switched between these two values in the software by setting and clearing the SPEED bit in Mode Register 2.

The external oscillator mask option is recommended when an accurate frequency standard is important. Either a ceramic resonator or quartz crystal can be connected between the OSC IN and OSC OUT lines with appropriate capacitors to provide the desired frequency clock. Alternatively, the OSC IN terminal may be driven with an externally derived clock signal. When the external oscillator option is used, the SPEED bit in Mode Register 2 has no function. The SETOFF instruction and $\overline{\text{INIT}}$ terminal disable the operation of the clock circuit.

Refer to the MSP50x3x Manual for the figures.

1.7.2 DAC Option

The DAC for the MSP50C30 can be selected as either a two-pin push pull or a one-pin analog. See section 1.5, *D/A Options*, for more information.

Chapter 2

MSP50C30 Device Architecture

This chapter describes the architecture and function of the MSP50C30 device including RAM, ROM, registers, flags, and the DAC.

Topic **Page**

2.1 MSP50C30 Architecture

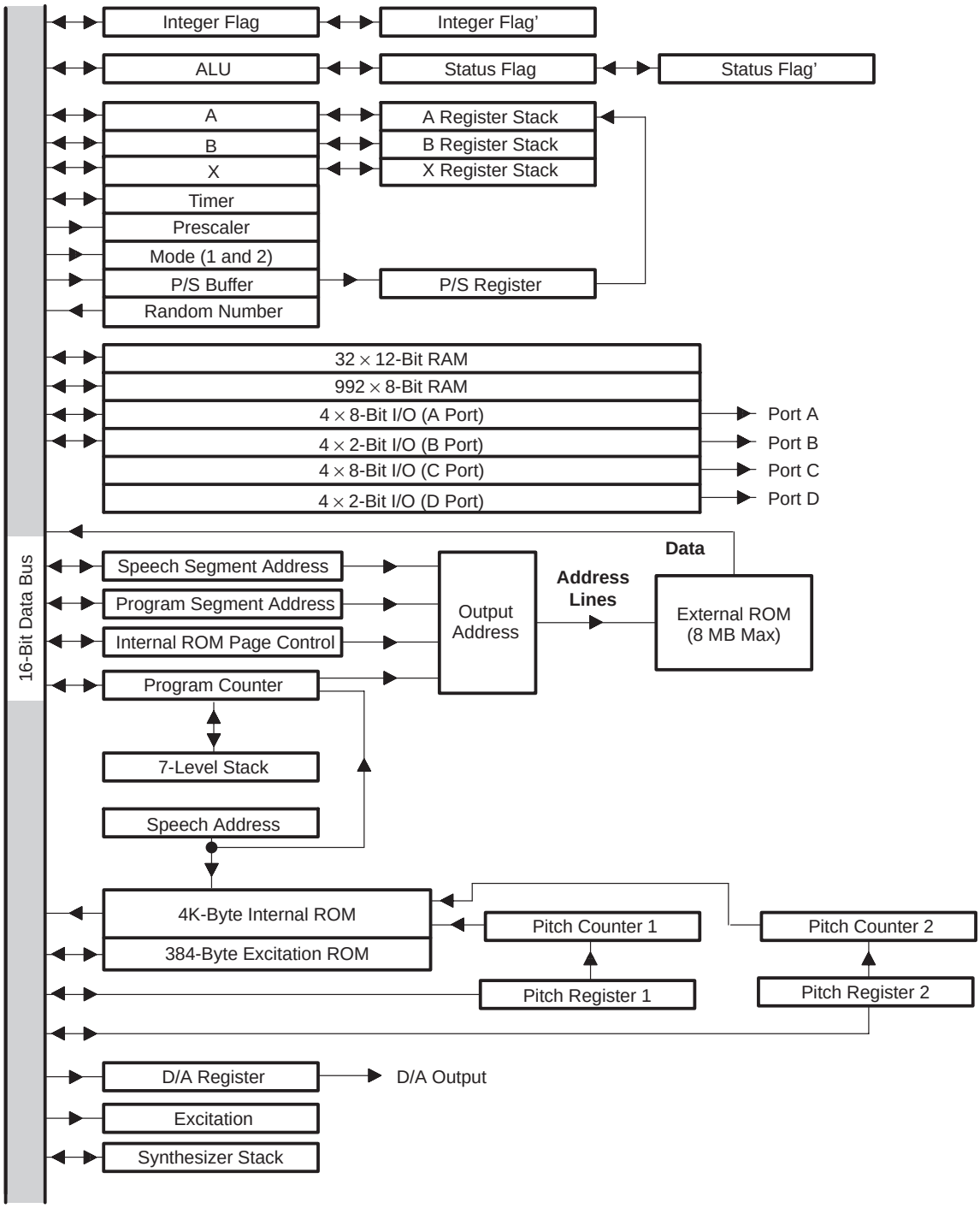
As shown in the block diagram of Figure 2–1, the major components of the MSP50C30 are a speech synthesizer, an 8-bit microprocessor, an internal ROM (4K), and input/output ports.

The system clock can be generated internally or driven externally. When the internal clock is used, the clock operates under software control and can be programmed to one of two frequencies, 19.2 MHz (used when LPC is operating at 10,000 samples per second) or 15.36 MHz (for 8,000 samples per second). See section 1.7 for more information about clock selection.

When LPC synthesis is disabled, instructions are fetched by the microprocessor at 1/16 of the clock frequency. These instructions control the actions of the MSP50C30. By placing different instruction patterns in the ROM, the MSP50C30 can be programmed to accomplish a wide variety of tasks. To generate speech, the processor accesses speech data from either the internal or external ROM. Once the data has been read, the processor must unpack and decode the individual speech parameters and store the results in a dedicated section of the RAM.

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

Figure 2–1. MSP50C30 System Block Diagram



2.1.1 Read-Only Memory (ROM)

The MSP50C30 has 4K bytes of internal ROM and can support up to 8M bytes of external ROM. Certain locations in the internal ROM, described in Table 2–1, are reserved for specific purposes.

Table 2–1. Reserved Internal ROM Locations

Address	Function
0000h	Execution start location after $\overline{\text{INIT}}$ rising edge
0002h	Execution start location after I/O wakeup falling edge
0010h – 001Fh	Interrupt start locations (see Section 2.2, *Interrupts*)
0FDBh – 0FFFh	Texas Instruments test code
1000h – 1200h	Excitation Code

In order to use the first 4K bytes of ROM space on the external ROM device, a one must be written to bit zero of the memory mapped register at FEEh (IRPC). When this is done, the address space between 000000h and 000FFFh is devoted to the lower 4K bytes of the external ROM. When clearing bit zero (default) of the IRPC, the address space between 000000h and 000FFFh is devoted to the internal ROM. The addresses higher than 000FFFh are always devoted to the external ROM, except in the case of the excitation code. See Section 2.1.11 for more information on the IRPC.

The ROM (both internal and external) can be accessed in the following four ways:

- ❑ The program counter addresses processor instructions (see MSP50x3x manual for instruction definitions).
- ❑ The GET instruction transfers 1 to 8 bits from the ROM to the A register. The GET counter is initialized by the LUAPS instruction. The SAR (speech address register), PSA (program segment address), and SSA (speech segment address) point to the location in ROM to be used.
- ❑ The LUAA instruction transfers a byte from the ROM (location dependent on the PSA, SSA, and IRPC) into the A register. The value in the A register when LUAA is executed points to the ROM address to be used.
- ❑ The LUAB instruction transfers a byte from the ROM (location dependent on the PSA, SSA, and IRPC) into the B register. The value in the A register when LUAB is executed points to the ROM address to be used.

Note: External ROM Address

The excitation ROM will be contained within the internal ROM starting at location 1000h.

2.1.2 Program Counter

The MSP50C30 has a 16-bit program counter that points to the next instruction to be executed. After the instruction is executed, the program counter is normally incremented to point to the next instruction.

The MSP50C30 also has a 23-bit composite program counter to access the entire ROM (internal and external). This composite address is created by combining the ROM address provided by the core with one of the two memory mapped registers as follows:

```
if (CA < 8000h)
        OA = CA + (PSA << 12)
    else
        OA = (CA & 7FFFh) + (SSA << 15)
```

where CA is the Core Address, PSA is the Program Segment Address, SSA is the Speech Segment Address, and OA is the Output Address.

Note: Core Address

The core address (CA) is the offset into the ROM space, which could be the program counter, the contents of the pitch period counter, the contents of the speech address register, the results of a lookup instruction, or the excitation ROM offset.

As a result of the above calculation, a segmented address is created using the PSA (ideally used for the program space) with overlapping 32K byte segments with an offset of 4K bytes as follows:

- Segment 0: 0K < address < 32K
- Segment 1: 4K < address < 36K
- Segment 2: 8K < address < 40K
- Segment 3: 12K < address < 44K
- etc.

The segmented address using the SSA (ideally used for the speech data space) with nonoverlapping 32K byte segments are as follows:

- Segment 0: 0K < address < 32K
- Segment 1: 32K < address < 64K
- etc.

See Sections 2.1.9 and 2.1.10 for more information on the program segment address (PSA) and the speech segment address (SSA).

When an interrupt occurs, the program counter is loaded with the interrupt vector address for the current page where execution resumes. See Section 2.2,

Interrupts, for more information. The following instructions modify the program counter.

- ❑ BR Branch
- ❑ BRA Branch to address in A register
- ❑ SBR Short branch
- ❑ RETN Return from subroutine
- ❑ RETI Return from interrupt
- ❑ CALL Subroutine call
- ❑ BR Long branch across segment boundaries (new opcode)

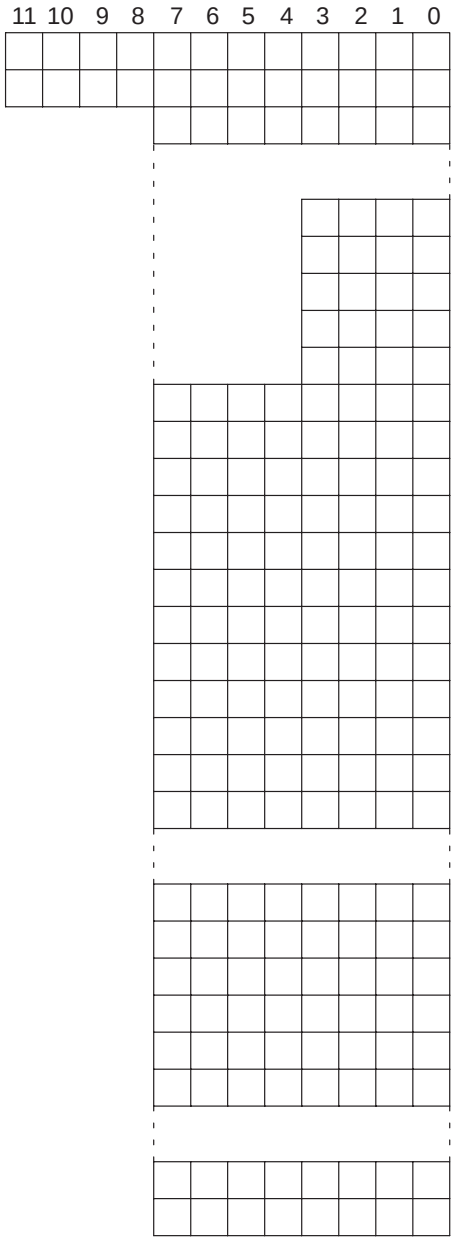
2.1.3 MSP50C30 Random Access Memory (RAM)

The MSP50C30 has 1024 locations of general purpose RAM (Figure 2–2). The first 16 RAM locations, which are 12 bits wide, are used by the first synthesizer when it is enabled. The next 16 RAM locations, which are also 12 bits wide, are used by the second synthesizer when it is enabled. The blocks of RAM associated with a synthesizer is released to general use when the synthesizer is not enabled. The remaining 992 RAM locations are 8 bits wide. When not synthesizing, the entire RAM can be used for algorithm data storage. The I/O control registers are mapped into the RAM address space from 0F0h to 0FFh when the I/O MAP bit in mode register 2 is low. When the I/O MAP bit is high, RAM is mapped into 0F0h to 0FFh. For more information, see subsection 2.1.7, *Input/Output Ports*. The upper 20 RAM locations (FECh – FFFh) are dedicated to memory mapped registers.

Note: Accessing RAM Above FFh

RAM locations 100h – 3FFh can only be accessed indirectly. In other words, TAMD, TMAD, and TMXD may not be used to access these RAM locations.

Figure 2–2. MSP50C30 RAM Map



11 10 9 8 7 6 5 4 3 2 1 0	Address		
	001h – 00Fh	Channel 1 Synthesizer RAM†	
	011h – 01Eh	Channel 2 Synthesizer RAM†	
	020h – 0EFh	General-Purpose RAM†	
		I/O MAP is Low†	I/O MAP is High
	0F0h	Port D DIR	F0 RAM
	0F1h	Port D PER	F1 RAM
	0F2h	Port D DDR	F2 RAM
	0F3h	Port D DOR	F3 RAM
	0F4h	Port C DIR	F4 RAM
	0F5h	Port C PER	F5 RAM
	0F6h	Port C DDR	F6 RAM
	0F7h	Port C DOR	F7 RAM
	0F8h	Port B DIR	F8 RAM
	0F9h	Port B PER	F9 RAM
	0FAh	Port B DDR	FA RAM
	0FBh	Port B DOR	FB RAM
	0FCh	Port A DIR	FC RAM
	0FDh	Port A PER	FD (RAM)
	0FEh	Port A DDR	FE (RAM)
	0FFh	Port A DOR	FF (RAM)
	100h – 3FFh	General-Purpose RAM‡	
	FECh	Program Segment Address Control – PSA	
	FEDh	Speech Segment Address Control – SSA	
	FEEh	Internal ROM page Control – IRPC	
	FEFh	Wake-Up Control Register	
	FE0h – FF3h	Speech Synthesis Registers	
	FF4h – FF7h	Speech Synthesis Registers, Temporary	
	FFEh	Mode Register 1	
	FFFh	Mode Register 2	

† Direct and indirect memory address
‡ Indirect memory address only

2.1.4 MSP50C30 Memory-Mapped Registers

Several internal registers are mapped into the RAM address space. Table 2–2 shows the memory-mapped register allocations for the MSP50C30 device.

Table 2–2. Memory-Mapped Registers

Address	Function	
F0h	Port D Input Register	(see subsection 2.1.7)
F1h	Port D Pullup Enable Register	(see subsection 2.1.7)
F2h	Port D Data Direction Register	(see subsection 2.1.7)
F3h	Port D Data Output Register	(see subsection 2.1.7)
F4h	Port C Input Register	(see subsection 2.1.7)
F5h	Port C Pullup Enable Register	(see subsection 2.1.7)
F6h	Port C Data Direction Register	(see subsection 2.1.7)
F7h	Port C Data Output Register	(see subsection 2.1.7)
F8h	Port B Input Register	(see subsection 2.1.7)
F9h	Port B Pullup Enable Register	(see subsection 2.1.7)
FAh	Port B Data Direction Register	(see subsection 2.1.7)
FBh	Port B Data Output Register	(see subsection 2.1.7)
FCh	Port A Input Register	(see subsection 2.1.7)
FDh	Port A Pullup Enable Register	(see subsection 2.1.7)
FEh	Port A Data Direction Register	(see subsection 2.1.7)
FFh	Port A Data Output Register	(see subsection 2.1.7)
FECh	Program Segment Address Control	
FEDh	Speech Segment Address Control	
FEEh	Internal ROM Page Control	
FEFh	Wake-Up Select Register	
FF0h	Y1S (16 bit)	(synthesizer control registers)
FF1h	PITCH2 (16 bits)	(synthesizer control register)
FF2h	Y1 (16 bits)	(synthesizer control register)
FF3h	PITCH (16 bits)	(synthesizer control register)
FF4h	TEMP (16 bits)	(synthesizer control register)
FF5h	TEMP (16 bits)	(synthesizer control register)
FF6h	TEMP (16 bits)	(synthesizer control register)
FF7h	TEMP (16 bits)	(synthesizer control register)
FFEh	Mode Register 1	(see subsection 2.1.8)
FFFh	Mode Register 2	(see subsection 2.1.8)

Note: Refer to MSP50x3x User's Guide

Please refer to the *MSP50x3x Mixed-Signal Processor User's Guide* (Literature Number SPSU006B) for information on the following topics:

- ❑ Arithmetic logic unit (ALU)
- ❑ A register
- ❑ X register
- ❑ B register
- ❑ Status flag
- ❑ Integer mode flag
- ❑ Timer register
- ❑ Timer prescale register
- ❑ Pitch register and pitch period counter
- ❑ Power control and initialization
- ❑ Clocks
- ❑ Program counter stack

2.1.5 Speech Address Register (SAR)

The speech address register (SAR) is a 16-bit register that points to data in the ROM. The LUAPS instruction transfers the value in the A register to the speech address register and loads the parallel-to-serial register with the ROM value pointed to by the SAR, SSA, PSA, and IRPC. This value is calculated by looking at the values in the SAR, SSA, PSA, and IRPC (see note below). The GET instruction can then bring 1 to 8 bits at a time from the parallel-to-serial register into the accumulator. Whenever the parallel-to-serial register becomes empty, it is loaded with the ROM value pointed to by the SAR, and the SAR is incremented.

Note: Changing Active Page Before a GET

When any of the four registers (SAR, SSA, PSA, and IRPC) are changed, the next fetch from ROM will be affected.

2.1.6 Parallel-to-Serial Register

The 8-bit parallel-to-serial register is used primarily to unpack speech data. It can be loaded with 8 bits of data from the ROM pointed to by the speech address register, SSA, PSA, & IRPC, or the internal RAM pointed to by the X register. The LUAPS instruction initializes the parallel-to-serial register and zeroes its bit counter. GET instructions can then transfer 1 to 8 bits from the parallel-to-serial register to the accumulator. When the parallel-to-serial

register is empty, it is automatically reloaded with the next consecutive address's data. When the GET is from the RAM, however, the X register is not automatically incremented. The RAMROM bit in mode register 1 controls the source of the parallel-to-serial register.

2.1.7 Input / Output Ports

In the MSP50C30 device, 28 bidirectional lines (8-bit Port A, 8-bit Port B, 8-bit Port C, and 4-bit Port D) are available for interfacing with external devices. Each bit is individually programmable as an input or output under the control of the respective data-direction register. In addition, each output bit can be individually programmed using the pullup enable register for one of two output modes—push pull or open-drain (no pullup). Each input bit can be programmed by the same register for resistive pullup or high impedance. The four registers associated with each of the four I/O ports are memory mapped. All 8 bits of ports A, B, and C and 4 bits of port D are available on the outside of the chip.

Note: Transfers From I/O Port Registers

Transfers from any of the I/O port registers to the A register leave the upper 8 bits (bits 15 – 8) undetermined.

I/O Terminal Addressing

It is recommended that the I/O terminals be addressed directly using the TAMD, TMAD, or TAM instructions. Using the ANDCM or ORCM instructions to set or reset bits may produce glitches on other bits.

Table 2–3. I/O Registers

		Location			
Register	Type	Port A	Port B	Port C	Port D
Data Input Register (DIR)	Read Only	FCh	F8h	F4h	F0h
Pullup Enable Register (PER)	Read/Write	FDh	F9h	F5h	F1h
Data Direction Register (DDR)	Read/Write	FEh	FAh	F6h	F2h
Data Output Register (DOR)	Read/Write	FFh	FBh	F7h	F3h

Table 2–4. I/O Terminal Functions

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

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

Unused I/O Terminals

Any unused I/O terminals which are programmed as inputs should be tied high or low. Floating I/O terminals can cause leakage current.

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

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

If the PCM1 and LPC1 mode register bits are both cleared and ENA1 (level 1 interrupt) is set, a high-to-low transition on B1 causes a level-1 interrupt. (The B1 pulse must have minimum width of 1 μs.) This condition can generate an interrupt with an external event.

2.1.8 Mode Registers

There are two 8-bit memory mapped mode registers that control the MSP50C30. The contents of the first mode register are cleared to 0 and the contents of the second mode register are set to 09h upon power-up reset, wake-up, or when the $\overline{\text{INIT}}$ pulses low. The contents of the two mode registers are not saved during a subroutine call or interrupt.

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

2.1.8.1 Mode Register 1

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

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

The LPC1, PCM1, and UNVOICE1 bits control the activity of the first synthesis channel. See Section on *Speech Synthesis* in the MSP50X3X manual.

Note: Be Careful Using the ORCM on an Unvoiced Bit

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

The slave bit enables a special I/O mode designed to enable the MSP50C30 to operate as a slave microprocessor under the control of an external master coprocessor. See the MSP50C3X Manual for more information.

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

Table 2–5. Mode Register 1

Bit	Name	Bit Low	Bit High
0	ENA1	Disables the level-1 interrupt	Enables the level-1 interrupt
1	LPC1	Disables the channel 1 LPC processor – all instruction cycles used by the microprocessor	Enables the channel 1 LPC processor – 25% of instruction cycles dedicated to service this channel. RAM locations 01h – 0Fh are dedicated to the LPC synthesis.
2	PCM1	Disables the PCM mode for channel 1	Enables the PCM mode for channel 1
3	ENA2	Disables the level-2 interrupt	Enables level-2 interrupt
4	Reserved	Leave this bit at 0	Do not set this bit
5	RAMROM	Enables data source for GET instructions to be ROM (internal or external)	Enables the data source for the GET instructions to be internal RAM
6	SLAVE	Enables I/O master operation. All available I/O terminals are controlled by the internal microprocessor	Enables the I/O slave operation. Terminal B0 becomes hardware chip enable strobe, and B1 becomes R/$\overline{W}$, Port A is controlled by B0 and B1
7	UNVOICE1	Enables the pitch-controlled excitation sequence when in LPC mode (PCM1 low, voiced) on channel 1	Enables the random excitation sequence when in LPC mode (PCM1 low, voiced) on channel 1

2.1.8.2 Mode Register 2

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

The LPC2, PCM2, and UNVOICE2 bits control the activity of the second synthesis channel. See section on *Speech Synthesis* in the MSP50X3X manual for more information.

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

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

The SPEED bit controls the speed of the internal oscillator. When this bit is cleared to 0, the internal oscillator generates a 15.36-MHz clock and synthesis operates at 8,000 samples/second. When this bit is set to 1, the internal oscillator generates a 19.2-MHz clock and synthesis operates at a 10,000 samples/second rate. When the external oscillator mask option is in effect, the SPEED bit has no effect.

The I/O_MAP bit controls the RAM address spaces located at F0h through FFh. When this bit is cleared to 0 (the default condition), the I/O ports (ports A, B, C, and D) are mapped into these locations. When this bit is set to 1, the I/O ports are hidden and RAM is mapped into these addresses. The RAM and the ports maintain separate storage locations so that different data can be maintained in the RAM location and the port latch while using the same address.

Table 2–6. Mode Register 2

Bit	Name	Bit Low	Bit High
0	PPC1	Cleared low using software	Set high when PPC decrements below 200h on channel 1 or by program init or wakeup
1	LPC2	Disables LPC mode	Enables LPC mode on channel 2 — 25% of the instruction cycles are dedicated to service this channel. RAM locations 11h – 1Eh are dedicated to LPC synthesis.
2	PCM2	Disables PCM mode	Enables PCM mode on channel 2
3	PPC2	Cleared low using software	Set high when PPC decrements below 200h on channel 2 or by program init or wakeup
4	CHANNEL	Synthesizer channel 1 selected. The TASYN instruction addresses first channel.	Synthesizer channel 2 selected. the TASYN instruction addresses the second channel.
5	SPEED	Selects internal clock speed of 15.36 MHz (8,000 samples/second sample rate)	Selects internal clock speed of 19.2 MHz (10,000 samples/second sample rate)
6	I/O MAP	I/O ports (ports A, B, C, and D) are mapped into 0xF0h through 0xFFh.	I/O ports are hidden and RAM is mapped into 0xF0h through 0xFFh.
7	UNVOICE2	Enables the pitch-controlled excitation sequence when in LPC mode (PCM1 low, voiced) on channel 2	Enables the random excitation sequence when in LPC mode (PCM1 low, voiced) on channel 2

2.1.9 Program Segment Address Register (PSA)

The program segment address register is used to control which page the program should execute from. If the core address is less than 8000h, then the value in the PSA register (left-shifted by 12) is added to the core address for the resulting 23 bit output address. The program space is therefore made up of overlapping 32K byte segments with an offset of 4K bytes. For example, segment 0 is from 0K to 32K, segment 1 is from 4K to 36K, etc.

When the PSA is changed, the effect of the change will be delayed for 2 instruction cycles. This will allow the following code to work correctly:

```
CLA
ACAAC  #FEC
TAX
TCA    NewBlock
TAM
LBR    New Address
```

Please see Chapter 5 for more information on how to use the PSA.

Note:

It is important that the LBR immediately follow the TAM which loads the new value to the PSA.

2.1.10 Speech Segment Address Register (SSA)

The speech segment address register is used to control which page the program should get data from. If the core address is greater than 8000h, then the value in the SSA register (left-shifted by 15) is added to the core address (ANDed with 7FFFh) for the resulting output address. Therefore, the data space is made up of non-overlapping 32K byte segments. For example, segment 0 is from 0K to 32K, segment 1 is from 32K to 64K, etc.

The GET, LUAA, LUAPS, LUAB are all affected by what is in the SSA.

When the SSA is changed, the effect is immediate. Please see Chapter 5 for more information on how to use the SSA.

2.1.11 Internal ROM Page Control Register (IRPC)

The internal ROM page control register, located at RAM location FEEh, is used to designate where the lower 4K of the address range will operate from. Since the internal ROM can have code in its lower 4K and the external ROM may also have code in its lower 4K, the MSP50C30 must know which of the two to

operate from. Writing a one to the IRPC will allow execution from the lower 4K of the external ROM. Writing a zero to the IRPC will allow execution from the lower 4K of the internal ROM. The IRPC is cleared upon $\overline{\text{INIT}}$. When the IRPC is changed, the effect is immediate. See Chapter 5 for more information on how to use the IRPC.

2.2 Interrupts

The MSP50C30 has two interrupts: the level-1 interrupt and the level-2 interrupt. Both are enabled and disabled by bits in mode register 1. The level-1 interrupt has higher priority and has more hardware support. When a level-1 interrupt occurs, the program counter is placed on the program counter stack, and the status flag, integer mode flag, A register, B register, and X register are all saved in dedicated storage registers. Then the program counter is loaded with the interrupt start location for the current page of execution (current page determined by value in PSA or SSA) and execution of the interrupt routine begins. For example, if an interrupt happens when execution is on page 2, then the interrupt start location is 2000h + normal interrupt vector address. When the interrupt routine returns, all of these registers are restored, and the program counter is popped from the stack.

See the MSP50x3x manual for more information on interrupts.

Chapter 3

MSP50C30 Assembler

This chapter describes the differences between the ASMX (MSP50C3X Assembler) and the new ASM30 (MSP50C30 Assembler).

Topic **Page**

3.1 MSP50C30 Assembler

The ASM30 assembler is used specifically for MSP50C30 development. It is similar to the ASMX assembler used for the 50C3X family with the exceptions discussed in the following sections.

3.1.1 TABSIZE Directive

The TABSIZE directive used in the ASMX assembler is not supported in this version of the assembler.

3.1.2 COPY Directive

The filename used with the COPY directive should not contain single quotes. Example: COPY a.byt

3.1.3 Arithmetic Expressions

Arithmetic expressions should contain no spaces between arguments and arithmetic operators.

3.1.4 Placing Binary Data Above #FFFF

The ASM30 assembler does allow values larger than #FFFF in an AORG statement. The following is allowed with ASM30: AORG #20000. (ASMX only allows up to #FFFF to be used with the AORG.)

3.1.5 EXTRNL Directive

The directive EXTRNL has been added to support internal and external code within the same source file. When using this directive, the 50C30 assembler (ASM30) will produce two output binary files, one for the internal code and one for the external code. The default mode is internal at the start of the source file and the resulting assembled binary data will be placed into a file with the .BIN extension. When the (new) directive EXTRNL is encountered, the mode is switched to external, the .BIN file is closed, the location counter is reset to zero, and the remaining binary output data is placed into a file with a .EXT extension. A single symbol table is used for both modes, so branches between internal and external codes are supported. Symbols must be unique across both codes.

3.1.6 Support for Code Segments

Three new opcodes have been added to the assembler to support code segments. These opcodes work with the PSA segments, not with the SSA segments.

3.1.6.1 SEGMNT Directive

The SEGMNT directive partitions the code into segments. When the SEGMNT directive is encountered, an implied AORG takes place at the start of the next 12-bit page. BR, CALL, and SBR commands cannot cross a segment boundary. This directive is not meant to be used to partition data segments, since data segments start on every 15-bit page and code segments start on every 12-bit page. The AORG directive may still be used to separate code segments also.

3.1.6.2 TSEGA Command

The TSEGA command will extract the 8-bit program segment value from the operand and load it into the A register (see example below). The purpose of this is to extract the value to load to the PSA and to facilitate the LBR command.

3.1.6.3 LBR Command

The LBR command allows a long branch between code segments. The following is an example of an LBR command. The only significant difference between the BR and LBR commands is that the BR command does not permit branches across segment boundaries and the LBR does.

```
PSA   equ      #FEC  ;define program segment address
                      register

      extrnl         ;switch mode to generate external code
      cla
      acaac    PSA
      tax            ;point X register to PSA (#FEC)
      tsega    abc   ;load the 8 bit segment value to A
      tam            ;store into the PSA
      lbr      abc   ;branch across pages to location

      segmnt         ;set beginning of next page
.
abc   .
```

Refer to the MSP50x3x Manual for more information on the ASMX assembler.

Note:

It is important that the PSA register be changed immediately before the LBR instruction with no intervening instructions. The effect of the change in the PSA register is delayed exactly two instruction cycles to permit time to execute the LBR branch.

Chapter 4

MSP50x3x Instruction Set

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

Topic **Page**

4.1 Instruction Format

The source code instruction format is:

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

The fields are:

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

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

4.2 MSP50C30 Assembly Instructions

The following section contains descriptions, opcodes, hex opcodes, and status flag information for the assembly instructions used to program the MSP5C30. Table 4–1 lists the assembly instructions in alphabetical order with operand size in bits, instruction cycles required, status conditions, number of bytes required, opcode, and a description.

Table 4–1. MSP50C30 Instruction Set

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

Table 4–1. MSP50C30 Instruction Set (Continued)

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

Table 4–1. MSP50C30 Instruction Set (Continued)

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

Table 4–2 lists the instructions by opcode.

Table 4–2. MSP50C30 Instruction Table

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

Please refer to the MSP50x3x User's Guide (SPSU006) for a detailed description of each instruction.

Chapter 5

Applications

This chapter contains information on using an external ROM with the MSP50C30. Code examples are given throughout this chapter.

Topic **Page**

5.1 Using an External ROM With the MSP50C30

The MSP50C30 contains 3 memory mapped registers, PSA, SSA, and IRPC that provide easy access to an external ROM. In order to understand these three registers, we will give a little background behind them. The purpose of the IRPC register is to provide access to the lower 4K of both the internal ROM and the external ROM. If this register did not exist, then the bottom 4K of the external ROM would not be accessible. The PSA register allows for overlapping 4K segments which provide the user with a way to keep certain subroutines called within one segment. This register is optimized for code access, but it can certainly be used to access both code and data. The SSA register is optimized to access data which is far removed from the current code segment. As with the PSA, there is no reason this register cannot also be used to access code as well as data. Together these two registers allow code and data of up to 64K to be accessed without having to change segments.

5.1.1 Using the PSA to Access an External ROM

The following is an example of using the PSA, accessing page 2, and executing from that page.

```
a     cla
      acaac    #fec
      tax
      tsega    a1
      tam
      lbr      a1
   .
   .
      segmnt                ;start of segment 1
   .
   .
      segmnt                ;start of segment 2
a1    cla
      acaac    (table&#FFF) ;must mask off upper 4 bits
      luaa                  ;load #34 into A
      luab                  ;load #12 into B
   .
   .
      aorg     #2410
table byte     #12,#34,#56
      byte     #78,#9A,#BC
```

When the PSA is loaded with a value, that segment becomes current and all addresses used (below #8000) are offset from that segment start. For example, in the above code, the long branch could have been made to 0 instead of a1 because the segment was set to 2, all addresses are offset starting at #2000.

Caution must be exercised when using LUAA, LUAB, and LUAPS. These instructions look at the core address and decide whether to use the PSA or the SSA. When looking up from tables, the address must be taken into account. If the table is at address #2410 and the PSA is set to 2, then only the lower 12 bits need to be loaded to read from the table because the table is offset #410 from the beginning of segment 2. If the upper 4 bits are not masked off, the assembler will generate a warning.

The following code shows how to make a CALL to another page:

```
   tca   3           ;currently in segment 0
   tamd     num
   call     fm_synth     ;call subroutine
   tca   4
   tamd     num
   call     fm_synth     ;call subroutine
   cla
   .
   .
fm_synth
   cla
   acaac    #fec         ;point to PSA
   tax
   tsega    fm_synth1 ;extract segment number from address
   tam                   ;load segment to PSA register
   lbr      fm_synth1    ;long branch to new address
fm_rtn      retn         ;return from original CALL

   .
   segmnt             ;start of segment 1
   .
   .
   segmnt             ;start of segment 2
   .                     ;include interrupt vectors
                          if being used
   .
fm_synth1 intgr        ;beginning of subroutine
   tmad     num
   sala
   .
   .
stop  .                   ;end of subroutine
   cla
   acaac    #fec
   tax                ;point to PSA
   cla
   tam                ;load original segment
                          (0 in this case)
   lbr   fm_rtn    ;long branch to return routine
```

Because the CALL is only 12 bits, the CALL must be made to a location in the current segment. From there, the PSA set and a long branch are executed to

the target location. The same case is true for the return from the subroutine. A long branch must be made back to the original segment and then the RETN should be executed.

Note: Internal Vectors in Different Segments

If interrupts are being used on any other segment than 0, the interrupt vectors must be placed within the segment. When interrupts happen, the PSA value is taken into account. See Section 2.2 for more information.

Note:

All interrupts must be disabled while executing the LBR instruction, which branches across segment boundaries. If this is not done, it is possible for an interrupt to happen immediately following the change in the PSA register (but before executing the LBR instruction), resulting in the page change happening without the proper LBR instruction. The interrupts can be re-enabled on the other side of the LBR instruction, if required.

5.1.2 Using the SSA to Access an External ROM

The following is an example of how to use the SSA:

```
   aorg     #1000
a  tca   1           ;use 1st element in speech table
   sala
   sala              ;shift twice for offset into
                      speech table
   acaac    speech   ;add table address to offset

   tamd     temp     ;store address temporarily
   luab              ;dummy luab since value is a byte
   iac
   luaa              ;get segment value from table
   tamd     segment  ;store segment value
   cla
   acaac #fed        ;point to SSA register
   tax
   tmad     segment  ;retrieve stored segment value
   tam               ;store segment into SSA
   tmad     temp     ;retrieve speech table address
   iac               ;increment to point to speech
                      address
   iac
   luab              ;get MSB of speech address
   iac
   luaa              ;get LSB of speech address
   xba
   sala4
   sala4
```

```
   abaac              ;combine MSB and LSB
   luaps              ;load speech address register

   get   4
   get   4
   .
   .
speech
   data  0,(voc0&#7FFF)|#8000  ;segment 0, phrase 0
   data  0, (voc1&#7FFF)|#8000 ;segment 0, phrase 1
   data  1, (voc2&#7FFF)|#8000 ;segment 1, phrase 2
   data  1, (voc3&#7FFF)|#8000 ;segment 1, phrase 3
   data  2, (voc4&#7FFF)|#8000 ;segment 2, phrase 4
   data  2, (voc5&#7FFF)|#8000 ;segment 2, phrase 5
   data  3, (voc6&#7FFF)|#8000 ;segment 3, phrase 6
   data  3, (voc7&#7FFF)|#8000 ;segment 3, phrase
   data  4, (voc8&#7FFF)|#8000 ;segment 4, phrase 8
         aorg  #3000           ;SSA segment 0
voc0     copy  voc0.byt
voc1     copy  voc1.byt
         aorg  #8000           ;SSA segment 1
voc2     copy  voc2.byt
         aorg  #B000           ;still SSA segment 1
voc3     copy  voc3.byt
         aorg #10000           ;SSA segment 2
voc4     copy  voc4.byt
voc5     copy  voc5.byt
         aorg #18000           ;SSA segment 3
voc6     copy  voc6.byt
         aorg #1A000           ;still SSA segment 3
voc7     copy  voc7.byt
         aorg #20000           ;SSA segment 4
voc8     copy  voc8.byt
```

Because the most significant bit signals to use the SSA, #8000 must be ORed to every address to make sure that bit is set. The speech address register only accepts 16 bits, so when the most significant bit is set, the SSA is shifted left 15 bits and added to the speech address register value (minus the top bit). See Section 2.1.2 for the equations calculating this output address.

If voc0 was at voc0.byt (without adding #8000) and the PSA was set to 2, the speech address register would contain #3000. Since this address is less than #8000, the PSA would be shifted left 12 bits and added to the speech address register value resulting in an output address of #5000, which is incorrect. For this reason, the #8000 is added to each address to make sure that the SSA is used. It is not incorrect to use the PSA for speech data, it just takes a little more careful planning.

Since the speech table is located below #8000, the LUAA, LUAB, and LUAPS will load from the current segment according to the PSA. In this example, the speech table address is less than 12 bits, so it is unnecessary to mask the upper 4 bits.

5.1.3 Using the IRPC to Access an External ROM

The function of the IRPC is to designate access to the lower 4K of either the internal ROM or the external ROM. When a 1 is written to this register (#FEE), any address accessed below 4K will be from the external ROM. When a 0 is written to this register, an address accessed below 4K will be from the internal ROM. The following code demonstrates how to access the lower 4K of the external ROM.

```
a  cla            ;beginning of internal ROM
   clx
   br    #1000    ;branch to external ROM
   br    #1000
   extrnl         ;reset address to 0 for external ROM
start
   cla            ;start of external ROM
   clx
ram_loop
   tamix
   xgec  max_ram
   br    ram_exit
   br    ram_loop
ram_exit    .
   .
b  cla
   acaac #fec
   tax            ;point to  PSA
   tsega d        ;extract segment value from address
   tam            ;store segment value in PSA
   lbr   d        ;branch to beginning of segment
loaded
   aorg  #1000
c  cla
   acaac #fee
   tax            ;point to IRPC
   tca   1
   tam            ;set IRPC for lower 4K of external

   <more code>    ;branch does not have to
                  ;happen immediately
   br    start
   segmnt         ;start of segment 2
      .
      .
   segmnt         ;start of segment 3
d  tca   1
   tamd  temp
```

In the previous example, if the IRPC had not been set in code segment c, the branch to start would have branched to address 0 in the internal ROM.

Chapter 6

Customer Information

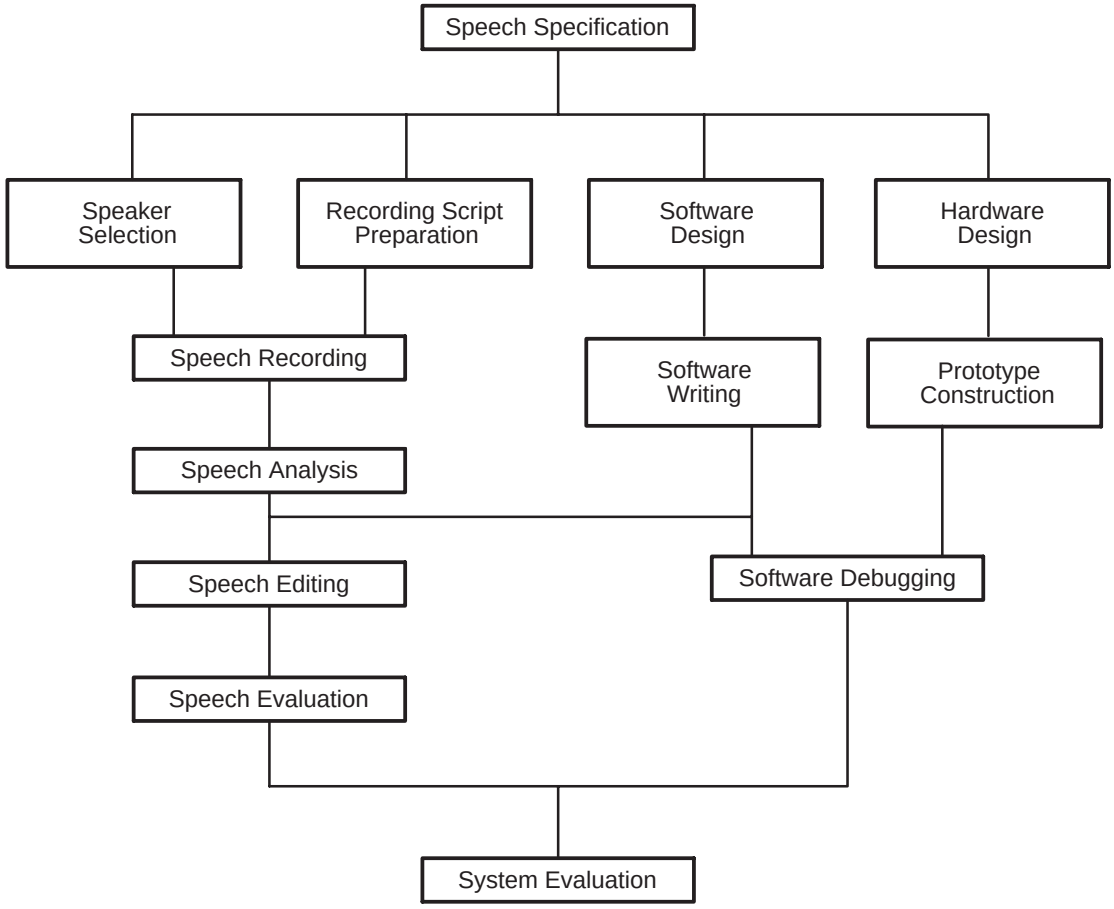
This chapter includes customer information on development cycle organization, development and production sequence, mechanical information and packaging availability, ordering information, and example ordering forms.

Topic **Page**

6.1 Development Cycle

The MSP50C30 development cycle is more complex than microprocessor development, because it adds speech development to the normal microprocessor development cycle. (Figure 6–1). The software design cycle is similar to that for other microprocessors. Speech development is discussed in Appendix A, *Script Preparation and Speech Development Tools*.

Figure 6–1. Speech Development Cycle



6.2 Summary of Speech Development/Production Sequence

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

1) For the speech development group at TI to accept a custom device program, the customer must submit a new product release form (NPRF). This form describes the custom features of the device (e.g., customer information, prototype and production qualities, symbolization). The NPRF is completed by product engineering and product marketing personnel within TI. A copy of the NPRF can be found in sections 6.5, *New Product Release Form.*

2) TI generates the prototype photomask and processes, manufactures, and tests 25 prototype devices for shipment to the customer. Limited quantities in addition to the 25 prototypes may be purchased for use in customer evaluation. All prototype devices are shipped against the following disclaimer: *It is understood that, for expediency purposes, the initial 25 prototype devices (and any additional prototype devices purchased) were assembled on a prototype (i.e., not production-qualified) manufacturing line whose reliability has not been characterized. Therefore, the anticipated inherent reliability of these devices cannot be expressly defined*.

3) The customer verifies the operation and quality of these prototypes and responds with either written customer prototype approval or disapproval.

4) A nonrecurring mask charge that includes the 25 prototype devices is incurred by the customer.

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

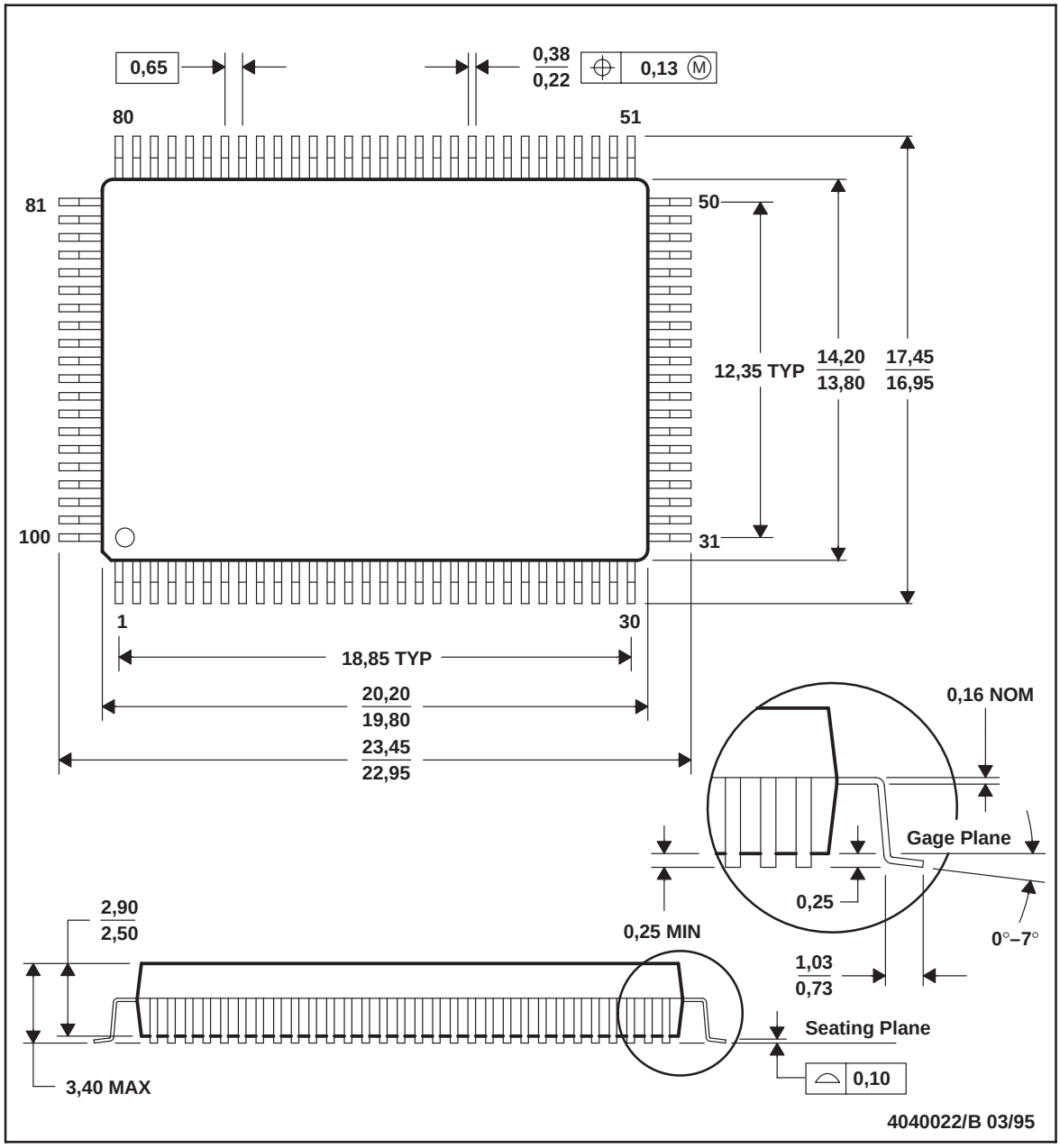
Note: Using Prototype Devices in Production Systems

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

6.3 Mechanical Information

The MSP50C30 is available in a 100-pin quad flatpack and in die form. See Table 1–3 for pad coordinates for the MSP50C30 die. The following figure shows the mechanical data for the 100-pin package.

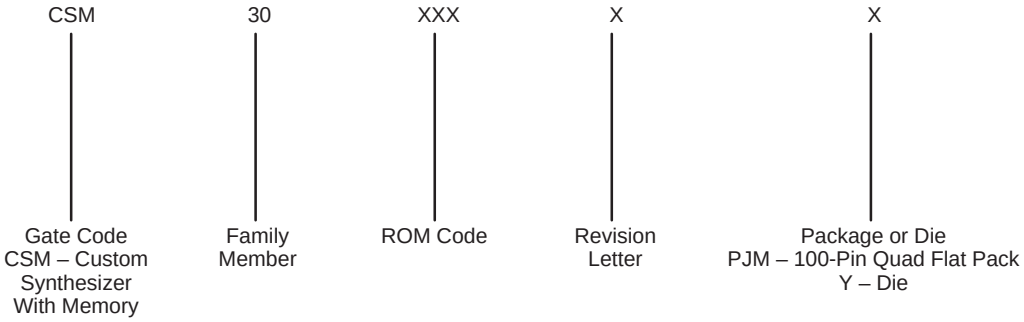
PJM (R-PQFP-G100) **PLASTIC QUAD FLATPACK**



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-022

6.4 Ordering Information

Because the MSP50C30 is a custom device, it receives a distinct identification as follows:



6.5 New Product Release Form

The new product release form is used to track and document all the steps involved in implementing a new speech code onto one of the parent speech devices. Blank forms are provided in this section (the addresses on these forms are subject to change). Copy the new product release forms (NPRF) provided or get one from your TI field sales office to initiate the implementation process. The next step is to complete Section 1. As seen on the blank forms, Section 1 allows you to choose the parent device for your particular code, as well as the options pertinent to the parent device you wish to use. Section 1 also allows you to choose your own customer part number used for ordering your parts. If no customer part number is indicated, then TI defaults to the CSM30xxxxx part number for ordering purposes. Completion of the company name, project name, and option fields is mandatory. Completion of all other fields in Section 1 is optional. After completion of Section 1, you must submit the NPRF (along with your speech code) to the speech products group via your local TI field sales office.

Once the speech products group receives the speech code and the NPRF, you have completed the initial steps involved in implementing this code onto production devices. Since all parent speech devices are mask programmable, the speech code must first be converted into a format that the speech products mask vendor can use to generate this new mask. This format is called a PG output. Once this PG output is generated, the original speech code is reconstructed from the PG output file and sent back to you for recheck. This recheck ensures that the PG output file was generated correctly. Along with the reconstructed speech code, the NPRF is also returned to you with Section 2 completed by TI. In this section, TI assigns your own CSM30xxxx part number and, in the case of packaged devices, TI also proposes a symbol format to you. If you wish to deviate from the suggested symbol format, you must consult TI for requested changes.

After you verify the reconstructed speech code and accept the proposed symbol format, you are required to sign section 3 as authorization for TI to generate the mask, prototypes, and risk units in accordance with the pertinent purchase order. You then send or fax the NPRF to the speech products group via the local TI field sales office. TI should have the prototypes shipped to you approximately six weeks after receiving the NPRF with section 3 signed. Once you receive these prototypes, you verify the functionality of the prototypes, sign section 4, and send the NPRF (with section 4 signed) back to TI. At this point, you can start ordering production units.

NEW PRODUCT RELEASE FORM FOR MSP50C30

SECTION 1. OPTION SELECTION
This section is to be completed by the customer and sent to TI along with the microprocessor code and speech data.

Company:_____ Division:_____
Project Name:_____ Purchase Order #:_____
Management Contact: _____ Phone:(___) _____
Technical Contact : _____ Phone:(___) _____
Customer Part Number:_____

D/A Output (check one):

___ 2 pin push–pull (2D)

___ Single pin double ended (1A)

Oscillator (check one):

___ Internal (Two ranges selected by software:
1) 15.36 MHz (14.89 MHz – 15.82 MHz)
2) 19.2 MHz (18.62 MHz – 19.77 MHz))

___ External

Package Type (check one):

___ PJM (100 pin QFP)

___ Die

****

SECTION 2A. ASSIGNMENT OF TI PRODUCTION PART NUMBER
This section is to be completed by TI.

TI Part Number:_____ (CSM30xxxY or CSM630xxxPJM)

SECTION 2B. PACKAGE UNIT SYMBOLIZATION
This section is to be completed by the customer.
The first line of the symbolization is fixed. Except EIA#/Logo.
The second and third lines are to be filled in by the customer.

Top Side Symbolization (100pin 'PJM')

```
                        +----------------------+  LLLL: LOT TRACE CODE
                        |     ??? YMLLLLT      |  YM:   DATE CODE
                        |    <optional 13 char> |  T:    ASSY SITE
                        |    <optional 11 char> |  ???:  TI EIA NO. or
                        +----------------------+           TI LOGO
```

For '100 PJM' packages, the customer may choose between TI EIA No. 980 or the TI LOGO on the first line. 2nd line is typically the TI Part Number.

****

SECTION 3. AUTHORIZATION TO GENERATE MASKS, PROTOTYPES, AND RISK UNITS
This section is to be completed by the customer and sent to TI after the following criteria have been met:

1) The customer has verified that the TI computer generated data matches the original data.

2) The customer of the symbolization format in Section 2B (Applies to packaged devices only).
original data.

I hereby certify that the TI generated verification data has been checked and found to be correct, and I authorize TI to generate masks, prototypes, and risk units in accordance with purchase order in section 1 above. In addition, in the instance that this is a packaged device, I also authorize TI to use the symbolization format illustrated in section 2B on all devices with the part number indicated in section 2A.

By:_____ Title:_____

Date:_____

(FAX this form to 214–480–7301. Attn: Code Release Team)

****

SECTION 4. APPROVAL OF PROTOTYPES AND AUTHORIZATION TO START PRODUCTION

This section is to be completed by the customer after prototype devices have been received and tested.

I hereby certify that the prototype devices have been received and tested and found to be acceptable, and I authorize TI to start normal production in accordance with purchase order #_____.

By:_____ Title:_____

Date:_____

****

Return to: Texas Instruments, Inc.
Attn: Code Release Team
P.O. Box 660199, M/S 8718
Dallas, TX 75266–0199

OR Fax to: (214)480–7301
Attn: Code Release Team

Have Questions?:
CALL: Code Release Team
(214)480–4444

OR E–MAIL: code–rel@msp.sc.ti.com

Appendix A

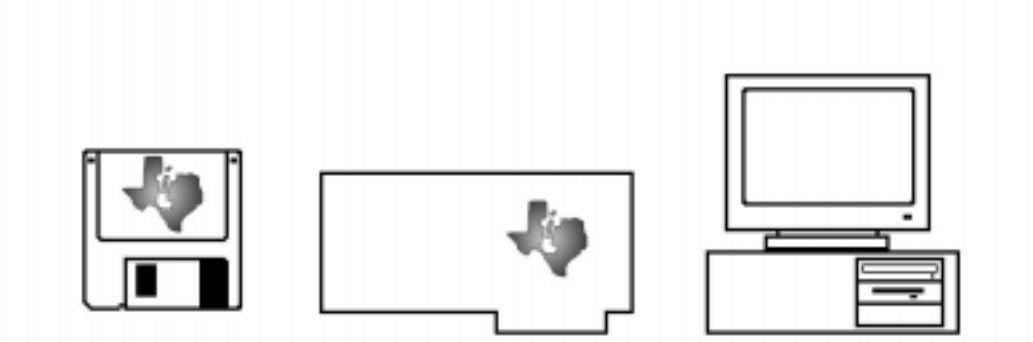
Script Preparation and Speech Development Tools

Topic Page

A.1 MSP50C30 Speech Development Tools

The following figures show the various development tools available and list the features of each.

Figure A–1. WINSDS



WINSDS **PC (At-Class Computer)**

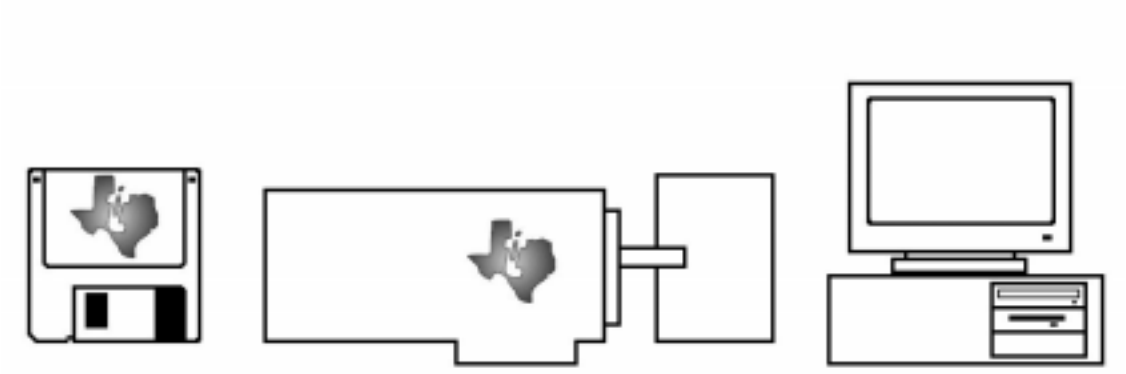
WINSDS Features

- ❑ High-speed speech analysis (real time) for LPC and MELP
- ❑ Graphical and numerical speech editing for LPC and MELP
- ❑ Microphone and line-level inputs
- ❑ Headphone and line outputs
- ❑ Supports TSP50C0x/1x and MSP50C3x devices
- ❑ Requires an AT–class computer (a 100-MHz 486 is recommended) with a VGA card and Windows™ 3.1, 3.11, or Windows 95™
- ❑ Uses TMS32031 Digital Signal Processor

Note: Required and Recommended Equipment

A hard disk drive is required and a tape backup system is strongly recommended for the WINSDS development system.

Figure A–2. EMU50C30



EMU50C30 Features

- ❑ In-circuit emulation
- ❑ Hardware breakpoints
- ❑ Single step
- ❑ Examine/modify registers/memory
- ❑ Includes assembler
- ❑ Requires one ISA card slot in AT-class computers (486 – 66MHz or better recommended)
- ❑ Optional real-time trace
- ❑ Built-in 1M memory to emulate external ROM
- ❑ Optional memory expansion cards (4M and 8M)

Appendix B

MSP50C3x Versus TSP50C1x

This appendix contains information about switching from a TSP50C1x family device to a MSP50C3x family device.

Topic **Page**

B.1 Summary of Changes From TSP50C1x Family

- Support dual LPC capability
  - RAM size is 256 locations
  - 32 12-bit RAM locations
  - Design part for double clock speed (19.2 MHz)
  - Double mode register to 2 memory mapped 8-bit registers
  - Add dual LPC algorithm
  - Add 2nd Pitch period counter
  - PPC < 0x200 sets the bit in the mode register instead of triggering an interrupt
  - Level 1 interrupt in LPC mode happens every 30 instruction cycles.
- Increase ROM size
  - Increase Program Counter register width to 16 bits
  - Increase A register width to 16 bits
  - Increase B register width to 16 bits
  - Increase SAR width to 16 bits
  - Increase ROM size to 64K bytes
  - Adjust for SSA, PSA, and IRPC usage
- Miscellaneous
  - No support for external ROM mode
  - Improved internal oscillator design
  - Improved D/A output design
  - Designed part for increased V_{DD} range (3.3V – 6.5V)
  - Remap A port to 0xFCh – 0xFFh RAM range
  - Remap B port to 0xF8h – 0xFBh RAM range
  - Added wakeup function
  - 3 oscillator modes (internal, external, and crystal/ceramic)
  - Status gets set by SETOFF, $\overline{\text{INIT}}$ terminal low, and interrupt
  - Fix DECMN instruction so that true decrement happens on 12-bit RAM
  - Fix ANDCM instruction so that operation on 12-bit RAM location does not affect the upper 4 bits
  - SETOFF does not change the I/O state
  - Memory map mode registers
  - I/O ports are paged in and out of the memory map by the IO_MAP bit in the mode register
  - Interrupt vector location is changed

B.2 Upgrading a TSP50C1x Program to a MSP50C3x Program

This section documents the changes necessary to upgrade a TSP50C1x program to a MSP50C3x program. This file assumes that none of the additional features of the MSP50C3x are being used. for example, the wakeup function does not exist on the TSP50C1x, so code changes necessary to accommodate the wakeup function are not included.

Assumptions:

- ❏ The wakeup function is not used
- ❏ Only the A and B ports are used
- ❏ Only 128 RAM locations or less are used
- ❏ Only 16K ROM locations are used
- ❏ All program code is in the lower 4K bytes of ROM

B.2.1 Normal Operation

Remap interrupt vectors

The interrupt vectors are I/O mapped in a different sequence on the MSP50C3x than they are on the TSP50C1x.

Table B–1.Interrupt Vectors for the TSP50C1x and the MSP50C3x

Address	Type	TSP50C1x	MSP50C3x
0x0010	Level 2 Interrupt	PCM = 0, LPC = 1	PCM = 0, LPC = 0
0x0012	Level 2 Interrupt	PCM = 0, LPC = 0	PCM = 0, LPC = 1
0x0014	Level 2 Interrupt	PCM = 1, LPC = 1	PCM = 1, LPC = 0
0x0016	Level 2 Interrupt	PCM = 1, LPC = 0	PCM = 1, LPC = 1
0x0018	Level 1 Interrupt	PCM = 0, LPC = 1	PCM = 0, LPC = 0
0x001A	Level 1 Interrupt	PCM = 0, LPC = 0	PCM = 0, LPC = 1
0x001C	Level 1 Interrupt	PCM = 1, LPC = 1	PCM = 1, LPC = 0
0x001E	Level 1 Interrupt	PCM = 1, LPC = 0	PCM = 1, LPC = 1

Remap I/O ports

The I/O ports are I/O mapped to different RAM locations on the MSP50C3x than they are on the TSP50C1x.

Table B–2.I/O Ports for the TSP50C1x and the MSP50C3x

Port	TSP50C1x Address	MSP50C3x Address
Port A Input Register	0x80	0xFC
Port A Pullup Register	0x81	0xFD
Port A Tristate Register	0x82	0xFE
Port A Output Register	0x83	0xFF
Port B Input Register	0x84	0xF8
Port B Pullup Register	0x85	0xF9
Port B Tristate Register	0x86	0xFA
Port B Output Register	0x87	0xFB

Adjust code for 16-bit register size

The A and B ports are 14 bits wide on the TSP50C1x family and are 16 bits wide on the MSP50C3x family. Any code depending on the 14-bit size of the TSP50C1x family must be adjusted. For example, a common way of excising bits at the most significant portion of the data word is to left shift the data in the A register, then right shift to properly justify the data. An additional 2 bits of shift would be needed on the MSP50C3x family.

Select clock speed

On the TSP50C1x family, the clock speed is fixed. On the MSP50C3x family, the internal clock speed is switchable. If the internal clock option is used on the MSP50C3x, then the speed needs to be set in software.

Adjust code for 12-bit RAM location

On the TSP50C1x family, RAM locations 0x10h through 0x1Fh are 8 bits wide. On the MSP50C3x family, these RAM locations are 12 bits wide. Any use of these RAM locations that assumes an 8-bit width needs to be fixed.

Adjust for clock speed

The MSP50C3x family operates at a clock speed that is twice that of the TSP50C1x family. Any clock speed dependence must be adjusted.

TSP60C18 support removed

The support for the TSP60C18/81 speech ROM has been removed. The external ROM bit has been removed from the mode register. To use an external ROM, the interface must be done in software.

Adjust code for 12-bit X register size

On the TSP50C1x family, the X register is 8 bits wide. On the MSP50C3x family, the X register is 12 bits wide. Any use of the X register that assumes an 8-bit width needs to be fixed.

Converting TSP50C19 code

There is no paging on the MSP50C3x family.

B.2.2 LPC

Adjust code for different LPC interrupt

On the TSP50C1x family, the LPC interrupt was caused by the underflow of the pitch period counter. On the MSP50C3x family, the LPC interrupt is invoked at a 20-kHz rate by the system clock. The underflow of the pitch period counter causes a bit to be set to 1 in mode register 2.

Modify the MSP50C3x code to periodically poll the mode register for the PPC bit to be set and then clear the bit and either branch or call the interpolation routine.

Clear memory-mapped register prior to turning on LPC

RAM locations 0xFF0h, 0xFF1h, 0xFF2h, and 0xFF4h should be cleared to 0 prior to turning on the LPC. The RAM location 0xFF3h should be set to 0x162h.

Replace the excitation function with the MSP50C3x excitation function.

Thie excitation function is the same as the TSP50C1x excitation function except that the unvoiced portion (#3A80) is sign extended to 16 bits (#FA80). The voiced portion of the excitation function may need to be multiplied by 2 to get the same volume as the TSP50C1x.

Move the excitation function

In TSP50C1x programs (except the TSP50C19), the excitation function needs to be moved from 0x4000 to 0x8000 when using the MSP50C33 or to 0x10000 when using the MSP50C34.

Note: TSP50C19

On the TSP50C19, the excitation function already has been moved to 0x8000h.

B.2.3 PCM

Adjust data for the different register size

On the TSP50C1x family, the TASYN instruction loaded the PCM data to a register 14 bits wide. The least significant 2 bits were unused and the most significant 2 bits were used for sign extension.

On the MSP50C3x family, the TASYN instruction loads the PCM data to a register 16 bits wide. The least-significant two bits are unused. The most significant two bits are used for overflow with the third most-significant bit being used for sign extension.

Appendix C

Quick Guide to Programming the MSP50C3x

This appendix contains information about programming the MSP50C30.

Topic **Page**

C.1 A Quick Guide to Programming the MSP50C30

The MSP50C30 has a PSA (program segment address) register and a SSA (speech segment address) register. These allow easy access to program code and data located anywhere in the 8M byte address space of the MSP50C30.

Typically, program code is placed in the first 1M byte of ROM and accessed through the PSA register. It is possible to write code in the SSA region (above 1M byte) but in a rather limited fashion.

Speech data may be placed anywhere in ROM (subject to the SSA segment boundaries) and accessed through the SSA register.

A PSA segment is 32K bits in length and is overlapping, with a 4K-bit offset between successive segments. An SSA segment is 32K bits in length and is non-overlapping.

C.1.1 General Information

- The MSP50C30 has 4K bytes of internal ROM and can address up to 8M bytes of external ROM.
- Program execution starts in internal ROM at #0000 (cold reset) or #0002 (warm reset). Program code in the external ROM can be reached by executing `BR #1000` from internal ROM
- At power up, the IRPC (internal ROM page control) register is cleared to zero. This allows access to the 4K bytes of internal ROM. When executing from external ROM, access the lower 4K bytes of external ROM by setting the IRPC to one.

C.1.2 Using the PSA

- Access the PSA register through the X register (see the following example).

```
CLA
ACAAC  #FEC
TAX
TSEGA  PSLABEL
TAM
```

- Extract the PSA segment number of a particular address by using TSEGA.
- To reach a different PSA segment, change the PSA register and then immediately execute an LBR. Alternately, use XBA followed by BRA.
- After writing to the PSA register, there is a 2-cycle delay before the change takes effect. This is intended for the LBR, or XBA/BRA instructions. Instructions other than these can cause unpredictable results.

- It is acceptable to single-step through an LBR but do not single-step through XBA/BRA. Instead, run to an appropriate breakpoint.
- The CALL instruction takes a 12-bit address, therefore, all subroutines must be located within the first 4K bytes of a PSA segment.
- The LBR instruction takes a 13-bit address, therefore, the desired branch address offset within a PSA segment must be in the range of #0000 to #1FFF.
- The 13-bit (8K bytes) boundary of the LBR must be kept in mind when moving between segments. For example, if the offset within a PSA segment is #4128 and a return to the original PSA segment with offset #0010 is desired, then the offset of #4128 must first be changed (using BSR) to an offset of less than #2000. The PSA can then be modified and a LBR #10 executed.
- The segment compiler directive starts a new PSA segment.

C.1.3 Using the SSA

- Access the SSA register through the X register (see the following example).

```
CLA
ACAAC  #FED
TAX
TCA    SSALABEL/#8000
TAM
```

- Extract the SSA segment number of a particular address by dividing by #8000.
- To reach the SSA region, execute a BRA with bit 15 of the A register set high. For example, if the SSA register is set to 33 and the A register contains #8000, performing a BRA changes the program counter to #108000.
- Only BRA will work (with bit 15 of the A register set high) in the SSA region. The BR, SBR, LBR, and CALL instruction will not work.
- Do not change the PSA register while in the SSA region. Instead, return to the previous PSA segment and then modify the PSA register as desired.
- The segment compiler directive is not applicable to SSA segments.

Appendix D

Using the PSA and SSA Routines

This appendix contains the code for using of PSA and SSA routines when writing code for the MSP50C30. There are four routines; the first three use the PSA, the last routine uses the SSA. The SSA routine is located above 1M byte; therefore, the EMU30 requires a memory expansion card in order to run this program.

Topic **Page**

D.1 Using the PSA and SSA Routines

SPLBR.ASM

The following code shows how to make use of the PSA and SSA when writing code for the C30. There are four routines; the first three use the PSA, the last routine uses the SSA. Note that the SSA routine is located above 1M byte; therefore, the EMU30 will require a memory expansion card in order to run this program.

```
          call   jump7                  ;jump to PSA segment 7, #07000
          call   jump17                 ;jump to PSA segment 17, #11000
          call   jump33                 ;jump to PSA segment 33, #21000
          call   jumpssa                ;jump to SSA segment 37, #128000
loophere  br     loophere
```

The program segment address (PSA) register is set to the segment address of the psa7 routine. This is achieved using the TSEGA command, which extracts the 8-bit program segment from a given address. The LBR instruction branches across program segments to the address offset, psa7, within the seventh segment.

```
*
* Jump to PSA 7 and come back
*
jump7     cla
          acaac  #fec                   ;point to PSA
          tax
          tsega  psa7                   ;get segment number
          tam                           ;set PSA
          lbr    psa7                   ;this MUST follow the TAM
```

In PSA segment 7 the A, B and X registers are set to 1, 2, and 3 respectively.

```
************************************************************
* psa7
*
* Set the A, B and X registers then LBR back to the calling
* routine in PSA segment 0.
************************************************************

          aorg   #7000
          segmnt
psa7      tca    1                      ;set A=1, B=2, X=3
          tab
          ibc
          tcx 3

* 0x07006
* A=1, B=2, X=3
```

In order to return to the zeroth segment the PSA is set to 0, and a LBR executed back to the segment from which psa7 was called.

```
          cla                          ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn               ;want to go to PSA segment 0
          tam
          lbr    longrtn               ;long branch there
The longrtn label is in PSA segment zero.
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn                   ;do a long branch back here, then return

This returns to the main program loop.
          call   jump17          ;jump to PSA segment 17, #11000

          call   jump33          ;jump to PSA segment 33, #21000
          call   jumpssa         ;jump to SSA segment 37, #128000

loophere  br     loophere
```

The procedure for branching to PSA 17 is the same as the branch to PSA 7, except the PSA is obviously set to 17 rather than 7.

```
*
* Jump to PSA 17 and do a CALL whilst there
*
jump17    cla
          acaac  #fec            ;point to PSA
          tax
          tsega  psa17           ;get segment number
          tam                    ;set PSA
          lbr    psa17           ;this MUST follow the TAM
```

In PSA segment 17 the A, B and X registers are set to 4, 5 and 6 respectively.

```
************************************************************
* psa17
*
* Set the A, B and X registers and then do a CALL within the
* same segment.
************************************************************

          aorg   #11000
          segmnt
psa17     tca    4               ;set A=4, B=5, X=6
          tab
          ibc
          tcx    6

* 0x11006
* A=4, B=5, X=6
```

A call to a subroutine within the same segment is then executed. This routine performs integer division, whereby the contents of A are divided by the contents of B, with the answer being stored in A and the remainder in B.

```
          tca    5               ;do 17 div 5
          tab
          tca    17
          call   divide          ;call divide routine in same segment
* 0x1100d
* A=3, B=2, X=5
```

The divide routine is located at #11600 which is also in PSA segment 17.

```
          aorg   #11600
divide    clx                           ;reset counter
divloop   sbaan                         ;do A –= B
          sbr    divdone                ;if negative we have finished
          ixc                           ;else increment counter
          sbr    divloop
          sbr    divloop
divdone   abaac                         ;get remainder in A
          xbx                           ;put answer in A and remainder in B
          xba
          retn
```

In order to return to the zeroth segment the PSA is set to 0, and a LBR executed back to the segment from which psa17 was called.

```
          cla                           ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn                ;want to go to PSA segment 0
          tam
          lbr    longrtn                ;long branch there
```

The longrtn label is in PSA segment zero.

```
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn                          ;do a long branch back here, then return
Control then returns to the main program loop.
          call   jump33                 ;jump to PSA segment 33, #21000
          call   jumpssa                ;jump to SSA segment 37, #128000

loophere  br     loophere
```

The procedure for branching to PSA 33 is the same as the branch to PSA 7, except the PSA is obviously set to 33 rather than 7.

```
*
* Jump to PSA 33 and then to PSA 255 and return
*
jump33    cla
          acaac  #fec                   ;point to PSA
          tax
          tsega  psa33                  ;get segment number
          tam                           ;set PSA
          lbr    psa33                  ;this MUST follow the TAM
```

In PSA segment 33 the A, B and X registers are set to 7, 8 and 9 respectively.

```
*************************************************************
* psa33
*
* Set the A, B and X registers then continue to LBR up to #ff000
* which is the psa255 routine.
*************************************************************

          aorg   #21000
          segmnt

psa33     tca    7                    ;set A=7, B=8, X=9
          tab
          ibc
          tcx    9

* 0x21006
* A=7, B=9, X=9
```

The highest PSA segment (255) is reached by setting the PSA register accordingly.

```
          cla                         ;now set up PSA for LBR up to #ff000
          acaac  #fec
          tax
          tsega  psa255               ;want to go to PSA segment 255
          tam
          lbr    psa255               ;long branch there
```

In PSA segment 255 the A, B and X registers are set to #a, #b and #c respectively.

```
*************************************************************
* psa255
*
* Set the A, B and X registers then LBR back to the original
* routine which first called psa33.
*************************************************************

          aorg   #ff000
          segmnt

psa255    tca    10                   ;set A=a, B=b, X=c
          tab
          ibc
          tcx    12

* 0xff006
* A=a, B=b, X=c
```

The zeroth segment is returned to in the usual way.

```
          cla                         ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn              ;want to go to PSA segment 0
          tam
          lbr    longrtn              ;long branch there
```

The longrtn label is in PSA segment zero.

```
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn                          ;do a long branch back here, then return
```

Control then returns to the main program loop.

```
          call   jumpssa                ;jump to SSA segment 37, #128000

loophere  br     loophere
```

The procedure for branching to a location in the speech segment address (SSA) space is different. The SSA register is set to the required value, 37 in this case, but to reach an address above 1M byte the BRA instruction must be used. Setting the MSB (bit 15) high forces the SSA register to be used in the calculation of the new program counter value. This gives an actual address of

```
   (#7fff & BRA address) + (SSA << 15)
```

So here the new address is simply (#0412 + 37 << 15), which equals #128412.

```
*
* Jump to SSA 37 and return
*
jumpssa   cla
          acaac  #fed                       ;point to SSA
          tax
          tca    (ssa37/SSASEG)
          tam                               ;set SSA to 37th segment
          cla                               ;now form 16 bit address
          clb
          acaac  ssa37&#ff                  ;LSB 8 bits
          xba
          acaac  ((ssa37&#7fff)+#8000)/#100
          sala4                             ;MSB with top bit set high
          sala4                             ;shift left 8
          abaac                             ;add LSB 8 bits
          bra                               ;branch to #128412
```

In SSA segment 37 the A, B and X registers are set to #d, #e and #f respectively.

```
*************************************************************
* ssa37
*
* Set the A, B and X registers then BRA back to the PSA segment
* zero.
*************************************************************
          aorg   #128412
ssa37     tca    13                         ;set A=d, B=e, X=f
          tab
          ibc
          tcx    15
```

```
* 0x128418
* A=d, B=e, X=f
```

In order to return to the zeroth PSA segment a BRA instruction is executed. This time the MSB (bit 15) is set low so that the SSA is ignored in the program counter calculation. Since the PSA is currently 0, the return address is simply equal to longrtn.

```
* Return to PSA region and original CALL
         cla                              ;get address of return in PSA 0
         acaac  longrtn                   ;branch there, ignoring SSA value
         bra
```

The longrtn label is in PSA segment zero.

```
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn                           ;do a long branch back here, then return
```

The program then waits in an infinite loop.

```
loophere  br     loophere
```

Note: LBR Instruction and Branching Across PSA Segments

The LBR instruction is used to branch across PSA segments. Because the LBR takes a 13-bit address, the entry into the segment is limited to the first 8K bytes. For example, an LBR from #0049 to #10000 (PSA 16) is valid but an LBR from #0049 to #17F42 (also PSA 16) produces an error “BR or LBR out of 8K page range”. However, if the #17F42 is in a segment starting at #16000 or #17000 then the LBR will work.

Note: Returning From One PSA Segment to the Other

The same constraint, outlined in the previous note, applies when returning from one PSA segment to the other. For example, if a segment begins at #10000 and ends at #17FFF then in order to return from #17Fxx to #004B it is necessary to use BRA to move to the first 8K of the segment at #10000, and then use LBR to jump to #004B in PSA 0.

The following code demonstrates the correct usage of LBR. Essentially, the program starts in PSA 16 at #10000, does a BRA to reach #17F42 within the same segment, then a BRA back to #10014 before doing an LBR to PSA 0.

```
         aorg   #10000
         segmnt
psa16    tca    1                        ;set A=1, B=2, X=3
         tab
         ibc
         tcx 3
```

```
* 0x10006
* A=1, B=2, X=3
```

In order to reach the #17Fxx region the BRA instruction must be used. LBR will not work because an 8K page boundary is crossed.

```
          cla                          ;now form 16 bit address
          clb
          acaac  adr17f&#ff            ;LSB 8 bits
          xba
          acaac  (adr17f&#7fff)/#100
          sala4
          sala4                        ;shift left 8
          abaac                        ;add LSB 8 bits
          bra                          ;branch to #17f42
```

At #17F42 the A, B and X registers are set to 7, 8 and 9 respectively.

```
          aorg   #17f42

adr17f    tca    7                     ;set A=7, B=8, X=9
          tab
          ibc
          tcx    9
```

Trying to return to the #10000 region with LBR will cause a compiler error.

```
*
* This will not compile because the LBR is outside the 8K limit.
* Need to BRA back to the first 8K of the segment and then LBR
* back to PSA 0.
*

*         cla                              ;now set up PSA for LBR back to start
*         acaac  #fec
*         tax
*         tsega  longrtn                   ;want to go to PSA segment 0
*         tam
*         lbr    longrtn                   ;long branch there ***ERROR***
```

Instead a BRA must be executed, in this case back to #10014.

```
          cla                              ;now form 16 bit address
          clb
          acaac  psa16rtn&#ff              ;LSB 8 bits
          xba
          acaac  (psa16rtn&#7fff)/#100
          sala4
          sala4                            ;shift left 8
          abaac                            ;add LSB 8 bits
          bra                              ;branch to #10014
```

Finally the LBR can be used to return to PSA 0, to the longrtn at #4B.

```
*
* 0x10014
* BRA here from higher up in the same segment
*
psa16rtn  cla                              ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn                   ;want to go to PSA segment 0
          tam
          lbr    longrtn                   ;long branch there
```

Appendix E

Using PSA and SSA With the TSEGA, LBR, CALL and BRA Instructions

This appendix contains code that demonstrates the use of the PSA and SSA routines with TSEGA, LBR, CALL and BRA instructions.

Topic **Page**

E.1 Using the PSA and SSA With the TSEGA, LBR, CALL, and BRA Instructions

```
* splbr.asm
*
* 27th May 1998
*
* Demonstrates usage of the PSA and SSA, with TSEGA,
* LBR, CALL and BRA instructions.
*
* jump7   call and branch to PSA 7, #7000
* jump17  call and branch upto PSA 17, #11000
* jump33  call and branch upto PSA 33, #11000
* jumpssa call and branch upto SSA 37, #128000
*
* Set breakpoints and observe the registers.
*
* ADDRESS    A  B  X
*   7006     1  2  3
*  11006     4  5  6
*  1100d     3  2  5
*  21006     7  8  9
*  ff006     a  b  c
* 128418     d  e  f
*
* Set a breakpoint at #34 which is the finish point.
      OPTION BUNLIST,DUNLIST,PAGEOF
************************************************************
* 8 bit RAM variables
************************************************************

************************************************************
* Constants
************************************************************
PSASEG    equ    #1000                ;PSA segment size
SSASEG    equ    #8000                ;SSA segment size
************************************************************
*    General Purpose Constant Definitions
************************************************************
*
*   DEVICE CONSTANTS
*
MAX_RAM    EQU     #EF    ;Highest RAM location
*
*   I/O Port Definitions
*
INPUT_A      EQU    #FC
PULLUP_A     EQU    #FD
DIRECT_A     EQU    #FE
OUTPUT_A     EQU    #FF

INPUT_B      EQU    #F8
PULLUP_B     EQU    #F9
DIRECT_B     EQU    #FA
OUTPUT_B     EQU    #FB
```

```
INPUT_C       EQU    #F4
PULLUP_C      EQU    #F5
DIRECT_C      EQU    #F6
OUTPUT_C      EQU    #F7

INPUT_D       EQU    #F0
PULLUP_D      EQU    #F1
DIRECT_D      EQU    #F2
OUTPUT_D      EQU    #F3
*-------------------------------------------------------------
*   Start of program
*-------------------------------------------------------------
*     AORG   #0000
************************************************************
*   Interrupt vectors
************************************************************
          AORG   #0010
          SBR    INT2_00                ;Timer Underflow, PCM=0, LPC=0
          SBR    INT2_00                ;Timer Underflow, PCM=0, LPC=0
          SBR    INT2_01                ;Timer Underflow, PCM=0, LPC=1
          SBR    INT2_01                ;Timer Underflow, PCM=0, LPC=1
          SBR    INT2_10                ;Timer Underflow, PCM=1, LPC=0
          SBR    INT2_10                ;Timer Underflow, PCM=1, LPC=0
          SBR    INT2_11                ;Timer Underflow, PCM=1, LPC=1
          SBR    INT2_11                ;Timer Underflow, PCM=1, LPC=1
          SBR    INT1_00                ;Pin (B1) goes low interrupt
          SBR    INT1_00                ;Pin (B1) goes low interrupt
          SBR    INT1_01                ;Clock interrupt, PCM=0, LPC=1
          SBR    INT1_01                ;Clock interrupt, PCM=0, LPC=1
          SBR    INT1_10                ;Clock interrupt, PCM=1, LPC=0
          SBR    INT1_10                ;Clock interrupt, PCM=1, LPC=0
          SBR    INT1_11                ;Clock interrupt, PCM=1, LPC=1
          SBR    INT1_11                ;Clock interrupt, PCM=1, LPC=1

INT1_10
INT1_01
INT2_10
INT2_00
INT2_01
INT2_11
INT1_00
INT1_11   CLA
          RETI
************************************************************
*   Do program INITs
************************************************************
GO        CLA                           ;Clear Memory mapped Registers
          intgr
          ACAAC  #FF0
          TAX
          CLA
          TAMIX
          TAMIX
          TAMIX
          TAMIX

          call   jump7                  ;jump to PSA segment 7, #07000
```

```
          call   jump17                    ;jump to PSA segment 17, #11000

          call   jump33                    ;jump to PSA segment 33, #21000

          call   jumpssa                   ;jump to SSA segment 37, #128000

loophere  br  loophere

*
* Jump to PSA 7 and come back
*
jump7     cla
          acaac  #fec                      ;point to PSA
          tax
          tsega  psa7                      ;get segment number
          tam                              ;set PSA
          lbr    psa7                      ;this MUST follow the TAM

*
* Jump to PSA 17 and do a CALL whilst there
*
jump17 cla
          acaac  #fec                      ;point to PSA
          tax
          tsega  psa17                     ;get segment number
          tam                              ;set PSA
          lbr    psa17                     ;this MUST follow the TAM

*
* Jump to PSA 33 and then to PSA 255 and return
*
jump33 cla
          acaac  #fec                      ;point to PSA
          tax
          tsega  psa33                     ;get segment number
          tam                              ;set PSA
          lbr    psa33                     ;this MUST follow the TAM

*
* Jump to SSA 37 and return
*
jumpssa   cla
          acaac  #fed                      ;point to SSA
          tax
          tca    (ssa37/SSASEG)
          tam                              ;set SSA to 37th segment

          cla                              ;now form 16 bit address
          clb
          acaac  ssa37&#ff                 ;LSB 8 bits
          xba
          acaac  ((ssa37&#7fff)+#8000)/#100
          sala4                            ;MSB with top bit set high
          sala4                            ;shift left 8
          abaac                            ;add LSB 8 bits
          bra                              ;branch to #128412
```

```
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn         ;do a long branch back here, then return
*********************************************************
* PSA Segment 1 – equivalent to aorg #1000
*
* Program Entry Point
*
* The program in the internal ROM branches to here.  Clear
* the RAM and do some initialization.
*********************************************************

          aorg   #1000
          segmnt

          CLA
          tca    #08                     ;set D3 as output
          tamd   direct_d
          cla
          tamd   pullup_d
          tamd   output_d

          CLA
          ACAAC  #FEE                    ;set Internal ROM Page Control
          TAX                            ;  (IRPC) so we can access
          TCA    1                       ;  lower 4K of external ROM
          TAM

          TMAD   0

          CLA                            ;Initialize mode register
          TAMODE

          CLX
RAM_LOOP  TAMIX                          ;Initialize All RAM to zeros
          XGEC   MAX_RAM
          SBR    RAM_EXIT
          SBR    RAM_LOOP

RAM_EXIT  TAM                            ;initialize last RAM location (FF)

          TCA    #AA
          TAMD   DIRECT_A                ;set even bits as inputs
          TAMD   DIRECT_B                ; and odd as outputs
          TAMD   DIRECT_C
          TAMD   OUTPUT_A
          TAMD   OUTPUT_B
          TAMD   OUTPUT_C
          TCA    #55
          TAMD   PULLUP_A                ;turn pullups on for inputs
          TAMD   PULLUP_B
          TAMD   PULLUP_C

          CLA
          ACAAC  #FEC
          TAX
          TCA    0
          TAM
```

```
*
* Now do a long branch back to the start of the external ROM
* program, where we do some branches in the PSA and SSA.
*
          LBR    GO
*****************************************************************
* psa7
*
* Set the A, B and X registers then LBR back to the calling
* routine in PSA segment 0.
*****************************************************************
          aorg   #7000
          segmnt
psa7      tca    1                      ;set A=1, B=2, X=3
          tab
          ibc
          tcx    3
* 0x07006
* A=1, B=2, X=3
          cla                           ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn                ;want to go to PSA segment 0
          tam
          lbr    longrtn                ;long branch there
*****************************************************************
* psa17
*
* Set the A, B and X registers and then do a CALL within the
* same segment.
*****************************************************************
          aorg   #11000
          segmnt
psa17     tca    4                      ;set A=4, B=5, X=6
          tab
          ibc
          tcx    6
* 0x11006
* A=4, B=5, X=6
          tca    5                      ;do 17 div 5
          tab
          tca    17
          call   divide                 ;call divide routine   in same segment
* 0x1100d
* A=3, B=2, X=5
          cla                           ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn                ;want to go to PSA segment 0
          tam
          lbr    longrtn                ;long branch there
```

```
          aorg   #11600
divide    clx                              ;reset counter
divloop   sbaan                            ;do A -= B
          sbr    divdone                   ;if negative we have finished
          ixc                              ;else increment counter
          sbr    divloop
          sbr    divloop
divdone   abaac                            ;get remainder in A
          xbx                              ;put answer in A and remainder in B
          xba
          retn
*************************************************************
* psa33
*
* Set the A, B and X registers then continue to LBR up to #ff000
* which is the psa255 routine.
*************************************************************
          aorg   #21000
          segmnt
psa33     tca    7                         ;set A=7, B=8, X=9
          tab
          ibc
          tcx    9
* 0x21006
* A=7, B=9, X=9
          cla                              ;now set up PSA for LBR up to #ff000
          acaac  #fec
          tax
          tsega  psa255                    ;want to go to PSA segment 255
          tam
          lbr    psa255                    ;long branch there
*************************************************************
* psa255
*
* Set the A, B and X registers then LBR back to the original
* routine which first called psa33.
*************************************************************
          aorg   #ff000
          segmnt
psa255    tca    10                        ;set A=a, B=b, X=c
          tab
          ibc
          tcx    12
* 0xff006
* A=a, B=b, X=c
          cla                              ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn                   ;want to go to PSA segment 0
          tam
          lbr    longrtn                   ;long branch there
```

```
*************************************************************
* ssa37
*
* Set the A, B and X registers then BRA back to the PSA segment
* zero.
*************************************************************

          aorg   #128412

ssa37     tca    13                          ;set A=d, B=e, X=f
          tab
          ibc
          tcx    15

* 0x128418
* A=d, B=e, X=f

* Return to PSA region and original CALL

          cla                                ;get address of return in PSA 0
          acaac  longrtn                     ;branch there, ignoring SSA value
          bra
```

Appendix F

Pseudo-CALL Instruction

This appendix contains the code for one way of accessing a subroutine from different locations in the PSA address space. It also demonstrates a pseudo-CALL instruction operating in the SSA region. The SSA routine is located above 1M byte, therefore the EMU30 requires a memory expansion card in order to run this program.

Topic **Page**

F.1 Pseudo-CALL Instruction

SPCALL.ASM

The following code shows one way of accessing a subroutine from different locations in the PSA address space. It also demonstrates a pseudo-CALL instruction operating in the SSA region. Note that the SSA routine is located above 1M byte, therefore the EMU30 will require a memory expansion card in order to run this program.

```
          call   jump7     ;jump to PSA segment 7, #07000
          call   jump255   ;jump to PSA segment 255, #ff000
          call   jumpssa   ;jump to SSA segment 33, #108000
loophere  br     loophere
```

The program segment address (PSA) register is set to the segment address of the psa7 routine. This is achieved using the TSEGA command, which extracts the 8-bit program segment from a given address. The LBR instruction branches across program segments to the address offset, psa7, within the seventh segment.

```
*
* Jump to PSA 7 and come back
*
jump7     cla
          acaac  #fec      ;point to PSA
          tax
          tsega  psa7      ;get segment number
          tam              ;set PSA
          lbr    psa7      ;this MUST follow the TAM
```

In PSA segment 7 the A, B and X registers are set to 1, 2 and 3 respectively.

```
************************************************************
* psa7
*
* Set the A, B and X registers then LBR back to the calling
* routine in PSA segment 0.
************************************************************

          aorg   #7000
          segmnt
psa7      tca    1         ;set A=1, B=2, X=3
          tab
          ibc
          tcx 3

* 0x07006
* A=1, B=2, X=3
```

The divide routine in the second PSA segment (at #2000) is then called.

```
          tca    6         ;do 10 div 6
          tab
          call   div7
```

In order to reach a subroutine in a different segment it is necessary to LBR there. In this case, because the divide routine is called from other locations as well, the segment number of the calling segment must be saved in callseg, an 8-bit RAM variable.

```
*
* A subroutine to call the divide routine at #2000.  Need to branch
* there then branch back.  Do a RETN to get back to the original
* calling point in this segment.
*
div7      tsega  psa7            ;set return segment
          tamd   callseg
          cla
          acaac  #fec            ;point to PSA
          tax
          tsega  divide          ;get segment offset
          tam                    ;set PSA
          lbr    divide          ;go and divide in PSA 2
```

The program counter is then set to #2000 and the division (10 by the contents of B) performed.

```
*************************************************************
* divide
*
* Divide A by B and leaves the remainder in B.  The A register
* is corrupted when doing a LBR here and back, so the routine
* is not much use.
*************************************************************

          aorg   #2000
          segmnt
divide    tca    10              ;do 10 div B
          clx                    ;reset counter
divloop   sbaan                  ;do A -= B
          sbr    divdone         ;if negative we have finished
          ixc                    ;else increment counter
          sbr    divloop
          sbr    divloop
divdone   abaac                  ;get remainder in A
          xbx                    ;put answer in A and remainder in B
          xba
*
* Set a breakpoint here at #200b to see the result.
* 10 div X gives A, remainder B
*
```

In order to return to the correct program segment, a branch is taken to the zeroth segment where the return address is calculated.

```
          cla                    ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  divrtn          ;want to go to PSA segment 0
          tam
          lbr    divrtn          ;long branch there
```

The value of callseg is examined and the return address (13 bit offset inside the appropriate segment) then used in the LBR instruction. Another way of achieving the same affect would be to store the return address in RAM, and branch there directly from the divide routine.

```
divrtn    tmad   callseg            ;see which segment did the call


*
* This code decides where to return to, ie which PSA segment.
* The 'divide' routine at #2000 always returns here, then we
* have to return to the place that called 'divide' in the first
* place.  Either PSA 0, 7 or 255.
*
          anec   #00                ;PSA 0?
          sbr    dr7

* Return to current segment

          retn                      ;yes, just return

dr7       anec   #07                ;PSA 7?
          sbr    dr255

* Return to PSA 7

          cla
          acaac  #fec               ;point to PSA
          tax
          tsega  div7rtn            ;get segment offset
          tam                       ;set PSA
          lbr    div7rtn            ;go back to #7000

dr255     anec   #ff                ;PSA 255?
          sbr    drquit

* Return to PSA 255

          cla
          acaac  #fec               ;point to PSA
          tax
          tsega  div255rtn          ;get segment offset
          tam                       ;set PSA
          lbr    div255rtn          ;go back to #ff000

* Give up and return to PSA 0

drquit    retn                      ;if in doubt, return
```

In this case the return address is div7rtn in the seventh PSA segment. Executing a RETN here returns from the original CALL DIV7 instruction. Program execution continues, and returns to the zeroth segment.

```
div7rtn   retn                      ;get here with a LBR from PSA 2
```

In order to return to the zeroth segment the PSA is set to 0, and a LBR executed back to the segment from which psa7 was called.

```
          cla                       ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn            ;want to go to PSA segment 0
          tam
          lbr    longrtn            ;long branch there
```

The longrtn label is in PSA segment zero.

```
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn                      ;do a long branch back here, then return
```

This returns to the main program loop.

```
          call   jump255            ;jump to PSA segment 255, #ff000
          call   jumpssa            ;jump to SSA segment 33, #108000
loophere  br     loophere
```

The procedure for branching to PSA 255 is the same as the branch to PSA 7, except the PSA is obviously set to 255 rather than 7. The code is virtually the same apart from the fact that the segment numbers are different. After jumping to PSA 255, jumping to the divide routine, jumping to PSA 0, jumping back to PSA 255, and then finally jumping back to PSA 0, the main program loop is reached once again.

```
          call   jumpssa            ;jump to SSA segment 33, #108000

loophere  br     loophere
```

The procedure for branching to a location in the speech segment address (SSA) space is different. The SSA register is set to the required value, 33 in this case, but to reach an address above 1M byte the BRA instruction must be used. Setting the MSB (bit 15) high forces the SSA register to be used in the calculation of the new program counter value. This gives an actual address of:

```
   (#7fff & BRA address) + (SSA << 15)
```

So here the new address is simply (#0000 + 33 << 15), which equals #108000.

```
*
* Jump to SSA 33 and return
*
jumpssa   cla
          acaac  #fed               ;point to SSA
          tax
          tca    (ssa33/SSASEG)
          tam                       ;set SSA to 33rd segment

          cla
          acaac  #800
          sala4                     ;set MSB #8000 to force use of SSA
          bra                       ;branch to #108000
```

In SSA segment 33 the A, B and X registers are set to 1, 2 and 3 respectively.

```
*************************************************************
* ssa33
*
* Some executable code at a high address in ROM, above 1MByte.
* Sets the A B X registers to various values and does a 'CALL'
* to a routine, before returning to PSA 0.
* Only BRA can be used to move around in this region (> 1MB) so
* writing code will be tricky.
*************************************************************

          aorg   #108000             ;SSA 33

ssa33     tca    1                      ;set A=1, B=2, X=3
          tab
          ibc
          tcx    3

* #108006
* A=1, B=2, X=3
```

The BR, SBR, LBR and CALL instructions cannot be used in the SSA address space. Therefore the only way to branch around is to use BRA. A pseudo-CALL can be performed by storing the return address in RAM.

```
*
* Prepare to do a call to a subroutine, without using CALL
*
          tca    11                     ;want to return 11 bytes ahead
          acaac  ($&#0fff)              ;current offset
          tamd   ssastack               ;save as a 12 bit address

          tca    #80                    ;equivalent of a CALL instruction
          sala4
          sala4
          acaac  (highcall&#0fff)
          bra                           ;do the 'CALL'
```

The highcall routine sets the A, B and X registers to 4, 5 and 6 respectively.

```
*
* A subroutine which returns, without using RETN
*

highcall  tca    4                      ;set A=4, B=5, X=6
          tab
          ibc
          tcx    6

* 0x108f06
* A=4, B=5, X=6
```

The return process gets the 12-bit address offset, adds #8000 so that the most significant bit (bit 15) is set, and then does a BRA. Since bit 15 is high the branch takes account of the value in the SSA register. In other words, the program counter remains in this SSA address space.

```
* Begin RETN process
          tmad   ssastack              ;get 12 bit return offset
          tab
          tca    #80                   ;need to set #8000
          sala4
          sala4
          abaac                        ;set MSB and add address
          bra                          ;return from call
```

The program returns to address #108013, and then sets the A, B and X registers to 7, 8 and 9 respectively.

```
* #108013
* Return to here, A is #8013, B is #13, X is #6
          tca    7                     ;set A=7, B=8, X=9
          tab
          ibc
          tcx    9
* 0x108019
* A=7, B=9, X=9
```

In order to return to the zeroth PSA segment a BRA is executed to the longrtn address. Because bit 15 is low, the SSA register is ignored and the program counter becomes equal to the longrtn address offset inside PSA 0.

```
* Everything is OK so far, now want to return to the PSA 0 space.
* Leave the SSA alone and just branch, this takes us back to the
* zeroth PSA segment.
          cla
          acaac  longrtn               ;assume 12 bit address
          bra
```

The longrtn label is in PSA segment zero.

```
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn                         ;do a long branch back here, then return
```

The program then waits in an infinite loop.

```
loophere  br     loophere
```

Appendix G

Calling Divide from PSA

This appendix contains the code for how to call one routine (divide) from different parts of the PSA space and demonstrates pseudo-CALL in SSA region.

Topic **Page**

G.1 Calling Divide from PSA

```
* spcall.asm
*
* 27th May 1998
*
* Shows how to call one routine (divide) from different
* parts of the PSA space.  Also demonstrates pseudo-CALL
* in SSA region.
*
* jump7   call and branch to PSA 7, #7000
* jump255 call and branch upto PSA 255, #ff000
* jumpssa call and branch upto SSA 33, #108000
*
* Set breakpoints and observe the registers.
*
* ADDRESS    A  B  X
*   200b     1  4  6
*      2  2  4
* 108006     1  2  3
* 108f06     4  5  6
* 108019     7  8  9
*
* Set a breakpoint at #32 which is the finish point.

      OPTION BUNLIST,DUNLIST,PAGEOF

*****************************************************************
* 12 bit RAM variables
*****************************************************************
ssastack  equ    #01                      ;save ssa offset for 'CALL'

*****************************************************************
* 8 bit RAM variables
*****************************************************************
callseg   equ    #20                      ;PSA segment of calling routine

*****************************************************************
* Constants
*****************************************************************
PSASEG    equ    #1000                    ;PSA segment size
SSASEG    equ    #8000                    ;SSA segment size

*****************************************************************
*    General Purpose Constant Definitions
*****************************************************************
*
*   DEVICE CONSTANTS
*
MAX_RAM      EQU    #EF                   ;Highest RAM location
*
*   I/O Port Definitions
*
INPUT_A      EQU    #FC
PULLUP_A     EQU    #FD
DIRECT_A     EQU    #FE
OUTPUT_A     EQU    #FF
```

```
INPUT_B       EQU    #F8
PULLUP_B      EQU    #F9
DIRECT_B      EQU    #FA
OUTPUT_B      EQU    #FB

INPUT_C       EQU    #F4
PULLUP_C      EQU    #F5
DIRECT_C      EQU    #F6
OUTPUT_C      EQU    #F7

INPUT_D       EQU    #F0
PULLUP_D      EQU    #F1
DIRECT_D      EQU    #F2
OUTPUT_D      EQU    #F3

*-------------------------------------------------------------
*   Start of program
*-------------------------------------------------------------
*     AORG   #0000
*************************************************************
*   Interrupt vectors
*************************************************************
          AORG   #0010
          SBR INT2_00       ;Timer Underflow, PCM=0, LPC=0
          SBR    INT2_00                ;Timer Underflow, PCM=0, LPC=0
          SBR    INT2_01                ;Timer Underflow, PCM=0, LPC=1
          SBR    INT2_01                ;Timer Underflow, PCM=0, LPC=1
          SBR    INT2_10                ;Timer Underflow, PCM=1, LPC=0
          SBR    INT2_10                ;Timer Underflow, PCM=1, LPC=0
          SBR    INT2_11                ;Timer Underflow, PCM=1, LPC=1
          SBR    INT2_11                ;Timer Underflow, PCM=1, LPC=1
          SBR    INT1_00                ;Pin (B1) goes low interrupt
          SBR    INT1_00                ;Pin (B1) goes low interrupt
          SBR    INT1_01                ;Clock interrupt, PCM=0, LPC=1
          SBR    INT1_01                ;Clock interrupt, PCM=0, LPC=1
          SBR    INT1_10                ;Clock interrupt, PCM=1, LPC=0
          SBR    INT1_10                ;Clock interrupt, PCM=1, LPC=0
          SBR    INT1_11                ;Clock interrupt, PCM=1, LPC=1
          SBR    INT1_11                ;Clock interrupt, PCM=1, LPC=1

INT1_10
INT1_01
INT2_10
INT2_00
INT2_01
INT2_11
INT1_00
INT1_11   CLA
          RETI
```

```
*********************************************************
*   Do program INITs
*********************************************************
GO        CLA                           ;Clear Memory mapped Registers
          intgr
          ACAAC  #FF0
          TAX
          CLA
          TAMIX
          TAMIX
          TAMIX
          TAMIX
          call   jump7                  ;jump to PSA segment 7, #07000
          call   jump255                ;jump to PSA segment 255, #ff000
          call   jumpssa                ;jump to SSA segment 33, #108000

loophere  br  loophere
*
* Jump to PSA 7 and come back
*
jump7  cla
       acaac     #fec                   ;point to PSA
       tax
       tsega     psa7                   ;get segment number
       tam                              ;set PSA
       lbr psa7                         ;this MUST follow the TAM
*
* Jump to PSA 255 and come back
*
jump255   cla
          acaac  #fec                   ;point to PSA
          tax
          tsega  psa255                 ;get segment number
          tam                           ;set PSA
          lbr    psa255                 ;this MUST follow the TAM
*
* Jump to SSA 33 and return
*
jumpssa   cla
          acaac  #fed                   ;point to SSA
          tax
          tca    (ssa33/SSASEG)
          tam                           ;set SSA to 33rd segment
          cla
          acaac  #800
          sala4                    ;set MSB #8000 to force use of SSA
          bra                      ;branch to #108000
*
* A generic RETN statement used by all returns from LBR routines
*
longrtn   retn                     ;do a long branch back here, then return
```

```
*
* This code decides where to return to, ie which PSA segment.
* The 'divide' routine at #2000 always returns here, then we
* have to return to the place that called 'divide' in the first
* place.  Either PSA 0, 7 or 255.
*
divrtn    tmad   callseg            ;see which segment did the call

          anec   #00                ;PSA 0?
          sbr    dr7

* Return to current segment

          retn                      ;yes, just return

dr7       anec   #07                ;PSA 7?
          sbr    dr255

* Return to PSA 7

          cla
          acaac  #fec               ;point to PSA
          tax
          tsega  div7rtn            ;get segment offset
          tam                       ;set PSA
          lbr    div7rtn            ;go back to #7000

dr255     anec   #ff                ;PSA 255?
          sbr    drquit

* Return to PSA 255

          cla
          acaac  #fec               ;point to PSA
          tax
          tsega  div255rtn          ;get segment offset
          tam                       ;set PSA
          lbr    div255rtn          ;go back to #ff000

* Give up and return to PSA 0

drquit    retn                      ;if in doubt, return

*********************************************************
* PSA Segment 1 – equivalent to aorg #1000
*
* Program Entry Point
*
* The program in the internal ROM branches to here.  Clear
* the RAM and do some initialization.
*********************************************************

          aorg   #1000
          segmnt
```

```
          CLA
          tca    #08                        ;set D3 as output
          tamd   direct_d
          cla
          tamd   pullup_d
          tamd   output_d

          CLA
          ACAAC  #FEE                       ;set Internal ROM Page Control
          TAX                               ;  (IRPC) so we can access
          TCA    1                          ;  lower 4K of external ROM
          TAM

          TMAD   0

          CLA                               ;Initialize mode register
          TAMODE

          CLX
RAM_LOOP  TAMIX                             ;Initialize All RAM to zeros
          XGEC   MAX_RAM
          SBR    RAM_EXIT
          SBR    RAM_LOOP

RAM_EXIT  TAM                               ;initialize last RAM location (FF)

          TCA    #AA
          TAMD   DIRECT_A                   ;set even bits as inputs
          TAMD   DIRECT_B                   ; and odd as outputs
          TAMD   DIRECT_C
          TAMD   OUTPUT_A
          TAMD   OUTPUT_B
          TAMD   OUTPUT_C
          TCA    #55
          TAMD   PULLUP_A                   ;turn pullups on for inputs
          TAMD   PULLUP_B
          TAMD   PULLUP_C

          CLA
          ACAAC  #FEC
          TAX
          TCA    0
          TAM
*
* Now do a long branch back to the start of the external ROM
* program, where we do some branches in the PSA and SSA.
*
          LBR    GO
*****************************************************************
* divide
*
* Divide A by B and leaves the remainder in B.  The A register
* is corrupted when doing a LBR here and back, so the routine
* is not much use.
*****************************************************************

          aorg   #2000
          segmnt
```

```
divide    tca    10                        ;do 10 div B
          clx                              ;reset counter
divloop   sbaan                            ;do A -= B
          sbr    divdone                   ;if negative we have finished
          ixc                              ;else increment counter
          sbr    divloop
          sbr    divloop
divdone   abaac                            ;get remainder in A
          xbx                              ;Put answer is in A and remainder in B
          xba
*
* Set a breakpoint here at #200b to see the result.
* 10 div X gives A, remainder B
*
          cla                              ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  divrtn                    ;want to go to PSA segment 0
          tam
          lbr    divrtn                    ;long branch there

*****************************************************************
* psa7
*
* Set the A, B and X registers then LBR back to the calling
* routine in PSA segment 0.
*****************************************************************

          aorg   #7000
          segmnt

psa7      tca    1                         ;set A=1, B=2, X=3
          tab
          ibc
          tcx    3

* 0x7006
* A=1, B=2, X=3

          tca    6                         ;do 10 div 6
          tab
          call   div7

          cla                              ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn                   ;want to go to PSA segment 0
          tam
          lbr    longrtn                   ;long branch there

*
* A subroutine to call the divide routine at #2000.  Need to branch
* there then branch back.  Do a RETN do get back to the original
* calling point in this segment.
*
```

```
div7      tsega  psa7                    ;set return segment
          tamd   callseg
          cla
          acaac  #fec                    ;point to PSA
          tax
          tsega  divide                  ;get segment offset
          tam                            ;set PSA
          lbr    divide                  ;go and divide in PSA 2

div7rtn   retn                           ;get here with a LBR from PSA 2
*************************************************************
* psa255
*
* Set the A, B and X registers then LBR back to the original
* routine which first called psa17.
*************************************************************

          aorg   #ff000
          segmnt

psa255    tca    4                       ;set A=4, B=5, X=6
          tab
          ibc
          tcx    6

* 0xff006
* A=4, B=5, X=6

          tca    4                       ;do 10 div 4
          tab
          call   div255                  ;call divide routine

          cla                            ;now set up PSA for LBR back to start
          acaac  #fec
          tax
          tsega  longrtn                 ;want to go to PSA segment 0
          tam
          lbr    longrtn                 ;long branch there

div255 tsega  psa255                     ;set return segment
          tamd   callseg
          cla
          acaac  #fec                    ;point to PSA
          tax
          tsega  divide                  ;get segment offset
          tam                            ;set PSA
          lbr    divide                  ;go and divide in PSA 2

div255rtn retn
*************************************************************
* ssa33
*
* Some executable code at a high address in ROM, above 1MByte.
* Sets the A B X registers to various values and does a 'CALL'
* to a routine, before returning to #f100 region.
* Only BRA can be used to move around in this region (> 1MB) so
* writing code will be tricky.
*************************************************************
```

```
          aorg   #108000                    ;SSA 33
ssa33     tca    1                         ;set A=1, B=2, X=3
          tab
          ibc
          tcx    3
* #108006
* A=1, B=2, X=3
*
* Prepare to do a call to a subroutine, without using CALL
*
          tca    11                        ;want to return 11 bytes ahead
          acaac  ($&#0fff)                 ;current offset
          tamd   ssastack                  ;save as a 12 bit address
          tca    #80                       ;equivalent of a CALL instruction
          sala4
          sala4
          acaac  (highcall&#0fff)
          bra                              ;do the 'CALL'
* #108013
* Return to here, A is #8013, B is #13, X is #6
          tca    7                         ;set A=7, B=8, X=9
          tab
          ibc
          tcx    9
* 0x108019
* A=7, B=9, X=9
* Everything is OK so far, now want to return to the PSA 0 space.
* Leave the SSA alone and just branch, this takes us back to the
* zeroth PSA segment.
          cla
          acaac  longrtn                   ;assume 12 bit address
          bra
          aorg   #108f00
*
* A subroutine which returns, without using RETN
*
highcall  tca    4                         ;set A=4, B=5, X=6
          tab
          ibc
          tcx    6
* 0x108f06
* A=4, B=5, X=6
* Begin RETN process
          tmad   ssastack                        ;get 12 bit return offset
          tab
          tca    #80                             ;need to set #8000
          sala4
          sala4
          abaac                                  ;set MSB and add address
          bra                                    ;return from call
```

Appendix H

The CSM30003 Catalog Device

This appendix contains information on the CSM30003 catalog device.

Topic **Page**

H.1 The CSM30003 Catalog Device

CSM30003 is the part number of TI's catalog part that is based on the MSP50C30. This catalog part is designed to have all of the software and speech data reside in an external ROM. The code for this device is written to allow interfacing with any size standard ROM or EPROM up to 64 Mbits. The following list summarizes the key benefits of this device.

- Supports rapid prototyping and ramp-up to production, because this catalog part can be used with readily available ROMs.
- Supports long speech duration with a wide range of synthesizers that allow speech quality and system cost to be optimized to the application.
- Interfaces easily to peripherals because of the large number of general-purpose I/O terminals.
- Enables complex game play because of microcontroller functionality.
- Saves cost by avoiding the need for external crystal and amplifier.

In addition to the CSM30003 catalog part, customers can develop their own custom codes based on the MSP50C30, or they can develop custom codes based on other MSP50C3x family members (e.g., MSP50C32, MSP50C33, MSP50C34).

H.2 CSM30003 Functionality

The CSM30003 is designed to connect to an external ROM. The 23 address lines of the CSM30003 connect to the address lines of the external ROM or EPROM. The 8 data lines of the CSM30003 connect to the data lines of the external ROM or EPROM. Port D3 of the CSM30003 is dedicated to enabling and disabling the external ROM or EPROM; and should be connected to the ENA line of the ROM or EPROM in most applications.

Immediately upon WAKEUP or $\overline{\text{INIT}}$, the CSM30003 programs the D3 port to a totem-pole output in a low-state. This output should be used to drive the ENA line of the external ROM. The program then branches to Address 0x1000h in the external ROM for an $\overline{\text{INIT}}$ or branches to address 0x1002h in the external ROM for a WAKEUP and begins executing the program code located at that address. The RAM is not cleared.

A branch to address 0x0043h of the internal ROM places the device into a low power state by:

1) Setting Port D3 to a high state to place the external ROM into a low power state.

2) Clearing the IOMAP bit in mode register 2

3) Executing a SETOFF to place the CSM30003 into a low power state.

CSM30003 Port D3

Port D3 is programmed as a totem-pole output low state. Take care not to change this state in software programmed to the external ROM. If this state is disturbed, the external ROM may be placed into a low-power state, disturbing operation of the system.

CSM30003 Branching to External ROM or EPROM

When the CSM30003 branches to the external ROM or EPROM, the addresses in the range between 0x0000h and 0x0FFFh are mapped into the internal ROM. To access this address range in the external ROM or EPROM it is necessary to write a 1 into the IRPC register.

Appendix I

MSP50C3x Family Data Sheet

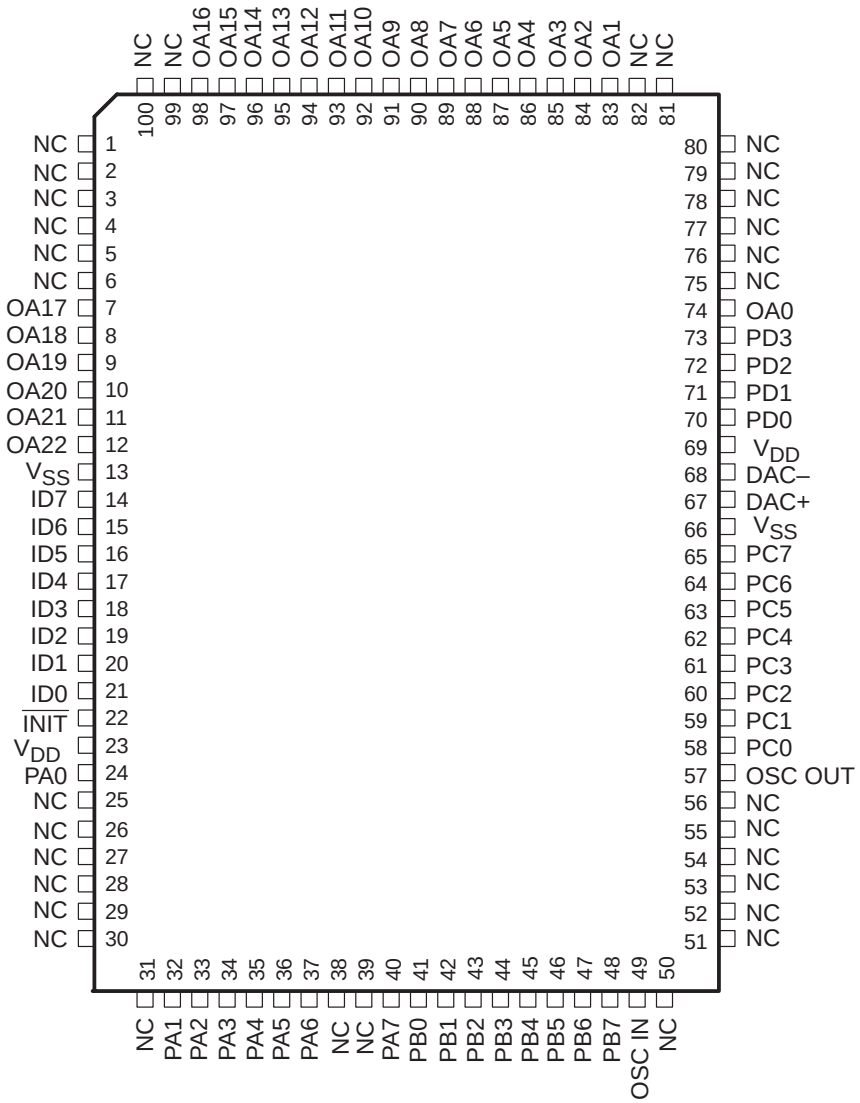
This appendix contains data sheet information for the MSP50C3x mixed-signal processor family.

- **Interface to External ROM/EPROM (Up to 8 MBytes)**
- **8-Bit Microprocessor with 61 instructions**
- **32 Twelve-Bit Words and 992 Bytes of RAM**
- **4K Internal ROM**
- **3.3V to 6.5V CMOS Technology for Low Power Dissipation**
- **28 Software-Configurable I/O Lines**
- **10-kHz or 8-kHz Speech Sample Rate**

description

The MSP50C30 combines an 8-bit microprocessor, two speech synthesizers, ROM, RAM, and I/O in a low-cost single-chip system. The architecture uses the same arithmetic logic unit (ALU) for the two synthesizers and the microprocessor, thus reducing chip area and cost and enabling the microprocessor to do a multiply operation in 0.8 μs. The MSP50C30 features two independent channels of linear predictive coding (LPC), which synthesize high-quality speech at a low data rate. Pulse-code modulation (PCM) can produce music or sound effects. For more information, see the MSP50C30 User's Guide (TI literature number SPSU012).

PJM PACKAGE
(TOP VIEW)



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

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 (MSP50C30)

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

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

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 ($\overline{INIT}$)	V_{DD} = 3.5 V			2		V
		V_{DD} = 6 V			3.4		
V_{T-}	Negative-going threshold voltage ($\overline{INIT}$)	V_{DD} = 3.5 V			1.6		V
		V_{DD} = 6 V			2.3		
V_{hys}	Hysteresis ($V_{T+} - V_{T-}$) ($\overline{INIT}$)	V_{DD} = 3.5 V			0.4		V
		V_{DD} = 6 V			1.1		
I_{lkg}	Input leakage current (except for OSC IN)					2	µA
$I_{standby}$	Standby current ($\overline{INIT}$ low, SETOFF)					10	µA
I_{DD}†	Supply current	V_{DD} = 3.3 V,			2.1		mA
		V_{DD} = 5 V,			3.1		
		V_{DD} = 6 V,			4.5		
I_{OH}	High-level output current (PA, PB)	V_{DD} = 3.3 V,	V_{OH} = 2.75 V	–4	–12		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V	–5	–14		
		V_{DD} = 6 V,	V_{OH} = 5.5 V	–6	–15		
		V_{DD} = 3.3 V,	V_{OH} = 2.2 V	–8	–20		mA
		V_{DD} = 5 V,	V_{OH} = 3.33 V	–14	–40		
		V_{DD} = 6 V,	V_{OH} = 4 V	–20	–51		
I_{OL}	Low-level output current (PA, PB)	V_{DD} = 3.3 V,	V_{OL} = 0.5 V	5	9		mA
		V_{DD} = 5 V,	V_{OL} = 0.5 V	5	9		
		V_{DD} = 6 V,	V_{OL} = 0.5 V	5	9		
		V_{DD} = 3.3 V,	V_{OL} = 1.1 V	10	19		mA
		V_{DD} = 5 V,	V_{OL} = 1.67 V	20	29		
		V_{DD} = 6 V,	V_{OL} = 2 V	25	35		
I_{OH}	High-level output current (D/A)	V_{DD} = 3.3 V,	V_{OH} = 2.75 V	–30	–50		mA
		V_{DD} = 5 V,	V_{OH} = 4.5 V	–35	–60		
		V_{DD} = 6 V,	V_{OH} = 5.5 V	–40	–65		
		V_{DD} = 3.3 V,	V_{OH} = 2.3 V	–50	–90		mA
		V_{DD} = 5 V,	V_{OH} = 4 V	–90	–140		
		V_{DD} = 6 V,	V_{OH} = 5 V	–100	–150		
I_{OL}	Low-level output current (D/A)	V_{DD} = 3.3 V,	V_{OL} = 0.5 V	50	80		mA
		V_{DD} = 5 V,	V_{OL} = 0.5 V	70	90		
		V_{DD} = 6 V,	V_{OL} = 0.5 V	80	110		
		V_{DD} = 3.3 V,	V_{OL} = 1 V	100	140		mA
		V_{DD} = 5 V,	V_{OL} = 1 V	140			
		V_{DD} = 6 V,	V_{OL} = 1 V	150			
	Pullup resistance	Resistors selected by software and connected between terminal and V_{DD}		10	20	50	kΩ
$f_{osc(low)}$	Oscillator frequency‡	V_{DD} = 5 V,	T_A = 25°C, Target frequency = 15.36 MHz	14.89	15.36	15.86	MHz
$f_{osc(high)}$	Oscillator frequency‡	V_{DD} = 5 V,	T_A = 25°C, Target frequency = 19.2 MHz	18.62	19.2	19.7	MHz

† Operating current assumes all inputs are tied to either V_{SS} or V_{DD} with no input currents due to programmed pullup resistors. The DAC output and other outputs are open circuited.

‡ The frequency of the internal clock has a temperature coefficient of approximately –0.2 %/°C and a V_{DD} coefficient of approximately ±1%/V.

switching characteristics

PARAMETER			TEST CONDITIONS			MIN	NOM	MAX	UNIT
t_r	Rise time	PA, PB, PC, PD, D/A	V_{DD} = 3.3 V,	C_L = 100 pF,	10% to 90%		50		ns
		OA	V_{DD} = 3.3 V,	C_L = 50 pF,	10% to 90%		50		
t_f	Fall time	PA, PB, PC, PD, D/A	V_{DD} = 3.3 V,	C_L = 100 pF,	10% to 90%		50		ns
		OA	V_{DD} = 3.3 V,	C_L = 50 pF,	10% to 90%		50		

timing requirements

			MIN	MAX	UNIT
Initialization					
t_{INIT}	$\overline{INIT}$ pulsed low while the MSP50x3x has power applied (see Figure 1)		1		µs
Wakeup					
$t_{su(wakeup)}$	Setup time prior to wakeup terminal negative transition (see Figure 2)		1		µs
External Interrupt					
$t_{su(interrupt)}$	Setup time prior to B1 terminal negative transition (see Figure 3)	f_{clock} = 15.36 MHz	1		µs
		f_{clock} = 19.2 MHz	1.5		
Writing (Slave Mode)					
$t_{su1(B1)}$	Setup time, B1 low before B0 goes low (see Figure 4)		20		ns
$t_{su(d)}$	Setup time, data valid before B0 goes high (see Figure 4)		100		ns
$t_{h1(B1)}$	Hold time, B1 low after B0 goes high (see Figure 4)		20		ns
$t_{h(d)}$	Hold time, data valid after B0 goes high (see Figure 4)		30		ns
t_w	Pulse duration, B0 low (see Figure 4)		100		ns
t_r	Rise time, B0 (see Figure 4)			50	ns
t_f	Fall time, B0 (see Figure 4)			50	ns
Reading (Slave Mode)					
$t_{su2(B1)}$	Setup time, B1 before B0 goes low (see Figure 5)		20		ns
$t_{h2(B1)}$	Hold time, B1 after B0 goes high (see Figure 5)		20		ns
t_{dis}	Output disable time, data valid after B0 goes high (see Figure 5)		0	30	ns
t_w	Pulse duration, B0 low (see Figure 5)		100		ns
t_r	Rise time, B0 (see Figure 5)			50	ns
t_f	Fall time, B0 (see Figure 5)			50	ns
t_d	Delay time for B0 low to data valid (see Figure 5)			50	ns
External ROM					
$t_{a(ROM)}$	ROM access time			400	ns

PARAMETER MEASUREMENT INFORMATION

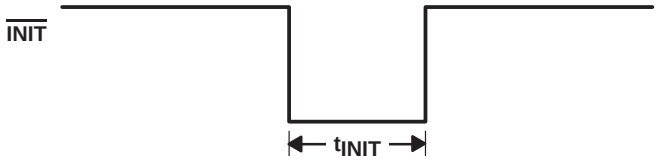


Figure 1. Initialization Timing Diagram

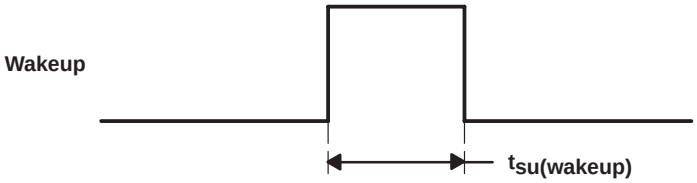


Figure 2. Wakeup Terminal Setup Timing Diagram

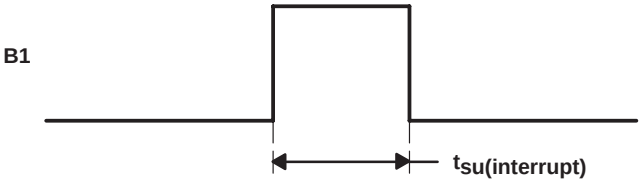


Figure 3. External Interrupt Terminal Setup Timing Diagram

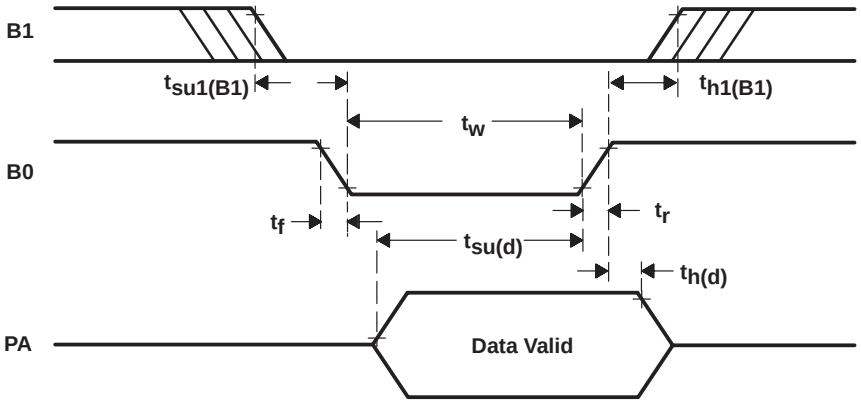


Figure 4. Write Timing Diagram (Slave Mode)

PARAMETER MEASUREMENT INFORMATION

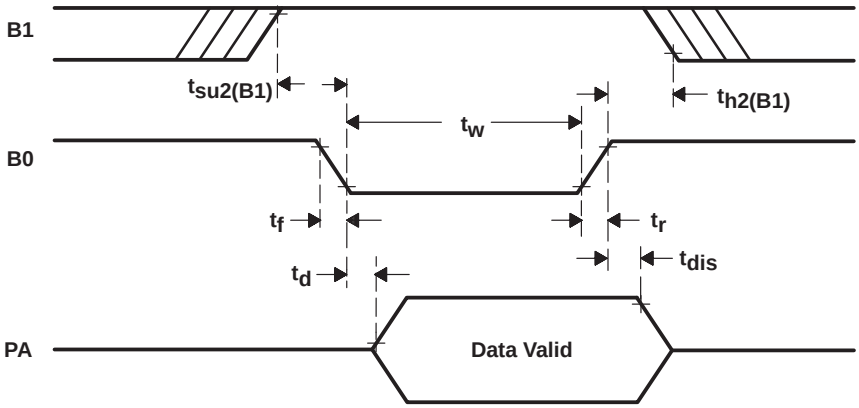
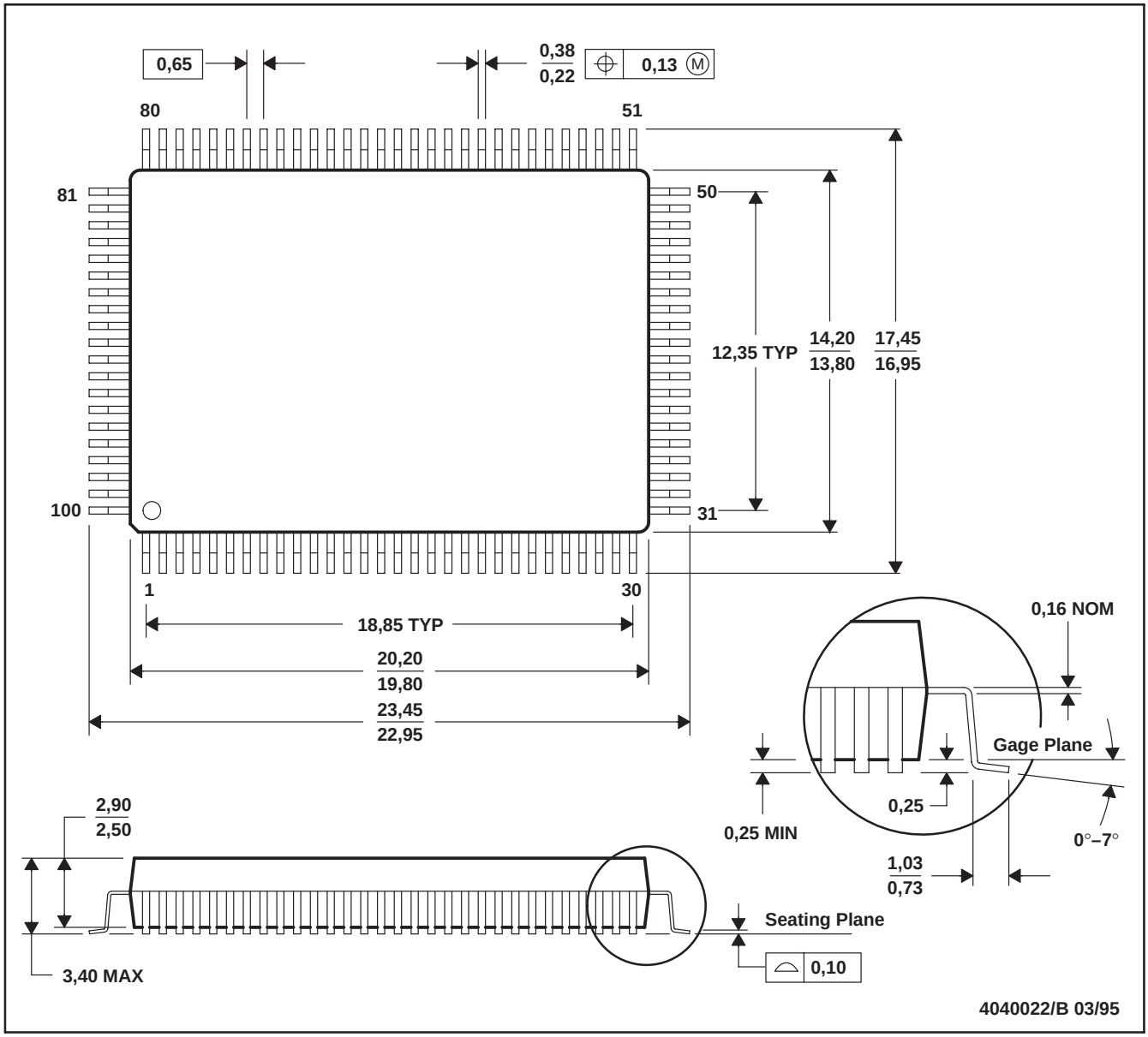


Figure 5. Read Timing Diagram (Slave Mode)

MECHANICAL DATA

PJM (R-PQFP-G100) **PLASTIC QUAD FLATPACK**



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS-022