



RSC-364

Development Kit Manual

with Sensory Speech 6 Technology

table of contents

Chapter 1 - Overview

Introduction	3
What's New in the RSC-364 Development Kit	3
Summary	4
Getting Started	6
Software Installation	6
Copying Files	7
Setting up the Environment	8
Configuring the Serial port of your PC	9
Software Contents	9
Assembling, Linking, and Loading Modules and Programs	12
Memory Module Configuration	14
Basic Configuration Jumpers	14
Default Configuration	14
DIP Socket U1	15
RAM Allocation	17
Allocating ROM for Speech and Weights	18
Data Space Placement of Speech and Weights	18
When 64K Code Space is not enough	19
Large Data Space Memory Model and Rapid Vocabulary Prototyping	19
RSC-264T/364 Simulator	19
Notes and Suggestions	20
A First Exercise	20
Debugging Help	20
Named Segments	20
Recognition Speed	20
Shoehorning	21
Error Messages	21
Manual Contents	21
Development Kit Contents	22
Software Development Library Contents	23
Technology Library Contents	23
Sample Program User Instructions	23
Using the Samples	24
Running a Sample Program	24
Sample Instructions	25

Chapter 2 - technology library

Important Notice	33
Recognition and Synthesis Vocabulary Usage	33
Using Speech Synthesis	33
Initialization	33
Includes	33
Linking	33

Filters	34
Talk Handler	34
Sentence-level Speech Synthesis	36
Macros	36
Sample Code	37
Using Pattern Generation	38
Initialization	39
Includes	39
Linking	39
Macros	39
Parameters	41
Code Flow	42
Silence Levels	42
Interrupts	42
Memory Handler	42
Results	42
Sample Code	42
Using Speaker Independent Speech Recognition	44
Common Pattern Generation Technology	44
Initialization	44
Includes	45
Linking	45
SI Memory Handler	45
Macros	46
Code Flow	47
Results	48
Sample Code	48
Using Speaker Dependent Speech Recognition	50
Common Pattern Generation Technology	50
Initialization	50
Includes	51
Linking	51
Pattern Memory Usage	51
Macros	52
Additional Parameters	54
Training Code Flow	54
Recognition Code Flow and Results	55
Sample Code	56
Using Speaker Verification	58
Common Pattern Generation Technology	58
Comparison with Speaker Dependent Speech Recognition	59
Initialization	59
Includes	59
Linking	60
Pattern Memory Usage	60
Macros	60
Expanded RECOGSV Macro Notes	62
Training Code Flow	63
Recognition Code Flow and Results	63

Sample Code	63
Using Continuous Listening	65
Comparison of CL and WS	65
Continuous Listening using Speaker Independent/Speaker Dependent/Verification	66
Similarity to Common Pattern Generation Technology	67
Similarity to Recognition Technologies	67
Initialization	67
Includes	67
Linking	67
Macros	67
Parameters	68
Performance	69
Code Flow	69
Sample Code	70
Using Wordspot SD Speech Recognition	71
Comparison of CL and WS	71
WS Recognition with SD technology	72
Initialization	73
Includes	73
Linking	73
Pattern Memory Usage	74
Macros	74
Parameters	75
Recognition Code Flow and Results	75
Sample Code	75
Using Dual Recognition Technology	77
Common Pattern Generation Technology	78
Initialization	78
Includes	78
Linking	78
DRT/SI Memory Handler	79
Macros	79
Training Code Flow	80
Recognition Code Flow and Results	80
Sample Code	80
Using Voice Record and Playback	82
Initialization	82
Includes	82
Linking	82
Thresholding	83
Simultaneous Pattern Generation and Voice Recording	83
Compression	84
Macros	85
Parameters	88
Sample Code	88
Using Touch-Tone Synthesis	90
Initialization	90
Includes	91
Linking	91

TTone Handler	91
Macros	92
Sample Code	93
Specifications	96
ROM Memory Requirements of Technology Code	97

Chapter 3 - Software development

1. Software and Linguistics Interactions	101
Synthesis	101
Recognition (see also Using Speaker Independent Speech Recognition)	101
Use of Priors	101
Noise Robustness; NOTA; Yeah-Yep-Yes and No-Nope	102
2. Using the RSC-364 Memory Map	102
Overview of 364 Memory Map	102
User Memory Map Description	103
Allocating Memory	107
The GLOBAL_SRET Code Segment	107
3. Using On-Chip Pattern Storage	108
Small-Vocabulary SD/SV	108
Wordspot	108
Fast Sequential Pattern Queuing	108
Initialization	109
Includes	109
Linking	109
Macros	109
4. System Configuration and I/O Mapping	110
I/O Mapping (IO.INC)	110
System Configuration (CONFIG.A, CONFIG.INC, CONST364.O)	110
System Initialization (STARTUP.A)	112
Default I/O Configuration	113
5. I/O Usage in RSC-364 Development Kit Summary	114
6. Using Debug Speech	114
Initialization	115
Includes	115
Linking	115
Functions	115
Macros	116
Sample Code	118
7. Using PC Output	118
Initialization	119
Includes	119
Linking	119
Macros	120
Sample Code	121
8. Using Co-Routines for External Memory Access	122
Memory Types	122
IMPORTANT NOTICE	123
The Validate_Template_SDV Function	127

RSC Memory Requirements	128
RSC-364 Co-Routine Memory pointers, usage, etc.	130
Sample Code	131
Memory Co-Routines Using I2C Serial EEPROM	134
Memory Co-routines Using "Page Mode" Flash ("sector program mode" EEPROM)	136
Memory Co-routines Using 4-Mbit SST Flash	137
9. Using the Bootloader	138
10. Building Multi-Bank Systems	142
11. Using the Floating Point Library Routines	145
Initialization	145
Includes	145
Linking	145
Functions	145
Format	146
Accuracy	147
Memory Usage	147
12. Stack Usage	148
13. Include, Link, and Co-Routine Requirements for 6. 0 Technology	148

Chapter 4 - hardware

1. DK264T/364 Motherboard Schematic	153
2. DK264T/364 Motherboard Parts Locator	153
3. DK264T/364 Memory Module Schematic	155
4. DK264T/364 Memory Module Schematic-2	155
5. K264T/364 Memory Module Parts Locator	155
6. DK264T/364 I/O Module Schematic-2	157
7. DK264T/364 I/O Module Parts Locator	159
8. Dialer Keypad Schematic	160
9. Connector Pin Specification Summary	161
A. DK264T/364 Motherboard	162
C. DK264T/364 IO Module	166
10. Adding Volume Control	167
11. Hardware Configuration	167
12. Installing and Removing the Sample Flash	174

Chapter 5 - simulator

1.Introduction	177
User Equipment Requirements	177
Software Description	177
2.Starting the Simulator	177
3.Simulator: Menu Bar	178
Using the Display Menu	179
4.Simulator: Control Window	183
5.Simulator: Source Code List	187
6.Simulator Tutorial	187
Tutorial Steps	188
7.Simulator Operation Notes	189

Chapter 6 - assembler & linker

1. General Reference	.192
Text File Input Syntax	.192
Expressions	.192
Expression Functions	.193
2. Sensory Assembler (SASM2) Reference	.194
Command Line Options	.194
Overall Source Syntax	.195
Directives	.196
The LIBINCL Directive	.197
The END Directive	.197
Storage Definition and Allocation Directives	.198
3. Sensory Linker (CLINK) Reference	.207
Command Line Options	.207
4. Linking Speech and Weights Data	.208
5. Sensory RSC-264T/364 Assembler Extensions	.209

Chapter 7 - upgrade

1. Upgrading from RSC-364, Release 5.0	.214
Summary of Changes	.214
Include Files	.214
Linker Commands	.214
Required Application Changes	.214
Optional Application Changes	.215
Sample Software Conversion	.216
Additional Notes	.218
2. Upgrading from RSC-264T or RSC-164V2	.219

Chapter 8 - databook

Chapter 9 - design notes

Chapter 10 - support

1. RSC-364 Development Kit Support Policy	.239
2. RSC-364 Development Kit Limited Warranty	.239
Limitation of Warranty	.239
Exclusive Remedies	.239
3. Important Notices	.240

List of tables

Table 1 - Frequency tolerance of the output tones for DTMF signaling	96
Electrical Characteristics	97
Register Space usage in the RSC-364	103
I/O usage in rsc-364 development kit summary	114
Individual SD_SV memory handler routines	127
Functions available in the floating point library	146
DK264T/364 Motherboard Schematic	Inside front cover
DK264T/364 Motherboard Parts Locator	154
DK264T/364 Memory Module Schematic 1 (Check the inside the front cover of this manual for this drawing)	Inside front cover
DK264T/364 Memory Module Schematic-2 (Check the inside front cover of this manual for this drawing).	Inside front cover
DK264T/364 Memory Module Parts Locator	156
DK264T/364 I/O Module Schematic-2	158
DK264T/364 I/O Module Parts Locator	159
JP1: Memory Module Interface	162
JP2:10 Module Interface	163
P1: RS232 Interface (DB9 Female)	163
DK264T/364 Memory Module	165
JP1: Motherboard Interface	166
JP2: Keypad Interface	167
DK264T/364 Motherboard	168
DK264T/364 Memory Module	171
Sample and Demo Programs	173

List of figures

Figure 1 Simulator Display	178
Figure 2 All Bank Ram window and edit dialog box	180
Figure 3 Display SFR RAM window and edit dialog box	181
Figure 4 Scratch Pad RAM window and edit dialog box	182
Figure 5 Internal Code List window	183
Figure 6 Breakpoints List Simulator Mode Dialog Box Controls	185

Copyright

(c) Copyright 2000, Sensory, Inc. You may not copy, modify, or translate this document or any part of this document. Nor can you reduce any part of it to any machine-readable form.

Trademarks

Sensory, Voice Dialer, Voice Direct, Sensory Voice Activation, Interactive Speech, Voice Extreme and Voice Password are registered trademarks of Sensory, Inc., and may not be reproduced or cited without permission from Sensory, Inc.

Technical Support

For product support and questions:

Marketing Communications

521 East Weddell Drive

Sunnyvale, CA 94089-2164

Tel 408-744-9000

Technical Support

521 East Weddell Drive

Sunnyvale, CA 94089-2164

Tel 408-744-9000

techsupport@sensoryinc.com

CHAPTER 1 - OVERVIEW

This chapter describes the capabilities and contents of the RSC-364 Development Kit with Sensory Speech™ 6 Technology. The kit provides a variety of tools that enable designers to build advanced applications for the RSC-364. In addition to identifying the hardware, software and documentation components of the kit, this chapter provides a summary of the latest hardware and software features available from Sensory. The chapter guides the user through the process of getting started with the kit and continues with detailed descriptions of memory socket configurations, RAM and ROM allocations, and some helpful notes and suggestions on how to use the kit.

1. Introduction

Welcome to the RSC-364 and the world of low-cost, high-performance speech recognition! The materials in the RSC-364 Development Kit allow a developer to produce innovative, powerful products that combine a high level of system integration, low cost, and leading-edge speech technologies. Creating a successful product with the RSC-364 requires developing application software, electronic hardware, and linguistic content. The RSC-364 Development Kit allows developers to create application software for the RSC-364. It also provides tools to build custom speech synthesis vocabularies with moderate compression levels. For hardware information, refer to the RSC-364 Data Book. For more information regarding linguistics, refer to Design Note on Designing with the Interactive Speech line of Chips, or contact Sensory.

This kit supports development using Sensory technologies for Speaker Independent (SI), Speaker Dependent (SD), Wordspot (WS), and Dual Recognition Technology (DRT) speech recognition, Speaker Verification (SV), speech synthesis and sound effects, Continuous Listening (CL), voice record and playback (RP), and Dual-Tone Multi-Frequency (DTMF) generation ("Touch tones"). This kit provides a demonstration and limited support for another Sensory technology, four-voice music synthesis. Please contact Sensory for assistance in developing applications using music technology.

This Development Kit supports the RSC-364 only. It cannot be used to develop products for the RSC-164 family of chips. The hardware is compatible with developing products for the RSC-264T, but the software is not. The software can also be used to develop RSC-364 applications using Sensory's Demo Unit as the development platform.

The RSC-364 chip provides the following cost-reducing features important for low-cost consumer products:

- Enhanced hardware resources to speed up recognition by 2X.
- Low-voltage operation from 2 alkaline batteries
- Powerdown sleep mode conserves power
- Pulse Width Modulator (PWM) to directly drive a loud speaker
- Memory sufficient for storing six Speaker Dependent or Speaker Verification words on chip
- Integrated microphone amplifier requiring only a few additional passive components

Note: The RSC-364 is NOT software source-compatible with the RSC-164. The RSC-364 can execute programs written for the RSC-264T, but the sources are not directly compatible. See the Upgrades tab for porting information.

This manual assumes the reader is an experienced software developer who understands assembly language programming, embedded systems development methods, relocatable object code, and similar general concepts, but who may not be familiar with the RSC-364 specifically.

What's New in the RSC-364 Development Kit

This section describes the changes since RSC-364 Development Kit release 5.0 (June 1999) in outline form. It will be especially useful to developers upgrading an existing RSC-364 application, but it also offers the first-time developer a preview of some features of the Development Kit. Also see Development Kit Upgrades at the Upgrades tab for details about upgrading from 5.0.

Highlights

- "Wordspot" technology can recognize individual words spoken in fluent context
- "Speech Framing" improves recognition accuracy and speed, simplifies applications
- Quality improvements to Record and Play
- Dramatic speed improvements to Continuous Listening
- Improved pattern generation and SD recognition algorithms
- Improved Flash memory support.
- Sample programs emphasize Flash usage
- Numerous minor feature enhancements and bug fixes
- "Quick Synthesis" tool provides means for the developer to create custom speech synthesis vocabularies
- Music synthesis demonstration and preliminary development support
- Development compatibility with Sensory's Demo Unit

Summary

New Library

A new library, \LIB364\WORDSPOT, contains code to support the Wordspot Speaker Dependent technology. A new section, Using Wordspot SD Speech Recognition, has been added to Chapter 2 of this manual, and a new sample program, WSPOT, has been added to the \SAMPL364\ library.

Hardware

The hardware development platform has been re-configured so that the default configuration matches Sensory's Demo Unit. This allows code to be developed and run on either hardware platform. Additional tools are provided to support transferring code and data to either hardware platform interchangeably.

Pattern Generation

An improved pattern generation algorithm combines the previously separate steps of silence measurement and pattern generation into a continuous process called "speech framing". This greatly simplifies application-level code, and – with other internal improvements -- makes better patterns. A new macro to invoke pattern generation especially for Wordspot is provided. Several old macros have been made obsolete.

Speaker Independent Recognition

The Priors function now allows removing a class from consideration (set its probability to zero) in addition to the previous ability to double its probability. Previous SI weights are not compatible with this new release. The weights tables included in \DATA\WEIGHTS\364 have been re-generated for the new technology. Applications upgrading from 5.0 must have new weights generated.

Speaker Dependent Recognition

The sd_performance parameter has been converted from an assembly-time definition to a run-time parameter. The major recognition algorithm has been re-written to run about 40% faster.

Speaker Verification

No functional improvements. SV gains the same speedup benefit from re-coding the SD algorithm.

Dual Recognition Technology

This technology has been generalized to support vocabularies other than fast digits. The code now handles 24-bit template addresses for flash compatibility. Minor bugs were fixed.

Interrupts, Stop Handlers

All Technology modules have been reviewed, and any unnecessary disabling of interrupts has been removed.

Memory Co-Routine Handlers

No functional changes. The description of the `VALIDATE_TEMPLATE_SDV` function has been improved. See Flash RAM co-routines, below.

Continuous Listening

Incorporation of speech framing makes this technology much more responsive. It now supports SI as well as SD/SV. This was previously called "CL4".

Continuous Listening 5

Support for this technology has been removed. Contact Sensory support for custom applications.

Speech Synthesis

The Talk code now preserves the IMR register contents.

A new tool, Quick Synthesis, allows developers to create custom speech vocabularies without Sensory assistance. These vocabularies are compatible with the Talk code in this release. A new section, Using the Quick Synthesis Tools, has been added to the Design Notes chapter of the manual.

Sentence-level Speech Synthesis

The code no longer incorporates silence measurements, simplifying application flow. The code is backward compatible with existing data files.

Record and Playback

This module incorporates new compression tables and filtering that improve the sound quality. Additional flexibility in thresholding is provided. Several bugs were fixed: under rare conditions, memory exhaustion was not detected; applications could not avoid linking CMP, even if not using compression. Post processing can now be called for any compression level, not just 4-bit.

Flash RAM Co-routines

Support for the Atmel-type 29x010-020 SPFlash (128-byte "sector program" or "page mode" devices) has been improved. The SD memory handler was re-written to be application-compatible with other devices. Samples that previously used SRAM now work with SPFlash. A bug in the 28SF040 RP routine that produced "clicks" for large-amplitude signals has been fixed.

Dual-Tone Multi-Frequency (DTMF) generation

The DTMF code now preserves the IMR register contents.

Music

A new sample program illustrates Sensory's 4-voice MIDI music synthesis capability.

Documentation

Portions of the manual have been re-written to provide better descriptions. A new section on Wordspot has been added to the Technology chapter. A new section, Using the Bootloader, has been added to the Software chapter. A new section, Using the Quick Synthesis Tools, has been added to the Design Notes section.

2. Getting Started

In order to start using the RSC-364 Development Kit, you need to accomplish the following "startup" tasks:

- Configure the Development Kit hardware for your needs.
- Set up the software development environment on an IBM-PC compatible computer.
- Become familiar with the application development cycle including the tools used to build an RSC-364 application using Sensory technology software.
- Become familiar with the capabilities of the various technologies offered for the RSC-364.

When these steps are done, you will be ready to begin developing your application. Sensory's goal is to make this process as clear and easy as possible. To achieve this goal, a set of sample tutorial programs is included, each of which emphasizes the use of one specific technology. We provide complete source code and tools to allow you to modify and re-build the samples. We encourage this hands-on exploration of these samples to gain experience and smooth the bumps in getting up the RSC-364 Development Kit learning curve. To further help with the four "startup" tasks:

- The Development Kit hardware is configured to allow most of the sample programs to run "out-of-the-box". Where necessary, we offer guidance in configuring the hardware for specific needs. The default configuration is compatible with the Demo Unit hardware.
- The section below, *Software Installation*, is a step-by-step guide to installing the software and setting up the development environment according to Sensory's recommended configuration. This also describes the sample programs available.
- The section below, *Assembling, Linking, and Loading Modules and Programs*, is a description of the tools and the development cycle.
- The sections below, Manual Contents, Development Kit Contents, and Diskette Contents, provide "laundry lists" for locating specific items in either the manual, the libraries, or else where in the Development Kit.
- The remaining sections in this chapter touch on a few important ideas that are discussed in detail in other chapters, and include some further suggestions and hints.

3. Software Installation

Software for the RSC-364 is organized into three categories:

- Source code, to define the application flow. This is usually in RSC-assembly language, designated by a file extension of ".A".
- Library code, to implement recognition and other Sensory proprietary technologies. This is usually in RSC relocatable object form, designated by a file extension of ".O".
- Tools code, to build applications. These are typically PC programs of extension ".EXE" or batch files of ".BAT" extension.

Copying Files

The software must be transferred to the IBM compatible PC, the development platform for the RSC-364. The software files supplied in the Development Kit are organized into various directories. Please maintain the organization of the software on your hard disk similar to the way it is organized on the Development Kit software CD. The recommended procedure is as follows:

1. Create a new directory, for example, SENSORY.
2. Change to this directory using the DOS CD command.
3. Use the DOS XCOPY command to copy the install program to this directory.
4. Execute the install program. This is a self-extracting archive containing all of the library files.
5. Install any additional software that may be provided as part of the development kit following the instructions provided.

Unless you follow this procedure, you may need to change some path names in the .BAT files and in source files that reference .INC files.

The directory structure should be similar to the following:

Directory of F:\Sensory

.	<DIR>	06-29-00	4:56p	.
..	<DIR>	06-29-00	4:56p	..
INSTALL	EXE 1,015,930	06-29-00	4:53p	Install.exe
BIN	<DIR>	06-29-00	4:57p	BIN
DATA	<DIR>	06-29-00	4:57p	DATA
SAMPLES	<DIR>	06-29-00	4:57p	SAMPLES
SAMPL364	<DIR>	06-29-00	4:57p	SAMPL364
SUPPLMNT	<DIR>	06-29-00	4:57p	SUPPLMNT
LIB	<DIR>	06-29-00	4:57p	LIB
LIB364	<DIR>	06-29-00	4:57p	LIB364
SA2	BAT 161	04-10-00	12:34p	SA2.BAT
SETLIB	BAT 159	11-16-98	6:57p	SETLIB.BAT
BOOTFILE	BAT 240	04-10-00	12:31p	BOOTFILE.BAT
SL	BAT 143	08-31-99	9:38a	SL.BAT
CONTENTS	TXT 7,248	06-29-00	4:03p	CONTENTS.TXT
	6 file(s)	1,023,881	bytes	
	9 dir(s)	7,508.68	MB free	

See the CONTENTS.TXT file for a complete listing of all files.

Setting up the Environment

In the listing of the directory structure, note a number of batch files at the root of the Sensory library. These files are used to invoke various tools such as the assembler and linker, and to set the library path environment variable, so they must be in your DOS path. This is most easily done by editing your AUTOEXEC.BAT file to include the Sensory directory in your "PATH=" statement. Note that this will not take effect until you re-boot, so you may also add to the path immediately by typing the following (substitute your path for "D:\SENSORY"):

```
D:\Sensory>SET PATH=%PATH%;D:\Sensory
```

If you do this, be sure to edit your AUTOEXEC.BAT, too.

As described below, Sensory recommends a development approach using "build files". This approach will add to your DOS environment variable strings, which may require allocating more memory for DOS environment variables. This can be done under Windows by the following:

1. Right-click on the DOS Icon and select "Properties".
2. Select the "Memory" tab
3. In the "Initial environment" window, select a number, at least 1024.
4. Click on "OK"

You may also edit the MSDOS.PIF file to change the environment size.

In DOS-only development environment, the environment size can be enlarged to 1024 by adding the following line to your CONFIG.SYS file:

```
shell=command.com /e:1024 /p
```

Note: This also works for windows 95/98

The final step in setting up the environment is to run the SETLIB.BAT file, which assigns new environment variables needed by the "build files". The syntax is SETLIB <sensory directory>.

```
D:\Sensory>setlib d:\sensory

D:\Sensory>rem Sets the path of the root of Sensory's Library

D:\Sensory>rem usage: SETLIB [path]

D:\Sensory>set LibPath=d:\sensory

D:\Sensory>set casmspec=-DLibPath:d:\sensory

D:\Sensory>
```


Configuring the Serial port of your PC

Many developers will use the PC serial port for downloading programs (using Sensory's bootloader software) and also for interacting with RSC applications (using a terminal emulator such as Hyperterminal). These two applications may compete for the PC serial port and can cause conflicts. If you use port2 for communication with the RSC, such conflicts can be minimized by inserting the following line in your \WINDOWS\SYSTEM.INI file under the [386Enh] section:

```
COM2AutoAssign=2
```

If you use port1, enter

```
COM1AutoAssign=2
```

For additional information about using the "COM<n>AutoAssign=" setting, please see the following article in the Microsoft Knowledge Base:

Q130402 Device Contention in Windows

Also, verify that the terminal emulator is using the correct IRQ and I/O address settings for the COM port it is using.

This completes the software installation and environment setup. Any additional software tools may be installed now.

Software Contents

The information below describes the general contents of the various subdirectories.

The \BIN directory contains the executable tools programs.

The \DATA directory contains directories with speech data files, Speaker Independent recognition weight files, and music data files. These files will be useful in prototyping.

The \LIB directory contains directories with the various object modules and includes files for Sensory technology code. Your programs will link or include these modules as required.

The \LIB364 directory contains directories with the various object modules and includes files for Sensory technology code that is specific to just the RSC-364. Your programs will link or include these modules as required.

The directory \SAMPLES contains directories with sources and executables for demo programs that illustrate the technologies supported in this release that are not specific to the RSC-364.

The directory \SAMPL364 contains directories with sources and executables for demo programs that illustrate the technologies supported in this release that are specific to the RSC-364. The sample programs are useful templates for developing applications that are more complex.

Each sample described below resides in its own subdirectory, which includes a "build" batch file. These batch files act as a "poor man's maker" to automate the assembly and linking operations and to facilitate using options such as different memories, options, or features. Each sample also includes a README file describing how to configure the hardware to run the demo. Eight of these demos are included in the ready-to-run Sample Flash, which is installed in location TSOP-B on the Memory Module board. When that Flash is installed in socket TSOP-B and shorting block JB4 (PC_DOWNLOAD) is installed, one of these programs can be selected using the rotary switch on the Memory Module. Other samples or developer programs may be loaded into the Sample Flash using the BOOTLOAD tool, or they may be run from an EPROM or ROM emulator in DIP socket U1. Following are brief descriptions of the sample programs.

Within the \SAMPLES directory are subdirectories with sample code:

The directory SIMPLE contains a simple test program, TIMING. The program is simple enough (it requires no technology code) that you should be able to re-create it at any time. It defines its own RAM memory structure. This small demo wiggles lots of pins, allowing you to use an oscilloscope to compare external read/write signals with the data sheet and to verify that things are working as they should. This program makes a good introduction to the simulator, too. To build this program simply type "buildtmg". It operates from an EPROM or ROM Emulator in DIP socket U1.

The TUTORIAL program in the TUTORIAL directory is intended for use with the RSC-364 simulator. This program is not typically executed on the RSC-364, but only on the simulator. See Chapter 5, Simulator, for more details.

Within the \SAMPL364 directory are subdirectories with more complex sample code:

The SI6 directory contains files for a sample of Speaker Independent recognition. One configuration of the SI6 sample is included in the Sample Flash as sample #0. See the README.TXT file for instructions to run this sample. This sample will let you experiment with Speaker Independent recognition. Study the calling examples. Also see Using Pattern Generation, Using (Speaker Independent) Speech Recognition and Using Speech Synthesis. Use SW-A or SW-B on the Motherboard for the switch required to run this sample. This sample is configured to demonstrate Simultaneous RecordRP and Patgen using the SPFlash memory.

The directory RS232 contains another small program, SERIAL, that illustrates use of the RS-232 routines for communicating with a terminal (commonly a PC terminal-emulator program). This program uses the standard RSC-364 RAM memory structure, and it is branched to by the standard startup memory module, modified for this sample. Since this program has minimal technology requirements but needs "hooks" to that code, this is a good example to use in learning about RAM allocation, IO configuration, and other startup issues. This sample operates from an EPROM or ROM Emulator in DIP socket U1.

The MUSIC folder contains a sample program that plays any of three tunes. This demonstrates the four-voice music synthesis capability of the RSC family of chips. Contact Sensory for information on developing custom tunes, ensembles, or musical instruments. This MUSIC program is included in the Sample Flash as Sample#7.

The directory SPKRDEP contains files for a demonstration of Speaker Dependent recognition. The

SPKRDEP sample is included in the Sample Flash as sample #1. See the README.TXT file for instructions to run this sample. See also Chapter 2, Using Speaker Dependent Speech Recognition. This sample uses SPFlash for read/write storage, but it can also be built with assembly-time options for storing templates in either the 24C65 Serial EEPROM or the SRAM. The version in the Sample Flash illustrates "name tagging", where recognition confirmation is provided by playing a stored voice recording made during training. Three different language versions may be linked: English, Japanese, or Korean.

The directory PASSWORD contains files for a demonstration of one-word Speaker Verification. The PASSWORD sample is included in the Sample Flash as sample #6. See the README.TXT file for instructions to run this sample. See also Chapter 2, Using Speaker Verification Technology. This sample also illustrates power down ("sleep mode") and storing words on-chip. No external read/write memory is required to run this sample. (Note: A typical Speaker Verification application should use at least two words -- preferably more -- in a sequence to obtain a reasonable security level. This one-word sample is designed for demonstration purposes rather than serious security applications).

The WORDSPOT directory contains files for a demonstration of one-word or sequential two-word Wordspot Recognition. The WORDSPOT sample is included in the Sample Flash as sample #4. See the README.TXT file for instructions to run this sample. See also Chapter 2, Using Wordspot SD Speech Recognition. This sample stores templates in on-chip memory, so no external read/write memory is required to run this sample.

The directory RECORDER contains files for a demonstration of a Voice Recorder. This will digitally record and playback audio sounds. The RECORDER sample is included in the Sample Flash as sample #5. See the README.TXT file for instructions to run this sample. See also Chapter 2, Using Voice Record and Playback. The sample stores the recorded voice in the SPFlash.

The directory CLSI contains files for a demonstration of Continuous Listening using Speaker Independent speech recognition. This sample operates from an EPROM or ROM Emulator in DIP socket U1. See also Chapter 2, Using Continuous Listening. Continuous Listening can also be used with SD or SV technology. The Speaker Verification sample can be configured for CL with SV.

The directory MATH contains another SI sample program. This sample gives additional working code samples of pattern generation, prompting, and SI recognition. It also illustrates usage of the recognition confidence and class to provide a range of appropriate speech feedback.

The directory SPKRVER contains a second Speaker Verification sample program (in addition to PASSWORD) that can be configured in a number of different ways using the builder batch file, BUILDSV.BAT. The SPKRVER sample is included in the Sample Flash as sample #2. In the default configuration it provides a 4-word sample security application with the words stored on-chip. Prompting language and debugging output may also be easily configured. The sample can be configured for Continuous Listening using SV words (see Using Continuous Listening).

The directory FASTDIG contains a sample program illustrating Dual Recognition Technology (DRT) and Pattern Queuing. This program prompts for a 7-digit string (e.g., a telephone number), then queues the sequence of seven patterns, allowing the digits to be spoken quickly. After all the digits have been spoken and queued, the program recognizes them with high accuracy using fast DRT. This sample is included in the Sample Flash as sample #3. See also Chapter 2, Using Dual Recognition Technology, and Chapter 3, "Using On-chip Pattern Storage".

The directory TTONES contains files for a demonstration of DTMF (touch-tone) synthesis. This sample operates from an EPROM or ROM Emulator in DIP socket U1. See also Chapter 2, Using Touch-tone Synthesis.

4. Assembling, Linking, and Loading Modules and Programs

Sensory uses the IBM-PC as the hardware platform for RSC software development. Source code is developed using standard PC editing tools under either DOS or Windows (saved as text files). Sensory does not provide an editing tool. Sensory's assembler tool, SASM2.EXE, and the linker tool, CLINK.EXE, run under DOS to create a binary file containing the executable RSC code. This file must then be transferred to a form that allows the RSC to execute it, typically either using an EPROM or FLASH programmer or a ROM emulator. The manual section "Assembler & Linker" contains reference information regarding detailed usage of these two tool programs. This section offers an overview of their use. (Note: The assembler and linker are actually cross-tools, since they execute on the PC, not the RSC.)

The assembler operates on one file at a time; it is usually invoked using the batch file, SA2.BAT, contained in the root of the Sensory directory. SA2 accepts the filename of an assembly source file and produces a relocatable object file (.O) and a listing file (.LST). The assembler must operate on each file that exists as source code. Most Sensory library modules are supplied in pre-assembled object format, not as source code. The standard file extension for assembly source files is ".A". The batch file, SA2, requires only the filename; it automatically appends the .A extension if none is given. An error during assembly will cause SA2 batch processing to pause for the user to enter a keystroke.

After the required source files have been assembled, the linker is invoked to link all the object modules needed for the program. Since linking normally involves many file names, the linker is usually run using the batch file, SL.BAT, also contained in the root of the Sensory directory. This command file requires no arguments – it reads a list of object file names from a command file, LINK.CMD, contained in the source directory. The linker combines all the object modules listed in LINK.CMD into a single binary executable and produces output files including a .MAP file and a .BIN file. The .BIN file is the binary RSC executable. Each application needs its own unique command file. The command file, typically named LINK.CMD, is usually generated automatically by the build batch file described below. See the sample programs for examples.

After linking, the .BIN file on the PC must be transferred to a memory device that the RSC can reference as Code Space. This could be an EPROM, a mask-programmed ROM, or -- in high-volume cases -- a custom RSC-364 ROM mask. During development it is convenient to use a ROM emulator for this purpose. A ROM emulator is supplied with software that allows downloading the .BIN file through a cable connected from the PC's parallel port to a circuit board containing a dual-port memory. A second cable from the ROM emulator connects to a socket on the developer's hardware. The RSC executes the code from the ROM emulator. Using such a tool, the developer can edit changes to the source, re-assemble, re-link, download, and begin execution in just a few seconds. Some developers may have difficulty finding ROM emulators. Sensory has had good success with ROM emulators from Tech Tools, PO Box 462101, Garland, Texas, 75046 (sales@tech-tools.com, or www.tech-tools.com). Many other companies make comparable products. Note that special emulators or adapters may be required to allow a ROM

emulator to operate with the RSC-364 Development Kit at 3V. The Tech Tools model FR2-xM-90 or Econo Rom ER2-xM-90 along with the ADP3V external power supply and low voltage adapter are recommended.

This release supports an alternative to ROM emulation for loading programs during development. This "bootload" approach transfers the binary file information from the PC to the Sample Flash on the DK Memory Module using the serial RS-232 port. Loading a program with this method is more cumbersome and slower (about 40 seconds) than using a ROM emulator (about two seconds), but it requires no additional hardware. See Using The Bootloader section in the Software Development chapter for details. See the Memory Module Configuration section below for information about configuring the development environment for either bootloading or ROM emulator development. Support for the bootload configuration allows the Demo Unit to be used as a primary development platform.

Sensory recommends using a batch builder methodology for building applications. The sample below illustrates this methodology for assembling and linking the sample program, TIMING.A. The BUILDTMG.BAT batch file assembles TIMING.A, creates a linker command file, LINK.CMD, and executes the link to create the binary, TIMING.BIN, and associated files. Portions in bold are typed by the developer. As shown, the linker command file, \SAMPLES\SIMPLE\LINK.CMD, includes just the source file. The other sample programs include library files, and the LINK.CMD for those samples is much larger.

A limitation of the linker command file is that it cannot have comments. Using the build file, BUILDTMG.BAT, allows adding comments and notes, and makes it easy and clear when trying different linking alternatives. See the sample CLSI for a more extensive illustration of using the build file and incorporating options. All of the sample programs include build files.

```
X:\samples\simple>buildtmg
RSC-264/364 ASSEMBLER V.2.01, release 14 Dec. 1998
copyright (c) 1995-1998 Sensory Inc.
** using MAXce.DEF Instruction Set, last rev:12-14-98 15:16:53
    No errors in assembly.
CLINK cross linker V2.0m
Copyright (C) 1994 by Sensory Circuits,ECO,&AWE
    No link errors.
X:\samples\simple>type link.cmd
timing

X:\samples\simple>
```

Only the TIMING.O file is linked in this sample. In any example using technology code the following files must always be linked (plus any others required by specific technologies):

```
;required link files with EVERY technology application
\lib\startup\startup.o      (or a local copy)
\lib\startup\config        (or a local copy)
\lib\util\delay14
\lib264\util\analog
\lib264\const364
```

5. Memory Module Configuration

The RSC-364 Development Kit consists of the three main modules.

- DK264T/364 Motherboard
- DK264T/364 Memory Module
- DK264T/364 I/O Module

A detailed description of the hardware configuration capabilities is contained in Chapter 4, *Hardware*.

The DK264T/364 Memory Module contains IC memory devices, sockets, and design features that should be understood by the programmer and hardware designer. The module, in conjunction with the motherboard, supports development of products in a variety of configurations:

- Compact products intended for production with all code executed from a single 64K ROM. The ROM could be external or could be the internal ROM in the RSC-364 ("Custom ROM" programs). SI6 might be an example of such a program.
- Products that may use up to 64K memory for executable code and fixed data such as synthesis words, but which require additional external read/write memory for storing templates, voice recordings, or other application-specific data. Recorder could be an example of such a product. A custom ROM would allow all code and static data to be on-chip, but a read/write memory would also be needed.
- Large scale RSC-364 products requiring more than 64K of code and fixed data, such as a product with a large synthesis vocabulary. These products can place recognition weights and synthesis data in Data Space, allowing access to more than 64K by using Banked Data Space. For a more complete discussion of this concept refer to Building Multi-Bank Systems in Chapter 3. Because an external ROM is needed to hold fixed data, this configuration would typically not use the RSC-364 internal ROM. Some applications may contain more than 64K of executable code, requiring Code Bank Switching. Code Bank Switching is not supported in this release.

Basic Configuration Jumpers

The four jumper blocks at JB2, JB3, JB4, and JB6 are used to configure one of four standard operating configurations. Normally no more than one of these jumpers should be installed at any one time (but see DIP socket U1, below). Non-standard configurations may require removal of all these jumpers. The PC-DOWNLOAD configuration executes special Sensory code from a device installed in socket PLCC-A that transfers RSC programs from the PC into the Sample Flash in TSOP-B for execution. The other three jumpers configure for running user code from one of three sockets, U1, PLCC-A, or PLCC-B, or for providing access to flash or SRAMRAM devices.

Refer also to the Hardware Chapter for more information on these jumpers.

Default Configuration

The RSC-364 Development Kit is shipped pre-configured for "bootload" development. In this configuration the hardware is compatible with Sensory's Demo Unit.

- Socket PLCC-A contains an OTP programmed with the RSC code to transfer data over the RS-232 serial connection. This program runs when the system is initialized with the BOOT switch depressed.
- Socket TSOP-B contains a pre-programmed 4 Mbit Code Space flash, "Sample Flash". This program runs when the system is initialized with the BOOT switch not depressed.
- Socket PLCC-B contains a 2-Mbit Data Space flash, "SPFlash" (sector programmed type 29xx020). This is the default read-write device.
- Jumper JB4, PC_DOWNLOAD, is installed and jumpers JB2, JB3, and JB6 are removed.

In this configuration the PROGRAM SELECT switch runs any of eight programs contained in the Sample Flash. Any of these selections can be re-programmed with developer's application code using the bootload procedure (See Using the Boot Loader in the Software Development chapter).

This configuration is suitable for developers who will not use a ROM emulator, and for whom the following limitations are acceptable:

1. The 29xx020 2Mbit flash is suitable as the only Data Space storage.
2. The 24C65 Serial EEPROM may or may not be used.
3. The Code Space requirements (program plus fixed data) are no more than 64K bytes.

If you need additional or different resources, or if you want to use a ROM emulator, you will need to use DIP Socket U1 for development.

DIP Socket U1

This socket accommodates an EPROM or ROM emulator of size up to 4 Mbits. To use DIP Socket U1 for development you must reconfigure the Memory Module board. Your code will execute from the device installed in the DIP Socket. First, realize that you always have access to serial EEPROM for storing data and up to 64 templates per device, since serial EEPROM is not in Data Space. Next, decide what Data Space resources your application will need:

1. Code Space only. (example: Program does not need flash for voice recordings, is small enough to fit with all SI weights and synthesis data into 64Kbytes)
2. Read-Only Data Space (example: Program using SI and a large amount of synthesis, but no voice recordings. SI weights and synthesis data can be placed in Data Space)
3. Read-Write Data Space (example: Program with SD and voice recording confirmation)
4. Mixed Read-only and Read-write Data Space (example: program+synthesis+weights is larger than 64K, and read-write memory is needed for voice recordings)

Based on the resource needs above, configure the board as described and install your EPROM/ROM Emulator in DIP Socket U1:

1. No Data Space. To select this operating mode install a shorting block at JB2: CODE_DIP and remove any shorting blocks in the adjacent three locations.
2. Read-Only Data Space: Most developers will use U1 for a combined Code+Data Space socket. In production, a single ROM will be used. DIP Socket U1 supports development of programs

with 512Kbits (64K bytes) of Code Space ROM and up to 3.5 Mbits (448 Kbytes) of Data Space ROM. (total Code+Data ROM =4Mbits). Extended address bits (A16 up to A21) are controlled by developer-specified IO pins (P0.2-P0.4 recommended). To use the Data Space ROM capability of U1, you must build the required decoding circuitry in the prototyping areas of the Memory Module and remove JB2 and the three adjacent jumpers. See a sample schematic in Building Multi-Bank Systems in the Software Development chapter. A simpler example can be found in the CLSI sample.

3. Read-Write Data Space. There are three cases:

- A) SRAM, up to 32K bytes. In this case, install jumpers JB6:DEMO-PLCC-B (enables SRAM as Data Space) and JB2:CODE-DIP. Remove the 29xx020 Flash from socket PLCC-B to prevent bus conflict with the SRAM.
- B) The 29xx020 SPFlash or another pin-compatible part is the (only) Data Space device. First install both jumpers JB4:PC_DOWNLOAD and JB2:CODE_DIP. JB4 enables PLCC-B as Data Space and JB2 enables U1 as Code Space. Next, remove the Sample Flash at socket TSOP-B to prevent bus conflict with the DIP socket. This is required because JB4 enables both the socket at PLCC-B (where the SPFlash is installed) and TSOP-B. NOTE: Take care in removing the Sample Flash. See Installing and Removing the Sample Flash in Chapter 4 for assistance.
- C) The Data Space device is not SRAM or pin-compatible with the 29xx020 SPFlash, or multiple Data Space devices are needed. You must use the prototyping section of the Memory Module to wire the appropriate circuitry. Any available sockets may be used.

4. Mixed Read-only and Read-write Data Space. You must use the prototyping section of the Memory Module to wire the appropriate circuitry. Any available sockets may be used.

PLCC-B

Socket PLCC-A provides up to 4 Mbit of OTP ROM for Code and/or Data. This socket normally contains a 2 Mbit 29xx020 SPFlash device accessed as Data Space, with a Code Space device in either U1 or TSOP-B as described above. The full 18-bit addressing capability of this part is available by installing JB4:PC-DOWNLOAD. Sensory library code supports this device for template and voice recording storage.

PLCC-B may also be used for Code Space in the development of applications using OneTimeProgrammable (OTP) ROMS up to 4 Mbit (Code and Data ROM). You must use the prototyping capability of the Memory Module to wire the appropriate circuitry.

Socket PLCC-A

Socket PLCC-A provides up to 4 Mbit of OTP ROM for Code and/or Data, similar to PLCC-B. Normally this socket contains a 2Mbit OTP with the "bootloader" RSC program for PC-DOWNLOAD operations. When the system is initialized with the BOOT switch depressed, the code in PLCC-A executes, transferring data from a program running on the PC to devices on the Memory Module. A separate document describes the bootloader (see the Design Notes Section).

This socket can also be used with a non-boot Code Space program by installing JB3:CODE-PLCC- A. When used with JB3 installed, certain restrictions apply:

- Only a 1Mbit or 2Mbit ROM can be installed, and only 512kbits may be addressed, controlled by address bits [A15:A0] of the RSC. Address bits A16 and A17 are grounded. A18 is ignored.

To employ this socket for general purpose Data Space read-only or read/write use, remove JB3 and connect the required decoding circuitry in the prototyping area. With JB3 removed, the socket can support up to 4 Mbits.

Serial EEPROM U12

Many Speaker Dependent applications need to store only templates, and a serial EEPROM is ideal for this. The 24C65 device at U12 can be configured to operate with any two IO signals on the same port (P0 or P1).

Flash TSOP-A

This IC site is reserved for user-specified memory parts. It is a memory footprint accommodating a 32-pin 8mm 20mm TSOP package.

Flash TSOP-B

This 4Mbit Flash device, M29W400T, is normally used for PC-DOWNLOAD operations, in which case it contains up to eight separate Code Space executable images. The rotary PROGRAM SELECT switch, S2, selects one of the eight programs. In its shipping configuration this Sample Flash contains the 364 sample programs. The bootloader may be used to replace these programs with developer programs. The Sample Flash can also be used for storing voice recordings, templates, and other data requiring non-volatile read/write memory. Connect the required decoding circuitry in the prototyping area. There is presently no software driver support for Data Space applications of this device.

Prototyping Area

The Memory Module is designed for flexibility. A minor amount of custom wiring can accommodate many different memory configurations. An area near the edge connector contains thru-holes for the control signals from the RSC-364 required for attaching to the RSC bus. Near each socket are thru-holes with the control signals for that socket. A variety of different memory devices may be installed in the sockets or are already assembled to the board. The RSC control signals should be connected to the appropriate memory device signal pins as required for a specific memory design. Three 14-pin dip sockets are provided to aid in construction of memory decoding circuitry. Two 40-pin DIP socket areas are provided to allow use of other memory packages (power and ground are not connected in this area). Additional Memory Modules may be ordered – please contact Sensory for ordering information.

6. RAM Allocation

Study the sample programs for definitions of RAM memory usage. Although the RSC-364 contains 448 bytes of Register Space RAM, almost all of this is needed for Sensory's technology code, despite heavy overlaying (e.g., speech synthesis and speech recognition uses the same RAM memory). The limited on-chip RAM requires careful product specification and design to assure memory availability and to avoid unintended interaction with Sensory's technology code.

The only RAM always available for applications code is a contiguous block of 32 bytes. This lower limit can be extended by virtually every application to at least 48 contiguous bytes (this is the default configu-

ration), and in most cases 58 contiguous bytes are readily obtainable. To assure no inadvertent overlaying of applications variables and Sensory variables, user variables should be allocated only from this user area. The file USERMEM.INC should be included to establish references for this memory. Also include MEMORY.INC to reference other common variables including some RAM locations that have public labels ("a", "dpl", "r0", etc.) because they are used to pass parameters to technology code modules, and may generally be used for temporary storage. Using the RSC-364 Memory Map in Chapter 3 describes the RAM in detail, and points out additional RAM available for applications under certain circumstances.

An additional on-chip memory area on the RSC-364 can be used for storing patterns or templates in certain applications. See the section Using On-Chip Template Storage in Chapter 3 for more information.

7. Allocating ROM for Speech and Weights

Building an application with either speech synthesis or Speaker Independent recognition requires linking your code with data files provided by Sensory. Speech data and weights are provided as object files that can be directly linked using the object linker (CLINK.EXE). Each separate synthesis vocabulary or SI recognition weight set has a name that is declared PUBLIC in the .O file. Normally the files containing this data would be included in the LINK.CMD linker command file as shown in the samples.

A common problem for applications with large amounts of speech is "ROM crunch" – insufficient space in the 64K Code ROM. When the speech has been compressed as much as possible and there is still not enough room, ROM allocation becomes a problem of managing memory larger than the 16-bit addressing capability of the RSC-364. For programs with moderate-sized executable code (64Kbytes or less) but large data requirements, placing the speech data in Data Space provides enough room.

Data Space Placement of Speech and Weights

Data Space placement of weights, speech, or other fixed application-specific data (e.g., tables) requires building two or more ROM images, one with the executable Code Space code, and the remainder with Data Space data. The linker can only deal with one 64Kbyte ROM bank, so manual assistance is required to resolve references between the two ROMS. This section describes that process for a 128Kbyte application using two extra files, IMPORT.A and LINKDATA.A. (The names are arbitrary.) Applications with Data Space larger than 128K follow the same general method, but need separate imports defined for each data bank.

Typically the Data ROM will contain one or more speech vocabularies and perhaps some SI weights sets or other data. Each object to be accessed from the Code ROM must have a PUBLIC label (speech, weights, and confidence levels for recognition sets have this automatically). The labels for any directly referenced application-specific data must also be PUBLIC. The Data ROM is typically built first, since the Code ROM references the Data ROM but not vice versa.

The steps to building the ROMs are:

1. Create a file, LINKDATA.A, containing any application-specific tables or information you want to access in Data Space. If you do not need any such data, the file should contain simply an END directive. Note this file is required.
2. Create a linker command file (similar to LINK.CMD) containing the names of all the Data

Space files. The first file in the list must be LINKDATA. Use the "-o<filename>" command-line option to give the .BIN file a different name.

3. Assemble LINKDATA, then link the Data Space ROM using the -m(map) option:
BIN\CLINK @<LINKFILENAME> -M
4. Use the map file from the Data Space ROM to determine the addresses of all public symbols
5. Edit these values into the IMPORT.A file as EQUates. All the symbols must also be declared PUBLIC in IMPORT.A.
6. Assemble IMPORT.A and link it into the Code Rom to resolve the Data Space addresses.

The Continuous Listening SI sample, CLSI, can be configured to use Data Space placement, and it has been set up to illustrate the process described above. See also Building Multi-Bank Systems and section 6, above, RAM Allocation.

When 64K Code Space is not enough

Some very large, complex applications may need more than 64K of Code Space ROM. This requires Code Bank Switching. The RSC-364 Development Kit with Sensory Speech 6 Technology does not support code bank switching.

Large Data Space Memory Model and Rapid Vocabulary Prototyping

Speech Synthesis data and SI Recognition weights may be located (independently) in either Code Space or Data Space. A single 64K binary can contain executable code, speech data, and weights all in Code Space. Alternatively, a larger 128K program can be built with 64K of executable code and 64K of speech data plus recognition weights (or 64K of code plus weights and 64K of speech data). A larger program of this type can fit in a single 1 Mbit ROM. Expansion to even larger Data Spaces is straightforward.

One benefit of placing speech data in Data Space is that a modest vocabulary can be processed to fit into 64K by Sensory's linguists very quickly and at a reduced cost compared to squeezing it to fit in, say, 32K. This can allow more rapid prototyping with correct target vocabulary. As a rule of thumb, rapid-turnaround compression is only possible with at least 1600 bytes/word of available memory for higher quality and at least 1100 bytes/word for lower quality. Thus, a 64K Data Space ROM could hold about 40 higher quality words, or about 60 lower quality words.

Vocabularies of this size can be processed by Sensory linguists and returned promptly.(NOTE: The developer must take care to assure proper device selection when using Data Space speech if read-write memory addressable in Data Space is also used for storing templates or voice recordings).

Data Space placement is also valuable for use with "Quick Synthesis", a tool that allows the developer to create speech synthesis vocabularies with no Sensory linguistic assistance. This provides quick development of vocabularies with good quality and compression rates of less than 2600 bytes/word.

8. RSC-264T/364 Simulator

The RSC-264T/364 simulator is a windows-based program called SIMULATE.EXE. This program simulates all processor operations and is very useful for working out timing and checking intermediate com-

putations. At the level of this simulation, the RSC-264T and RSC-364 are similar, so one simulator works for both chips. The simulator allows loading binary executables, and provides single stepping, run to breakpoints, and memory display. This can be of value in testing loop termination conditions, algorithms, and other code segments that intensively use the RSC-364 Register Space.

The simulator does not support the specialized on-chip hardware needed for speech synthesis or speech recognition, so you cannot use the simulator to check these parts of your code. Sensory is aware that this program does not comply with all Microsoft standards for Windows programs.

9. Notes and Suggestions

This section includes a compilation of helpful information gathered from Development Kit users. These notes should be briefly reviewed for familiarization before using the kit, but most will be more useful later as reference.

A First Exercise

The SI6 directory contains the file SI6.BIN. As an exercise, you may want to rename this file to ORIGINAL.BIN, rebuild this sample by re-assembling and re-linking (by running BUILD SI6.BAT), and confirm that the final SI6.BIN file matches this ORIGINAL.BIN file.

Debugging Help

The color LEDs and switches SW-A, SW-B, and SW-C are for general-purpose use in debugging and for operating the sample programs. The disk directory DEBUG_S contains object modules for a speech debugger that provides spoken data information about the contents of internal memory ("speech dump"). This can be a useful complement to the emulator. The speech debugger can also provide Sensory standard recognition feedback (class, level, silence, maxpower) during recognition. This capability incorporates a DebugEnable byte that can be set to different values to control the amount of feedback provided (the default is 0: all disabled). See Using Debug Speech in chapter 3, for information on standard feedback and on using speech output for debugging assistance. "Speech" debugging can alternatively be directed to the RS232 output by linking a different file.

Named Segments

The assembler supports the named SEGMENT directive, which is helpful in determining memory allocation from the .MAP file. Liberal use of this directive is encouraged. Small pieces of code in their own segments can often be tucked into the holes around speech or weights files by the linker, freeing up more ROM in tight applications.

Recognition Speed

Speech recognition is a two-step process of collecting data from the speech waveform in real-time while processing to generate a pattern, and then analyzing the pattern with a neural network. The neural net processing does not begin until the speech utterance is complete, so there may be a perceptible delay from completion of speech until the recognition is finished. This overall delay is the result of a variety of built-in delays (to improve accuracy), some of which can be reduced. The ultimate minimum delay is determined by the processing time for the neural net, which presently requires from 0.1 seconds (for Yes/No) to about 0.3 seconds (for 0-9) using Speaker Independent recognition. The time required for

Speaker Dependent recognition depends on the vocabulary size -- normally about five msec per vocabulary word for large (50-60 word) vocabularies. The maximum SD time per word -- about 30 milliseconds -- occurs with vocabularies in the range of 8-10 words. A Speaker Dependent parameter, SD_PERFORMANCE, may be adjusted to gain quicker recognition processing time at the cost of slightly reduced recognition accuracy. At the fastest setting, a 10-word SD vocabulary can be recognized in about 50 msec rather than the normal 200 msec. Speaker Verification always requires about 30 msec per word in the vocabulary.

Shoehorning

The limited on-chip resources of the RSC-364 create common problems for many applications developers. These problems include insufficient RAM, insufficient stack, and insufficient ROM.

The usual solution to RAM shortage is overlaying. Overlaying can resolve conflicts between temporary RAM needs, but it may require re-writing some modules to use different temporary variables. Sensory recommends a method of allocating temporary variables using the "set" directive that works somewhat like the assignment of automatic variables in the C language. This allows a local variable to have a meaningful name in one subroutine, and the same memory location to be used in a different subroutine by a different name. See the RECORDER sample program for an example. Overlaying within USER-RAM cannot solve the need for additional static (global lifetime) variables, however. This involves overlaying into the system RAM. Here are some suggestions for overlaying:

- The 5 bytes at 01Bh-01Fh are available unless Voice Record and Play is used.
- The 17 bytes at 010h-020h are available if neither SD Recognition, SV Recognition, Wordspot Recognition, Dual Recognition, nor Voice Record and Play is used.

10. Error Messages

The linker requires that the first file linked contain a path to the assembler. This is assured if the file is assembled on the user's computer, as is the case for typical applications. Attempting to link with the first file being a library module will cause an error message similar to the one shown below (but with a different path to the file, SASM2.EXE).

```
CLINK cross linker V2.0m
Copyright (C) 1994 by Sensory Circuits, ECO, &AWE
Can't open file 'F:\SENSORY\BIN\SASM2.EXE': No such file or directory
```

11. Manual Contents

The RSC-364 Development Kit Manual contains ten chapters designed to allow fast access to information. The manual consists of the following chapters:

1. **Overview.** Lists and briefly describes the hardware and software components of the Development Kit. This chapter also describes basic information needed for "getting started" with the Kit. IF YOU READ NOTHING ELSE, BE SURE TO READ "GETTING STARTED". There are several chapters describing memory module configuration, RAM allocation, ROM

allocation and a variety of notes and suggestions on using the Development Kit. This section also provides a complete directory listing of all software files provided with the Kit.

2. **Technology Library.** Describes how an application programmer can access the various speech and audio processing capabilities of the RSC-364. This chapter contains separate sections that describe the initialization, include files, linking, and macros required to access each technology. Separate sections describe Speech Recognition (Independent and Dependent), Pattern Generation, Speech Synthesis, Speaker Verification, Continuous Listening (two sections for the two different methods), Wordspot Recognition, Dual Recognition, DTMF, and Voice Recording and Playback.
3. **Software Development.** Contains reference documents used when programming the RSC-364. The first document describes issues concerning the insertion of silence durations and delays when working with speech synthesis. Another important document describes the User Memory Map of the Register Space (SRAM) available to programmers. Other documents describe use of the system configuration modules, Building Multi-Data-bank Systems, software routines for interfacing to a variety of external memories, use of Debug Speech, PC (RS232) Output, and the Floating Point Library.
4. **Hardware.** This chapter contains descriptions, schematics, and parts locator drawings for the Printed Circuit Assemblies provided in the Development Kit. This section also includes listings of pinouts, connectors, jumper configurations, LEDs, switches, and other hardware items.
5. **Simulator.** This chapter describes the Personal Computer (PC) software program, SIMULATE.EXE, a software simulation of the RSC-364. The chapter gives guidelines for use, including a tutorial.
6. **Assembler and Linker.** Describes usage of the Sensory cross assembler, SASM2, including command line options, source syntax, and directives. There is also a description of assembler extensions (Intel 8051-like instructions) for the RSC-364. This chapter also provides reference materials for the Sensory object linker, CLINK. The assembler, linker, and simulator all run on the IBM-PC platform.
7. **Upgrades.** This chapter provides a guide to experienced developers for "porting" RSC-364 applications to this library release 6. It also outlines how to incorporate new features into existing applications. If you are a new developer you may safely ignore this chapter.
8. **Data Book.** Contains the Data Book for the RSC-364. This provides a detailed description of the chip's architecture, features, memory and I/O interface, bonding pad descriptions, electrical characteristics, instruction set and other details of its use.
9. **Design Notes.** Contains detailed and comprehensive documents that will help a product designer define, specify, and develop successful products utilizing speech recognition and synthesis. These documents contain many suggestions for various stages of the product design cycle, that if closely followed will yield products with high recognition accuracy, quality, and reliability.
10. **Support.** Describes the Sensory support and warranty policies.

12. Development Kit Contents

The RSC Development Kit includes the hardware, software, programming tools, and documentation necessary to program the RSC-364. Following is the complete list of items included in the kit (improved designs with different part numbers may be substituted):

- DK264T/364 Motherboard - one of the two primary development hardware platforms (1)
- DK264T/364 Memory Module - the other primary development hardware platform (1)
- DK264T/364 I/O Module – platform enhancement with a prototyping area with a connection to the Sensory's 12-key keypad for developing applications. (1)
 - A 12-key keypad with a telephone-like key arrangement (1)
 - Microphone, modified by Sensory (1)
 - Speaker - 8 Ohm (1)
 - Spare RSC-364 chips (2)
 - Spare 3V 4MB Flash IC M29W400T (1)
 - Spare 5V 4MB Flash IC M29F400T (1)
 - Spare 3V 2MB Flash IC SST29LE020 (2)
 - Spare 5V 2MB Flash IC SST29EE020 (2)
 - Cable assembly for power connection (1)
 - Software Diskette (1)
 - Demo Unit 364 Software CD (1)
 - Jumper wires, installed (5)
 - Bootload OTP (1)
 - RSC-364 Development Kit Manual (1)

13. Software Development Library Contents

Refer to the library file \CONTENTS.TXT for a complete listing of the files in the Sensory Development Kit Software (compressed in INSTALL.EXE).

14. Sample Program User Instructions

This section describes the use of the Sample programs, the sources for which are included in the development kit. The goal of the Sample programs is to demonstrate Sensory technologies in a straightforward manner. Another group of programs, the Demo programs, is supplied with the Sensory RSC-364 Demo Unit. These Demo programs have advanced features that are useful for evaluating the technologies. Sources for these programs are not included in the Development Kit because their feature complexity obscures the tutorial goal of the Sample programs. A diskette with the additional Demo programs from the RSC-364 Demo Unit has also been included with the Development Kit. You may load either the Demo programs or the Samples programs using the bootloader. See the Design Notes section for instructions on operating the 364 Demo programs.

The sample programs demonstrate the following technologies:

Sample Name	Switch Number	Speaker Independent	Speaker Dependent	Wordspot	Speech Synthesis	Speaker Verification	Other Technologies
SI6	0	√			√		Patgen+Record
Spkrdep	1		√		√		
Spkrver	2				√	√	CL4 w/ SV
Fastdig	3	√	√		√		Dual Recognition
Wordspot	4			√	√		
Recorder	5	√			√		Record and Play
Password	6				√	√	Sleep, words-on-chip

Using the Samples

These samples provide simple illustrations of each technology available on RSC-364. An appropriate sample program is commonly used as the starting "template" for a custom application. Changes to the usage or application flow of any technology can be made for specific applications.

Please keep in mind that environmental noise and room conditions may affect speech recognition in these samples. Different types of noise affect the recognizer differently: directional noise is easier to screen out than ambient noise. In addition, a carpeted room absorbs noises while hard floors reflect them. Thus, if a radio is playing in the corner of a carpeted room, and the microphone is closer to the user than to the radio, the recognizer will still achieve a high rate of accuracy. Accuracy will suffer on a busy street corner or in a mall.

Running a Sample Program

The eight sample programs are pre-programmed in the 4Mbit Sample Flash installed at location TSOP-B. Software supplied with the Development Kit simplifies re-programming the Sample Flash with the samples at any time. Verify that the socket labeled PLCC-B contains the IC labeled "Sample Flash". If you are using your development kit for the first time, the Sample Flash chip should be pre-installed in the TSOP-B socket. If you need to re-install the Sample Flash chip, see Installing and Removing the Sample Flash in Chapter 4 for assistance.

Next, plug in the microphone and speaker, and then turn on the power. You are now ready to run the demo programs. Each program is selected with the "Program Select" rotary switch on the memory module. The demo programs are controlled via three buttons located on the motherboard: SW-A, SW-B, and SW-C. In general, SW-B is used for training (if relevant), SW-A is used for recognition, and SW-C is used to change configuration. All the demos beep at power up to signal that they are ready for use.

Troubleshooting Hints: If the demo does not work, check the following items:

- Make sure the microphone and speaker are plugged into the correct jacks.
- Make sure the power supply is at the appropriate level (3.0 or 5.0V).
- Check that the Sample Flash is installed with pin 1 correctly oriented in the socket, (text is upside down) and the lid is securely closed.

Sample Instructions

Speaker Independent

This sample highlights Sensory's speaker-independent recognition using a six-word set: {record, play, erase, call, modify, or skip}. In addition to recognizing elements of the recognition set, the demo unit calculates the probability that it has recognized the word correctly. The default configuration stores a voice recording of the word during pattern generation for recognition, and replays it after announcing the result.

To Start:

1. Set the Program Select switch to 0.
2. Turn power on.
3. A beep will sound to verify that the demo is ready for operation.
4. Press button SW-A. The unit gives a speech prompt listing the elements of the recognition set: "Please say 'record,' 'play,' 'erase,' 'call,' 'modify,' or 'skip.'"
5. Say a word from the recognition set, for example "modify."
6. The unit replies appropriately, based on the word it has recognized and the confidence level it has calculated. It also replays a recording of your voice made at step 5. When the unit is sure that it has not recognized the correct word, it beeps and asks, "What did you say?" If it is unsure of the result, the beep is omitted.
7. In addition to the recognition responses, the unit will provide feedback when recognition is encountering difficulty. For example, if the user is speaking at a low volume or if the microphone is too far away, the unit will request, "Please talk louder." If the unit repeatedly responds, "Please talk softer," one suggestion is to hold the microphone a little further away.
8. Press button SW-B to get a prompt that does not list all the recognition set elements: "Please say a word."

Fast Digits

The Fast Digits sample simulates a digit-dialing application using Dual Recognition Technology for recognition. This technology combines the strengths of Speaker Independent and Speaker Dependent recognition to achieve the necessary high accuracy for this application. Templates for the SD portion are stored in the Serial EEPROM. The patterns to be recognized are "queued" in on-chip memory.

To Start:

1. Set the Program Select switch to 3.
2. Turn power on.
3. A beep will sound to verify that the demo is ready for operation.
4. Press buttons SW-A and SW-B together. This erases any patterns stored in the memory and announces "Memory erased".

To Train the Digit Templates

1. Press button SW-A.
2. A speech prompt says, "Please say zero".
3. Say the word "zero" as you would when saying the digits of a phone number.
4. A speech prompt says, "Please repeat".

5. Say "zero" a second time.
6. The steps from 2-5 repeat for the digits 1-9.
7. A beep sounds when templates for all the digits have been trained.

To "Dial" a Number

1. Press SW-B.
2. A speech prompt says, "please say the number you want to call"
3. A "click" prompts you to say a digit.
4. Say a digit from 0-9 (be sure to say "zero", not "oh").
5. The steps 13-14 repeat until you have said seven digits. The clicks are paced by your speech. A pattern is generated for each word and saved on-chip.
6. After seven digits have been recorded, they are recognized.
7. Speech tells you the digit string that was recognized.

Record and Playback

This demo displays the RSC-364's ability to record and play back audio messages. It also uses speaker-independent speech recognition to choose between the record and playback functions. The length of message that can be stored depends on the memory available – the more memory, the longer the potential message. In this demonstration, about 16 seconds of audio input can be recorded in 512-kbit portion of the 2Mbit SPFlash used for the initial recording.

To Start:

1. Set the Program Select switch to 5.
2. Turn power on.

To record a message:

1. Press the SW-A button.
2. The unit prompts, "Record or Play?"
3. Respond with "Record."
4. The unit says, "Recording."
5. Begin recording your message. You can record for the full sixteen seconds or press the SW-A button to stop recording. If your message exceeds sixteen seconds, the demo will interrupt you with a double beep.

To play a message:

1. Press the SW-A button.
2. The unit prompts, "Record or Play?"
3. Respond with "Play."
4. The unit says, "Playback."
5. The unit then plays back your message.

To change the compression setting:

1. Press the SW-B button.
2. The unit notifies you of the next compression level, compresses the speech, and plays back the recording at that level. Note that compression takes roughly as long as the duration of the recording.

Three different compression levels are possible with this demo: 4-bit, 3-bit, and 2-bit. (For more information on compression, see below.)

3. After the three compressions have been made, subsequent presses of SW-B cycle through them, announcing the level and playing the recording compressed to that level.

The recording remains in flash when the power is turned off.

The record and playback demo shows the RSC-364's ability to perform on-chip compression. The compression can be set to different levels depending on the application. Higher bit rates have better sound quality, but require more memory. For example, with 512Kbit of memory, a 16 second message could be stored at 4 bits, 8kHz, or 32,000 bits per second (BPS). An 32 second message could be stored in the same Flash at 2 bits, 8kHz, or 16,000 BPS. With Sensory's silence thresholding technology, even longer recordings can be made. The level of compression used depends on the quality and quantity of playback desired: the more compressed a message, the lower the sound quality, but the lower the memory requirements.

Music

The music sample demonstrates the RSC-364 synthesizing high-quality four-voice music from MIDI data. "Four voice" refers to four different instruments or notes playing simultaneously.

To use this sample, rotate the Program Select knob to position 7. Turn the power on.

1. A beep will sound to verify that the demo is ready for operation.
2. Press the A button to play Für Elise.
3. Press the B button to play Row, Row, Row Your Boat.
4. Press the C button to play When the Saints Go Marching In.
5. Any tune can be interrupted by pressing one of buttons A, B, or C during play back.
6. When the program has finished playing the tune, or has been interrupted with a button press, it will respond with a double beep. Press button A, B, or C again to select a song.

Because the music is being synthesized in real-time, rather than played from a digital recording, this three-song demo is stored in about 4K bytes of ROM.

Voice Password

This sample highlights several technologies and features available with the RSC-364.

- The sample will verify a single password after it has been trained. If no word has been trained, it will return the error message, "Error. Memory empty." The password is stored inside the RSC-364 chip rather than in an external memory.
- The password is stored inside the RSC-364 chip rather than in an external memory.
- After training or recognizing the password, the RSC-364 goes "to sleep" (i.e., into a low-power state). It awakens on a button press (IO signal change).

To Start:

1. Set the Program Select switch to 6.
2. Turn power on.

Password Training

1. Press the SW-B button.
2. The program prompts, "Please say your password."
3. Say a password up to three seconds in length. Passwords with multiple syllables work the best.
4. The program prompts you to repeat your password, "Repeat."
5. Repeat your password.
6. If the responses do not match, the program says "Training error," and asks you to "repeat" again.

After training, the program says "Going to sleep. Press button A to wake up".

Password Listening

1. After setting your password, press button SW-A.
2. The program prompts, "Please say your password."
3. Say a word.
4. The program responds with either, "Password accepted" or "Password rejected." The program then goes back to sleep

Changing Security Levels

1. Press button SW-C after power-on.
2. The demo says the new security level (1-5).
3. Continue pressing button SW-C until the desired security level is reached.
4. You can then press button SW-B to prompt and listen for a spoken password.

Some things to try: Using different threshold levels, try varying your intonation or saying a different word when asked for your password. Ask someone to try to crack your password. Now tell that person what your password is and see if the device will accept his or her voice.

Voice Password demonstrates an extremely low cost method of providing biometric security in a variety of industries, including auto security, home security, appliances, and smart cards. A realistic application would use multiple passwords for improved security.

Speaker Dependent

Speaker Dependent demonstrates Sensory's speaker-dependent speech recognition technology. The user can train the system to identify from 1 to 32 words, with each word trained corresponding to a word number. Any words, from any language, can be used. During training, a voice recording of each word is made for replay during recognition.

To Start:

1. Set the Program Select switch to 1.
2. Turn power on.

To Train:

1. Press button SW-B.
2. The program prompts, "Please say word one."
3. Say a word.

4. The program then prompts, "Repeat."
5. The training continues as the unit asks you to say word two, and so on. A set smaller than 32 words can be created. There are three ways to end training:
When all 32 words are trained, the training cycle ends automatically.
When silence is recorded during training, the training cycle ends. A beep will signal that training has ended.
When the SW-A button is pressed while the unit is listening for a response.
6. If fewer than 32 words are stored, new words can be added at any time by pressing the SW-B button again. Training will automatically begin with the next available number.

To Recognize:

1. Press button SW-A.
2. The program prompts, "Say a word."
3. Say any word.
4. The program responds with the appropriate word number. For instance, if you say "Operator," and it is the seventh word in your set, the program responds, "You said word seven." It then replays the voice recording made when you trained that word.
5. If the unit cannot match what you have said to a word from the set, it responds, "Repeat" You can then repeat your word, or say a different one.

To Erase:

Press buttons SW-A and SW-B simultaneously to erase all trained words. You will hear a beep, confirming that the vocabulary has been erased. In the Flash version of this program, the trained words are stored in a non-volatile memory, so they remain trained when the power is turned off.

The Speaker Dependent demonstration can provide very high recognition accuracy, often above 99%. However, there are factors, such as word selection and the noise level, which can affect overall accuracy.

Speaker Verification

This sample program highlights the speaker verification technology available with the Interactive Speech™ chips, using the pattern queueing technique available with the RSC-364. The demo will verify passwords once they have been trained. Otherwise, it will give the error message, "Error." This demo will verify from one to four passwords.

To use this demo, rotate the Program Select knob to position 2. Turn the power on. A beep will sound to verify that the unit is ready for operation.

To Train your Password(s)

1. Press the B button.
2. The demo prompts, "Please say your (first) password".
3. Say your first password up to three seconds in length. Passwords with multiple syllables work the best.
4. The demo prompts you to repeat your password with "Repeat."
5. Repeat your first password.
6. If the two words do not match, the demo says "Training error," and restarts the training sequence from that password.

7. The program repeats steps 2-6 for the second, third, and fourth passwords.
8. At any point in the training sequence, if there is no response or if the A button is pressed, the unit responds with "Training complete," and exits the training mode. You may train up to four passwords.

To Listen to your Password(s)

1. After setting your password(s), press button A.
2. The demo prompts, "Please say your (first) password."
3. Say your first password.
4. The program repeats steps 2-3 for the second, third, and fourth passwords.
5. When the sequence is complete, the demo responds with either, "Password accepted" or "Password rejected."

To Change Security Levels

1. Press button C.
2. The demo says the new security level (high, medium, or low) and beeps.
3. Continue pressing buttons C until the desired security level is reached.

Wordspot

The Wordspot sample demonstrates Sensory's wordspot speech recognition technology. In this program, the user can train either one or two "trigger words" to be spotted. The program then listens for the first word, ignoring any other words. If a second word is trained, the program listens for it for 3 seconds after the first word is recognized (the two words are recognized in sequence, but other words may be spoken between them). Wordspot is based on Speaker Dependent recognition, so any words, from any language, can be used. The trigger words should be multi-syllable words or short phrases, such as "what time is it?"

To Start:

1. Set the Program Select switch to 4.
2. Turn power on.

To Erase any previous words and Train:

1. Press button SW-B.
2. The program prompts, "Say word one."
3. Say a word or short phrase, such "voice activation"
4. The program then prompts, "Repeat."
5. Say the word again.
6. The program prompts, "Say word two."
7. Press the A button or remain quiet if you want only one word. If you want two words, say a second word or phrase.
8. If you said a second word, the program repeats steps 4-5.
9. The program beeps to indicate end of training.

To Recognize:

6. Press button SW-A.

7. The program prompts, "Say a word." 8. Begin talking in a natural way. The program will listen for just your trigger word, ignoring other words or sounds. The program will continue to run until a button is pressed.
8. Begin talking in a natural way. The program will listen for just your trigger word, ignoring other words or sounds. The program will continue to run until a button is pressed.
9. If you trained a single word, the program will say "one" each time it recognizes that word. If you trained two words, it will say "two" each time it recognizes the first word followed by the second word within 3 seconds. Other words may be spoken between the two trigger words as long as the second word is spoken within 3 seconds of the first. After three seconds, the program returns to listening for the first word if the second has not been recognized.
10. The green LED indicates that the first word is being spotted; the yellow LED indicates the second word is being spotted.
11. If you train short, mono-syllabic words with little linguistic content the program will false-trigger occasionally or often. If you train two short phrases, false triggers will be rare.

To Change the Threshold:

Press button SW-C to cycle to the next threshold setting. The threshold determines how precisely you must say the trigger words. A higher threshold number requires more precision, but reduces false triggers.

With proper trigger words, the Wordspot demonstration can provide high recognition accuracy of words spoken in ordinary fluent speech. However, there are factors, such as word selection and the noise level, that can affect overall accuracy.

CHAPTER 2 - TECHNOLOGY LIBRARY

The creation of products that use Sensory's speech recognition, speech synthesis, or other speech processing technologies requires the use of the associated technology software modules. This chapter describes how to use and integrate the technology software modules within an application.

For each of the technologies being used, there is a subsection that describes the initialization procedure for the RSC-364 processor, the Include files that must be included with the application program, and a listing of the object modules that must be linked with the application module. The core technology software routines are typically invoked through macros. The goals of this approach are to simplify the details of invocation, and to improve portability between releases. The macros and any associated input and output parameters are described in each subsection. There is also a description of the basic Code Flow requirements and Sample Code listings that provide examples of how to implement each technology.

Technology code is generally provided in the form of relocatable object modules. Some sources are also supplied to allow customization.

Some of Sensory's technologies are compatible with the hardware resources of all of the RSC-series of integrated circuits. Library code for these technologies can be found in the \LIB directory and its subdirectories. These technologies include speech synthesis, record and play, and DTMF generation, for example. Other technologies are optimized to the hardware resources of specific chips. These include recognition technologies, for example. Library code for such IC-dependent technologies can be found in the \LIBXXX directories and subdirectories, where XXX is the chip identifier. Thus the library code specific to the RSC-364 resides in the \LIB364 directory. Similarly, sample code for the RSC-364 is now located in \SAMPL364 and its sub-directories.

Use this chapter to understand how to employ Sensory's technologies in an application. Refer to "Software Development" for general guidelines on building applications.

1. Important Notice

Recognition and Synthesis Vocabulary Usage

The RSC-364 Development Kit supplies developers with a variety of speech recognition sets and speech synthesis words/phrases. These recognition sets and synthesis words/phrases are provided for demonstration purposes only. The recognition sets and synthesis words/phrases are not intended for use in final products and such use is strictly prohibited.

You may use the Quick Synthesis tool in this Development Kit to create your own custom synthesis vocabularies. Please contact Sensory regarding any speech recognition or additional speech synthesis vocabulary development that may be needed for your final product. Sensory can provide information and assistance on selecting recognition sets and synthesis words/phrases.

2. Using Speech Synthesis

The speech synthesis software consists of a group of object files containing the linkable code. Using this code requires an application-dependent file (or files) containing the actual synthesis data. The software can drive either or both of the RSC-364's two speech outputs, the DAC (analog waveform) and the Pulse Width Modulator (PWM). The synthesis code may be invoked at a phrase level, or at an entire sentence level. This section describes how to use speech synthesis

Initialization

Initialization includes setting up the hardware and the interrupt vectors by linking in the module `\LIB\STARTUP\STARTUP`. This module sets the processor speed to oscillator1 1 with 1 wait state, and initializes interrupt vectors to provide for speech output. The default configuration for `STARTUP.A` enables speech synthesis.

In addition, certain I/O configuration must be defined. The module `\LIB\STARTUP\CONFIG` accomplishes this task for the standard Development Kit hardware, and can be modified for other designs. See *System Configuration and I/O Mapping* for more information.

STARTUP jumps to the label `INITIALIZE`, which should be placed at the start of the application program and must be `PUBLIC` for linker references.

Includes

The speech synthesis code should be called using an appropriate function macro. Applications should `LIBINCL` the macros defined in `\LIB\MACROS\TALK.INC`. The functions use memory locations that are defined in `\LIB\MEMORY\MEMORY.INC`, so this file must be `LIBINCL`'d in the application program. See *Using the RSC-364T Memory Map* for detailed information.

Linking

Link the application code with the `\LIB\TALK\TALK`, and `\LIB\TALK\TALKTBL`. Also link modules `\LIB\STARTUP\STARTUP` and `\LIB\TALK\TALK_H` or local, application-specific versions of these files. Applications using Sentence-level synthesis must also link in `\LIB\TALK\SENTALK`.

You must also insert the application-specific compressed speech data into the ROM image by linking in the appropriate object file(s). The speech data can be placed either in Code Space or Data Space.

Applications using Sentence-level synthesis must also link in the application-specific sentence table (see below). Placing speech data in Data Space requires a separate linking operation, since the Data Space portion of a final ROM image must be built separately. Retrieval of the speech data is accomplished by TALK_H, described below. See Allocating ROM for Speech and Weights in Overview Chapter.

You must also link in \LIB\STARTUP\CONFIG (or a local, application-specific version) to bring in the speech filter coefficient.

Filters

Some hardware configurations will require digital filtering of the speech signal before it is sent to the outputs. In general, if the hardware does not have filtering, the digital filter should be invoked. This is the case for typical applications that use the PWM. The digital filter is controlled by a coefficient parameter, which should normally be set to 2 when the filter is required, and set to 0 when the filter should be off.

Speech synthesis gets the filter coefficient from the module CONFIG.A. When the coefficient is non-zero, the code performs digital filtering on the PWM output only. The DAC output is never filtered in software. External filter components should be incorporated with the power amplifier required to use the DAC output.

See System Configuration and I/O Mapping under the Software Development tab for more information on filter coefficients.

Talk Handler

The speech synthesis sample data files in this Toolkit are compiled into linker-compatible object (".O") files. The functions in TALK.INC can access these speech data in either code or data space. This is accomplished via a memory handler located in TALK_H. The developer may edit this handler to read speech data from data space (MOVX), or from code space (MOVC), or from any other memory device. This feature is especially useful when prototyping, and in creating language-independent products. Sentence table data (see below) can be placed in the same or different memory space from the talk data.

The talk functions may be interrupted by arbitrary events as determined by the application. This is achieved by the technology code periodically executing a "callout" to the application code. The user must supply a handler for this callout. If there is no requirement to interrupting the speech, the handler can just return with the carry bit clear. If the handler wants to interrupt the speech, it sets the carry bit and returns. The external logic for deciding about interrupting is under the programmer's control, but it should execute as quickly as possible to avoid missing speech samples. The sample callout handler shown below is from TALK_H.A, which should be used as a template and linked with all programs that use the macros in TALK.INC. NOTE: the talk callout handler is actually a "jumpout" handler, since it returns with a JMP rather than a RET. Jumpouts are used in some technologies to conserve stack space. The TALK jumpout rate is roughly 40 times per second.

```
talk_handler:      segment "CODE"
;
; This routine allows the talk code to break on an external event,
; such as a button-press. Return with carry set to halt synthesis.
;
```

```

TalkHandler:

;<< insert the handler code here >>

;return with carry set on button A low
;    stc
;    tm BTN_A_PORT,#BTN_A_BIT
;    jz    th_exit
;    clc
th_exit:
    jmpTalkHandlerRet
;
;-----
;
; The talk functions access memory by this routine.
;
; Input:          Address of speech data in talk_mem_ptr
;
; Output:         Byte fetched in A
;
; Notes:         This routine may not use the hardware stack
;                or call anything else that does.
;                This routine may only alter registers A and Trash
;
; For addressing memory spaces larger than 64K, set up additional address
; bits before invoking Talk or SenTalk.
;
; SenTalkMemDriver and TalkMemDriver can access different spaces.
; If they both use the same space, one routine is sufficient
; By default, both use Code Space

TalkMemDriver:
    ;<< replace this with your driver code >>
    ; Get the talk data from data space, for example
    ;; movx    a,@talk_mem_ptr
    ;;ret

SenTalkMemDriver:
    ;<< replace this with your driver code >>
    ; Get the sentalk data from code space.
    movc    a,@talk_mem_ptr

    ret

END

```

Sentence-level Speech Synthesis

Using Sentence-level speech requires an additional module, SENTALK.O, and application-specific speech tables XXXXA.O. The tables contain information that can be used to link speech messages (from a standard .O speech file) into larger sentences which can include pauses and silence level measurements. This allows the programmer to implement an entire prompt phrase or other sentence with only one macro call. This technique is especially useful for developing products which use more than one language—simply replace the speech file (.O) and the sentence table (A.O), and the new language is in place. The sentence table takes care of the grammar and other structural issues.

Sentence tables will normally have the same name as the coordinating speech file, with an 'A' added at the end (i.e. SDDEMO.O; SDDEMOA.O). Speech and sentence files for the same project in different languages should have identical names, stored in different directories. This allows the project to be rebuilt for the new language by simply changing the appropriate file paths in the .BAT file and re-linking (no re-assembling necessary). See the speaker dependent sample code for an example of this technique.

The SENTALK macros call standard speech synthesis as a subroutine, so specifics about interrupts, etc. are the same.

Macros

In most applications only one speech synthesis macro is needed, but there are six versions to choose from. One set of macros invokes the older “Talk” routines; another set of macros invokes Sentence-level synthesis. Sentence-level synthesis simplifies pre-recognition prompting and makes conversion to different languages much easier. Alternate versions of the macros output to the DAC, to the PWM, or to both. The macros are contained in the file \LIB\MACROS\TALK.INC.

The Talk macros in this software release have fewer parameters than previous versions. Accordingly the “xxxQuickxxx” macros have been eliminated. The Talk Macro input parameters are defined as follows:

= 1-byte integer constant; *reg* = 1-byte register variable; *val* = either *#* or *reg*;
addr = 16-bit address constant; *page* = high-byte of address constant (page address).

For DAC (only) output:

CALLTALKDAC (*val message*, *page vocpage*, *val defaultrate*)
SENTALKDAC (*val sentencenum*, *addr sentencerom*, *page vocpage*, *val defaultrate*)

For PWM (only) output:

CALLTALKPWM (*val message*, *page vocpage*, *val defaultrate*)
SENTALKPWM (*val sentencenum*, *addr sentencerom*, *page vocpage*, *val defaultrate*)

For both DAC and PWM:

CALLTALKBOTH (*val message*, *page vocpage*, *val defaultrate*)
SENTALKBOTH (*val sentencenum*, *addr sentencerom*, *page vocpage*, *val defaultrate*)

Each of these macros synthesizes the specified message from compressed speech data, where the passed parameters are:

Input:	message	number of the message within the compressed speech data file.
	sentencenum	number of the sentence within the sentence table (.A).
	sentencerom	16-bit address where sentence table begins.
	vocpage	page address where speech data begins (a <i>page</i> is 256 bytes).
	defaulttrate	default sample rate override (normally 0 to use the rate imbedded in the speech data file).

Output: carry flag carry is set if an interrupt occurred.

Sample Code

The following shows two versions of a short program using speech synthesis. One version uses talk-level speech code; the other uses sentence-level speech code.

```
;This program is an example of using talk-level speech synthesis code
;Link with STARTUP, TALK, TALKTBL, TALK_H, and speech file VPHELLO.O
```

```
LIBINCL      "\lib\memory\memory.inc"          ; This is the memory map
LIBINCL      "\lib\macros\talk.inc"             ; This is the Talk macros

EXTERN       VPhello                          ; address of speech data

PUBLIC       Initialize

VR      equ   00h                            ; voice default sample rate

msg_hello   equ   0                          ; index of message "hello" in speech data area

ex:  segment "CODE"

Initialize:
CallTalkBoth #msg_hello,#high VPhello,#VR
jmp  $

END
```

```
;This program is an example of using Sentence-talk speech synthesis code
;Link with STARTUP, TALK, TALKTBL, TALK_H, and speech files VPHELLO.O and
VPHELLOA.O
```

```
LIBINCL      "\lib\memory\memory.inc"          ; This is the memory map
LIBINCL      "\lib\macros\talkint.inc"         ; This is the Talk macros

EXTERN       VPhello                          ; speech data address (page-aligned)
EXTERN       SNhello                          ; sentence table (arbitrary alignment)

PUBLIC       Initialize
```

```

VR      equ    9330/64          ; override the voice default sample rate

sn_hello    equ    1          ; Address of message "hello" in sentence table

ex:      segment "CODE"

Initialize:
SenTalkBoth #sn_hello, SNHello, #high VPhello, #VR
jmp      $

END

```

See any sample code in the directory \SAMPL364 for an illustration of how to implement sentence-level speech synthesis.

3. Using Pattern Generation

Sensory's speech recognition technologies consist of two main parts: pattern generation and recognition. The recognition portion of each technology is a specific algorithm for the particular recognition task (speaker independent, speaker dependent, dual recognition, wordspot, or speaker verification), but the pattern generation portion is similar for all three technologies. For speaker independent, pattern generation can make use of certain information contained in the weight set, allowing a slight increase in recognition performance. To take advantage of this feature, a special macro function, PATGENW is available. This function is identical to PATGEN, except that it includes a parameter that indicates the address of the weights table. A third macro invokes pattern generation for wordspot training. The pattern generation software is only available in object code format in this release. The pattern generation code in this Developer Kit is compatible ONLY with the RSC-364 and upward-compatible chips. It cannot run on the RSC-164 or RSC 264T.

A pattern is a compact representation of the acoustic information in a word. The RSC-364 generates patterns in real-time using software algorithms and specialized on-chip hardware. This chapter describes the important developer tasks required to generate a good pattern for subsequent analysis. Study this chapter in conjunction with the appropriate recognition technology chapter.

The RSC-364 and Version 6 Technology Code support two different pattern formats, small and large. Small templates (size 10) occupy less memory and recognize faster with a possible slight reduction in accuracy compared to large templates (size 16). Speaker Dependent, Speaker Verification, and Wordspot applications use large templates for improved accuracy. Speaker Independent weight sets will be selected by Sensory to be either small or large, as appropriate. The PatgenW macro will automatically create the correct pattern size for the specified weight set. The Patgen (no "W") and PatgenWS macros automatically uses larger templates (size 16). There is no need for the application explicitly to set the template size. NOTE: PatgenW must be used for SI recognition, to assure the correct size pattern is created. Also, PatgenWS must be used when training templates for Wordspot.

Templates for SD, SV, DRT, and Wordspot are stored in compressed format in Release 6. The storage requirements are 81 bytes for small templates, and 128 bytes for large templates. For simplicity, Sensory sample code uses 128 byte records for either size.

A feature of the RSC-364 is the capability of making a voice recording simultaneously with pattern gen-

eration. This feature is described in the section, Simultaneous Voice Recording, below. Another feature is the option to ignore certain Pattern Generation errors and continue listening. This "NO_ERRORS" option is described under the Patgen macro description and code flow section below.

Initialization

Initialization includes setting up the hardware and the interrupt vectors. This is easily implemented by linking-in the module \LIB\STARTUP\STARTUP. This module sets the processor speed to oscillator1 1 with 1 wait state, and initializes interrupt vectors. This clock state is required for pattern generation.

In addition, certain I/O configuration must be defined. The module \LIB\STARTUP\CONFIG accomplishes this task for the standard Development Kit hardware, and can be modified for other designs. See System Configuration and I/O Mapping for more information.

STARTUP jumps to the label INITIALIZE, which should be at the start of the application program and needs to be PUBLIC'd for linker references.

Includes

The pattern generation routines should be called using macros. The pattern generation macros are used by all recognition technologies. Applications using pattern generation code should LIBINCL the macros defined in \LIB364\MACROS\PATGEN.INC.

The pattern generation and recognition macros use a few memory locations that are defined in \LIB\MEMORY\MEMORY.INC, so this file must be LIBINCL'd in the application program. See Using the RSC-364 Memory Map for detailed information.

Linking

Link the application code with the initialization module \LIB\STARTUP\STARTUP, the configuration module \LIB\STARTUP\CONFIG, and the pattern generation modules BLOCK, NORMAL, NULOGFNS, PATGEN, DIGAGC, STREAM, and PATGEN_H (the default handler of the PATGEN interrupt function). This function, called every 12.5 msec, allows interrupting pattern generation. The pattern generation modules all reside in the \LIB364\PATGEN directory. Also link the appropriate recognition technology files, any required speech synthesis modules, and any required neural net weights (object files).

Macros

There are four macros in the pattern generation package, located in \LIB364\MACROS\PATGEN.INC. The macro definitions have been adjusted for normal demo performance, but they may be redefined to change their operating characteristics for specific applications (see *parameters* below).

There are three major steps involved in word recognition. They are:

silence-level determination

pattern generation

pattern recognition

In release 6.0, the first step is performed automatically as part of the second step. This simplifies the

application programming significantly. During wordspotting, all three steps run concurrently.

The Patgen macro input parameters are defined as follows:

= 1-byte integer constant; *reg* = 1-byte register variable; *val* = either # or *reg*.

PATGENRESULT ()

Returns the result of the previous pattern generation attempt.

Input: none

Output: a : 0 = ok; 1 = timeout (no data); 2 = too long (too much data);
3 = too noisy; 4 = too soft; 5 = too loud; 6 = spoke too soon;
7=too choppy (too many segments); 8=bad weights;
FF = interrupted

Notes: Use this macro to recover the return value from patgen if the A register has been destroyed. Normally this is invoked by technology code within other macros. This macro must be used after Simultaneous Patgen+RecordRP, because RecordRP destroys Patgen's value of the A register.

PATGEN (*val run_how*, *val errors*) [for speaker dependent (SD) and speaker verification (SV)]

PATGENW (*val run_how*, *val page*, *val errors*) [for speaker independent (SI)]

PatgenWS (*val run_how*, *val errors*) [for Wordspot (WS) training]

Generates a pattern for the spoken word.

Input: **run_how** controls how patgen runs (see run_how, below)

page page address where weights begin (a page is 256 bytes).

errors =0 : report all errors

<>0 : ignore certain errors and continue with pattern generation.

Output: a : 0 = ok; 1 = timeout (no data); 2 = too long (too much data);
3 = too noisy; 4 = too soft; 5 = too loud; 6 = spoke too soon;
7=too choppy (too many segments); 8=bad weights;
FF = interrupted

Notes: *Automatically* performs a silence measurement and may include up to 150 milliseconds of delay before pattern generation starts (to allow prompt echoes to decay).

Using PATGENW improves speaker independent recognition accuracy by providing extra information about the word set. PATGEN uses default values.

PatgenW produces a pattern of the correct size for the weight set. Patgen and PatgenWS produce a size16 pattern.

Errors due to "too soft" (4), "too loud" (5), or "too soon" (6) may be ignored if the NO_ERRORS option is selected (**errors** <> 0). In this case, patgen continues until the user says another word or until the (longer) timeout expires.

Parameters: (may be redefined)

MAX_WORDS – maximum number of words to process (default 2)

SEP_SIL – silence period that separates two words

PRE_SIL – 16-bit silence period to process before quitting (no sound). This parameter

specifies the timeout when NO_ERRORS is zero. It is automatically lengthened by 3 times when NO_ERRORS is not zero.

run_how

The **run_how** input parameter determines which of three modes of operation are used by Patgen:

#STANDARD – This is the normal mode of running Patgen as a stand-alone operation. The line of code following the Patgen macro will not execute until Patgen finishes.

#BACKGROUND – In this mode, Patgen operates as normal, but allows RecordRP to run in a multi-tasking fashion. When the macro is executed, Patgen begins operation but suspends itself and control returns to the application. Patgen execution resumes when the RecordRP macro is invoked. See

Simultaneous Pattern Generation and Voice Recording, in *Using Voice Record and Playback*, below.

#RP_THRESH – In this mode Patgen operates in a background mode, but does not produce a pattern. It simply provides threshold information for RecordRP. This mode would be used to record voice messages lasting longer than a few seconds, with silence thresholding

Parameters

The pattern generation macros use the following parameters to control operation characteristics. Default values are defined in \LIB364\MACROS\PATGEN.INC, but the user may redefine these values in an application program by using the SET directive. NOTE: The SET directive overrides any previous set values, so it can be applied locally before each instance of Patgen. This allows one instance of Patgen, for example, to listen for one word only, and another to listen for two.

Set MAX_WORDS to 1 if you want quick response to spoken words. This setting has the disadvantage that it may process and attempt to recognize a pre-vocalization sound such as a loud inspiration before speech actually begins. Setting it to 2 or more will process both the inspiration and the word and allow recognizing the word. The SEP_SIL parameter controls the maximum silence allowed after a sound before a succeeding sound is entirely ignored. Two sounds that are separated by silence less than SEP_SIL may be identified as separate words, in which case the louder of the two words will be used (unless MAX_WORDS is 1, in which case the first is used). For example, suppose MAX_WORDS=2 and your trigger word is "light switch" (a bad choice because it has a long interior silence). If you said "light switch" with no interior pause, this would be treated as a single word; if you said "light <pause> switch", PATGEN would prepare a pattern for the louder of light and switch. If you said, "light <pause> <pause> <pause> switch", PATGEN would prepare a pattern for light and would completely ignore "switch".

The PRE_SIL parameter determines how long PATGEN listens to silence before returning. If the NO_ERRORS mode is in effect, PRE_SIL is lengthened by 3X automatically.

The SEP_SIL parameter determines how long after the first word that PATGEN will listen for a second word (only used if MAX_WORDS is 2 or more).

The NO_ERRORS mode (**errors** parameter ≤ 0) allows Pattern Generation to ignore the "too soft", "too loud", and "too soon" error conditions and to continue operation. When errors is non-zero, the normal 3-second timeout is lengthened to around 10 seconds and the following errors are momentarily ignored:

- 1 - no data

- 4 - too soft
- 5 - too loud
- 6 - too soon

When one of these errors conditions is detected, control is not returned to the application. Instead, Patgen is re-started to listen for a new utterance. This allows the user time to realize the first utterance was not recognized and to repeat the word. If an acceptable utterance is not received within the 10 second window, then the error condition is returned to the application. Note that other error conditions such as "too long" or "bad weights" are returned immediately regardless of the state of the errors parameter.

Code Flow

Typically a synthesized speech prompt is used to elicit the vocal response that is to be recognized. The application should call Talk to produce the prompt, then call Patgen or PatgenW. The patgen function provides up to 100 milliseconds to allow echoes to die down before the processing begins. If an adequate silence background is not achieved within this time, the error code "spoke too soon" is returned, and processing is aborted. This assures that word beginnings are not clipped, thus improving recognition accuracy. This error and the "too loud" and "too soft" errors are ignored if NO_ERRORS is enabled. In this case Pattern Generation continues to listen for the user to repeat the word. Most users will naturally repeat the word if there is no response to the first utterance.

Silence Levels

The method of measuring silence levels required care in previous releases of the technology library. In release 6, silence measurement is automated within the patgen process. No special application steps are required.

Interrupts

The pattern generation software periodically executes a "jumpout" (poll) to allow the application to interrupt the pattern generation process. Using PATGEN_H.A as a template, you may make your own jumpout handler to use this feature.

Memory Handler

The file PATGEN_H.A also includes a memory handler to allow PATGENW to read information from a weights table in either code space or data space. The developer may edit this handler to work with the hardware configuration he has selected.

Results

The PATGEN routine returns (in the A register) a variety of error condition codes that may be used to produce appropriate feedback ("please talk louder", "it's too noisy", etc.). Recognition should not be attempted if any error is returned ($a > 0$). The error code returned by Patgen can also be recovered later by invoking the PATGENResult() macro.

Sample Code

The following sample program fragment uses pattern generation and SI recognition. Study these examples to understand the correct flow for macro invocation and handling the results of pattern generation. The complete source for this sample, in SI6.A, can be used as a SI recognition source-code template. (Some lines in the original source have been eliminated and the code has been simplified for clarity of

illustration).

```
; SI6.A -- patgen and SI example for RSC-364 using talk-level synthesis

LIBINCL "\lib\memory\memory.inc"
LIBINCL "\lib\memory\usermem.inc"

LIBINCL "\lib\macros\talk.inc"
LIBINCL "\lib\startup\io.inc"                ;use project-specific i/o

LIBINCL "\lib364\macros\patgen.inc"
LIBINCL "\lib364\macros\si.inc"
LIBINCL "\lib364\macros\debug_s.inc"

PUBLIC  Initialize

EXTERN  Wtsi6
EXTERN  CONFsi6

EXTERN  Vpsidemo3

UserRam                ;define RAM storage labels

best_class      fl      1
best_level      fl      1
say_num         fl      1
msg1            fl      1
msg2            fl      1
error_cnt       fl      1
weights         fl      1
.
.
.
ask:
    mov         msg2,#msg_short_qn
    mov         msg1,#msg_sil0
    mov         weights,#high Wtsi6
    call        ask_and_record
    cp          a,#1                ;if error, quit
    jz          main
    cp          a,#0FFh            ;if interrupted, quit
    jz          main

    Recog       weights            ;recognize using "weights"

    mov         best_class,a
    mov         best_level,b
    DbgRecogSI

results_nums:
    dec         r0
    cp          a,r0
    jnc         what_did_you_say1_nota ;if nota, say "what did you say?"
```

```

        cp            b,#CONFsi6                ;if confidence low, say "what did you
say?"
        jc            what_did_you_say1
you_said:
        mov          a,#msg_you_said            ; good recognition, acknowledge
.
.
.
;----- Here is the patgen portion (simplified) -----
ask_and_record:
repeat_question:
        CallTalkBoth  msg1,#high Vpsidemo3,#0
        CallTalkBoth  msg2,#high Vpsidemo3,#0

start_recording:
        PatGenW #FOREGROUND,weights, #0 ;generate pattern for spoken word

        DbgPatGen                                ;optional debugging output

        PatgenResult
        cp          a,#0
        jnz          error                        ;**error handling code omitted
rec_exit:
        ret

```

4. Using Speaker Independent Speech Recognition

The Speaker-Independent (SI) Speech recognition software consists of two main parts: pattern generation and neural net processing (recognition). This section describes the procedure used to access SI speech recognition.

Common Pattern Generation Technology

The pattern generation portion of speaker independent recognition is the same code used in speaker dependent and speaker verification. The details of intermixing speech prompts with silence checks and dealing with the results of pattern generation are described in Using Pattern Generation. That chapter should be studied carefully in conjunction with this chapter; that information is not repeated here. Note that the macro PATGENW should be used with SI recognition (instead of PATGEN).

Initialization

Initialization includes setting up the hardware and the interrupt vectors. This is normally implemented by linking-in the module \LIB\STARTUP\STARTUP. This module sets the processor speed to oscillator1 1 with 1 wait state, and initializes interrupt vectors.

In addition, certain I/O configuration must be defined. The module \LIB\STARTUP\CONFIG accomplishes this task for the standard Development Kit hardware, and can be modified for other designs. See System Configuration and I/O Mapping for more information.

STARTUP jumps to the label INITIALIZE, which should be at the start of the application program and

must be PUBLIC for linker references.

Includes

The SI speech recognition routines should be called using macros. Applications should LIBINCL the macros defined in \LIB364\MACROS\SI.INC.

The recognition macros use memory locations that are defined in LIB\MEMORY\MEMORY.INC, so this file must be LIBINCL'd in the application program. See Using the RSC-364 Memory Map for detailed information.

Linking

Link the application code with the initialization modules STARTUP and CONFIG in \LIB\STARTUP\, the modules ANALOG, FLOAT2 and INTMATH in \LIB364\UTIL\, and the SI Recognition module, \LIB364\SIRECOG, (and, optionally, PRIOR).

SI Recognition must also be linked with the module RECOG_H. This module contains two memory handlers that allow RECOG to access weights tables from either code space or data space. The developer should edit both handlers to reflect the selected hardware configuration. This feature is especially useful in designing language-independent products.

Recognition uses neural net weights data contained in object (.O) format, which must also be linked. If the weights data are to be located in data space, they must be separately linked.

Also link the pattern generation modules BLOCK, NORMAL, NULOGFNS, PATGEN, DIGAGC, STREAM, and PATGEN_H, all in \LIB364\PATGEN\.

SI Memory Handler

The macros in SI.INC read data from code space or data space, depending on the memory handlers in RECOG_H.A (see above). Two different handlers are required. A sample of such handlers appears below.

```
; RECOG_H.A: callout routines for recognition functions.

        PUBLIC      RecognMemDriver1
        PUBLIC      RecognMemDriver2

recog_h:      segment "CODE"

;
; The recognition driver accesses weights by this routine.
; It fetches one byte from an address in the memory space
;   specified by this routine.
;
; Input:      Address of weight data in r6/r7 register pair
;
; Output:     Byte fetched in A register
;             Value in r6/r7 preserved
;
```

```

; Notes:      This routine may not make a hardware call
;             or call anything else that does.
;             This routine may only alter registers A, B, and Trash
;             This routine uses a hardware 'ret' to return.
;
RecogMemDriver1:

    ;<< replace this with your driver code >>

    ; Get the memory from code space.
    movc  a, @r6

    ret

; The recognition driver accesses weights by this routine.
; It fetches two bytes from an address in the memory space
;   specified by this routine.
;
; Input:      Address of weight data in r6/r7 register pair
;
; Output:     Word fetched in r4/r5 register pair
;             Value in r6/r7 preserved
;
; Notes:      This routine may not make a hardware call
;             or call anything else that does.
;             This routine may only alter registers r4, r5, A, B, and Trash
;             R6 MUST be an even number.
;             This routine uses a hardware 'ret' to return.
;
RecogMemDriver2:

    ;<< replace this with your driver code >>

    ; Get the memory from code space.
    movc  r4, @r6
    inc   r6
    movc  r5, @r6
    dec   r6

    ret

;-----
END

```

Macros

The file \LIB364\MACROS\SI.INC defines two macros used for SI speech recognition. The input parameters for the SI speech recognition macros are defined as follows:

= 1-byte integer constant; *reg* = 1-byte register variable; *val* = either # or *reg*.

RECOG (*val* **page**)

Determines which word was spoken.

Input:	page	page address where weights begin (a <i>page</i> is 256 bytes).
Output:	a	best match (index, 0 to number of words in set - 1).
	b	confidence level (0 to 100).
	r0	number of words in set including REJECT category (for use with
PRIOR).		
	r2	second best match (invalid if confidence = 0).
	r3	confidence level of second best match.
	r4	third best match (invalid if confidence = 0).
	r5	confidence level of third best match.

Notes: Words in a set are normally arranged in alphabetic order (exception: number sets are arranged numerically). The “Yes/No” set then has the following output values:
a=0 : No a=1: Yes a=2: REJECT r0=3 (2 words + REJECT)

See *Results* section below for explanation of REJECT category.

PRIOR (*val weight, val action*)

Adjusts the prior probability of one answer relative to other possibilities (optional).

Input:	weight	answer to adjust (index, 0 to number of words in set).
	action	<>0 : emphasize the answer by doubling its probability =0 : set the probability for the answer to zero.
Output:	a	best match (index, 0 to number of words in set).
	b	confidence level (0 to 100).
	r0	number of words in set including REJECT category.
	r2	second best match (invalid if confidence = 0).
	r3	confidence level of second best match.
	r4	third best match (invalid if confidence = 0).
	r5	confidence level of third best match.

Notes: Use PRIOR when there is a single correct or preferred answer, or a single answer that should be removed from consideration. Can be called repeatedly for multiple answers. The “zero” feature should only be applied to low-probability answers. Attempting to zero an answer with a high probability will cause PRIOR to return a=REJECT, b=0, and meaningless results in the other registers.

Code Flow

After the pattern has been successfully generated (as indicated by no error returned by PATGENW), the pattern is analyzed by a neural network to recognize the word. Speech or RS232 debugging may be used during development to determine the class and confidence level. The application should use the confidence level results in an appropriate way as shown in the sample below. Note that the pass/fail confidence level may be different for each weight set. Weight sets include a public symbol, CONFxxx (where xxx is the name of the weights table). This value is the pass/fail confidence level for the weight set, which varies from set to set. The "confidence" value returned by the RECOG routine should be compared to CONFxxx to decide whether to accept or reject the recognition result. The preferred value is easily accessed by EXTERN'ing in the application the symbol CONFWEIGHTS, which is defined in

each WEIGHTS.O file.

Results

There are two types of recognition questions. Questions with a single correct answer ("what is two + three?"), and questions with multiple correct answers ("what is your favorite color?"). Recognition levels can be improved for single-correct-answer questions by calling the PRIOR macro. This routine adjusts the recognition results slightly to emphasize the correct answer.

RECOG and PRIOR return the best match (what the recognizer believes was said) and the confidence level (how sure the recognizer is about the best match). These values can be tested to dispatch the code to various responses. For instance, if the confidence level is above the pass/fail value and the correct answer is returned, the program could say "great!" If the confidence is above the pass/fail value, but the wrong answer is returned, the program could say "try again," and re-ask the question. If the confidence is below the pass/fail value, the program could say "did you say [answer]?" and recognize a yes/no response.

RECOG and PRIOR also return the second-best and even the third-best answers. These can be used when the best answer is incorrect. In the example above, the user could answer "did you say [answer]?" with a response of "no." The program could then ask, "did you say [second-best]?"

These macros may also return a value in the "a" register that corresponds to the REJECT category. This indicates that the spoken word does not match any of the words in the recognition set, even with a very low confidence level, and thus must be placed in this unrecognized word category. This category is indicated if the best match value returned in the "a" register is greater than the highest return value for a valid word. For example, in the word set NOYES, there are three possible index values that can be returned in the "a" register that correspond to words "No", "Yes", and the REJECT category. The word "No" is assigned the index value of 0, The word "Yes" is assigned the value of 1, and the REJECT category is assigned the value of 2. The index value assigned to the REJECT category is equal to the number of words in the recognition set. If the REJECT category is returned as the best match, then the application program should ask what was said and repeat the recognition sequence (or some similar action).

Sample Code

The following code fragment from SAMPL364\MATH illustrates use of all of the return values from SI speech recognition. Study this example (and the pattern generation example in Using Pattern Generation) to understand the correct flow for macro invocation and handling the results of recognition in the general case. Most applications require simpler tests. See the SI6 sample program for such an example.

```
.                                ;prompting, pattern generation, & error checking
EXTERN      CONFdigits
.
.
Recog weights                        ;compare against weight set for nums
Prior answer, #1                    ;emphasize correct answer

dec     r0                          ;set r0 to NOTA
mov     nota,r0                     ;save NOTA
```



```

        mov     best_class+0,a           ;save best result
        mov     best_class+1,r2        ;save 2nd best result
        mov     best_class+2,r4        ;save 3rd best result

        mov     best_level+0,b         ;save best level
        mov     best_level+1,r3        ;save 2nd best level
        mov     best_level+2,r5        ;save 3rd best level

        DbgRecogSI

        cp      best_class,nota        ;if NOTA
        jnc     what_did_you_say      ; "what did you say?"
        cp      best_level,#CONFdigits ;elseif good recognition
        jnc     high_conf              ; "great!" or "try again"
check_valid:
        cp      best_level,#0          ;elseif valid level
        jnz     did_you_say           ; "did you say _?"
what_did_you_say:
        mov     msg1,#msg_sil20        ;else
        mov     msg2,#msg_sil20        ; "what did you say?"
        mov     msg3,#msg_sil20        ;endif
        mov     msg4,#msg_what
        jmp     ask1
high_conf:
        cp      best_class,answer      ;if correct answer
        jnz     try_again              ; "great!"
        CallTalkBoth #msg_great,#high VPsample,#VR
        jmp     main
try_again:
        ;else
        mov     msg1,#msg_try_again    ; "try again"
        mov     msg2,operand1          ;endif
        add     msg2,#msg_numbers
        mov     msg3,operator
        add     msg3,#msg_plus
        mov     msg4,operand2
        add     msg4,#msg_numbers
        jmp     ask1
did_you_say:
        mov     msg1,#msg_sil20        ;"did you say _?"
        mov     msg2,#msg_sil20
        mov     msg3,#msg_did_you_say
        mov     msg4,best_class
        add     msg4,#msg_numbers
        mov     weights,#high WTnoyes
        call    ask_and_patgen
        cp      a,#1                   ;if error, quit
        jz      main
        Recog weights                   ;compare against weight set for yes/no
        DbgRecogSI
        cp      b,#CONFnoyes           ;if poor recognition
        jc      did_you_say            ; "did you say _?"
        cp      a,#ANSWER_YES          ;elseif answer YES
        jz      high_conf              ; "great!" or "try again"

```

```

        cp      a,#ANSWER_NO          ;elseif answer NO
        jz      get_next              ; get next best
        jmp     did_you_say           ;else (NOTA)
                                       ; "did you say _?"
get_next:
        mov     best_class+0,best_class+1
        mov     best_class+1,best_class+2
        mov     best_class+2,#0
        mov     best_level+0,best_level+1
        mov     best_level+1,best_level+2
        mov     best_level+2,#0
        cp      best_class,nota       ;if NOTA
        jnc     what_did_you_say      ; "what did you say?"
        jmp     check_valid

```

5. Using Speaker Dependent Speech Recognition

A speaker dependent (SD) vocabulary is a collection of words that are to be discriminated from one another using SD recognition. Sensory's technology supports vocabularies potentially as large as 255 words, although 64 words is a practical maximum size. An application can easily switch between different vocabularies as needed. The number of vocabularies in an application is limited only by the external memory available, so the RSC-364 can provide recognition of a very large number of words if they are partitioned properly into vocabularies and if external memory is available. Even with no external memory, the RSC-364 can perform SD recognition for a small vocabulary (up to 6 words) by storing the words on-chip.

An application using SD recognition must provide a way for the user to train the vocabulary words. Typically this training would be done one-time-only as part of an initial product setup procedure. The training procedure creates and saves a user-specific "speaker dependent" template for each word of the application. (Training is also called enrollment). The software in this release uses templates made by averaging two patterns of each word to increase accuracy.

The SD speech recognition software consists of two main parts: pattern generation and recognition. The necessary software libraries are available as object code for linking into a ROM image. This chapter describes the use of SD speech recognition. Speaker Verification (SV) technology is quite similar to SD and is described in a following section.

Common Pattern Generation Technology

The pattern generation portion of speaker dependent recognition is the same code used for speaker independent and speaker verification. The details are described in Using Pattern Generation. It will be useful to study the section, Using Pattern Generation in conjunction with this section. The information in that section is not repeated here. Speaker Dependent recognition automatically uses the larger "Size16" templates. No application calls are necessary to specify this. A special macro, PatgenWS, must be used when training Wordspot templates.

Initialization

Initialization includes setting up the hardware and the interrupt vectors. This is easily implemented by

linking-in the module \LIB\STARTUP\STARTUP. This module sets the processor speed to oscillator1 1 with 1 wait state, and initializes interrupt vectors. This clock state is required for pattern generation. In addition, certain I/O configuration must be defined. The module \LIB\STARTUP\CONFIG accomplishes this task for the standard Development Kit hardware, and can be modified for other designs. See System Configuration and I/O Mapping under the Software Development tab for more information.

STARTUP jumps to the label INITIALIZE, which should be at the start of the application program and needs to be PUBLIC'd for linker references.

Includes

The SD speech recognition routines should be called using macros. LIBINCL the macros defined in \LIB364\MACROS\PATGEN.INC (for pattern generation) and in \LIB364\MACROS\SDSV.INC.

The recognition macros use memory locations that are defined in \LIB\MEMORY\MEMORY.INC, so you must also LIBINCL this file in the application program. See Using the RSC-364 Memory Map for detailed information.

Except when using the ON_CHIP option (see the PASSWORD and SPKRVER sample programs), speaker dependent recognition requires external read/write memory for storing templates. (A template is made by averaging two or more patterns to improve recognition.) The file "\LIB364\SD_SV\XRAM_SDV.INC" must always be included to define the memory co-routine interface (see Using Co-Routines for External Memory Access).

Linking

Link the application code with the initialization module \LIB\STARTUP\STARTUP, the math module \LIB364\UTIL\INTMATH, the configuration module \LIB\STARTUP\CONFIG, and the pattern generation modules BLOCK, NORMAL, NULOGFNS, PATGEN, DIGAGC, STREAM, and PATGEN_H in \LIB364\PATGEN. Next, link in the SD recognition modules, SD.O, SDWEIGHTS.O, and TEMPLATE.O from \LIB364\SD_SV\ . In special cases a different version of SDWEIGHTS.O may be substituted for specific applications such as recognition over telephone lines.

Finally link in one of the external memory co-routine modules such as SRAM_SDV.O (or your replacement) and any other memory access, technology, or application modules you use. To use the WORDS ON CHIP option, link in \LIB364\SD_SV\ONCHIP.O as the on-chip memory handler, and also link \LIB364\SD_SV\AUXTMPLT.O, the auxiliary template handler.

Pattern Memory Usage

Patterns and templates in the RSC-364 may be stored in external memory, or may temporarily reside in one or more of four areas within the 364 chip. The technology software identifies these four areas as:

- #TMPLT_UNKNOWN - This is the area where Patgen places the new pattern.
- #TMPLT_KNOWN - Typically used for retrieving a template from external memory.
- #TMPLT_MODIFIED - May be used to hold a second pattern or average of two patterns.
- #TMPLT_WS - This area is used by Wordspot to hold the template being spotted.

Except for Patgen and Wordspot, usage of these areas is determined by the application program through use of the macros described below. An additional template storage area is provided for use by the ON_CHIP feature. This feature allows storing up to four WS templates on chip when no external tem-

plate storage is used. See Using On-Chip Template Storage in Chapter 3, for details.

NOTE: in this release, TMPLT_MODIFIED and TMPLT_WS are the same physical area. Application use of this area limits on-chip template storage to 4 templates.

Except for Patgen and Wordspot, usage of these areas is determined by the application program through use of the macros described below. An additional template storage area is provided for use by the ON_CHIP feature. This feature allows storing up to six SD templates on chip when no external template storage is used. See Using On-Chip Template Storage in Chapter 3, for details.

To conserve storage space, the template management routine compresses the template data before storing it, and decompresses it when retrieving. The template size information (10 or 16) is also stored with the template data.

Macros

The Speaker Dependent/Speaker Verification recognition module contains six macros: two for template transfer, and one each for training and recognition for SD and SV. The SV macros are described in the Speaker Verification section. Of the four macros used by SD, three are for training and one is for recognition.

The speaker dependent input parameters are defined as follows:

= 1-byte integer constant; reg = 1-byte register variable; val = either # or reg.

PUTTMPLT (val **tmpltX**) [TRAINING]

Saves a template (pattern) from a specific area in technology RAM to external memory.

Input: **tmpltX** Identifier of one of three areas where the template can be located in the RSC-264T memory.

Output: None

Notes: Requires external memory handler at location put_byte_sdv.

Memory handler must set up memory initialization before invoking

PUTTMPLT. The parameter tmpltX can have the following system-defined values:

#TMPLT_UNKNOWN - usually the most recent patgen pattern

#TMPLT_KNOWN - typically an older pattern from external memory

#TMPLT_MODIFIED - typically an averaged pattern

#TMPLT_WS - This area is used by Wordspot

GETTMPLT (val **tmpltX**) [TRAINING]

Retrieves a template (pattern) from external memory into a specific area in technology RAM.

Input: **tmpltX** Identifier of one of three areas where the template can be located in the RSC-264T memory.

Output: None

Notes: Requires external memory handler at location get_byte_sdv.

Memory handler must set up memory initialization before invoking GETTMPLT. The parameter tmpltX can have the following values:

#TMPLT_UNKNOWN - usually the most recent patgen pattern

#TMPLT_KNOWN - typically an older pattern from external memory
 #TMPLT_MODIFIED - typically an averaged pattern
 #TMPLT_WS - This area is used by Wordspot

TRAINS (val **tmplA**, val **tmplB**, val **tmplR**, val **sd_performance**) [TRAINING]

Compares two patterns and conditionally averages them to make a template.

Input: **tmplA** template area identifier for one pattern
tmplB template area identifier for a second pattern
tmplR template area identifier for the average of the two patterns
sd_performance range 1-5: controls speed vs. accuracy tradeoff (see below)

Output: a 0=good match, 1= match failed
 b score: range is 0 (perfect) to 255 (poor) match.

Notes: If the template comparison matched, the user should call PUTTMPLT to write the averaged template to external memory. The memory handler must set up external memory initialization before invoking PUTTMPLT.

In the case of a match failure, the application program must decide whether to re-train both templates or to continue by trying to match one of the two existing templates. Attempting to re-match the oldest utterance could fail repeatedly if the template is bad (for example, because a background sound was picked up during the first patgen). For this reason, usually the averaged template is re-written to the same template area as the oldest pattern -- keeping the newest pattern intact -- but this is the responsibility of the user. The choices and consequences are:

tmplR value	good match	fail match
tmplA	tmplA has average	tmplB only is valid
tmplB	tmplB has average	tmplA only is valid
tmplR <> A or B	tmplR has average	tmplA,B both valid but non-matching

The parameters **tmplA**, **tmplB**, **tmplR** can have the following values:

#TMPLT_UNKNOWN - usually the most recent patgen pattern
 #TMPLT_KNOWN - typically an older pattern from external memory
 #TMPLT_MODIFIED - typically an averaged pattern

NOTE: in this release, application use of the TMPLT_MODIFIED area limits on-chip template storage to 4 templates. To store up to six templates on-chip, set **tmplR** to TMPLT_KNOWN or TMPLT_UNKNOWN. See the samples.

RECOGS (val **patnumber**, val **sd_performance**) [OPERATION]

Input: **patnumber** number of templates to be compared (i.e., size of recognition set)
sd_performance range 1-5: controls speed vs. accuracy tradeoff (see below)

Output: a best match (index, 0 to number of words in set - 1).
 b score: range is 0 (perfect) to 255 (worst) match.
 r0 number of words in set.

- r1 result: 0=pass; 1=fail (based on best score), 2=uncertain
- r2 2nd best match (index as for "a")
- r3 2nd-best score (range as for "b")
- r4 (r2-b) = difference between two best scores.

Notes: Requires external memory handler at location *get_byte_sdv*.

Requires external template validation handler at *validate_template_sdv*.

No match is indicated if a=r0.

Physical limit of **patnumber** is 255, but less than 64 gives better recognition.

r1=2 (uncertain) return indicates that more than one template produces a good match.

Score may be of help in deciding if a new vocabulary word is too close to an existing word (see below).

Additional Parameters

The RecogSD macro uses the following parameters to control operation characteristics. Default values are defined in \LIB364\MACROS\SD_SV.INC, but the user may redefine these values in an application program by using the SET directive.

The parameter SD_PERFORMANCE controls convenience/speed/accuracy tradeoffs in SD recognition. Set it to 1 if you want the fastest and easiest possible response; set it to 5 for the highest accuracy with strict pronunciation constraints. Although the recognition time for SD in Version 6 is somewhat faster than in previous releases, the value of SD_PERFORMANCE still has a strong influence over the total recognition time. This value may be set from 1 to 5 (inclusive), with the default value of 3 giving the recommended compromise between speed and accuracy. For SD sets larger than 5 words, the execution time in milliseconds is approximately given by

$$T_e = 2 + 3 \cdot n + 30 \cdot \text{SDP}$$

where n = number of templates in the set and

SDP = 1,3,3,5,5 for SD_PERFORMANCE = 1,2,3,4,5.

This formula shows that a small, 5-entry set can be recognized in less than 50 msec with SD_performance set to 1. Conversely, recognition from a large 64-word set with SD_performance equal to 5 requires about one-third second. Very little degradation in accuracy occurs in reducing SDP from 5 to 2, but accuracy does decrease somewhat for SDP = 1. Note: recognition times above are based on flash storage of templates. Slower memories such as Serial EEPROM will increase the recognition time.

The parameter SD_WEIGHTS is normally taken to be the library default, suitable for generic electret microphones. Special SD weight sets may be needed for particular environments such as over-the-telephone.

Training Code Flow

As explained above, the application should be divided into a training phase and a normal operation phase. Each time a pattern is produced, the appropriate pattern generation macros must be called as described in Using Pattern Generation. After the first pattern has been generated during training, it is stored to external memory using PUTTMPLT. This step is necessary because there is normally insufficient space in the RSC-364 RAM to store the original pattern while generating the comparison pattern. The technology code manages an area of RSC-364 RAM for temporary storage of a second pattern dur-

ing comparisons, but this area is used by the PATGEN macro. When the WORDS ON CHIP option is used, a special version of PUTTMPLT saves the first pattern internally in a temporary location.

A second pattern is generated and the two patterns are compared. If they match well enough, the patterns are averaged and the resulting template is written to external memory. If the patterns do not match well enough, the application should require another patgen. See the discussion at the description of TRAINSD above for considerations about training failures.

The next important step is to check that the new template is sufficiently different from the other templates in the vocabulary. Similar words should be avoided in order to keep the recognition rate high. After the new template has been trained, but before including it in the vocabulary, recognize the averaged pattern against the existing vocabulary and reject the word if it matches too closely to an existing word. This topic is discussed in more detail below.

This process is repeated for each word in each vocabulary. When all the words have been trained, the vocabulary can be used for recognition.

Recognition Code Flow and Results

To recognize a word the pattern is generated as described elsewhere. If the pattern generation software returns no error, the RECOGSD macro is then called. During execution of the RECOGSD macro every template of the vocabulary is read at least once-some templates may be read multiple times. Templates in the RSC-364 are stored externally in compressed form occupying up to 128 bytes (the actual number of bytes depends on whether the template size is set to 10 or 16). The RECOGSD macro calls *get_byte_sdv* as many as 128 times for each of the patnumber templates, and there may be additional system reads as well, so many external memory accesses are required for each recognition using a large vocabulary. The memory co-routine is responsible for determining the proper memory address of each byte, and for switching to a new pattern when necessary as illustrated in the code samples. See also Using Co-Routines for External Memory Access.

A match criterion is calculated for each of the patnumber templates. The RECOGSD macro returns an index in the "a" register that indicates the closest match and an index in the "r2" register that indicates the second-closest match. If there is no match, "a" will be returned with the same value as "r0" and the "r1" register will be non-zero. A RecogSD match can fail if no template gives a good match (r1=1), or if multiple templates give good matches (r1=2).

The "b" register contains the score, a distance measure of the fit of the unknown pattern to the closest matching template. The lower the score, the better the match. The score for the second-closest match is returned in the "r3" register. These scores can be used to assure that new vocabulary words are not too similar to existing words. For example, the words "Composition" and "Competition" sound enough alike that it may not be wise to use both words in a single SD recognition vocabulary. To detect the similarity between a newly trained word and an existing word, call RECOGSD immediately after calling the TRAINSD macro. (No calls to synthesis or other technologies may intervene or the pattern may be disturbed.) Each macro will return a score in "b". The score from TRAINSD indicates how closely the two utterances (presumably of the new word) matched, while the score returned by RECOGSD indicates how well the averaged template matches the closest other vocabulary word. If these two scores are close, the speaker should be instructed to choose a different word. As a rule of thumb, the RECOGSD score should be at least 12 more than the score from TRAINSD. See the SPKRDEP sample for an exam-

ple of usage.

The memory co-routine for SD has no real-time constraints but must be fast enough to avoid adding noticeable delays to the recognition time. For large vocabularies recognition time is roughly 6 msec per word in the vocabulary plus any extra time for external memory access. With some memory technologies this extra time may become significant when reading a large number of templates. Using the serial EEPROM code supplied in this toolkit configured for sequential reading in FAST mode, recognizing a word from a vocabulary of 64 words requires approximately 550 milliseconds. The Serial EEPROM memory handler consumes approximately 40% of this time. (See the comments at "additional parameters" above regarding speed/accuracy tradeoffs in Speaker Dependent Recognition.) NOTE: All bytes (up to 128) of an individual template will always be read sequentially, but the templates may be read in any order.

Accuracy decreases and recognition time increases as the size of the recognition vocabulary increases. In a practical sense this limits the maximum recognition set size, but the actual limit is set at 255 by the 8-bit **patnumber** value.

A function called `VALIDATE_TEMPLATE_SDV` is included in the memory handler module. This function may be called by the application to assist in setting up memory pointers. It is also called by the technology code before reading and recognizing each template, and it may be used by the application to indicate that the template should not be analyzed. This allows templates to be non-contiguous in memory, greatly easing memory management. A stub routine can be found in any of the memory handler suites. See also *Using Co-Routines for External Memory Access*.

Sample Code

The short program fragments below illustrate the use of the SD recognition macros. Refer to the example in *Using Pattern Generation* for sample code illustrating prompting and pattern generation. These examples, taken from the sample program `\SAMPL364\SPKRDEP`, show only certain speaker dependent portions.

```
SetPwd:                                ;TRAINING - GENERATE, SAVE, AND AVERAGE A
TEMPLATE
    cp      ctr,#MAX_CLASSES    ;max number of words in recognition set
    jnc     main

    mov     msg1,#sen_say_word_x
    add     msg1,ctr

spla:
    call    ask_and_record

; . . . omitted code to deal with errors. Fall-thru if no error

    mov     dpl,#LOW data_offset    ;set start of first pattern
    mov     dph,#HIGH data_offset
    mov     r7, #Template_Bank
    call    init_xwrite_sdv         ;init co-routines
    mov     a,ctr
    call    validate_template_sdv
    PutTmpl #TMPLT_UNKNOWN         ;save template
```



```

        call    exit_xwrite_sdv            ;finish up co-routines
; fall into confirm
SetPwdCnfrm:
    mov        msg1,#sen_repeat
    call       ask_and_record
    cp         a,#0
    jnz        SetPwdError                ;error

    mov        dpl,#LOW data_offset        ;set start of first pattern
    mov        dph,#HIGH data_offset
    mov        r7, #Template_Bank

    call       init_xread_sdv              ;init co-routines
    mov        a,ctr
    call       validate_template_sdv
    GetTmpl    #TMPLT_KNOWN                ;get first template
    call       exit_xread_sdv              ;finish up co-routines

; Note: We use TMPLT_UNKNOWN as a trainSD target (rather than TMPLT_MODIFIED)
;       because this is convenient for the 'too similar' test below.

    TrainSD    #TMPLT_UNKNOWN,#TMPLT_KNOWN,#TMPLT_UNKNOWN ;average tem-
plates

    cp         a,#0
    jz         TrainPass
    jmp        SetPwdFail

SetPwdError:
    ;code to deal with error

TrainPass:

;OMITTED CODE TO CHECK FOR "TOO SIMILAR" BEFORE INCLUDING IN VOCABULARY****
see sample program

    mov        dpl,#LOW data_offset        ;set start of first pattern
    mov        dph,#HIGH data_offset
    mov        r7, #Template_Bank

    call       init_xwrite_sdv            ;init co-routines
    mov        a,ctr
    call       validate_template_sdv
    PutTmpl    #TMPLT_unknown              ;save template
    call       exit_xwrite_sdv            ;finish up co-routines

;-----

AskPwd:                                     ;recognition fragment
    mov        msg1,#sen_say_a_word
    call       ask_and_record
    cp         a,#0
    jnz        AskPwdError                ;patgen error

```

```

        mov     dpl,#LOW data_offset      ;set start of first pattern
        mov     dph,#HIGH data_offset
        mov     r7, #Template_Bank

        call    init_xread_sdv            ;init co-routines
        RecogSD ctr, #SD_PERFORMANCE      ;recognize against template list
        call    exit_xread_sdv            ;finish up co-routines

        DbgRecogSD                        ; optional debugging output (speech
or RS232)

        cp      r1,#1
        jc      AskPwdPass
        jz      AskPwdFail
        jmp     AskPwdUncertain
    .
    .

```

6. Using Speaker Verification

Sensory's Speaker Verification (SV) technology provides text-dependent, multiple-word, voice security screening. The technology uses the combination of a password (a secret word or brief phrase known only to the speaker) and a template (a characterization of the specific way in which the speaker says the word or phrase) to provide two barriers to entry for an impostor:

- The impostor must first learn the true speaker's secret password, and
- The impostor must then also imitate the natural voice of the true speaker.

Use of multiple passwords further limits access by impostors.

An application using SV must provide a way for the user to train the password(s). Typically this training would be done one-time-only as part of an initial product setup procedure. In most applications execution of the training code should be under supervision to assure security. (Impostors should not be able to train passwords.) The training procedure creates and saves a user-specific template for each password. The software in this release uses templates made by averaging two patterns of each password to increase accuracy.

The speaker verification software consists of two main parts: pattern generation and verification. The necessary software libraries are available as object code for linking into a ROM image. This chapter describes the use of SV technology.

Common Pattern Generation Technology

The pattern generation portion of speaker verification is the same code used for speaker independent and speaker dependent recognition. The details of pattern generation are described in Using Pattern Generation. It will be useful to study the section, Using Pattern Generation in conjunction with this section. The information in that section is not repeated here. Speaker Verification recognition automatically uses the larger "Size16" templates. No application calls are necessary to specify this.

Comparison with Speaker Dependent Speech Recognition

Speaker verification is a special variation of speaker dependent recognition. The main conceptual difference is that SD recognition attempts to ignore differences in the way a given word is spoken by different speakers, while the SV recognition attempts to detect such individual differences. Another difference is the typical vocabulary size: SD may often be 30, 40, or more words, but a multiple-word password has a maximum length of four words. Speaker verification applications typically may screen many users, but an SD application is usually trained for a single user. Apart from these differences, the two technologies share the need for external non-volatile read-write memory, memory co-routine handlers, and separation of the application into training and operational phases. It will be useful to study Using Speaker Dependent Speech Recognition in conjunction with this section. The option of ON_CHIP allows the RSC-364 to store a single password sequence without needing any external memory.

Note that Sensory's SV technology is Speaker Verification, not Speaker Identification. In verification, the identity of the user is already known. A limited form of Speaker Identification is available, however. Due to the memory requirements of the algorithms, a maximum of five different speakers can be tested in a single SV recognition attempt. Further, because of memory limitations, the identification must be made based only on the first password and not the entire sequence. An application could use SV technology alone to identify and verify a user based only on the password sequence if the total number of users was five or fewer. However, an application that needs to voice-identify one user from a group larger than five users would need to use SD recognition (on the first utterance) to identify the best sub-group of five, then SV to verify from that sub-group. Alternatively users could be placed into sub-groups of five during training, then during operation the utterance would be recognized against each of the sub-groups with SV. Both approaches require careful use of template memory management to assure that the correct next template is provided to the recognizer.

Note also that the meaning of "next template" might vary within the application. For example, assume four users have each trained 3-word passwords, and the first user's first password has just been provided to the recognition routine. Then the "next template" would be the first user's second password when doing Speaker Verification, but would be the first template of the second user when doing Speaker Identification (testing each user's first template for the best match to the unknown pattern). A local variable may be used by the application to signal which of these two memory arrangements is to be used by the memory co-routine, `validate_template_sdv`. The Speaker Verification code can accommodate a maximum of five users in this way for Speaker Identification.

The speaker verification macros parallel those of SD, but they are given different names because they invoke different underlying technology code and may return different values. The speaker dependent macros are named XXXXSD, and the corresponding speaker verification macros are named XXXXSV. The template manipulation macros are identical for both technologies.

Initialization

Initialization is identical to speaker dependent. The `SD_PERFORMANCE` parameter does not effect speaker verification.

Includes

The SV routines should be called using macros. LIBINCL the macros defined in `\LIB364\MACROS\PATGEN.INC` (for pattern generation) and in `\LIB364\MACROS\SDSV.INC`.

The recognition macros use memory locations that are defined in \LIB\MEMORY\MEMORY.INC, so you must also LIBINCL this file in the application program. See Using the RSC-364 Memory Map for detailed information.

Except when using the ON_CHIP option (see the SPKRVER sample program), speaker verification requires external read/write memory for storing templates. (A template is made by averaging two or more patterns to improve recognition.) The file "\LIB364\SD_SV\XRAM_SDV.INC" must be included to define the external memory co-routine interface (see Using Co-Routines for External Memory Access).

Linking

Link the application code with the initialization module \LIB\STARTUP\STARTUP, the math module \LIB364\UTIL\INTMATH, the configuration module \LIB\STARTUP\CONFIG, and the pattern generation modules BLOCK, NORMAL, NULOGFNS, STREAM, PATGEN, DIGAGC, and PATGEN_H in \LIB364\PATGEN. Next, link in the SV recognition modules, SV.O, SVWEIGHT.O, and TEMPLATE.O from \LIB364\SD_SV\ . Speaker Verification uses some SD code, so you must also link \LIB364\SD_SV\SD.O. Finally link in one of the external memory co-routine modules such as SRAM_SDV.O (or your replacement) and any other memory access, technology, or application modules you use. To use the WORDS_ON_CHIP option, link in \LIB364\SD_SV\ONCHIP.O as the on-chip memory handler, and also link \LIB364\SD_SV\AUXTMPLT.O, the auxiliary template handler.

Pattern Memory Usage

Patterns and templates in the RSC-364 may be stored in external memory, or may temporarily reside in one or more of four areas within the 364 chip. These four areas are identified by the technology software as:

- #TMPLT_UNKNOWN - This is the area where Patgen places the new pattern.
- #TMPLT_KNOWN - Typically used for retrieving a template from external memory.
- #TMPLT_MODIFIED - May be used to hold a second pattern or average of two patterns.
- #TMPLT_WS - This area is used by Wordspot to hold the template being spotted.

Except for Patgen and Wordspot, usage of these areas is determined by the application program through use of the macros described below. An additional template storage area is provided for use by the ON_CHIP feature. This feature allows storing one 4-word SV sequence of templates on chip when no external template storage is used. See Using On-Chip Template Storage in Chapter 3, for details.

NOTE: in this release, TMPLT_MODIFIED and TMPLT_WS are the same physical area. Application use of this area limits on-chip template storage to 4 templates.

To conserve storage space, the template management routine compresses the template data before storing it, and decompresses it when retrieving. The template size information (10 or 16) is also stored with the template data.

Macros

Speaker Dependent/Speaker Verification recognition module contains 6 macros, two for template trans-

fer, and one each for training and recognition for SD and SV. The SD macros are described in the Speaker Dependent section. Of the 4 macros used by SV, three are for training and one is for operation.

The speaker verification input parameters are defined as follows:

= 1-byte integer constant; *reg* = 1-byte register variable; *val* = either # or *reg*.

PUTTMPLT (*val* **tmpltX**) [TRAINING]
 Saves a template (pattern) from a specific area in technology RAM to external memory.
 Input: **tmpltX** Identifier of one of three areas where the template can be located in the RSC-364 memory.
 Output: None
 Notes: Requires external memory handler at location *put_byte_sdv*.
 Memory handler must set up memory initialization before invoking **PUTTMPLT**. The parameter **tmpltX** can have the following system-defined values:
 #TmPLT_UNKNOWN - usually the most recent patgen pattern
 #TMPLT_KNOWN - typically an older pattern from external memory
 #TMPLT_MODIFIED - typically an averaged pattern

GETTMPLT (*val* **tmpltX**) [TRAINING]
 Retrieves a template (pattern) from external memory into a specific area in technology RAM.
 Input: **tmpltX** Identifier of one of three areas where the template can be located in the RSC-364 memory.
 Output: None
 Notes: Requires external memory handler at location *get_byte_sdv*.
 Memory handler must set up memory initialization before invoking **GETTMPLT**. The parameter **tmpltX** can have the following values:
 #TMPLT_UNKNOWN - usually the most recent patgen pattern
 #TMPLT_KNOWN - typically an older pattern from external memory
 #TMPLT_MODIFIED - typically an averaged pattern

TRAINSV (*val* **tmpltA**, *val* **tmpltB**, *val* **tmpltR**, **sd_performance**) [TRAINING]
 Compares two patterns and conditionally averages them to make a template.
 Input: **tmpltA** template area identifier for one pattern
 tmpltB template area identifier for a second pattern
 tmpltR template area identifier for the average of the two patterns
 sd_performance range 1-5: controls speed vs. accuracy tradeoff (see above)
 Output: a 0=good match, 1= match failed
 b score: range is 0 (perfect) to 255 (poor) match.
 Notes: If the template comparison matched, the user should call **PUTTMPLT** to write the averaged template to external memory. The memory handler must set up external memory initialization before invoking **PUTTMPLT**.
 In the case of a match failure, the application program must decide whether to re-train both tem-

plates or to continue by trying to match one of the two existing templates. Usually the averaged template is re-written to the same template area as the oldest template, but this is the responsibility of the user. See the discussion at TrainSD above.

The parameters `tmpltA`, `tmpltB`, `tmpltR` can have the following values:

- `#TMPLT_UNKNOWN` - usually the most recent patgen pattern
- `#TMPLT_KNOWN` - typically an older pattern from external memory
- `#TMPLT_MODIFIED` - typically an averaged pattern

The score returned in "b" is similar to the score returned by RECOGSD, but is different from the composite score returned by RECOGSV, below.

RECOGSV (val **word**, val **size**, val **sec_level**, val **num_tmpls**) [OPERATION]

Recognizes a password sequence spoken by a single user.

Input: **word** word number in sequence (1...size)
 size number of words in sequence (1,2,3, or 4)
 sec_level security level (0,1,...5) [see notes]
 num_tmpls number of templates in external memory to test (1,...)

Output: a sequence decision: 0 = pass; 1 = fail; FF = undecided
 b current word: 0 = pass; 1 = fail
 r7 best class (if r7=num_tmpls then error)
 r6 current composite score (0=bad, 255=good)

Notes: Requires external memory handler at location `get_byte_sdv`.

The value returned in "score" is significantly different from the score returned by the TRAINSV macro.

Unlike Speaker Dependent, SV makes a pass/fail decision on each utterance and never returns an "undecided" result for the current word (b).

Expanded RECOGSV Macro Notes

This macro offers a variety of configurations for SV recognition. By setting size appropriately, as many as four passwords may be verified sequentially. The `sec_level` parameter controls the security level, which is used by the verifier to make a decision regarding a match. All speaker verification systems are subject to two types of errors: a False Accept (FA), in which an impostor is allowed access, and a False Reject (FR), in which the true speaker is prevented access. The relative importance of each kind of error depends on the application. In general, the security level controls the tradeoff between FA and FR errors, while using multiple passwords may reduce both kinds of errors. The default (intermediate) security level is designed to produce "equal error rates" (FA=FR) with any selected number of passwords. Higher values of `sec_level` decrease the FA rate at the cost of increased FRs (acceptance more difficult). Lower values of `sec_level` make acceptance easier.

The special value of zero (0) for `sec_level` forces SV into a special mode appropriate for Continuous Listening applications. This value causes the size parameter to be forced to one (1) regardless of the value passed. It also causes SV to use a special "extra low" security level that is more appropriate for most CL applications. Other `sec_level` values may be used with CL at the programmer's discretion. NOTE: To reduce false triggering, the CLSV word should be used as a "wakeup" word, and should not be used to invoke actions. See Continuous Listening below.

Enforcement of a strict sequence for a multi-word password is accomplished by setting `num_tmplts` to 1. This forces a match to a single template. Setting `num_tmplts` to size permits any of the user's passwords to be spoken at each utterance time. The memory co-routine must be aware of the ownership and position-in-sequence of all templates in order to supply the correct templates. See the sample code below and in the `\MEMORY\SRAM\SD_SV`, `\MEMORY\SEEPROM\SD_SV`, and `\MEMORY\FLASH\SD_SV` directories. A Speaker Verification application knows the identity of the speaker before the password is spoken, so it is only necessary to test those templates owned by the claimed speaker. The code may also be used in a Speaker Identification mode by testing the pattern of the first utterance against all stored passwords if the number of users is five or fewer. See the comments at "Comparison with Speaker Dependent Speech Recognition" above.

The macro returns a "sequence decision" in register A after each word is verified. Several words may be required before a "pass" or "fail" is returned because the algorithm computes a composite score based on all the words. The "sequence undecided" value is returned until the sequence pass/fail decision can be made. Good security design would typically require the user to speak all of the passwords (maximum of four) before making the decision known, even if a decision is available earlier. Use of the sequence decision return value allows the application designer to employ "early accept" and/or "early reject" if desired. The designer may also want to reject a sequence if any one of the words would fail on its own. This may be accomplished by using the "current word" status returned in register B. The value of `sec_level` for an entire sequence is fixed by the value passed on the call for the first word.

Note: The composite score returned in the "score" register is significantly different from the score returned by the corresponding SD macro, `RECOGSD`, or by the `TRAINS` macro.

Training Code Flow

The overall flow for SV is very similar to SD, consisting of separate training and operation phases. See the "Training Code Flow" chapter of *Using Speaker Dependent Speech Recognition*, which is generally the same for SV.

Recognition Code Flow and Results

To recognize a word, the pattern is generated as described in *Using Pattern Generation*. If the pattern generation software returns no error, the `RECOGSV` macro is then called. During execution of `RECOGSV`, password templates are read as specified in the calling parameters. Each template is 128 bytes. The `RECOGSV` macro `CALLS` to `get_byte_sdv` 128 times for each of the `num_tmplts` being tested, and there may be additional system reads as well, so many external memory accesses may be required. The memory co-routine is responsible for determining the proper memory address of each byte, and for switching to a new template when necessary, as illustrated in the samples.

Recognition time for speaker verification is substantially slower than speaker dependent—roughly 30 msec per template plus any extra time for external memory access. Recognition time is generally not an issue, since a true SV application needs to test only a few templates for each word recording.

Sample Code

The following short program fragment illustrates the use of the SV recognition macro in a multi-word application. Refer to the example in *Using Pattern Generation* for sample code illustrating prompting and pattern generation. The only difference between the training sequence for SD and SV is the names of the

macros, so please refer to the "Sample Code" section, TRAINING of section 5, Using Speaker Dependent Speech Recognition for that sample. The sample below, taken from the SAMPL364\SPKRV-ER sample, illustrates only the verification phase of multi-word speaker verification. If a single pass-word is used, recognition flow is similar to Speaker Dependent. A more sophisticated application would set flags for conditions like EARLY_EXIT or EXIT_1_FAIL, and would not announce failure until after the user speaks the entire sequence. See the PASSWORD demo for a simple single-word SV application.

The RecogSV macro returns the "best class" in register r7. Normally this value is not needed. If it is needed, be sure to save it before invoking exit_xread_sdv, as this call overwrites r7.

```

AskPwd:                                ;VERIFICATION - VERIFY A PASSWORD STRING
mov   word,#0                          ;first word of sequence
AskPwd1:
    cp   word,ctr                      ;if all words recognized
    jnc  AskPwdTst                     ; test final decision

    mov  msg1,#sen_say_x_pwd           ;prompt message "please say your <nth>
password"
    add  msg1, word
    call ask_and_record                ;prompt and create template
    cp   a,#0
    jnz  AskPwdError                   ;error (bad pattern)

    IF SEQ_RANDOM                      ;SEQ_RANDOM<>0: ALLOWS RANDOM SEQUENCE OF
PASSWORDS
        mov dpl,#LOW tmplt0           ;set start of first pattern
        mov dph,#HIGH tmplt0
        call init_xread_sdv           ;init co-routines
        mov trash,word
        inc trash
        RecogSV trash,ctr,acc_level,ctr ;recognize against all templates
    ELSE                               ;SEQ_RANDOM=0: ENFORCE STRICT SEQUENCE OF PASS-
WORDS
        mov dph,word                  ;calc offset for word
        mov dpl,#0                    ; ** (for this sample, templates are
placed)
        shr dph                        ; ** (on 128 byte boundaries for simplicity)
        rrc dpl                        ; ** (in calculating addresses)
        add dpl,#LOW tmplt0            ;set start of specific pattern
        adc dph,#HIGH tmplt0
        call init_xread_sdv           ;init co-routines
        mov trash,word
        inc trash
        RecogSV trash,ctr,acc_level,#1 ;recognize against specific tem-
plate
    ENDIF

    call exit_xread_sdv                ;finish up co-routines

    IF EARLY_EXIT                      ;EARLY_EXIT<>0: ALLOWS STOPPING BEFORE ctr
WORDS

```



```

        cp    a,#0FFh
        jnz   AskPwdTst      ;exit if pass or fail
    ENDIF      ;EARLY_EXIT=0: FORCE ALL WORDS TO BE SPOKEN

    IF EXIT_1_FAIL      ;EXIT_1_FAIL<>0: FAILS IF ANY ONE WORD
FAILS
        cp    b,#1
        jz    AskPwdFail    ;exit if any single word fails
    ENDIF      ;EXIT_1_FAIL=0: ALLOW PASS EVEN IF ONE WORD
FAILS

        inc   word
        jmp   AskPwd1
AskPwdTst:      ;test final pass/fail decision
        cp    a,#1
        jc    AskPwdPass    ;accepted
        jz    AskPwdFail    ;rejected
        jmp   AskPwdError    ;undecided
        .
        .
    .

```

7. Using Continuous Listening

Continuous Listening (CL) is a variation of Sensory's speech recognition technology that provides the capability to listen continuously for a "trigger" word or phrase to be spoken. Continuous Listening is suitable for use in Speaker Independent/Speaker Dependent/Speaker Verification applications. This technology was previously referred to as "CL4". In addition to CL, release 6 supports Wordspotting (WS), which may be used with Speaker Dependent only.

Continuous Listening has been significantly improved and simplified in release 6 by incorporation of speech framing technology. In the discussion below, the term "word" is used in the strict sense, but also in the general sense of an utterance -- to include individual words as well as brief phrases.

Comparison of CL and WS

Sensory supports two different methods of recognizing unprompted utterances. This section compares and contrasts the two methods. It should help identify the appropriate method for a given application.

In CL applications, a word (or multi-word sequence) is spoken with pauses surrounding each word. For example, the phrase "place quick call" is spoken as "<pause>place <pause> quick <pause> call<pause>". Each word is separately recognized, but the speaker must intentionally create the separations between words. Because each word is independently recognized, this CL method can be used with either SI or SD/SV. The recognition is unconstrained, so each word could be recognized from a set of SI or SD words. That is, CL can listen for several utterances in parallel. For example, the first word could be from a set of SI words consisting of {File, Edit, View}. Depending on the result of the recognition, the second word could be from a set of SI or SD sub-menus.

WS applications allow the trigger words to be embedded within other speech. Using the previous sample phrase, WS could - in principle -- recognize the three trigger words from the sentence "I would like to

place a quick telephone call, please". The software ignores irrelevant words and "spots" the trigger words. In practice, such a fast sequence of short words may not work very well. Presently WS can be used only with Speaker Dependent recognition technology. WS can listen for only one word at a given moment in time.

CL technology cannot detect a trigger phrase embedded within a sentence, but only a slight silent period is required before the trigger phrase may be issued as a command. For example, the trigger phrase "get mail" would not be recognized in the sentence, "I hardly ever get mail". It would be recognized in the sentence, "Let me check if I have received your message <pause> Get <pause> mail<pause>". Wordspot could potentially recognize the "I hardly ever get mail" utterance).

The following sections describe how to implement Continuous Listening. See Using Wordspot for information on implementing WS using Speaker Dependent technology.

Continuous Listening using Speaker Independent/Speaker Dependent/Verification

The Continuous Listening flow consists of four main states: 1) initialization, 2) pattern generation, 3) checking for appropriate duration, and 4) recognition. When intermittent extraneous speech or noise is ongoing, the software constantly cycles between states 2 and 3, occasionally attempting to recognize a random sound of the appropriate duration. Most of these recognitions fail because the sound is not the trigger word. When the background is quiet, the software spends most of the time in state 2 waiting for a word to be spoken.

Although continuous listening can be implemented with a single trigger word, Sensory strongly recommends using two or more words in sequence (the first word acts as a "gateway"). This will greatly reduce the number of false triggers (random speech recognized as the trigger word). The first trigger word in a sequence should be a specific gateway word, such as "lights". The final trigger word may be a specific action word or part of a command set, such as "on" or "dimmer". Trigger words in a sequence will need to be spoken with approximately 250 msec. of silence between each word. The sample uses two words. Note that each word in the sequence is a general recognition, so it could be any member of an SI set or a set of trained SD/SV templates. That is, CL offers multi-word parallel recognition.

An application using SD or SV for CL must provide a way for the user to train the trigger word(s). The training phase should be implemented as described in Using Speaker Dependent Speech Recognition. Pattern generation and recognition modules are described in other chapters of this manual. This section emphasizes the special steps required to use continuous listening. Be sure you understand the appropriate recognition technology section before beginning this section. The RSC-364 Development Kit offers two complete samples of Continuous Listening: the \SAMPL364\SPKRVER sample can be configured for 2-word SV Continuous Listening; and the \SAMPL364\CLSI\CLSI sample illustrates 2-word SI Continuous Listening.

During continuous listening, the software occasionally checks the duration or attempts to recognize a sound. At such times it does not listen, so a trigger word spoken at these times would be missed. Both of these operations are fast, so the period of non-responsiveness is greatly reduced compared to previous releases. To improve accuracy, Sensory recommends use of LEDs or similar visual cues to indicate that the RSC is ready to receive a command word. See the sample code for an example.

Similarity to Common Pattern Generation Technology

The pattern generation code used for Continuous Listening is the same code used for other recognition technologies, but the lower level routines are called in a slightly different way. The details of pattern generation are described in Using Pattern Generation. That section should be studied carefully in conjunction with this section; that information is not repeated here. The samples below illustrate the special way of using the pattern generation routines for CL. The CL macros provide the special controls needed to tune the software's performance to fit the application.

Similarity to Recognition Technologies

The training and recognition phases of Continuous Listening are accomplished as described for each recognition technology. Refer to the appropriate chapter in this section. Using CL with SI words requires no training.

Initialization

Initialization is identical to the specific recognition technology used.

Includes

The CL speech recognition routines should be called via macros. The macros and parameter definitions are contained in \LIB364\MACROS\CNTLSTN.INC. Also see the INCLUDE comments in the appropriate recognition chapter. The Speaker Verification CL sample also includes a local file, CLSV.INC. This file allows the developer to configure the application for the specific vocabulary. The CLSI sample program uses an analogous file, CLSI.INC.

Linking

Refer to the appropriate recognition chapter for linking information. In addition to the object files described there, you must link in \LIB364\CL\CNTLSTN.O and the callout handler, \LIB364\CL\CL_H (or an application-specific version of CL_H). The Continuous Listening routines make callouts to several different external co-routines depending on the state of the CL recognizer. The sample code in CL_H illustrates how to provide a visual indication of the state using LEDs. These callouts are made too infrequently to be useful for interrupting the program - use the PATGEN_H handler for termination.

Macros

Continuous Listening contains four macros. The macro definitions have been adjusted for normal demo performance, but they may be redefined to change their operating characteristics for specific applications (see parameters and performance below).

The continuous listening input parameters are defined as follows:

= 1-byte integer constant; *reg* = 1-byte register variable; *val* = either # or *reg*.

LISTEN (*val* **cl_performance**)

Initializes continuous listening, including loading the filter control file.

Input: **cl_performance** tradeoff speed and accuracy
1=fastest, 2=intermediate, 3=most accurate

Output: none

Notes:

Parameters:

CLPATGEN (*reg* **utterance**) [for speaker dependent and verification]
 CLPATGENW (*reg* **utterance**, *val* **page**) [for speaker independent]
 Generates pattern in preparation for recognition of "trigger" word.
 Input: **utterance** position of word in a sequence (start with 0).
 page page address where weights begin (a page is 256 bytes).
 Output: a : 0 = ok; 1 = timeout (no data); 2 = too long (too much data);
 3 = too noisy; 4 = too soft; 5 = too loud; 6 = spoke too soon;
 7=too choppy (too many segments); 8=bad weights;
 FF = interrupted
 Parameters: (may be redefined)
 CONT_PRE_SIL - 16-bit silence period to process before quitting (no sound)
 for the first word of the sequence (utterance = 0).
 Notes: Automatically includes silence measurement and up to 100 milliseconds of delay before processing starts.

CHECKDUR (*val* **length**)
 Checks duration of sound to see if reasonable.
 Input: **length** index of word's duration: 0=short, 1=medium, 2=long,
 3=variable/unknown, 4=precise (see clsv.inc for examples)
 Output: carry flag: 0 = ok; 1 = too long
 Parameters: Precise duration values may be specified by over-riding the macro.
 r0 = minimum duration (when a=4)
 r1 = maximum duration (when a=4)
 The duration here is measured in units of 16 milliseconds.

Parameters

The continuous listening macros use the following parameters to control operation characteristics.

The CONT_PRE_SIL parameter determines how long CLPATGEN listens to silence before returning. The time in seconds that CLPATGEN will listen to silence before returning is CONT_PRE_SIL/80. This should never be less than 10 seconds (800). In environments where the background noise is steady, it can be set to several minutes.

The **length** parameter limits the length of an utterance that will be passed on to the recognition phase. Words that are either longer or shorter than the window defined by length will be ignored and no recognition will be attempted. The CLPATGEN macro may process up to 3.2 seconds of data, but trigger words are usually 1 second or less. Setting length to SHORT is appropriate for trigger words such as "on" or "off". Words of longer durations should set length to MEDIUM or LONG. Using VARIABLE allows a wider window appropriate for SV words. For precise control of duration, set length to PRECISE and specify minimum and maximum durations in r0 and r1.

Performance

Some applications of Continuous Listening require maximum accuracy with fewest false triggers. This configuration is appropriate for an application such as a light switch, where a false trigger may be more than a minor inconvenience. A consequence of this high accuracy is relatively slow responsiveness. Other applications - games, for example - can accept occasional false triggers, but need fast response.

The tradeoff between CL accuracy and responsiveness is controlled by the parameter, `cl_performance`. This parameter may be set to `CL_FASTEST`, `CL_MEDIUM`, or `CL_MOST_ACCURATE`, with the default value giving the highest accuracy, but the slowest response. NOTE: The improved silence methods in release 6 mean that even with `cl_performance` set to `CL_MOST_ACCURATE`, response is fast. The speedup effect of `cl_performance` is more dramatic with SD/SV than with SI recognition.

Code Flow

In the example using speaker verification recognition, Continuous Listening uses templates made from averaging two patterns to increase accuracy. The template training process is described in Using Speaker Verification Technology. The discussion below uses "CLPATGEN", but SI applications should always call CLPATGENW.

After the templates have been prepared (if necessary), the application should call LISTEN once to initialize the code and to specify the CL_PERFORMANCE level. Then begin a loop with a call to CLPATGEN (to generate a pattern to recognize). When CLPATGEN returns a value indicating it has a pattern, the duration should be checked with CHECKDUR, and patterns which fail (too short or too long) should be ignored (return to CLPATGEN). If the duration is OK, then a recognition routine can be called in the normal manner.

If the first word is recognized, code flow should move on to check any additional words, repeating the CLPATGEN, CHECKDUR, and recognition cycle above, making sure to indicate that this is not the first word of the sequence. LISTEN should not be called between multiple trigger words. When the final word is recognized, the code can then perform the requested action. The CLSV code sample illustrates a single acceptable utterance for the second word, but straightforward modifications allow testing for any of several words in the set (for example, different actions on "up" and "down").

Typically the CLPATGEN, CHECKDUR, recognition loop will recur many times without a trigger. During even a brief lull in the conversation, duration checking and recognition attempting will run quickly and return to CLPATGEN. It can be useful in an application to provide a visual indication of when CLPATGEN is running. The sample program uses a green LED.

Most of the time, CLPATGEN will hear continuing conversation (or nothing) and will return an error or a pattern with the wrong duration, in which case CLPATGEN is called again. Occasionally CLPATGEN will falsely detect a word of an appropriate duration and call recognition to detect the trigger word. On some of these occasions the utterance will pass the first trigger word recognition and the second word will be listened for. Only rarely will the entire sequence false-trigger.

NOTE: Using a single word to activate an action rather than a two-word sequence as described above may give unacceptably high false trigger rates.

Sample Code

The following program fragment illustrates the use of the CL recognition macros. Refer to the examples in the various Using... sections for the actual code illustrating prompting, pattern generation, template production, template saving, and template averaging. This example shows only the relevant Continuous Listening portions (with SV recognition). The comments for the various CL_XXX routines indicate the action performed by the demo code, but the developer can customize these actions. See the sample code in \SAMPL364\SPKRVER\ for the complete speaker verification application. To configure this sample for Continuous Listening, enable the "set runType=cl" directive in BUILDSV.BAT. An example of CL with SI is in \SAMPL364\CLSI\.

```
Cont_Listen:
    call    CL_Done

    Listen  #CL_BEST_ACCURACY      ;initialize
    cp      ctr,#0
    jnz     cl0
    SenTalkBoth    #sen_mem_empty,SENTENCE,#high SPEECH,#VR
    jmp     main1
cl0:
    mov     word,#0                ;look for 1st CL word
    cp      a,#0FFh
    jz      cl_exit                ;if interrupt, exit
cl_led_on:
    cp      word, #1
    jc      >ready
    jmp     >trig
%ready
    call    CL_Ready                ;turn on green LED
    mov     length, #length0
    jmp     cl_record
%trig
    call    CL_Trigger                ;turn on yellow LED
    mov     length, #length1
cl_record:
    CLPatGen word                    ;Do continuous listening record
cl_led_off:
    call    CL_Busy

    cp      a,#0FFh                ;if interrupt, exit
    jz      cl_exit
    cp      a,#0
    jnz     cont_listen              ;if record error, continuous listen

    CheckDur length                  ;check if word is right duration
    jc      cont_listen

    call    CL_Busy
    mov     dpl,#0                  ;word * 256
    mov     dph,word
    shr     dph                      ;/2 = 128 bytes/pattern
    rrc     dpl
    add     dpl,#LOW tmpl0          ;add start of first pattern
```

```

        adc      dph,#HIGH tmplt0
        call     init_xread_sdv          ;init co-routines
        RecogSV #1,#1,#0,#1             ;use CL score, check just one word
        call     exit_xread_sdv          ;finish up co-routines

        cp       a,#1
        jz       cont_listen             ;if fail, continuous listen

        inc      word                    ;next word
        cp       word,ctr                ;if all words recognized
        jz       cl_accept               ; accept

        jmp      cl_led_on                ; record next word

cl_accept:
        call     CL_Activate
        CallTalkBoth #msg_sensopow,#high VPsensopow,#VR

        call     CL_Done
        jmp      cont_listen             ;continuous listen

cl_exit:

```

8. Using Wordspot SD Speech Recognition

Wordspot (WS) is a variation of Sensory's Speaker Dependent speech recognition technology that provides the capability to "spot" a trigger word or phrase, even when the word is embedded in fluent speech. In addition to Wordspot, this release supports Continuous Listening for use in Speaker Independent/Speaker Dependent/Speaker Verification applications.

Wordspot is new in release 6.0. In the discussion below, the term "word" is used in the strict sense, but also in the general sense of an utterance -- to include individual words as well as brief phrases.

Comparison of CL and WS

Sensory supports two different methods of recognizing unprompted utterances. This section compares and contrasts the two methods. It should help identify the appropriate method for a given application.

In CL applications, a word (or multi-word sequence) is spoken with pauses surrounding each word. For example, the phrase "place quick call" is spoken as "<pause>place <pause> quick <pause> call<pause>". Each word is separately recognized, but the speaker must intentionally create the separations between words. Because each word is independently recognized, this CL method can be used with either SI or SD/SV. The recognition is unconstrained, so each word could be recognized from a set of SI or SD words. That is, CL can listen for several utterances in parallel. For example, the first word could be from a set of SI words consisting of {File, Edit, View}. Depending on the result of the recognition, the second word could be from a set of SI or SD sub-menus.

WS applications allow the trigger words to be embedded within other speech. Using the previous sample phrase, WS could - in principle -- recognize the three trigger words from the sentence "I would like to place a quick telephone call, please". The software ignores irrelevant words and "spots" the trigger

words. In practice, such a fast sequence of short words may not work very well. Presently WS can be used only with Speaker Dependent recognition technology. WS can listen for only one word at a given moment in time.

CL technology cannot detect a trigger phrase embedded within a sentence, but only a slight silent period is required before the trigger phrase may be issued as a command. For example, the trigger phrase "get mail" would not be recognized in the sentence, "I hardly ever get mail". It would be recognized in the sentence, "Let me check if I have received your message <pause> Get <pause> mail<pause>". Wordspot could potentially recognize the "I hardly ever get mail" utterance).

The following sections describe how to implement WS using SD recognition. See Using Continuous Listening for information on implementing CL using Speaker Independent/Dependent/Verification.

WS Recognition with SD technology

In Sensory's Continuous Listening technology, the pattern generation phase and the recognition phase alternate. In Wordspot, pattern generation and recognition both run concurrently as multi-tasked operations. This "listening while recognizing" feature gives Wordspot its unique capabilities. The ability of Wordspot to spot a trigger word within a sentence is useful for unprompted recognition, but it can also be valuable in conjunction with a prompt. For example, if a child is asked to identify, say, the green animal in a picture, wordspot would recognize "alligator", but also "the alligator", "I think it's an alligator", and "the alligator is green".

Although Wordspot can trigger based on a single word, Sensory recommends using two or more words in sequence unless a prompt is used. This will greatly reduce the number of false triggers (random words falsely recognized as the trigger). Each word should be a multi-syllabic word or a brief phrase. Wordspot will not work optimally with very short words because they lack sufficient acoustic information. Conversely, long phrases may contain too much acoustic information. An ideal word will have 3-5 syllables. The table below suggests some good and bad Wordspot words.

GOOD	NOT SO GOOD
hands free kit	phone
voice dialer	dialer
dial number	dial
call home	home
what time is it?	time

When used with multiple words, there may be a brief time after one word is recognized before the next word can be spoken. The time is needed for the application to read the second template from memory and re-start the software. Depending on the words, this time delay may or may not be a problem.

As an example, consider the two trigger words to be "voice dialer" and "call home". When these two phrases are spoken together in a natural way, the recognition of the first phrase is typically complete before "dialer" is finished, so the second phrase is recognized without difficulty even when it follows

immediately. Each of the following phrases will trigger correctly:

"My **voice dialer** is smart enough to **call home**"
"I want my **voice dialer** to **call home**"
"**Voice dialer**, **call home**"

As a counter-example, consider the two trigger words to be "dial fast" and "start now". The following phrase will trigger correctly:

"Please **dial fast** and **start now**"

But is likely that this phrase will NOT trigger correctly:

"**Dial fast**, **start now**"

The problem is that the adjacent "st" sounds in these two words would commonly be run together in fluent speech and spoken as "dial faststart now". By the time the first word is recognized, the speech is well into the second word. The first example works because it is difficult to run together the words "dialer" and "call" and because "voice dialer" is long enough that it may be recognized before the word is complete. The best designs use words that are naturally separated by speech that is not part of the trigger phrase, or words that cannot easily be run together.

After the first word of a multiple-word sequence is recognized, Wordspot begins attempting to recognize the second word. If the second word is not spoken within about 3 seconds, Wordspot returns to listening for the first word. This prevents triggering from independent utterances out of context. It may be useful to use LEDs or other visual indicators of which word is being listened for. The sample program uses the green LED to indicate the first word, and the yellow LED to indicate the second word.

NOTE: Using a single word to activate an action rather than a multi-word sequence as described above may give unacceptably high false trigger rates.

An application using WS must provide a way for the user to train the trigger word(s). The training phase should be implemented as described in Using Speaker Dependent Speech Recognition. Be sure to use the PatgenWS macro for training. This section emphasizes the recognition aspects of Wordspot. Be sure you understand the SD recognition technology section before beginning this section. The RSC-364 Development Kit offers a Wordspot sample program in \SAMPL364\WORDSPOT.

Initialization

Initialization is identical to SD recognition.

Includes

The WS speech recognition routines should be called via macros. The macros and parameter definitions are contained in \LIB364\MACROS\WORDSPOT.INC. Also see the INCLUDE comments in the SD recognition chapter. Most Wordspot applications will use SD recognition flow and macros for training (template storing and retrieving, etc.).

Linking

Refer to the Using Speaker Dependent Speech Recognition for linking information. In addition to the

object files described there, you must link in the wordspot files \LIB364\WS\WORDSPOT.O, and \LIB364\WS\WSSTREAM.O.

Finally link in one of the external memory co-routine modules such as SRAM_SDV.O (or your replacement) and any other memory access, technology, or application modules you use. To use the WORDS ON CHIP option, link in \LIB364\SD_SV\ONCHIP.O as the on-chip memory handler, and also link \LIB364\SD_SV\AUXTMPLT.O, the auxiliary template handler. NOTE: Wordspot uses the on-chip memory reserved for templates 4 and 5, so only four templates (0-3) may be stored on-chip in Wordspot applications.

Pattern Memory Usage

Patterns and templates in the RSC-364 may be stored in external memory, or may temporarily reside in one or more of four areas within the 364 chip. The technology software identifies these four areas as:

#TMPLT_UNKNOWN - This is the area where Patgen places the new pattern.

#TMPLT_KNOWN - Typically used for retrieving a template from external memory.

#TMPLT_MODIFIED - May be used to hold a second pattern or average of two patterns.

#TMPLT_WS - This area is used by Wordspot to hold the template being spotted.

Except for Patgen and Wordspot, usage of these areas is determined by the application program through use of the macros described below. An additional template storage area is provided for use by the ON_CHIP feature. This feature allows storing up to four WS templates on chip when no external template storage is used. See Using On-Chip Template Storage in Chapter 3, for details.

NOTE: in this release, TMPLT_MODIFIED and TMPLT_WS are the same physical area. Application use of this area limits on-chip template storage to 4 templates.

To conserve storage space, the template management routine compresses the template data before storing it, and decompresses it when retrieving. The template size information (10 or 16) is also stored with the template data.

Macros

The macros described in Using Speaker Dependent Speech Recognition should be used for training. Note that a special macro, PatgenWS, is used for pattern generation during wordspot training. Wordspot recognition uses one macro.

The wordspot input parameters are defined as follows:

= 1-byte integer constant; reg = 1-byte register variable; val = either # or reg.

WrdSpt (val ws_performance, val timeout) [operation]

Input: ws_performance range 1-4: controls accept/reject tradeoff (see below)

timeout controls how long to listen before returning to application

0= no timeout; returns only after successful recognition

1= returns after 3 seconds if no recognition

Output: a result:

0=OK;

1=timeout;

FF=interrupted

Notes:

The template must have been previously loaded into the `tmplt_ws` area

Parameters

The parameter `ws_performance` controls the tradeoff between false accepts (triggers on an un-trained word) and false rejects (fails to trigger on a trained word) in WS recognition. When set to 1, a word is recognized with wide pronunciation tolerance, and a wide latitude is allowed in the speed with which the word is spoken. When set to 2 or 3, these limits are progressively tightened. When set to 4, the limits are most strict, and false triggering will be minimized. The developer (or end user) can tune the value of `ws_performance` to suit a particular application. Typically a setting of 2 or 3 should be best.

The parameter `timeout` controls how long Wordspot listens for the trigger word before giving up. Set `timeout` to 0 to listen without a time limit. This is the appropriate setting for non-prompted use. Set `timeout` to 1 to listen for 3 seconds. This is appropriate when used with a prompt.

Recognition Code Flow and Results

A template must be trained for each word before recognition is attempted. Refer to Using Speaker Dependent Speech Recognition for training information. Recognition code flow requires that the correct template be loaded into the `TMPLT_WS` area using the `GetTmplT` macro (described in Using Speaker Dependent Speech Recognition). The `WrdSpt` macro is then invoked. It will run until the trigger word is recognized, until it is interrupted (via the `PATGEN_H` handler), or until it times out. The recognition decision is controlled by the `WS_PERFORMANCE` parameter as described above. This process is repeated for each word in the sequence.

Wordspot templates in the RSC-364 are stored externally in compressed form occupying 128 bytes. The application, using the appropriate memory co-routines, is responsible for determining the proper memory address and for loading a new template when necessary. Since management of individual templates in Wordspot is the responsibility of the application, the developer needs to be familiar with the memory co-routine procedures. See also Using Co-Routines for External Memory Access.

Sample Code

The short program fragment below illustrates the use of the WS recognition macro. Refer to the example in Using Pattern Generation for sample code illustrating prompting and pattern generation. Refer to the example in Using Speaker Dependent Speech Recognition for sample code illustrating template training. The example, adapted from the sample program `\SAMPL364\WORDSPOT`, shows only the wordspot recognition portion with one or two sequential words. It speaks the number of words spotted (one or two) after recognizing the sequence, then continues looping attempting another recognition sequence.

```
; parameter "knob" is set upon entry
AskPwd:
        cp          ctr,#0
        jz          main          ;need at least one trained word in
```

```

memory

        mov     word,#0                ;start at word 0
        mov     timeout,#TIMEOUT_NO    ;value is 0 (listen until recog-
nized)

        mov     a,#sen_say_a_word
        SenTalkBoth    a,SENTENCE,#high SPEECH,#VR

AskLoop
        mov     dpl,#LOW data_offset    ;set start of first pattern
        mov     dph,#HIGH data_offset
        call    init_xread_sdv          ;init co-routines
        mov     a,word
        call    validate_template_sdv
        GetTplt    #TMPLT_WS            ;get templatekPwdPass:

        call    exit_xread_sdv          ;finish up co-routines

        cp      word,#0                ;LEDs indicate which word is being
spotted
        jnz     >yellow
        GREENON
        jmp     >go
%yellow
        YELLOWON
%go
        WrdSpt   knob,timeout
        GREENOFF
        YELLOWOFF

        cp      a,#1
        jc      AskPwdPass
        jz      AskPwdTimeout          ;timeout
        jmp     interrupt

AskPwdPass:
        inc     word                    ;advance to next word
        mov     timeout,#TIMEOUT_YES    ;return in 3 seconds if not recogni-
tion

        cp      word,ctr
        jnz     skip_announce

        mov     A,#sen_zero
        add     a,word
        SenTalkBoth    A,SENTENCE,#high SPEECH,#VR

skip_announce

        cp      word,ctr
        jc      AskLoop

```

```

mov      word,#0                      ;reset to word 0
mov      timeout,#TIMEOUT_NO
jmp      AskLoop
jmp      main1

AskPwdTimeout:
mov      word,#0                      ;reset to word 0
mov      timeout,#TIMEOUT_NO
jmp      AskLoop
SenTalkBoth    #sen_beep_beep,SENTENCE,#high SPEECH,#VR
jmp      main1

```

9. Using Dual Recognition Technology

Sensory's Dual Recognition Technology (DRT) provides fast, highly accurate recognition by combining Speaker Independent and Speaker Dependent techniques. This approach achieves higher accuracy than either method alone. DRT is suitable for recognition sets of fewer than 16 words: it requires a normal set of SI weights trained in advance on the entire vocabulary as well as a complete set of SD templates trained during operation of the application. During recognition, the pattern to be recognized is processed by both SI and SD. If either SI weights or a complete set of SD templates is not available, DRT may not be used. This release provides general support for recognition using DRT, but the focus is on digits ("fast digits"). Contact Sensory for help with other applications.

An application using DRT must provide for user training of one SD template for each vocabulary word. This is most simply done through an explicit training phase, as is usual for SD applications. Multiple users may be accommodated by training and recognizing with separate SD template sets for each user. Alternatively, SD template training may be done in the background (perhaps without the user's knowledge or active participation). In this method, the product operates as SI-only at initial startup; as each word is recognized, the patterns are saved and averaged to create templates. Once templates have been created for all the words, recognition can switch over to using DRT. Using this technique allows the product to "adapt" to a specific speaker over time.

IMPORTANT

Several important considerations must be remembered if adaptation is to be used:

- 1) Initial recognition accuracy based on SI alone will be lower than the accuracy obtained later when DRT is used;
- 2) Before patterns are used to create templates there **MUST** be confirmation that the word was recognized correctly by SI, otherwise the templates will be trained with incorrect patterns.
- 3) If the product allows multiple users, each must have a separate set of templates, and the appropriate template set must be used for each user.

The attraction of starting with SI and improving performance "behind the scenes" by adapting automatically is appealing, but the confirmation problems are often so unwieldy that it is better to simply have an explicit training phase. Sensory strongly recommends this explicit training approach.

Common Pattern Generation Technology

The pattern generation portion of dual recognition technology is the same code used in speaker independent, speaker dependent, and speaker verification. The details of pattern generation are described in Using Pattern Generation. That chapter should be studied carefully in conjunction with this chapter; the information there is not repeated here.

As described elsewhere, patterns in the RSC-364 can be of two different sizes. Since DRT processes one pattern through both SI and SD, the pattern must be the correct size for both technologies. This is accomplished by using the macro PATGENW for DRT recognition, instead of PATGEN. This macro provides the pattern size and other parameters for the SI weight set to the pattern generator. NOTE: PATGENW with the target vocabulary SI weights must also be used during an explicit SD training phase.

Initialization

Initialization includes setting up the hardware and the interrupt vectors. This is easily implemented by linking-in the module \LIB\STARTUP\STARTUP. This module sets the processor speed to oscillator1 1 with 1 wait state, and initializes interrupt vectors. This clock state is required for pattern generation. In addition, certain I/O configuration must be defined. The module \LIB\STARTUP\CONFIG accomplishes this task for the standard Development Kit hardware, and can be modified for other designs. See System Configuration and I/O Mapping under the Software Development tab for more information.

The value SD_PERFORMANCE normally controls speed/accuracy tradeoffs in SD recognition. In DRT this parameter is ignored and performance is internally optimized for speed and accuracy.

STARTUP jumps to the label INITIALIZE, which should be at the start of the application program and needs to be PUBLIC'd for linker references.

Includes

The DRT recognition routines should be called using macros. LIBINCL the macros in \LIB364\MACROS\DRT.INC. Also include the usual macros for pattern generation, SI recognition, and SD recognition.

The recognition macros use memory locations that are defined in \LIB\MEMORY\MEMORY.INC, so you must also LIBINCL this file in the application program. See Using the RSC-364 Memory Map for detailed information.

The Speaker Dependent recognition portion of DRT requires external (or on-chip) read/write memory for storing templates. (A template is made by averaging two or more patterns to improve recognition.) The file \LIB364\SD_SV\XRAM_SDV.INC must always be included to define the memory co-routine interface (see Using Co-Routines for External Memory Access).

Linking

Link the application code with the initialization modules STARTUP and CONFIG in \LIB\STARTUP\, the modules ANALOG, FLOAT2 and INTMATH in \LIB364\UTIL\, and the DRT Recognition module, \LIB364\DRT\DRT. Link \LIB364\SD_SV\AUXTMPLT.O, the auxiliary template handler, which contains some DRT routines. Also link the SI recognition module, \LIB364\SI\RECOG and the SD recognition modules \LIB364\SD_SV\SD and \LIB364\SD_SV\TEMPLATE.

DRT Recognition must also be linked with the module RECOG_H. This module contains two memory handlers that allow the SI portion of DRT to access weights tables from either code space or data space.

DRT recognition uses neural net SI weights data contained in object (.O) format, which must also be linked. If the SI weights data are to be located in data space, they must be separately linked and RECOG_H must be appropriately modified as described elsewhere. SI weights used for DRT contain extra information and must be specially generated.

In addition, special SD weights for DRT applications must be linked. These special SD DRT weights are generated by Sensory and must be in code space. Included in this release are SD DRT weights based on "fast digits". Some applications may require custom SD DRT weights. Contact Sensory for advice. For the fast digit sample, the weight sets are \DATA\WEIGHTS\364\FDSI and \LIB364\DRT\FDDRT. The FDDRT SD weights (or replacements obtained from Sensory) should be linked with all DRT applications.

Also link the pattern generation modules BLOCK, NORMAL, NULOGFNS, PATGEN, DIGAGC, STREAM, and PATGEN_H, all in \LIB364\PATGEN\.

Finally link in one of the external memory co-routine modules such as SRAM_SDV.O (or your replacement) and any other memory access, technology, or application modules you use. If the vocabulary is small enough, the SD templates may be stored on-chip. To use the WORDS ON CHIP option, link in \LIB364\SD_SV\ONCHIP.O as the on-chip memory handler. NOTE: using WORDS_ON_CHIP conflicts with Pattern Queuing, which is commonly used for fast recognition. See Using On-Chip Pattern Storage in Chapter 3 for details of both pattern queuing and on-chip template storage. The sample program uses the on-chip memory for pattern queuing and stores templates in Serial EEPROM.

DRT/SI Memory Handler

The macros in DRT.INC read data from code space or data space, depending on the memory handlers in RECOG_H.A (see above). See the discussion under Memory Handler in the section Using SI Recognition for details.

Macros

The file \LIB364\MACROS\DRT.INC defines the macro used for DRT recognition. The input parameters for the DRT recognition macro are defined as follows:

= 1-byte integer constant; *reg* = 1-byte register variable; *val* = either # or *reg*,
daddr = 16-bit address in Data Space

RecogDual (*val* **siWeights**, *val* **index**, *val* **tmpltX**, daddr **digTmplts**, *val* **bank**)

Recognizes a pattern using both SI and SD recognition methods.

Input:	siWeights	page address of SI-digits weights
	index	has INDEX of REJECT_THRESHOLD (0-2)
	tmpltX	page address in Internal SRAM space of the unknown pattern. (normally (TMPLT_UNKNOWN))
	digTmplts	(16-bit) base address of SD_DIGITS templates in data space memory
	bank	bank address of SD_DIGITS templates (address bits 16-23)

Output: a best match (index, 0-NUM_CLASS) [NUM_CLASS=NOTA]
 b 100*combined log probability

Notes:

- 1) Assumes SD templates exist for all digits. If not, use SI recognition, not DRT.
- 2) Requires external memory handler `get_byte_sdv`.
- 3) Requires external `validate_template` routine.
- 4) The macro invokes a code entry point that uses fast digits SD DRT weights.
- 5) The index parameter controls the acceptance threshold. Approximately index percent of recognitions will be rejected as having insufficiently high confidence. Set index to 0 to accept every recognition, or 1 or 2 to reject 1% or 2%.
- 6) Return value b: the `log(probability)` is negative;
b is an 8-bit unsigned number, interpreted as a negative value
b=0 : `log_prob` = 0, confidence=100% [Special Value]
b=1 (= -255) : `log_prob` < -2.55, confidence < 0.3%
other b: UNSIGNED NEGATIVE 8-bit value in range -254 (2) < b < -1 (255)
the smaller the absolute 8-bit value of b, the lower the probability.

Training Code Flow

As explained above, the application may be divided into a training phase and a normal operation phase, or training may be performed in the background. Refer to "Training Code Flow" under Using SD Recognition for details on training. The important point for DRT is to be sure to use PatgenW during training.

IMPORTANT

If an explicit training phase is omitted and templates are trained in the background, the application MUST provide some means to confirm that the SI recognition was performed correctly before using the pattern to train a template. Unless this is done, a pattern incorrectly recognized by SI will be used to train the wrong template. The confirmation should be based on feedback from the user, either directly or indirectly through subsequent actions.

Recognition Code Flow and Results

To recognize a word (digit in this release) the pattern is generated as described elsewhere. If the pattern generation software returns no error, the RECOGDUAL macro is then called. During execution of the RECOGDUAL macro every template of the vocabulary is read at least once-some templates may be read multiple times. See more discussion about memory handlers for templates under Using SD Recognition.

Sample Code

The short program fragments below illustrates the use of the DRT recognition macro and of pattern queuing. Refer to the example in Using Pattern Generation for sample code illustrating prompting, silence measurement, and pattern generation. Refer to the example in Using SD Recognition for sample code illustrating template training. This example, taken from the sample program \SAMPL364\FAST-DIG, shows only certain Dual Recognition Technology and pattern queuing portions. The macros for pattern queuing are described in Chapter 3, Using On-Chip Pattern Storage.


```

; Queue the patterns for 7 digits in on-chip memory, then recognize the lot
;
AskPwd:
    cp        ctr,#MAX_CLASSES          ; all digits must be trained
    jc        main
    SenTalkBoth #sen_say_number,SENTENCE,#high SPEECH,#VR

    mov        digitindex, #0

    LdCtl                                ;set up filters
NumberLoop:
    mov        askctr,#2                ;3 tries before exit
    mov        msg1,#sen_click
    mov        error_cnt,#2
    call       prompt_user
    cp         a,#0
    jnz        AskPwdError              ;error

    cp         digitindex, #06h          ;queued 7 digits?
    jnc        RecQueueDigitPatterns
    QueuePat   digitindex, #TMPLT_UNKNOWN

    inc        digitindex
    jmp        NumberLoop

RecQueueDigitPatterns: ; recognize in reverse order
    mov        digitptr, #digits+6
    mov        digitindex, #6

QueueDigitPatterns1:
    mov        dpl,#LOW data_offset      ;set start of first pattern
    mov        dph,#HIGH data_offset
    mov        r7, #dataBank
    call       init_xread_sdv            ;init co-routines

    RecogDual #HIGH Wtfdsi, #1, #TMPLT_UNKNOWN, data_offset, #dataBank

    call       exit_xread_sdv            ;finish up co-routines
    mov        @digitptr, a

    dec        digitptr
    dec        digitindex
    jc         SpeakDigits

    GetQueuePat digitindex, #TMPLT_UNKNOWN
    jmp        QueueDigitPatterns1

SpeakDigits:
    mov        digitindex, #6
    mov        digitptr, #digits
; speak each digit..

```

10. Using Voice Record and Playback

The Voice Record and Playback software is available in a group of object code files for linking into a ROM image. This technology requires external memory such as SRAM or Flash to hold the compressed voice data. The Playback software can drive either or both of the RSC-364's two speech outputs, the DAC (analog waveform) and the Pulse Width Modulator (PWM). This section describes the procedure used to access Voice Record and Playback.

Initialization

Initialization includes setting up the hardware and the interrupt vectors. This should be implemented by linking-in a local copy of the module `\LIB\STARTUP\STARTUP`. The `STARTUP` module sets the processor speed to oscillator1 1 with 1 Wait State and initializes interrupt vectors. As a default, `STARTUP` sets the timer interrupt for speech synthesis only-to use voice record and playback, set the assembler variable `RECPLAY` to 1 at the beginning of the module (see System Configuration and I/O mapping). Sensory recommends making a local copy of `STARTUP.A` before editing.

In addition, certain I/O configuration must be defined. The module `\LIB\STARTUP\CONFIG` accomplishes this task for the standard Development Kit hardware, and can be modified for other designs. See System Configuration and I/O Mapping for more information.

`STARTUP` jumps to the label `INITIALIZE`, which should be at the start of the application program and needs to be `PUBLIC`'d for linker references. A recorder-specific version of `STARTUP.A` appears in the `\SAMPLES\RECORDER` directory.

Includes

The voice record and playback (RP) routines should be called using macros that are defined in `\LIB\MACROS\RP.INC`. This file also defines various constants used by the Record and Play code. The macros use a few memory locations that are defined in `\LIB\MEMORY\MEMORY.INC`. These files must be `LIBINCL`'d in the application program. See Using the RSC-364 Memory Map for detailed information.

RP uses external read/write memory, so you will need to include the definitions of the memory handler registers in `\LIB\MEMORY\XRAM.INC`. This would normally also be included in a separate device driver module that you must supply for the external memory device. See Using Memory Co-Routines for External Memory Access.

If your application uses other technology code (recognition, synthesis, etc.), include the appropriate `.INC` file(s) as described in the corresponding Using... section.

Linking

Link the application code with the local version of the initialization module `STARTUP`, the math module `\LIB364\UTIL\INTMATH`, and the voice record and playback modules `REC`, `POST`, `PLAY`, and `RP_H` in `\LIB\RP`. The `rp_h` file contains the default jumpout handler, which may be used to stop recording and/or playback. The record jumpout is called approximately 700 times per second. Voice Record and Playback uses resources from other technologies, so you must also link `\LIB\TALK\TALKTBL`.

The Voice Recording technology for the RSC-364 uses the thresholding algorithm in pattern generation. If thresholding is to be enabled, link all of the pattern generation modules BLOCK, NORMAL, NULOGFNS, PATGEN, DIGAGC, STREAM, and PATGEN_H (our your replacement handler of the patgen interrupt function). The pattern generation modules all reside in the \LIB364\PATGEN directory. These modules must also be linked if the Simultaneous Pattern Generation and Voice Recording feature is used (see below). Alternatively, if voice thresholding is to be disabled, and if your application does not include any form of recognition, none of these files have to be linked.

To use 3-bit and/or 2-bit compression, link CMP, 3BIT2BIT, RP3BIT and RP2BIT from the \LIB\RP directory, as appropriate (see compression below). Record and playback uses external read/write memory, so you will also need to link in your external memory handler(s), and the configuration module CONFIG.O (which defines buffers for memory management-see System Configuration and I/O Mapping).

Link in the memory handler for your specific external memory device. For example, the SRAM driver is \LIB\MEMORY\SRAM\RP\SRAM_RP. A different memory device requires a different driver. To accommodate long or multiple messages, Record and Play memory handlers should normally allow more than 16 bits of address. The technology code supports up to 24 bits of recording data address.

Link in any required technology code modules as described in the appropriate "Using..." section.

Thresholding

The recording code can dynamically check the signal level against a threshold. When the signal amplitude is small enough to correspond to a silence period the stored data encodes the duration of silence rather than the actual data itself. This technique improves the effective compression ratio.

RecordRP uses the threshold algorithm from pattern generation. The developer may choose to make voice recordings with or without thresholding. When thresholding is used, it can be applied to all silences, or alternatively it can be used only on the initial silence before speech begins. The sample, RECORDER, illustrates all three methods.

To make a voice recording with thresholding, first make the normal pattern generation macro calls with the pattern generation "run_how" macro parameter set to #RP_THRESH. This starts the pattern generation threshold operation running and returns. The RECORDRP macro is then invoked with its threshold parameter set to the desired thresholding algorithm. See the explanations and sample code below.

To make a voice recording without thresholding, simply invoke RECORDRP with the threshold parameter set to RP_NO_THRESH.

Voice Recordings may also be made simultaneously with pattern generation. Certain restrictions apply as described in the following section.

Simultaneous Pattern Generation and Voice Recording

Voice recording in the RSC-364 can run simultaneously (multi-tasked) with Pattern Generation. In this way a single utterance can produce both a pattern for recognition (or template averaging) and a voice recording. A voice dialing application, for example, could use this feature to record a "voice tag" to be

saved along with the template. Later, when an utterance is recognized with that template, the voice tag could be replayed as confirmation that the correct name was recognized ("Dialing John Smith").

Using Simultaneous Pattern Generation and Voice Recording is straightforward. Set up the memory handler for Voice Recording as described elsewhere in this section. Initialize pattern generation as described under Using Pattern Generation, then invoke the PATGEN (or PATGENW) macro with the run_how parameter set to #BACKGROUND. This step starts patgen, but it pauses, waiting until RecordRP is also running. Finally invoke RecordRP to begin the simultaneous patgen and recording. The threshold parameter for RecordRP can be set to any desired value. The sample programs \SAMPL364\SI6 and \SAMPL364\SPKRDEP can each be configured to illustrate simultaneous pattern generation and voice recording.

The duration of a Simultaneous Voice Recording is determined by the pattern generation algorithm. Once Patgen detects that the utterance is complete, the voice recording stops. The POST macro (see below) can be used to clean up the recording before saving it.

NOTE: when recording simultaneously, the recording is stopped when pattern generation terminates or when the allocated recording memory is exhausted. In the latter case pattern generation continues until the utterance is finished (up to 3 seconds). The pattern will be correct, but the recording will contain only the first part of the utterance.

Compression

The RSC-364 provides three levels of voice compression. The default 4-bit compression produces a maximum data rate of 4067 bytes per second. 3-bit compression lowers the rate to 3389 bytes per second, and 2-bit gives the best rate at a maximum of 2711 bytes per second. These rates may be substantially lower if silence compression (thresholding) is invoked and there are many silences in the recorded speech (see below).

During real-time recording the RP module saves voice data in 4-bit samples. Two samples are packed into one byte and a JMP is made to an external memory handler for storage. The external handler is responsible for managing pointers, counters, and other variables related to external memory. See Using Co-Routines for External Memory Access for details on external memory management. Note that some memory devices may require a memory buffer during recording. Up to 128 bytes of buffer space can be made available. See the \LIB\MEMORY\SPFLASH library for an example (using the 29EE020 flash).

After the real-time processing is completed the stored data can be selectively "post-processed", then compressed to either 3-bits or 2-bits using macros in this package. Compression to fewer bits reduces both memory and quality, so compression can be managed intelligently by the application to maintain highest quality. Post processing normalizes the gain and -- if thresholding is enabled -- removes clicks and strips any leading silence.

The developer can select whether 3-bit and/or 2-bit code is included in the application. Omitting this code can save a substantial amount of code space if 3-bit or 2-bit compression is not needed. Edit (a local copy of) the file 3BIT2BIT.A, and set or clear the appropriate compiler flags THREEBIT and TWOBIT (this module creates dummy stubs for modules that will be omitted from the link). Re-assemble 3BIT2BIT, and include it in the link. If either TWOBIT or THREEBIT is true, you must also link CMP and the appropriate module(s) RP3BIT and/or RP2BIT. The example below shows how to setup

3BIT2BIT.A and the LINK.CMD file for a system that includes 2-bit, but not 3-bit. NOTE: the 3-bit and 2-bit code modules are quite large, so an application can save ROM memory by omitting compression if possible.

```
; 3BIT2BIT.A - 3-bit and 2-bit inclusion for record/playback

; v0001 30jan97 - Initial version for MTP-2

; RP program allows user to include or exclude 3-bit and/or 2-bit compres-
sion.

THREEBIT      equ    0      ;set to 1 to include 3-bit, and link rp3bit
                                ;set to 0 to exclude 3-bit, do not link rp3bit

TWOBIT                equ    1      ;set to 1 to include 2-bit, and link rp2bit
                                ;set to 0 to exclude 2-bit, do not link rp2bit
.
.
.

; LINK.CMD
.
.
.
\lib\rp\rec
\lib\rp\post
\lib\rp\play
\lib\rp\rp_h
\lib\rp\cmp
\lib\rp\3bit2bit
\lib\rp\rp2bit
```

Macros

There are ten macros in the voice record and playback package, although most applications need only a few. Alternate versions of the PLAYXXX and PLAYFASTXXX macros output to the DAC, to the PWM, or to both. The macros used for recording and playback may be interrupted by arbitrary events as determined by the application. This is achieved by the technology code periodically executing a "jumpout" to the application code. The user must supply handlers for these jumpouts. If there is no need for interrupting the recording or playback, the handler can just jump back. If the handler wants to interrupt, it sets the carry flag. The external logic for deciding about interrupting is under the programmer's control, but it should execute as quickly as possible to avoid missing voice samples. The default (stop if Button A is pushed) jumpout handlers are in the module \LIB\RP\RP_H.A. This should be used as a template and linked with all programs that use the macros in RP.

Note that some parameters have changed with this release. Although not formal parameters in the calling sequences, arrays of external storage data pointers are needed for RP. The technology code allocates storage for all the required pointers. Some of these variables must be initialized with begin and end addresses before invoking macros. See the sample code.

Input parameters are defined as follows:

= 1-byte integer constant; reg = 1-byte register variable; val = # or reg.

RECORDRP (val **maxtime**, val **thresh_type**)

Records speech and compresses to 4 bits in real-time, optionally thresholding.

Input: **maxtime** Maximum time to record (in seconds). Zero indicates no time limit.

Also see parameters notes below.

thresh_type controls thresholding method

RP_NO_THRESH -- all samples recorded

RP_TRIM_THRESH - threshold initial silence only

RP_FULL_THRESH - threshold all silences

Output: none

Notes: Automatically detects if patgen is running and uses it for threshold information. See thresholding above.

Calls handler at stop_rp_recd to allow interrupt.

Terminates if maxtime is reached, if allocated memory is filled, or if recording simultaneously and Patgen terminates.

Recording starts at the address provided to init_xwrite_rp.

For DAC (only) output:

PLAYRPDAC ()

PLAYFASTRPDAC ()

Plays compressed speech through the DAC output. PLAYFASTRP plays at accelerated rate (for message scanning).

Input: none

Output: none.

Notes: Calls handler at stop_rp_play to allow interrupt.

Playback starts at the address provided to init_xread_rp.

For PWM (only) output:

PLAYRPPWM ()

PLAYFASTRPPWM ()

Plays compressed speech through the PWM output. PLAYFASTRP plays at accelerated rate (for message scanning).

Input: none

Output: none.

Notes: Calls handler at stop_rp_play to allow interrupt.

The filter coefficient used by PLAYRP for the PWM may be set in CONFIG.A.

Playback starts at the address provided to init_xread_rp.

For both DAC and PWM output:

PLAYRPBOTH ()

PLAYFASTRPBOTH ()

Plays compressed speech through the DAC and PWM outputs simultaneously.

PLAYFASTRP plays at accelerated rate (for message scanning).

Input: none

Output: none.

Notes: Calls handler at stop_rp_play to allow interrupt.

The filter coefficient used by PLAYRP for the PWM may be set in CONFIG.A.

Playback starts at the address provided to init_xread_rp.

POSTRP ()

Deglitches, trims, and gain-adjusts recorded data.

Input: none

Output: none

Notes: Use immediately after RECORDRP. The external memory buffer allocated for post-processed data should be the same size as the buffer allocated for RECORDRP.

PostRP can strip silences from the the beginning of a recording and remove glitches IF thresh olding was used. If thresholding was not enabled, PostRP simply adjusts gain levels.

The pre-processed (raw) data starts at the address provided to init_xread_rp.

The post-processed (post) data starts at the address provided to init_xwrite_rp.

COMPRP (val destlevel)

Compresses data to 3 bits or 2 bits.

Input: **destlevel** Type of compression to perform:

3 = compress to 3 bits

2 = compress to 2 bits

Output: a error code: (0 =OK, 1=error)

Notes: This function will truncate the speech data if the destination buffer is too small to hold the entire message.

Approximate compression ratios are:

4 bits to 3 bits: .80

4 bits to 2 bits: .60

3 bits to 2 bits: .75

Silence blocks are not further compressed, so these ratios will not be exact (allow a little extra space for silences).

The un-compressed data starts at the address provided to init_xread_rp.

The compressed data starts at the address provided to init_xwrite_rp.

FINDENDRP ()

Finds end address of speech data.

Input: none

Output: none

Notes: This function can be useful for Flash memory erase procedures, etc. Before using FINDENDRP, the co-routine INIT_XREAD_RP should be called to initialize the start pointer. When FINDENDRP is complete, use EXIT_XREAD_RP to retrieve the pointer to the end of the message.

Parameters

The RecordRP **maxtime** parameter can be optionally be specified in units of half-seconds rather than seconds. This gives better resolution in storage-tight applications. Edit a local copy of the RecordRP macro (found in \LIB\MACROS\RP.INC) to set HALF_SECONDS to 1 and include this file.

The RecordRP **thresh_type** parameter should be set to RP_NO_THRESH if patgen is not running. When used with patgen, any valid value may be used. The PostRP function will remove any initial silence if RP_TRIM_THRESH or RP_FULL_THRESH is selected. Internal clicks and pops will be removed if RP_FULL_THRESH is selected.

Sample Code

The following is a short program segment that uses voice record and playback in a single message application. See \SAMPL364\RECORDER for the full sample program. This sample uses SRAM for storing the voice message, but non-volatile memories may also be used. (The program would have to be modified.) The directory folder \LIB\MEMORY\FLASH\ contains other folders with sample code using RP with specific Flash memories.

For clarity the sample code illustrates calling post-processing immediately after the real-time recording is finished. In a real application this processing (and any compression) would probably be deferred until idle time was available. This would allow smooth program flow without a processing delay after the real-time recording. If the message is needed for replay before it can be post-processed, it can simply be replayed as 4-bits from the original recording buffer. Careful design of memory management, buffer allocation, and program flow should allow applications to avoid any processing delays that would be objectionable to the user.

The sample illustrates using an assembly-time constant, USE_THRESH, to control whether thresholding is performed. This control could also be a run-time decision. Simultaneous pattern generation would be invoked with the same flow, but the Patgen parameter would be #BACKGROUND instead of #RP_THRESH. See the SAMPL364\SI6 sample program for an example.

```
play_mode:
    .
    .
    .
    mov    a,rp_start+0
    mov    b,rp_start+1
    mov    r0,rp_start+2
    call   init_xread_rp          ; init ext memory read

    PlayRPBoth                    ; output to DAC and PWM

    call   exit_xread_rp          ; finish up ext memory
    .
    .
    .
record_mode:
USE_THRESH equ    1              ;set to 0 for no thresholding
    .
```



```

.
.
mov     a,#RECRD_ST_0
mov     b,#RECRD_ST_1
mov     r0,#RECRD_ST_2
mov     dpl,#RECRD_END_0
mov     dph,#RECRD_END_1
mov     r7,#RECRD_END_2
call    init_xwrite_rp          ;init ext memory write: start and end
addresses

IF USE_THRESH
    PatGen #RP_THRESH          ; run threshold part of patgen
ENDIF
RecordRP #0, #RP_NO_THRESH     ;record speech data, no time limit, no thresh-
olding

    call exit_xwrite_rp        ;finish up ext memory
    mov     rp_start+0,#RECRD_ST_0
    mov     rp_start+1,#RECRD_ST_1
    mov     rp_start+2,#RECRD_ST_2

    mov     a,rp_start+0
    mov     b,rp_start+1
    mov     r0,rp_start+2
    call    init_xread_rp      ;init external memory read

    mov     a,#B4BIT_ST_0
    mov     b,#B4BIT_ST_1
    mov     r0,#B4BIT_ST_2
    mov     dpl,#B4BIT_END_0
    mov     dph,#B4BIT_END_1
    mov     r7,#B4BIT_END_2
    call    init_xwrite_rp      ;init external memory write

    PostRP          ;post-process data

    call    exit_xread_rp      ;finish up external memory
    call    exit_xwrite_rp     ;finish up external memory
    mov     rp_start+0,#B4BIT_ST_0
    mov     rp_start+1,#B4BIT_ST_1
    mov     rp_start+2,#B4BIT_ST_2
.
.
.

compress_mode:                ; illustrates 2-bit, could be 3-bit
.
.
.
mov     a,#B2BIT_ST_0          ; set 2-bit buffer
mov     b,#B2BIT_ST_1
mov     r0,#B2BIT_ST_2

```

```

mov    dpl,#B2BIT_END_0
mov    dph,#B2BIT_END_1
mov    r7,#B2BIT_END_2
call   init_xwrite_rp          ; init external memory write

mov    a,rp_start+0
mov    b,rp_start+1
mov    r0,rp_start+2
call   init_xread_rp          ; init external memory read

CompRP #2                      ; compress speech data to 2 bits

call   exit_xread_rp          ; finish up external memory
call   exit_xwrite_rp         ; finish up external memory

mov     rp_start+0,#B2BIT_ST_0 ; set 2-bit buffer start
mov     rp_start+1,#B2BIT_ST_1
mov     rp_start+2,#B2BIT_ST_2
.
.
.

;variables for sram_rp - all are allocated in technology memory

xram_tmpptr os    2
xram_getptr os    2
xram_getend os    2
xram_putptr os    2
xram_putend os    2

xram_address      os    2

; variables needed from user memory

UserRam

rp_start    fl    3      ;local start pointer for message

```

11. Using Touch-Tone Synthesis

This section describes the use of Sensory's DTMF (Dual Tone Multi-Frequency) synthesis, commonly known as "Touch Tones". The software adds acoustic tone dialing capability to RSC applications. Touch-tone synthesis software is available as object code for linking into a ROM image. The code supports both the DAC and PWM outputs on the RSC-364.

Initialization

Initialization includes setting up the hardware and the interrupt vectors. This is easily implemented by linking in the module \LIB\STARTUP\STARTUP. This module sets the processor speed to oscillator1 1 with 1 wait state, and initializes interrupt vectors. STARTUP jumps to the label INITIALIZE, which

should be at the start of the application program and needs to be PUBLIC'd for linker references.

Includes

Touch-tone synthesis should be called via a macro. LIBINCL the macros defined in \LIB\MACROS\TTONE.INC.

The TTONE macros use memory locations that are defined in \LIB\MEMORY\MEMORY.INC, so this file must be LIBINCL'd in the application program. See Using the RSC-364 Memory Map for detailed information.

Linking

Link the application with the object modules `\LIB\STARTUP\STARTUP`, `\LIB\MEMORY\MEMORY`, `\LIB\STARTUP\CONFIG` and the modules `TTONE` and `TTONE H` in the `\LIB\TTONE` directory.

TTone Handler

The TTONE code requires a user-supplied handler routine to know when keypresses or other events occur. When an event occurs, the handler sets the carry bit. The TTONE code uses this flag in two different ways: if the tone being synthesized is a dial tone, it will be terminated when the carry is set; if the tone being synthesized is not a dial tone, it will continue sounding until the carry is clear. The external logic for deciding about interrupting is under the programmer's control, but it should execute as quickly as possible to avoid missing samples. The sample callout handler shown below is from the file `TTONE_H.A`, which should be used as a template and linked with the application program.

```
; TTONE_H.A: Touch-tone software interrupt service routine

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      INCLUDES                                                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

LIBINCL "\lib\memory\memory.inc"
LIBINCL "\lib\startup\io.inc"                                ;use project-specific i/o

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      PUBLICS                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PUBLIC   TToneHandler

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      NOTES                                                         ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; The stop flag is used two different ways:
;
; When generating touch tones, the stop flag (carry) being set causes
;   the tone to continue (until the flag is no longer set).
;
; When generating dial tone, the stop flag (carry) being set causes
;   the tone to halt.
```

```

////////////////////////////////////
;      CONSTANTS                                     ;
////////////////////////////////////

////////////////////////////////////
;      START OF CODE                               ;
////////////////////////////////////

TTone_handler:  segment "CODE"

TToneHandler:

;<< insert the handler code here >>

;
; Example: set the flag on button A low
;
        clc
        tm      BTN_A_PORT, #BTN_A_BIT
        jnz     tth_exit
        stc                                ;Carry is the stop flag

tth_exit:
        ret

        END

```

Macros

In most applications, only one macro is needed for touch-tone synthesis, but there are three versions to choose from. Alternate versions of the macros output to the DAC, to the PWM, or to both. The macros are defined in the file TTONE.INC, which can be found in the \lib\MACROS directory.

The TTONE macro input parameters are defined as follows: *#* = 1-byte integer constant; *reg* = 1-byte register variable; *val* = either *#* or *reg*.

For DAC (only) output:

TTONEDAC *val* tone_number, *val* tone_duration, *val* sil_duration

For PWM (only) output:

TTONEPWM *val* tone_number, *val* tone_duration, *val* sil_duration

For both DAC and PWM:

TTONEBOTH *val* tone_number, *val* tone_duration, *val* sil_duration

Each of these macros synthesizes the specified touch-tone, where the passed parameters are:

Input: tone_number number of the tone to be synthesized. The codes are:

0 '0'

1 '1'

```

2 '2'
3 '3'
4 '4'
5 '5'
6 '6'
7 '7'
8 '8'
9 '9'
10 '*'
11 '#'
12 'A'
13 'B'
14 'C'
15 'D'
255   dial tone

```

tone_duration duration of tone in 10ms increments (minimum in North America is about 50ms). If dial tone is specified, then duration is measured in 1 second increments. A default value of 100ms (10 seconds for dial tone) is defined by the constant DEF_TONE_DUR in the TTONE.INC file.

sil_duration duration of silence after tone in 10ms increments (minimum in North America is about 50ms). A default value of 90ms is defined by the constant DEF_SIL_DUR in the TTONE.INC file.

Output: none

Sample Code

The following page shows a short complete program using touch-tone synthesis. The source files are in the \SAMPL364\TTONES directory.

```

; TTONES.A - Touch-tone demo program

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   INCLUDES                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

LIBINCL "\lib\memory\memory.inc"
LIBINCL "\lib\memory\usermem.inc"
LIBINCL "\lib\startup\io.inc"           ;use project-specific i/o

LIBINCL "\lib\macros\ttone.inc"
LIBINCL "\lib\macros\talk.inc"

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   PUBLICS                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

PUBLIC      Initialize

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      EXTERNS                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

      EXTERN      VPnumbersd

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      VARIABLES                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

      UserRam      ;static variables

digit_ptr    fl      2
digit_ctr    fl      1
use_areacode      fl      1
tmp_ptr      fl      2
tmp_ctr      fl      1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      CONSTANTS                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TONE_DUR      set      DEF_TONE_DUR      ;(this value from include file)
SIL_DUR        set      DEF_SIL_DUR      ;(this value from include file)
DT_DUR         set      2                ;two seconds dial tone

VR             equ      9330/64          ; VoiceRate - default sample rate (Hz/64)
max=9330

; speech message constants in VPnumbersd

msg_numbers    equ      0                ; 0-9, A-F
msg_beep       equ      16
msg_beepbeep   equ      17
msg_sil30      equ      18
msg_sil60      equ      19
msg_sil120     equ      20
msg_sil250     equ      21
msg_sil500     equ      22
msg_sil1000    equ      23
msg_sil2000    equ      24
msg_sil4000    equ      25

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      Copyright notice                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Copyright:    segment "CODE"
              db      " Copyright (c) 1996,97,98,99      "
              db      " Sensory, Inc.,"

```

```

        db      " All Rights Reserved "

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      START OF CODE                                           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TTONE_TEST: segment "CODE"

initialize:
        CallTalkBoth    #msg_beep,#high VPnumbersd,#VR
main:
        tm      BTN_A_PORT,#BTN_A_BIT                ;button A:  local
        jz      local
        tm      BTN_B_PORT,#BTN_B_BIT                ;button B:  long distance
        jz      long_distance
        jmp     main
long_distance:
        mov     use_areacode,#1
        jmp     say_number
local:
        mov     use_areacode,#0
say_number:
        mov     digit_ptr+0,#LOW PhoneNumbers ;point to phone number
        mov     digit_ptr+1,#HIGH PhoneNumbers
        mov     digit_ctr,#11                    ;dial 11 digits
        cp      use_areacode,#0                    ;if local call
        jnz     say_number1
        add     digit_ptr+0,#4                    ; skip 1+area code
        adc     digit_ptr+1,#0
        mov     digit_ctr,#7                    ; dial 7 digits
say_number1:
        mov     tmp_ptr+0,digit_ptr+0
        mov     tmp_ptr+1,digit_ptr+1
        mov     tmp_ctr,digit_ctr
say_number2:
                                ;speak phone number with
        cp      tmp_ctr,#4                    ;pause at selected points
        jz      pause
        cp      tmp_ctr,#7
        jz      pause
        cp      tmp_ctr,#10
        jz      pause
        jmp     say_number3
pause:
        CallTalkBoth    #msg_sil250,#high VPnumbersd,#VR
say_number3:
        movc    a,@tmp_ptr
        CallTalkBoth    a,#high VPnumbersd,#VR
        inc     tmp_ptr+0
        adc     tmp_ptr+1,#0
        dec     tmp_ctr
        jnz     say_number2
dial_number:
        tm      BTN_A_PORT,#BTN_A_BIT                ;wait for button press

```

```

        jz     dial_tone
        tm     BTN_B_PORT,#BTN_B_BIT
        jz     dial_tone
        jmp    dial_number
dial_tone:
        tm     BTN_A_PORT,#BTN_A_BIT           ;wait for button release
        jz     dial_tone
        tm     BTN_B_PORT,#BTN_B_BIT
        jz     dial_tone
        TToneBoth #255,#DT_DUR,#SIL_DUR           ;dial tone
dial_number1:
        movc   a,@digit_ptr
        TToneBoth a,#TONE_DUR,#SIL_DUR           ;dial digit sequence
        inc    digit_ptr+0
        adc    digit_ptr+1,#0
        dec    digit_ctr
        jnz    dial_number1

        CallTalkBoth #msg_sil1000,#high VPnumbersd,#VR
        jmp    Initialize

;-----

PhoneNumbers:
        db     1, 4, 0, 8, 7, 4, 4, 9, 0, 0, 0           ; Sensory's Main Number

;-----

        END

```

Specifications

The tables below show the tolerance and levels for the DTMF output.

Table 1 - Frequency tolerance of the output tones for DTMF signaling - $f_{xtal} = 14.318180 \text{ MHz}$

ROW/COLUMN	STANDARD FREQUENCY (Hz)	TONE OUTPUT FREQUENCY (Hz)	FREQUENCY DEVIATION (%)
ROW 1	697	696.99	.002%
ROW 2	770	769.96	.005%
ROW 3	852	851.89	.013%
ROW 3	941	940.93	.007%
COL 1	1209	1208.92	.007%
COL 2	1336	1335.94	.004%
COL 3	1477	1476.90	.007%
COL 4	1633	1632.94	.003%

Table 2 - Electrical Characteristics

Parameter	Conditions	Min	Typ	Max	Units
DTMF output levels for HIGH group	$V_{DD} = 5V$, $R_L = 10M\Omega$		880		mV
DTMF output levels for LOW group	$V_{DD} = 5V$, $R_L = 10M\Omega$		660		mV
High Group Pre-Emphasis	$V_{DD} = 5V$, $R_L = 10M\Omega$		2.5		dB
Frequency Deviation	$f_{xtal} = 14.318180$ MHz	-0.013		0.0	%
Total Harmonic Distortion	$V_{DD} = 5V$, $R_L = 10M\Omega$			0.35	%
Output Impedance			22		$k\Omega$
Tone Duration		100			mS
Interdigital Interval		90			mS

12. ROM Memory Requirements of Technology Code

This section outlines the Read Only Memory (ROM) required to use the speech and audio technologies available with the RSC-364 and Sensory Speech 6 Technology. Note that other RSC chips have different memory maps. These technologies are implemented by using one or more of the technology modules provided in the RSC-364 Development Kit software library. The following table lists each technology module, a brief description of its function, its code size (in kilobytes), and which technologies it is used by. For example, an application using Speaker-Independent Recognition and Speaker-Dependent Recognition will only require a total of 15.5 kilobytes of memory (Pattern Generation and Fixed-Point Math are only included once). The optional technology modules listed in the bottom table can be used with the identified Sensory technologies. **For applications using multiple technologies, it is only necessary to include each technology module once. Note that some features, such as record and playback, may also require off-chip RAM or flash.**

Technology Module	Description	Approx Size (kB)	SI	SD	SV	DRT	WS	RP	Talk	Music	TTone
Patgen	Pattern Generation	8.0	√	√	√	√	√				
SI	Speaker-Independent	1.7	√			√					
SD	Speaker-Dependent	2.5		√	√	√	√				
SV	Speaker Verification	0.3			√						
DRT	Dual Recognition Technology	0.9				√					
WS	Wordspot	2.6					√				
RP	Record and Playback	7.4 ¹						√ ¹			
Talk	Speech Synthesis	3.4							√		
Music	Music Synthesis ⁵	1.7								√ ⁵	
TTone	Touch-tone Synthesis	0.6									√
Math(1)	Integer Math	0.8	√	√	√	√	√	√	√	√	
Math(2)	Floating-point Math	2.1	√		√	√		√			
Total Sizes			12.6	11.3	13.7	20.0	13.9	10.3	4.2	2.5	0.6

Optional Technologies											
RP	Simultaneous Record and Pattern Generation	8.0	(√ ⁶)	(√ ⁶)	(√ ⁶)	(√ ⁶)		(√ ⁶)			
CL ²	Continuous Listening	0.3	(√)								
SEEP ³	Sample Serial EEPROM	1.1		(√)	(√)						
Flash ⁴	Sample Flash RAM	0.8		(√)	(√)	(√)		(√)			
RS232	RS232 drivers	0.3									
DRT Digits Weights	DRT Digits weightset	4.0				(√)					
SI Digits Weights	Speaker Independent weightset	4.3	(√)			(√)					

1. Size of Record and Playback code can be reduced if 3-bit and/or 2-bit compression is not used. Add Patgen size if thesholding is used.
2. Continuous Listening specified for Speaker-Independent Recognition only.
3. Serial EEPROM is optional for Speaker-Dependent Recognition, fast digits DRT, or Speaker Verification technologies.
4. Flash RAM code optional for Record and Playback, Speaker-Dependent Recognition, fast digits DRT, Speaker Verification and Speaker Adaptive technologies.
5. Music Synthesis also requires music notes and the data, which can be as much as 50kB.
6. Simultaneous Pattern Generation and recording is optional on the RSC-300/364.

CHAPTER 3 - SOFTWARE DEVELOPMENT

This chapter provides the developer with a variety of information to facilitate software development. This information is primarily reference material, not tutorial; it contains considerable detail. Be sure to read the Overview Chapter, especially Getting Started and the sections dealing with hardware configuration sections before digging into this chapter.

The chapter includes an explanation of important considerations for inserting silence durations with speech synthesis. There is also information about Speaker Independent recognition accuracy and noise robustness. Another section discusses software I/O mapping and other software configuration.

This chapter also describes the user memory map of the Register Space (on-chip RAM) for the RSC-364, which is essential information for programming the chip. This chapter describes a utility program, Debug Speech, which can be used to debug speech recognition code. This utility uses speech synthesis to tell the programmer the value of important parameters used in speech recognition.

Many applications require the use of external memory to store speaker dependent words, speaker verification words, and voice record and playback messages. Sensory has co-routine interfaces for SRAM, Flash memory, and serial EEPROM that are described in this chapter. There is also a description of how to build systems that need to access memory larger than the 64K bytes that can be addressed by the 16-bit address bus. Another section describes the Floating-Point Library Routines. This chapter also includes reference tables describing stack usage by technology modules and the include/link file and co-routine requirements for each technology.

1. Software and Linguistics Interactions

In order to create successful products, applications programmers must become knowledgeable about certain linguistics issues during the development of code using speech synthesis and recognition software. This chapter gives some guidelines to assist that understanding.

Synthesis

Inserting Silences in the Synthesized Speech

Generally, the linguist who generates the vocabulary will handle the silences within sentences. The programmer should generally insert the silences between sentences. For example, consider the two synthesized sentences, "Password rejected" and "Say your password". These linguistic building blocks might be used separately in some places in the application, but the second might follow the first in one place in the application flow. To create a natural pace to the speech, the programmer might put a silence of about 500 milliseconds between these two sentences. Directions or questions requiring some thought or direct-action should be broken into two parts with a suitable silence, for example "Name not found" <silence> "Do you wish to enter this name?" Avoid adding silences that can be confusing. In this example, the user might begin speaking after the first message: "What did you say?" <silence> "repeat". Silences can be produced by calling sentences that are silence for fixed periods of time. Each vocabulary typically contains addresses for generating 20, 40, 80, 160, 320, 500, 1000, 2000 and 4000 milliseconds of silence.

Inserting a Delay After the Question and Before Calling Pattern Generation

Except for continuous listening and wordspot applications, speech recognition is typically preceded by a speech prompt. Before recognition can begin there must be time for the room echo of the prompt to decay away. The library code provided in this release automatically monitors the echo level and detects when the echoes have died. If sufficient decay is not achieved within 150 milliseconds, a "spoke too soon" error is returned. It is important that the last word of the prompt stop abruptly. If the last word before recognition has a long period of low power at the end, the user may not hear this and will begin speaking during the echo decay period, causing a "spoke too soon" error. The NO_ERROR option provides an unobtrusive way of dealing with "spoke too soon" errors.

Recognition (see also Using *Speaker Independent Speech Recognition*)

Inserting Measurements of the Silence Threshold Before Pattern Generation

The software in previous releases required measuring the silence threshold (background noise level) before each call to the pattern generation algorithm. With the speech framing feature in this release, that requirement is no longer necessary.

Use of Priors

The neural network returns speaker-independent (SI) results that include a probability distribution across the various classes (answers) in the recognition set. The PRIOR macro can be used to bias the final decision. If the recognition question has a preferred answer ("Are you sure?"), set the prior for that answer to twice that of other priors. This will increase the chance of the correct answer being recognized. If all answers to the question are equally likely ("Delete?"), do not use the PRIOR macro. The PRIOR macro may also be used to remove an answer from consideration (set its probability to zero).

Noise Robustness; NOTA; Yeah-Yep-Yes and No-Nope

The code in this release provides improved noise robustness and increased recognition accuracy. This is mostly achieved in the pattern generation phase, but also includes re-tuned recognition algorithms and new SI weight sets. New weight sets are located in the \DATA\WEIGHTS\364 directory as linkable object (.O) files.

Improved noise robustness means that recognition will perform significantly better in environments with a steady noise background (e.g. a loud fan or motor) and should also show some improvement in environments with fluctuating noise. This is particularly true of speaker dependent recognition (SD) and speaker verification (SV).

Speaker independent weight sets include an extra recognition class, REJECT (or NOTA -- none of the above), which helps recognition in two ways. First, if the user says an unexpected word, the recognizer will usually return the REJECT class. For example, if the application prompts with "Continue?" and is expecting YES/NO, but the user says, "continue", the application can deal with this result in an appropriate way (e.g., by saying, "Please say yes or no"). Second, if an abrupt or irregular noise occurs during recognition (such as a sneeze or a door slamming), the recognizer can return the REJECT class.

As in previous releases, the NOYES weight set includes the words "yeah" and "yep" (in addition to "yes") and "nope" (in addition to "no").

2. Using the RSC-364 Memory Map

This section is a description of the User Memory Map of the Register Space (on-chip RAM) for the RSC-364 Software Development Kit.

Overview of 364 Memory Map

This sub-section gives a general overview of the organization of Register Space on the RSC-364. Following sections provide details. Refer to \LIB\MEMORY\ for the memory definition files.

The RSC-364 register space is divided into ten different sections as shown in Table 1- Register Space usage in the RSC-364.

The table indicates that some of these sections are for the exclusive use of Sensory technology code, some of the sections can be used by both technology and application code, and some sections can be reserved exclusively for application use. Most memory is heavily overlaid (used for multiple purposes). Memory that must remain exclusively used for a single purpose is called "static" memory. The table, along with the descriptions in the following sections, should help the developer decide how to make best use of the memory available.

Addresses	Register Description	DK Use	Notes
00h-0Fh	global registers	Both	temporary app use only
10h-20h	XRAM registers	Both	memory co-routine variables
21h-7Fh	Sensory registers	Tech	no app use
80h-85h	System Statics	Tech	app reference via macros
86h-8Bb	Debugger Reserved	None	future debugger use
8Ch-8Fh	Bank Code ROM Reserved	None	optional Code bankswitch
90h-9Fh	Stack Reserved	App	can use for general app
A0h-BFh	User Statics	App	no tech use
C0h-DFh	Banked RAM	Both	temporary use only
E0h-FFh	SFR registers	Both	IO ports, etc.

Table 1· Register Space usage in the RSC-364

User Memory Map Description

The global registers area begins at address 00hex and consists of sixteen byte-wide registers. Sensory technology code uses these registers extensively, but they may be used at other times for temporary storage by the application. Assume that any technology call changes the values of all these locations.

Address	00h	01h	02h	03h	04h	05h	06h	07h
Register	r0	r1	r2	r3	r4	r5	R6	r7

Address	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh
Register	dpl	dph	a	b	trash	trashx	temp0	temp1

These registers are commonly used to pass and return values to technology code through macros. The dpl/dph pair may be addressed as dptr, the temp0/temp1 pair may be addressed as tempptr. The dptr, a, and b register names are aliases for special registers in the Intel 8051; note, however, that any register (pair) in the RSC-364 could perform the same operations.

The trash and temp locations are for very short-term storage only. The scall and sret macros as well as most library functions destroy the contents of these registers.

The XRAM registers in address range 10h-20h are allocated for use as pointers for accessing external read/write memory. Speaker Dependent/Verification, Dual Recognition, and Voice Record and Playback (RP) technologies use these registers to read and write patterns, templates, and voice recordings. The specific usage is different for each application and may vary for different memory parts. All memory drivers should conform to the usage of these 17 bytes. These variables may be used for other purposes when technology code is not active. Applications not using RP technology normally may use locations 1Bh-1Fh as statics (although some SD device drivers may make use of these locations, for example see the serial EEPROM driver). Applications not using SD or RP technologies may use all of the XRAM registers as statics.

Address Range	10h-14h	15h-1Ah	1Bh-20h
RP use	low-level driver	upper-level pointers	upper-level pointers
SD use	low-level driver	upper-level pointers	device-specific use (20h not free)

The Sensory Registers in the range 021h-07Fh are reserved for the exclusive use of technology code. Any application use of this area will cause certain failure of some technologies.

The System Statics registers in address range 80h-85h are normally written only by technology code or by macros. These registers are primarily used for communication between different technologies regarding information such as the "owner" of the timer1 interrupt, whether to use PWM or DAC for speech, etc. Applications should normally not access these registers directly.

The six registers at 86h-8Bh are the Debugger Reserved registers. Future Development Kit releases may provide a debugger that would use these registers, but this release does not support such a debugger. These registers are reserved as statics for such possible future use. For this release, these registers may be reclaimed by applications.

The four registers at 8Ch-8Fh are the Bank ROM Reserved registers. Previous releases of the RSC-164 developer kit supported ROM Code Bank switching, but this release does not support ROM Code Bank switching. These registers are reserved as statics for such possible future use. Applications for this release may reclaim these registers.

An optional software stack is implemented at addresses 90h-9Fh. These Stack Reserved registers are used by applications that employ the scall (and sret) macros or use the push/pop RSC-364 instructions. The RSC-164 required extensive use of the scall and sret macros due to a shallow stack, but the RSC-364 has a deeper stack, so these macros are not required for most applications. The RSC-364 technology code makes no use of either the scall/sret macros or the push/pop instructions, so use of the software stack is the choice of the developer. Information is provided here to allow using the software stack, but most applications should reclaim this memory for expanding the User Statics area. Refer to \LIB\MEMORY\USERMEM.INC for more information and to enable use of the software stack.

Address Range	90h	91h	92h-9Fh
Stack Usage	data_stack_ptr	addr_stack_ptr	Data and adress stack

The software stack uses an 8-bit data_stack_pointer, an 8-bit address_stack_pointer, and a combined address and data stack. The address stack begins at address 92hex and grows to higher addresses (92h, 93h, 94h...). The scall macro pushes two bytes of return address onto the stack and increases addr_stack_ptr to point to the next free location. The sret macro pops two bytes of return address off the stack and decreases the addr_stack_ptr to point to the next used location. The sret macro expands into 18 bytes of code, so there is typically a single instance of it at location GLOBAL_SRET. To execute an sret, the application jumps to GLOBAL_SRET. (See also The GLOBAL_SRET Code Segment below.)

The data stack begins at address 9Fhex and grows down to lower addresses (9F, 9E, 9D, 9C...). If a developer wants to use the software stack, but needs fewer than 14 bytes, this upper address, DATA_STACK, can be reduced by editing \LIB\MEMORY\MEMUSER. The bytes freed up may be used as static storage. The push instruction stores a byte at the address pointed to by the stack_pointer and decrements the pointer. The pop instruction increments the pointer and reads a byte at the address pointed to by the stack_pointer. Note that address and data corruption will occur if the data and address spaces in the stack overlap. The GLOBAL_SRET routine (see below) provides optional stack collision detection.

WARNING

The software stack is not enabled by default. You must edit \LIB\MEMORY\USERMEM.INC, and you must re-build STARTUP.A to use the stack. If you use scall or sret without doing so, your program will crash.

Before an application can use the software stack, the pointers must be initialized. \LIB\STARTUP\STARTUP.A contains conditional code that performs this operation. After initialization, a byte at location addr is pushed on the data stack or popped from the data stack by invoking the instructions shown (the instructions are similar to indirect mov and cannot take a constant argument, #byte). The instructions automatically adjust the value of data_stack_ptr. Because the stack can be located anywhere in Register space, the instructions require a pointer argument.

```
push @data_stack_ptr, addr      ; push byte at addr onto software stack
pop  addr, @data_stack_ptr      ; pop byte into addr from software stack
```

The User Statics registers are addressed at locations A0h-BFh. These registers are guaranteed to be untouched by any Sensory technology code. These 32 bytes may be used for application variables. In addition, virtually all RSC-364 applications will be able to obtain more storage by expanding the application variable registers downward through the Stack Reserved registers, the Bank ROM Reserved registers, and the Debugger Reserved registers. This allows the application memory configurations shown.

Lower Boundary of Application Variables	User Start Address	Number of User Bytes
User Statics	A0h	32
Stack Reserved (default configuration)	90h	48
Bank ROM Reserved	8Ch	52
Debugger Reserved	86h	58

The upper boundary of the application memory is always BFh. Under no circumstances may the lower boundary of User Statics be set below 86h.

Most applications will define the application-static ram area by using the macro, UserRam. This macro, defined in \LIB\MEMORY\MEMUSER, automatically sizes the user statics area based on information regarding the presence and size of the software stack. The following example illustrates a typical usage. By default, the user memory is 48 bytes. The stack can be enabled, disabled, or changed in size (see Stack Reserved discussion above).

```

;--- sample application usage#1: User variables up to 48 bytes
; define user static ram locations (starts at 90h by default or after stack)
      UserRam
variable1      fl      1      ;90h
variable2      fl      1      ;91h
pointer1       fl      2      ;92h
variable3      fl      1      ;94h, etc.
. . .

```

The application may also explicitly define specific locations within the User Statics area by use of the "fl" directive ("fixed label") relative to labels defined in \LIB\MEMORY\MEMORY.INC. The code fragment below illustrates this.

```

;--- sample application usage#2: User variables need up to 58 bytes
;      give up reserved areas for Debugger, Bank ROM, and software stack

custom_area fixlabel REG at dbg_stat_start      ;start at 86h
custom_area fixlabel REG end at dev_area_end    ;stop at BFh

;static variables

variable1      fl      1      ; 86h
variable2      fl      1      ; 87h
pointer1       fl      2      ; 88-89h
inPtr          fl      1      ; 8Ah
outPtr         fl      1      ; 8Bh
bigBuffer      fl      50     ; 8Ch-BEh
bufferEnd      fl      0      ; BFh (see Note below)
lastByte       fl      1      ; BFh
. . .

```

NOTE: The "fl size" directive defines the label relative to the previous "fixlabel" or "fl size" directive and steps by the size specified. No actual memory allocation is performed. In the example above, the "fl 0" following the bufferEnd label assigns the value 0BFh to bufferEnd, but does not change the reference for subsequent "fl" directives. Therefore the lastByte symbol also has the value 0BFh. No error message is issued in this case. This side effect is a consequence of the "fl" directive, the intention of which is to avoid excessive error messages when building applications that use memory overlays. Applications for the RSC-364 should normally build with no error messages.

The Banked RAM is used by technology code, and may also be used temporarily for buffer areas, for example with Flash memories. Refer to Chapter 8, the RSC-364 Data Book for details on addressing and mapping of the banked register space.

The register memory (on-chip RAM) is grouped into 32 byte banks as defined in the following table:

Ram address	Bank #Used for
00 - 1F0	global registers
20 - 3F1	technology functions
40 - 5F2	technology functions
60 - 7F3	technology functions

80 - 9F	4	system statics, debugger, bank switching, user registers
A0 - BF	5	user registers
C0 - DF	6	bank switched ram (maps other RAM)
E0 - FF	7	special function registers
100 - 11F	8	technology functions
120 - 13F	9	technology functions
140 - 15F	A	technology functions
160 - 17F	B	technology functions
180 - 19F	C	technology functions / buffer
1A0 - 1BF	D	technology functions / buffer
1C0 - 1DF	E	technology functions / buffer
1E0 - 1FF	F	technology functions / buffer

The area from 100h - 1FFh can be read/written only by using the C0h - DFh bank switched ram area under the control of the RAM BANK Select register (SFR address 0FCh) bits 0 - 4. Refer to the RSC-364 Data Book for a detailed description of the RAM Bank Select Register. The area from 100h - 1FFh is used extensively for technology code and should be considered as temporary storage only if these functions are called. For convenience in accessing and re-defining, the assembler defines labels "bank00" - "bank1f". These symbols map to the addresses C0h - DFh.

Allocating Memory

The memory definitions described above are mostly found in \LIB\MEMORY\MEMORY.INC. This file should be LIBINCLd in every application that uses scalls, technology macros, or that needs to access the global registers for passing arguments or for temporary scratch use. The software stack and user static areas are defined in \LIB\MEMORY\USERMEM.INC, which must also be included in every application that uses local variables. A local copy of USERMEM.INC can be edited and included to change the default user memory allocation and re-claim additional static storage as described above. NOTE: To enable the software stack code, you must rebuild STARTUP.A and make sure it INCLUDEs your edited version of USERMEM.INC. Other memory definitions may be made by include files required by various technologies

The software stack is disabled by default. It is enabled by setting the ENABLE_SW_STACK conditional assembly flag in (usually a local copy of) \LIB\MEMORY\USERMEM.INC.

The RSC-364 assembler does not perform physical allocation of RAM memory. The labels are simply defined and equated to the memory locations. A minor consequence of this is that the .MAP file does not indicate the addresses of RAM variables. This RAM address information is contained in the .SYM file and in the .LST files.

The GLOBAL_SRET Code Segment

The (optional) code segment GLOBAL_SRET is located in \LIB\STARTUP\STARTUP.A. This segment simply consists of an SRET instruction that is accessible to all code modules via a JMP instruction. This feature is designed to save code space by using the instruction JMP GLOBAL_SRET anywhere that an SRET would normally be used, saving 15 bytes for each instance. This optional code is only useful in applications using the software stack. NOTE: Most RSC-364 applications will not need to use the software stack. See System Configuration and I/O Mapping for more information. GLOBAL_SRET can

also be configured to check on software stack overflow due to excess scalls (but not pushes). This code is automatically enabled when STARTUP.A is rebuilt if the user has enabled software stack in \LIB\MEMORY\USERMEM.INC.

3. Using On-Chip Pattern Storage

The RSC-364 contains a memory block called "on-chip" or internal SRAM. This memory differs from the usual Register Space SRAM (which is also on-chip and internal) in the method of access and in the use. On-chip SRAM is used intensively by the Version 6 technology software, and much of it is dedicated for technology features. A portion of the internal SRAM memory is available for certain application uses, however. This section describes that usage.

In Release 6, Sensory is supporting three application types for on-chip memory usage: small-vocabulary SD or SV, Wordspot, and fast sequential pattern queuing. Support for additional applications may be offered in future releases.

Small-Vocabulary SD/SV

"Small-vocabulary" SD/SV applications are limited to 6 templates maximum. The RSC-364 can store these six templates in the on-chip SRAM, avoiding the need for any external memory. An ideal small-vocabulary application is SV for individual access, which requires no more than four templates. To prevent loss of template data, the chip **MUST BE KEPT CONTINUOUSLY POWERED** (it can be placed in "sleep" mode, where power consumption is negligible, but power cannot be removed). See the SPKRVER sample program, which can be configured to store templates on-chip. Also refer to Chapter 2, Using Speaker Dependent Recognition, for details about using the special on-chip memory handler. On-chip template storage requires no special application coding; simply code as normal and link the special modules, LIB364\SD_SV\ONCHIP.O and \LIB364\SD_SV\AUXTMPLT.O.

An application using on-chip memory for storing templates should provide a mechanism to determine if the templates are valid - this could be used to tell if the chip had lost power. One possibility, illustrated in the SPKRVER sample, uses an ID string and a count of the number of templates trained. A reset or "sleep" mode would not destroy the templates or the ID string, but a power loss would.

Note, however, the ID string for onchip should NOT be stored in the on-chip template area: this would waste one of the six available template slots, and also there is no supported mechanism to write individual bytes to this memory. It is better to allocate a few bytes from UserRam to store the ID string and number of templates (password length), since the on-chip SRAM and the UserRam would both be either valid or invalid.

Wordspot

Wordspot is a specialized form of SD recognition that uses parts of the on-chip template storage area during operation. When Wordspot is used, only the first four template storage locations are available on-chip. The Wordspot templates can be stored in some of these locations. The default configuration of the wordspot sample program uses on-chip storage for its templates.

Fast Sequential Pattern Queuing

Fast sequential pattern queuing is a technique used for recognizing a rapidly spoken sequence of words, for example the digits of a telephone number. The additional on-chip memory provides space for storing

patterns after pattern generation but before recognition. The patterns are queued during speaking and recognized after speaking finishes. This speeds up the speaking by avoiding the recognition time. Each new word should be prompted by a brief "click" to indicate that pattern generation for the previous word is done. This technique is used in the 364 sample of Fast Digits. Note, however, that the general speedup of DRT recognition in this release makes recognizing-on-the-fly competitive with queuing.

Initialization

Template sizes are managed automatically in this release, so no initialization is needed. Up to ten Size10 patterns may be queued, or six Size16 patterns. Normally SD, SV, and Wordspot patterns are Size16. SI patterns may be Size10 or size 16.

Includes

The pattern queuing routines should be invoked with macros defined in the include file, `\LIB364\MACROS\ONCHIP.INC`.

Linking

Link the low-level pattern management object code, `\LIB364\SD_SV\AUXTMPLT.O` with the rest of your code.

Macros

Two macros provide the services to transfer pre-recognized patterns to and from the on-chip storage area. The on-chip macro input parameters are defined as follows:

= 1-byte integer constant; reg = 1-byte register variable; val = either # or reg,

QUEUEPAT (val **index**, val **tmpltX**)

Transfers the pattern from a specific area in technology RAM to on-chip memory.

Input: **index** pattern position in sequence (0-MAX_PATTERNS-1)
 tmpltX Identifier of one of three areas where the pattern can be
 located in the RSC-264T memory. (Normally TMPLT_UNKNOWN)
Output: carry bit 0 = ok; 1 = error

Notes: TemplateSizeXX (XX=10 or 16) must have been called previously.

The parameter index is limited to:

TemplateSize10 : 0 - 9

TemplateSize16 : 0 - 5

The parameter **tmpltX** can have the following system-defined values:

#TmPLT_UNKNOWN - usually the most recent patgen pattern

#TMPLT_KNOWN - typically an older pattern from external memory

#TMPLT_MODIFIED - typically an averaged pattern

GETQUEUEPAT (val **index**, val **tmpltX**)

Transfers the pattern from on-chip memory to a specific area in technology RAM.

Input: **index** pattern position in sequence (0-MAX_PATTERNS-1)
 tmpltX Identifier of one of three areas where the pattern can be
 located in the RSC-264T memory. (Normally TMPLT_UNKNOWN)
Output: carry bit 0 = ok; 1 = error

Notes: TemplateSizeXX (XX=10 or 16) must have been called previously.

The parameter index is limited to:

TemplateSize10 : 0 - 9

TemplateSize16 : 0 - 5

The parameter **tmpltX** can have the following system-defined values:

#TmPLT_UNKNOWN - usually the most recent patgen pattern

#TMPLT_KNOWN - typically an older pattern from external memory

#TMPLT_MODIFIED - typically an averaged pattern

NOTE: The Pattern Queue routines CANNOT be used to save and restore previously trained templates during SD/SV or Wordspot recognition. The RecogSD, GetTemplate, and PutTemplate macros manage that template handling through the usual co-routine memory handlers described elsewhere in this chapter. The Queue routines are only for patterns to be recognized (i.e., "unknown" patterns). Since the Pattern Queue routines use the same on-chip memory as the Small-vocabulary SD/SV on-chip storage, these two features may not be combined in a single application. Pattern Queuing is also not compatible with Wordspot.

4. System Configuration and I/O Mapping

This release includes four modules that provide global information about the system configuration (amplifier, Data Space bankswitching, PWM filtering, etc.) and I/O usage. This section describes how to use these modules, and their interaction with the standard STARTUP module. For hardware configuration, refer to Chapter 4, Development Kit Hardware.

I/O Mapping (IO.INC)

The included file \LIB\STARTUP\IO.INC is used to define the functions of various I/O bits. Some of the standard definitions in this module are: buttons, color LEDs, bankswitching, external memory control (XML/XMH), serial EEPROM, and flash RAM. The default definitions refer to I/O usage on the RSC-364 Development Kit Motherboard.

If you need an IO configuration different from the default, Sensory recommends copying the IO.INC module into your working directory, and using that copy for edits (be sure that your application INCLUDEs that version). Please refer to the programmer note at the top of the file to determine which modules should be re-assembled following a change to IO.INC (these files include STARTUP and CONFIG, memory management modules and interrupt handlers). In general, any I/O function may be re-mapped to any I/O bit.

NOTE: Although bankswitching is defined in this module, ROM bankswitching is NOT supported in this release.

System Configuration (CONFIG.A, CONFIG.INC, CONST364.O)

Certain system constants are declared in module \LIB364\CONST364.O. This file should be LIBINCLd in every application using the RSC-364.

System configuration is defined in the module \LIB\STARTUP\CONFIG.A. The related include file (CONFIG.INC) simply creates external references to values defined in CONFIG.A, and should not be modified. CONFIG.A is divided into multiple sections, each of which deals with a specific item.

Global Amplifier Configuration

The RSC-364 contains an integrated preamplifier that always runs at maximum gain. Automatic gain control (AGC) is handled through software. Applications will normally use the integrated amplifier by enabling the INTERNAL symbol (this is the default operation).

It is possible to use the RSC-364 with an external amplifier by disabling the INTERNAL symbol. This adds additional components and cost, so an external amplifier should not be used except in special circumstances. When an external amplifier is used, it should also be fixed gain. Earlier AGC designs for external amplifiers are obsolete. The code dealing with AGC has been removed.

NOTE: External amplifier design is a critical element in a voice recognition system. Consult with Sensory before attempted such a design.

Global Bankswitching Configuration

The RSC-364 Developer Kit with Sensory Speech 6 Technology does not support ROM Code bankswitching. The definitions in this section are maintained for possible future use.

Debug Speech and Debug RS232

Sensory provides standard speech or serial debugging output to assist developers in evaluating system performance. A byte in ROM memory at location DebugEnable controls the amount of debugging information output. The value of that byte, DEBUG_ENABLE, is defined in this section. Applications that link CONFIG.O will output debugging information accordingly. This value may also be modified directly in the final ROM binary image. See the section Using Debug Speech in this chapter for more information. By default, no debug output is enabled.

Filtering for Talk and RP playback

In most hardware systems, the PWM output is connected directly to a speaker, requiring low-pass digital filtering before outputting the signal. In this case, the adjustable filter coefficient should normally be set to the default value of 2. A larger value decreases the relative amount of high frequencies in the output, and a smaller value increases the relative high frequencies. Digital filtering for speech synthesis (TALK) and RP playback are controlled separately.

Digital filtering is not done on signals passed through the DAC output.

Speaker Dependent Performance

In this release, the sd_performance parameter is selectable at run-time, so a constant definition is not needed. See Using Speaker Dependent Speech Recognition for more information.

Continuous Listening Performance

In this release, the cl_performance parameter is selectable at run-time, so a constant definition is not needed. See Using Continuous Listening for more information.

Flash Buffers

Sensory has defined locations for read and write buffers to be used in conjunction with external Flash RAM. Drivers for other memory devices that require buffers should use these values.

System Initialization (STARTUP.A)

The module \LIB\STARTUP\STARTUP.A provides general startup initialization that can be used for most applications as-is or with minimal editing. Sensory recommends copying the STARTUP.A module into your working directory, and using that copy for edits (be sure that your linker .CMD file reflects this change).

This module sets up standard system values such as processor clock speed, wait states, wake-up registers, I/O configuration, and interrupt vectors. If the software stack is being used, the module adds code for the global_sret routine and optional stack checking, and it initializes the stack pointers. It includes a one-second-delay loop to allow the analog electronics plenty of time to power-up. Unless the application does immediate recognition, this delay loop can be reduced or eliminated.

I/O configuration is done through a series of SET pre-processor instructions. The final values are then moved into the I/O registers, requiring a minimal amount of actual code. The amplifier setup is provided for the case of an external amplifier, but most applications will use the integrated amplifier and will not enable this code.

Common interrupt vectors are already set up in the code, with compiler flags to enable or disable each one. Simply set the flag for each technology that is linked into your program. NOTE: The default configuration of STARTUP.A requires the linker to resolve interrupt vector labels defined in pattern generation and synthesis modules. If you wish to build an application using STARTUP.A without using speech or pattern generation, you must disable these flags in STARTUP.A or else you will have to link in the speech and pattern generation files to resolve the symbols.

The default version of STARTUP.A in \LIB\STARTUP has the RPM option enabled. This is required for using the Demo Unit as a development kit, or for using the Bootload development configuration. This option allocates IO pin P0.7 for use in the bootloading operation. If you do not need the bootload development configuration or want unrestricted use of P0.7, set RPM to 0.

This module also contains the top-level Interrupt Service Routine (ISR) for Timer1. The timer 1 interrupt is used by many different technologies, so there must be some way to distinguish which routine should receive the interrupt. This is accomplished by the system subroutine, TIC_ctl_set. The call to this routine is normally made by the macro that invokes the technology (e.g., CallTalkBoth), making this transparent to most developers. TIC_ctl_set sets bits in the system statics area to indicate the current user of the timer1 interrupt. The timer1 interrupt code in STARTUP.A then checks these bits to decide where to branch. The compiler flags determine which branch tests get built into the code.

The developer can modify this code to allow independent access to timer1. See the comments in the code and also further details in \LIB\UTIL\DELAY.INC.

The STARTUP.A module LIBINCLs \LIB\MEMORY\USERMEM.INC, which defines the presence and size of the optional software stack. If the software stack is present, STARTUP.A will assemble code to initialize the stack and to supply the GLOBAL_SRET module. Normally these code sections are not included.

The GLOBAL_SRET Code Segment (optional)

The STARTUP module also (optionally) includes the code segment GLOBAL_SRET (located at the bottom of the file). This segment consists of a single SRET instruction that is accessible to all code modules via a jmp instruction. This feature is designed to save code space by using the instruction JMP GLOBAL_SRET anywhere that an SRET would normally be used, saving 15 bytes for each instance. No technology code uses scall/sret, so use of the global_sret is a developer's choice. The code is added automatically when startup is assembled if the software stack is enabled in \LIB\MEMORY\USER-MEM.INC. Most RSC-364 applications will not enable the software stack.

As an added benefit, usage of the GLOBAL_SRET segment allows checking for some stack overflow conditions. The overflow handler routine STACK_OVERFLOW may be customized to provide appropriate feedback (the default routine turns on all LED's and then does an infinite loop). Note that this feature requires several extra processor instructions, which may cause a slight system slowdown (especially when external memory handlers are used). Overflow checking is enabled by default. To disable it, simply set the compiler flag OVERFLOW_ENABLE to 0. Note that the overflow check is done only when an sret macro is invoked, so it will not detect stack overflow due to push/pop within a subroutine whose scall just fills the stack - this case will destroy the sret return address.

NOTE: The RSC-364 has native instructions push and pop which operate somewhat like the older push and pop macros. These macros have been renamed spush and spop. The software stack code uses the new instructions, and the old macros should be considered obsolete. They are slower, take more space, and have a different syntax.

Default I/O Configuration

The files described above set the I/O configuration to the default or standard state for the RSC-364 Motherboard used with the I/O Module. The chart on the next page shows the standard I/O usage in various configurations of these boards.

5. I/O Usage in RSC-364 Development Kit Summary

I/O PIN	General Button, Color LEDs, RS-232	Memory Control Extended Address Bits	Keypad ²	Bar LEDs	Serial EEPROM
P0.0	RS-232 rcv			LED 1	
P0.1	RS-232 tx			LED2	
P0.2		A16 ¹		LED3	
P0.3		A17 ¹	R(1)	LED4	
P0.4		A18 ¹	R(7)	LED5	
P0.5		A16 ³	C(1)	LED6	
P0.6		A17 ³	C(3)	LED7	
P0.7	<bootloader> ⁵			LED8	
P1.0	Button A			LED9	SEEP-SCL
P1.1	Button B			LED10	SEEP-SDA
P1.2	Button C			LED11	
P1.3	Red LED		R(4)	LED12	
P1.4			R(0)	LED13	
P1.5	Green LED		C(2)	LED14	
P1.6	Yellow LED			LED15	
P1.7	RS-232 pwr ⁴			LED16	

Table 2: Summary of I/O usage in RSC-364 Development Kit

I/O Usage Notes:

1. Recommended assignment for extended address bits in large ROM-only memory systems. See also Building Multi-Bank Systems for more details.
2. Keypad rows and columns are named for the left-most or top-most key, except the last row is R0.
3. Recommended assignment for extended address bits with large systems with read/write Data Space (Flash).
4. Power control on the Development Kit is controlled by JB12, but the software uses P1.7 for custom applications where the power must be switched dynamically. The Development Kit noise level is low enough that dynamic switching is not required.
5. P0.7 is used to latch the Bootload OTP to allow downloading to the Sample Flash. Applications should not change P0.7 when using the Bootload development method.

6. Using Debug Speech

Debugging information can be selectively provided through speech or the RS232 port. This section describes the speech interface. All speech debug features described here are also available with serial debugging. The next section describes the additional features available with the serial interface. The debug speech software module requires using a set of three files: the object code, an include file, and a

speech data object file. There are two sets of routines in this module. The first set consists of routines that speak the decimal or hexadecimal value of a given nibble, byte, or word. The second set of routines provides Sensory's standard feedback for speech recognition.

Initialization

The debug speech software uses the Speech Synthesis code, so initialization should be done as indicated for that code.

Includes

The debug speech routines are invoked with CALL. For ease of use, macros are provided for the routines that give Sensory standard feedback for speech recognition. These macros, as well as external definitions for all debug speech routines, are in the LIBINCL file \LIB364\MACROS\DEBUG_S.INC.

Include this file in any modules where you want to use debug speech. Remember that these routines use Speech Synthesis, so make sure that \LIB\MEMORY\MEMORY.INC is LIBINCLuded as indicated in Using Speech Synthesis.

Linking

Link the debug speech object code, \LIB364\DEBUG_S\DEBUG1.O with the rest of your code. Also link the speech data object file \DATA\SPEECH\NUMBERSD.O. The speech file should be linked into your Code Space ROM, or into a Data Space ROM that contains other speech data. See Using Speech Synthesis.

Functions

There are six functions in the debug speech package that speak the value of the input data. There are five additional functions that provide the Sensory standard feedback for speech recognition (these routines are incorporated into the two macros described below, so the functions are not described here). NOTE: NEVER USE THESE FUNCTIONS BETWEEN PATTERN GENERATION (PATGEN, PATGENW, CLPATGEN, etc.) AND RECOGNITION (RECOG, RECOGSD, RECOGSV). The debug speech functions use memory that overwrites some of the information generated during pattern generation and which is used by recognition. Placing a debug speech routine between these functions may cause recognition to return unexpected results. Debug speech can be called safely if the PUTTMPLT macro has been called first to save the template information and the GETTMPLT macro is called following the debug output.

NOTE: The following DEBUGXXX are functions (called), not macros.

DebugH4(A)

Speaks hex value of low nibble in A

Input: A Value to be spoken

Output: '0', '1', ... 'F'. No zero-padding.

DebugH8(A)

Speaks hex value of byte in A

Input: A Value to be spoken

Output: '00', '01', ... 'FF'. Zero-padding to two digits.

DebugH16(A)

Speaks hex value of word at RAM address pointed to by A.

Input: A Address of low byte of value to be spoken (word is stored in lsb, msb order)
Output: '0000', '0001', ... 'FFFF'. Zero-padding to four digits. (Word is spoken in normal msb, lsb order)

DebugD8(A)

Speaks decimal value of unsigned byte in A

Input: A Value to be spoken
Output: '000', '001', ... '255'. Zero-padding to three digits.

DebugD16(A)

Speaks decimal value of unsigned word at RAM address pointed to by A (word stored in lsb, msb order)

Input: A Address of low byte of value to be spoken
Output: '00000', '00001', ... '65535'. Zero-padding to five digits.

DebugD100(A)

Speaks decimal value 0 through 100 of byte in A

Input: A Value to be spoken
Output: '00', '01', ... '99', or '100'. Double-beep if A > 100. Zero-padding to two digits.

Macros

Four macros provide Sensory standard speech feedback for recognition. The output is enabled by several bits in the byte `DEBUG_ENABLE` which may be edited in the file `CONFIG.A` (see System Configuration and I/O Mapping above). Each bit in this byte controls a specific debug output, with a zero in a bit disabling the specified output, and a one enabling the output. The values that are output are as follows:

- Bit 0 Class "Answer" (which word) returned from recognition in A.
Level Confidence level or score returned from recognition in B.
- Bit 1 SilenceLevel Noise level measured by INITSL/GETSL.
MaxPower Maximum power in spoken word measured by PATGEN.
- Bit 2 AGC Setting of automatic gain control during patgen (0=lowest gain, 3=highest gain).
- Bit 3 Error Code Error code returned by PATGEN in A.

The 'Class' and 'Level' values will be useful to developers for debugging program flow. The Class value returned is the (0-based) index of the recognized word in the weight set. Except for the digit set, weight sets are always organized in alphabetic order. 'Error Code' will also be useful in debugging program flow. 'SilenceLevel,' 'MaxPower' and 'AGC' are most useful to Sensory engineers and development supporters. Use the `DEBUG_ENABLE` byte to enable/disable some or all of these functions without removing them from the code. The `DEBUG_ENABLE` byte does not effect the `DEBUGHX()` and `DEBUGDX()` functions.

These macros give general support for all recognition technologies.

DBGPATGEN()

If pattern generation is OK, stores SilenceLevel and MaxPower in safe locations during recognition, otherwise, speaks values of Error Code, SilenceLevel, MaxPower, and AGC.

Input: none

Output: speech if error, else none

Notes: Must be called immediately following PATGEN. This macro does not invoke Speech Synthesis if the pattern generation was successful. The value of DEBUG_ENABLE indicates which values to speak as indicated above.

The possible ErrorCodes returned by PATGEN and spoken by DBGPATGEN are:

- 1: timeout (no data)
- 2: spoke too long (too much data)
- 3: background level too noisy
- 4: spoke too softly
- 5: spoke too loudly
- 6: spoke too soon
- 7: too choppy (too many speech segments)
- 8: bad weights (format not correct)
- FF: user interrupted patgen operation

DBGRECOGSI()

Speaks values of recognition variables: Classes, Levels, SilenceLevel, MaxPower, and AGC.

Input: none

Output: speech

Notes: Must be called immediately following a call to RECOG (or following PRIORS if used). The value of the DEBUG_ENABLE byte indicates which values to speak as described above. If the Class/Level bit is set, the following data are spoken in order:

- Best Class (0 to number of words in set minus 1; double-beep if NOTA)
- Confidence level (1-100)
- Second best match
- Confidence of second best match (*)
- Third best match
- Confidence of third best match (*)

(*) If the confidence is 0, a short "double-beep" sounds. Double-beep also is used to indicate NOTA as the class. RS232 output denotes NOTA as "99".

DBGRECOGSD()

Speaks values of Speaker Dependent recognition variables: Classes, Scores, SilenceLevel, MaxPower, and AGC.

Input: none

Output: speech

Notes: Must be called immediately following a call to RECOGSD. The value of the DEBUG_ENABLE byte indicates which values to speak as described above. If the

Class/Level bit is set, the following data are spoken in order:

- One digit pass code
- Best Class (0 to number of words in set minus 1)
- Best Score (0-255), lower is better
- Second best match Class
- Second best score (0-255)

DBGRECOGSV()

Speaks values of Speaker Verification recognition variables: Pass/Fail, Class, Scores, SilenceLevel, MaxPower, and AGC.

Input: none

Output: speech

Notes: Must be called immediately following a call to RECOGSV. The value of the DEBUG_ENABLE byte indicates which values to speak as described above. If the Class/Level bit is set, the following data are spoken in order:

- One digit sequence pass code (0:pass, 1:fail, FF(double-beep)=undecided)
- One digit utterance pass code (0:pass, 1:fail)
- Best Class (0 to number of words in set minus 1)
- Cumulative Score (0-255) - unlike SD, Higher is better

Sample Code

The following is a short program segment using debug speech and Sensory standard feedback for Speaker Independent speech recognition. A similar model is used for Speaker Dependent.

```
.
    mov    a,value1          ;byte to speak
    call   DebugH8           ;speak byte A in hexadecimal
    mov    a,#dpl            ;address of word to speak
call   DebugD16              ;speak 16-bit word at address dpl:dph in decimal
.
.
.
start_patgen:
    PatgenW      #high Wtweights, #0 ;generate the pattern for a SI spoken
word
    DbgPatgen                ;save silence_level and max_power values
.                            ;on error, speak error code, silence level, max power,
AGC
.
.
patgen_ok:
    Recog #high WTweights    ;compare against weight set
    Prior answer, #1         ;prioritize answer (optional)
    DbgRecogSI               ;speak classes, levels, silence level, max
power, AGC
.
```

7. Using PC Output

The RSC-364 Development Kit incorporates software and hardware to allow use of familiar serial communications using the RS-232C standard. The Motherboard contains circuitry to provide the required signal levels to external devices such as terminals or a PC, even when the Development Kit is operated at 3 Volts. A shutdown control is provided in the software to reduce circuit noise during pattern generation and voice recording.

Operation is simplified by hardware design that uses only serial transmit and serial receive signals; all other control signals are "looped back". No flow control is implemented.

Software comprises two modules, one of which contains a few basic, low-level routines to configure the RS-232 system, to receive a character, or to send a character or string. A second module provides formatted serial output of the familiar "debug speech" functions. These functions offer Hex or Decimal output of nibble, byte, or word values. The formatted output package contains equivalents for all the macros and functions described in Using Debug Speech. The `DEBUG_ENABLE` byte described in that section has the same operation when the output is directed to RS-232. See that section for details on use of the "debug" output capabilities. The default operation of the `\SAMPL264\SI6` demo is to output full debugging information to the RS232 port.

If you use the bootloader software to load programs, be sure to read the section on configuring the Serial port in the Overview chapter.

Initialization

The RS-232 software uses a default baud rate of 9600 bits/sec. (An exception is that the bootload program runs at 115K baud). The baud rate is determined by the constant, `BAUD_COUNT`, defined in `CONFIG.A`. Edit and re-assemble `CONFIG.A` to change the baud rate. The software also needs three IO bits for transmit, receive, and power control. The default values for these are set in `IO.INC`. Changing any of these bits requires editing `IO.INC` and re-assembling `\LIB\RS232\RS232.A`. If you use the default configuration, initialize the RS-232 operations using the `INIT232` macro. The software uses polling rather than interrupts, so no other initialization is required. The 364 Development Kit and Demo Unit each include circuitry to produce the required +5V/-10V serial interface voltages when the unit is operating from 3V. This RS232 power converter is electrically noisy, and with certain PCB layouts may generate noise that can affect recognition. It is recommended that the `Idle232` macro be invoked before each call to pattern generation to prevent this potential problem. The `Idle232` macro includes required time delays. NOTE: Insert the `Idle232` macro before the prompt, NOT between the prompt and the call to `Patgen`. Otherwise the user will likely begin talking during the RS232 shutdown time, causing a "spoke too soon" error. After pattern generation returns, the RS232 can be re-enabled with the `Init232` macro. The RS-232 should be disabled during continuous listening or wordspot.

Includes

The RS-232 routines should be called using the macros defined in `\LIB\MACROS\RS232`. Include this file in your application. These macros are described below.

Linking

Link the application code with the lower-level driver routines in `\LIB\RS232\RS232.O`. To use the decimal and hex formatted numeric output routines, link in `\LIB364\DEBUG_S\DEBUG232.O`. The names of routines in this module are the same as those in the speech version, `\LIB364\DEBUG_S\DEBUG1`,

so you may not link both of these modules in the same application.

Macros

There are seven macros in the lower-level RS-232 package. The RS232 macro input parameters are defined as follows:

= 1-byte integer constant; reg = 1-byte register variable; val = either # or reg,

caddr = 16-bit address in Code Space

PUTBYTE232 (val ascii)

Transmits one character out of the RS232 channel.

Input: ascii ASCII-format character to transmit

Output: carry bit 0 = ok; 1 = error

Notes:

WAITBYTE232 ()

Waits until a character is received on the RS232 channel and returns it in the A register.

Input: none

Output: a ASCII-format character received

Notes: Waits forever if no character is received.

ANNOUNCE (caddr message)

Sends a null-terminated Code Space string to the RS232 channel.

Input: message ASCII-format character string to transmit

Output: none

Notes: Message must be terminated by a NULL character (00h). The null character is not transmitted.

CRLF ()

Transmits a carriage-return/line-feed sequence out of the RS232 channel.

Input: none

Output: carry bit 0 = ok; 1 = error

Notes:

INIT232 ()

Prepares the RS-232 channel for operation.

Input: none

Output: none

Notes: Powers up the RS232 circuits and sets port directions.

IDLE232 ()

Puts the RS-232 channel in idle state.

Input: none

Output: none

Notes: Powers off the RS232 circuits. Should be done before Patgen and GETSL.

The following macro requires interrupt-driven software not supplied in this release. The code is provided for developers who wish to develop such software for higher-performance applications. This macro should not be used with the RS232 low-level routines provided in \LIB\RS232. Use WAITBYTE232 instead.

GETBYTE232 ()

Reads a character on the RS-232 channel immediately and returns it in the A register.

Input: none

Output: a ASCII-format character received

Notes: Assumes the start bit has already begun when called.

Sample Code

The following is a short program segment using RS-232 functions to implement a simple command dispatcher using one-character commands received from a terminal. See \SAMPL364\RS232 for the full source code, expandable working sample menuing system, and more RS-232 examples.

```
; SERIAL communication RS232 sample
.
.
.
Initialize:                                ;startup comes here
    Init232                                ; ready rs232 for operation
    Announce Greeting                      ;display greeting
    Announce Help                          ;display menu
Loop:                                        ; Loop to get command and dispatch
    WaitByte232                            ;get char in A
    or      a, #20h                        ; tolower(a)
    cp      a, #'a'
    jz      DoCommandA
    cp      a, #'b'
    jz      DoCommandB
; . . . additional one-character commands as above
    cp      a, #'?'
    jz      HelpCmd
    cp      a, #';'                        ; comment
    jz      Comment
    jmp     Loop                          ;ignore unknown commands
```

```

; . . . more code for command handling

;----- Message strings in Code Space -----
-
Messages      segment "CODE"
NULL equ 0      ;end-of-message
Greeting
    db 13, 10      ;crlf in string
    db "Hello, World"
    db NULL
Help
    ; db continues for menu lisitng and other messages as required

```

8. Using Co-Routines for External Memory Access

Many of Sensory's technologies require access to information in external memory. Speaker Dependent recognition, Speaker Verification recognition, Wordspot SD recognition, Dual Recognition, and Voice Record and Playback each may require external read/write memory to hold the template or voice data. Speaker Independent recognition and Speech Synthesis each require read-only memory to hold the recognition weights or speech data. Due to the wide range of memory products available (SRAM, Flash RAM, Serial EEPROM, etc.), the differences from one device to another, and the very fast rate at which memory technologies change, Sensory cannot provide individual support for every memory device. Instead, Sensory's technology library routines call customer-supplied memory handler co-routines to provide external read/write (or read-only) memory access. Sensory supplies software handlers for a variety of popular memories. These handlers may be used directly or as models for other devices.

This chapter describes the co-routine interface. Unless otherwise mentioned, memory in this chapter means external memory. The memory handler is the co-routine part of the application code written by the user that handles the operations necessary to deal with the memory. The technology code provides specific services (currently Voice Record and Playback, Speaker Verification, Speaker Dependent recognition, Speaker Independent recognition, Dual Recognition, and speech Synthesis), and requests the memory handler to manage the details of communicating with the memory. The memory handler and the technology code typically communicate on a byte-by-byte basis. The interface between the technology code and the memory handler is the same for all memory types. The technology code is unaware of the details or even the type of external memory used by the application.

In concert with the general approach to software handling, the Development Kit hardware supports virtually any device by use of the various sockets, prototyping area, and logic prototyping locations on the Memory Module. As part of the system design, keep in mind that interruption of the power during a write cycle to external memory can cause loss of data.

Memory Types

Each class of memory device has specific issues to address. These issues need to be well understood before a memory device is selected and the co-routines are implemented.

Data Space

External memory devices with parallel data organization will typically interface to the RSC-364 via the address and data buses. Any such read/write device must be in external Data Space, but some programs may also use external Data Space ROM for speech and recognition files. Using a parallel device in such an application normally requires a dedicated I/O pin (e.g., ROM/RAM) and decoding to select between Data Space access to the read/write memory and Data Space access to the ROM. Although the Motherboard does not support such a configuration directly, it can readily be wired up on the Memory Module, as it is a common configuration. In this release of the Developer Kit, the SRAM and the Flash memories interface with parallel data space access.

The RSC-364 maps some addresses in the last page of each Data Space bank (addresses 0xFFxxh) to internal functions. This last page is inaccessible to external devices and must be avoided in any memory handler. That is, the next sequential access to external memory after 0FEFFh must be 10000h. Driver examples in the \MEMORY\FLASH sub-directories illustrate different methods of avoiding page FF.

IMPORTANT NOTICE

The handler code to map around page FF may cause reads/writes to be directed to physical addresses different from the logical addresses supplied. For most use this is not important. But the programmer who needs to access a specific physical location (for example, to unlock write-protection for some device) should not use these handlers, or should comprehend the details of how they work.

SRAM

This is the easiest type of memory to work with, having a parallel interface. Co-routines can simply consist of a MOVX followed by a return (or jmp). There are no specific issues to address other than Data Space decoding. In most instances the application will need to implement a pointer. The major disadvantage of SRAM is that data are lost when power is removed (volatility).

Flash RAM/EEPROM

This memory technology has many variations, but the main characteristic is that the storage is non-volatile. The names "Flash" and "EEPROM" are used more-or-less interchangeably. Some forms of this memory function like an SRAM or EPROM when reading, but may have a fairly complex write procedure that varies from device to device. A major issue in dealing with Flash is often erasing. Flash RAM must sometimes be explicitly erased (in whole-sometimes-large-blocks) before new data can be written. Flash is available in large sizes suitable for extended voice recordings. Interface may be parallel, so Data Space decoding must be considered (for example, the Atmel or SST Flash).

Using Flash RAM may require a portion of the RSC-364's on-chip RAM to be used as a buffer. In the sample code, RAM banks 0Ch-0Fh are used to create a 128-byte temporary buffer. In addition, special consideration should be given to the Flash RAM's block size and write cycle timing. The RSC-364 can be programmed with extended wait states for movx instructions to create a long write or read pulse that may be suitable for some slow Flash memories. See the description of the Oscillator1 Extended Register OSC1EXT (0F7h) in the RSC-364 Data Book. The Atmel page-mode Flash driver (in \LIB\MEMORY\FLASH\SPFLASH) and the SST 28SF040 driver (in \LIB\MEMORY\FLASH\SST) illustrate use of extended movx wait states.

Sensory provides complete working sample code for one popular group of Flash parts that interface via the address bus and dedicated I/O pins (which provide extended address bits). Data transfers are strobed by the -WRD and -RDD signals (MOVX instructions). For more details, see Using Page Mode Flash, below. A sample Flash driver is also supplied for the 28SF040, a 4-Mbit flash from SST.

Serial EEPROM

Access to and from these non-volatile parts is achieved via a bit-serial data bus. Two common protocols are 2-wire (or I2C) and 3-wire. 3-wire parts are typically faster and smaller than 2-wire parts. Like a Flash RAM, data must be erased before the memory space can be overwritten, although most modern parts erase transparently.

SEEPROMs are commonly available in sizes up to 8K bytes. This size is too small to store voice recordings, but large enough to store many speaker dependent or password templates. Because of its low cost and small physical size, Serial EEPROM is a good medium for SD applications that do not require voice recording. Speed is generally not an issue when storing templates because of the small amount of data, but it is an issue for voice recordings with serial memories. Voice Record and Playback requires a continuous throughput rate of roughly 4000 bytes/second.

Sensory provides sample code for using Microchip 2-wire SEEPROMS from 24C01 through 24C64/65. These routines allow storing and retrieving templates for password or speaker dependent technologies, as well as randomly reading and writing individual bytes. The code has also been used successfully with serial EEPROMS from Atmel and other manufacturers.

Memory Co-routines for Read-Only Devices

Memory co-routines are also used with some data stored in read-only memory. The speech synthesis and SI recognition technologies call memory handlers to provide the required data bytes. Typically these data would be stored in EPROM or ROM, and the co-routine approach supports placement of speech data and SI weights (independently) in either Code Space or Data Space as the developer chooses. Use of Data Space placement can ease the "ROM crunch" in larger applications and can allow rapid prototyping of speech vocabularies. A high level of speech compression is not required because there is more available memory. For example, a single 64K binary can be created with executable, speech data, and weights all in Code Space (if the code fits). Typically such a program would consist of about 32K of code and 32K of speech data. If this application had a 40 word vocabulary, then speech would have to be compressed to 800 bytes/word. This level of compression at good quality requires many hours of linguistic work.

Alternatively, a larger 128K program can be built with 64K of executable code and 64K of speech data plus recognition weights (or 64K of code plus weights and 64K of speech data). The 40-word vocabulary now needs to be compressed only to 1600 bytes/word, and this can be done quickly and inexpensively. A larger program of this type can fit in a single 1Mbit ROM. A simple decoding scheme could use Code Space in the lower 64K and Data Space in the upper 64K, with address bit A16 controlled by the -RDC signal. Note, however, that other use of Data Space (e.g., parallel Flash) requires additional device decoding as described above.

Implementation Overview

The function of a co-routine is to transfer one byte of data between the memory handler code and the

technology code. The memory handler code moves the data into or out of external memory. This can be implemented by a simple MOVX command, a serial transfer via GPIO, or a buffering system that transfers data in blocks. In any case, the interface to the technology code is one byte.

The data transfer routines supplied by the memory handler are co-routines because they are not normally called directly by the application; rather the application makes a request for technology service, and the technology code calls the co-routine. The technology code may also provide additional information when it calls the co-routine, or it may call additional helper co-routines. For example, during Speaker Dependent recognition, the recognition technology code makes 128 calls to the memory handler to obtain each byte of the first template. Before beginning this sequence of calls, the co-routine `validate_template_sdv` would be called with the first template number. This co-routine sets up the pointers to access the template and initializes any counters. The sample code illustrates this.

The application must initialize the memory co-routines before calling the technology function that will use them. Initialization involves providing a starting address. Some memory types may require additional specific setup or shut down procedures, for example to write out a partially-filled buffer. The appropriate initialization routine should be implemented in the application and called once before calling the technology code that will read or write. The exit routine should be called after the technology code has returned. As shown in the samples, most memory co-routines should support the following entry points (XXX is either SDV or RP):

- `init_xwrite_XXX` (called by application to initialize co-routine)
- `exit_xwrite_XXX` (called by application to terminate co-routine)
- `init_xread_XXX` (called by application to initialize co-routine)
- `exit_xread_XXX` (called by application to terminate co-routine)
- `get_byte_XXX` (called by technology code when it needs a byte)
- `put_byte_XXX` (called by technology code when it has a byte)

Detailed descriptions and examples of these functions for SD_SV are given below.

In some cases data storage or fetch requests may be made by the technology code via Jumps, not CALLS. This is mainly to minimize stack depth. The data to be transferred will typically be passed in variable "A". Memory addressing is the responsibility of the co-routine. The co-routine must not disturb global registers other than "A". All of the required pointers required for SDV and RP technologies are allocated in system memory and defined in `\LIB\MEMORY\XRAM.INC`. Any other required storage must be provided by the application. Use the samples as a guide to writing drivers for other devices.

Affected Code

Sensory currently has three technologies that rely upon external read/write memory: Speaker Dependent recognition, Speaker Verification, and Voice Record and Playback. Two technologies use co-routines for read-only access: speech synthesis and SI recognition. Co-routines may be developed individually for each of these functions to allow the greatest flexibility (e.g., a product uses Flash RAM for record and playback, but stores passwords in a serial EEPROM). The memory requirements of speaker verification and speaker dependent recognition are basically identical, so they may share the same co-routines. Wordspot can recognize only a single utterance at a given time, so the application can load the template for the trigger word directly, and the wordspot technology code does not need to reference external memory.

Speaker Dependent/Speaker Verification (SD_SV)

This technology uses external memory to store one or more templates of 128 bytes each. Memory writes are done by a call to `put_byte_sdv`, while reads are accomplished with a call to `get_byte_sdv`. There are no critical timing issues for these routines, but the execution time of `get_byte_sdv` will effect the overall response time when recognizing or verifying an unknown word. The resulting delay may become more noticeable when a large number of templates are in the speaker dependent recognition set. An additional co-routine, `validate_template_sdv`, provides the means for the application to tell the technology code to skip certain templates (for example, ones that have been deleted). This can speed up recognition and simplify memory management. Refer to The `Validate_template_SDV` Function, below, for details.

Read/write co-routines are typically implemented as two layers. The low-level modules provide general purpose access routines for writing, reading, erasing, and the like. These lower-layer routines are contained in `\LIB\MEMORY\XXX`, (where XXX is FLASH, SEEPRO, OR SRAM). The flash directory has sub-directories for each supported part. Higher-level modules provide the technology-specific memory management for SD_SV in `\LIB\MEMORY\XXX\SD_SV`.

This release also includes SD/SV support for a group of Flash RAMs (see the example code in `\LIB\MEMORY\FLASH\XXX\SD_SV`).

A special set of memory co-routines for SD/SV provides support for 6 or fewer words stored on-chip with no external memory in the RSC-364. These routines are in the `\LIB364\SD_SV\ONCHIP.O` module rather than in the memory folder.

In order to access templates for SD/SV, the application program takes the following actions:

- Initialize `xram_sdv_base` to point to the base of the template set and perform any hardware initialization needed to access external memory (done by `init_xread_sdv` or `init_xwrite_sdv`, because different initialization is needed for read/write)
- Initialize `xram_sdv_ptr` to point to the start of the current template (done by `validate_template_sdv`)
- Setup hardware lines, as needed, to access the current template (may be done by `validate_template_sdv`, once per template, or by `get_byte_sdv/put_byte_sdv`, once per byte)
- Call one of the following technology routines: `RecogSD`, `RecogSV`, `TrainSD`, `TrainSV`, `GetTemplate`, or `PutTemplate`. These routines in turn call `get_byte_sdv`, `put_byte_sdv`, `validate_template_sdv`
- Finish up memory access, e.g. writing out buffered flash page (done by `exit_xread_sdv` or `exit_xwrite_sdv`, different actions needed for reads, writes)

The table shows the individual SD_SV memory handler routines. See overall flow below for usage examples.

init_xread_sdv	Sets up 24-bit xram_sdv_base pointer to point to the base of the template set and performs any device-dependent actions necessary for reading
init_xwrite_sdv	Sets up 24-bit xram_sdv_base pointer to point to the base of the template set and performs any device-dependent actions necessary for writing
exit_xread_sdv	Returns 24-bit xram_sdv_base pointer and performs any device-dependent actions necessary to complete reading
exit_xwrite_sdv	Returns 24-bit xram_sdv_base pointer and performs any device-dependent actions necessary to complete writing
validate_template_sdv	Sets up xram_sdv_ptr to point to the template number passed in A (0-based) and performs any hardware initialization to allow subsequent "movX @xram_sdv_ptr" instructions to access this template.
get_byte_sdv	Reads the byte pointed to by xram_sdv_ptr and increments xram_sdv_ptr
put_byte_sdv	Writes the byte pointed to by xram_sdv_ptr and increments xram_sdv_ptr

Table 3: Individual SD_SV memory handler routines

The Validate_Template_SDV Function

The validate_template_sdv function is called both by the technology code (e.g. RecogSD) and by the application program. It performs two different functions. The first function, described in the table above, is to set the address pointer to the correct position for reading the first byte of a specific template. In the case of a memory larger than 64kBytes, this means setting up the hardware address bits for the template bank (and adjusting for "page FF holes", if necessary). In some applications, the template itself might be offset within a longer record having additional information. In this case, validate_template_sdv would need to adjust the address pointer by the offset.

The optional second function may be added by the developer: to check the template for validity, returning carry clear if OK, or carry set if invalid. The result of this check would usually be invisible to the application program; it would just be used to direct RecogSD to skip certain templates in the set. Validate_template_sdv would use the passed-in template number to access a corresponding flag to decide whether to include the template or not. The logic involved in the decision could be simple or complex, depending entirely on the application needs.

As an illustration, consider how an application can delete a template. Suppose, for example, that word number 3 is deleted from a seven-word set. The application can physically compact the set to make a 6-word set, and pass this set size when next doing recognition. Alternatively, the application can re-map the requested template numbers around the "hole" created by the missing word3. Most simply, however, the application can still call for recognition on a 7-word set, and simply invalidate the template for word3 via the validate_template_sdv co-routine. This method preserves the template numbers of the other set members. Validate_template_sdv will be called by the technology code before the bytes for the template are requested. If the template is marked invalid, the next template will immediately be requested. Because the method of "marking invalid" is application-dependent, the validate_template_sdv function in this development kit has only stub comments for the feature.

A simple illustration depends on the fact that the first byte of a template encodes the template size (normally 010h = 16d). An application could "erase" a template (remove it from recognition consideration) by writing a "0" to the first byte of the template. Then `validate_template_sdv` would check the first byte: if its value were 16, the template would be marked valid; if 0, it would be marked invalid. The application could later return the template to the recognition set by restoring the 16 value.

A more complicated dialer application might maintain an array of user IDs and directory memberships indexed by template number. `Validate_template_sdv` would access this array to decide about template validity.

NOTE: Different devices might have different assumptions about how templates are stored, and this is OK as long as `validate_template_sdv` can calculate where a given template is stored (e.g. some devices might have 128 byte pages, while others might have 256 byte pages).

Voice Record and Playback (RP)

This technology uses external memory to store voice data recorded through the microphone. Four-bit compression allows a storage ratio of about one second per four Kbytes of memory, so products using this technology will probably use a large SRAM or Flash RAM. There are several components in the RP package, with varying needs. The playback routines only read from external memory, so timing issues should be minimal. Compression (to 3 or 2 bits) requires a lot of reading and writing, and care needs to be taken when using a memory that must be erased (Flash RAM). Compression routines do not run in real-time, so timing is not critical, but processing large amounts of data is time consuming, and every cycle counts. Recording writes data to external memory in real-time and also does some post-processing. Great care should be taken in developing related co-routines - the recording code is both complex and time-critical, and integrating Flash RAM write cycles and block divisions can get very tricky. As a further complexity, Record must often operate concurrently in a multi-tasking fashion with pattern generation.

Sensory has implemented sample co-routines for Voice Record and Playback for SRAM and for certain 1-, 2-, and 4-Mbit Flash memories. The RP co-routine implementation contains several "housekeeping" routines for adjusting pointers and the like. Record and Play co-routines typically use the same low-level code as `SD_SV`. See the RP-specific sample codes in the `\LIB\MEMORY\FLASH\XXX\RP` directory.

RSC Memory Requirements

Memory co-routines may use RSC-364 Register memory available in the XRAM Registers area for any permanent storage (for example, pointers or counters). For Voice Record and Playback, an additional common buffer space has been made available in the RSC's on-chip RAM. This 128 byte buffer space is located in banks C through F of the banked RAM area (accessible through addresses C0h - DFh, and the BANK register). This space is only available during portions of the RP code which need external memory access; it cannot be used during portions of the `SD_SV` technology.

Overall Flow

A simple illustration of the overall co-routine flow in pseudo-code is shown for the case of storing a single speaker dependent recognition pattern.

Application (Main code)	Technology	Memory Handler Co-Routine (a 2nd code module)
Call InitMemory		(initializes memory access)
Call PutTmplt	(PutTmplt): ---process to obtain one byte --- call put_byte_sdv (put_byte_sdv_return): ---process to obtain one byte --- call put_byte_sdv (put_byte_sdv_return): . . . [repeat above for total 128 bytes] [return from PutTmplt]	(put_byte_sdv): write byte 1 to external memory Return (put_byte_sdv): write byte 2 to external memory Return

In addition to providing 128 sequential bytes, the memory co-routine has the responsibility of locating the next template to be processed. For simplicity, in Sensory demos this is typically at the next sequential 128-byte boundary. Applications may have different requirements: for example, templates may be stored within a data base record that includes telephone numbers, voice recordings, or other structured data. The memory co-routine would have to take this structure into account in order to locate the next template. Normally this would be managed by the validate_template_sdv co-routine as described above.

Examples:

In all of these examples, the templates are assumed to start at the 24-bit address:

TemplateBank : high Template0 : low Template0

ctr = number of templates already trained (0-based)

level = recognition security level

To store a new template

NOTE: PutTmplt calls put_byte_sdv repeatedly

```

mov      dpl, #low Template0      ; Point to start of template storage
mov      dph, #high Template0
mov      r7, #TemplateBank
call     init_xwrite_sdv          ; Init co-routines for writing
mov      a, ctr                  ; ctr = index of new template
call     validate_template_sdv    ; Point to template[ctr]
PutTmplt #TMPLT_UNKNOWN          ; Save new template
call     exit_xwrite_sdv          ; Finish up co-routines

```

To recognize against a set of templates

NOTE: RecogSD contains a call to validate_template_sdv for each template and also calls get_byte_sdv repeatedly

```

mov     dpl, #low Template0      ; Point to start of template storage
mov     dph, #high Template0
mov     r7, #TemplateBank
call    init_xread_sdv           ; Init co-routines for reading
RecogSD ctr, level               ; Recognize against template list
call    exit_xread_sdv           ; Finish up co-routines

```

To recognize against a single template in a set (as for Speaker Verification in a strict sequence) the application must first adjust the template base before calling any of the co-routines. Assume `a = word = which template to recognize (0-based index)`

```

mov     r7, #TemplateBank
mov     dph, a                   ; Use a as a 256-byte index
mov     dpl, #0
shr     dph                      ; Double shift right for 128 bytes
rrc     dpl
add     dpl, #low Template0      ; Triple add into start of templates
adc     dph, #high Template0
adc     r7, #0

call    init_xread_sdv           ; Init co-routines
mov     trash, word
inc     trash                    ; Convert to 1-based indexing
RecogSV trash, ctr, level, #1    ; Recognize against template[word]
call    exit_xread_sdv           ; Finish up co-routines

```

RSC-364 Co-Routine Memory pointers, usage, etc.

To keep the Technology code independent of memory type, the macros for RP, SD, and SV do not contain parameters for the required pointers, counters, and other memory control variables. These variables must be initialized using the memory initialization routines for each memory type before reading or writing to the device. The sample code illustrates this.

More complex devices such as Flash memory may require yet additional variables. See RECORDER.A for an illustration of the pointers and counters required for Record and Playback, and see SPKRDEP.A for the corresponding pointers and counters required for Speaker Dependent/Verification. All of these variables are allocated from the system XRAM space, not the user variable space. Note that, due to the limited SRAM available, application-specific overlays are necessary. Thus use of RP will destroy SD pointers. Usage of all these variables should therefore be considered temporary.

Memory devices are commonly larger than 64K bytes and so require more than 16 bits of address. The memory co-routines in this release support 24-bit addressing (the extended address bits are supplied by IO pins).

Note that hardware stack usage can be significant when technology code is active (see the table near the end of this chapter). The sample co-routines supplied with this release have all been tested in simple demonstration programs that invoke technology macros with no more than one stack level used. Stack overflow is possible when using these drivers in applications that invoke technology macros from within

nested CALLs. For the same reason, memory co-routines should be written to be as "flat" as possible and to avoid using deeply-nested CALLs.

Sample Code

The Development Kit includes several different code samples to illustrate the design of memory handler co-routines. Developers are free to use or modify the code samples, but any responsibility for their use rests with the developer. Because specifications change, developers should obtain data sheets from the manufacturer or authorized distributor and thoroughly understand the operation of any memory device considered for design. Below are the access points for the manufacturers of the memory devices described below.

Device	Manufacturer	Web Site
AT29C010	ATMEL	http://www.atmel.com/
24C01, ..., 24C65	Microchip	http://www.microchip.com/
28SF040, 29x010, 020	Silicon Storage Technology	http://www.ssti.com/

The following is a short program segment illustrating the (16-bit, not 24-bit) simple co-routine SRAM interface for the speaker dependent recognition and speaker verification technologies. This code may be used as a template for developing co-routines for other types of memory. As examples of such alternate implementations, see the directories \LIB\MEMORY\SEEPROM and \MEMORY\FLASH\XXXX. The SEEPROM directory contains an implementation for 2-wire serial EEPROMs that can be linked with either the password or speaker dependent sample programs. The FLASH directories contain the same functions for the Atmel AT29C010 1-Mbit Flash, the SST AT29x020 2-Mbit EEPROM, and the SST 28SF040 4-Mbit Flash (and compatibles). Most of the sample programs in this release can be configured for the 29x020 SPFlash.

Note how the get_byte_sdv routine simply provides the byte, and the validate_template_sdv routine uses the template number provided in the "a" register to calculate the location of the template when it is called.

```
SRAM_SDV.A: SRAM control for speaker dependent and speaker verification.
;
; INCLUDES
;

LIBINCL "\lib\memory\memory.inc"

;
; PUBLICS
;

PUBLIC  init_xread_sdv
PUBLIC  exit_xread_sdv
PUBLIC  init_xwrite_sdv
PUBLIC  exit_xwrite_sdv
PUBLIC  get_byte_sdv
PUBLIC  put_byte_sdv
```

```

PUBLIC validate_template_sdv

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EXTERNS ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; START OF CODE ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

xram_sdv:      segment "CODE" at 04000h

;-----
;
; Setup for reading & writing static ram
; inputs:      dptr      16-bit base address of template list
; outputs:     none
; porting information:
;      Set xram_sdv_base to the input pointer.
;
init_xread_sdv:
init_xwrite_sdv:
        mov     xram_sdv_base+0,dpl      ;set start of first pattern
        mov     xram_sdv_base+1,dph
        ret

;-----
;
; Cleanup after reading & writing static ram
; inputs:      none
; outputs:     dptr      16-bit base address of the template list
; porting information:
;      Return the base address of the template list.
;
exit_xread_sdv:
exit_xwrite_sdv:
        mov     dpl,xram_sdv_base+0      ;return base address
        mov     dph,xram_sdv_base+1
        ret

;-----
get_byte_sdv:
;
; Read a byte from static ram
; called by GETTMPSD, which may be called from RECOGSD
;
; inputs:      none
;
; outputs:     a          data read
;
; regs:        xram_sdv_ptr maintains the pointer into the template
;

```

```

; Notes:
;-This function keeps a pointer to indicate where to get the next byte of
; data. The pointer is initialized in validate_template_sdv, so that
; routine must be called at the start of each template.
;-Changing the template number is accomplished in validate_template_sdv.
;
; Porting information:
;     Read the value from external memory at xram_sdv_ptr.
;     Increment xram_sdv_ptr.
;     Return the byte in A.
;
    movx    a,@xram_sdv_ptr                ;get byte from xram
    inc     xram_sdv_ptr+0                 ;increment pointer
    adc     xram_sdv_ptr+1,#0
    ret

;-----
;
; Write a byte to static ram
;
; inputs:      a                data to write
;
; outputs:     none
;
; regs:        xram_sdv_ptr maintains the pointer into the template
;
; Notes:
;-This function keeps a pointer to indicate where to get the next byte of
; data. The pointer is initialized in validate_template_sdv, so that
; routine must be called at the start of each template.
;-Changing the template number is accomplished in validate_template_sdv.
;
; Porting information:
;     Write the byte in A to external memory at sram_sdv_ptr.
;     Increment xram_sdv_ptr.
;
put_byte_sdv:
    movx    @xram_sdv_ptr,a                ;put byte into xram
    inc     xram_sdv_ptr+0                 ;increment pointer
    adc     xram_sdv_ptr+1,#0
    ret

;-----
;----- USER FUNCTIONS -----
;-----

;-----
;VALIDATE_TEMPLATE_SDV
;
; Tells the system if the user code invalidated a template, and
; sets the memory pointer to the data portion of the template.
;
; inputs:      A                number of template to use (0..?)

```

```

;
; outputs:      carry set if the template is not valid
;               xram_sdv_ptr points to template's data area
;               A and B can return data to user code
;
; regs:         This routine can alter registers A, B, and trash.
;               This routine sets xram_sdv_ptr.
;
; Porting information:
;   Applications that erase templates without compacting the list
;   can use this routine to prevent RecogSD from using those templates.
;   User application code may use this routine for other purposes.
;
;   This routine must set xram_sdv_ptr to the start of the data area
;   of the selected template before exiting.
;
validate_template_sdv:

;set xram_sdv_pointer to address of template

        mov     xram_sdv_ptr+0,#0
        mov     xram_sdv_ptr+1,a
        shr     xram_sdv_ptr+1
        rrc     xram_sdv_ptr+0
        add     xram_sdv_ptr+0,xram_sdv_base+0
        adc     xram_sdv_ptr+1,xram_sdv_base+1

;   << Put optional user code here >>
;   << - Add offset of validation byte >>
;   << - Test validation byte >>
;   << - Move pointer to start of data area >>

;xram_sdv_ptr must point to starting address of the data area of
;template 'A' at this point.

;if slot is valid, clear carry bit, else set it

        clc
        ret

;-----
END

```

Memory Co-Routines Using I2C Serial EEPROM

This release offers full support for development of products using a wide variety of 2-wire (I2C) Serial EEPROMs such as the 24xCxx devices manufactured by Microchip Technology, Inc.

1. The Memory Module contains a 24C65 chip already wired into the circuit.
2. The IO.INC file contains default Serial EEPROM ("SEEP") bit assignments.
3. The driver supports Speaker Dependent and Speaker Verification applications using devices

therange in size from one SD_SV template to 64 templates. The 24- series of serial EEPROMs is too slow for Voice Record and Play.

4. The Speaker Dependent sample and some others have options for storing non-volatile templates in the 24C65.
5. Some varieties of SEEP can operate at lower voltages with slower access speed. The driver code offers configuration options for standard speed parts and fast parts.
6. Some 24xCxx parts allow multiple devices be wired together on the same serial bus for extended addressing. The driver code supports full 16-bit addressing, allowing multiple devices as suitable.

This section discusses some details of the SEEP interface and additional important programming considerations.

The key features of I2C Serial EEPROMs are:

- Bit-serial access via I/O lines (serial clock, SCL, and serial data, SDA)
- Byte-read and byte-write command sequences
- Sequential-read and sequential-write command sequences
- Automatic erase before write
- Hardware write-protect (some types)

Commands

The Serial EEPROM is accessed via command sequences over the serial bus. The RSC-364 always acts as the I2C master device and produces the SCL serial clock signal; the SEEP acts as the slave device.

Addressing and Device Selection

All addresses are virtual; since it is not accessed through a memory space, there is no physical address for the serial EEPROM. The virtual address is provided as part of a command. Commands either include an address or imply an address already latched in the device by a previous command. Every command starts with a control byte that may contain a physical device select. (Some device types support multiple devices on the serial bus; others do not.) The driver code handles the different protocols used by different-size parts. The code supports 16-bits of address, with the 2 bytes being passed in registers dpl and dph. The largest device (24C65) uses only 13-bits, and smaller devices use fewer. For devices that can be bussed, device selection is done by addressing. That is, the first 24C65 (device #0) is addressed at locations 0000-1FFFh, the second device is addressed at locations 2000-3FFFh, and so on for a maximum of 8 devices. The 24C65 on the Memory Module is configured as device #0.

Sequential and Byte Access

Serial EEPROMs can be accessed byte-by-byte, or they can be accessed sequentially. The disadvantage of byte access is slow speed because a complete command sequence must be transacted for each byte. Typically this is 3 or 4 bytes (control, address-low, maybe address-high, data). Sequential access speeds up the access significantly, but the software is more complex. Each template is typically written only once (at training). But every template is read one or more times for each recognition. Because of this asymmetry, Sensory recommends using byte-write and sequential read when accessing templates. The code supplied is pre-configured this way.

RSC on-chip Buffers

The SEEP driver code needs no buffers except in the case of sequential writing. Unfortunately different devices have different buffer sizes, which also may vary between manufacturers. If you decide you need the minimal speed advantage of sequential write, study the manufacturer's data sheet carefully and modify the code accordingly before enabling sequential write.

Hardware and Software Data Protection

Some of the Serial EEPROM types offer a write protect pin. Some devices offer security options that allow software write protection of portions of the device. Sensory's code ignores these features.

Memory Co-routines Using "Page Mode" Flash ("sector program mode" EEPROM)

This release supports development of products using a number of similar 1- and 2-Mbit Flash Memories that utilize a 128-byte page/sector. The features of these parts are all similar to the Atmel AT29C010, so these parts are referred to generically as "Atmel".

1. The Memory Module contains PLCC sockets that can accept one of these parts. Socket PLCC-B has a 29x020 installed by default.
2. All appropriate code samples have been configured to operate with this type of flash as the default.
3. Tested sample driver code is provided (\LIB\MEMORY\FLASH\SPFLASH). This driver contains all the hooks necessary to do both template management and voice recording and playback. If you intend to use this driver we recommend you first gain familiarity by studying the SPKRDEP or RECORDER demo.
4. Several utility functions are provided in the driver module.

This section discusses some details of the Atmel interface and additional important programming considerations.

The key features of the Atmel Flash are:

- Parallel access via address and data bus
- Byte-read capability
- Block-write only, 128-bytes (called a "page" in the code)
- Automatic erase before write
- Software data protection

The memory handler supplied for the Atmel Flash will also work with the SST equivalent, SST 29x010, the Winbond equivalent, W29x010, and the 2-Mbit equivalent parts, the SST 29x020 and Winbond W29x020. NOTE: Atmel's AT29C020 will not work! This device uses a 256 byte page size, and there is insufficient SRAM on the RSC for the required buffer.

Addressing

The Atmel (and equivalent) Flash requires a 17-bit address for 128 kbytes (or an 18-bit address for 256 kbytes). The low 16 bits of address are transferred to the device via the address bus, and the upper bit(s) is provided by I/O signal(s). The driver code supports 18-bits of address, with the 3 bytes being passed

in registers dpl, dph, and r7. The I/O pins assignments for A16 and A17 are made in \LIB\STARTUP\IO.INC. The driver allows these bits to be re-mapped to any I/O pins.

Since the Atmel Flash obtains addresses A0-A15 by the address bus, care must be taken to avoid conflicts with Data Space page 0XFFXXh, which is used internally by the RSC. The driver code illustrates one way to map around the FF pages.

RSC on-chip Buffers

Reading is similar to an EPROM or SRAM and requires no buffer. Writing templates is un-buffered using the Atmel "byte-load" operation (also similar to writing to SRAM). Since all the data are available at the start, the code can simply stream the data to the Flash at a rate fast enough (<150 (sec per byte) to assure that the Flash does not begin its automatic internal programming cycle until all bytes have been written.

Speech samples during Voice Recording are available at a speed (approximately 250 (sec per sample) too slow to allow use of the byte-load approach, so writing Voice Recording data to the Atmel Flash requires buffered writes. The large (128-byte) write sector size of the Atmel requires great care in allocating the RAM buffer. It happens that RSC RAM banks C through F are available during the voice recording operation, so these banks are used for the sector buffer. Note that these specific banks are not available when other technology code is active.

Software Data Protection

The driver code contains a macro, DO_SDP_HEADER, that is used to prevent accidental writes to the Atmel Flash. The macro is invoked before every sector write to disable write protection temporarily. Protection is automatically restored after the write.

Memory Co-routines Using 4-Mbit SST Flash

This release incorporates support for development of products using the Silicon Storage Technology SST 28SF040 4-Mbit SuperFlash EEPROM ("SST Flash"):

1. Sample co-routine driver code (but not complete sample programs) for the chip (\LIB\MEMORY\FLASH\SST). These drivers provide all the hooks necessary to do template management or voice recording and playback. If you intend to use these drivers we recommend you first gain familiarity by attaching the driver to the SPKRDEP or RECORDER demo.
2. The IO.INC file contains default SST Flash extended address bit assignments. These may be modified as required.
3. The driver files contain a description of the modifications to the Memory Module required to use the SST Flash.
4. Socket PLCC-B can accommodate the PLCC package version of this device, making it easy to develop with no modification to the Memory Module.

The SST code illustrates adding wait states during read access to allow use of slow (280 nsec) parts.

This section discusses some details of the SST interface and additional important programming considerations.

The key features of the SST Flash are:

- Parallel access via address and data bus
- Byte-read capability
- Byte-write capability
- Erase-before-write required, 256 byte erase block

Addressing

The SST Flash requires a 19-bit address for 512 kbytes. The low 16 bits of address are transferred to the device via the address bus, and bits A16-A18 are provided by I/O signals. The code supports 19-bits of address, with the 3 bytes being passed in registers dpl, dph, and r7. Extension to larger parts is straightforward with additional I/O pin assignments.

Since the SST Flash obtains addresses A0-A15 by the address bus, care must be taken to avoid conflicts with Data Space page 0XFFXXh, which is used by the RSC. The code to map around the FF pages is contained inline at the top of the routines where it is needed. The driver allows the three address extension bits to be re-mapped to any I/O pins (see IO.INC).

RSC on-chip Buffers

Reading with the SST SuperFlash is similar to an EPROM. Commands are used to initiate other memory operations. Write operations ("byte_program") consist of a command byte and one data byte. No buffer is required.

A byte must be erased before it can be programmed. Utility routines are provided to erase by sectors (256 bytes) or to erase the entire chip. No buffering is required.

Software Data Protection

The SST powers up with active software data protection to prevent accidental writes. The driver code contains a subroutine, FLASH_INITIALIZE, that removes the software data protection. Minor modification of this code (shown in comments) allows re-enabling software data protection.

9. Using the Bootloader

A 264T or 364 Development Kit can be configured to emulate a Demo Unit 264T or 364 by selecting PC DOWNLOAD mode on the Memory Module. In this mode, the development kit hardware functions identically to the hardware found on the Demo Unit, allowing evaluation of the demonstration programs and use of the bootload facility. This section describes the bootload process on the Development Kit hardware. See separate documentation for using the bootloader on the Demo Unit.

In this document, commands to be typed by the user are shown in bold.
For help when using the bootload.exe program, type:

```
C : >BOOTLOAD ?
```

The bootload program, BOOTLOAD.EXE, is placed in the \BIN library by the RSC-364 Development Kit installation process. Running under DOS, this file will reference the current directory for its data file.

As described elsewhere, the \LIB directory must be in your path.

How to start the boot loader and connect it to the PC:

1. Using a serial cable, connect the serial port of the Development Kit to the serial port of the PC.
2. Make sure the Boot Load OTP is installed in socket PLCC-A, the Sample Flash is installed in socket TSOP-B, and the PC_DOWNLOAD jumper, JB4, is installed. Remove any adjacent jumpers.
3. While holding down the BOOT button (located on the Memory Module near the rotary PROGRAM SELECT switch), press and release the RESET button (the button nearest the RSC-364 chip). If the boot loader program in the Boot Load OTP successfully started, you will hear a beep and the Green LED and LEDs 1 and 8 will light. Release the BOOT switch after the beep.
4. Type "bootload" at the DOS prompt on your PC. Press Enter or Return. The boot loader defaults to com port 2, at 115,200 baud. The baud rate CANNOT be changed from 115,200 baud. To use com port 1, at the prompt type:

```
C:\ bootload -P1 -B115
```

5. If the boot loader successfully communicated with the PC, the Yellow LED will turn on, and the PC screen will read:

```
Initializing communication on port 2 at 115200 baud...
```

```
Boot Loader
RSC 364
Version 1.2
Sensory, Inc.
```

```
(D)ownload to RPM or (U)pload from RPM?
```

The version number might be different if you use a Demo Unit.

If the boot loader did not start correctly, see below for possible causes.

6. To download new code or data into the Memory Module, press "d". Press Enter or Return.
- The PC program will prompt for the type of memory device to download into. To download a new program into the Sample Flash, select code space.

```
Download to (C)ode space, (D)ata space, or (S)EEPROM?
```

To download to code space (Sample Flash), press "c".

To download into data space (SPFlash), press "d".

To download into the serial EEPROM, press "s". Press Enter or Return.

- If you selected code space or data space (Flash), the program will prompt for the sector to download into:

```
Select code space sector (0-7):
```

If downloading to the Sample Flash, rotate the PROGRAM SELECT switch on the Memory Module to the number of the program sector where you wish to download the new program. Enter the corresponding sector number at the PC prompt (eg. "0"), and press Enter or Return.

- For downloading to code space, data space or serial EEPROM, the PC program will then prompt for an input file. This should be a binary image of the data to be loaded into the memory device:

```
Enter name of file to read:
```

Type in the name of the file that you want to download. If the file is in the same directory that you called the bootload.exe program from, you can simply type in the file name (eg. "code.bin"). If the file is in another directory, you must type in the path of the file relative to the directory from where bootload.exe was called. Press Enter or Return after typing in the file name.

- The PC program will report the current download settings, and the progress of the download. After approximately 40 seconds, the program will beep twice, indicating a successful download. (Sectors 0 and 7 take a little longer). The "RPM" comment refers to Sensory's Rapid Prototyping Board, a small hardware platform useful for prototyping.

```
Downloading c:\sensory\fddemo\code.bin to RPM code space, sector
0.....
.....
.....
.....
.....
.....
.....
Download completed successfully.
```

7. To upload data from the Memory Module (from data space SPFlash or serial EEPROM only), press "u". Press Enter or Return.

- The PC program will prompt for the type of memory device to upload from. Data can only be uploaded from the data space Flash memory, or the serial EEPROM. It CANNOT be uploaded from the code space Sample Flash (where programs reside).

```
Upload from (D)ata space or (S)EEPROM?
```

To upload from data space Flash, press "d". To upload from the serial EEPROM, press "s". Press Enter or Return.

- If you selected data space, the program will prompt for the sector to upload from:

Select data space sector (0-3):

Enter the corresponding number at the PC prompt (eg. "0"), and press Enter or Return.

- The PC program will then prompt for a target filename. This file will be created by the PC program, and will be a binary image of the data retrieved from the memory device. NOTE: the PC program will overwrite any file of the same name as the file specified as the output target file.

Enter name of file to write:

Type in the name of the file that you want to upload into. If the file is in the same directory that you called the bootload.exe program from, you can simply type in the file name (eg. "code.bin"). If the file is in another directory, you must type in the path of the file relative to the directory from where bootload.exe was called. Press Enter or Return after typing in the file name.

- The PC program will report the current upload settings, and the progress of the download. After approximately 10 seconds, the program will beep twice, indicating a successful upload.

```
Uploading RPM data space, sector 0, to
test.bin.....
.....
.....
.....
.....
.....
.....
Upload completed successfully.
```

Troubleshooting Tips

If the boot loader does not work, check the following items:

- The 9 pin RS232 cable is correctly and firmly plugged into the computer and the Motherboard.
- The Boot Load OTP is installed correct in socket PLCC-A
- The Sample Flash is installed correctly in socket TSOP-B.
- The JB4 PC_DOWNLOAD jumper is installed and other adjacent jumpers are removed.
- The unit was reset with the BOOT switch held down.
- The Program Select knob is rotated to the correct number.
- The correct port setting was used in the command to start the bootloader
- The computer supports serial communications at 115,200 baud.

NOTICE

The Bootloader program can appear to lose data when uploading or downloading to the Data Space flash (SPFlash) in socket PLCC-B. Contact Sensory for assistance when attempting to upload or download to that device. Bootload works reliably for writing to the Sample Flash and reading and writing the Serial EEPROM.

10. Building Multi-Bank Systems

Applications using large amounts of speech or very large program code may require managing memory larger than the 16-bit addressing capability of the RSC-364. Such applications must use Bank Switching. Physically, bank switching is accomplished by using one or more I/O pins to act as extended address bits for larger memories. Most applications larger than 64K need abundant storage for synthesis data or recognition weights, but may require less than 64K of executable code. These applications can be implemented using only Data Space placement of speech data and SI recognition weights (see Banked Data Space below).

Programs requiring more than 64K of executable code must use ROM Code Bank switching. In Release 6, Sensory does not support ROM Code Bank Switching on the RSC-364. Only ROM Data Bank Switching is supported. Therefore any developer requiring ROM Code Bank switching will have to implement the required mechanisms (subroutine jump and return tables, interrupt vectors, parameter passing, etc). It is recommended to contact Sensory before beginning such a development effort.

Banked Data Space

The simplest multi-bank system has all the executable code in one 64K Code Space ROM and data (only) in one or more additional banks. The additional banks are mapped to Data Space, selected by I/O pins. This configuration is referred to as "Data Space placement" of weights and synthesis data (or other application-specific fixed data). The RSC-364 Developer Kit supports configurations of this type as large as 4 Megabits requiring only one inexpensive external decoder IC. Larger systems may be built with slight additional decoding. The general approach is the following:

- The executable code is placed in the lowest 64K bank of the extended address space.
- The ROM output_enable pin is enabled by either the -RDC signal or the -RDD signal. This assures that the ROM is accessed by code fetches as well as movc and movx instructions.
- The extended address bits of the ROM (A16, A17, A18) are logically conditioned to be driven by bank-selection I/O pins whenever the -RDD signal is active (low), but driven to 0 whenever the -RDC signal is active. This assures that the lowest bank is accessed by code fetches and movc instructions, and the other banks are accessed by movx instructions.
- The speech and weights "handlers" are configured to supply data from Data Space.
- If the application requires read/write Data Space memory (for example Flash to store a voice recording), additional decoding must be implemented to choose the correct device when reading from Data Space.

The figure below illustrates an example of a 4 Mbit application with no external read/write memory. The example shows that the 64K bytes of code space is programmed in the lowest bank (address 00000h ~ 0FFFFh), and data space is programmed into the upper banks (address 10000 ~ up to 3FFFFh). P0.2, P0.3, and P0.4 are used for A16, A17, and A18 respectively for the standard extended address bus.

When the RSC364 reads "code" from the memory IC, the RDC signal goes low. The logic shown in the example will select the lowest bank by forcing A16, A17, and A18 to go low, and it will enable the IC and its outputs for the "code" to be read.

When "data" need to be read, the RDD signal goes low. It will select one of the upper banks by enabling the extended standard address bus of A16, A17, and A18. NOTE: This simple design works only if there is no external read/write memory. The requirement for Flash, SRAM, or other data space device requires additional decoding.

In the example shown, any one of sockets (DIP, PLCC-A, and PLCC-B) or TSOP-A can be used for a multi-bank program. (PLCC-A is limited to 2Mbit).

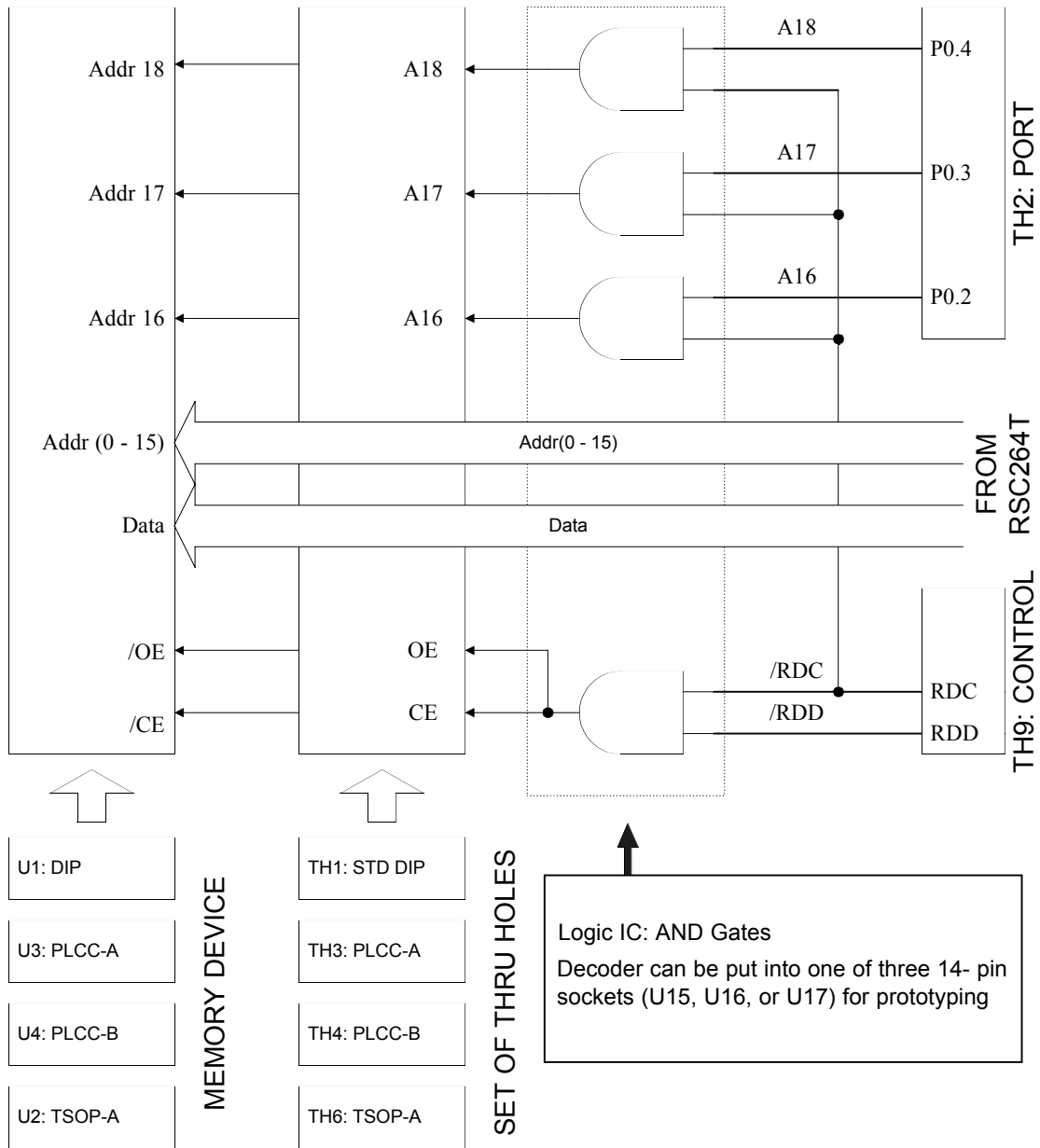
To Use PLCC-A for 2Mbit OTP such as AT27C020-12TC:

- Remove a shorting block from JB4, JB6, JB2, or JB3 if there is any.
- Data bus and Address A0 ~ A15 are already connected to PLCC-A
- Install a 74HC08 in one of three 14-pin DIP sockets (U15, U16, U17), and solder the hook-up wires from TH2: PORT and TH9: CONTROL to the 74HC08.
- Solder the hook-up wire from the 74HC08 to TH3: PLCC-A.

The RSC-364 assembler and linker can deal with only 64Kbytes, so the entire application must be built in sections of this size. Each Data Space bank must be built separately, and data may not "wrap" from one bank to the next. The application must explicitly select the bank control bits before calling synthesis or recognition. Remember also that the last page of each Data Space bank (addresses xFFxxh) is mapped internally in the RSC-364 and is not accessible externally.

As an illustration of this software method, the CLSI demo can be configured to allow Data Space placement of the weights and speech data.

See also Memory Module Configuration, RAM Allocation, and Allocating ROM for Speech and Weights in the Overview section.



11. Using the Floating Point Library Routines

Sensory's technology code uses a library of floating point math routines. This section documents the format and usage of that software for the developer whose application also needs floating point. The floating-point software is available in object code form in the directory \LIB364\UTIL. The file \LIB\UTIL\SAMPLEFP.TXT contains code snippets illustrating usage of the FP routines.

Initialization

No special initialization is required to use the FP routines.

Includes

The FP routines are accessed via CALL functions rather than by macros. Applications should LIBINCL the file \LIB\UTIL\FPCONST.INC to define the required bit definitions. If you use integer multiply or divide, LIBINCL the file \LIB\UTIL\MATH.INC.

The FP routines use some memory locations that are defined in MEMORY.INC, so this file must be INCLUDE'd in the application program. See Using the RSC-364 Memory Map for detailed information.

Linking

Link the application code with the modules \LIB364\UTIL\INTMATH.O (to include integer multiply or divide) and \LIB364\UTIL\FLOAT2.O. If you use the integer routines, be sure to read the comments in \LIB\UTIL\MATH.INC.

Functions

The floating-point library maintains an internal floating-point accumulator (FAC). The FAC is not accessed directly. Instead values are loaded into the FAC using the LDFAC function and stored from the FAC using the STFAC function. 4-byte floating-point operands are referenced with the DPTR pseudo-register pair (DPL, DPH). Floating-point operands are assumed to reside in internal data RAM unless the FACRNM bit is set in the FACBIT byte. If FACRNM is set, the operand is in code space. FACBIT is stored in the technology portion of internal RAM and includes the following bits:

FACSGN	equ	80h	; Sign of value in FAC
FACBTM	equ	40h	; Bit temporary
FACRNM	equ	20h	; Flag to signal ROM resident constant
NANFLG	equ	10h	; Not-A-Number Result Flag
OVFFLG	equ	08h	; Overflow (Infinite) Result Flag
UNFFLG	equ	04h	; Underflow Result Flag

The two least significant bits are reserved for result codes. They are zero if the operation was successful. Otherwise they are set as follows:

ERUNF	equ	1	; FPERR underflow error code
EROVF	equ	2	; FPERR overflow error code
ERNAN	equ	3	; FPERR invalid operation error code

Floating-point results are always placed in the floating-point accumulator, FAC. Integer operands and

results are stored in the B, A pseudo-register pair. B holds the most significant byte, and A the least significant. The following functions are available in the floating-point library. Each is accessed via "call <function>":

Table 4 Functions available in the floating point library

Function	Description
LDFAC	Load the FAC with the single precision operand referenced by DPTR.
STFAC	Store the single precision value in the FAC to the address referenced by DPTR.
FPADD	Add the single precision floating point operand referenced by DPTR to the FAC.
FPMUL	Multiply the FAC by the single precision floating point operand referenced by DPTR.
FPDIV	Divide the value in the FAC by the single precision floating point operand referenced by DPTR.
FPRDIV	Divide the single precision floating point operand referenced by DPTR by the value in the FAC.
FPCMP	Compare the value in the FAC to the operand referenced by DPTR. The A pseudo-register is set according to the results of the comparison: 00 if FAC = OPN within the most significant 20 bits 01 if FAC > OPN FF if FAC < OPN 80 if no relation between FAC and OPN NOTE: the FPCMP operation destroys the contents of FAC.
FLOAT	Convert the two's complement integer in B, A into a single precision floating point value in the FAC.
INT	Convert the single precision floating point value in the FAC into the greatest integer less than or equal to the FAC value. Store the two's complement result in B, A. This operation rounds down.
FIX	Convert the single precision floating point value in the FAC into an integer. Truncate any fractional part. Store the two's complement result in B, A. This operation rounds toward zero.
FPEXP	Compute e to the power specified in the floating point accumulator.

Format

The floating point library routines store single precision floating point values in four bytes according to the following format based on the IEEE 754 standard.

Address	0	1	2	3
Format	MMMM MMMM	MMMM MMMM	EMMM MMMM	SEEE EEEE

Note that floating point values - as all multi-byte values in the RSC-364 -- are stored with Intel byte

ordering (i.e., most significant bytes at highest addresses).

The bit fields making up the floating point value are interpreted as follows:

S	Sign bit: 1 if negative, 0 otherwise.
EEEE EEEE	Base 2 exponent biased by 127 decimal. 0 and 255 are reserved for special values (see below). Therefore: $-126 \leq \text{exponent} \leq 127$
MMM MMMM MMMM MMMM MMMM MMMM	23 bit normalized mantissa with an implicit 1 in the most significant position (24th bit). A binary point is assumed after the implicit 1 in the most significant position. Therefore: $1.0 < \text{mantissa} < 2.0$.

Several special values are also defined:

SPECIAL VALUE	S	EEEE EEEE	MMM MMMM MMMM MMMM MMMM MMMM
Zero	0	0	0 (no implicit 1 in most significant bit)
Signed Infinity	0 or 1	0FFH	0 (no implicit 1 in most significant bit)
Not a Number (NaN)	don't care (usually 1)	0FFH	non-zero (usually 7FFFFFFH)

Accuracy

Floating-point results were compared against results obtained using the Borland C++ version 4.0 math routines. In general the results agreed. One test case in the addition test disagreed in the least significant bit. In this case a higher precision calculation would have had a one in the next least significant bit, and zeroes in all succeeding less significant bit. Borland seems to round this down; the FPADD routine rounds up. All other test results agreed with the Borland library except for exponentiation (FPEXP). Most results had errors in at most the least significant one or two bits. One case however differed in the least significant six bits. This still provides 18 bits of significance, which should be adequate for most applications.

Memory Usage

The floating-point library uses a variety of RAM locations to store the floating-point accumulator, FACBIT, and other temporary storage locations. All of these locations are overlaid with other technology code when floating point is not being used. Thus the floating-point memory is not free and available in applications not using floating point. The ROM requirement for the routines is about 3K bytes.

12. Stack Usage

Each Routine is invoked by a call (perhaps via a macro) that uses one stack level. This call is included in the stack requirements. The hardware stack on the RSC-364 is used by subroutines and interrupts and is 8 levels deep. The table includes any required interrupt usage.

Library Macro/ Function Name	HW Stack (levels) {8 max}	Library Macro/ Function Name	HW Stack (levels) {8 max}
Speech Synthesis		Speaker Independent Recognition	
CallTalkDAC ²	2	Recog	6
CallTalkPWM ²	2	Prior	1
CallTalkBoth ²	2	Speaker Dependent Recognition	
SenTalkDAC ^{2, 3}	3	PutTmpl ⁶	1
SenTalkPWM ^{2, 3}	3	GetTmpl ⁷	1
SenTalkBoth ^{2, 3}	3	TrainSD	2
Pattern Generation		RecogSD ⁷	3
LdCtl	1	Speaker Verification	
InitSL	1	PutTmpl ⁶	1
GetSL ³	1	GetTmpl ⁷	1
Patgen ³	3	TrainV	2
PatgenW ³	3	RecogSV ⁷	3

13. Include, Link, and Co-Routine Requirements for 6. 0 Technology

(See notes at end of table. Wordspot does not require co-routine support)

Technology Used	Include .INC (\libxxx\macros\)	Link .O (\libxxx\)	Co-routines Required¹
All Applications using technology code		Analog.o delay14.o const364.o	
Pattern Generation	patgen memory	Patgen\patgen Patgen\block Patgen\digagc Patgen\stream Patgen\nulogfns Patgen\normal	PatgenHandler ³ PatgenMemDriver ³
Speaker Independent Recognition	si memory	si\recog util\intmath util\float2 si\prior ⁷	RecogMemDriver1 ³ RecogMemDriver2 ³
Speaker Dependent Recognition	sds memory	util\intmath sd_sv\sd	init_xread_sdv init_xwrite_sdv
Speaker Verification	memory\xram sds memory memory\xram	sd_sv\template sd_sv\sdweight sd_sv\svweight sd_sv\onchip ² sd_sv\auxtmplt ²	exit_xread_sdv exit_xwrite_sdv get_byte_sdv put_byte_sdv validate_template_sdv
Continuous Listening	patgen cntlstn memory	cl\cntlstn Patgen\patgen Patgen\block Patgen\digagc Patgen\stream Patgen\nulogfns Patgen\normal	PatgenHandler ³ PatgenMemDriver ³ cl_busy ³ cl_ready ³ cl_done ³
Voice Record and Play	rp memory \lib\xram_rp memory\xram	rp\rec rp\play rp\post rp\cmp ⁴ rp\mem_rp rp\3bit2bit ⁴ rp\rp3bit ⁴ rp\rp2bit ⁴	stop_rp_play ³ stop_rp_record ³ init_xread_rp init_xwrite_rp exit_xread_rp exit_xwrite_rp get_byte_rp put_byte_rp get_byte_tmp set_tmpptr_getptr set_getptr_tmpptr set_putptr_putend add_getptr sub_getptr add_tmpptr cp_tmpptr_getptr

Technology Used	Include (\lib\macros\)	Link .O (\lib\)	Co-routines Required¹
Dual Recognition	fastdig memory si sd memory\xram	drt\drt (also all SI, SD)	(all SI, SD)
Speech Synthesis	talk memory	talk\talk talk\talktbl	TalkHandler ³ TalkMemDriver ³
Sentence Talk		talk\sentalk ⁵	SenTalkMemDriver ³
Debug Speech	debug_s ⁶	debug_s\debug1 ⁶ \data\speech\numbersd ⁶	
Touch Tone Synthesis	ttone memory	ttone\ttone	TToneHandler ³
RS-232	rs232 debug_s ⁸	\lib\rs232\rs232 \lib\rs232\debugpc\debug1 ⁸	

Notes:

1. User must supply the indicated co-routines or link in defaults. See *Using Co-Routines for Software Development*
2. Only needed if using words-on-chip
3. The default interrupt handler routine is located in the same \lib or \lib364 directory as the code and usually has the name xxx_h.o (e.g. recog_h.o or talk_h.o)
4. Need depends on compression level
5. Only needed if using Sentence Talk
6. Only needed if using Debug Speech
7. Only need if adjusting priors
8. Only needed if using "debug speech" output via RS-232.

CHAPTER 4 - HARDWARE

This chapter contains the schematics and parts locator drawings for the DK264T/364 Motherboard, DK264T/364 Memory Module, DK264T/364 IO Module, and Dialer Keypad. These reference documents are provided to allow the developer to readily interface to the Development Modules and to perform hardware and software debugging.

It also provides the connector pin specifications on for each of the Development Kit boards.

1. DK264T/364 Motherboard Schematic

DK264T/364 Motherboard Schematic (Check the inside front cover of this manual for this drawing).

2. DK264T/364 Motherboard Parts Locator

2. DK264T/364 Motherboard Parts Locator

3. DK264T/364 Memory Module Schematic

DK264T/364 Memory Module Schematic 1

(Check the inside the front cover of this manual for this drawing).

4. DK264T/364 Memory Module Schematic-2

DK264T/364 Memory Module Schematic-2

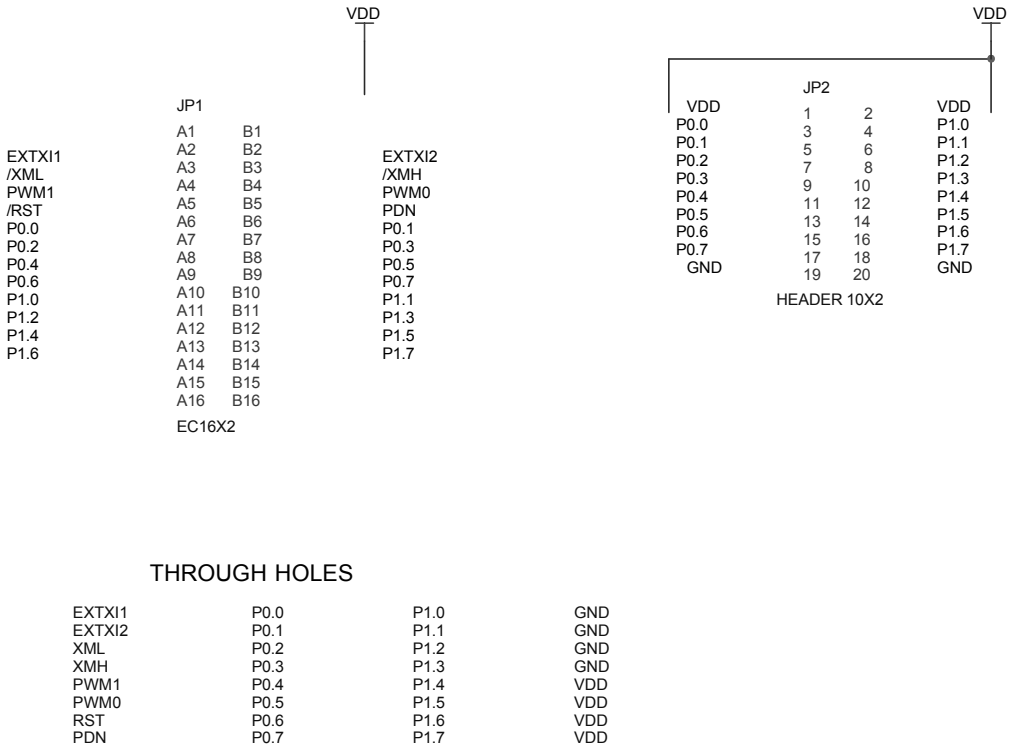
(Check the inside front cover of this manual for this drawing).

5. K264T/364 Memory Module Parts Locator

K264T/364 Memory Module Parts Locator

6. DK264T/364 I/O Module Schematic-2

6. DK264T/364 I/O Module Schematic-2



Sensory, Inc.

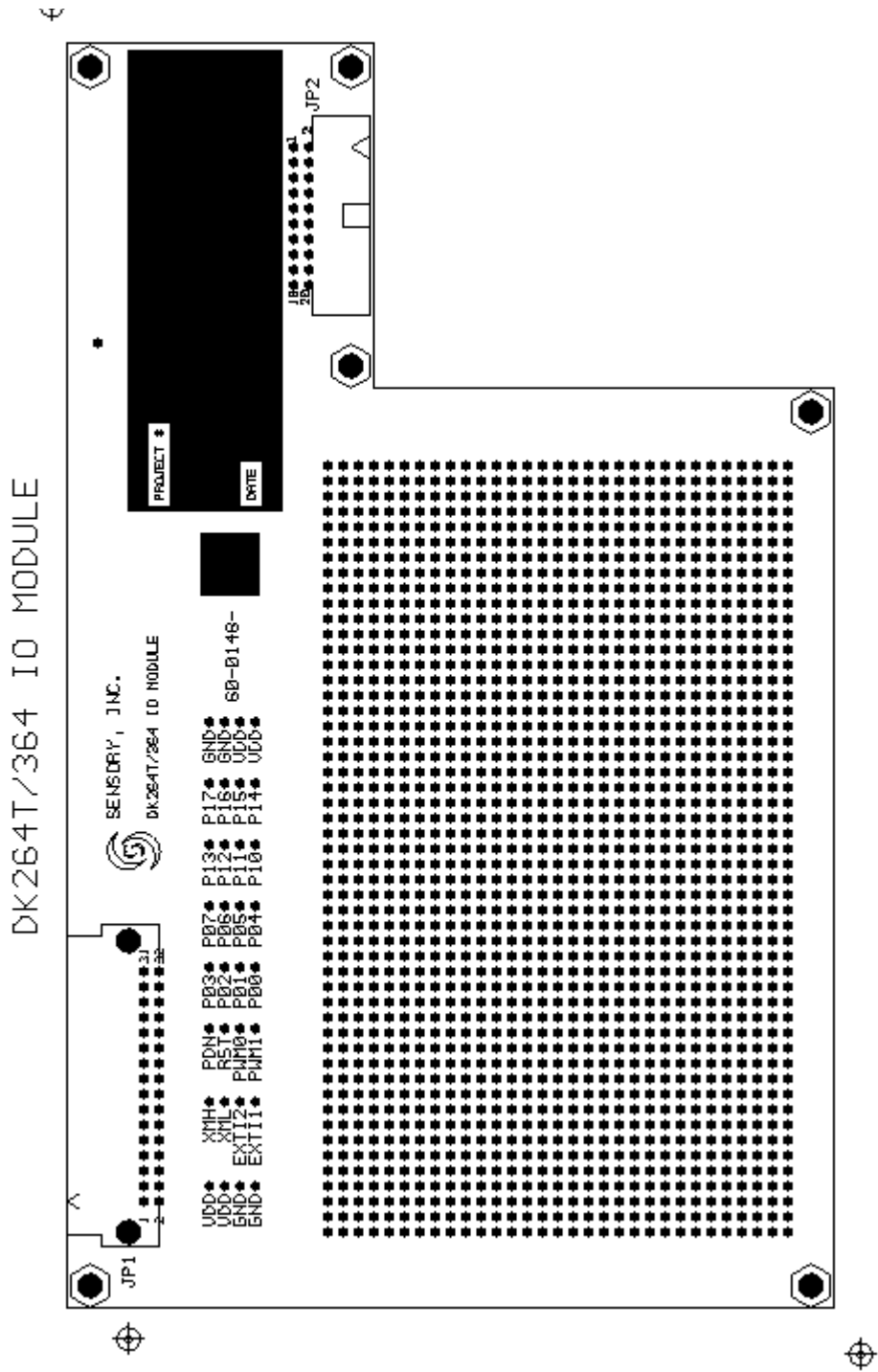
Title
DK264T/364 IO Module

Size
A Document Number
70-0043

Date: Thursday, January 07, 1999

7. DK264T/364 I/O Module Parts Locator

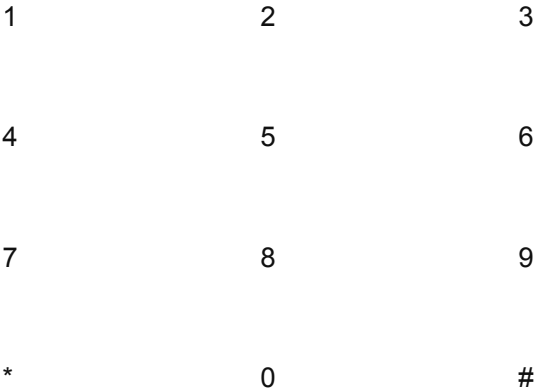
DK264T/364 I/O Module Parts Locator



8. Dialer Keypad Schematic

Dialer Keypad Schematic

KEYPAD



KEYPAD BOX



Ribbon Cable

IDE 20 pin Housing: Bottom View

Ribbon
Cable

1
2
3
4
5
6
7

VDD 1
VDD 2
P0.0 3
P1.0 4
P0.1 5
P1.1 6
P0.2 7
P1.2 8
P0.3 9
P1.3 10
P0.4 11
P1.4 12
P0.5 13
P1.5 14
P0.6 15
P1.6 16
P0.7 17
P1.7 18
GND 19
GND 20

IDE 20 Pin Housing

TO DK264T/364 IO MODULE

Sensory, Inc.
521 Weddell Drive
Sunnyvale, CA 94089-2164
Copyright(C) 1996 Sensory, Inc. All Rights Reserved

Title
Dialer Sample Keypad
Size
A Document Number
70-0017

Date: Monday, December 14, 1998 Sheet 1 of 1

Rev
B

9. Connector Pin Specification Summary

A. DK264T/364 Motherboard

Table 1: JP1: Memory Module Interface

Pin Number	Description	Pin Number	Description
A1	GND	B1	GND
A2	VDD	B2	VDD
A3	D0	B3	D1
A4	D2	B4	D3
A5	D4	B5	D5
A6	D6	B6	D7
A7	A0	B7	A1
A8	A2	B8	A3
A9	A4	B9	A5
A10	A6	B10	A7
A11	A8	B11	A9
A12	A10	B12	A11
A13	A12	B13	A13
A14	A14	B14	A15
A15	/WRD	B15	/RDD
A16	/WRC	B16	/RDC
A17	GND	B17	GND
A18	GND	B18	GND
A19	GND	B19	GND
A20	/XML	B20	/XMH
A21	TE1	B21	TE2
A22	/RST	B22	PDN
A23	P0.0	B23	P0.1
A24	P0.2	B24	P0.3
A25	P0.4	B25	P0.5
A26	P0.6	B26	P0.7
A27	P1.0	B27	P1.1
A28	P1.2	B28	P1.3
A29	P1.4	B29	P1.5
A30	P1.6	B30	P1.7
A31	VDD	B31	VDD
A32	GND	B32	GND

Table 2: JP2: I/O Module Interface

Pin Number	Description	Pin Number	Description
A1	GND	B1	GND
A2	VDD	B2	VDD
A3	EXTXI1 (unconnected)	B3	EXTXI2 (unconnected)
A4	/XML	B4	/XMH
A5	TE1	B5	TE2
A6	/RST	B6	PDN
A7	P0.0	B7	P0.1
A8	P0.2	B8	P0.3
A9	P0.4	B9	P0.5
A10	P0.6	B10	P0.7
A11	P1.0	B11	P1.1
A12	P1.2	B12	P1.3
A13	P1.4	B13	P1.5
A14	P1.6	B14	P1.7
A15	VDD	B15	VDD
A16	GND	B16	GND

Table 3: P1: RS232 Interface (DB9 Female)

Pin Number	Name	Description
1	CD	NC
2	RXD	Received Data (DK264T motherboard's output)
3	TXD	Transmitted Data (DK264T motherboard's input)
4	DTR	NC
5	GND	Signal GND
6	DSR	NC
7	RTS	NC
8	CTS	NC
9	RI	NC

Note: At P1 connector,
 - DTR connected to DSR
 - RTS connected to CTS

J4: Power Input (Right Angle 2-pin Header)

Pin Number	Description	Pin Number	Description
1	Power	2	GND

J1: Microphone Input (Phone Jack)

J2: Audio Output: PWM (Phone Jack)

J3: Audio Output: Speaker (Phone Jack)

Amplified DAC output

J5: Audio Output: DAC (Straight 2-pin Header)

Un-amplified DAC output

B. DK264T/364 Memory Module

Table 4: JP1: Memory Module Interface

Pin Number	Description	Pin Number	Description
A1	GND	B1	GND
A2	VDD	B2	VDD
A3	D0	B3	D1
A4	D2	B4	D3
A5	D4	B5	D5
A6	D6	B6	D7
A7	A0	B7	A1
A8	A2	B8	A3
A9	A4	B9	A5
A10	A6	B10	A7
A11	A8	B11	A9
A12	A10	B12	A11
A13	A12	B13	A13
A14	A14	B14	A15
A15	/WRD	B15	/RDD
A16	/WRC	B16	/RDC
A17	GND	B17	GND
A18	GND	B18	GND
A19	GND	B19	GND
A20	/XML	B20	/XMH
A21	TE1	B21	TE2
A22	/RST	B22	PDN
A23	P0.0	B23	P0.1
A24	P0.2	B24	P0.3
A25	P0.4	B25	P0.5
A26	P0.6	B26	P0.7
A27	P1.0	B27	P1.1
A28	P1.2	B28	P1.3
A29	P1.4	B29	P1.5
A30	P1.6	B30	P1.7
A31	VDD	B31	VDD
A32	GND	B32	GND

C. DK264T/364 IO Module

Table 5: JP1: Motherboard Interface

Pin Number	Description	Pin Number	Description
A1	GND	B1	GND
A2	VDD	B2	VDD
A3	EXTXI1	B3	EXTXI2
A4	/XML	B4	/XMH
A5	TE1	B5	TE2
A6	/RST	B6	PDN
A7	P0.0	B7	P0.1
A8	P0.2	B8	P0.3
A9	P0.4	B9	P0.5
A10	P0.6	B10	P0.7
A11	P1.0	B11	P1.1
A12	P1.2	B12	P1.3
A13	P1.4	B13	P1.5
A14	P1.6	B14	P1.7
A15	VDD	B15	VDD
A16	GND	B16	GND

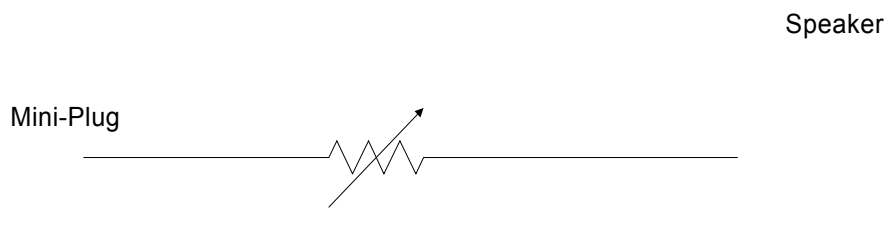
Table 6: JP2: Keypad Interface

Pin Number	Description	Pin Number	Description
A1	GND	B1	GND
A2	VDD	B2	VDD
A3	EXTXI1	B3	EXTXI2
A4	/XML	B4	/XMH
A5	TE1	B5	TE2
A6	/RST	B6	PDN
A7	P0.0	B7	P0.1
A8	P0.2	B8	P0.3
A9	P0.4	B9	P0.5
A10	P0.6	B10	P0.7
A11	P1.0	B11	P1.1
A12	P1.2	B12	P1.3
A13	P1.4	B13	P1.5
A14	P1.6	B14	P1.7
A15	VDD	B15	VDD
A16	GND	B16	GND

10. Adding Volume Control

Adding a volume control to the PWM output of the RSC-264T/364 Development Kit Motherboard can be easily accomplished with the following procedure:

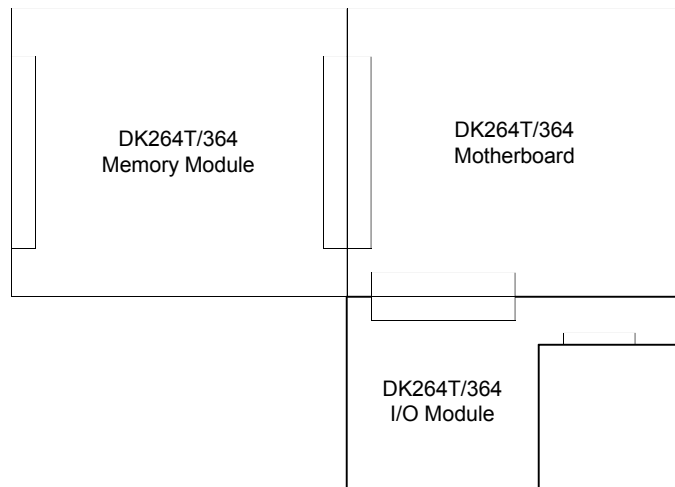
- Disconnect the speaker cord from the mini-plug jack on the Motherboard.
- Cut into one side of the cord and insert a 200 Ohm potentiometer in series with the wire.



11. Hardware Configuration

DK264T/364 Development System consists of the three main modules:

- DK264T/364 Motherboard
- DK264T/364 Memory Module
- DK264T/364 I/O Module



A. DK264T/364 Motherboard

- **Power Supply**

There are three settings for the power supply voltage.

5V

Input: 5.3 ~ 9VDC (regulated or unregulated power supply)

Select 5V for the on-board voltage regulator

Adjustable (2.7V ~ 5V)

Input: 3.0 ~ 5.3VDC (regulated power supply)

Select 5V for the on-board voltage regulator

3V

Input: 3.3 ~ 9V DC (regulated or unregulated power supply)

Select 3V for the on-board voltage regulator

- To select 5V, place the shorting block on "5V" at JB1
- To select 3V, place the shorting block on "3V" at JB1

CAUTION: Ensure that the memory IC's added to the Memory Module are rated at the correct voltage.

Oscillator-1 and Oscillator-2

There are two different modes of operation for Oscillator-1 and two for Oscillator-2 as shown in the table below.

	External Clock Inputs	On-board Oscillators (Crystal)
Oscillator-1	Yes (14.318MHz)	Yes (14.318MHz)
Oscillator-2	Yes	Yes (<i>Note: Oscillator-2 components are not populated on the board.</i>)

External Clock Inputs On-board Oscillators (Crystal)

Oscillator-1

- The Motherboard is shipped with the 14.318MHz crystal installed.
- To use an external clock instead of the preinstalled crystal, remove C21 and Y1
 - (a) To connect an external oscillator to the Motherboard:
- Solder a jumper wire from the clock circuit's output to the XI1 through-hole located below R15.
- Solder a jumper wire from the clock circuit's ground to the GND through-hole located below XI1.
 - (b) To bring in a clock signal built on the I/O module, solder a jumper wire from the XI1 through-hole located below R15 to the XI1 through-hole located near R22.

Oscillator-2

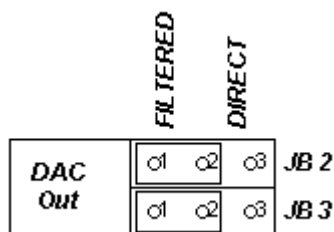
- The Motherboard is shipped without crystal Y2 installed.
- To use the on-board oscillator, install Y2, C19 (68pF). C18 can be omitted.
- To use the external clock.
 - (a) To connect an external oscillator to the Motherboard:
- Solder a jumper wire from the clock circuit's output to the XI2 through-hole located below R28.
- Solder a jumper wire from the clock circuit's ground to the GND through-hole located below XI2.
 - (b) To bring in a clock signal built on the I/O module, solder a jumper wire from the XI2 through-hole located below R28 to the XI2 through-hole located near R22.

DAC Output

The un-buffered DAC output (before LM4862) can be tapped at J5.

To tap the DAC output after the filter,

place the shorting blocks at FILTERED on JB-2 and JB-3.



To tap the DAC output directly from the RSC chip, place the shorting blocks at DIRECT on JB-2 and

JB-3.

Note: When DIRECT is selected, there will be no sound at the amplified DAC output at J3 (labeled SPEAKER OUT).

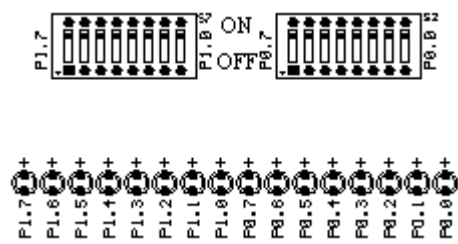
SW-A, SW-B, and SW-C Push Buttons

The connections for the SW-A, SW-B, and SW-C push buttons have to be made in the wiring area on the motherboard. Each switch can be connected to any of GPIO (P0.0 ~ P1.7) by connecting wires from JP4 to JP3 and JP4 to JP5. In some revisions, SW-A, SW-B, and SW-C at JP4 are labeled as '1', '2', and '3' respectively.

LEDs and DIP Switches

To make connections from GPIO to LEDs, place the DIP switches in the ON positions.

For example, to connect P0.0 output to the P0.0 LED, set the right-most DIP switch setting of S2 to the ON position.



Note: Although all GPIO ports are buffered with 74HC240s to drive the LEDs, when the connections are made with the DIP switch settings, the ports are pulled up to Vdd with 100K resistors.

RS232 Interface

There are three configuration settings for the RS232 Interface.

To enable the RS232 Interface IC (U3) all the time:

	Disconnect	Connect
JB4		X
JB5		X
JB6		X

JB7		X
	P1.7-RS232EN	Manual-RS232EN

To control the RS232 Interface IC (U3) by software:

Note: P1.7 will be used to disable or enable U3.

	Disconnect	Connect
JB4		X
JB5		X
JB6	X	

JB7	X	
	P1.7-RS232EN	Manual-RS232EN

To disable the RS232 Interface IC (U3) all the time:

	Disconnect	Connect
JB4	X	
JB5	X	
JB6	X	

JB7		X
	P1.7-RS232EN	Manual-RS232EN

B. DK264T/364 Memory Module

Memory Types

There are 7 memory ICs in Memory Module board as shown in the table below.

DIP	Accepts 28 and 32 pin. 512Kbit to 4Mbit. Note: The DIP socket is useful for an EPROM emulator.
PLCC-A	Accepts 32 pin. 512Kbit to 4Mbit.
PLCC-B	Accepts 32 pin. 512Kbit to 4Mbit.
TSOP-A	Foot-print for the 32 pin memory IC for TSOP (Unpopulated)
TSOP-B	M29W400T, 4Mbit
SRAM-A	TC55257DFTL, 32Kbit
Serial EEPROM	24C65, 64Kbit

Basic Memory Configurations

The four basic memory configurations can be selected by setting jumpers JB2, JB3, JB4, and JB6. Their descriptions are listed in the table below:

JB 4	PC Download
JB 6	DEMO: PCCC-B
JB 2	CODE: DIP
JB 3	CODE: PLCC-A

Label on PCA	Code	Data	Steps	Application Examples
PC DOWNLOAD	TSOP-B (4Mbit Flash) PLCC-A (2Mbit OTP)	PLCC-B (2Mbit Flash)	To run program: Place the 4Mbit Flash into TSOP-B. Place the 2Mbit OTP programmed with boot loader (60-0176) into PLCC-A. Place the 2Mbit Flash into PLCC-B. Select the position of Rotary SW for the program to run. Turn on the power. To download the programs into the 4Mbit flash from a PC, refer to the boot loader instructions.	Sample Programs, "Bootload" development configuration
DEMO: PLCC-B	PLCC -B (2Mbit OTP)	SRAM-A (256Kbit)	To run program: Place programmed 2Mbit OTP (60-0166) into PLCC-B. Select the position of Rotary SW for the program to run. Turn on the power.	Not currently used
CODE: DIP	DIP (512Kbit)		To run program: Place programmed OTP, EPROM, or emulator into DIP. Turn on the power Selecting 28/32 Pin: Placed the shorting block for the 28-pin DIP.	ROM emulator development configuration
CODE: PLCC	(PLCC-A)		Similar to DIP. Note: Address bits 16 and 17 will be forced to low, so only the lowest 64Kbytes can be accessed.	

Custom Memory Configuration

The custom memory configuration provides flexibility for connecting a wide range of memory ICs. For custom memory devices and configurations, the following features have been provided:

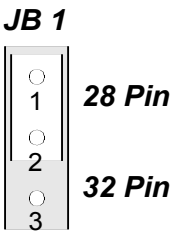
- Through-holes for the memory ICs data and address lines, memory control lines, and RSC-364 GPIO ports.
- Three 14-pin DIP sockets for the standard 14-pin logic ICs (74HC04, 74HC00, etc.) with connections for the power and ground.
- Two sets of through holes for a 40-pin wide DIP IC (power and ground are not connected).
- Prototyping area

32-pin DIP (ZIF) Socket

JB1 must be selected for a 28-pin or a 32-pin IC. If the shorting block is not inserted, the memory device will not be powered.

64K Bits Serial EEPROM

SCL and SDA for the serial EEPROM can be connected to any GPIO line by connecting wires between JP3, JP5 and JP4.



C. Sample and Demo Programs

The Development Kit is supplied pre-programmed with the sample programs. A different set of "demo" programs can be loaded using the Bootloader. The demo programs are more feature-rich and less tutorial than the sample programs. Complete source code for the sample programs is supplied.

Sample Programs

Eight sample programs are provided with the Development Kit and are run from the 4Mbit Flash in the TSOP-B (U5) socket. To select a sample program, set the dial of the PROGRAM SELECT rotary switch (S2), as shown in the following table:

Rotary Switch	Demo Program	Description	Address	
7	Music	Three song music sample.	Start:	00000
			End:	0FFFF
6	Password	Speaker Verification, words-on-chip, Sleep with IO wakeup	Start:	10000
			End:	1FFFF
5	Record and Play	Voice record and playback demo with 2-bit and 3-bit compression.	Start:	20000
			End:	2FFFF
4	Wordspot	Speaker Dependent recognition of words within fluent speech	Start:	30000
			End:	3FFFF
3	Fast Digits	Telephonic fast digits demonstration using Dual Recognition and Speaker Adaptive Technology.	Start:	40000
			End:	4FFFF
2	Speaker-Verification	Speaker verification recognition using queuing for one to four words.	Start:	50000
			End:	5FFFF
1	Speaker-Dependent	Up to 32-word speaker dependent recognition with simultaneous pattern generation and nametag recording.	Start:	60000
			End:	6FFFF
0	Speaker-Independent	Speaker independent demo featuring the SI6 and Yes/No weightsets.	Start:	70000

Demo Programs

Eight demo programs are provided with the Development Kit. These files can be loaded and run from the 4Mbit Flash in the TSOP-B (U5) socket. See separate descriptions of these programs.

12. Installing and Removing the Sample Flash

Some configurations require installing or removing the Sample Flash in location TSOP-B. This operation must be done with care, as the leads of the TSOP package are delicate. Follow the procedure described below to install the part.

1. Work at an ESD-safe workstation. Memory devices are especially susceptible to ESD.
2. Make sure the power is turned off before inserting any new IC into the Development Kit.
3. Place the flash memory (which is either a Toshiba TC58F400T/B or an SGS-Thompson M29W400T) into the TSOP-B socket with pin 1 closest to C5. The plastic portion of the socket contains a well that receives the body of the IC, and the leads rest on top of the socket pins. When installed correctly, the chip will rest snugly in the well and the text on the chip will be upside-down compared with the text silk-screened on the developer kit Memory Module.
4. Note that the small aluminum door is shaped somewhat like a sled, with one pair of smooth tabs and one pair of serrated tabs. When properly installed, all four tabs will be captured by slots in

the socket. The edge of the door near the serrated tabs has a small notch.

5. Insert the serrated tabs into the slots on the socket and flip the door down to cover the Sample Flash, taking care not to disturb the position of the IC.
6. While pressing down firmly on the sides of the door near the smooth tabs, slide the door closed. It may help to use a small screwdriver to push against the notch or the square in the middle of the door while you press down on the sides to close it. Note that the small tabs in the aluminum door will slide through slots in the socket and the door will "click" into place.
7. Install jumper block JB1:BOOT_LOAD. Other jumpers may also be installed, depending on the configuration.

CAUTION: make sure the Sample Flash voltage rating (3V or 5V) is the same as the voltage setting of the board before applying power.

To remove the Sample Flash, follow the safety steps above and reverse the procedure. To release the door, insert a small screwdriver vertically between the aluminum door and the socket body and lever the door towards the serrated tabs until the smooth tabs are free.

CHAPTER 5 - SIMULATOR

This chapter provides a description of the simulator software and user interface for the RSC-264T and the RSC-364. The simulator is a software only program that allows a personal computer to run a simulation of the RSC-264T/364 processor. Although some internal details of these two chips differ significantly, the portions simulated in software are similar, so this description applies to both chips. No external hardware (other than a PC) is required to operate the simulator. The chapter begins with a description of how to start the simulator and then proceeds with a description of the various user interface windows. This chapter also contains a simulator tutorial, which will familiarize the user with the simulator functions.

1. Introduction

This is a description of the simulator in the RSC-264T/364 Development Kit. It assumes the reader is familiar with the architecture of the RSC-264T/364 chip, and has the appropriate Data Book available for detailed descriptions of the chip's features and functions.

The simulator is a software executable program that allows an IBM compatible personal computer to operate on RSC-264T/364 binary code to perform a software simulation of the RSC-264T/364 processor. The simulator does not support simulation of the specialized on-chip hardware needed for speech synthesis or speech recognition. Therefore, the simulator cannot be used to check these sections of code in a program. No external hardware is required to run in simulator mode.

User Equipment Requirements

The simulator requires an IBM compatible PC running Windows 3.1 or version 3.11 (Windows for Workgroups), Windows 95, or Windows 98.

Software Description

The simulator program resides in a single executable called simulate.exe.

2. Starting the Simulator

To start the simulator, run the simulate.exe program (located in the \BIN directory). The display window contains a Menu bar, a dialog box called Control Window and a Source Code List window that shows the current code space (see Figure 1). The figure also shows two optional windows, Display SFR RAM and Scratchpad RAM. These are described below. The actual display may be different depending on screen resolution, video card and driver, and the Windows operating system version.

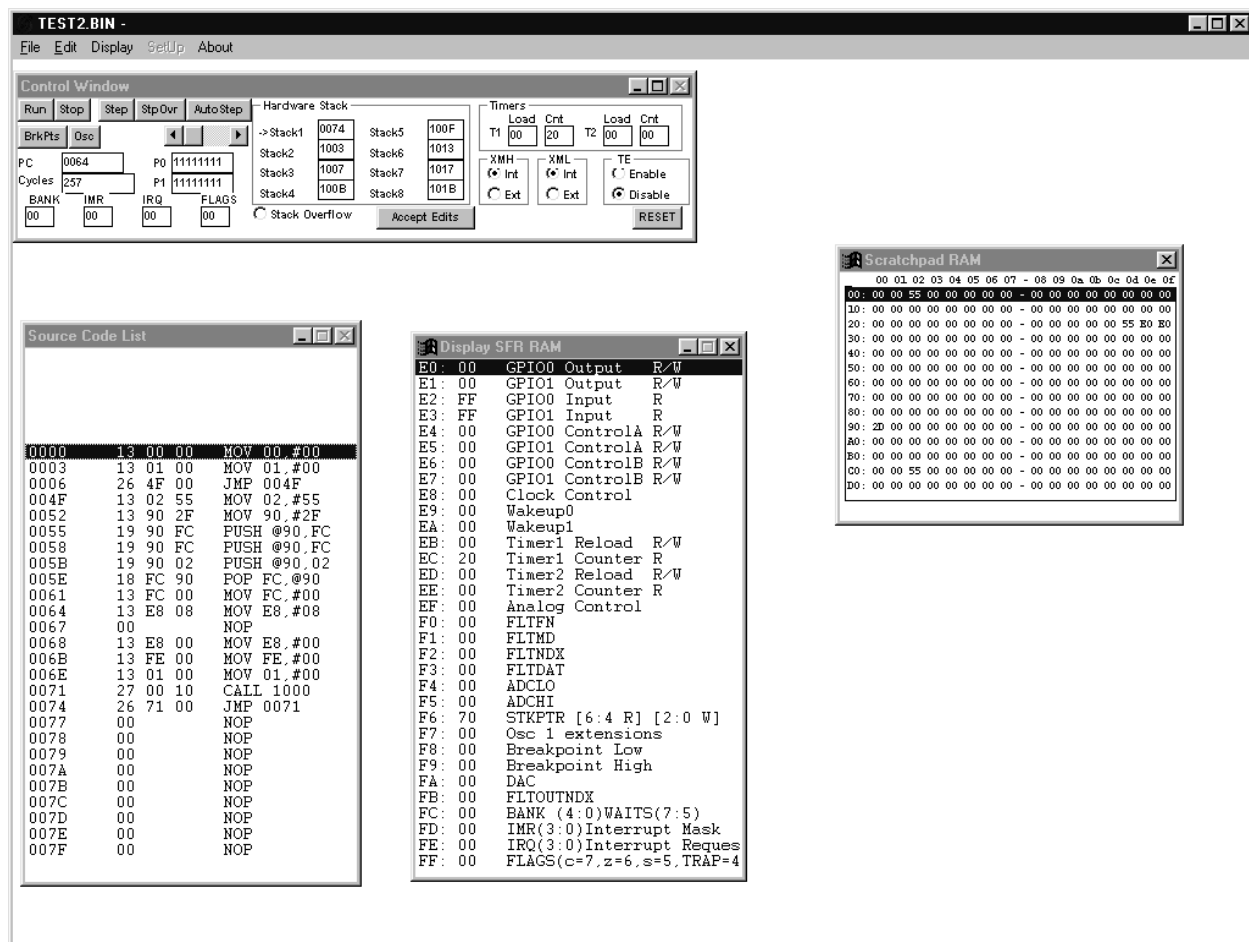


Figure 1 Simulator Display

3. Simulator: Menu Bar

The **Menu** bar has the following menu items: **File Edit Display SetUp About**

Each of the menu items is briefly described below.

File:

Load:

Internal Code

External Code

External Data

Exit

Loads a file from disk to internal code space.

Loads a file from disk to external code space.

Loads a file from disk to external data space.

Exits the simulator and returns to the PC operating system.

Edit:

The edit feature is disabled in this version of the simulator.

Display:

All Bank RAM	Loads and displays a window showing RAM contents 100h-1FFh (Banks 8-F).
SFR RAM	Loads and displays a window showing RAM contents E0h-FFh. These "Special Function Registers" are not actually RAM locations, but they map to RAM addresses.
ScratchPad RAM*	Loads and displays a window showing RAM contents 0h-DFh. The data from C0h-DFh show the contents of the currently selected bank.
Internal CODE Space	Loads and displays a window showing internal code space.
External CODE Space	Loads and displays a window showing external code space.
External DATA Space	Loads and displays a window showing external data space.

* The ScratchPad RAM is the on-chip RAM referenced in the Data Book as Register Space.

Note: The Display menu items are explained in more detail in the Using the Display Menu subsection that follows. This subsection also describes the editing options available for several of the menu items.

SetUp: This menu item is disabled in the current version.

About:

Names This brings up a box showing the Sensory, Inc. copyright notice, programmers' credits and the software version.

Using the Display Menu

This section provides more information on how to display register (on-chip RAM) and code contents and edit register contents for the appropriate menu items shown below.

Display:**All Bank RAM**

The All Bank RAM section of the on-chip RAM is divided into 32-byte banks which are referred to as Bank8 through BankF (see Figure 2). This menu item loads and displays a window showing the on-chip RAM contents 100h-1FFh (see Figure 2). If necessary, re-size the window to display the bank of interest. Double clicking on a line of register contents brings up an edit dialog box that allows the contents of the registers to be modified.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
BANK8:	48	2C	41	43	D2	01	93	1C	61	2C	01	43	7E	01	00	1C	hex
BANK8:	AC	73	50	40	A0	AF	12	04	A0	07	C0	CB	00	BC	E3	27	hex
BANK9:	E0	40	C4	49	D0	14	CD	05	C3	EC	36	3D	3C	72	C0	90	hex
BANK9:	30	8B	02	22	C5	66	22	1C	A3	88	11	78	E1	C1	20	53	hex
BANKA:	50	A0	09	08	0C	04	4C	ED	90	4A	11	52	0C	A0	76	98	hex
BANKA:	9F	D4	06	08	0A	05	04	90	C0	C6	E4	A3	40	88	35	F0	hex
BANKB:	40	C0	6C	69	15	E9	DC	2A	60	A5	11	07	C7	4F	1C	02	hex
BANKB:	57	C6	0F	24	21	3D	BB	8F	8E	DE	D0	C8	A8	EE	2E	9A	hex
BANKC:	44	50	E0	36	10	13	E0	CD	2F	41	CC	81	4B	A1	28	97	hex
BANKC:	9C	D3	3C	4E	0C	64	A3	3C	85	41	CE	02	42	14	C1	93	hex
BANKD:	41	95	22	3F	98	AD	CD	1D	E1	44	49	E0	69	18	8B	05	hex
BANKD:	C9	14	8F	C8	14	10	28	7A	07	84	86	13	20	80	C8	A0	hex

Edit All Bank RAM																
48	2C	41	43	D2	01	93	1C	61	2C	01	43	7E	01	00	1C	
100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F	
BANK8: Ram Address 100																OK
																Cancel

Figure 2
All Bank Ram window and edit dialog box

Display: SFR RAM

This loads and displays a window showing on-chip RAM contents E0h-FFh. The window must be re-sized by dragging the borders with the mouse to display all the SFR's in one view (see Figure 3). Double clicking on a line of register contents brings up an edit dialog box that allows the contents of the registers to be modified. If the vertical scroll bar is used to bring a SFR into view and then a Run to Breakpoint or Step executes an instruction which affects the SFRs, the display is reset to the top of the SFRs. If a SFR has been selected (single click) then the display does not shift position when the SFRs are affected.

Display SFR RAM		
E0: 00 hex	GPI00 Output	R/W
E1: 00 hex	GPI01 Output	R/W
E2: FF hex	GPI00 Input	R
E3: FF hex	GPI01 Input	R
E4: 00 hex	GPI00 ControlA	R/W
E5: 00 hex	GPI01 ControlA	R/W
E6: 00 hex	GPI00 ControlB	R/W
E7: 00 hex	GPI01 ControlB	R/W
E8: 00 hex	Clock Control	
E9: 00 hex	Wakeup0	
EA: 00 hex	Wakeup1	
EB: 00 hex	Timer1 Reload	Register
EC: 00 hex	Timer1 Counter	Register
ED: 00 hex	Timer2 Reload	Register
EE: 00 hex	Timer2 Counter	Register
EF: 00 hex	Analog Control	Register
F0: 00 hex	Reserved	
F1: 00 hex	Reserved	
F2: 00 hex	Reserved	
F3: 00 hex	Reserved	
F4: 00 hex	Reserved	
F5: 00 hex	Reserved	
F6: 00 hex	Reserved	
F7: 00 hex	Reserved	
F8: 00 hex	Reserved	
F9: 00 hex	Reserved	
FA: 00 hex	Reserved	
FB: 00 hex	Reserved	
FC: E0 hex	BANK (4:0)WAITS(7:5)	
FD: 00 hex	IMR(3:0)Interrupt Mask Reg	
FE: 00 hex	IRQ(3:0)Interrupt Request	
FF: 00 hex	FLAGS(c=7,z=6,s=5,TRAP=4,I	

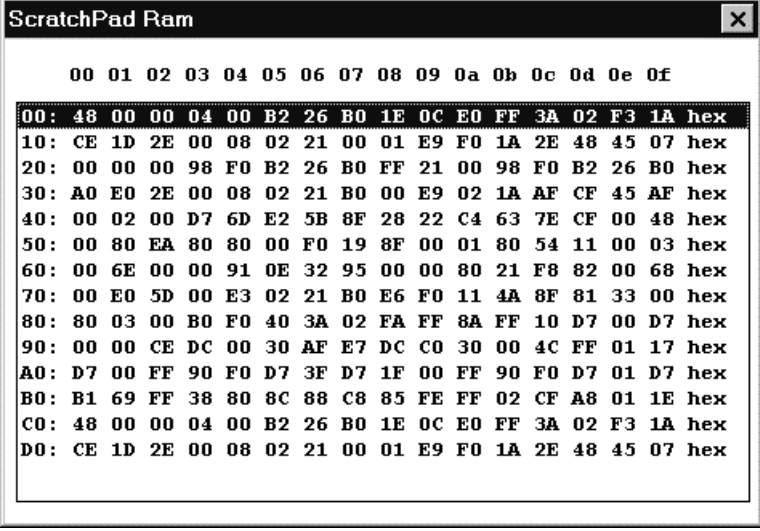
Edit SFR List	
00	OK
EF	Cancel

Figure 3
Display SFR RAM window and edit dialog box

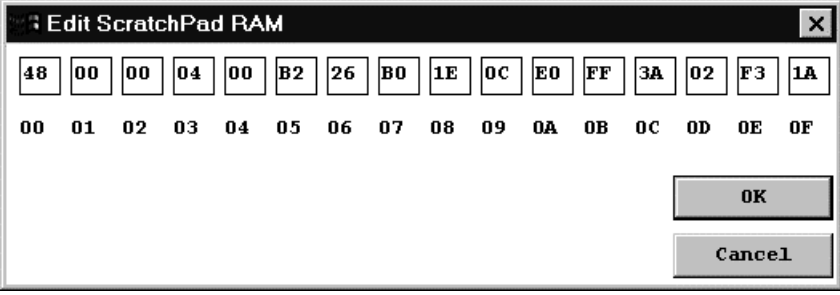
Display:

ScratchPad RAM

This loads and displays a window showing on-chip RAM contents 0h-DFh (see Figure 4). Double clicking on a line of register contents brings up an edit dialog box that allows the contents of the registers to be modified.



	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00:	48	00	00	04	00	B2	26	B0	1E	0C	E0	FF	3A	02	F3	1A hex
10:	CE	1D	2E	00	08	02	21	00	01	E9	F0	1A	2E	48	45	07 hex
20:	00	00	00	98	F0	B2	26	B0	FF	21	00	98	F0	B2	26	B0 hex
30:	A0	E0	2E	00	08	02	21	B0	00	E9	02	1A	AF	CF	45	AF hex
40:	00	02	00	D7	6D	E2	5B	8F	28	22	C4	63	7E	CF	00	48 hex
50:	00	80	EA	80	80	00	F0	19	8F	00	01	80	54	11	00	03 hex
60:	00	6E	00	00	91	0E	32	95	00	00	80	21	F8	82	00	68 hex
70:	00	E0	5D	00	E3	02	21	B0	E6	F0	11	4A	8F	81	33	00 hex
80:	80	03	00	B0	F0	40	3A	02	FA	FF	8A	FF	10	D7	00	D7 hex
90:	00	00	CE	DC	00	30	AF	E7	DC	C0	30	00	4C	FF	01	17 hex
A0:	D7	00	FF	90	F0	D7	3F	D7	1F	00	FF	90	F0	D7	01	D7 hex
B0:	B1	69	FF	38	80	8C	88	C8	85	FE	FF	02	CF	A8	01	1E hex
C0:	48	00	00	04	00	B2	26	B0	1E	0C	E0	FF	3A	02	F3	1A hex
D0:	CE	1D	2E	00	08	02	21	00	01	E9	F0	1A	2E	48	45	07 hex



48	00	00	04	00	B2	26	B0	1E	0C	E0	FF	3A	02	F3	1A
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

OK

Cancel

Figure 4
Scratch Pad RAM window and edit dialog box

Internal CODE Space, External CODE Space, and External DATA Space

The code and data space windows all operate in essentially the same manner. They each initially display a section of code or data space beginning at address 0000h (see Figure 5). The 64K code or data space is divided into 32 blocks for display; each block is 2048 bytes in length. The window size can be increased

by selecting the window edge with the mouse and dragging it. Use the vertical scroll bar to display different areas within this window. If the end of the list box is encountered the NEXT BLOCK line is displayed. Double click on the NEXT BLOCK line to open an edit box that allows the beginning address of the next list area to be selected. The user is allowed to enter more than four digits, however only the four LSBs are used. Double clicking on a line of the window contents brings up an edit dialog box that allows the contents of three code or data space address locations to be modified.

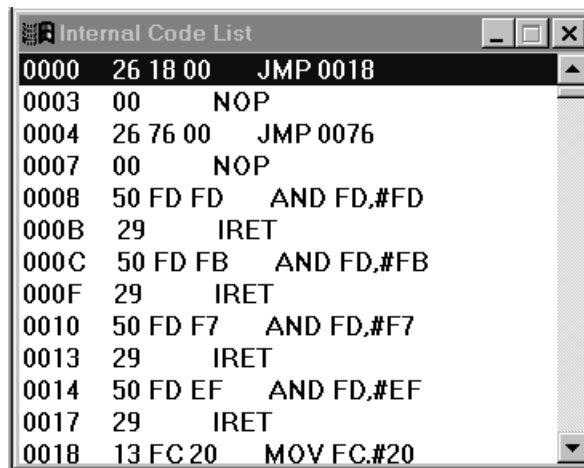


Figure 5
Internal Code List window

4. Simulator: Control Window

This dialog box is the main user interface for control of the simulator. It is divided into several sections with related controls that include: Simulator Command Buttons, PC and Oscillator Controls, Hardware Stack Display, P0 and P1 Display, Timers Display, and BANK, IMR, IRQ, FLAGS Display.

Simulator Command Buttons

The command buttons are the controls in the shaded boxes and they perform the following functions.

Run

This causes a run-to-breakpoint command. If no breakpoint is set or none encountered, the simulator will run until the user intervenes. Run mode is indicated by the button flashing.

Stop

This causes a stop command which results in the previous command being terminated.

RST (Reset)

This causes a processor reset, resulting in the PC, and cycle count being set to zero.

Step

This causes a single step to be executed. Note that the Source Code List window displays the step history.

StpOvr (StepOver)

This causes a run to the next return instruction (RET) up to the current level of subroutines to be executed.

Example: Executing the following code from address 0000h by using the StpOvr button will result in the simulator stopping at address 0004h.

```
0000 00 nop
0001 27 CALL 000A ;calls a subroutine at address 000Ah
0004 00 nop ;returns to here and stops simulation
0005 00 nop
0006 00 nop
0007 00 nop
0008 00 nop
0009 00 nop
000A 00 nop ;subroutine starts here
000B 28 RET ;and returns from here to address 0004h
000C 00 nop
```

AutoStep and Slider Control

These two controls allow multiple single step operations to be executed. The Slider controls the speed at which the steps are executed. Autostep stops if a breakpoint is reached.

BrkPts (Breakpoints)

This button brings up a dialog box that allows up to four breakpoints to be set. It is initialized to one breakpoint at 0000h so that processing stops at the beginning address to indicate a software restart or wrap-around from address FFFFh. The dialog box commands are shown in Figure 6.

The components of this dialog box are: four breakpoint boxes that allow the user to enter four hex digits for each breakpoint, an OK button that closes the dialog box after entries have been made, and a Cancel button which closes the dialog box.

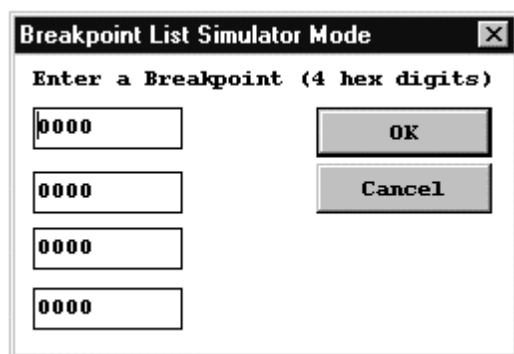


Figure 6
Breakpoints List Simulator Mode Dialog Box Controls

Osc

This button is described in the PC and Oscillator Controls section (see following page).

Accept Edits

This button acts as the default button to accept edits in the edits fields.

Simulator Mode Controls (radio buttons)

The mode controls consist of the XML and XMH radio buttons and the TE Enable and Disable radio buttons.

The two **XML** and two **XMH** radio buttons control the code space from which the simulator is retrieving instructions on which to operate. The code space can be defined in 32K blocks as Low or High pages of the 64K code space in Internal or External code space. The two **XML** buttons and the two **XMH** buttons are mutually exclusive. For example, if the **XML Int** button is enabled and **XMH Ext** button is enabled, the simulator will operate on code in the Internal space from 0000h to 7FFFh and on code from External space from 8000h to FFFFh.

The **TE (Test Enable)** radio buttons determine whether or not the TEST jump vector gets executed when a Reset occurs. If the TE Enable button is selected, the processor will vector to FFFCh (the TEST jump vector) when a Reset occurs. If the **TE Disable** button is selected, the processor will go to 0000h when a Reset occurs.

PC and Oscillator Controls (edit windows)

These controls consist of the PC (program counter) edit box, Cycles (cycle count) edit box and the Osc (set oscillators) button that brings up a dialog box that allows the oscillator values to be set.

The **PC** edit box takes a hexadecimal value and loads it to the simulator program counter. The update occurs when the input focus leaves the edit box via the Accept Edits button, the keyboard Enter key, the keyboard Tab key, or selecting a different window with the mouse. The Source Code List window is updated to show the code at the new address.

The Cycles edit box takes a decimal value and loads it to the simulator cycle count location. This allows the cycle count to be initialized to a desired starting value.

The **Osc** button opens a dialog box that allows the user to set values for oscillator 1 and oscillator 2. The ratio of these values is used to determine the timer count (see Timers Display on next page) of the oscillator and are NOT used as the processor clock source. These values have no effect on Cycle Count or the timer count of the processor clock source. Therefore, the processor source driven timer will always show the same result for a given instruction. The default values are 14318100 (symbolizing 14.3181MHz) for Oscillator 1 and 32768 for Oscillator 2.

Hardware Stack Display

The Hardware Stack display consists of eight edit boxes that display the contents of the eight hardware stack locations. An arrow points to one of these locations to indicate that this is the next location to be written into when a CALL instruction is encountered. There is also a Stack Overflow radio button, which simulates operation of the corresponding bit in the flags register. The Stack Overflow radio button is set when the last stack location has been written. This is an early warning that the stack is full and that overflow will occur if another call is made or interrupt happens.

P0 and P1 Display (Output Ports)

The P0 and P1 displays indicate the status of the pins in port0 and port1 as they are seen at the input ports. This is because in the RSC-264T/364, the output port is wrapped into the input port.

Register address E2h: GPIO0 Input Register	(P0IN as referenced in Data Book)
Register address E3h: GPIO1 Input Register	(P1IN as referenced in Data Book)

The configuration of the I/O ports as inputs or outputs is defined by the SFR registers at the following locations.

Port0:

Register address E4h: GPIO0 Control A	(P0CTLA as referenced in Data Book)
Register address E6h: GPIO0 Control B	(P0CTLB as referenced in Data Book)

Port1:

Register address E5h: GPIO1 Control A	(P1CTLA as referenced in Data Book)
Register address E7h: GPIO1 Control B	(P1CTLB as referenced in Data Book)

The value of the output port is defined in the SFR registers at the following locations:

Port0:

Register address E0h: GPIO0 Output	(P0OUT as referenced in Data Book)
------------------------------------	------------------------------------

Port1:

Register address E1h: GPIO1 Output	(P1OUT as referenced in Data Book)
------------------------------------	------------------------------------

Example: If the GPIO0 Control A and Control B register contain 0Fh, then the bits 0-3 are configured as outputs and the bits 4-7 are configured as Input-Weak pull-up. If the GPIO0 Output register contains CCh then the P0 display for port0 contains the value 11111100 binary. Bits 0-3 are outputs with the value Ch and bits 4-7 are weak pull-up inputs.

Timers Display

Timer values are calculated for each operation executed based on the number of processor cycles it takes to perform the operation (including wait states), the timer pre-scalers, and the ratio of the two oscillator frequency values.

There are two edit boxes for T1 (Timer1) and two edit boxes for T2 (Timer2). The Load boxes display the contents of the Timer1 and Timer2 Reload Registers of the Special Function Registers (SFRs). The Cnt boxes display the contents of the Timer1 and Timer2 Counter Registers of the SFRs. The registers are summarized below.

Register address EBh: Timer1 Reload Register.

This is the value reloaded to the counter register when it overflows. (T1R in Data Book)

Register address ECh: Timer1 Counter Register

This is the timer counter value. It is a read-only register, writes to this register cause the reload register value to be loaded. (T1V in Data Book)

Register address EDh: Timer2 Reload Register

This is the value reloaded to the counter register when it overflows. (T2R in Data Book)

Register address EEh: Timer2 Counter Register

This is the timer counter value. It is a read-only register, writes to this register cause the reload register value to be loaded. (T2V in Data Book)

BANK, IMR, IRQ, FLAGS Display (SFR: Special Function Register)

This section of the Control Window displays four frequently monitored SFR registers in edit boxes. The four registers are summarized below.

Register address FCh: BANK RAM bank select (bits0-4) and Wait States (bits 5-7)

Register address FDh: IMR Interrupt Mask Register bits.

Register address FEh: IRQ Interrupt Request Register bits.

Register address FFh: FLAGSFlags Register bits.

5. Simulator: Source Code List

This window shows the code in three sections, which reflect the stages of code execution. The "history" section shows the code that has been processed and is shown above the highlighted line. The single highlighted line represents the current code pointed to by the program counter. The "look-ahead" section shows the code that will be processed and is shown below the highlighted line. When the RST (Reset) button is pressed, the "history" section is blanked to indicate that no code has been processed.

6. Simulator Tutorial

This section is a tutorial to familiarize the Simulator user with the following functions:

- Loading Files.
- Switching between different code spaces and the use of the XML and XMH controls.
- Displaying code space.
- The Run, Stop, RST, Step, StpOvr, AutoStep, and Brkpts buttons.
- Displaying and editing registers.
- Editing code space.
- Editing the Control Window edit boxes.
- Exiting the Simulator.

Tutorial Steps

1. Load the file called tutorial.bin into external code space by using the File: Load: External Code menu bar selections. The file is located in the \SAMPLES\TUTORIAL directory of supplied floppy disk or on the user's hard drive.
2. Set the XML radio buttons to Ext (External). Select the RST button (Reset) to load the current code into the Source Code List window.
3. Use the Step button to single step to address 8000h. The Source Code List window shows what happens when the processor transitions across the 32K high and low address space. The history shows code that has executed in the low part of external space and the current line shows the code that is about to execute in the high part of internal space. Note that no code is loaded here and that it is all " 00 NOP " instructions.
4. Set the XMH radio buttons to Ext and reset the Simulator by selecting the RST button. Use the Step button to single step to address 8000h and note that the list now shows executable code in the high part of external code space.
5. Use the Display menu bar item to display the External CODE Space. Use the vertical scroll bar to move to the NEXT BLOCK line at the end of the code list. Double click on the NEXT BLOCK line and use the edit box to set the beginning of the list at 8000h. Check that the External Code List displays the same code as in the Source Code List window.
6. Step through the code to demonstrate how the Source Code List displays the code space. Note the CALL and RET instructions.
7. AutoStep through the code and use the AutoStep Slider control to set the execution rate. Use the Step and Stop buttons to end the AutoStep command.
8. Use the Run command button to run through the code. Note that the Run button is flashing and that none of the displays are being updated. Use the Stop button to stop the Run command. Note that the displays are updated to where the Simulator stopped running.
9. Step to address 8000h. Use the StpOvr button to run to the instruction after the RET instruction at address 802Ch. Set a breakpoint at 8000h by using the BrkPts button. Run to this breakpoint.

AutoStep to this breakpoint. Remove the breakpoint and Run again. The Simulator is now running and no displays are being updated. Stop the Simulator.

10. Use the Display menu bar item to display the ScratchPad RAM registers. AutoStep through the code and note that register 0 displays the values being written to it. Stop the Simulator and edit register 0 by double clicking on the line that contains register 0. Insert a new value then Step the Simulator to display the edited value being overwritten by the executed instructions.
11. Edit the External Code List window by double clicking on address location 8024h. Change the code at location 8026h from 80 to 81. Observe the new instruction in the list windows. This forces a jump to a new section of code.
12. Step through the new section of code up to memory location 8101h. At address 8101h, edit the FLAGS register in the Control Window to load the value 00h. This will clear the Carry bit and cause the JNC instruction to jump to address 8100h upon the next Step command. Step up to memory location 8105h. At address 8105h edit the FLAGS register in the Control Window to load the value 80h. This sets the Carry bit and the JNC instruction at 8105h will not jump. The code "falls through" to address 8108h, which is an error since no code is loaded here. Edit the PC in the Control Window to set the Simulator to address 8100h and hit the computer Enter key to insert this value into the PC. Use the Step button to demonstrate that the Simulator continues stepping from this new location.
13. Exit the Simulator by choosing the File: Exit menu bar selections or use the icon in the top left of the screen to close the window. This completes the Simulator tutorial.

7. Simulator Operation Notes

This section contains information on known problems with the simulator software operation. The user should carefully read each of the notes below to be aware of the problems, and to know what to do when they are encountered during simulator operation.

1. The External Data List window may open up in a different size after being opened three or more times during a session. The user will have to resize as desired.
1. 1. 2. When using Windows 3.1: the Internal Code List, External Code List, or External Data List windows will be displayed in the "Home" corner, reduced to their minimum dimensions, each time they are opened after the first time of being opened and closed. The user will have to move the window and enlarge as desired.
3. The Internal Code List, External Code List, External Data List, Source Code List, and the SFR RAM windows can not be moved on top of the Control Window, All Bank RAM window, or ScratchPad RAM window.
4. When in an Edit SFR List window, if the Cancel button is clicked, an "Error During Edit" message occurs. The error window can be cleared and the system continues to function.
5. The selection of the RST command button does not cause the program counter and cycle count boxes to display zero. The boxes do display the proper information after the next Step command is selected.

CHAPTER 6 - ASSEMBLER & LINKER

This chapter contains reference information on the Sensory RSC-264T/364 assembler, SASM2.EXE, and linker programs. The assembler subsection provides information on command line options, source code syntax, and descriptions of the various directives. The linker subsection provides a description of its command line options.

Applications that use speech synthesis and/or speaker independent speech recognition require the linking of speech data files and neural net weight tables with the other object code modules. The procedure for doing this is described in this section.

This chapter also contains a description of the RSC-264T/364 Assembler Extensions related to the Intel 8051(tm), which may be useful to developers experienced with that processor. A table lists the supported Intel 8051(tm) instructions, the corresponding RSC-264T/364 Assembler form of the instruction, and information on any source code modifications required.

The Linker provided in this release is compatible with previous versions of Sensory Developer Kit releases. The Assembler, SASM2.EXE, is designed expressly for the new hardware architecture of the RSC-264T/364, and is NOT compatible with previous versions.

1. General Reference

Each of the programs (SASM2, CLINK) is normally started by typing a command at the MS-DOS command prompt. The reference chapters for those programs describe the syntax and available options of these commands.

Each program returns an exit code of zero if there were no errors in processing. A non-zero exit code may be used by a parent program or MS-DOS batch file to detect errors and stop processing.

Text File Input Syntax

Each line of input is terminated by an ASCII carriage return/line feed sequence. These lines of text are further subdivided into a sequence of source tokens. Tokens are groups of ASCII characters, separated by blanks.

There are five kinds of tokens recognized by SASM2:

Numeric constants

Always start with a digit 0-9. A suffix letter following the number will determine the radix: "x" or "h" means hexadecimal, "o" or "q" means octal, "b" means binary. If none of these suffixes appears, the number is considered to be decimal. Hex constants may contain the letters [A-Fa-f], but must start with a digit --- FF00h is illegal, but 0FF00h is acceptable. Numeric constants are converted to a 32-bit integer form for internal use.

String Literals

Consist of any ASCII text enclosed in quotation marks. String literals may be any length that fits on a single line. Either single or double quotes may be used as the delimiter.

End Of Line

Is either a carriage return, linefeed, or semicolon. The semicolon begins a comment so the text scanner ignores the remainder of the line.

Symbolic Names

A group of characters generally resembling a word of English text. A symbol name must begin with an alphabetic character (either upper or lower case) or the underbar character "_". Following the starting character any combination of alphabetic characters, underbars, periods, and decimal digits may be used. Symbol names may be up to 19 characters in length.

Special Characters

Any nonblank characters that are not interpreted as one of the above tokens. These can be parentheses, punctuation, operator characters, etc. Each special character token contains a single character.

Expressions

SASM2 expressions use the familiar algebraic infix notation. There are four major data types: numeric, address, external address, and character string. The operands and results of an expression may be any of these.

Numeric operands are specified either as literal constants or by using a symbolic name. Such a name

may have any value of any type.

Address data consist of a 16-bit unsigned offset from the beginning of a memory segment. An address is absolute if it is in an absolute segment (a segment opened with an AT clause). Otherwise, the address is relocatable. Many of the arithmetic operations are illegal on addresses since the actual address value may not be known until CLINK time.

An external address value consists of a 16-bit unsigned offset from a symbol that was specified by an EXTERN directive; such a symbol is defined in a different .A file and its value will not be known until CLINK time.

The following operators are supported in expressions:

Precedence	Operators	Operation
1 (highest)	+ - ~ !	unary plus, two's complement, 1's complement, not
2	* /	multiply, divide
3	+ -	add, subtract
4	> >= < <= = !=	comparison
5	&	logical and
6 (lowest)	^	logical or, exclusive-or

As usual, parentheses may be used to alter precedences.

A unary operator will operate on the operand that appears to its right in the expression and form a new operand from the result.

Comparison operators return a value of -1 if true and 0 if false. This allows the logical AND (&), OR (|), and XOR(^) functions to serve as both bitwise and Boolean operators. The difference between the NOT (!) and INVERT (~) operators is that the NOT returns -1 if its operand is zero and returns zero in all other cases, but INVERT returns the bitwise inversion (1's complement) of its operand. The unary minus (-) operator returns the 2's complement of its operand.

All operators may be applied to numeric data types. Address values may be operated upon only by the comparison operators, except a number may be added to an address, subtracted from an address, or two addresses may be subtracted. Two addresses cannot be added.

Expression Functions

Expressions functions may be used to control the assembly operation. An expression function can appear anywhere an expression is legal and takes the form:

FunctionName (arg1, arg2,...)

DEFINED (<expression>)

The defined function returns FALSE (zero) if its argument expression is of undefined type. It returns TRUE (-1) if the expression can be evaluated. This will usually be used to test whether a particular sym-

bol has been defined.

If the expression contains a forward reference, DEFINED will return FALSE during pass1 and TRUE during pass 2 of SASM2.

OFFSET (<expression>)

The offset function takes an address expression and returns a number. The result is the distance of the argument expression from the beginning of the segment in which it resides. External addresses are not legal.

MOD (<number>,<number>)

The mod function is the standard modulus function. Mod(a,b) returns the remainder from the integer division a/b.

SHL (<number>,<shift-count>) or <number> **SHL** <shift-count>
SHR (<number>,<shift-count>) or <number> **SHR** <shift-count>

SHL and SHR are shift-left and shift-right functions. The shift count must be a number in the range [0..31]. The value returned is the first argument shifted by the number of bits given in the second argument. SHR performs an arithmetic right-shift - that is, the original sign bit is retained through the operation.

LOW (<expression>)

HIGH (<expression>)

LOW and HIGH are byte-extraction functions. The expression must evaluate to a numeric value in the range [-32768..65535]. The value returned is either the low 8-bits or the high 8-bits of the argument.

NOTE: LOW and HIGH may be used only in the context of an instruction opcode with immediate addressing (e.g., mov a, #LOW (VALUE)).

2. Sensory Assembler (SASM2) Reference

SASM2 is the assembler program that converts the assembly language programs (source code) into relocatable machine language (object code) modules.

Command Line Options

SASM2.EXE is the assembler program itself. It is started by a command of the following format:

SASM2 <source-name> <options>

The source name is a full path name acceptable to MS-DOS. If no source name is specified, SASM2 will prompt for one. The default file type string ".A" will be appended to the name specified if no file type was specified, and the resulting string used to open the source file. Therefore, if the specified source-name is /SRC/MAIN, SASM2 will attempt to read file /SRC/MAIN.A. If the specified source file does not exist, SASM2 will terminate with an error message. Only one source file may be specified for each run of SASM2.

While starting up, SASM2 looks for an MS-DOS environment variable by the name of CASMSPEC. If present, CASMSPEC contains command-line text used to set default options for SASM2 runs. Options in CASMSPEC have the same syntax as the command line options. Options actually specified in the command line always override CASMSPEC specifications.

Options each start with the "-" (minus) character, followed by a flag letter which selects the option. Options may be given in any order or omitted entirely. The flag letter may be followed (with no intervening blanks) by additional text relating to the option. Options are separated from each other and from the source name by blanks. All option text is case insensitive. The following options are legal:

-l

-lNAME

The -L option causes the assembler to generate a listing file. If -L is not present, no listing file will be generated. NAME is optional -- if it is not present the source name will be used. NAME may be a full path name which is not in the current working directory. If no file type is given, the file type string ".LST" will be appended to NAME (or the source name) and the resulting string used to open the listing file for output. If the specified .LST file already exists it will be deleted and overwritten.

-tPATH

The -T option selects the drive or directory path name where SASM2 can create its temporary file. If the -T option is not used, SASM2 will look for a shell environment variable named TMP and use its contents instead. If variable TMP is also absent, the temporary file will be created in the current working directory. SASM2's temporary file is named SASM.TMP and will be deleted before SASM2 terminates.

-oNAME

The -O option selects the filename to be used for the object file output. If -O is used it must have a NAME specified. If the -O option is not given the source name will be used as NAME. NAME may be a full path name which is not in the current working directory. The file type string ".O" will be appended to NAME (or the source name) and the resulting string used to create the object file. If the specified .O file already exists it will be deleted and overwritten.

-dNAME

-dNAME=TEXT

-dNAME:TEXT

The -D option makes a symbol "NAME" available to SASM2 during assembly. NAME will be a character string containing the string TEXT, or a string of length zero if no text was given. Colon and equal sign have identical meaning in this option; the colon must be used if you intend to use the -D option within your CASMSPEC environment variable since the MS-DOS SET command will not allow an equal sign to appear in the environment string.

Overall Source Syntax

The source file is processed as a sequence of lines. Each line is terminated by a carriage return/line feed sequence. While reading the source lines, SASM2 keeps count of the line number and includes these numbers in listings and error messages. Only one statement is allowed per line. Empty lines are acceptable and will be ignored by SASM2 but included in the line count.

Comments are denoted by the semicolon ";" character. Any text to the right of a semicolon will be ignored by SASM2 but shown in the listing file. As far as SASM2, is concerned, a semicolon indicates the end of the line and further discussions of source line syntax will make that assumption. Therefore, a line starting with a semicolon is an empty line.

The first character position of each line is processed specially: if it contains a percent sign, "%", or any character which is a legal starting character for a symbol name then the line will be considered to have a label. If not, SASM2 will rule that no label is present for this statement. For most statements, the label will define a symbol which may be used in expressions elsewhere in the program. Typically (but not always) the value of that symbol will be the current assembly address.

If a label is present, it may optionally be followed with a colon ":" character. The colon is never mandatory and if present is always ignored. This is for compatibility with many different assemblers, some of which require colons and some of which do not. If the colon is not present then a blank or tab character should be used to separate the label from any following text. A label may be alone on a line with no text, or with only a comment following it.

If no label is present but the line is not empty, the first character of the line should be a blank. The remaining text on the line will be processed as an assembler statement. All statements which generate code and/or allocate memory in the target processor are referred to as instructions. There are a number of predefined statements which are used to control the operation of SASM2; these are referred to as SASM2 directives.

Following the label field, the source line is interpreted as a sequence of tokens, which may be separated by blanks. The blanks are not required unless the tokens could not otherwise be distinguished. Multiple blanks may be used anywhere that a single blank character is legal, and SASM2 will ignore the extra blanks.

Directives

SASM2 directives are all of the same general form: the first token (after the label, if any) is an identifier word which determines the type of directive. The nature of the text to the right of this directive name will depend on the particular directive involved. This additional text is referred to as arguments to the directive. These directives are defined by SASM2 itself and their operation cannot be altered.

The INCLUDE Directive

Example:

```
INCLUDE "..\INCLUDE\FILE.INC"
```

The INCLUDE directive allows no label. It takes one argument which must be a character string expression. The string will be used as a filename (full paths allowed) to open a file for input. No file type will be appended to this filename, it will be used just as is. If there is no file by that name, SASM2 will issue an error message and terminate. Source lines will be read from this file (an include file) until the end of file is reached. At that time the include file will be closed and the source file reopened to continue where it left off.

When an INCLUDE directive opens an include file, the line number maintained by SASM2 starts over at line one and counts from there for the duration of the file. When the include ends, the correct line

number for the source file is restored. Include file text appearing in SASM2 listing files show include file line numbers with a "+" sign so that you know that the text you are seeing is not from the main source file. Error messages issued by SASM2 will also tell you the file name and the line number as well to avoid confusion.

Include files can contain INCLUDE directives that include other files. They can be nested as deep as desired.

The LIBINCL Directive

The LIBINCL directive is similar to the INCLUDE directive, except that it uses the DOS environment variable %LIBPATH% as the root of its file path. For example,

```
LIBINCL    "\LIB364\MACROS\PATGEN.INC"
```

Will include the file contained at %libpath%\lib364\macropatgen.inc. The libpath can be a different directory or drive. This directive allows switching libraries by simply changing the libpath variable, with no need to change the source file.

The END Directive

The END directive allows no label and no arguments. It marks the end of the source file. When SASM2 encounters an END directive it stops reading source lines. The END directive is optional. If SASM2 encounters the end of the source file it will automatically assume an END directive.

An END directive in an include file ends the include file but not the assembly. SASM2 returns to the source file that issued the INCLUDE directive.

The SEGMENT Directive

Example:

```
NAME      SEGMENT "Mem.ROM" ALIGN(8) INPAGE(256)
```

The SEGMENT directive defines an address segment.

A SEGMENT directive normally has a label, which defines the name of the segment. CLINK will combine all segments bearing the same name into a single segment. If the label on a SEGMENT directive duplicates one used on an earlier SEGMENT directive in the same source file, SASM2 reopens the previous segment and adds the subsequent data onto that segment. When reopening a previous segment, no arguments are allowed on the SEGMENT directive.

If no label is specified, the segment is unnamed. SASM2 and CLINK will each invent names for internal use in referring to unnamed segments, and these generated names will appear in listing and map files.

The SEGMENT directive accepts a list of option clauses separated by blanks. None of these options are mandatory -- a SEGMENT directive with no arguments is legal and useful. The following options are allowed:

```
"<class-name>"
```

The class-name clause appears as a quoted string constant. Complex string expressions are not allowed. Class names may contain any characters -- they are not constrained by the rules that control other identifiers. The class name defines the address class where this segment will be placed by CLINK. If no class-name appears in a SEGMENT directive, the segment will be given a null class. The allowable class names are "CODE", "REG", "DATA", "REGLIN", "REGOVERLAY", "REGSTACK", "REGUSER". Most of these are for use by Sensory technology code - developers will have little need for any other than "CODE".

AT <expression>

The AT clause declares an absolute segment, starting at the address given. The expression must be numeric.

ALIGN <expression>

The ALIGN clause tells the linker to place this segment starting at an address evenly divisible by the given number. Such a segment is called an aligned segment. ALIGN cannot be used on an absolute segment. The expression must be numeric. For readability, it can be useful to put the expression in parentheses as in the example above, but this is not required.

INPAGE <expression>

The INPAGE clause tells the linker to place this segment entirely within a page of the specified size. The page boundaries are considered to fall at locations where (address MOD pagesize = 0).

INPAGE does not imply ALIGN but they may both be specified for a given segment. However, INPAGE cannot be used on an absolute segment.

COMMON

The COMMON clause tells the linker that each segment of this name will be assigned the same addresses. If COMMON is not specified the segments of the same name will be appended together in the order they are encountered. COMMON segments of the same name from different source modules must all be the same size. COMMON segments are often used to define hardware features such as "page zero" memory or special function registers that are shared by all modules of a program.

Storage Definition and Allocation Directives

The DB Directive

The DB directive allows the assignment of constant 8-bit values to storage in code space segments. Values must be in the range [-128...255]. An optional label is a symbol of address data type.

Example:

Squares: DB 0, 1, 4, 9, 16, 25, 36, 49

The DW Directive

The DW directive allows the assignment of constant 16-bit values to storage in code space segments. Values must be in the range [-32768...65535]. An optional label is a symbol of address data type.

Example:

Factorials: DW 0, 1, 2, 6, 24, 120, 720, 5040, 40320

The DS directive

The DS directive allocates uninitialized storage in RAM segments. Use of the DS directive leads to spurious error messages when overlaying memory, so Sensory has stopped using this directive. An optional label is a symbol of address data type.

Example:

```
variable    ds 1
pointer     ds 2
temp        ds 1
```

The FL and FIXLABEL Directives

The FL and FIXLABEL directives define address symbols relative to a specific fixed address. These directives provide the standard method of defining RAM addresses in the RSC-264T/364. The example below illustrates defining the address symbols of three variables.

Example:

```
custom_area fixlabel REG at FIRST_ADDRESS    ; FIRST_ADDRESS must be defined
custom_area fixlabel REG end at LAST_ADDRESS  ; this defines the range

variable fl 1      ; symbol value is FIRST_ADDRESS+0
pointer  fl 2      ; symbol value is FIRST_ADDRESS+1
temp     fl 1      ; symbol value is FIRST_ADDRESS+3
```

Each instance of fl is checked against the "end at" value to assure the address is in bounds. NOTE: the fl directive does not allocate storage, but merely defines the symbol value and equates it to a RAM address.

The EQU Directive

Example:

```
SECDAY EQU 24*60*60
```

The EQU directive allows the definition of symbols of any arbitrary value. The label field must contain the name of the symbol to be defined. The EQU directive takes a single argument which is an expression of numeric, string, or address type or an enumerated symbol. The new symbol will take on the value of that expression. The expression must not contain forward references.

The symbol in the label field must not be already defined when the EQU statement is executed, and it cannot be redefined afterward.

The SET Directive

The SET directive is similar to the EQU directive in that it defines symbols to the value given in the argument. However, a symbol defined by SET can be redefined as often as wanted by subsequent SET directives. In addition, the SET expression can contain forward references but if it does the symbol will

be UNDEFINED during pass 1 of SASM2.

When a SET symbol changes value several times during Pass One, the value will change during Pass Two to track its Pass One value. This assures consistent results when forward reference expressions are evaluated during Pass Two.

The EXTERN Directive

Example:

```
EXTERN  SYMBOL1, SYMBOL2, SYMBOL3
```

The arguments to the EXTERN directive consist of a list of symbols separated by commas. Each listed symbol named will be defined as EXTERNAL. Any reference to an EXTERNAL symbol elsewhere in the source file will require CLINK to resolve this address. CLINK will match up EXTERNAL references with PUBLIC definitions in other source modules. No label is allowed on an EXTERN directive.

The PUBLIC Directive

Example:

```
PUBLIC  SYMBOL1, SYMBOL2, SYMBOL3
```

The arguments to the PUBLIC directive consist of a list of symbols separated by commas. Each listed symbol must be defined elsewhere in the source file, and it will be marked as PUBLIC and made known to CLINK. When a symbol is made PUBLIC in one source module it must not be made public in any other module that will be linked with its module. PUBLIC symbols can be referenced from within other source modules if they are made EXTERNAL from those modules. If a symbol is not mentioned in a PUBLIC directive it will not be known outside of the source module it is defined in. No label is allowed on a PUBLIC directive.

The symbols listed in the PUBLIC directive may be addresses or numbers. The PUBLIC directive for a symbol may appear before the symbol is defined - forward references are legal.

The LIST Directive

The LIST directive controls generation of the listing file. The LIST directive has no effect whatsoever if the -L option was not specified in the SASM2 command line. There is no label allowed on a LIST directive. LIST takes one or more arguments separated by commas. The following arguments are legal:

LIST OFF

Turns OFF the listing. No listing will be generated for subsequent source lines until a LIST ON directive turns listing on again.

LIST ON

Turns the listing ON after it has been turned off by a LIST OFF directive.

LIST -COND

Lines of conditionally assembled code that are within an inactive IF block will not be listed.

LIST +COND

LIST COND

Turns COND listing on again after it has been turned off by a LIST -COND.

LIST -MAC

Lines of code generated by the expansion of a macro will not appear in the listing file.

LIST +MAC

LIST MAC

Turns MAC listing on again after it has been turned off by a LIST -MAC.

LIST -INC

Lines of code brought in by an INCLUDE directive will not be listed.

LIST +INC

LIST INC

Turns INC listing on again after it has been turned off by a LIST -INC.

LIST -SYM

The symbol table will not be listed at the end of the listing file.

LIST +SYM

LIST SYM

Reverses the effect of LIST -SYM.

LIST +GENSYM

LIST GENSYM

Causes SASM2's internally generated symbols to be listed as part of the symbol table at the end of the listing file. These symbols are often not of interest and serve to clutter and confuse the symbol table when many macros are used.

LIST -GENSYM

Suppresses the generated symbols in the symbol table listing.

LIST +ALLSYM

LIST ALLSYM

Causes symbols derived from the processor definition file to be included in the symbol table listing. Such reserved symbols include enumerated symbol names and any symbol defined within the automatic include section of the processor definition file. These symbols are normally hidden from the assembly language programmer.

LIST -ALLSYM

Suppresses the "hidden" symbols in the symbol table listing.

If no LIST directives appear in a source file, the default setting is:

LIST ON, +COND, +MAC, +INC, +SYM, -GENSYM, -ALLSYM

The PAGE Directive

Example:

```
PAGE    80                ; 80 lines per page
PAGE                    ; Start a new listing page
```

The PAGE directive has two forms. Its first form has no arguments, and causes the listing file to advance to the top of a new page. The other form takes a single argument which must be a numeric expression. This sets the page size to the given number of lines but does not advance to the next page. Neither form allows a label, and neither form has any effect if the L option was not specified on the SASM2 command line.

If page length is not set by a PAGE directive, it defaults to 60 lines of text.

The TITLE Directive

Example:

```
TITLE   "RS232 Driver Module Version 1.0"
```

The TITLE directive takes a character string expression as its argument. The top heading line of every page in the listing file will use the given string as a title. The TITLE directive may be used only once in any source file, and applies to the entire listing no matter where it appears in the source file. No label is allowed on the TITLE directive.

The SUBTITLE Directive

The SUBTITLE directive has the same form as the TITLE directive. When SUBTITLE is encountered in a source file then SASM2 begins to set the second heading line of every page in the listing file will use the given string. If SUBTITLE appears in the first four lines of a listing page, the new subtitle takes effect on that page. Otherwise, the new subtitle takes effect on the following page. The SUBTITLE directive may be used as many times as desired in a source file to change the page subtitle. No label is allowed on the SUBTITLE directive.

The IF Directive

Example:

```
IF      COUNT <= 5
```

The IF directive starts a conditional assembly block. It takes a single argument, which must be a numeric expression. If that expression is TRUE (non-zero) then the source lines that follow (which are said to be inside the IF block) will be assembled normally. If the argument is FALSE (zero) then the lines in the IF block are skipped over and not assembled. Such lines are marked in the listing file with a "-" character next to the line number to indicate that the line is not active. Alternatively, the LIST -COND directive

can cause inactive lines to be removed entirely from the listing to avoid confusion. An IF block is terminated when an ELSE, ELSEIF, or ENDIF directive is encountered. IF directives may be nested inside IF blocks. No label is allowed on an IF directive.

Within a conditional assembly block started with an IF directive, there may be any number of ELSEIF blocks and one optional ELSE block. When the IF expression was TRUE (and the IF block was assembled), then none of the ELSEIF or ELSE blocks will ever be assembled. In this case SASM2 skips down to the ENDIF without processing any source lines. When the IF expression was FALSE (and the IF block was skipped), SASM2 will process following ELSEIF or ELSE directives to see if another block should be assembled. If there are no ELSE or active ELSEIF blocks, then the entire conditional assembly block may be skipped.

The ELSEIF Directive

Example:

```
ELSEIF COUNT > 1
```

The ELSEIF directive appears at the end of an IF or ELSEIF block. ELSEIF allows no label, and takes one argument which is a numeric expression. If the preceding IF or any preceding ELSEIF since was active then the ELSEIF block will always be skipped. Otherwise, ELSEIF checks its argument value. If TRUE, the ELSEIF block will be assembled. If FALSE, the ELSEIF block will be skipped. An ELSEIF block is terminated by an ENDIF, ELSEIF, or ELSE directive.

The ELSE Directive

The ELSE directive must appear at the end of an IF or ELSEIF block. It takes neither arguments nor label. The ELSE block is assembled only if no IF or ELSEIF block in this conditional assembly block was assembled. An ELSE block consists of all the lines of source following the ELSE directive until the ENDIF is encountered.

The ENDIF Directive

The ENDIF directive allows no label or arguments. It marks the end of a IF, ELSE, or ELSEIF block, and also the end of the entire conditional assembly block.

The MACRO Directive

Examples:

```
MACRO  
MACRO NOLIST
```

The MACRO directive starts the definition of a macro. No label is allowed on the MACRO directive. The argument NOLIST may be optionally specified - this will prevent the text generated by expansions of this macro from appearing on the listing file, regardless of the presence of LIST MAC directives.

The next line after a MACRO directive is a prototype line which defines the appearance of the macro being defined. Following the prototype line will be the body of the macro which consists of a series of

source lines that will be stored but not assembled during the MACRO definition. They will be processed and assembled as source (macro expansion) later when SASM2 encounters a source line matching the prototype (a macro call).

The general form of the macro prototype line is:

`%labelparam` ANY TEXT `%param1, %param2, ...`

The identifiers of the form "`%varname`" are macro parameters. When the macro is called, these variables will accept text from the macro call line which will be used the macro. As the macro body is expanded, each line will be scanned for "`%name`" symbols. When a "`%name`" is found that matches one of the macro parameters it will be deleted from the line and replaced with the text stored in that parameter. If a "`%name`" is found which was not one of the parameters in the prototype line, it acts as a "generated symbol". Each time a macro is called, SASM2 will generate a unique symbol name for each generated symbol and store the text of that name into the `%name` variable. This allows labels of the form "`%name`" to be defined and referenced within the macro but be unavailable outside the macro expansion. A prototype may have any number (including none) of parameters up to a limit of 128 discrete `%name` parameter symbols per macro.

The `%labelparam` shown above in the prototype line is optional. If a label parameter is present in the prototype line then any label on the macro call will be stored into the `%labelparam` variable and this can be used during expansion of the macro body. If no label parameter is present on the prototype line, any label appearing on the macro call will be treated the same way a label on a normal instruction is treated. That is, it is placed in the symbol table with an address type and value of the current assembly location. When a macro is called with no label, its label parameter will contain a null string.

Macro parameters and generated symbols should not be confused with local labels (discussed elsewhere). While the `%name` format resembles a local label, these are different entities. In fact, local labels cannot be used within a macro.

The `%` flag character can have other functions within the macro body as well. The following summary lists all the uses of `%` in a macro body:

`%name`

Replaced by generated symbol or macro parameter text as described above.

`%%`

Replaced by `%`. Used when you want to have a percent sign in the expanded line.

`%;`

Causes the rest of the line to disappear. This can be used to put comments on a line in the macro body that will not appear in the expansion.

`%&`

Deleted from the line and the line closed up over the two character positions it used. Can be used to concatenate macro parameter text with other text.

%<blank>

Deleted from the line, and all further scanning of the line is stopped at this point. Can be used when placing a macro definition (or REPT directive) within a macro definition to prevent % flags and ENDM statements from confusing the macro processing.

%(<string-expression>)

Substitutes the text from the string expression into the line. The string expression is evaluated at the time of macro expansion. Any macro parameter symbols in that string expression will be substituted out before the expression is evaluated.

Note that these % operations will work even within string constants that are set off by quotes, or within comment text.

A macro is called from assembly language by a source line that matches the text given in the macro prototype line. The macro definition must precede the call in the source file. When the macro call is encountered, text from the macro body (with the substitutions described above) will be assembled as if these lines had appeared in the source file.

In the macro call, parameter text must be listed with commas separating the individual parameters. With a parameter, the sequence %% may be used to represent a % character to be passed to the macro. The sequence %, may be used to place a comma into the macro parameter (without the percent sign, a comma is interpreted as a parameter separator). An alternative way to prevent misinterpretation of these characters is to use braces in the form {--any-text--} to delimit the macro parameter. In that form, all text within the braces is passed through unmodified.

The ENDM Directive

The ENDM directive marks the end of a macro body. When encountered during macro definition (after a MACRO directive) the definition is complete. No label or arguments are allowed.

The EXITM Directive

The EXITM directive can be used within a macro to cause expansion to stop immediately. This is usually used inside an IF block within the macro body to "bail out" of the macro expansion early. No label or arguments are allowed.

The REPT Directive

A REPT directive starts a repeat block which is a special form of macro processing. REPT takes one argument which is a repeat count. The REPT directive is followed by macro body text in the same form as the macro body following a MACRO directive and prototype. The REPT directive has no prototype so it has no macro parameters but it can still have local variables.

The REPT block ends with an ENDM just as a macro block does. However, when the ENDM is encountered the body of the REPT is immediately expanded, and the expansion is repeated the number of times given by the repeat count. If the repeat count is zero or negative there is no expansion at all.

If an EXITM directive is encountered during REPT block expansion, the repeat operation is immediately

terminated even if the repeat count has not been completed.

SASM2 Local Labels

Local labels are a means of providing a label which is not part of the SASM2 symbol table name and can only be used by nearby instructions (usually jumps). This is convenient when building small loops or if/then structures.

A local label consists of a percent sign % in the label field followed by an optional symbol name. This label can be referred to as a forward reference by using the > character where an address expression is expected. Use a < to make a backward reference to a local label. The syntax looks like this:

```
JMP    >NAME    ; Jump forward to the local label
.
.
%NAME:                ; The local label is here
.
.
JMP    <NAME    ; Jump backward to the local label
```

The rules for use of local labels are as follows:

- References to local labels cannot go past a normal symbolic label. Attempts to do so will generate an error message.
- A particular local label name need not be unique to the assembly, or even to the segment. When resolving a local label reference, SASM2 uses a linear search technique to find the first matching name in its local label list.
- The local label must reside in the same segment as the reference to it.
- Local labels may not be used within macros or REPT blocks. Macros contain their own generated symbol facilities which can easily perform the same job
- A percent sign with no symbol after it is a valid local label. It would be referenced with a > or < character with no symbol.

Local labels can be used to impose some degree of structure readability on your assembly language code, as in the following examples:

```
; If/then/else example
...
JZ      >ELSE
...      ; Executed for THEN condition
JMP     >ENDIF
%ELSE:
...      ; Executed for ELSE condition
%ENDIF:

; Loop example
...      ; Loop initialization
```

```

%LOOP:
    . . .                ; Body of loop
    JZ      <LOOP

```

3. Sensory Linker (CLINK) Reference

CLINK performs the usual link/locate functions for binding separately compiled modules together, assigning addresses, and generating output files.

Command Line Options

CLINK.EXE is the linker program for SASM2. It is started by a command of the following format:

```
CLINK @<command-file> <obj-name> <obj-name> ... <options>
```

The object file names are full path names acceptable to MS-DOS. At least one object name must be specified. If no file type is given, the default file type string ".O" will be appended to the name specified and the resulting string used to open an object file for input. If any specified object file does not exist, CLINK will terminate with an error message. These object files will have been produced by SASM2.

CLINK uses an MS-DOS environment variable CLINKSPEC to provide command line defaults similar to the CASMSPEC variable used by SASM2. Text in the CLINKSPEC will be processed as if it had been present on the command line.

CLINK also uses environment variable TMP to determine a directory path name for its temporary file CLINK.TMP. If TMP is not present, the temporary file goes into the current working directory.

If desired, the "@<command-file>" specification may be used to specify more option text than will fit on a command line. Text from the command file will be treated as if it were found in the CLINK command line in place of the @ specification. There is no default file type for command files. Any file type must be explicitly supplied.

As with SASM2, options each start with the "-" (minus) character, followed by a flag letter which selects the option. Options may be given in any order or omitted. You may not specify any given option more than once in a single command. The following options are legal:

-oNAME

The -O option specifies the output files name. The name specified may be a full path which is not in the current working directory. It will be used to create all output files. If the -O option is not used, CLINK will use the first object file name specified. CLINK creates multiple output files based on this name, each with a type indicating the format and contents. The default file types created are ".HEX", ".DMP", and ".BIN". If any of the output files already exist, they will be deleted and overwritten. The output files contain the actual data which will be executed by the target processor.

-m

-mNAME

If present, the -M option causes a map file to be written. The map file contains information for the user about the allocation of memory to the program being linked. If no name is given, the output file name (without the suffix) will be used. If no file type is given, the default is .MAP.

-S

The -S option creates a symbol table that lists addresses and variables at the end of the listing file. If no name is given, the output file name (without the suffix) will be used. If no file type is given, the default is .SYM.

4. Linking Speech and Weights Data

Building an application with either speech synthesis or SI speech recognition requires linking your code with speech data files or neural net weight tables. The format of Sensory speech data files and neural net weight tables is compatible with the CLINK linker. Linking speech and weights files does require some minor special considerations, even when using the newer linking method:

1. Speech and weights files must always be located on page boundaries (the address must be evenly divisible by 256). This requirement arises from the use of only the #high byte of the address within the technology code. The files supplied in this release are all assembled with an "ALIGN 256" directive.
2. Speech and weight files must be linked with the rest of the application through pre-defined labels, which the programmer must know and understand.

Linking Speech and Weights Files Directly

The speech and weights data files are provided in subdirectories of \DATA as standard object (.O) files. Accompanying .TXT files describe the contents of the object files. In order to bring one of these .O files into your executable file you need to include the .O filename in the linker command file. You must also edit your source code to declare the file's internal label as an EXTERNAL reference. Finally you must edit your source code to reference the file's internal label correctly. Specifically, Sensory technologies usually deal only with the #high byte of the address. The code fragment below illustrates how a program would speak the phrase "three" (entry number 3) from the vocabulary in the file NUMBERSD.O.

```
        EXTERN      VPnumbersd ; label for numbersd vocabulary

VR      equ        9330/64 ; VoiceRate      - default sample rate
(Hz/64)

; (preliminary steps omitted) . . .
        mov a, #3    ; phrase #3, for example
        CallTalkBoth a, #high VPnumbersd, #VR      ; speak the message
```

The convention we have used for naming labels in the speech and weights files is:

1. Speech file labels have a prefix "VP" followed by the object file name (with the ".O" omitted).
2. Speech sentence file labels have a prefix "SN" followed by the object file name (with the ".O" omitted).
3. Weights file labels have a prefix "WT" followed by the object file name (with the ".O" omitted).
The preceding code sample illustrates this convention for a speech file. After the source code is assembled it would be linked with the required technology files and with the file NUMBERSD.O. The physical location of the speech data within the final binary image is decided by the linker. The speech and weights object files contain information that forces the linker to align them properly on page boundaries.

5. Sensory RSC-264T/364 Assembler Extensions

The RSC-264T/364 assembler contains Intel 8051 pseudo op-codes for use by assembly language programmers experienced in that device.

WARNING !

Not all 8051 instructions are supported and some require modification to allow them to be assembled. Be sure to read all descriptions and instructions **CAREFULLY!**

The following table shows the original form of the supported 8051 instruction, the RSC-264T/364 Assembler form of the instruction, whether a source code modification is required and a comment showing the type of source code modification. Some modifications are needed simply because the 8051 instruction conflicts with a native RSC instruction. Table entries requiring modifications are in bold.

For example, in the first entry in the instruction table:

ADD A,@R1	Add Rn,@Rm	No	
-----------	------------	----	--

ADD A,@R1 is the 8051 form of the instruction.

add Rn,@Rm is the RSC-264T/364 form of instruction, note that the destination register is defined as any register Rn and the indirect source register is defined as any register Rm.
No source code modification is required.

For example, in the sixth entry in the table:

ANL C,FACSGN	anl C,Rm,bitnum	yes	bit address->Rn,bitnum
---------------------	------------------------	------------	----------------------------------

ANL C,FACSGN is the 8051 form of the instruction that uses bit addressing which is not supported in the RSC-264T/364.

anl C,Rm,bitnum is the RSC-264T/364 form of the instruction that requires a register and bit number definition.

Source code modification is required.

The modification is that the "bit address" is replaced with a register and bit number.

8051 instruction	RSC-264T/364 instruction	mod?	Modification required
ADD A,@R1	add Rn,@Rm	no	
ADDC A,R5	addc Rn,Rm	no	
ADDC A,#0	addc Rn,#data	no	
ADDC A,@R1	addc Rn,@Rm	no	
ANL A,#01111111B	anl Rn,#data	no	
ANL C,FACSGN	anl C,Rm,bitnum	yes	bit address->Rn,bitnum
CJNE A,B,FLT03	cjne Rn,Rm,label	no	
CJNE R2,#000,FX03	cjne Rn,#data,label	no	
CLR C	clr c	no	
CLR A	clr Rn	no	
CLR ACC.0	clr Rn,bitnum	yes	bit address->Rn,bitnum
CPL C	cpl c	no	
CPL A	cpl Rn	no	
CPL FACSGN	cpl Rn,bitnum	yes	bit address->Rn,bitnum
DEC A	decc Rn	yes	dec->decc (RSC has native dec)
DJNZ R7,FOPR01	djnz Rn,label	no	
DIV AB	NOTE: see FPMACROS		Implemented as source code macro with differences
INC A	incc Rn	yes	inc->incc (RSC native inst)
INC @R1	inc @Rn	no	
INC DPTR	incdptr DPL	yes	inc->incdptr (RSC native inst)
JB ACC.0,FPA035	jb Rn,bitnum,label	yes	bit address->Rn,bitnum
JNB B.7,FLT01	jnb Rn,bitnum,label	yes	bit address->Rn,bitnum
JNZ address	jnza address	yes	jnz->jnza (RSC native inst)
JZ address	jza address	yes	jz->jza (RSC native inst)
MOV A,Rn	mov a,Rn	no	
MOV A,@Rn	mov a,@Rn	no	
MOV A,#data	mov a,#data	no	
MOV ACC.0,c	mov a,bitnum,c	yes	bit address->Rn,bitnum
MOV @R0,#0	mov Rn,#data	no	
MOV c,b.7 (mov c,facsgn)	mov c,Rn,bitnum	yes	bit address->Rn,bitnum
MOV FACSGN,C	mov Rn,bitnum,c	yes	bit address->Rn,bitnum
MOV DPTR,#INVLN2	movdptr DPL,#data	yes	mov->movdptr (RSC native inst)
MOVC A,@A+DPTR	movc a,@a,dptr	yes	@a+dptr->@a,dptr
MUL AB	NOTE: see FPMACROS		Implemented as a source code macro with differences
ORL FACBIT,#EROVF	orl Rn,#data	no	

RSC-264T/364			
8051 instruction	instruction	mod?	Modification required
ORL A,R3	orl Rn,Rm	no	
ORL C,FACBTM	orl c,facbit,6	yes	bit address->Rn,bitnum
POP B	pop Rn, @data_stack_ptr	yes	see stack instructions
PUSH ACC	push @data_stack_ptr, Rn	yes	see stack instructions
RETI	IRET	yes	reti->iret
SETB C	setb c	no	
SETB b,7 (setb FACSGN)	setb Rn,bitnum	yes	bit address->Rn,bitnum
SJMP FLT05	sjmp label	no	
SUBB A,#FBIAS	subb Rn,#data	no	
SUBB A,R0	subb Rn,Rm	no	
SUBB A,@R0	subb Rn,@Rm	no	
XCH A,B	xch Rn,Rm	no	
XRL A,R4	xrl Rn,Rm	no	

CHAPTER 7 - UPGRADES

This chapter describes the process of upgrading to the RSC-364 Development Kit with Sensory Speech 6 Technology from Release 5.0 of the RSC-364.

- *All new Library code is required. Sensory has tried to keep the Application Interface as familiar as possible, but there are minor changes.*
- *Minor changes are required to application level code to "port" it to Release 6*
- *Adding new application-level features in Version 6 is straightforward.*

1. Upgrading from RSC-364, Release 5.0

Summary of Changes

The Overview chapter lists the new features in this release. In order to take advantage of these features you will need to make the following changes:

- Modify application code to incorporate required feature changes (macro parameters, function name changes)
- Optionally modify application code to incorporate new features available with this release
- Optionally modify linker commands to incorporate modules containing new features

This section describes the required steps involved in the upgrade process and works through a sample conversion. Be sure to read the Overview and other documentation sections for details that may not be repeated here. This release is intended to be a complete replacement of the previous (5.0) release. THE TECHNOLOGY CODE IN RELEASE 6 IS NOT COMPATIBLE WITH THE RSC-164 OR RSC-264T AND WILL NOT RUN ON THOSE CHIPS. The sample conversion follows Sensory's recommended migration method. We suggest you work through the sample on your own system before beginning to migrate your own application.

Most of the changes in version 6 are "transparent" and require no application-level changes.

Include Files

The former \LIB364\MACROS\FASTDIG.INC is now named \LIB364\MACROS\DRT.INC. This reflects that Dual Recognition Technology now works with recognition sets other than Fast Digits. See Section 11 of Chapter 3, "Include and Link Requirements" for a summary of included file requirements for all technology modules.

Linker Commands

This release incorporates recognition improvements that require a change in the content of recognition weights files; accordingly, any weights files must be upgraded. All the sample weights files included in this release have been updated. If you are using custom weight files, contact Sensory about upgrading them for Version 6 Technology.

The number and names of all technology object files remain the same, so no changes are required to application LINK.CMD or builder batch files.

Required Application Changes

Developers wishing to upgrade to Release 6 must make minor changes (described below) in any application to accommodate these technology changes:

1. The arguments for some technology macros have changed.
2. Some parameters that previously had assembly-time binding now have run-time binding.

These technology-specific changes are required:

1. If you use pattern generation or voice recording, remove all references to the macros LdCtl, InitSL, and GetSL, as well as code that tests the results of these macro functions. The speech framing feature of version 6 makes these functions obsolete
2. Provide a new argument to the pattern generation macro to specify how to deal with certain errors. This parameter replaces the former NO_ERRORS constant. See Using Pattern Generation for more discussion.
3. If you use the Prior macro function in Speaker Independent recognition, provide an additional parameter to the macro to indicate how to treat the selected class. See Using Speaker Independent Speech Recognition for more discussion.
4. If you use Record and Play, provide an additional parameter to the RecordRP macro to specify the threshold type. See Using Voice Record and Playback for more discussion.
5. Remove all references to the TemplateSize10 and TemplateSize16 macros. These functions have been automated.
6. If you use Continuous Listening 4, the technology is now called simply Continuous Listening. Remove any code related to the CheckSL macro. The cl_performance parameter is now supplied as a run-time value to the Listen macro.
7. If you use Speaker Dependent recognition, provide a new argument to the RecogSD macro to specify the performance level. This parameter replaces the former SD_PERFORMANCE constant. See Using Speaker Dependent Speech Recognition for more discussion.
8. If you use Speaker Dependent recognition to perform a "too close" test during training, change the value of the "THRESHOLD_PLUS" parameter for the new SD weights. See Using Speaker Dependent Speech Recognition for more discussion.
9. If you use the SleepIO function, modify your code for the new, more general arguments. See the password sample for an example.
10. If your application uses local versions of STARTUP.A or CONFIG.A, compare your files with the library versions in this release and incorporate any changes.
11. The Dual Recognition macro, RecogDrtDigit, is now more general and has been re-named RecogDual. You must now supply an additional parameter to the macro to specify the extended address bits (DRT now supports 24-bit template addresses).

Optional Application Changes

Refer to the appropriate "Using..." document for details on the improvements summarized below. Also see the sample programs as described.

1. If you already use the SPFlash memory handler, you should use the new library code. This will require minor application changes. The new SPFlash memory handler for SD has an application interface similar to SRAM and SST flash; the version in 5.0 required different code flow.
2. Some developers will want to incorporate Wordspot SD technology, See Using Wordspot SD Speech Recognition for more information.
3. Applications may take advantage of the multi-pin IO wakeup capability available in "sleep mode" in this release. See \LIB364\MACROS\SLEEP.INC for details.
4. Applications may take advantage of the generalized Dual Recognition Technology in this release to gain improved recognition. NOTE: DRT makes calls to validate_template_sdv. The old SPFlash interface is not compatible with this use. To use DRT with SPFlash, you must use the

new handler.

IMPORTANT

Conversion of most applications can be completed in a few minutes. Be sure to make a backup of your application code before beginning the conversion.

Sample Software Conversion

As a sample of the process of porting an application from the June 1999 release 5.0 RSC-364 Development Kit to this version 6 release, we describe here converting the Speaker Dependent demo in \SAMPLES\SPKRDEP. An updated and improved version of this program with "name tagging" using SPFlash appears in this kit under \SAMPL364\SPKRDEP, but here we "port" the Release 5.0 program to use the new library modules. As we will see, this conversion requires only minor editing of SPKRDEP.A. As an example of the migration path for any application, we recommend:

1. First, install the Release 6 software in a new directory according to the instructions in the "Getting Started" section of the Overview Chapter. The following assumes the version 6 software is installed in directory C:\SENSORY, and the 5.0 software is in C:\DEVOLD.
2. Set the DOS environment variable, LibPath, to the root of this Sensory directory, and make sure your DOS path includes this directory.
3. Copy the old sample, C:\LIBOLD\SAMPLES\SPKRDEP, to a new folder in your working directory. For illustration, the following assumes this directory is called "X:\WORKING\SPKRDEP".
4. Change directory to the sample folder and work through the conversion.

This approach allows checking that the conversion works before any of the existing (old) library or application structure is touched. The editing changes are described below following the format:

```
<original data (removed or commented out)
--
>new data (replacement or added)
```

After installing the software and changing your DOS path, define the DOS environment variable, LibPath (use your own directory path name). The root of the Sensory directory contains a batch file, setlib.bat to simplify this - you type only the command in bold. This command also defines the environmental variable, casmspec, used by the assembler.

```
C:\sensory>setlib c:\sensory

C:\sensory>rem Sets the path of the root of Sensory's Library

C:\sensory>rem usage: SETLIB [path]

C:\sensory>set LibPath=c:\sensory

C:\sensory>set casmspec=-DLibPath:c:\sensory

C:\sensory>
```

Now create the new WORKING directory, copy the old SPKRDEP sample to the new

WORKING\SPKRDEP directory on X: (do NOT copy to \SAMPL364\SPKRDEP or you will overwrite the new sample code.), and change to that directory.

```
X:\>md working

x:>cd working

X:\working>XCOPY D:\DEVELOLD\SAMPLES\SPKRDEP\*. * SPKRDEP
Does spkrdep specify a file name
or directory name on the target
(F = file, D = directory)?d
Eng_seep.cmd
Eng_sram.cmd
Jap_seep.cmd
Jap_sram.cmd
Kor_seep.cmd
Kor_sram.cmd
lnkflash.cmd
patgen_h.a
PATGEN_H.O
spkrdep.a
        10 File(s) copied

x:\working>cd spkrdep

x:\working\spkrdep>
```

Editing SPKRDEP.A:

First edit SPKRDEP.A to remove the macros LdCtl and InitSL. These functions are automated in release 6. 524,525d523

```
<          LdCtl                               ;set up filters
<          InitSL
```

Next remove the code related to the GetSL macro, which is also automated in release 6. Make sure that any code testing the macro results is also removed.

```
523a524,527
>          GetSL
>          cp          a,#0                      ;check if too noisy
>          jnz          error
>
```

The Patgen macro now takes an additional argument that specifies the "no errors" condition (similar to the NO_ERRORS configuration constant in 5.0). To keep the operation the same as in 5.0 (all errors are reported), set the new parameter to 0. Note: in version 6 this parameter can be a run-time variable.

```
535c535
<          PatGen    #STANDARD
---
>          PatGen    #STANDARD, #0
```

The RecogSD macro now takes two arguments; the new argument replaces the former configuration constant, SD_PERFORMANCE, which can now be passed as an argument. Note: although shown here as a constant, in version 6 this parameter can be a run-time variable. The following change is made in two places:

```
<          RecogSD ctr
---
>          RecogSD ctr, #DEFAULT_SD_PERFORMANCE
```

The TrainSD macro now also requires a performance level parameter (which could also be a run-time variable).

```
<          TrainSD #TMPLT_UNKNOWN, #TMPLT_KNOWN, #TMPLT_UNKNOWN ;average tem
plates
---
>          TrainSD #TMPLT_UNKNOWN, #TMPLT_KNOWN, #TMPLT_UNKNOWN,
          #DEFAULT_SD_PERFORMANCE
```

Finally, we remove the call to TemplateSize16. This is now set automatically whenever the Patgen function is called.

```
532d531
<          TemplateSize16                                ; use large patterns
```

With these changes, the SPKRDEP sample now builds and executes correctly. This completes the conversion to version 6

Performance Checking

The build file creates the final binary file, ENG_SEEP.BIN. This file can be loaded into the DK264T/364 development hardware and it executes correctly. Examination of the last UNUSED portion of the .MAP file shows that the program occupies about 44Kbytes (unused is 4D2Ah =19754 bytes in this example.).

```
Address B461, length 4B9F, <UNUSED>
```

In the 5.0 version, this unused ROM was 1242 bytes larger:

```
Address AF87, length 5079, <UNUSED>
```

The size increase in this sample is mainly from patgen modules.

Additional Notes

This simple example illustrates all the changes that will be needed for most applications. In some applications additional error messages may still occur after this level of modification. For example, you may get an error similar to this:

```
>>>          RecogSD ctr                                ;recognize against template
```

```
list
>>>                                     ^
Error in s2.A:474 -- Too few arguments for this macro call.
```

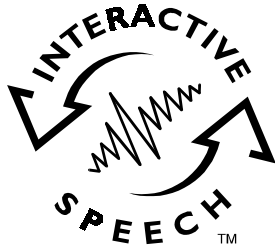
In such a case, refer to the Technology chapter for clarification of the proper macro arguments.

2. Upgrading from RSC-264T or RSC-164V2

Sensory does not supply separate upgrading instructions for chips outside the 364 family. If you are upgrading from an earlier release, contact Sensory.



CHAPTER 8 - DATABOOK



RSC-300/364 Data Book

GENERAL DESCRIPTION

The RSC-300/364, from the Interactive Speech™ family of products, is designed specifically for speech applications in consumer electronic products. The RSC-300/364 combines an 8-bit processor with neural-net algorithms to provide high-quality speaker-independent speech recognition, speaker-dependent speech recognition, and speaker verification. The chip also supports speech synthesis, voice record/playback, 4-voice music synthesis, and system control. This CMOS device includes on-chip RAM, ROM (RSC-364 only), 16 general-purpose I/O lines, A/D and D/A converters, a microphone pre-amplifier, and a 4-MIPS dedicated processor. The RSC-300 is designed for ROM-less applications that need more ROM space and consequently use off-chip memory.

In addition to providing the horsepower needed to perform speech recognition and speech synthesis, the processor has sufficient cycles available for general-purpose product control. The RSC-300/364 Development Kit allows developers to create custom applications. The Development Kit includes an assembler, linker, simulator, hardware development platform, and library of Sensory technology object code.

The highly-integrated nature of this chip reduces external parts count. A complete system may be built with only a few passive components in addition to a battery, speaker, and microphone. Low power requirements and low-voltage operation make the RSC-300/364 an ideal solution for battery-powered and hand-held devices.

The RSC-300/364 uses a pre-trained neural network to perform speaker-independent speech recognition, while high-quality speech synthesis is achieved using a time-domain compression scheme that improves on conventional ADPCM. Four-voice music synthesis allows multiple, simultaneous instruments for harmonizing. Automatic Gain Control can compensate for people not optimally positioned with respect to the microphone or for people who speak too softly or loudly.

FEATURES

High-Performance Processor

- 4-MIPS performance at 14.32 MHz
- 16 general purpose I/O lines
- Interrupts, timers and counters
- Fully static operation; clock rate: DC to 14.32 MHz

Highly-Integrated Single-Chip Solution

- Internal 64 kB of ROM (364 only), 2.5 kB of RAM
- 12 bit A/D (Analog to Digital) converter
- Microphone Pre-amplifier
- Internal 32kHz secondary timer
- 24 x 24 Multiplier
- Can store 6 Speaker Dependent words on-chip

Low Power Requirements

- Requires single 2.4V to 5.25V power supply
- ~10mA operating current at 3V
- Low Power 32kHz oscillator for clock applications
- Power-down current less than 5 μ A

High-Quality Recognition and Synthesis

- Recognition accuracy better than 97% (Speaker Independent) and 99% (Speaker Dependent).
- Synthesis data rates from 5,000-15,000 bits per second
- 4-voice music synthesis capabilities
- AGC control compensates for variations in input signal

Easily Expanded to larger-scale systems

- Separate 16-bit Address and 8-bit Data buses compatible with common memory components
- Separate Code and Data address spaces and memory strobes

IMPORTANT NOTICES

Sensory reserves the right to make changes to or to discontinue any product or service identified in this publication at any time without notice in order to improve design and supply the best possible product. Sensory does not assume responsibility for use of any circuitry other than circuitry entirely embodied in a Sensory product. Information contained herein is provided gratuitously and without liability to any user.

Reasonable efforts have been made to verify the accuracy of this information but no guarantee whatsoever is given as to the accuracy or as to its applicability to particular uses.

Applications described in this data sheet are for illustrative purposes only, and Sensory makes no warranties or representations that the Interactive Speech™ line of products will be suitable for such applications. In every instance, it must be the responsibility of the user to determine the suitability of the products for each application. Sensory products are not authorized for use as critical components in life support devices or systems.

Sensory conveys no license or title, either expressed or implied, under any patent, copyright, or mask work right to the Interactive Speech™ line of products, and Sensory makes no warranties or representations that the Interactive Speech™ line of products are free from patent, copyright, or mask work right infringement, unless otherwise specified. Nothing contained herein shall be construed as a recommendation to use any product in violation of existing patents or other rights of third parties. The sale of any Sensory product is subject to all Sensory Terms and Conditions of Sales and Sales Policies.

June 2000



S E N S O R Y

521 East Weddell Drive
Sunnyvale, CA 94089

TEL: (408) 744-9000
FAX: (408) 744-1299

© 1999 SENSORY, INC.
ALL RIGHTS RESERVED
P/N 80-0165-6

Sensory® is registered by the U.S. Patent and
Trademark Office. All other trademarks of registered
trademarks are the property of their respective owners.

Table of Contents

1. INTRODUCTION.....	1
2. RSC-300/364 HARDWARE SPECIFICATIONS.....	3
3. USING THE RSC-300/364	4
4. MEMORY ORGANIZATION.....	5
5. MEMORY MAP	6
6. GENERAL PURPOSE I/O.....	6
7. INTERRUPTS	8
8. RESET AND CLOCKS	9
9. TIMERS AND COUNTERS	9
10. POWER DOWN AND WAKEUP OPERATION	10
11. ANALOG OUTPUTS	10
12. HARDWARE DEBUG FEATURES	11
13. DESIGN CONSIDERATIONS	13
14. OMNI-DIRECTIONAL MICROPHONE	14
15. POWER CONSUMPTION AND POWER SUPPLY CONSIDERATIONS	17
16. DIE BOND PAD AND QFP PIN DESCRIPTIONS	18
17. DIE PAD RING.....	19
18. RSC-300/364 DIE BONDING PAD LOCATIONS	20
19. ABSOLUTE MAXIMUM RATINGS	21
20. D.C. CHARACTERISTICS	21
21. VDD vs. IDD	21
22. A.C. CHARACTERISTICS (EXTERNAL MEMORY ACCESSES)	22
23. TIMING DIAGRAMS	23
24. RSC-300/364 INSTRUCTION SET	24
25. RSC-300/364 SPECIAL FUNCTION REGISTER (SFR) SUMMARY	27
26. SPECIAL DATA SPACE ADDRESSES SUMMARY.....	45
27. QUALITY AND RELIABILITY	48
28. PACKAGING.....	50
29. ORDERING INFORMATION	50

1. INTRODUCTION

The RSC-300/364 is the newest member in a family of high-performance 8-bit microprocessors featuring a high level of integration, targeted to high-accuracy, low-cost speech recognition applications. The RSC-300/364 is designed to bring accuracy, fast response time and versatility to low-cost, power-sensitive consumer applications.

A design goal of the RSC-300/364 was to reduce total system cost while increasing system performance. By including microphone signal amplification, data conversion, recognition and synthesis functionality, and ROM¹ storage (RSC-364 only) with a CPU core on a single chip, dramatic cost and power reductions are achieved. Thus, the RSC-300/364 is able to provide 4 MIPS of integer performance at 14.32 MHz. This allows customer applications to achieve maximum performance at minimum cost.

The CPU core embedded in the RSC-300/364 is an 8-bit, variable-length-instruction, microprocessor. The instruction set is somewhat similar to the Zilog™ Z8 having a variety of addressing mode *mov* instructions. The RSC-300/364 processor avoids the limitations of dedicated registers by having completely symmetrical source and destinations for all instructions. Of the 2.5 Kbytes of internal SRAM, 2 Kbytes are organized as a Data Space, and 0.5 Kbytes is for register space. All arithmetic operation instructions may be applied to any register. Any pair of adjacent registers (at an even address) may be used as the 16-bit pointer to either the source or destination for a data movement instruction. Instruction classes allow the pointer to access internal or external Code Space, internal Register Space, or external Data Space.

Architecturally, the RSC-300/364's separate data and address buses allow use of standard EPROMs, ROMs, and SRAMs with little or no additional decoding. Provision for separate read and write signals for each external memory space further simplifies interfacing.

Creating applications using the RSC-300/364 requires the development of electronic circuitry, software code, and speech/music data files ("linguistics"). This document provides detailed information on those aspects of the RSC-300/364 architecture that are important to product designers and programmers. It describes the physical interface to the chip, printed circuit board layout and other design considerations, the RSC-300/364's instruction set, and memory organization. Refer to the RSC-300/364 Development Kit Manual for information on using Sensory's technology code for speech recognition, speaker verification, speech synthesis, and voice record and playback. Description of vocabulary development ("linguistics") information is beyond the scope of this document and is covered in a Design Note.

Custom Mask Capabilities of the RSC-364

The RSC-300 provides significant and flexible expansion capabilities through the use of external RAM or ROM. Products using the custom-mask version of the chip, the RSC-364, may save considerable per-unit cost by avoiding the need for other active devices. The RSC-300 requires an external Code Space ROM memory to contain the program instructions, synthesis data, and Speaker Independent recognition weights. The custom-masked RSC-364 with no additional external memory devices must rely on the fixed internal memory for all of its ROM and RAM requirements. The internal ROM in the RSC-364 is application specific, with the amount available for user applications decreasing as the number of synthesis words or other technology usage increases.

These finite resources restrict the capabilities of products based on the RSC-364. The product specification for the RSC-364 must be carefully crafted in consultation with Sensory to maximize the use of on-chip memory. Each application will have its own specific limitations, but the table below summarizes some useful guidelines for planning purposes. Not all of the maximums can be achieved in a single custom-masked RSC-364 design. For example, a recognition vocabulary of 40 words may limit the speech synthesis to substantially less than 25 seconds. NOTE: The RSC-364 (Custom Mask) column assumes *no external memory*.

Description	RSC-300	RSC-364 (Custom Mask) ¹
Capabilities:		
Speaker independent (SI) recognition	✓	✓
Speaker dependent (SD) recognition	✓	Limited
Speech synthesis and special sound effects	✓	✓
Speaker verification	✓	✓
Four-voice music generation	✓	Limited support
Voice record and playback	✓	Not supported ⁶
SI Recognition Capacity :		
Maximum number of words per recognition set ³	15	15
Total recognition vocabulary size in words, all sets	Unlimited	40 words ³
SD Recognition Capacity :		
Maximum number of words per recognition set ³	64 ²	6 ³ /64 ⁴
Total recognition vocabulary size in words, all sets	Unlimited	6 ³ /512 ⁴
Speaker Verification Capacity :		
Number of speakers identified per set ³	64 ²	1 ³ /64 ⁴
Synthesized Speech Capacity:		
Maximum total length of all messages	Unlimited	25 seconds ³
Music Synthesis Capacity		
Number of simultaneous independent musical voices	4	4
Number of musical octaves available	2-4 ⁵	2
Number of musical tunes available	Unlimited	6
Requirement for custom ROM masks:		
Custom-masked parts (RSC-364) are not stocked by Sensory	No Internal ROM	Custom masked ROM required

¹ Software for the RSC-364 (Custom Masked) applications may be completely developed and verified using the RSC-300/364 Development Kit and an external 64K ROM memory before committing to an RSC-364 custom ROM mask.

² Practical limitations to maintain accuracy above 95%.

³ Assumes the use of on-chip ROM/RAM only

⁴ Assumes external serial EEPROM memory.

⁵ Depends on choice of musical instrument.

⁶ Requires external storage for recordings.

2. RSC-300/364 HARDWARE SPECIFICATIONS

Architectural Overview of the RSC-300/364

The RSC-300/364 is a highly integrated device that combines:

- An 8-bit RISC microprocessor.
- On-chip ROM (64 Kbytes, RSC-364 only), Register RAM (448 bytes), Data RAM (2 Kbytes) and the ability to address off-chip RAM or ROM.
- Analog-to-digital converter, digital-to-analog converter, and a pulse width modulator.
- A microphone pre-amplifier

The RSC-300/364 has an external memory interface for accessing external RAMs, ROMs or other parallel memory devices. The RSC-364 also has an internal ROM that can be enabled or disabled (partially or fully) by pin inputs (signals -XMH, -XML; See figure 2).). When the internal ROM of the RSC-364 is disabled, its performance is identical to the RSC-300. With the RSC-364, the entire program must reside in the internal masked ROM. External memory can only be used to store data.

The 8-bit processor can directly access 448 on-chip general-purpose registers (RAM), and 32 additional Special Functions Registers (SFRs). The instruction set accessing these registers is completely symmetrical, allowing *movs*, arithmetic, and logical operations with any register as the destination. Two bi-directional ports provide 16 general-purpose I/O pins to communicate with external devices (See page 6). The RSC-300/364 has a high frequency (14.32 MHz) oscillator as well as a low frequency (32,768 Hz) oscillator suitable for timekeeping applications. The processor clock can be selected from either source, with a selectable divider value. Sensory's technology code requires the use of the 14.32 MHz clock. There are two programmable 8-bit counters / timers, one derived from each oscillator. A variety of wait state configurations allow fast code execution and easy interfacing to slow peripheral memories.

An inexpensive electret microphone connects directly to the microphone input of the RSC-300/364. The internal preamplifier converts the tiny microphone signal to a level suitable for Analog-to-Digital Conversion. (ADC), The RSC-300/364 uses a Sample and Hold (SH) circuit and ADC converter to convert the amplified analog speech signal into digital data. The chip may also be used with line-level inputs. The output audio signal of the RSC-300/364 is derived either from a DAC (Digital-to-Analog Converter) or a PWM (Pulse Width Modulator).

In addition to its on-chip ROM (RSC-364 only) and RAM, the RSC-300/364 has 8 data lines (D[7:0]) and 16 address lines (A[15:0]), along with associated control signals (-RDC, -RDD, -WRC, -WRD, -XML, -XMH) for interfacing to external memory. The memory control signals on the RSC-300/364 and the processor instruction set provide independent Code and Data spaces, allowing configuration of systems up to 192 Kbytes with no additional hardware decoding. The RSC-300/364 features 16 general-purpose I/O pins (Px.y) for product and memory bank control.

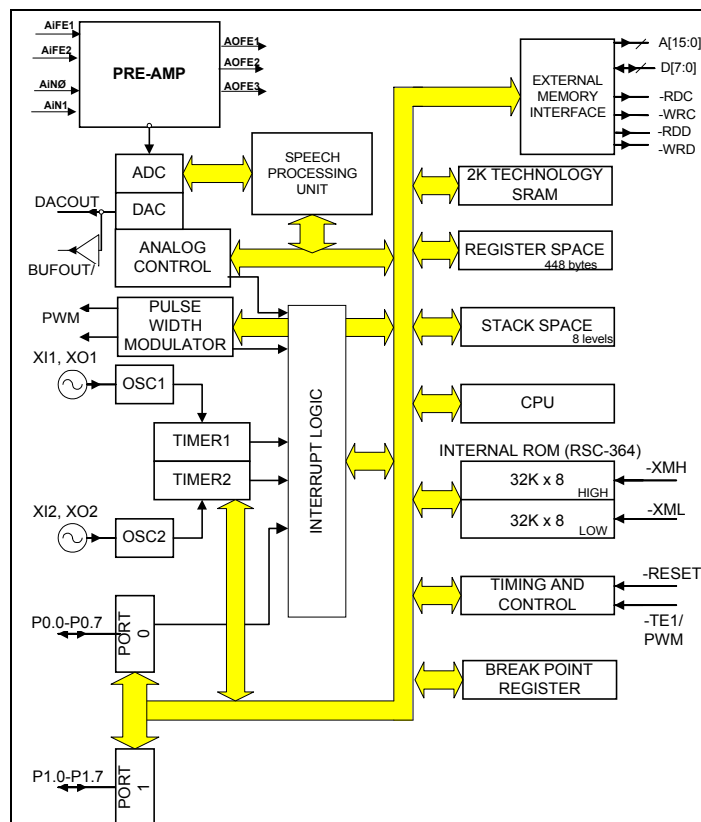


Figure 1 -- RSC-300/364 Block Diagram

3. USING THE RSC-300/364

Creating applications using the RSC-300/364 requires the development of electronic circuitry, software code, and speech/music data files. Software code for the RSC-300/364 can be developed by Sensory or by external programmers using the RSC-300/364 Development Kit. For more information about development tools and services, please contact Sensory, or visit www.SensoryInc.com. A typical product will require about \$0.30 - \$1.00 (in high volume) of additional components, in addition to the RSC-300/364.

The following sample circuit provides an example of how the RSC-300/364 might be used in a low volume consumer electronic product. The external ROM contains the application program, recognition weights, and speech data. A high volume application would likely use an RSC-364 (custom-masked ROM), and would not require the separate ROM.

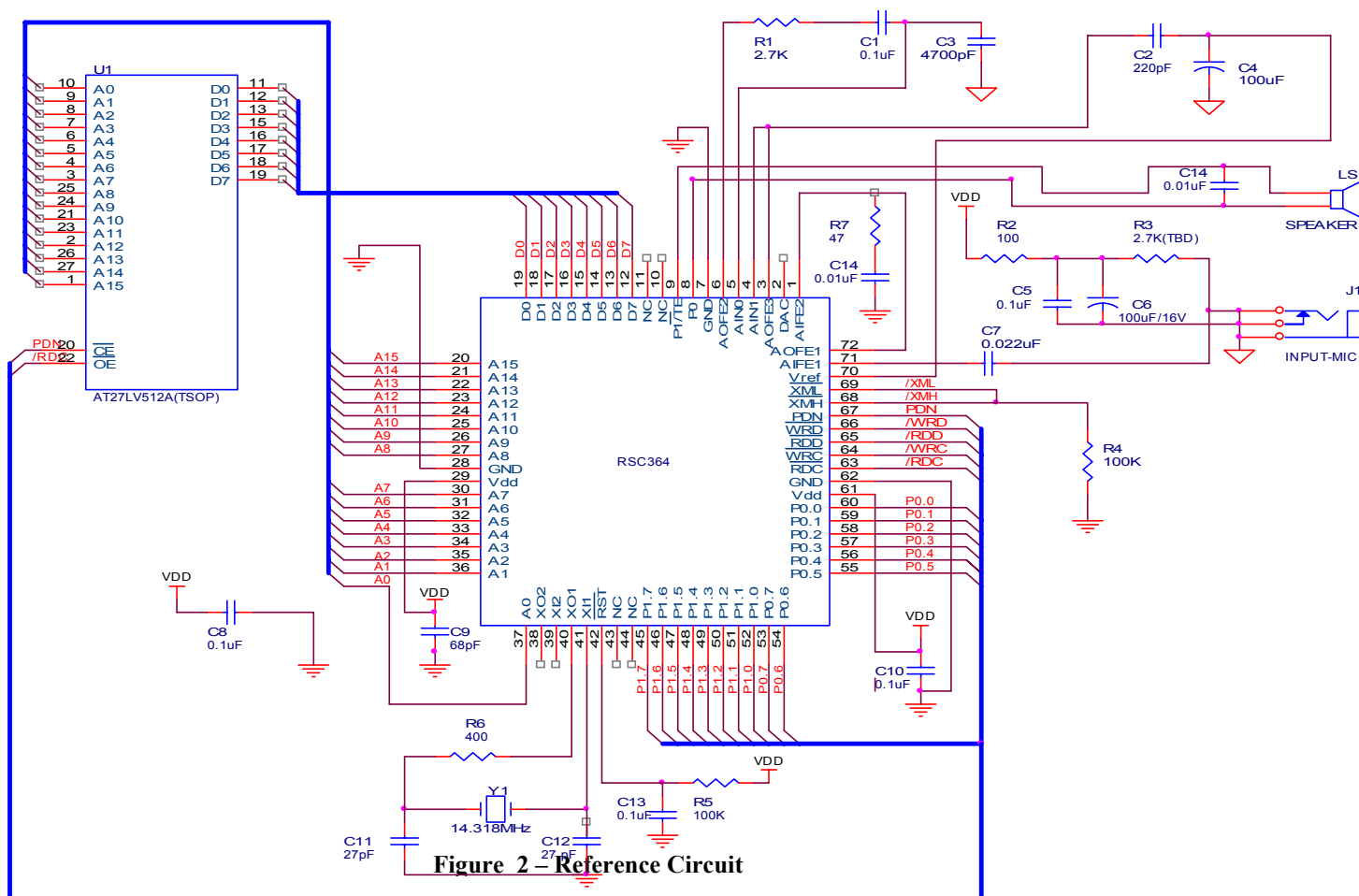


Figure 2 – Reference Circuit

Note: Applications using the PWM output may not meet FCC or CE (such as EN55022 Class A or B) standards for radiated emissions. For applications that must meet these standards, Sensory recommends turning off the PWM in the application software, and using an external audio power amplifier connected to the DAC output. Please refer to Sensory's Application Note 80-0105 "DAC Output" for sample circuits and design guidelines."

4. MEMORY ORGANIZATION

Internal ROM Memory (RSC-364 only)

Internal ROM is organized as two banks of 32 Kbytes each, both mapped into code space. Either of the two internal banks may be independently disabled using external inputs; input pin -XML disables the lower 32K [0000h-7FFFh] bank, while input pin -XMH disables the upper 32K [8000h-FFFFh] bank. When a bank is disabled, read accesses to it are directed to off-chip code space. In most applications the -XML and -XMH signals will both be grounded (for use with external ROM program memory) or both will be left floating (to use internal ROM memory for program memory). Write accesses to the code space are directed to external memory off-chip. Except for specific addresses in the last page of memory (described on page 6), read and write accesses to data space are always directed to external memory.

External Memory

The RSC-300/364 allows for extended message lengths and expanded program functionality by using external memory. There are 30 pins that provide an interface between the RSC-300/364 and external ROM or RAM. The 16 address line outputs, A[15:0], are shared for accesses to external code space or data space. The 8 data lines, D[7:0], are bi-directional, and are normally inputs except when there is a write to external memory. Refer to *MEMORY MAP* (on page 6) for details on accessing external code and data spaces through *movc* and *movx* instructions.

The RSC-300/364 uses the -RDC, -WRC, -RDD and -WRD signals to strobe data to or from memory or I/O devices. The -RDC and -WRC strobes are provided for accessing code space, while the -RDD and -WRD strobes are used to access data space. These four memory strobes are all active low (see page 23 for timing information). Using these strobes, the RSC-300/364 can directly access 64K of external code space (either internal or external) and 64K of external data space. External memory and I/O devices can reside in code space (RSC-300 only) or data space, as determined by the user application. Executable code *must* reside in code space; tables and other data may reside in code space or data space. Using I/O bits, additional external decoding can be used to bank select between multiple RAMs or ROMs. This method allows for external Data Space storage requirements larger than the combined 128K addressed directly by the RSC-300/364. Figure 3 below illustrates a typical large Data Space ROM-only memory configuration. The lowest 64K bank is addressed as Code Space, and the other 7 banks are addressed as Data Space. The ROM is selected by both -RDD and -RDC. Additional logic may be required to use both ROM and RAM/Flash in external Data Space.

Sensory's RSC-300/364 technology code requires code execution at 14.3 MHz with one wait state. External ROM Code memory speeds should be 120 nS or faster. External Data Space access speed may be independently controlled by the insertion of additional wait states for *movx* instructions, simplifying access to slow read/write devices.

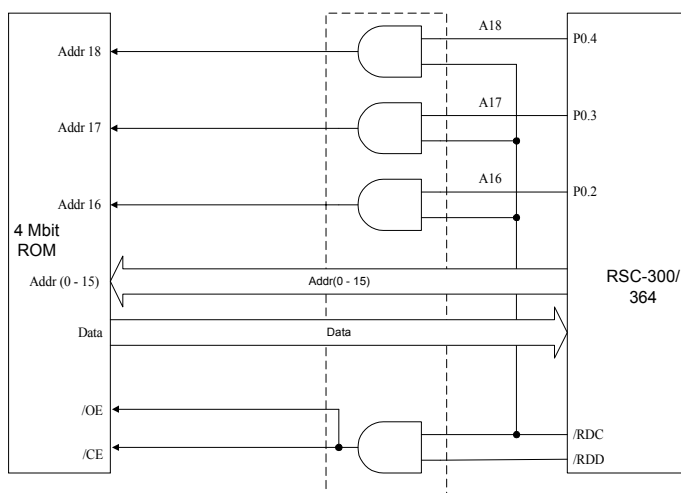


Figure 3 -- Large ROM Decoding

5. MEMORY MAP

The RSC-300/364 has three address spaces: Code Space, Data Space, and Register Space. Code space is typically ROM. Data space may be ROM, RAM, Flash, or other parallel read/write memories. Register space is limited to on-chip SRAM. The instruction set provides separate instructions for accessing each space. Executable code *must* reside in code space; tables and other data may reside in code space or data space. Register space is intended primarily for variables.

The internal ROM (RSC-364 only) and off-chip code space memory (RSC-300 only) may be accessed using *movc* instructions. The off-chip data space is accessed using *movx* instructions, while the on-chip register space is accessed using *mov* instructions. Writes to code space are always directed off-chip.

The internal ROM (RSC-364 only) is organized as two code space banks of 32 Kbytes each. The banks can be independently disabled using external inputs: the low bank (address range 0000h-7FFFh) can be disabled by asserting pin input -XML while the high bank (address range 8000h-FFFFh) can be disabled by asserting pin input -XMH. This feature may be used to expand External addressable code space beyond 64 Kbytes.

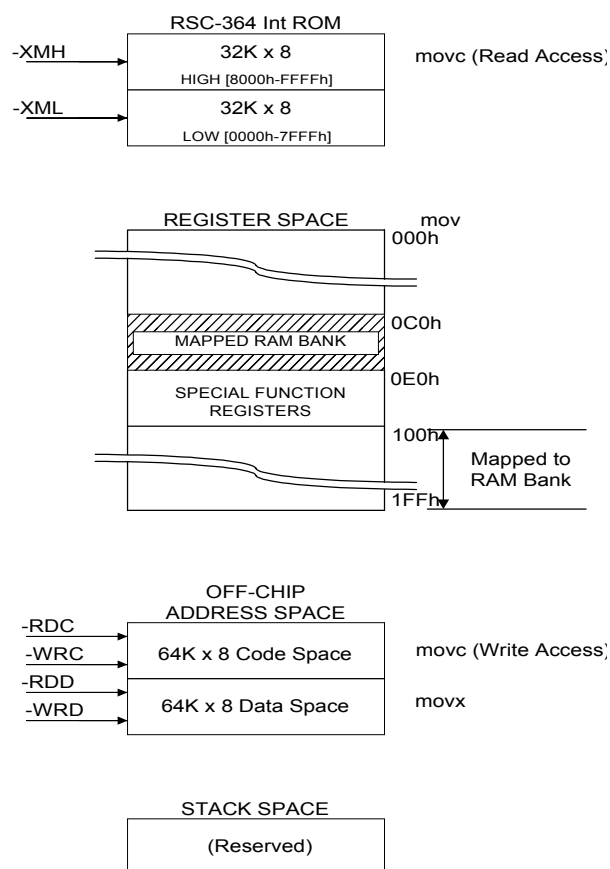
The SRAM register space supports 8-bit addresses, so only 256 bytes may be directly addressed. General-purpose registers are located between addresses 000h and 0BFh. A 32-byte bank of SFRs (special function registers) resides at addresses 0E0h-0FFh. The 32-byte bank at addresses 0C0h-0DFh may be mapped to any of the six lower 32-byte banks in page 0 (addresses 000h through 0BFh), or to eight additional 32-byte banks in page 1 (addresses 100h-1FFh). A special function register controls this mapping, providing a total of 448 bytes of SRAM register space. The RSC-300/364 also contains 2 Kbytes of internal data space RAM that is reserved for technology use.

Off-chip memory (and memory-mapped I/O) is accessed using a 16-bit address bus and an 8-bit data bus. Separate read / write strobes are generated for access to external code and data spaces. This allows the RSC-300/364 to directly access 64 Kbytes of external code memory and 64 Kbytes of external data memory. Bank switching is commonly implemented using I/O pins to select additional off-chip memory.

Certain addresses in the range of 0FF00h-0FFFFh of Data Space are mapped internally, so addresses in this last page of data space are not generally accessible. (See Page 45)

The RSC-300/364 allows software to adjust the speed of off-chip memory access. This allows using fast memory for performance needs or (if feasible) slower memory for cost savings. The off-chip memory access time can be stretched using wait states defined by the BANK register, and the software can dynamically change the wait state value depending on the particular memory or I/O peripheral. Wait states for external Data Space may be selected independently of Code Space wait states.

There is modest stack space on chip. The stack is required for interrupts and allows a limited number of nested calls. Programmers are encouraged to write inline code instead of making deeply-nested subroutine calls. The RSC-300/364 offers limited support for a software stack to allow more deeply nested calls or to store parameters, although this is not usually required. Macros using this software stack are accessible to developers.



6. GENERAL PURPOSE I/O

The RSC-300/364 has 16 general purpose I/O pins (P0.0-P0.7, P1.0-P1.7). Each pin can be programmed as an input with weak pull-up ($\sim 200\text{K}\Omega$ equivalent device); input with strong pull-up ($\sim 10\text{K}\Omega$ equivalent device); input without pull-up, or as an output. This is accomplished by having 32 bits of configuration registers for the I/O pins (Port Control Register A and Port Control Register B for ports 0 and 1). (*See page 27*)

After reset, all of the I/O pins are set to be inputs with weak pull-ups. Designers may make use of this start-up feature to assure enabling or disabling of particular functions controlled by I/O pins.

For electrical specifications regarding the general purpose I/O pins, please refer to D.C. CHARACTERISTICS. All I/O pins are diode clamped to V_{DD} and ground, and are capable of sinking up to 200 mA if V_{DD} is exceeded.

In addition to providing general purpose I/O, port 0 bit P0.0 can serve as an interrupt input (using IMR and IRQ registers). Any I/O pin may be used as a “wakeup” event when the chip is in “sleep” mode.

Typically some of the I/O pins may be used for extended address bits in larger systems having more than 64K of memory. This may be done by connecting port pins directly to Address bit 16 and higher pins of large memory devices.

7. INTERRUPTS

The RSC-300/364 allows for six interrupt sources, as selected by software. Each has its own mask bit and request bit in the IMR and IRQ registers respectively. The global interrupt enable flag, which enables or disables all interrupts, is located in the FLAGS registers. Bit assignments for the IMR and IRQ registers are listed on page 43. The following events can generate interrupts:

- Positive edge on Port 0, bit 0
- Overflow of Timer 1
- Overflow of Timer 2
- Two Sensory Proprietary functions
- Completion of PWM sample period

If an IRQ bit is set high and the corresponding IMR bit is set high and the global interrupt enable bit is set high, an interrupt will occur. Interrupts cannot be nested. The flags register is copied to a holding register and then the global interrupt enable is cleared, preventing subsequent interrupts until the IRET instruction is executed. The IRET instruction will restore the flags register from the holding register.

If the corresponding mask register bit is clear, the IRQ bit will not cause an interrupt. However, it can be polled by reading the IRQ register. IRQ bits can be cleared by writing a 0 to the corresponding bit at address 0FEh (the IRQ register). IRQ bits can *not* be set by writing to 0FEh. Writing a one is a no-op.

The IRQ bits must be cleared within the interrupt handler by an explicit write to the IRQ register rather than by an implicit interrupt acknowledge.

Important: clear interrupts this way:

```
mov    IRQ, #BITMASK    ; right
```

not this way:

```
and    IRQ, #BITMASK    ; wrong
```

The ‘and’ instruction is not atomic, it is a read-modify-write. If an interrupt occurs during an ‘and IRQ’ operation the interrupt will be cleared before it is seen, possibly disabling the interrupt until the system is reset.

Because you cannot set bits in the IRQ register, a ‘mov IRQ’ is a safe, effective, and atomic way to clear bits in the IRQ register. Use it the way you would use an ‘and’ in other registers.

Note: if Port 0.0 (the external IRQ) is set as an *output*, the external IRQ flag will be set if the output is driven from 0 to 1 under program control.

For each interrupt, execution begins at a different code space address:

Interrupt #0	Address 4
Interrupt #1	Address 8
Interrupt #2	Address 0Ch
Interrupt #3	Address 10h
Interrupt #4	Address 14h
Interrupt #5	Address 18h

Normally the instruction at the interrupt address is a jump to an Interrupt Service Routine (ISR). This jump is called a *vector*. The vectors located at each of these addresses are typically in ROM. .

8. RESET AND CLOCKS

Reset

The reset pin, -RESET, is an active low Schmitt trigger input. The reset pin is provided with hysteresis in order to facilitate power-on reset generation via an RC network. Reset is held internally for 10 msec after the -RESET input signal is de-asserted. This allows the oscillator to stabilize before enabling other processor subsystems. The -RESET signal must be asserted for a minimum of 2 clock periods.

Oscillators

Two independent oscillators in the RSC-300/364 provide a high-frequency clock and a 32 kHz time-keeping clock. The oscillator characteristics are as follows:

Oscillator #1	Pins XI1 and XO1	14.32 MHz
Oscillator #2	Pins XI2 and XO2	32,768 Hz

Oscillator #1 works with an external crystal, a ceramic resonator, or LC. Use of Oscillator #2 requires a crystal for precise timekeeping.

Each oscillator has an enable control. When disabled, the inverter is high-impedance, and a weak pull-up device (~100 K Ω) holds the inverter output high. Both oscillators are controlled by the Clock Control Register (CPU register 0E8h). By default, Oscillator #1 is enabled by reset, while Oscillator #2 is disabled by reset. The effect of reset therefore requires that Oscillator #1 be functional in all designs. The Clock Control Register also determines internal division of the CPU clock source (see below).

Each oscillator has an associated timer that is fully programmable. The RSC-300/364 timers are described in the following section.

Processor Clock

The RSC-300/364 uses a fully static core; the processor can be stopped (by removing the clock source) and restarted without causing a reset or losing contents of internal registers. Static operation is guaranteed from DC to 14.32 MHz. The processor clock is selected from either the Oscillator #1 output (gated by wake-up 10 mS delay) or the Oscillator #2 output, based on bit 2 of the Clock Control Register. This bit is cleared by reset, which selects Oscillator #1. It is the responsibility of the firmware not to select Oscillator #2 until both oscillators have been enabled and stabilized.

After source selection, the processor clock can be divided-down in order to limit power consumption. Bits 3 and 4 of the Clock Control Register determine the divisor for the processor clock. Between zero and seven wait states must also be selected for the processor clock. Wait states are inserted on reads or writes to all addresses except Register Space RAM and, under certain configurations, internal ROM (RSC-364 only).

Sensory technology code must run with a processor clock of 14.32 MHz, a clock divisor of one, and one wait state. This creates internal RAM cycles of 70 nsec duration and internal ROM (RSC-364 only) or external cycles of 140 nsec duration. Careful design of external decoding logic and close analysis of gate delays may allow operation with Code Space memories having 120 nsec access times. Additional wait states may be selected for external Data Space access.

9. TIMERS AND COUNTERS

The two independent oscillators of the RSC-300/364 provide counts to two internal timers. Each of the two timers consists of an 8-bit reload value register and an 8-bit up-counter. The reload register is readable and writeable by the

processor. The counter is readable with precaution taken against a counter change in the middle of a read. If the processor writes to the counter, the data is ignored. Instead, the counter is preset to the reload register value. That is, *any* write to a counter will cause it to be reloaded. This is the usual way of initializing the counter. When the timer overflows from FF to 00, a pulse is generated that sets IRQ #0 (timer #1) or IRQ #1 (timer #2). If the corresponding IMR bit is set and the Global Interrupt Bit is set, an interrupt will be generated. Instead of overflowing to 00, the counter is automatically reloaded on each overflow.

For example, if the reload value is 0FAh, the counter will count as follows:

0FAh, 0FBh, 0FCh, 0FDh, 0FEh, 0FFh, 0FAh, 0FBh etc.

The overflow pulse is generated during the period *after* the counter value was 0FFh.

Timer #2 may be used as a wakeup when the processor has powered-down.

Refer to the following registers for more information about using timers and counters:

T1R: Timer 1 Reload Register (page 34)
 T1V: Timer 1 Counter Register (page 34)
 T2R: Timer 2 Reload Register (page 35)
 T2V: Timer 2 Counter Register (page 35)

10. POWER DOWN AND WAKEUP OPERATION

The RSC-300/364 can be powered down through software by setting the PD bit (bit7) of the Clock Control Register (*See page 33*). Setting this bit halts the processor until a “wakeup” event clears the bit. The instruction that causes the power down event may also set or clear other bits in the Clock Control Register to enable or disable any of the clocks, and to select a clock to be used as the processor clock upon wakeup. A wakeup event can be generated from either of two sources:

- bit transition(s) of port 0 or port 1 pins, or
- an overflow pulse from timer 2.

For low power consumption, oscillator #1 (bit 0) and the FC clock (bit 5) should always be disabled during power down. Oscillator #2 (bit 1) must be enabled if the wakeup condition is a timer 2 event. If the wakeup event is an IO pin event, all clocks should be disabled for lowest power consumption.

If oscillator #1 is disabled during power down, *and* the selected processor clock source is oscillator #1, then a wakeup event will require that oscillator #1 be started and stabilized before its output can be used as the processor clock. The oscillator is started when the wakeup event clears bit 0 of the Clock Control Register, but the processor clock is delayed by 10 milliseconds to assure stability.

If the RSC-300/364 is powered down with oscillator #2 selected as the processor clock source, a timer2 wakeup event does not require a delay because it is assumed that oscillator #2 is running and stable. Due to long startup time for oscillator #2, the RSC-300/364 should not be powered down with oscillator #2 disabled *and* selected as the processor clock.

A wakeup event does not cause a reset. The processor, which was “stopped-in-its-tracks” when the PD bit was set, is restarted without loss of context.

11. ANALOG OUTPUTS

The RSC-300/364 offers two separate options for analog output. The DAC (Digital to Analog Converter) output provides a general-purpose 10-bit analog output that may be used for speech output (with the inclusion of an audio

amplifier), or that may be used for other purposes requiring an analog waveform. Many speech applications may require only driving a small speaker, however, and cost can be saved in these applications by using the Pulse Width Modulator (PWM) outputs of the RSC-300/364 instead of the DAC output.

Special Note: Applications using the PWM output may not meet FCC or CE (such as EN55022 Class A or B) standards for radiated emissions. For applications that must meet these standards, Sensory recommends turning off the PWM in the application software, and using an external audio power amplifier connected to the DAC output. Please refer to Sensory's Application Note 80-0105 "DAC Output" for sample circuits and design guidelines.

DAC Output

The digital-to-analog converter (DAC) provides its output through DACOUT, with an output impedance of $22\text{K}\Omega$. V_{DACOUT} can swing from 0V to V_{DD} . An external audio amplifier and optional volume control must be used to drive a speaker. Filtering above 5 kHz is recommended to provide the correct frequency response.

PWM Output

The two PWM outputs are designed for driving a 32-Ohm speaker at audio frequencies. These signals produce good quality audio with no additional components. These outputs are actually digital outputs that produce a series of high frequency pulses at varying rates that, when filtered by the mechanical dynamics of a speaker, create the effect of a continuously varying analog signal.

Although it is called a pulse *width* modulator, the RSC-300/364 actually incorporates a pulse *count* modulator: a programmable number of pulses is produced during each sample period. The two PWM outputs connect directly to the two speaker terminals. During operation, the first of the two signals will be at ground and the second will have a pulse train. The pulses will cause the speaker cone to push out (or pull in, depending on the wiring connections). If there are many pulses in a sample period, the speaker will push out a large amount; if there are few pulses, the speaker will push out just a little. Switching the pulse train to the first output and holding the second output at ground effectively reverses the polarity of the signals, so now the speaker will pull in. By controlling the number of pulses in each sample period and the output on which they appear, the speaker can be made to move in and out as required to reproduce an audio waveform.

The PWM0 pin is shared with the BUFOUT signal, and the PWM1 pin is shared with the -TE signal. At power-on, these pins are configured for non-PWM operation. The BUFOUT and -TE signals are both test signals that are typically not used during normal operation. When the Pulse Width Modulator is enabled via software, the pins are switched over to the PWM function. See page 46 for information about programming the PWM. (Typically the PWM is controlled by Sensory library code, so there is little need for application programs to alter it.)

12. HARDWARE DEBUG FEATURES

Special debugging hardware has been incorporated into the RSC-300/364 to assist developers in producing code quickly. The specialized circuitry provides the following features:

- A 16-bit break register that holds one ROM breakpoint address
- A TRAP bit in the Flags register (Address 0FFh) that enables or disables the breakpoint.
- A vector at 0FFF8h to which the processor is directed when the Program Counter equals the breakpoint address and the TRAP bit is enabled. The break occurs *before* execution of the instruction at the breakpoint address.
- A special page of Code Space (0FF00h-0FFFFh) for holding the resident debugger software. When this page is entered *via a break*, timers and interrupts are suspended. When execution resumes outside this page, timers and interrupts are restored. Entering this page by means other than a break has no special effect, and timers and interrupts continue to operate as normal.

- A vector at 0FFFCh to which the processor is directed if the -TE pin is held low at power-on. This provides a means for starting up in the debugger.

The debug circuitry, in conjunction with debug monitor software resident in the last page of Code ROM, allows examining the contents of Register, Code, and Data Space addresses. Register space contents may be modified, as may Data or External Code space addresses if implemented in a RSC-writeable device.

Additionally, the RSC-300/364 supports a 4-byte, bi-directional communication interface using external hardware registers. This interface provides one means for another computer to interact with the resident debug monitor.

Sensory expects to provide debugging support for the RSC-300/364 at some future date.

13. DESIGN CONSIDERATIONS

Speech recognition accuracy can be degraded by a number of factors. A common problem that causes accuracy degradation is noise: both electrical noise within the system and audio noise picked up by the microphone. A major innovation in the RSC-300/364 is the incorporation of an audio preamp circuit right on the chip. The signal from a typical electret microphone is of the order of a few millivolts, and an overall preamp gain of 200 or more is needed to make this signal useable by the RSC. The RSC-300/364 requires only a few external passive components to provide this amplification. Good grounding practice and elimination of crosstalk into the analog circuitry will further help ensure good recognition accuracy. Product design that encourages the user to speak loudly and close to the microphone helps attain a good signal-to-noise ratio.

Analog Design

The schematic in Figure 5 illustrates a reference input audio preamp design for use with the RSC-300/364. The microphone resistor shown as 2.7K has a large influence on the system gain, so the value will depend on the sensitivity of the microphone. The value of 2.7K is typical.

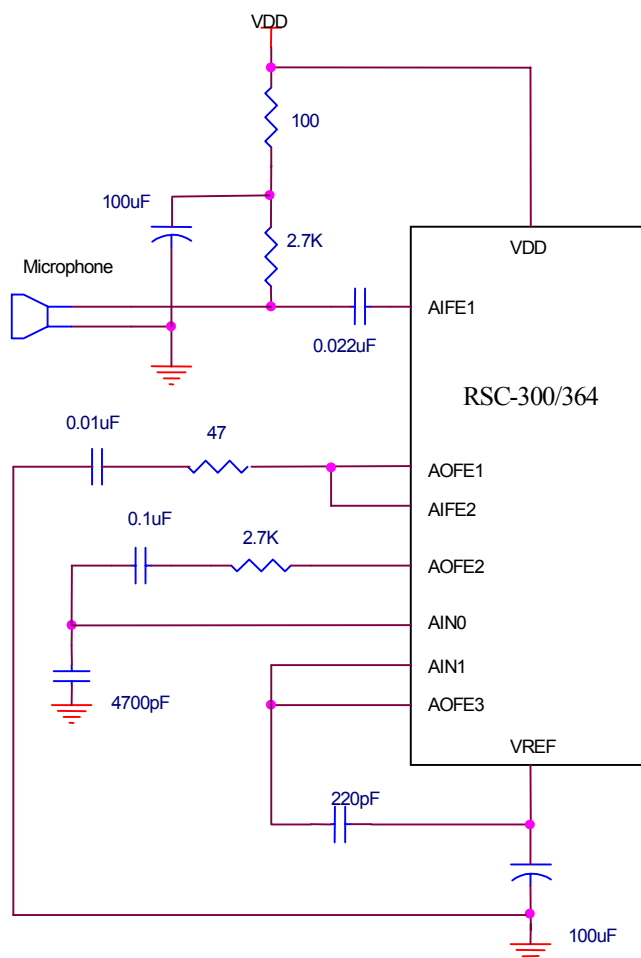


Figure 5 -- Preamplifier Schematic

Some applications may use the RSC-300/364 with input signals from a device other than an electret microphone. For assistance with such designs, contact Sensory.

PCB Design

A double-sided printed circuit board (PCB) with ground plane is recommended. The ground plane should cover the analog circuitry area and only be tied to ground near the RSC device. In order to reduce crosstalk, the analog and digital circuits should be physically separated as far as is practical. Take special care to keep high-speed clocked lines (e.g., address and data) away from the microphone components and traces.

A 0.1 μ F bypass capacitor should be installed immediately next to each digital IC and near the V_{DD} pins of the RSC chip. The bypass capacitors should be stacked or monolithic ceramic type, rated at 50 volts. If a three-terminal voltage regulator (such as a 7805) is used, tantalum bypass capacitors should be connected close to the regulator between the input/output pins and ground.

In a practical application using replaceable AA or AAA batteries, incorporating a protection diode in series with the power supply will avoid damage to the circuit if batteries are inserted with the wrong polarity.

If the RSC is used in a system with other digital clocks (switching power supplies, LCD driver, etc.) take special care to prevent these signal from getting into the audio circuitry of the RSC.

Locating and Mounting the RSC-300/364

The RSC-300/364 is supplied as a bare, tested die or in a 64 lead, 10 x 10 x 1.4 mm TQFP package. The die form may be wire bonded directly to the main PCB or, in some cases, may be bonded to a separate chip-on-board (COB) circuit board. In production this COB assembly may be functionally tested, then attached to the main board as a working module.

There are several methods of attaching the COB to the main board, and a careful choice should be made by the designer. The RSC-300/364 is a 72-pad device requiring good attaching methodology for correct operation. Since cost is always a consideration, COB boards may often be designed as single-sided PCBs.

The simplest way of attaching a single-sided COB to a main board is to lay it, chip side up, on the main board and make solder bridges from the main board up along the thickness of the COB to the electrical contacts on the top of the COB board. In production this is not a reliable technique.

A second technique is to use wires or pins to connect thru-holes between the main board and the COB. This reliable technique may be more time consuming.

A third technique is to put a hole in the main board at the center of the COB location and to mount the COB to the main board UPSIDE DOWN, that is, with the chip facing down into the hole. Then the COB and main boards may be soldered together. In this case, the orientation of the signal leads of the RSC-300/364 is different from that of the other arrangements.

14. OMNI-DIRECTIONAL MICROPHONE

Selecting a Suitable Microphone

For most applications, an inexpensive omni-directional electret capacitor microphone with a minimum sensitivity of -60 dB is adequate. In some applications, a directional microphone might be more suitable if the signal comes from a different direction than the audio noise. Since directional microphones have a frequency response that depends on their distance from the sound source, such microphones should be used with caution. For best performance, speech recognition products should be used in a quiet environment with the speaker's mouth in close proximity to the microphone. If the product is meant to be used in a noisy environment, care should be taken to design around the noise. Improving the signal-to-noise ratio will help make the product a success.

Design of Microphone Housing

Proper design and consistent manufacturing of the microphone housing is important, because improper acoustic positioning of the microphone will reduce recognition accuracy. This section describes several important considerations that must be carefully followed in designing the microphone mounting and housing. Many mechanical arrangements are possible for the microphone element, and some will work better than others. We recommend the following guidelines for the microphone housing:

FIRST: In the product, the microphone element should be positioned as close to the mounting surface as possible and should be fully seated in the plastic housing. There must be **NO** airspace between the microphone element and the housing. Having such an airspace can lead to acoustic resonance, which can reduce recognition accuracy.

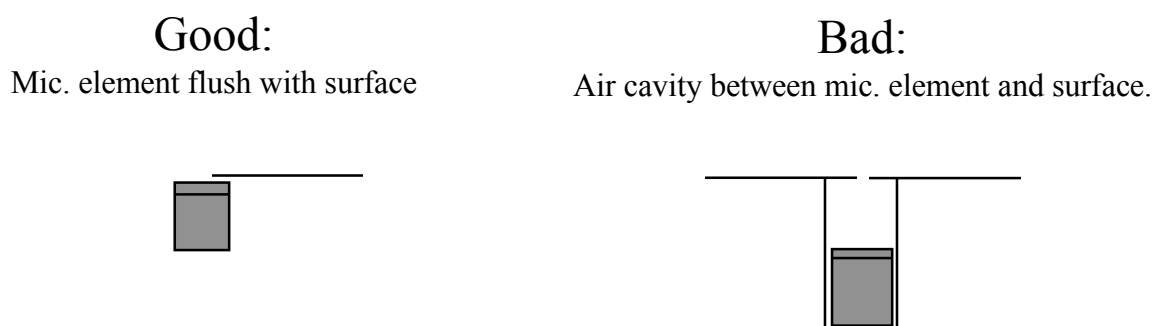


Figure 6 -- Microphone Mounting (1)

SECOND: The area in front of the microphone element must be kept clear of obstructions to avoid interference with recognition. The diameter of the hole in the housing in front of the microphone should be at least 5 mm. Any necessary plastic surface in front of the microphone should be as thin as possible, being no more than 0.7 mm if possible.

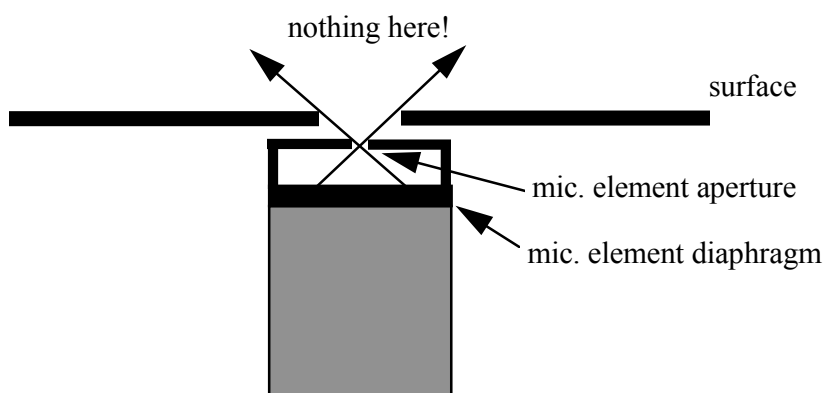


Figure 7 -- Microphone Mounting (2)

THIRD: The microphone should be acoustically isolated from the housing if possible. This can be accomplished by surrounding the microphone element with a spongy material such as rubber or foam. Mounting with a non-hardening adhesive such as RTV is another possibility. The purpose is to prevent auditory noises produced by handling or jarring the product from being “picked up” by the microphone. Such extraneous noises can reduce recognition accuracy.

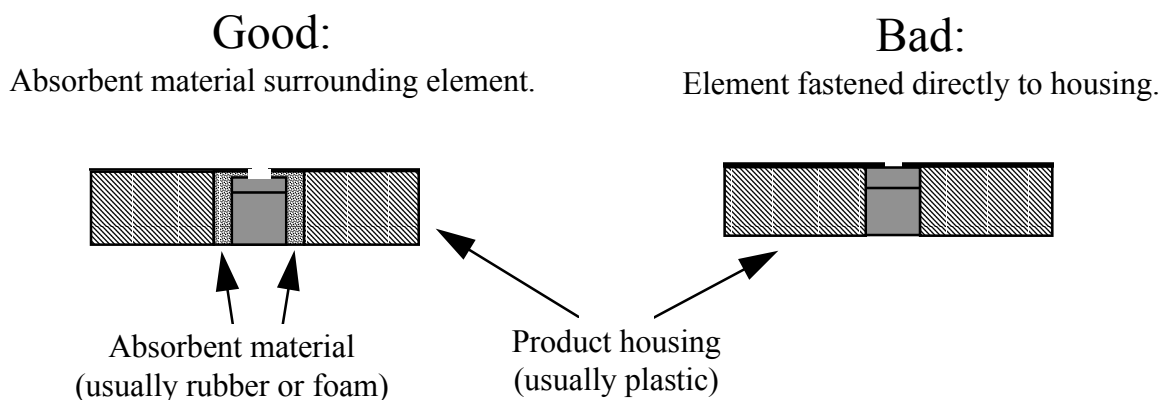


Figure 8 -- Microphone Mounting (3)

If the microphone is moved from 6 inches to 12 inches from the speaker's mouth, the signal power decreases by a factor of four. The difference between a loud and a soft voice can also be more than a factor of four. Thus, the recognizer must function over a wide dynamic range of input signal strength and it will have reduced recognition ability if the input signal either saturates the A/D converter or is too weak. The RSC-300/364 provides Automatic Gain Control (AGC) to partially compensate for a too-large or too-small speech signal. This AGC operation is performed inside the microphone preamplifier. If the AGC control range is exceeded, software can provide auditory feedback to the speaker about the voice volume. The product can achieve this by saying "please talk louder" or "please don't talk so loud."

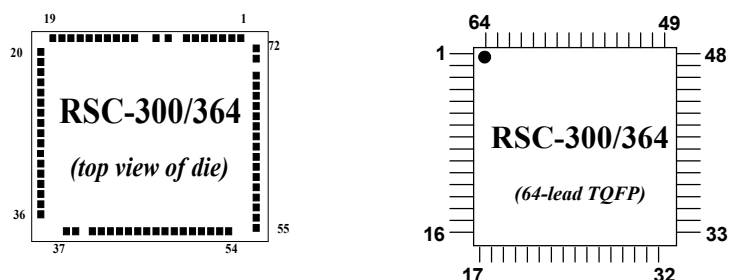
15. POWER CONSUMPTION AND POWER SUPPLY CONSIDERATIONS

In operation, the speech recognition circuit may draw a current of around 10 mA. If the system is powered on continuously to listen for a given word, it will drain a button battery in a few hours, or a large alkaline battery in several days. Thus, if the application requires that the recognizer be on all of the time, the system should operate from mains power. Conversely, if the product is designed to operate from batteries, it must usually remain in the low-power “sleep” mode most of the time, until it is occasionally awakened for a few seconds to recognize a word. The RSC-300/364 can be awakened from a button press or other I/O event, or from the countdown of the Osc2 timer. It cannot be awakened by a speech signal applied to the microphone input.

In order to conserve battery power, the product should be designed to be activated by the user each time it is required to recognize a word. A well-designed program will allow a few seconds after activation before attempting recognition in order to allow reference voltages to stabilize. A speech prompt is a good means of providing this delay. Current drain is higher during speech or music synthesis because power must be delivered to the speaker.

Mains-powered supply ripple should not exceed 5 millivolts measured between the VDD and GND terminals. When using mains power, regulated DC supplies will typically be required. Three-terminal voltage regulators (such as the 7805) are used commonly for this purpose and will provide a minimum of 47 dB of ripple rejection. The power dissipated by these regulators is a function of the voltage differential between the input and output terminals. For instance, powering a 7805 regulator at 9V and delivering 10 mAmp through it will dissipate 40 mWatts of power.

16. DIE BOND PAD AND QFP PIN DESCRIPTIONS



Name	Die Pad	QFP Pin	Description	I/O
A[15:0]	20-27, 30-37	1-8, 11-18	External Memory Address Bus	O
AIN0	5	52	Analog In, low gain. (range AGND to AVDD/2.)	I
AIN1	4	51	Analog In, hi gain (8X input amplitude of AIN0, same range)	I
AOFE1	72	49	Output of 1 st stage of preamplifier	O
AOFE2	6	53	Output of 2 nd stage of preamplifier	O
AOFE3	3	51	Output of 3 rd stage of preamplifier	O
AIFE1	71	48	Output of 1 st stage of preamplifier	I
AIFE2	1	49	Output of 2 nd stage of preamplifier	I
NC	10,11,43,44		Not Connected	-
PWM0	8	55	Pulse Width Modulator Output0	O
DACOUT	2	50	Analog Output (unbuffered).	O
D[7:0]	12-19	57-64	External Data Bus	I/O
Vss	7,28,62	9, 39,54	Vss	-
PDN	67	44	Power Down. Active high when powered down.	O
P1[7:0], P0[7:0]	45-52,53-60	22-29, 30-37	General Purpose Port I/O. Pin P0.0 can act as an external interrupt input. All I/O pins can act as “wake up” inputs.	I/O
/RDC	63	40	External Code Read Strobe	O
/RDD	65	42	External Data Read Strobe	O
/RESET	42	21	Reset	I
/TE1 or PWM1	9	56	Test Mode <i>or</i> Pulse Width Modulator Output1 (multiplexed)	I <i>or</i> O
VREF	70	47	Reference Voltage = Vdd/2 or Vdd/4. Depends on software	-
VDD	29,61	10,38	Supply Voltage	-
/WRC	64	41	External Code Write Strobe	O
/WRD	66	43	External Data Write Strobe	O
/XMH	68	45	External Hi-memory enable (low active)	I
/XML	69	46	External Low-memory enable (low active)	I
XO1	40	19	Oscillator 1 output (high frequency)	O
XI1	41	20	Oscillator 1 input	I
XO2	38	NA	Oscillator 2 output (32768 Hz)	O
XI2	39	NA	Oscillator 2 input	I

Note: Substrate should be connected to VSS

p	p	p	p	p	p	p	p	p	p	N	N	r	x	x	x	x	a
0	0	1	1	1	1	1	1	1	1	C	C	s	i	o	i	o	0
6	7	0	1	2	3	4	5	6	7			t	1	1	2	2	
												b					



18. RSC-300/364 DIE BONDING PAD LOCATIONS

pitch 123.85 (only for some pads)

Note: All locations are the center of the pad.

Pad #	South	X (µm)	Y (µm)	Pad #	East	X (µm)	Y (µm)	Pad #	North	X (µm)	Y (µm)	Pad #	West	X (µm)	Y (µm)
1	A1FF2	339.05	96.15	20	A15	3268.4	370.7	37	A0	2663.35	3211.6	55	P05	101.1	2513.5
2	DAQUT	462.9	96.15	21	A14	3268.4	494.55	38	XO2	2533.5	3211.6	56	P04	101.1	2389.65
3	AOFF3	586.75	96.15	22	A13	3268.4	618.4	39	XI2	2365.45	3211.6	57	P03	101.1	2265.8
4	AIN1	710.6	96.15	23	A12	3268.4	742.25	40	XO1	2235.65	3211.6	58	P02	101.1	2141.95
5	AIN0	834.45	96.15	24	A11	3268.4	866.1	41	XI1	2105.8	3211.6	59	P01	101.1	2018.1
6	AOFF2	958.3	96.15	25	A10	3268.4	989.95	42	RSTB	1975.9	3211.6	60	P00	101.1	1894.25
7	GND	1090	96.15	26	A9	3268.4	1113.8	43	NC	1830.65	3211.6	61	VDD	101.1	1764.85
8	P0/BF	1312.8	96.15	27	A8	3268.4	1237.65	44	NC	1700.8	3211.6	62	GND	101.1	1635
9	P1/T1	1665	96.15	28	GND	3268.4	1381.95	45	P17	1556.35	3211.6	63	RDCB	101.1	1504.85
10	NC	1870	96.15	29	VDD	3268.4	1511.8	46	P16	1432.5	3211.6	64	WRCE	101.1	1381
11	NC	1999.5	96.15	30	A7	3268.4	1652.1	47	P15	1308.65	3211.6	65	RDCB	101.1	1257.15
12	D7	2131.95	96.15	31	A6	3268.4	1775.95	48	P14	1184.8	3211.6	66	WRDB	101.1	1133.3
13	D6	2255.8	96.15	32	A5	3268.4	1899.8	49	P13	1060.95	3211.6	67	PDN	101.1	1009.45
14	D5	2379.65	96.15	33	A4	3268.4	2023.65	50	P12	937.1	3211.6	68	XMHB	101.1	885.6
15	D4	2503.5	96.15	34	A3	3268.4	2147.5	51	P11	813.25	3211.6	69	XMIB	101.1	761.75
16	D3	2627.35	96.15	35	A2	3268.4	2271.35	52	P10	689.4	3211.6	70	VREF	101.1	637.9
17	D2	2751.2	96.15	36	A1	3268.4	2395.2	53	P07	565.55	3211.6	71	A1FF1	101.1	470.05
18	D1	2875.05	96.15					54	P06	441.7	3211.6	72	AOFF1	101.1	346.2
19	D0	2998.9	96.15												

Notes:

- 1- Die Size: 3.3695 x 3.3127 mm (NOT including Scribe)
- 2- Scribe: 100x100 µm
- 3- Number of Probes: 72
- 5- There is 1 power supply: VDD(#29, #61).
- 6- Chip origin is at 0.0, 0.0 mm (Lower Left Hand Corner).
- 7- Pad size is 100µm x 100µm

19. ABSOLUTE MAXIMUM RATINGS

Any pin to GND	-0.1V to +6.5V
Operating temperature(T_O)	-20°C to +70°C
Soldering temperature	260°C for 10 sec
Power dissipation	1 W
Operating Conditions	-20°C to +70°C; $V_{DD}=2.4 - 5.25V$ $V_{SS}=0V$

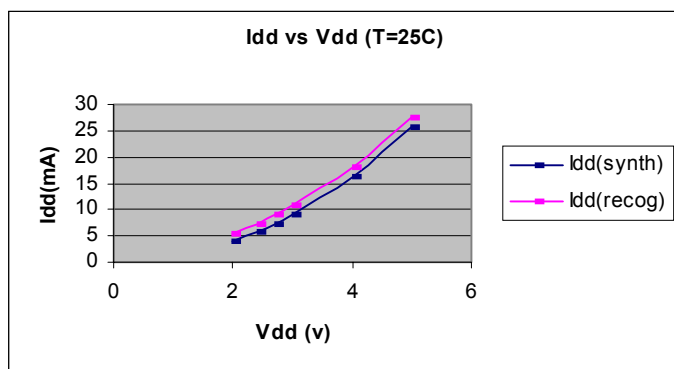
WARNING: *Stressing the RSC-300/364 beyond the “Absolute Maximum Ratings” may cause permanent damage. These are stress ratings only. Operation beyond the “Operating Conditions” is not recommended and extended exposure beyond the “Operating Conditions” may affect device reliability.*

20. D.C. CHARACTERISTICS

($T_O = -20^\circ\text{C}$ to $+70^\circ\text{C}$, $V_{DD} = 2.4V - 5.25V$)

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-0.1		0.75	V	
$V_{IH}(V_{CC}<3.6)$	Input High Voltage	$0.8 \cdot V_{DD}$		$V_{DD}+0.3$	V	
$V_{IH}(V_{CC}>3.6)$	Input High Voltage	3.0		$V_{DD}+0.3$	V	
V_{OL}	Output Low Voltage		0.3	$0.1 \cdot V_{DD}$	V	$I_{OL} = 2 \text{ mA}$
V_{OH}	Output High Voltage (I/O Pins)	$0.8 \cdot V_{DD}$	$0.9 \cdot V_{DD}$		V	$I_{OL} = -2 \text{ mA}$
I_{IL}	Logical 0 Input Current		<1	10	uA	$V_{SS} < V_{pin} < V_{DD}$
$I_{DD1}(V_{CC}=3.3V)$	Supply Current, Active		10	20	mA	Hi-Z Outputs
$I_{DD3}(V_{CC}=3.3V)$	Supply Current, Powerdown		1	10	uA	Hi-Z Outputs
R_{pu}	Pull-up resistance P0.0-P1.7 I/O Pins	5,80, Hi-Z	4.5,200, Hi-Z		k Ω	Selected with software
	/XML,/XMH		200		k Ω	Fixed

21. VDD vs. IDD



V_{CC} $I_{DD}(\text{synth})$ $I_{DD}(\text{recog})$
(No Load)

2.4	5.7	7.2
2.7	7.4	8.9
3	9.2	10.7
4	16.3	18
5	25.6	27.3

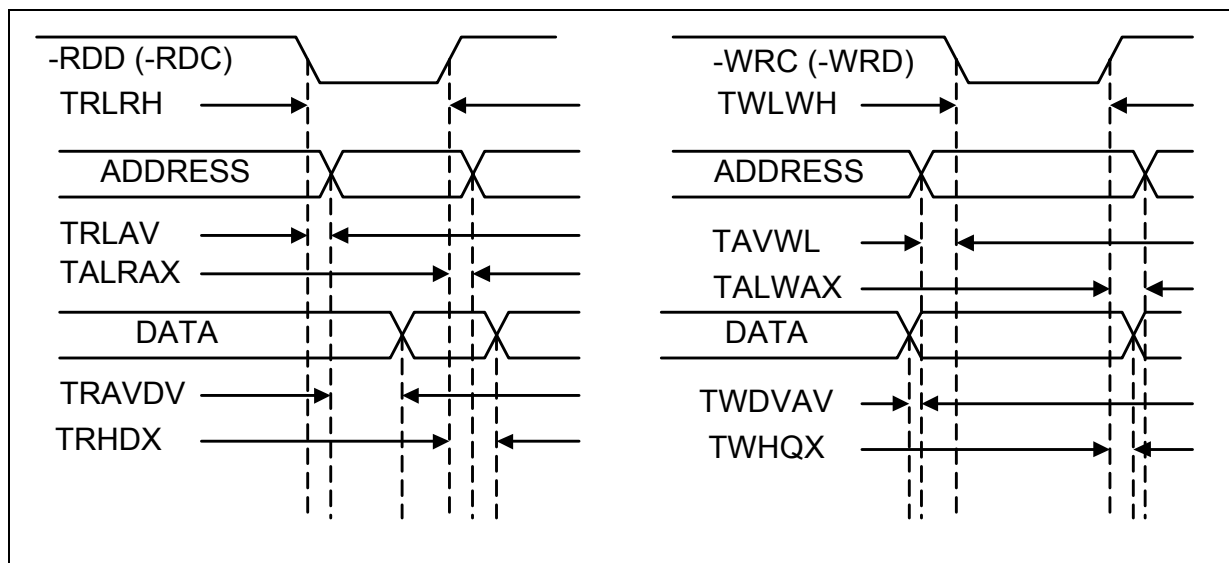
22. A.C. CHARACTERISTICS (EXTERNAL MEMORY ACCESSES)

(T_O = -20°C to +70°C, V_{DD} = 3.3V; load capacitance for outputs = 80 pF; Osc=14.32 MHz)

SYMBOL	PARAMETER	CPU=osc/1, 1 WS		CPU=osc/2, 0WS		UNITS
		MIN	MAX	MIN	MAX	
1/TCL1	Processor Clock frequency		14.32		7.16	MHz
TRLRH	-RDC (-RDD) Pulse Width		140		140	ns
TRLAV	-RDC (-RDD) Low to Address valid		5		5	ns
TALRAX	Address hold after -RDC (-RDD)		0		0	ns
TRAVDV	Address valid to Valid Data In		135		135	ns
TRHDX	Data Hold after -RDC (-RDD)	0		0		ns
TWLWH	-WRC (-WRD) Pulse Width		140		140	ns
TAVWL	Address Valid to -WRC (-WRD)	35		70		ns
TALWAX	Address Hold after -WRC (-WRD)	35		70		ns
TWDVAV	Write Data Valid to Address Valid		5		5	ns
TWHQX	Data Hold after -WRC (-WRD)	35		70		ns

23. TIMING DIAGRAMS

Note that the -RDC signal does not necessarily pulse for every read from code space, but may stay low for multiple cycles.



External Read Timing

External Write Timing

24. RSC-300/364 INSTRUCTION SET

The instruction set for the RSC-300/364 has 54 instructions comprising 10 move, 7 rotate, 11 branch, 11 register arithmetic, 9 immediate arithmetic, and 6 miscellaneous instructions. All instructions are 3 bytes or fewer, and no instruction requires more than 10 clock cycles to execute. The column “Cycles” indicates the number of clock cycles required for each instruction when operating with zero wait states. Wait states may be added to lengthen all accesses to external addresses or to the internal ROM (RSC-364 only), but not internal SRAM. The column “+Cycles/Waitstate” shows the number of additional cycles added for each additional wait state. The wait states used for *movx* instructions may be set independently of those used for *movc* or code fetches. This table assumes *movx* and fetch wait states are the same. Opcodes are in HEX.

MOVE Group Instructions

Register-indirect instructions accessing code (*movc*) or data (*movxspace*) locations use an 8-bit operand (“@source” or “@dest”) to designate an SRAM register pointer to the 16-bit target address. The “source” or “dest” pointer register must be at an even address. The LOW byte of the target address is contained at the pointer address, and the HIGH byte of the target address is contained at the pointer address+1. Register-indirect instructions accessing register (*mov, push, pop*) space locations use an 8-bit operand (“@source” or “@dest”) to designate an SRAM register pointer to the 8-bit target address. The carry, sign, and zero flags are not affected by *mov* instructions.

Instruction	Opcode	Operand 1	Operand 2	Description	Bytes	Cycles	+Cycles/ Waitstate
MOV	10	dest	source	register to register	3	5	3
MOV	11	@dest	source	register to register-indirect	3	5	3
MOV	12	dest	@source	register-indirect to register	3	6	3
MOV	13	dest	#immed	immediate data to register	3	4	3
MOVC	14	dest	@source	code space to register	3	7	4
MOVC	15	@dest	source	register to code space	3	8	4
MOVX	16	dest	@source	data space to register	3	7	4
MOVX	17	@dest	source	register to data space	3	8	4
POP	18	dest	@++source	register to register data stack pop (source pre- incremented)	3	10	3
PUSH	19	@dest--	source	register to register data stack push (dest post- decremented)	3	9	3

ROTATE Group Instructions

Rotate group instructions apply only directly to register space SRAM locations. The carry flag is affected by these instructions, but the sign and zero flags are unaffected.

Instruction	Opcode	Operand 1	Operand 2	Description	Bytes	Cycles	+Cycles/ Waitstate
RL	30	dest	-	rotate left, c set from b7	2	5	2
RR	31	dest	-	rotate right, c set from b0	2	5	2
RLC	32	dest	-	rotate left through carry	2	5	2
RRC	33	dest	-	rotate right through carry	2	5	2
SHL	34	dest	-	shift left, c set from b7,	2	5	2
SHR	35	dest	-	shift right, c set from b0,	2	5	2
SAR	36	dest	-	shift right arithmetic, c set	2	5	2

BRANCH Group Instructions

The branch instructions use direct address values rather than offsets to define the target address of the branch. This implies that binary code containing branches is not relocatable. However, object code produced by Sensory's assembler contains address references that are resolved at link time, so .OBJ modules *are* relocatable. The indirect jump instruction uses an 8-bit operand (“@dest”) to designate an SRAM register pointer to the 16-bit target address. The “dest” pointer register must be at an even address. The LOW byte of the target address is contained at the pointer address, and the HIGH byte of the target address is contained at the pointer address+1.

Instruction	Opcode	Operand 1	Operand 2	Description	Bytes	Cycles	+Cycles/ Waitstate
JC	20	dest low	dest high	jump on carry = 1	3	3	3
JNC	21	dest low	dest high	jump on carry = 0	3	3	3
JZ	22	dest low	dest high	jump on zflag = 1	3	3	3
JNZ	23	dest low	dest high	jump on zflag = 0	3	3	3
JS	24	dest low	dest high	jump on sflag = 1	3	3	3
JNS	25	dest low	dest high	jump on sflag = 0	3	3	3
JMP	26	dest low	dest high	jump unconditional	3	3	3
CALL	27	dest low	dest high	direct subroutine call	3	3	3
RET	28	-	-	return from call	1	2	1
IRET	29	-	-	return from interrupt	1	2	1
JMPR	2A	@dest	-	jump indirect	2	4	2

MISCELLANEOUS Group Instructions

Instruction	Opcode	Operand 1	Operand 2	Description	Bytes	Cycles	+Cycles/ Waitstate
NOP	00	-	-	no operation	1	2	1
CLC	01	-	-	clear carry	1	2	1
STC	02	-	-	set carry	1	2	1
CMC	03	-	-	complement carry	1	2	1
CLI	04	-	-	disable interrupts	1	2	1
STI	05	-	-	enable interrupts	1	2	1

ARITHMETIC/LOGICAL Group Instructions

Arithmetic and logical group instructions apply only to register space SRAM locations. The results of the instruction are always written directly to the SRAM “dest” register. All but the INCrement and DECrement instructions have both register source and immediate source forms.

In each of the following instructions the sign and zero flags are updated based on the result of the operation. The carry flag is updated by the arithmetic operations (ADD, ADC, SUB, SUBC, CP, INC, DEC) but it is *not* affected by the logical operations (AND, TM, OR, XOR). Note: the carry is set **high** by SUB, CP, SUBC, DEC when a borrow is generated.

Instruction	Op code	Operand 1	Operand 2	Description	Bytes	Cycles	+Cycles/ Waitstate
AND	40	dest	source	logical and	3	6	3
TM	41	dest	source	like AND, destination unchanged	3	6	3
OR	42	dest	source	logical or	3	6	3
XOR	43	dest	source	exclusive or	3	6	3
SUB	44	dest	source	subtract	3	6	3
CP	45	dest	source	like SUB, destination unchanged	3	6	3
SUBC	46	dest	source	subtract w/carry	3	6	3
ADD	47	dest	source	add	3	6	3
ADC	48	dest	source	add w/carry	3	6	3
INC	49	dest	-	increment	2	5	2
DEC	4A	dest	-	decrement	2	5	2
AND	50	dest	#immed	logical and	3	5	3
TM	51	dest	#immed	like AND, destination unchanged	3	5	3
OR	52	dest	#immed	logical or	3	5	3
XOR	53	dest	#immed	exclusive or	3	5	3
SUB	54	dest	#immed	subtract	3	5	3
CP	55	dest	#immed	like SUB, destination unchanged	3	5	3
SUBC	56	dest	#immed	subtract w/carry	3	5	3
ADD	57	dest	#immed	add	3	5	3
ADC	58	dest	#immed	add w/carry	3	5	3

25. RSC-300/364 SPECIAL FUNCTION REGISTER (SFR) SUMMARY

This section describes the registers located in addresses 0E0h through 0FFh of the register space. These special function registers (SFRs) are generally used for configuration control and system level functions. In many cases an applications programmer might need to access these registers only to initialize the ports. Since the SFRs are extensively used by Sensory's library functions, thorough understanding is essential before changing the contents of these registers. Some registers in this address range are for the exclusive use of Sensory Technology functions and are not further described..

The Symbol shown is the name recognized by the assembler for the associated register.

Symbol	Address	Register Name	Type	See Page
P0OUT	0E0h	Port 0 Output Register	R/W	28
P1OUT	0E1h	Port 1 Output Register	R/W	28
P0IN	0E2h	Port 0 Input Register	Read	29
P1IN	0E3h	Port 1 Input Register	Read	29
P0CTLA	0E4h	Port 0 Control Register A	R/W	30
P1CTLA	0E5h	Port 1 Control Register A	R/W	31
P0CTLB	0E6h	Port 0 Control Register B	R/W	30
P1CTLB	0E7h	Port 1 Control Register B	R/W	31
CKCTL	0E8h	Clock Control Register	R/W	32
WAKE0	0E9h	Wakeup configuration, Port 0	R/W	33
WAKE1	0EAh	Wakeup configuration, Port 1	R/W	33
T1R	0EBh	Timer 1 Reload	R/W	34
T1V	0ECh	Timer 1 Counter	R/W	34
T2R	0EDh	Timer 2 Reload	R/W	35
T2V	0EEh	Timer 2 Counter	R/W	35
ANCTL	0EFh	Analog Control Register	R/W	36
STKPTRS	0F6h	Stack Read and Write Pointers	R/W	38
OSC1EXT	0F7h	Oscillator 1 Extended Functions	R/W	39
BRKLO	0F8h	Break Address Low Byte	R/W	40
BRKHI	0F9h	Break Address High Byte	R/W	40
DAC	0FAh	DAC Hold Register	R/W	41
BANK	0FCh	RAM Bank Select Register	R/W	42
IMR	0FDh	Interrupt Mask Register	R/W	43
IRQ	0FEh	Interrupt Request Register	R/W	43
FLAGS	0FFh	Flags Register	R/W	44

Port 0 Output Register (Address 0E0h)

msb		P0OUT						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used to write values to Port 0. Values written to this register affect bits that have been configured as outputs. Bits that have been configured as inputs are not affected.

Bit Description:

P0OUT: Output bits D0 through D7
 Initialization: All bits cleared to 0 upon reset
 Read Access: Read output bits from Port 0
 Write Access: Write output bits to Port 0
 Also refer to: P0CTLA, P0CTLB

Port 1 Output Register (Address 0E1h)

msb		P1OUT						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used to write values to Port 1. Values written to this register affect bits that have been configured as outputs. Bits that have been configured as inputs are not affected.

Bit Description:

P1OUT: Output bits D0 through D7
 Initialization: All bits cleared to 0 upon reset
 Read Access: Read output bits from Port 1
 Write Access: Write output bits to Port 1
 Also refer to: P1CTLA, P1CTLB

Port 0 Input Register (Address 0E2h)

msb		P0IN						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used to read values from Port 0.

Bit Description:

P0IN: Input bits D0 through D7
Initialization:
Read Access: Input bits from Port 0
Write Access:
Also refer to: P0CTLA, P0CTLB

Port 1 Input Register (Address 0E3h)

msb		P1IN						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used to read values from Port 1.

Bit Description:

P1IN: Input bits D0 through D7
Initialization:
Read Access: Input bits from Port 1
Write Access:
Also refer to: P1CTLA, P1CTLB

Port 0 Control Register A (Address 0E4h)

msb		P0CTLA						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used with P0CTLB to control the function of the general-purpose port 0.

Bit Description:**P0CTLA:**

Initialization: All bits cleared to 0 upon reset

Read Access:

Write Access:

Also refer to: P0CTLB

Port 0 Control Register B (Address 0E6h)

msb		P0CTLB						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used with P0CTLA to control the function of the general-purpose port 0.

Bit Description:**P0CTLB:**

Initialization: All bits cleared to 0 upon reset

Read Access:

Write Access:

Also refer to: P0CTLA

The control registers A and B together control the function of the general-purpose port 0:

B	A	Function
0	0	Input - Weak Pull-up
0	1	Input - Strong Pull-up
1	0	Input - No pull-up
1	1	Output

For example, if register P0CTLB bit 4 is set high, and register P0CTLA bit 4 is low, then pin P0.4 is an input without a pull-up device.

Port 1 Control Register A (Address 0E5h)

msb		P1CTLA						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used with P1CTLB to control the function of the general-purpose port 1.

Bit Description:**P1CTLA:**

Initialization: All bits cleared to 0 upon reset

Read Access:

Write Access:

Also refer to: P1CTLB

Port 1 Control Register B (Address 0E7h)

msb		P1CTLB						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register is used with P1CTLA to control the function of the general-purpose port 1.

Bit Description:**P1CTLB:**

Initialization: All bits cleared to 0 upon reset

Read Access:

Write Access:

Also refer to: P1CTLA

The control registers A and B together control the function of the general-purpose port 1:

B	A	Function
0	0	Input - Weak Pull-up
0	1	Input - Strong Pull-up
1	0	Input - No pull-up
1	1	Output

For example, if register P1CTLB bit 3 is cleared, and register P1CTLA bit 3 is set high, then pin P1.3 is an input with a strong pull-up.

Clock Control Register (Address 0E8h)

msb		CKCTL						lsb
PD	T2	FC	CD1	CD0	PCS	DO2	EO1	

This register is used to enable the two oscillators, to select the processor clock source and the internal clock divider, and to enable some sleep/wakeup functions. Sensory's library code may initialize specific settings for this register, so it should be changed only with care.

Bit Description:

CKCTL.0: EO1
 0: Enable Oscillator #1 inverter
 1: Disable Oscillator #1
 Cleared by reset or (wakeup & (PCS=0))

CKCTL.1: DO2
 0: Disable Oscillator #2 inverter
 1: Enable Oscillator #2 inverter
 Cleared by reset.

CKCTL.2: PCS
 0: Processor clock source = Oscillator 1
 1: Processor clock source = Oscillator 2
 Cleared by reset.

CKCTL.4-CKCTL.3: CD1-CD0
 Select processor clock divisor. The processor clock rate is the fraction of the source clock shown.

CD1	CD0	Division
0	0	1/2
0	1	1/1
1	0	1/8
1	1	1/256

Cleared by reset.

CKCTL.5: FC
 0: Disable reserved function clock
 1: Enable reserved function clock
 Cleared by reset.

CKCTL 6: T2
 0: Timer #2 overflow does not cause wakeup
 1: Timer #2 overflow causes wakeup
 Cleared by reset.

CKCTL.7: PD
 0: Processor is in operational state
 1: Processor is in PowerDown state ("sleep")
 Cleared by reset and wakeup.

Port 0 Wakeup Configuration Register (Address 0E9h)

msb		WAKE0						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

The Wake0 register controls the enabling of I/O events on port0 that can cause a processor “wakeup”. These wakeup events are recognized independently of whether the processor is running or stopped. The processor may also be waked up by countdown of Timer2. For a general description, see Power Down and Wakeup Operation, page 10.

Bit Description:

WAKE0.7-WAKE0.0	Wake Enable
0:	Wakeup is not enabled on corresponding P0 port bit
1:	Wakeup is ENABLED on corresponding P0 port bit

Each bit, if set, enables the corresponding general purpose I/O pin of ports P0.0-P0.7 to generate a wake-up event. If enabled, the polarity of the wake-up trigger is determined by the corresponding bit in the output register: if the input matches the output register, a wake-up even is generated. It is assumed that the general-purpose pin is configured to be an input.

Each bit, if clear, prevents the corresponding general-purpose I/O pin of ports P0.0-P0.7 from generating a wake-up event.

Initialization: All bits cleared to 0 upon reset
 Also refer to: WAKE1, P0IN, P0CTLA, P0CTLB

Port 1 Wakeup Configuration Register (Address 0EAh)

msb		WAKE1						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

The Wake1 register controls the enabling of I/O events on port1 that can cause a processor “wakeup”. These wakeup events are recognized independently of whether the processor is running or stopped. For a general description, see Power Down and Wakeup Operation, page 10.

Bit Description:

WAKE1.7-WAKE1.0	Wake Enable
0:	Wakeup is not enabled on corresponding P1 port bit
1:	Wakeup is ENABLED on corresponding P1 port bit

Each bit, if set, enables the corresponding general-purpose I/O pin of ports P1.0-P1.7 to generate a wake-up event. If enabled, the polarity of the wake-up trigger is determined by the corresponding bit in the output register: if the input matches the output register, a wake-up even is generated. It is assumed that the general-purpose pin is configured to be an input.

Each bit, if clear, prevents the corresponding general purpose I/O pin of ports P1.0-P1.7 from generating a wake-up event.

Initialization: All bits cleared to 0 upon reset
 Also refer to: WAKE0, P1IN, P1CTLA, P1CTLB

Timer 1 Reload Register (Address 0EBh)

msb		T1R						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register contains an 8-bit reload value for timer 1. The reload register is readable and writeable by the processor. When the timer overflows from FF to 00, a pulse is generated that sets IRQ #0 (timer #1).

Bit Description:**T1R:**

Initialization: All bits cleared to 0 upon reset

Read Access: Timer #1 Counter Reload (2's complement of period)

Write Access: Timer #1 Counter Reload (2's complement of period)

Also refer to: T1V

Timer 1 Counter Register (Address 0ECh)

msb		T1V						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

Timer 1 counter is a read-only register. If the processor writes to the counter, the data is ignored, and the counter is preset to the reload register value from T1R. Instead of overflowing to 00, the counter is automatically reloaded on each overflow.

For example, if the reload value is 0FAh, the counter will count as follows:

0FAh, 0FBh, 0FCh, 0FDh, 0FEh, 0FFh, 0FAh, 0FBh etc.

The overflow pulse is generated during the period *after* the counter value was 0FFh.

The input clock for Timer 1 is always generated from Oscillator #1, gated by the wake-up delay, gated by bit 7 of the Clock Control Register, CKCTL.7 flag = 0, then divided by 16. For normal operation with a 14.3 MHz crystal, Timer 1 counts at a rate of 0.895 MHz. Thus the longest duration that can be directly timed is $255/(0.895 \text{ MHz}) = 285$ microseconds. If the T1P bit of the Oscillator 1 Extension register is set (OSC1EXT.6=1), an additional division by 2 is performed.

Bit Description:**T1V:**

Initialization: All bits cleared to 0 upon reset

Read Access: Timer #1 current counter value

Write Access: Force asynchronous load of counter from reload register

Also refer to: T1R, OSC1EXT

Timer 2 Reload Register (Address 0EDh)

msb		T2R						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

This register contains an 8-bit reload value for timer 2. The reload register is readable and writeable by the processor. When the timer overflows from FF to 00, a pulse is generated that sets IRQ #1 (timer #2).

Bit Description:**T2R:**

Initialization: All bits cleared to 0 upon reset
 Read Access: Timer #2 Counter Reload (2's complement of period)
 Write Access: Timer #2 Counter Reload (2's complement of period)
 Also refer to: T2V, CKCTL

Timer 2 Counter Register (Address 0EEh)

msb		T2V						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

Timer 2 counter is a read-only register. If the processor writes to the counter, the data is ignored, and the counter is preset to the reload register value from T2R. Instead of overflowing to 00, the counter is automatically reloaded on each overflow.

For example, if the reload value is 0FAh, the counter will count as follows:

0FAh, 0FBh, 0FCh, 0FDh, 0FEh, 0FFh, 0FAh, 0FBh etc.

The overflow pulse is generated during the period *after* the counter value was 0FFh.

The input clock for timer #2 is generated from Oscillator #2 divided by 128. With typical operation with a 32,768 Hz crystal for Oscillator #2, the count rate for Timer 2 is 256 Hz. When T2R is set to zero, T2R overflows once per second

The processor can be configured to “wakeup” from the powerdown state on an overflow of the timer2 counter. If Oscillator 1 is turned off during this time, the only current consumption is due to the slow 32 kHz Oscillator2 and counter circuits.

Bit Description:**T2V:**

Initialization: All bits cleared to 0 upon reset
 Read Access: Timer #2 current counter value
 Write Access: Force asynchronous load of counter from reload register
 Also refer to: T2R, CKCTL

Analog Control Register (Address 0EFh)

msb		ANCTL						lsb
MD		LS1	LS0	DM	BE	DE	M	

The analog control register configures the A/D and D/A. Since the RSC analog signals are normally dedicated to functions associated with Sensory's library code, there is seldom need for applications programs to access this register.

Bit Description:

ANCTL.7: Mode Bit (MD)

If the mode bit is 0, the other bits are as follows:

ANCTL.0: M
 1: ADC Mode, comparator powered-up.
 0: DAC Mode, comparator powered-down.
 Cleared by reset.

ANCTL.1: DE
 1: Enable analog output DACOUT
 0: Disable analog output DACOUT
 Cleared by reset.

ANCTL.2: BE
 1: Enable buffered output BUFOUT.
 0: Disable buffered output BUFOUT.
 Cleared by reset.

ANCTL.3: DM
 0: D/A is full-scale.
 1: D/A is half-scale.
 Cleared by reset.

ANCTL.4: LS0
 Provides LSB for full-scale D/A mode.
 Cleared by reset.

ANCTL.5: LS1
 Provides LSB for half-scale D/A mode, and second to LSB for full-scale D/A mode.
 Cleared by reset.

ANCTL.6: Reserved

Analog Control Register (Address 0EFh) (continued)

If the mode bit is 1, the other bits are as follows:

ANCTL.0-ANCTL.1:		Control the inverter strength of the 32,768 Hz oscillator
Bit 1	Bit 0	Strength
0	0	5 μ A
0	1	10 μ A
1	0	20 μ A
1	1	40 μ A

ANCTL.2-ANCTL.3:		Control the output resistor of the 32,768 Hz oscillator
Bit 3	Bit 2	Resistance
0	0	50 K Ω
0	1	100 K Ω
1	0	200 K Ω
1	1	400 K Ω

ANCTL.4-ANCTL.6: Reserved

When reading from the Analog Control Register, the side containing the 32,768 Hz oscillator control parameters is **not** read, while the other side is read.

Read Access:

Write Access:

Initialization: On reset, both sides of the Analog Control Register are set to zero.

Also refer to:

Stack Pointers Register (Address 0F6h)

msb		STKPTRS						lsb	
	WR2	WR1	WR0		RD2	RD1	RD0		

The RSC-300/364 has an 8-level hardware stack. This register contains the read and write pointers for the stack. Access to these registers is normally required only by a debugger program.

Bit Description:

STKPTRS.7 Reserved

STKPTRS6-4 Stack Write Pointer. Contains the 3-bit stack address where the next stack write will occur.

STKPTRS.3 Reserved

STKPTRS.2-0 Stack Read Pointer. Contains the 3-bit stack address where the next stack read will occur.

Oscillator1 Extension Register (Address 0F7h)

msb		OSC1EXT						lsb
IRW	T1P	MX	WX4	WX3	WX2	WX1	WX0	

This register controls a variety of extended use options for prescalers and extra wait states derived from Oscillator 1.

Bit Description:

OSC1EXT[4:0] WX4-WX0 Control the number of wait states inserted for *movx* instructions when OSC1EXT.5=1. This allows simplified operation with slow external read/write memories. If no wait states are used, *movx* access duration is one clock. Controlled program delays in the microsecond range can be obtained by setting the desired number of wait states and executing a dummy *movx*.

0 0 0 0 0 Use zero additional wait states (70 nsec at 14.3 MHz)
 1 1 1 1 1 Use 31 additional wait states (2.24 μ sec at 14.3 MHz).

OSC1EXT.5 MX
 0: *movx* instructions use wait states in BANK[7:5]
 1: *movx* instructions use wait states in OSC1EXT[4:0]
 Cleared by reset.

OSC1EXT.6 T1P
 0: Timer1 Prescaler uses Osc1
 1: Timer1 Prescaler uses Osc1/2
 Cleared by reset.

OSC1EXT.7: IRZ
 0: Internal ROM (RSC-364 only) accesses use wait states in Reg BANK[7:5]
 1: Internal ROM (RSC-364 only) accesses use zero wait states
 Cleared by reset.

Also refer to: CKCTL, BANK

Break Address Low Register (Address 0F8h)

msb		BRKLO						lsb	
A7	A6	A5	A4	A3	A2	A1	A0		

This register contains the low 8-bits of the 16-bit Break Address. The high 8-bits of the Break Address are held in the Break Address High register. These registers may be read or written only when the trap bit is set (*See page 44*). When the trap bit is set, each pending PC address is compared with the breakpoint address. When a match occurs, then the PC does not fetch the instruction at the breakpoint address, but rather performs a trap. When a trap is performed the flags register is saved, timers are stopped, interrupts are disabled, and execution branches to location 0FFF8h.

Bit Description:

BRKLO.7-0 Break Address Low bits [7:0]
Set to 0FFh by reset.

Also refer to: BRKHI, FLAGS

Break Address High Register (Address 0F9h)

msb		BRKHI						lsb	
A15	A14	A13	A12	A11	A10	A9	A8		

This register contains the high 8-bits of the 16-bit Break Address. The low 8-bits of the Break Address are held in the Break Address Low register. These registers may be read or written only when the trap bit is set (*See page 44*). When the trap bit is set, each pending PC address is compared with the breakpoint address. When a match occurs, then the PC does not fetch the instruction at the breakpoint address, but rather performs a trap. When a trap is performed the flags register is saved, timers are stopped, interrupts are disabled, and execution branches to location 0FFF8h.

Bit Description:

BRKHI.15-8 Break Address High bits [15:8]
Set to 0FFh by reset.

Also refer to: BRKHI, FLAGS

DAC Hold Register (Address 0FAh)

msb		DAC						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

Holds the eight most significant bits of the value to be converted to an analog signal. Since the analog signal is typically controlled by Sensory library code, there is seldom need for applications programs to access this register. This register is not affected during A/D conversions. It contains a signed, 8-bit number and is cleared to 0 by reset.

Bit Description:**DAC:**

Initialization: All bits cleared to 0 upon reset

Read Access: DAC Hold Value Write Access: DAC Hold Value

Also refer to:

RAM Bank Select Register (Address 0FCh)

msb		BANK						lsb
W2	W1	W0	B4	B3	B2	B1	B0	

The RSC-300/364 architecture supports 1024 bytes of Register Space (RAM). Only 448 of the maximum 1024 bytes are implemented in the RSC-300/364. Since the register space instructions support 8-bit addresses, some registers must be addressed through a banking scheme. The bank register generates address bits 9-5 for accesses associated with register space locations 0C0h-0DFh, the “map bank”. A 10-bit address is always output to the internal SRAM register space (maximum of 1024 bytes). Bit9 is always 0 in the RSC-300/364, and Bit8 is zero unless the lower bits are in the map bank range.

Bit Description:

BANK.0-BANK.4: These bits select a specific 32-byte block of registers to be addressed as “banked RAM” at locations 0C0h to 0DFh. Any 32-byte block in the internal register space may be mapped to banked RAM except for the SFR block and the map bank itself (the block must be aligned on a 32-byte boundary).

The Special Function Registers (0E0h-0FFh) may only be directly addressed. The first 192 locations (000h-0BFh) may be directly addressed or they may be mapped to the map bank, while the remaining 256 locations (100h-1FFh) may only be accessed via the map bank through bank selection. Sensory’s technology code makes extensive use of register space.

Bits BANK.0-BANK.4 select the appropriate bank in RAM as follows.

BANK[4:0] value	Address mapped to 0C0h-0DFh	BANK[4:0] value	RAM bank mapped to 0C0h-0DFh
00h	000h-01Fh	08h	100h-11Fh
01h	020h-03Fh	09h	120h-13Fh
02h	040h-05Fh	0Ah	140h-15Fh
03h	060h-07Fh	0Bh	160h-17Fh
04h	080h-09Fh	0Ch	180h-19Fh
05h	0A0h-0BFh	0Dh	1A0h-1BFh
06h	Not Allowed	0Eh	1C0h-1DFh
07h	Not Allowed	0Fh	1E0h-1FFh
010h or greater	The system will wrap around (SRAM bit9 is ignored in RSC-300/364)		

BANK5-BANK7: WAIT STATES.

These bits define the default number of wait states for external/internal memory access (set to 7 on reset). Note that both the internal and external code and data spaces, but not the SRAM, are controlled by these bits. Sensory technology code requires specific wait states, so changes to this register must be restored before invoking any technology code. The OSC1Ext register allows additional flexibility in wait state generation.

BANK:

Initialization: Bits 5-7 set to 1, all other bits cleared to 0 upon reset

Read Access:

Write Access: Wait State Configuration, Bank selection

Also refer to: OSC1EXT

Interrupt Mask Register (Address 0FDh)

msb		IMR						lsb
		EI5	EI4	EI3	EI2	EI1	EI0	

Bit Description:

IMR0-IMR4: EI5: 1= enable interrupt request #5 (Reserved)
 EI4: 1= enable interrupt request #4 (PWM complete)
 EI3: 1= enable interrupt request #3 (Positive edge of P00)
 EI2: 1= enable interrupt request #2 (Reserved)
 EI1: 1= enable interrupt request #1 (Overflow of Timer 2)
 EI0: 1= enable interrupt request #0 (Overflow of Timer 1)

IMR6-IMR7: Unused

Initialization: All bits cleared to 0 upon reset

Read Access:

Write Access: Interrupt Source Selection

Also refer to: IRQ, FLAGS

Interrupt Request Register (Address 0FEh)

msb		IRQ						lsb
		IR5	IR4	IR3	IR2	IR1	IR0	

Bit Description:

IRQ0-IRQ4: IR5: 1= interrupt request #5 (Reserved)
 IR4: 1= interrupt request #4 (PWM complete)
 IR3: 1= interrupt request #3 (Positive edge of P00)
 IR2: 1= interrupt request #2 (Reserved)
 IR1: 1= interrupt request #1 (Overflow of Timer 2)
 IR0: 1= interrupt request #0 (Overflow of Timer 1)

IRQ6-IRQ7: Unused

Initialization: All bits cleared to 0 upon reset

Read Access:

Write Access: The bits in this register *cannot be set* by writing a one to any bit, but they *can be cleared* by writing zeroes. To assure that pending interrupts are not lost, a bit should be cleared by using a *mov* instruction, not an *and* instruction. For example, the Interrupt Service Routine for Timer1 should clear the interrupt by the instruction:

```
mov    irq, #~1
```

Do **not** use 'and' instructions in the IRQ register.

Also refer to: IMR, FLAGS

Flags Register (Address 0FFh)

msb		FLAGS						lsb
C	Z	S	T	K				GIE

The flags register contains three bits related to the result of the last arithmetic/logical/rotate/misc instruction, one bit that controls breakpoint enabling, one bit for warning of stack full condition, two reserved bits, and one bit that controls the generation of interrupts. The flags register is not affected by branch instructions, except that an IRET instruction restores the value preceding the interrupt. The flags register is not affected by mov instructions unless it is the destination register. Other instructions affect the flags register in different ways. Refer to RSC-300/364 INSTRUCTION SET.

Bit Description:

FLAGS.0: GIE (Global Interrupt Enable)
 0: All interrupts disabled
 1: Interrupts Enabled
 Cleared by reset

FLAGS.1-FLAGS.2: Reserved. Do not change values in these bits.

FLAGS.3 K (Stack)
 0: Stack not full
 1: Stack filled, possibly overflowed
 Cleared by reset. The Stack bit is set when there is no more room in the stack. This can occur under normal program operation, but it may indicate program malfunction. Once set, the bit can only be cleared by reset. This bit may be a useful indicator during development.

FLAGS.4: T (Trap)
 0: Breakpoint function disabled
 1: Breakpoint function enabled
 Cleared by reset. When the Trap bit is set, the processor will jump to the debug monitor when the Program Counter equals the value in the breakpoint register. Normally only used by a debugger.

FLAGS.5: S (Sign)
 0: Result of last Arithmetic/logical operation was non-negative.
 1: Result of last Arithmetic/logical operation was negative
 Cleared by reset

FLAGS.6: Z (Zero)
 0: Result of last Arithmetic/logical operation was non-zero.
 1: Result of last Arithmetic/logical operation was zero.
 Cleared by reset

FLAGS.7: C (Carry)
 0: No carry from last arithmetic/rotate/misc operation.
 1: Last arithmetic/rotate/misc operation produced a carry.
 Cleared by reset

26. SPECIAL DATA SPACE ADDRESSES SUMMARY

As described previously, the RSC-300/364 uses *movx* instructions to access all Data Space locations. Typically Data Space locations are external, but a few specific locations are mapped internally in the last page (0FF00h-0FFFFh) of Data Space. For this reason, it is generally best to plan to use *no* external addresses in the last page of Data Space. (The important exception to this is the debugger interface, mapped externally at 0FFFCCh-0FFFFh.) The internally-mapped addresses include the Stack registers and the Pulse Width Modulator registers. The RSC-300/364 also has 2 Kbytes of internal data space SRAM. This SRAM is reserved for technology functions and does not conflict with external SRAM.

Stack Registers (8 each of 16 bits)

Occasionally it is useful to manipulate the stack directly (for example, to leave a deeply nested series of calls without unwinding when a fatal error is detected.). Since the RSC-300/364 stack space is limited, the need for such manipulations is unlikely, but the information here allows doing so if desired. The stack pointer registers are described on page 38.

Address	Location Name	Type	Notes
0FFC0h	Stack 0 Low byte	R/W	
0FFC1h	Stack 0 High byte	R/W	
0FFC2h	Stack 1 Low byte	R/W	
0FFC3h	Stack 1 High byte	R/W	
0FFC4h	Stack 2 Low byte	R/W	
0FFC5h	Stack 2 High byte	R/W	
0FFC6h	Stack 3 Low byte	R/W	
0FFC7h	Stack 3 High byte	R/W	
0FFC8h	Stack 4 Low byte	R/W	
0FFC9h	Stack 4 High byte	R/W	
0FFCAh	Stack 5 Low byte	R/W	
0FFCBh	Stack 5 High byte	R/W	
0FFCCh	Stack 6 Low byte	R/W	
0FFCDh	Stack 6 High byte	R/W	
0FFCEh	Stack 7 High	R/W	

Pulse Width Modulator (PWM) Registers

The Pulse Width Modulator registers enable and control the operation of the PWM. When producing speech or music, these registers are controlled by Sensory's technology code and should not be touched by applications.

Address	Location Name	Type	Page
0FFE0h	PWMCTRL	W/O	45
0FFE1h	PWMA	W/O	45
0FFE2h	PWMDATA	R/W	46

PWM Control Register (Data Space Address 0FFE0h)

msb		PWMCTRL						lsb
D7	D6	D5	PWMEN	FADJEN	S2	S1	S0	

This register is used to enable the PWM, the PWM frequency adjust, and to select the sample period. The PWM sample rate is derived from Osc#1, divided by various factors controlled by the PWM registers. Sensory's library code may initialize specific settings for this register, so it should be changed only with care.

Bit Description:

PWM_CTRL[2:0]: Sample period

The PWM rate is proportional to $1/(8-S)$. The fastest rate occurs with $S=7$. The slowest rate occurs with $S=0$.

Cleared by reset

PWMCTRL.3: FADJEN

When the FADJEN bit is set, the PWM rate is further reduced by the value in the PWM_A register as described below.

0: Disable PWM Frequency adjust. PWM sample rate = $OSC1/(256*(8-S))$

1: Enable PWM Frequency adjust. PWM sample rate = $OSC1/((512-A)*(8-S))$

Cleared by reset.

PWMCTRL.4: PWMEN

0: Disable Pulse Width Modulator outputs

1: Enable Pulse Width Modulator outputs

Cleared by reset.

PWM Adjust Register (Data Space Address 0FFE1h)

msb		PWMA						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

The PWM Adjust Register contains an 8-bit unsigned value that may further reduce the sample rate of the PWM. If the FADJEN bit is set, then $(256-A)$ wait states are inserted at the end of each sample period.

- If $A=0$, 256 wait states are inserted.
- If $A=0FFh$, one wait state is inserted.

During the wait period, the active output goes to zero. Thus, in addition to lowering the sample rate, smaller values of A also reduce the filtered analog output level.

Bit Description:

PWM_A: Frequency adjust bits D0 through D7

Initialization: All bits cleared to 0 upon reset

Read Access:

Write Access:

Also refer to: PWMCTRL, FLAGS, IMR, IRQ

PWMDATA Register (Data Space Address 0FFE2h)

msb		PWMDATA						lsb
D7	D6	D5	D4	D3	D2	D1	D0	

The PWMDATA Register contains an 8-bit unsigned value that determines the output pulse pattern (duty cycle) of the active PWM output. When D7=0, the PWM0 output is active and the PWM1 output is zero. The largest output signal for PWM0 is obtained with D=00h. When D7=1, the PWM1 output is active and the PWM0 output is zero. The largest output signal for PWM1 is obtained with D=0FFh. The PWM repeats each data pulse pattern once each sample period. At the end of a sample period, the PWM sets PWMIRQ. The interrupt service routine may provide a new data value. If the IRQ is not serviced, the PWM continues to output the data originally stored in PWMDATA. Output data always lags input by one PWM sample period.

Bit Description:

PWMDATA: Pulse Width Modulator data bits D0 through D7
Initialization: All bits cleared to 0 upon reset
Read Access:
Write Access:
Also refer to: PWMCTRL, PWM_A

27. QUALITY AND RELIABILITY

Sensory strives to improve customer satisfaction through the on-time delivery of quality products that exceed customer expectations and requirements. To meet these needs, Sensory is committed to the goal of continuous quality improvement. Each organization within the company is empowered to develop and implement programs that improve the quality of products delivered by Sensory. Sensory's quality program extends not only to internal employees, but to its subcontractors and consultants. Sensory works closely with subcontractors to integrate their quality programs within Sensory's own program.

Reliability and Overview

Sensory's reliability program characterizes Sensory products and identifies areas for future improvement. Sensory's overall program is divided into three main categories:

1. Qualification - This program ensures that new product designs, processes, and packages meet their established specifications (e.g., absolute maximum ratings and worst case criteria for use). A variety of tests and statistical analyses are used to create a high confidence level for determining device performance.
2. Monitoring - After qualification of a device, a monitoring program is used to check that ongoing products continue to meet the established operating conditions. A randomly selected sample of devices are used to monitor the predicted reliability of the products.
3. Evaluation and Improvement - This program continuously evaluates the results of the qualification and monitoring programs to identify areas for improvement. Failure analysis is used to understand the test results and insure quality products are being shipped to customers.

Sensory's reliability testing is designed to deliver commercial grade parts. Since Sensory uses top quality manufacturers and designs to stringent requirements, the RSC-300/364 may meet higher reliability standards (e.g., industrial, automotive).

Reliability Tests

Sensory's reliability testing focuses on the RSC-300/364 in a 64-pin TQFP package. Reliability testing is accomplished by subjecting devices to a variety of stress conditions that accelerate failure mechanisms. The tests used by Sensory have been defined by several industrial standards. JEDEC 22 is the source for most of the testing methods used by Sensory in its reliability program.

This document outlines Sensory's reliability testing for the RSC-300/364. This testing outlines the set of tests performed on the RSC-300/364 in order insure that our products meet their listed specifications. Customers interested in higher levels of reliability should contact Sensory. Sensory uses a combination of independent testing houses and vendors to augment Sensory's internal staff.

Reliability Test Descriptions

The following tests were used to determine the reliability of the RSC-300/364. Table 1 summarizes each of the tests performed.

Table 1

Test	Standard	Conditions
Physical Dimensions	JEDEC-STD-22B-B100	
Mark Permanency	JEDEC-STD-22B-B107	
Solderability	JEDEC-STD-22B-B102	
Autoclave	JEDEC-STD-22B-A102-A	96 hours, 30 psi
Preconditioning	JEDEC-STD-22-A113	Level 3
Bias Life	JEDEC-STD-22B-A108	+125C, 1008 hours
HAST	JEDEC-STD-22-A110-A	100 hrs, 120C, 85%RH, 33.3 psi
Lead Integrity	JEDEC-STD-22B-B105-A	
Resistance to Soldering Heat	JEDEC-STD-22B-B106	
Thermal Shock	JEDEC-STD-22B-A106	-65/+150C, 500 cycles
Temperature Cycling	MIL-STD-883-1010.7	-65/+150C, 2000 cycles
ESD	MIL-STD-883-3015.7	HBM, 2000V
Latch Up	EIJ/JESD-STD-78	Class 1

Package/Process

1. **Autoclave** - reference method JEDEC-STD-22, Method A102-A
This unbiased test evaluates the moisture resistance of nonhermetic solid state devices. Devices are subjected to severe conditions of pressure, humidity and temperature to accelerate the penetration of moisture through the external protective material or along the interface between the external protective material and the metallic conductors that pass through it. This test is performed under the following conditions: TA=~121C, 100% RH, P=30 psi, 96 hours.
2. **Preconditioning** - reference method JESD-STD-22, Method A113, Level 3.
This test method simulates a typical industry multiple solder reflow operation. Plastic surface mounted devices are subjected to this test before being submitted to reliability testing. Level 3 refers to the exposed shelf life of the device, which is 168 hours after removal from a vapor barrier bag. This method is also used as an indicator for the package's resistance to typical moisture conditions found in board assembly.

Package/Chip

1. **Bias Life** - reference method JEDEC-STD-22, Method A108
This test is performed to determine the effects of bias conditions and temperature on solid state devices over an extended period of time. This test accelerates failure mechanisms that are activated by temperature while under bias, and is used to predict long term failure rates based on accepted methods of calculation using acceleration by temperature. It is intended primarily for device qualification and reliability monitoring. This test is performed at a temperature of 125°C for 1008 hours.
2. **Highly Accelerated Temperature And Humidity Stress Test (HAST)** - reference method JESD-STD-22, Method A110-A
This test is performed to determine the reliability of non-hermetic packaged solid-state devices in humid environments. It accelerates the penetration of moisture through the device by subjecting the device to severe temperature, humidity, and bias. This test usually activates the same failure modes as JEDEC-STD-22, Method A101 (85/85). The testing conditions are 120C, 85% RH, and 33.3 psia for a duration of 100 hours.

- 3: **Electrostatic Discharge (ESD)** - reference method MIL-STD-883, Method 3015.7
This test is performed to determine the susceptibility of the device to a defined electrostatic Human Body Model discharge.
- 4: **Latch Up** - reference method EIJ/JESD-STD-Method 78
This test is performed to determine IC latch up characteristics in order to ensure reliability and minimizing failures due to Electrical Overstress

Package Design

1. **Lead Integrity** - reference method JEDEC-STD-22, Method B105-A
This overall test includes various tests for determining the integrity of device leads, welds, and seals. Devices are subject to various stresses and are then examined for failure criteria.
2. **Resistance to Soldering Heat** - reference method JEDEC-STD-22, Method B106
This test provides a method for determining whether device leads can withstand the effects of temperature during soldering.
3. **Thermal Shock** - reference method JEDEC-STD-22, Method A106
This test determines the ability of solid state devices to withstand exposure to extreme changes in temperature. Temperature is cycled to expose damage caused by differing expansion coefficients of the die and package. This test occurs with the chip immersed in liquid. The test consists of 500 cycles between low temperature (-65 °C) and high temperature (+150°C) with an immersion time at each temperature of at least five minutes and transitions of ten seconds or less.
4. **Temperature Cycling** - reference method MIL-STD-883-1010.7
This test determines the resistance of a device to extremes of high (+150°C) and low (-65°C) temperatures, and alternate exposures to these extremes. This test accelerates the effects of temperature changes caused by differing expansion coefficients of the die and package. This test consists of 2000 cycles between low temperature and high temperature with a transition time not to exceed five minutes. Dwell at each extreme is ten minutes.

28. PACKAGING

The RSC-300/364 is available as tested, singulated die, tested wafers, or in a 64 lead, 10 x 10 x 1.4 mm TQFP package.

29. ORDERING INFORMATION

Part	Marketing P/N	Description
RSC-300 Die	C300XD1B	Tested, Singulated RSC-300 die in wafer pack
RSC-300 DWF	C300XS1P	Tested RSC-300 die in wafer form
RSC-300 QFP	C300XT1T	RSC-300 64 pin 10 x 10 x 1.4 mm TQFP
RSC-364 Die	C364XD1B	Tested, Singulated RSC-364 die in wafer pack
RSC-364 DWF	C364XS1P	Tested RSC-364 die in wafer form
RSC-364 QFP	C364XT1T	RSC-364 64 pin 10 x 10 x 1.4 mm TQFP

CHAPTER 9 – DESIGN NOTES

This chapter contains a collection of design and application notes related to creating products based on Sensory's technologies. Additional notes, as well as the latest version of all documentation, may be found on Sensory's website at www.VoiceActivation.com.

In This Chapter

SELECTING AN INTERACTIVE SPEECH CHIP

DESIGN STEPS AND OPTIONS

DESIGNING WITH SPEECH RECOGNITION

SPEECH RECOGNITION HARDWARE DESIGN

MEMORY REQUIREMENTS

QUICK SYNTHESIS INSTRUCTIONS



Selecting an Interactive Speech Chip

Selecting an Interactive Speech Chip

The Interactive Speech™ Product Line

The Interactive Speech family of integrated circuits give product designers maximum flexibility and functionality in creating innovative products utilizing speech and audio technologies. Designers can choose from the RSC-series of chips: RSC-200/264T, RSC-300/364; from our Application Specific Standard Products (ASSP): Voice Direct™ 364, Voice Dialer™ 364, and Voice Extreme™.

RSC-200/264T and RSC-300/364

The RSC-200/264T, RSC-300/364 are 8-bit microcontrollers featuring 64K of internal ROM (RSC-264T and RSC-364 only), on-chip A/D and D/A converters, 16 general-purpose I/O lines, and an on-chip output amplifier. These microcontrollers are customized to perform the speech and audio functions in addition to product control. All offer Speaker-Dependent and Speaker-Independent Recognition, Speaker Verification, Voice Record and Playback, and Speech and Music Synthesis. All have an external memory bus that is configurable for serial or parallel operation, and can be custom masked.

The features and functions of the RSC-series of chips are summarized below:

Feature	RSC-200/264T	RSC-300/364
Sensory Speech™ Technology Supported	4.x	4.x, 5.x, 6.x
External Memory Bus	Parallel or Serial	Parallel or Serial
Package	Die	64-QFP, Die
Custom Mask	Optional	Optional
On-Chip Microphone Preamp	Yes	Yes
Operating Voltage Range	2.4V-5.25V	2.4V-5.25V
Power Down (Sleep Mode)	Yes	Yes
Words Stored on-chip	1	6
Multi-word Continuous Listening	No	Yes
Wordspotting	No	Yes
Simultaneous Record & Patgen	No	Yes
Average Cost (100K, Die)	<\$3	<\$4
Samples Availability	Now	Now
Production Availability	Now	Now

It should be also noted that the RSC-300/364 includes hardware and software technology to provide the highest level of recognition accuracy.

ASSP: Voice Direct™ 364 IC

This non-programmable pin-configurable chip provides considerable functionality and is designed for easy implementation. The IC operates in two modes: stand-alone and external host. In stand-alone mode, it is fully pin configurable and does not require an external microcontroller.

External host mode allows the chip to be controlled by an external microcontroller, thereby providing a full set of chip capabilities to the external controller. This mode enables a product developer to easily integrate the IC into consumer electronic products equipped with an existing microcontroller.

Voice Direct™ 364 is specifically designed for high performance, speaker-dependent speech recognition, making it ideally suited for consumer telephony products. A user trains the product in a quick and simple process, where the user says the words to be recognized twice. The trained words can then be recognized either with a button press, or via Sensory's hands-free Continuous Listening technology. Voice Direct™ 364 is language-independent and targeted to products that are customized by users.

ASSP: Voice Dialer™ 364 IC

This chip is designed for use as a slave chip controlled by an external host processor. The external host sends commands to control the operation of the chip, which include a number of advanced speech technologies. Voice Dialer™ 364 features speech recognition technology that allows users to dial phone numbers by saying the name of the person they want to call. The Voice Dialer™ 364 chip manages a complete telephone directory, including name, phone number and speech recognition template. The Voice Dialer™ 364 IC is targeted for a broad number of cost-sensitive telephony applications.

ASSP: Voice Extreme™ IC

Voice Extreme™ allows quick development of application programs using *Sensory Speech*™ technologies. Voice Extreme allows a developer to write the application program in a higher-level “C-like” language on a PC, accessing the Sensory Speech technologies through calls to “built-in” functions. The Voice Extreme package streamlines development by:

- Allowing programs to be written in C, a commonly used higher level language
- Allowing a simple means of linking the program and the data files it requires
- Providing access to technology functions in a manner that conceals many of the details

The short development time makes Voice Extreme™ ideal for prototypes and moderate volume products. An application using the Voice Extreme™ ASSP would always include a 2MB Flash memory device.

Selecting an Interactive Speech™ Chip

In selecting the optimal Interactive Speech™ chip for your product, a number of factors need to be considered. The main considerations involve:

- 1) The expected volume of the end product.
- 2) Product development timeline.
- 3) The speech and audio technologies desired.
 - The type of speech recognition (Speaker-Independent vs. Speaker-Dependent)
 - The type of speech synthesis (Customized speech vs. Standardized speech)
 - The amount of ROM required for speech synthesis and recognition
- 4) General product control.
 - Does the product already have an existing microcontroller?

Expected Product Quantity

Expected product volume is important because there are minimum volume requirements for our RSC chips. The RSC-series are general-purpose microcontrollers that must be programmed to perform the functions specific to a particular product. The RSC chips are most cost-effective for high volume applications. (Quantities in excess of 100,000 are considered high volume and quantities under 100,000 are considered low volumes.)

The Voice Direct™ 364 and Voice Dialer™ 364 chips perform specific functions. These chips require no additional programming or speech support from Sensory and can be easily integrated into a product. As with all Sensory ASSPs, there are no minimum volume requirements for these chips, making them ideal for both low and high volume products.

Timeline of Product Development

The RSC chips require generally 4-9 months of development time. Development involves detailed product design including hardware and software. Integrating speech into a product with Voice Direct™ 364 or Voice Dialer™ 364 chips is less extensive than the RSC chips, since these chips require no internal custom software. Voice Extreme™ requires programming, but this can be done in a C-like language, thereby simplifying software development.

Speech and Audio Technologies Desired

The actual capabilities of a particular product using Interactive Speech ICs are restricted only by the product's desired functions and available memory.

As previously mentioned, the ASSP chips feature specific technologies. The Voice Dialer™ 364 chip features speaker-dependent technology and DTMF tone generation capabilities. Voice Extreme™ offers access to most, but not all RSC technologies.

Types of Speech Recognition

Sensory offers both speaker-independent and speaker-dependent speech recognition. Speaker-independent speech recognition is designed to recognize pre-determined words spoken by any user and is language-dependent. Speaker-dependent recognition is user-trained speech recognition where a user trains the product to recognize specific words. Speaker-dependent speech recognition is not language-dependent. Speaker-dependent has a typically higher accuracy rate than speaker-independent recognition.

Both speaker-dependent and speaker-independent recognition are available on the RSC chips. The RSC-300/364 also supports Dual Recognition Technology, which provides higher recognition than either speaker-dependent or speaker independent alone, as well as Wordspotting, which is the ability to recognize a word or phrase out of the middle of a sentence. See Sensory for more information regarding this technology.

Speech Synthesis

Using speech synthesis in a product allows a product to interact with the user by providing prompts and cues on using, programming and troubleshooting a product.

Product designers can choose to use standardize speech synthesis or customized speech synthesis. With the RSC chips, designers have the option of using standardized or customized speech synthesis. With Voice Direct™ 364, and Voice Dialer™ 364 only standardized speech synthesis is available. Product designers who wish to customize speech synthesis with our ASSP chips need to contact Sensory for more details.

Product Control

The RSC and Voice Extreme™ chips integrate a fully programmable 8-bit microcontroller that can function as the main microcontroller or as an adjunct processor. Voice Direct™ 364 is a non-programmable, pin-configurable chip that can operate in a stand-alone mode or in conjunction with an existing microcontroller. Voice Dialer™ 364 requires an external host processor.

Summary

In discussing the different considerations involved in selecting an Interactive Speech chip, please refer to the previous table summarizing the different speech and audio capabilities of each chip along with its individual features and benefits. Additional information can be found on our website at www.VoiceActivation.com, or by emailing us at sales@SensoryInc.com.



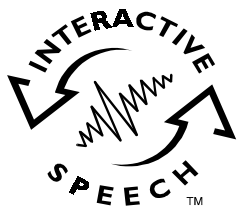
S E N S O R Y

521 East Weddell Drive
Sunnyvale, CA 94089

TEL: (408) 744-9000
FAX: (408) 744-1299

IMPORTANT NOTICE

Reasonable efforts have been made to verify the accuracy of information contained herein, however no guarantee can be made of accuracy or applicability. Sensory reserves the right to change any specification or description contained herein.



Design Steps and Options

This document outlines the various design steps and design options available to a product designer who is using an Interactive Speech chip. Each design step is detailed with various considerations that are important in creating a high quality, high accuracy, speech recognition product.

Design Steps and Design Options

The design of a complete product using the Interactive Speech™ line of chips includes the following six steps:

1. Creation of product specification
2. Review of product specification by Sensory (Recommended)
3. Selection of product developer
4. Development of prototype
5. Hardware review of product
6. Manufacture of product

The first four steps will take approximately 4-6 months to complete. Each of the design steps discussed relate only to integrating speech into the consumer product, and do not cover other product design issues associated with the consumer product.

To assist customers, Sensory offers a variety of services and resources to help customers through each of the various design steps. A description of Sensory services can be found in the last section of this document.

Creating a Product Specification

A detailed specification for the product serves as the guide for using the speech capabilities of an Interactive Speech™ chip. It is extremely important to create a complete product specification to avoid potential delays that can occur later on in product development.

A detailed product specification should contain the following information:

- An outline of the product concept
- An exact description of what the product does, usually in the form of a flow chart
- An exact description of what the product interacts with (lights, switches...)
- Selection of the appropriate Interactive Speech™ chip
- Documentation of the speech recognition vocabulary (if any)
- Documentation of the speech synthesis vocabulary (if any)

Although Sensory can provide services to help designers better define product concepts and identify speech needs, the customer is responsible for the final product specification.

Sensory Review of Specification

It is highly recommended that customers send in preliminary product specifications to Sensory for review. Sensory's Application Engineers have considerable experience incorporating speech recognition into a variety of products and can help identify potential problems that are often overlooked in incorporating speech recognition and speech synthesis.

This stage of development will finalize which Interactive Speech™ IC is best for the product and identify the additional electronic components needed to implement the product.

Selecting an Interactive Speech™ Developer

Customers can choose from several options:

1. Customers can develop the product themselves
2. Customers can contract development services from Sensory
3. Customers can use an independent development house.

For designs with the RSC-200/264T, and RSC-300/364 Sensory offers dedicated Development Kits for each IC. Each Development Kit provides all software and hardware tools needed for experienced product developers to integrate Interactive Speech™ chips into a consumer product. Products using Voice Extreme™ can be created using a low-cost development kit for developers with basic C-language experience. Applications using Voice Dialer™ 364 or Voice Direct™ 364 do not require a development kit, but will in most cases require the programming of an external microcontroller.

If Sensory is chosen to develop a product using an RSC chip, the following services are available:

- Speaker-Independent recognition sets, speech synthesis files, and music synthesis files.
- Application coding of the RSC.
- Hardware design of the electronics associated with the Sensory chips.

Sensory will provide any requested speaker-independent recognition sets, speech synthesis files, and music synthesis files, even if Sensory is not providing software development services.

The following table outlines the steps involved in developing a product and the responsibilities of those parties involved with product development.

Action	Sensory As Developer	Other Developer
Product definition	Sensory assists customer*	Customer or developer
Specification review	Sensory	Sensory assists customer or developer
Speaker-independent recognition sets	Sensory	Sensory
Speaker-dependent recognition sets	No development required	No development required
Speech Synthesis	Sensory	Sensory, or developer with speech development kit**
Music	Sensory	Sensory
Application coding	Sensory	Developer with RSC development kit
Circuit diagram	Sensory	Developer
Bill of materials	Sensory	Developer or manufacturer
PCB layout	Manufacturer	Developer or manufacturer
Manufacturing of product	Manufacturer	Manufacturer

* Sensory is not responsible for any deliverables. Also note that “development” is not the same as “project management”; Sensory is not a manufacturer nor can we manage or be responsible for the efforts of manufacturers.

Product Prototype Development

After the product is properly defined, a prototype can be built. A functioning prototype is important for ensuring that the product will work as envisioned. There is often some fine-tuning required after a prototype is built and tested. The amount of time needed to revise a product depends on the type of change required. Small changes to the product’s application flow can usually be made relatively quickly. Changes to the product’s speech synthesis are more time consuming and can take several weeks, or longer, if a new recording session is required. (This does not apply if Sensory’s Quick Synthesis application is used for creating speech.) Changes to speaker-independent recognition words, although sometimes simple, may take months if new words need to be recorded for the set (each word requires approximately 500 recorded voices).

Hardware Review (Important)

Speech recognition circuitry is extremely sensitive to noise. Therefore, proper design and PCB layout is essential. Before going to production, the developer should carefully check out the hardware design as well as the microphone housing. Please refer to Design Notes for Hardware Design for the steps to review a hardware design. Sensory can also provide hardware review—please see your application engineer for more detail.

Product Manufacturing

After the prototype is completed and accepted, there are three steps that must still be completed:

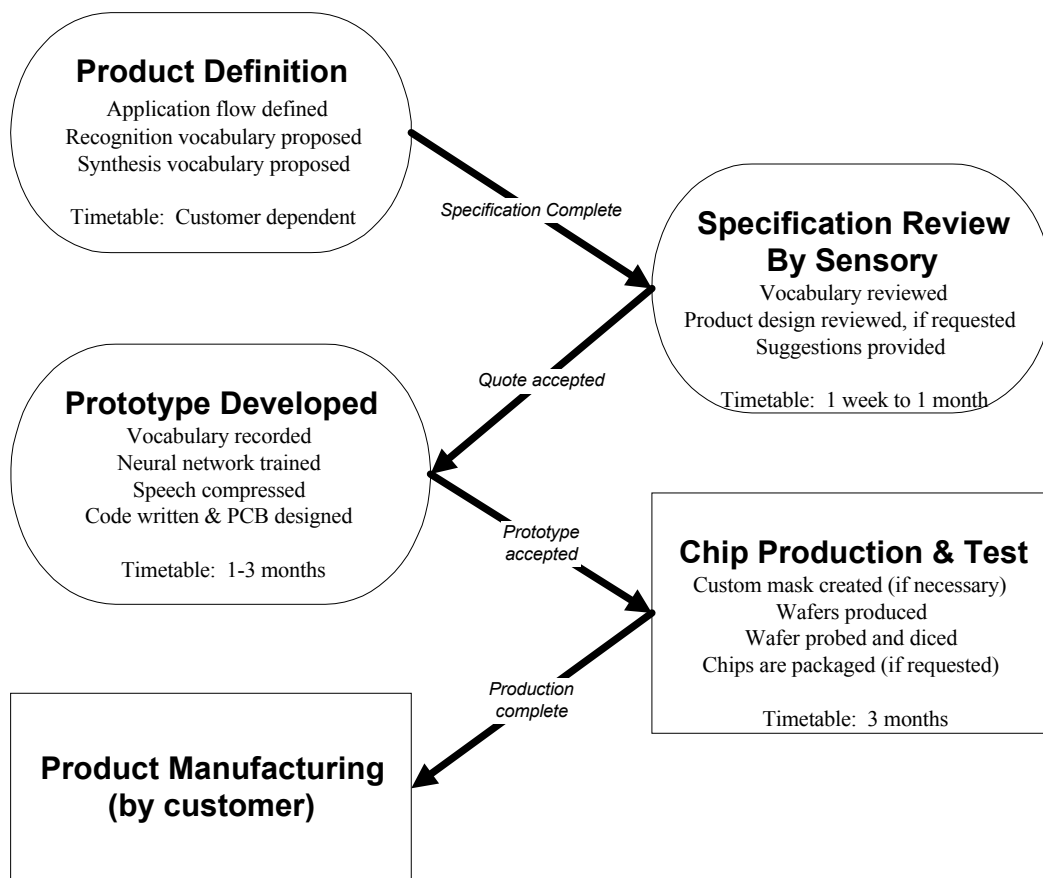
- 1) **Select Chip Packaging.** Prices quoted by Sensory are for chips in die form, unless otherwise specified. The RSC-200/264T is currently available in die form only, whereas all other chips are offered in die and 64-lead TQFP. PLCC versions of all RSC ICs are available for use with Development Kits.
- 2) **Order chips.** Sensory has a typical order lead time of three months. Custom-masked versions of ICs obviously cannot be stocked. All code must be frozen prior to

ordering custom masked chips. Once masked, the code on these chips cannot be changed.

- 3) **Start product manufacturing.** This begins once chips are received from Sensory and all program code is available.

Sample Timeline for Product Development

The following is a timeline for product development when Sensory and the customer work together to create a new product.



Sensory Support Services

Sensory offers many services to support development of those products using the Interactive Speech™ line of chips. For each service, there is a required amount of information about the product that the product designer must provide to Sensory.

Application Programming

Application programming for the RSC is available at hourly labor rates. Project quotes are also available.

Circuit Design

Circuit design services are available at hourly rates to design the circuitry using an Interactive Speech™ chip. Project quotes are also available.

Other Support Services

Product design, product definition and any other support services are available by arrangement.



S E N S O R Y

521 East Weddell Drive
Sunnyvale, CA 94089

TEL: (408) 744-9000
FAX: (408) 744-1299

IMPORTANT NOTICE

Reasonable efforts have been made to verify the accuracy of information contained herein, however no guarantee can be made of accuracy or applicability. Sensory reserves the right to change any specification or description contained herein.



Designing with Speech Recognition

Designing with Speech Recognition

The successful application of speech recognition in a product depends heavily on the specification of the product and the recognition technology used. This document provides an overview of the speech recognition technologies and design guidelines for increasing the reliability of speech recognition in consumer products. Effective use of the speech recognition technologies can not only enhance recognition accuracy, but also reduce memory requirements, and overall product cost.

Speech Recognition: Dependent vs. Independent

Through proper design and specification it is easy to create a product that is 99.5% accurate using the Interactive Speech™ technology. Determining the type of speech recognition to use is one key element in the product's specification.

The two general classes of speech recognition are “speaker-dependent” and “speaker-independent” recognition. With speaker-dependent recognition, the user trains the device to recognize his/her voice by speaking each of the words to be recognized several times. The product then recognizes these pre-trained words when spoken by the user. Training is a quick and simple process. In speaker-independent recognition, the product is *pre-trained* on the voices of *many* different speakers. The product is ready to use and requires no additional training by the user.

Speaker-dependent recognition yields slightly higher recognition rates than speaker-independent speech recognition because it is trained on a specific user's voice. For example, a speaker-dependent device recognizing the digits from 0 through 9 will have an accuracy higher than 99%, while a speaker-independent device performing the same task may have a recognition rate a few percentage points lower. Speaker-dependent recognition devices are ideal for applications requiring complicated recognition tasks. Speaker-dependent recognition systems will require storage of information about the speaker's voice; and thus, depending on the number of words and the Sensory processor chosen, these products may require some type of external memory (SRAM, EEPROM, or Flash).

Through proper product design, speaker-independent devices can yield high recognition rates. For example, a speaker-independent recognizer that distinguishes “yes” from “no” will have an accuracy higher than 99% because the recognition task is simple. More complicated recognition tasks can be performed with an accuracy just a few percentage points lower than speaker-dependent recognition if the product is designed carefully. Product design includes careful definition of the product's target users. This is extremely important in enhancing speaker-independent recognition rates. Speakers' accents, age, gender, as well as regional, socio-economic, and ethnic backgrounds are demographics that must be taken into consideration when *pre-training* the product for speaker-independent speech recognition.

Because no additional writeable memory is required, this technology is ideal for inexpensive consumer electronic products. A final advantage of speaker-independent recognition is that it requires no training by the user - a product using speaker-independent recognition will work right out of the box.

The following table outlines the differences between independent and dependent recognition:

Feature	Speaker-Independent	Speaker-Dependent
Works right out of the box	Yes	No
Can recognize any language	No	Yes
Requires no writeable memory	Yes	No

The RSC-200/264T, RSC-300/364, and Voice Extreme™ offer options for both speaker-dependent and speaker-independent recognition.

Gameplay with Speech Recognition

In addition to technology selection, five keys to attaining the greatest recognition accuracy possible are:

1. Selection of an appropriate recognition vocabulary.
2. Questions which elicit appropriate answers.
3. Recognition triggers which activate the chip to “listen.”
4. Speech synthesis supporting the speech recognition tasks.
5. Control of background noise environments.

Below are some tips on scripting a product’s gameplay for successful speech recognition.

Selecting Recognition Sets

A recognition set is the group of words that the product expects to recognize at any given time. A recognition set can contain up to 14 words for speaker-independent recognition and up to approximately 60 words for speaker-dependent recognition. The success rate of recognition depends primarily on the following two parameters:

1. **Number of words in each set.** Every time a speech recognition product “hears” a word, it compares that word against those in the active recognition set. The more words a recognition set contains, the more likely the recognizer will make a mistake. It is important to limit the number of words in each recognition set whenever possible.
2. **Phonetic distinctiveness of each word in the recognition set.** Words in each recognition set must be chosen carefully. For example, a product that is required to distinguish between the words “three,” “free,” and “tree” will have a much lower recognition rate since these words are phonetically similar. Similarly, if the task is to distinguish “cat” from “rat,” the product will be improved if the recognition set is changed to “cat” and “mouse.”

At different stages during the product's use, the recognizer can be programmed to listen for different sets of words. For example, a product that distinguishes the answers "yes" from "no" at one stage might later recognize a different word set which contains possible answers such as "dog," "horse," "elephant," and "dinosaur." The limited number of answer choices, the different numbers of syllables and the phonetic distinctiveness of each word combine to significantly enhance recognition accuracy.

Phrasing the Question

For the speech recognition chip to work successfully, a question must be phrased to elicit only one *specific* answer. The answer must be specific both by being uniquely correct (the only correct word) as well as by avoiding many forms. (E.g. a cloud, the cloud, cloud, the clouds, or clouds).

Uniquely Correct. Consider the question, "What is three plus four?" This is a good question because there is only one correct answer to this question. In contrast, the question, "Where does water come from?" is ambiguous and may elicit many different correct words, such as "clouds," "the faucet," "rivers," "rain," and "the reservoir." If the product has been coded to accept only "clouds" as the correct answer, then it will appear not to work upon rejection of these other correct answers. A better question would be, "What is white and puffy and gives us rain?"; it is specific enough to rule out different vocabulary items like "the faucet," "rivers," "the reservoir," etc. It is extremely difficult for the chip to choose intelligently from among all the possible correct answers to a loosely worded question.

Form. A question must also be crafted to elicit exactly one form of an answer. For example, the question, "What is white and puffy and gives us rain?" has a uniquely correct answer, but the correct answer could occur in different forms: "clouds," "a cloud," or "the clouds." It is therefore important that questions be phrased such that they do not permit variations in number (singular vs. plural) or in use of an article ("a" or "the"). It would be an impractical goal for the recognizer to take into account the different combinations of number and article use for each answer.

Now consider the question, "Does water come from clouds or from sunshine?" From the standpoint of speech recognition accuracy, this is a good question because it elicits one of two specific answers, each with a single form (no variation of number or article use). Both possible answers, "clouds" and "sunshine," would be programmed into the recognizer and the product would be able to determine the correct answer with great accuracy.

Triggering the Speech Recognition Chip

There are three possible modes of operation for the speech recognition chip: normal, continuous listening and wordspotting modes (not available on the RSC-200/264T). In continuous listening and wordspotting modes, the chip is ready at all times to detect a recognition word or phrase. Wordspotting offers the ability to identify a long word or short phrase out of the middle of a sentence, whereas continuous listening requires that the trigger words be separated by silence. In normal mode, it is necessary to ensure that the chip knows when to listen. The latter is a more robust approach and should be used whenever possible; continuous listening and wordspotting should be used only for specific applications that require this feature.

Ensuring the chip knows when to listen

The best way to ensure that the chip knows when to listen is through speech prompts (also referred to as speech synthesis). Usually, the end of a question (“What is three plus four?”) or a synthesis prompt (“Record or Play?”) helps to signal when the user is supposed to speak and when the speech recognition chip is expecting a response.

Another approach is to activate the chip manually by pushing a button before speaking. This approach can be avoided with appropriate use of speech synthesis.

Continuous Listening and Wordspotting

The use of continuous listening or wordspotting will generally reduce the product’s overall recognition accuracy. It is difficult for the chip to clearly distinguish a single word or phrase from all the other audio signals that it receives when it is continuously listening. Random noise can be mistaken for the recognition word or phrase. Continuous listening can be implemented using either speaker-independent speech recognition or speaker verification; wordspotting is speaker-dependent only.

Consider a wall clock that announces the time when you say a particular phrase. In a noisy environment such as an office or a conference room, it will occasionally announce the time even though the trigger phrase might not have been spoken. This apparent “mistake” occurs when the chip “hears” sounds that are phonetically similar to a trigger phrase.

The frequency of such “false positive” responses depends on the uniqueness of word to be recognized. For example, its operation may be acceptable if the word to be recognized includes a trigger recognition phrase. For example, the phrase “wall clock” can be used to trigger the clock to listen for the word “time”. If the word “time” is then said, the clock will know to announce the time. This two step process will have a lower error rate than a recognition process only using the word “time.” Sensory recommends using two words for continuous listening to improve accuracy.

Power consumption is also an important factor in continuous listening. In operation, the speech recognition chip may draw a current of 10 milliamperes. If it is powered to continuously listen for some given phrase, it will drain a button battery in several hours or a large alkaline battery in several days. Thus, if the application requires that the recognizer be listening all of the time, it should operate from AC wall power. If the product is to operate on button batteries, then it must be awakened from a low power “sleep” mode for a period of a few seconds every time it is asked to recognize a phrase.

Synthesis Support

The Interactive Speech recognition chip can also interact with the user through speech synthesis to clarify responses. When recognizing words or phrases, Sensory’s recognition technology calculates its own probability of success. The product can thus be designed to prompt for cues if a desired level of recognition accuracy hasn’t been reached. For example, if a recognition doll is told to “walk” and the chip calculates that it has greater than an 80% chance of being accurate, it

can accept the “walk” command. If it is only 50-80% sure of its accuracy, the recognizer can prompt, “Did you say walk?” The answer, “yes” or “no,” is very easy for the chip to determine and this provides a robust method of (eventually) getting the right answer. If less than 50% confident, the chip can ask, “What did you say?” effectively starting the recognition task over.

The following phrases should be included in all products using synthesis. These phrases will help improve overall product accuracy by notifying the user of possible problems.

- Please talk louder / Please talk into the microphone.
- Please talk softer.
- Please say...*[each recognition item]*.
- Did you say... *[each recognition item]*?
- What did you say?
- You spoke too soon.

The following phrases may be desirable, depending on the product:

- I think you said...*[each recognition item]*.
- In the picture, what do you see that ...

Controlling the Noise Environment

Just like people, speech recognition systems have difficulty recognizing words in a noisy environment. Speech recognition products should be designed for use in a quiet environment whenever possible. If the product is meant to be used in a noisy environment, care must be taken to try to control the noise. For example, consider speech recognition being used in a video game with shoot-'em-up noise and music. Either the user should be provided with headphones for listening to the sounds (without their being heard by the microphone), or the sounds should be muted when the user is expected to talk, or a headset microphone should be used to provide a good voice signal to the recognition system.

Designing with Synthesis

Synthesis is the product's ability to “talk.” The use of music, sound effects and speech synthesis can be easily incorporated to produce a product that is interactive and user-friendly. Speech synthesis allows the product to interact with user to provide feedback or directions as to how to better use the product. Although synthesis does require additional memory which can increase overall product cost, it's overall value to the end user is priceless.

Speech Synthesis Overview

Creating speech synthesis files for the Interactive Speech™ chips begins with making a list of all words/phrases the product will ever say. From this list, a recording script will be generated. If Sensory is performing the speech synthesis development, then the recording script and all subsequent steps will be done for you. If not, then the next step is to use a voice talent to record phrases based on the script. After recording, the best of all the recorded files must be chosen for use in the product. These files are then compressed into one master file (a .O file) that a developer will use to make the chip talk. This can be done either by Sensory, or by the developer using Sensory's Quick Synthesis application. Creating speech synthesis using Quick Synthesis

can be done in minutes, but with some sacrifice in speech quality and data compression compared with Sensory-compressed files.

A technical note: The term “speech synthesis” can be misleading. The Interactive Speech™ chips perform “time domain compression” that sometimes re-uses sections of speech in different words. This re-use of speech sections is the reason this process is referred to as “synthesis.” In general, however, it is more useful to think of the Interactive Speech™ line as using “time domain compression” instead of “speech synthesis”.

Re-using Words and Phrases

Overall memory requirements for synthesis can be reduced by re-using words and phrases as much as possible. However, the re-use of words and phrases is not as simple as it may appear. A given word is not always suitable for re-use in other phrases. Words cannot be strung together in any order to form any sentence. Words contain qualities such as intonation, volume, duration, and emphasis that can vary according to the meaning and function of the sentence in which they occur. These qualities must be preserved in order to replicate natural-sounding speech. The challenge for a scriptwriter is to balance these considerations and maximize the vocabulary within a limited amount of memory.

Pronunciation of a given word often depends on context. For example, the word “the” is pronounced several different ways depending on the context. “The” is usually pronounced “thee” before vowels (e.g., “thee elephants,” “thee apple”). Before consonants, “the” is often pronounced as “thuh” or just “th” (e.g., “thuh red,” “th fish”).

Intonation is also an important factor in determining the suitability of words and phrases for re-use elsewhere in the product. By “intonation,” we mean the melody or variations of pitch in spoken words and phrases. The intonation of a word or phrase depends on its location in a sentence, as well as the meaning of the sentence. Proper intonation not only sounds natural but also provides listeners with important cues revealing the speaker’s meaning and intent. For example, consider the word “fish” in “Is this a fish?” and “This is a fish.” Different versions of “fish” should be used in these sentences to ensure natural-sounding, comprehensible speech. Recording a script in a near monotone would reduce variations in intonation and allow greater re-use of words. However, the result would be dull, stilted speech.

Re-use of speech is thus more effective if a script preserves sentence patterns and re-uses sentence units. If the number of sentence patterns is kept to a minimum, units of speech can be more readily interchanged. For example, consider the sentences “At the beep, press a button,” and “At the beep, tell me your name.” The single phrase “at the beep” can be used in both sentences because they call for identical intonation and emphasis. This phrase could not be re-used in a sentence with a different pattern, such as “Say your name at the beep,” without losing the intonation that makes speech sound natural. Given the proper conditions, words and short phrases can be used in different parts of sentences, or even in different sentence structures. The more the speech is chopped up and recombined, however, the more disconnected the resulting compressed speech will sound.

Counting Words in a Script

It is essential to estimate how much memory will be required and what level of effort/cost will be associated with compressing the speech synthesis phrases. These factors effect the overall cost of the product and its development. Counting words is the best way to approximate the total requirements. It is usually best to count words conservatively (e.g., count “fireman” and “firehouse” as two words each).

Each time a word is said in a different sentence, in a different way, or surrounded by different words, that word should be counted again. For a conservative estimate it is best to repeat phrases, not words. Sensory’s team of expert linguists can evaluate phrase lists and determine how words or phrases may be formulated for re-use within the product.

Synthesis in More Detail

This section details exactly how much sound (words, character-voices, sound effects, music, beeps, and bells) can fit on an RSC-264T or RSC-364. The total amount depends on both subjective (e.g. sound quality) and qualitative (e.g. compression rate) factors.

The procedure for determining the number of seconds of sound playback available: Calculate the amount of **available ROM** (in bits) and **divide** by the **data rate** (bits/second) for the sound you want played back.

Available ROM

The RSC-264T and RSC-364 chips have 64 kilobytes (that’s 64,000 bytes or $64,000 \times 8 = 512,000$ bits) of ROM on-chip – but not all of it is available for sound output. The ROM space is used for other functions, like speech recognition. Ultimately, even a product with an emphasis on speech output will have under 50kB (400,000 bits) of ROM available on-chip for the customer to store sound output. The customer can add as much off-chip ROM or Flash as desired in order to increase the sound capability of the product. External ROM or Flash is required with the RSC-200 and RSC-300. Voice Extreme™ is designed to operate in conjunction with a 2MB Flash memory device.

Data Rate

Sound exists as an analog signal that must be converted to a digital signal for storage on a chip. During this conversion, recorded sound is compressed to reduce the amount of storage needed on-chip. The RSC IC can compress sound down to data rates from 15,000 bits/sec to under 5,000 bits/sec. The data rate depends on the following factors:

- 1) **Desired quality.** Sound compression involves a tradeoff between quality and quantity. The more we compress the sound, the more will fit on the chip. The less we compress it, the better it sounds. This is subjective; what sounds great to a toy designer may sound horrible to an audiophile.
- 2) **Nature of the sound.** Some sounds compress better than others. If a customer *wanted* the product to sound like a robot, then we could achieve very low data rates (under 5,000 bits/sec). However, a high-pitched voice, such as a licensed reproduction of Minnie Mouse, might require more than 15,000 bits/sec. There are two reasons for this. First, we want to retain the *personality* of Minnie Mouse’s voice. Second, high-pitched sounds don’t compress

as well as low-pitched ones. As another example, sound effects are often very difficult to compress, requiring up to 20,000 bits/sec.

- 3) **Repeatability of the sound.** After you use a sound once, you can use it again without requiring extra memory. This can happen on several different levels. Sound effects, for example, can be looped such that a single “whomp” plays repeatedly to produce the sound of a helicopter: “whomp whomp whomp whomp...” Words and phrases can also be re-used in the same part of different sentences or, sometimes, in different parts of sentences. Our ability to do this depends on several factors: the desired quality, the script, and the intonation of the words and phrases selected for use in synthesis.

As a baseline, a custom masked RSC can fit from 25 to over 100 seconds of sound on-chip. A word usually takes about 0.5 sec and a sound effect about 1sec. Therefore, the RSC chips can fit from approximately 50 to 200 words on-chip. As mentioned previously, additional sound can be stored in off-chip ROM or Flash.

Sensory Support Services

Sensory offers a variety of services and resources to help product designers integrate speech into their products.

Speaker-Independent Recognition Sets

Currently it is required that Sensory creates the independent recognition sets for use with the Interactive Speech™ chips for all customers. This section identifies the information and steps required for building speaker-independent sets.

The following information is required to create the sets:

1. List of recognition words divided into appropriate recognition sets.
2. Description of the environment the product will be used in.
3. Recordings of approximately 500 people saying the entire word list. The demographics of this pool must accurately reflect that of the product’s target market.
4. Sample background noises.

The recognition list (include English translation if appropriate) should be arranged with the words in a single list as well as in their respective sets. The following is an example of a recognition list for a simple record and playback device:

Set 1:	Set 2:	Entire List:
1. Record	1. Yes	Record
2. Play	2. No	Play
3. Erase		Erase
		Yes
		No

Sensory will review the word list, paying careful attention to the selections in each set in order to ensure accurate recognition. If Sensory does not foresee any problems, then the acquisition/recordings of the appropriate words can start.

To record voices, an IBM PC or compatible computer with a high quality sound card is required. Sensory will support the customer or sales rep as needed.

Speech Synthesis

Sensory has available services that allow them to add speech synthesis to their products. Current procedure requires that Sensory create the speech synthesis files for use with the Interactive Speech™ chips. In order for Sensory to create synthesis files, the customer should create a phrase list containing every phrase to be spoken by the product.

The phrase list should contain all phrases to be said by the product. Include a translation into English if appropriate. No phrase or word concatenation should be performed at this stage. An example of a phrase list for a simple record and playback device follows:

Example:

1. Record, play, or erase?
2. Recording.
3. Playback.
4. Erased.
5. Please talk louder.
6. Please talk softer.
7. It is too noisy here.
8. Just say: record, play, or erase.
9. Did you say “record”?
10. Did you say “play”?
11. Did you say “erase”?
12. Please wait until I’m finished / Please wait for the beep.
13. What did you say?

A development station is being designed that will enable customers to create their own synthesis files. Contact Sensory for further information.

Music Synthesis

Sensory also provides customers with services that allow them to use music synthesis in their products. Currently Sensory creates the music synthesis files for use with the Interactive Speech™ chips for customers. In order for Sensory to compress musical compositions for use on

the Interactive Speech™ chips, the customer must create files in four-voice MIDI format. In these MIDI files only certain percussive and musical instruments, with designated note ranges,

are compatible with Sensory's compressed music note library. These restrictions on the MIDI instruments and note ranges are specified in the following table:

Acceptable Instruments and Note Ranges

Piano	D2-E7
Trombone	C3-C6
Clarinet	C3-E7
Banjo	C3-E7
Bass Drum	n/a
Snare Drum	n/a
Cowbell	n/a

In the above notation system, C3 is two octaves below C5, which designates middle C (C5 = 523.25Hz).

No more than four 'events', i.e. any combination of four musical notes and/or drumbeats, may play at any one time for music files. For example, at time 04:01:000 in the composition only one piano note, one trombone note, one banjo note and one percussion beat may play simultaneously. Likewise, a piano chord of, say, three notes may be accompanied by only one other event, such as a single clarinet note.

For the fullest sound, the composer should make the most of the four voices available. If you do not have in-house music talent that can recompose songs to meet the above criteria, you can send a "plain" MIDI file to Sensory and we will recompose it for you. The only disadvantage to this approach is that the song may sound slightly different than you expect by the time we have made adjustments for compatibility. To ensure the best reproduction, we recommend sending an audio tape.

Other Support Services

Product design, product definition and other support services are available from Sensory by arrangement.



S E N S O R Y

521 East Weddell Drive
Sunnyvale, CA 94089

TEL: (408) 744-9000
FAX: (408) 744-1299

IMPORTANT NOTICE

Reasonable efforts have been made to verify the accuracy of information contained herein, however no guarantee can be made of accuracy or applicability. Sensory reserves the right to change any specification or description contained herein.



Speech Recognition Hardware Design

Speech Recognition Hardware Design

Introduction

Proper hardware design is essential to creating a high quality speech recognition product. The Interactive Speech™ ICs have the best speech recognition quality/cost ratio available today, but without an effective hardware design, the performance of these ICs will be severely degraded. This design note focuses on hardware issues, guiding the reader through important hardware design considerations. This note should be used in conjunction with the other Interactive Speech™ design notes to ensure the creation of a high quality speech recognition product.

Sensory highly recommends that this procedure be used to design and test ALL new product designs.

Table of Contents

SPEECH RECOGNITION HARDWARE DESIGN	1
MICROPHONE	2
SELECTING A SUITABLE MICROPHONE	2
DESIGN OF MICROPHONE HOUSING	2
PCB DESIGN	6
ANALOG SECTION	6
LAYOUT REQUIREMENT	8
LOCATING AND MOUNTING INTERACTIVE SPEECH ICs	8
CIRCUIT VERIFICATION PROCEDURE	9
PROCEDURE	9

Microphone

Selecting a Suitable Microphone

For most applications, an inexpensive omni-directional electret capacitor microphone with a minimum sensitivity of -60 dB is adequate. In some applications, a directional microphone might be more suitable if the desired signal comes from a different direction than the audio noise. Since directional microphones have a frequency response that depends on their distance from the sound source, such microphones should be used with caution. If the product is intended to be used in a noisy environment, care should be taken to design around the noise. Improving the signal-to-noise ratio will help increase a product's speech recognition accuracy. For Speaker Independent applications, Sensory always recommends that training words be recorded using the same microphone and pre-amp as will be used in the final product.

Design of Microphone Housing

Proper design and consistent manufacturing of the microphone housing is important, because improper acoustic positioning of the microphone will reduce recognition accuracy. This section describes several important considerations that must be carefully followed in designing the microphone mounting and housing. Many mechanical arrangements are possible for the microphone element, and some will work better than others. Sensory recommends the following guidelines for designing the microphone housing:

FIRST: In the product, the microphone element should be positioned as close to the mounting surface as possible and should be fully seated in the plastic housing. There must be NO airspace between the microphone element and the housing. Having such an airspace can lead to acoustic resonance, which can reduce recognition accuracy.

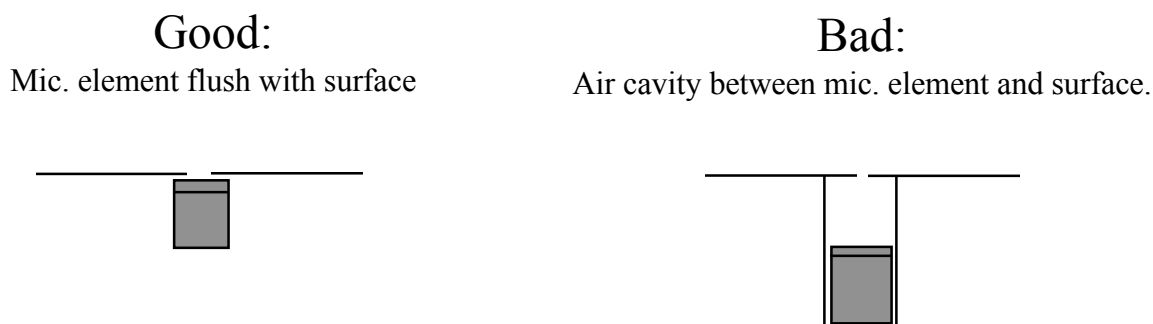


Figure 1 -- Microphone Mounting (1)

SECOND: The area in front of the microphone element must be kept clear of obstructions to avoid interference with recognition. The diameter of the hole in the housing in front of the microphone should be at least 5 mm. Any necessary plastic surface in front of the microphone should be as thin as possible, being no more than 0.7 mm, if possible.

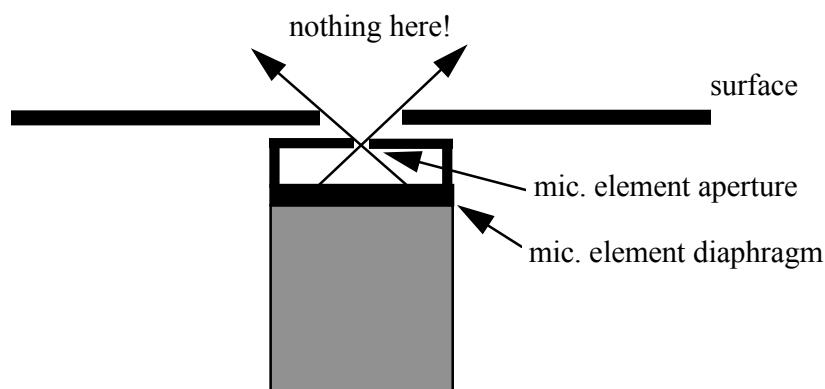


Figure 2 -- Microphone Mounting (2)

THIRD: The microphone should be acoustically isolated from the housing if possible. This can be accomplished by surrounding the microphone element with a spongy material such as rubber or foam. Mounting with a non-hardening adhesive such as RTV is another possibility. The purpose is to prevent auditory noises produced by handling or jarring the product from being “picked up” by the microphone. Such extraneous noises can reduce recognition accuracy.

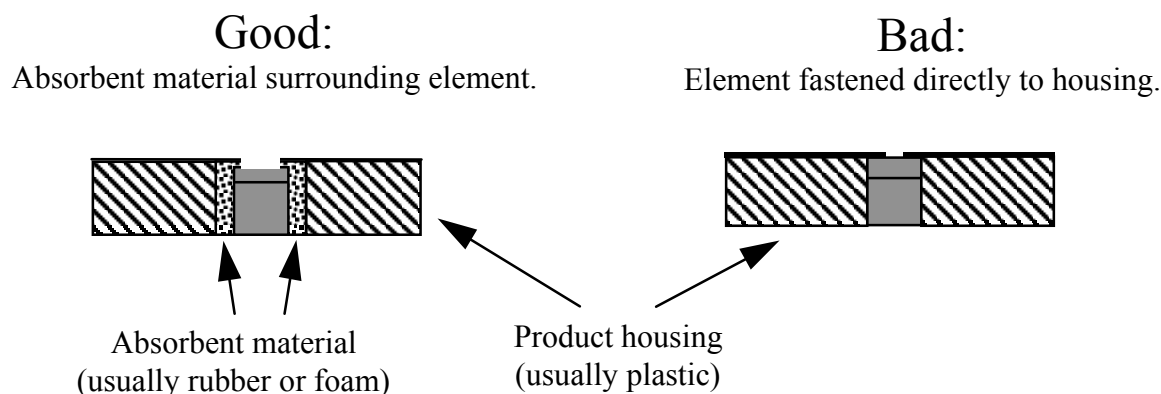


Figure 3 -- Microphone Mounting (3)

If the microphone is moved from 6 inches to 12 inches from the speaker’s mouth, the signal power decreases by a factor of four. The difference between a loud and a soft voice can also be more than a factor of four. Although the internal preamplifier of the RSC-264T/364 compensates for a wide dynamic range of input signal strength, if its range is exceeded, software can provide auditory feedback to the speaker about the voice volume. The product can achieve this by saying such as “please talk louder” or “please don’t talk so loud.

Analog Design

The diagram in **Figure 4** shows the microphone input circuit and the required external components and their connections to the RSC-264T/364 for the internal pre-amplifier circuit. The components for each circuit are listed in **Table 1** and **Table 2**.

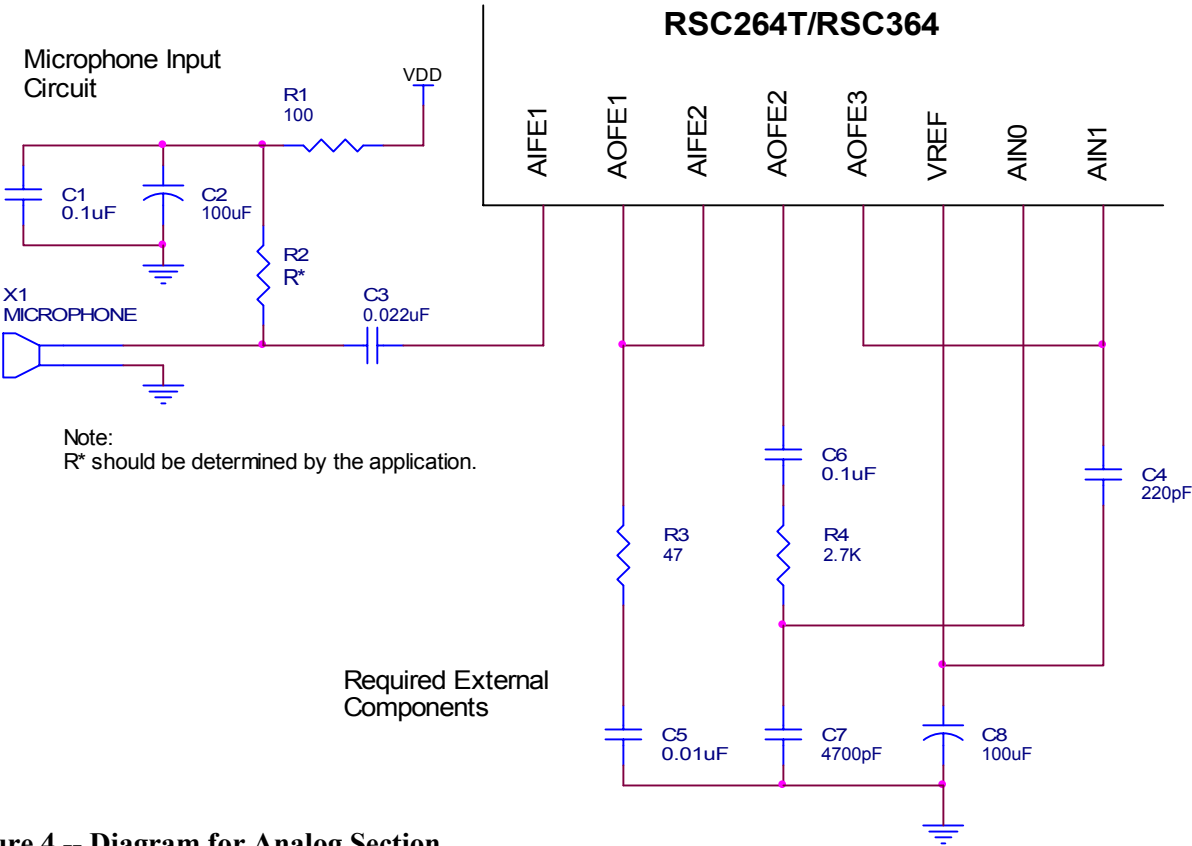


Figure 4 -- Diagram for Analog Section

Table 1 – Components for Microphone Input Circuit

Resistors			
	R1	100, 5%	
	R2	2.7K, 5%	Recommended value. Select to match the microphone specification
Ceramic Capacitors			
	C1	0.1uF, Z5U	Optional de-coupling capacitor
	C3	0.022uF, X7R	
Electrolytic Capacitor			
	C2	100uF	Recommended value

Table 2 – Required External Components for Internal Pre-amplifier

Resistors			
	R3	47, 5%	Critical Value
	R4	2.7K, 5%	Critical Value
Ceramic Capacitors			
	C4	220pF, X7R	Critical Value
	C5	0.01uF, X7R	Critical Value
	C6	0.1uF, X7R	Critical Value
	C7	4700pF, X7R	Critical Value
Electrolytic Capacitor			
	C8	100uF	Recommended value

Note: Some applications may use the RSC with input signals from a device other than an electret microphone. For assistance with such designs, please contact Sensory.

PCB Design

Analog Section

The circuit board should be physically organized to allow the largest practical separation of the microphone input and analog pre-amp circuit from the digital portions of the system. The microphone signal is a relatively high impedance signal with an amplitude of only one millivolt or so – digital noise and pickup must be avoided if good recognition accuracy is to be obtained.

Products should use a double-sided (or multi-layer) printed circuit board (PCB) with ground plane. The ground plane should cover all analog circuitry area and only connected to digital ground near the RSC device. A star ground system should be employed to ensure that this single point is the only connection between the digital and analog grounds. Care should be taken to make sure that the analog ground does not carry any digital path current. In order to reduce cross talk, the analog and digital circuits should be physically separated as far as is practical.

The reference surface mount PCB layout for the analog section is illustrated in Figure 5.

Note: The ground plane should only be connected to the star of the star ground system, and should not carry any return current.

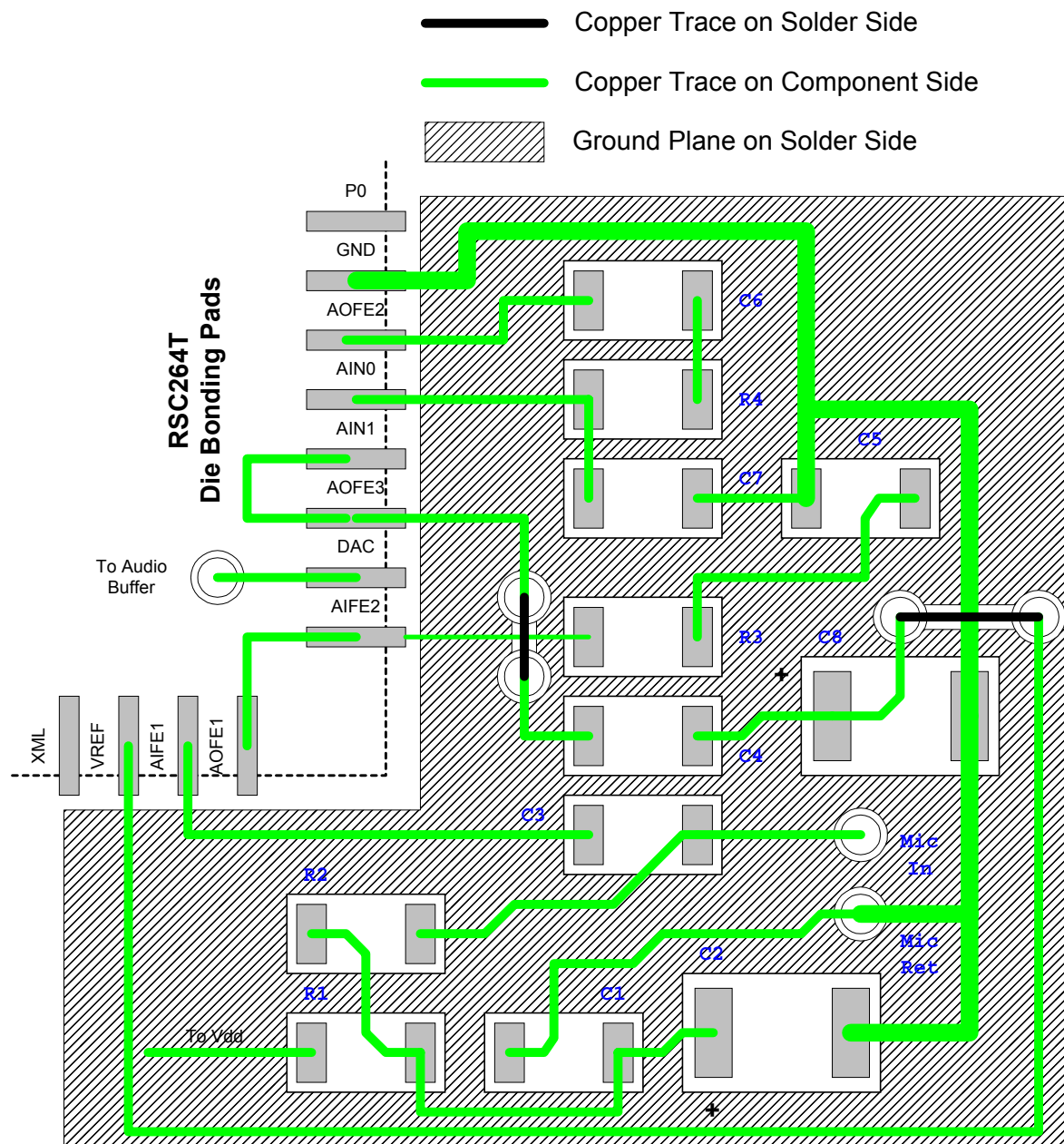


Figure 5 -- Parts Reference PCB Layout for the analog section

Layout Requirement

Proper layout techniques should be applied not just to the analog section of the PCB, but to the entire PCB layout, including the digital section.

De-coupling Capacitors

A 0.1 μ F bypass capacitor should be installed immediately next to each digital IC and near the VDD pins of the RSC chip. The bypass capacitors should be a monolithic ceramic type, rated at 50 volts. If a three-terminal voltage regulator (such as a 7805) is used, Aluminum electrolytic or Tantalum capacitors should be connected close to the regulator between the output pin and ground, and also the input pin and ground.

Data and Address Bus

The data lines should be as wide as possible to reduce impedance. The bi-directional nature of data lines causes large instantaneous currents to get switched around. With inductive loading, these currents can cause data lines to ring and generate large switching spikes. Although these spikes and the ringing will have dampened when the data is sampled, they could affect the operation of other devices connected to the same bus.

Traces for the data and address line should be routed for the minimal length - make them as short as possible.

The address, strobe and control signals are outputs of the RSC-264T/364 and have a nominal capacitance; their trace widths and length aren't critical, with a preferred width of more than 0.012". These traces can be made around the power traces and the data line traces, but should avoid analog traces.

Locating and Mounting Interactive Speech ICs

Some Interactive Speech ICs are available in packaged form or as bare die. Bare die may be wire bonded directly to the main PCB or, in some cases, may be bonded to a separate chip-on-board (COB) circuit board. In production this COB assembly may be functionally tested, then attached to the main board as a working module.

There are several methods of attaching the COB to the main board, and a careful choice should be made by the designer. Since cost is always a consideration, COB boards may often be designed as single-sided PCBs.

The simplest way of attaching a single-sided COB to a main board is to lay it, chip side up, on the main board and make solder bridges from the main board up along the thickness of the COB to the electrical contacts on the top of the COB board. However, in production this is not a reliable technique.

A second technique is to use wires or pins to connect thru-holes between the main board and the COB. This reliable technique may be more time consuming.

A third technique is to put a hole in the main board at the center of the COB location and to mount the COB to the main board UPSIDE DOWN, that is, with the chip facing down into the hole. Then the COB and main boards may be soldered together. In this case, the orientation of the signal leads of the RSC-264T/364 is different from that of the other arrangements.

A fourth technique is to put a slot in the main board that is the same width as the thickness as the COB, and the same length as the COB. The COB can then be mounted perpendicular to the main board, and the pads soldered together. This arrangement offers dimensional advantages in certain applications.

All RSC's in QFP packages are shipped in dry-packed trays, and meet JEDEC level-3 moisture sensitivity standards. Level-3 specifies that once the bag has been opened, the devices should be board mounted within 168 hours (1 week). If not, they must be re-baked at 125C for 24 hours. Profiling of the IR-Reflow ovens should be carefully monitored, with settings established in accordance with JEDEC standards.

Circuit Verification Procedure

Procedure

After completing the hardware design, the product's speech recognition performance should be carefully verified. Sensory has developed a software utility located in the subdirectory of the Developer Kit "\DKxxx\supplmnt\ampstest\". If you do not have the program, please contact Sensory Technical Support. These tests should be performed on an exact sample of the hardware that will be used in production, prior to volume production. The programs should be programmed into an EPROM or FLASH (if possible) to avoid noise introduced by a ROM emulator. These tests can also be performed on a sampling of production units to test the consistency of the production hardware. **Sensory recommends that all customers complete the following tests.**

REQUIRED EQUIPMENT:	Sine wave Oscillator Oscilloscope
REQUIRED PROGRAM:	<i>264Noise</i> for RSC-264T <i>364Noise</i> for RSC-364

Description of Test Program

At power up this program cycles, repeatedly measuring the noise and synthesizing the noise value every few seconds. It requires a speaker and microphone, but it does not require any other special circuitry.

Procedure

Install the EPROM or OTP with “Noise” for the RSC-264T or “364Noise” for the RSC-364 program onto the board, and power up the board without the microphone connected.

A. Amplifier Noise Test

1. DC Bias Voltage

Check the DC bias voltage at AOFE2. It should be $\frac{1}{2}$ of VDD.

2. Noise at AOFE2

Check the noise (about 1MHz) with an oscilloscope at AOFE2. It should be less than 100mVp-p. If the noise is greater than 100mVp-p, install a 1000pF capacitor from AOFE1 to ground. If this is not adequate for decreasing the high frequency noise, contact Sensory.

3. Noise Value

Record the noise value synthesized by the “264Noise” program. If the average value is less than 5, the design is good.

B. Microphone Gain Test

If the board passed the noise check, proceed with Microphone gain setting.

1. Noise Value with Microphone Connected

Connect the microphone. IN A VERY QUIET ROOM, the noise value synthesized should be somewhat larger than it was without the microphone, typically a value around 7 to 12.. If so, it is a well-designed system.

2. Adjusting Microphone Gain

If the noise with microphone connected is not larger than measured without, the microphone gain is probably too low. The gain can be increased by increasing the resistance of the microphone load, which is nominally 2.7 Kohms. If this resistance is increased too much, the gain will actually decrease.

For typical electret microphones, the maximum value of this resistance is about 10K Ohms.

If the system noise with the microphone connected is not greater than with the microphone input floating, and if the microphone resistance is 10K, please contact Sensory for additional assistance.

The following table represents the microphone gain as a function of the microphone resistance, for a particular microphone. Measurements were taken by putting a

constant amplitude 1500 Hz sine wave into a speaker located at a fixed distance from the microphone. On a 5V system, the voltage at AIN1 was:

Resistance	Voltage
2.7 K	1.1 V
5.6 K	1.9 V
7.5 K	2.5 V
11 K	3.0 V
15 K	3.5V
22 K	3.1 V
30 K	1.4V



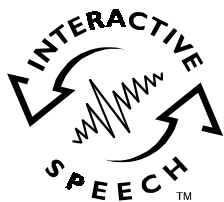
S E N S O R Y

521 East Weddell Drive
Sunnyvale, CA 94089

TEL: (408) 744-9000
FAX: (408) 744-1299

IMPORTANT NOTICE

Reasonable efforts have been made to verify the accuracy of information contained herein, however no guarantee can be made of accuracy or applicability. Sensory reserves the right to change any specification or description contained herein.



Design Note

Memory Requirements

Memory Requirements

This document outlines the memory requirements for each of Sensory's speech and audio capabilities. Examples of hypothetical applications are provided to give developers a better understanding of the memory requirements for different combinations of speech technologies.

Memory Usage on the RSC-300/364

The RSC-364 has two types of on-chip memory, ROM and RAM. (The RSC-300 is a ROM-less version of the RSC-364). ROM is used for five purposes: control code, technology code, speaker-independent speech recognition weights, speech synthesis data, and music synthesis data. The on-chip RAM has two primary functions: to store information (like speaker-dependent passwords) and to provide scratch pad memory while performing its normal functions. Off-chip, the RSC-300/364 can also address ROM and a variety of read/write memories (e.g., SRAM, Flash, EEPROM).

ROM

The RSC-364 has 64 kilobytes (kB) of ROM on-chip and can address (use) a virtually unlimited amount of off-chip (additional) ROM.

ROM is used for a variety of functions:

- 1) **Control Code.** This refers to the set of instructions that control the product (e.g., when the chip hears Red, flash the Red light). This takes about 2kB for a modest application but could take well over 10kB in a complex application. Examples given later in this document assume a modest use of 4kB for control code.
- 2) **Technology Code.** This refers to the set of instructions that control the speech and audio technologies. Memory requirements for each desired technology are outlined on the following page. Some features, such as record and playback require off-chip RAM for data storage.
- 3) **Speech Recognition and Speech Synthesis.** Aside from memory requirements for the technology code, there are additional ROM requirements for speech recognition sets and speech synthesis. ROM requirements are outlined in Section 4 of this document.
- 4) **Four-Voice Music Synthesis and Sound Effects.** For four-voice music synthesis, memory requirements range from 5-10 kilobytes per instrument plus 100 bytes per second of music. ROM requirements for sound effects range from 20 bytes (a simple beep) to 5,000 bytes (an explosion) depending on the complexity of the sound effects.

Memory allotment on the RSC-364 is flexible. The customer can trade off memory usage between recognition sets, synthesis, music, or special applications depending on which features

are most important to their product. If the desired features will not fit in the on-chip ROM, the customer can use an RSC-364 and off-chip ROM.

ROM Requirements - Technology Code

This section outlines the Read Only Memory (ROM) required to use the speech and audio technologies available with the RSC-300/364 and Sensory Speech 6.0 Technology. Note that other RSC chips have different memory maps. These technologies are implemented by using one or more of the technology modules provided in the RSC-364 Development Kit software library. The following table lists each technology module, a brief description of its function, its code size (in kilobytes), and which technologies it is used by. For example, an application using Speaker-Independent Recognition and Speaker-Dependent Recognition will only require a total of 15.5 kilobytes of memory (Pattern Generation and Fixed-Point Math are only included once). The optional technology modules listed in the bottom table can be used with the identified Sensory technologies. **For applications using multiple technologies, it is only necessary to include each technology module once. Note that some features, such as record and playback, may also require off-chip RAM.**

Technology Module	Description	Approx Size (kB)	SI	SD	SV	DRT	WS	RP	Talk	Music	TTone
Patgen	Pattern Generation	8.0	√	√	√	√	√				
SI	Speaker-Independent	1.7	√			√					
SD	Speaker-Dependent	2.5		√	√	√	√				
SV	Speaker Verification	0.3			√						
DRT	Dual Recognition Technology	0.9				√					
WS	Wordspot	2.6					√				
RP	Record and Playback	7.4 ¹						√ ¹			
Talk	Speech Synthesis	3.4							√		
Music	Music Synthesis ⁵	1.7								√ ⁵	
TTone	Touch-tone Synthesis	0.6									√
Math(1)	Integer Math	0.8	√	√	√	√	√	√	√	√	
Math(2)	Floating-point Math	2.1	√		√	√		√			
Total Sizes			12.6	11.3	13.7	20.0	13.9	10.3	4.2	2.5	0.6

Optional	Technologies										
RP	Simultaneous Record and Pattern Generation	8.0	(√ ⁶)	(√ ⁶)	(√ ⁶)	(√ ⁶)		(√ ⁶)			
CL ²	Continuous Listening	0.3	(√)								
SEEP ³	Sample Serial EEPROM	1.1		(√)	(√)						
Flash ⁴	Sample Flash RAM	0.8		(√)	(√)	(√)		(√)			
RS232	RS232 drivers	0.3									
DRT Digits Weights	DRT Digits weightset	4.0				(√)					
SI Digits Weights	Speaker Independent weightset	4.3	(√)			(√)					

1. Size of Record and Playback code can be reduced if 3-bit and/or 2-bit compression is not used. Add Patgen size if thresholding is used.
2. Continuous Listening specified for Speaker-Independent Recognition only.
3. Serial EEPROM is optional for Speaker-Dependent Recognition, fast digits DRT, or Speaker Verification technologies.
4. Flash RAM code optional for Record and Playback, Speaker-Dependent Recognition, fast digits DRT, Speaker Verification and Speaker Adaptive technologies.
5. Music Synthesis also requires music notes and the data, which can be as much as 50kB.
6. Simultaneous Pattern Generation and recording is optional on the RSC-300/364.

ROM Requirements-Speech Recognition and Synthesis

Speaker-Independent Recognition	1.5-5 kilobytes per set
Speech Synthesis	500-1500 bytes per word
Music Synthesis	5-10 kilobytes per instrument plus 100 bytes per second

For each speaker-independent recognition set (up to 10 words per set), there is a 2-5kB memory requirement. The size of the set (i.e., 2 words vs. 8 words) will have a small effect on ROM usage. Speaker-dependent recognition and the password feature both require off-chip memory (~100 Bytes / word).

For estimation purposes, we generally use .5 kBytes per word (8,000 bits per second of speech, one-half second per word). This will be changed depending on the speech quality desired.

Memory Examples

The RSC-364 can address virtually unlimited memory, greatly extending the functionality of the chip. This section will help illustrate the use of memory.

The first is an application with several recognition sets and some synthesis. The second is an application with an emphasis on synthesis without the record and playback feature. Despite the fact that both applications have very different parameters, both use 128kB of ROM.

Hypothetical application: Recognition intensive, some synthesis, no other features.

RSC-364 + 64kB off-chip	128kB
Use	Size
Control Code	3kB
SI Recognition (technology code)	12kB
Recognition Weights (6 sets)	24kB
Synthesis (technology code)	4kB
Synthesis (150 words, 10 SFX)	85kB
Total	128kB

Hypothetical application: Synthesis intensive, minimal recognition, no other features.

RSC-364 + 64kB off-chip	128kB
Use	Size
Control Code	4kB
SI Recognition (technology code)	12kB
Recognition Weights (2 sets)	8kB
Synthesis (technology code)	4kB
Synthesis (200 words, no SFX)	100kB
Total	128kB

Hypothetical application: Record & Playback feature, balance between recognition and synthesis.

RSC-364	64kB
Use	Size
Control Code	6kB
SI Recognition (technology code)	12kB
Recognition (2 sets)	8kB
Synthesis (technology code)	4kB
Synthesis (30 words / 2 SFX)	24kB
Other: Record & Playback*	10kB
Total	64kB

Hypothetical application: SI and SD recognition and synthesis, no other features.

RSC-364	64kB
Use	Size
Control Code	3kB
SI Recognition (technology code)	12kB
SI Recognition (4 sets)	16kB
SD Recognition**	4kB
Synthesis (technology code)	4kB
Synthesis (50 words)	25kB
Total	64kB

*Record & Playback requires off-chip RAM to store the recordings.

**SD recognition requires off-chip RAM to store more than 6 template.

RAM and Other Read/Write Memories

The above sections only discuss the use of ROM (Read Only Memory). Systems using record and playback, password, and/or speaker-dependent functions require read and write memory.

The RSC-300/364, with its parallel interface memory busses, can address SRAM, which is volatile, or Flash, which is non-volatile. “Volatile” refers to memory that cannot retain stored information unless there is power to keep the memory active. If power to volatile memory is disrupted, any stored information will be lost. When designing the product, you must decide if anything that the chip learns must be stored for an extended period of time and coordinate with the Product Design Engineer to ensure that this will be possible.

The following table outlines what types of files can be stored and accessed through parallel and serial buses.

	Password/Dependent Weights	Gameplay Variables	Record & Playback files
Parallel bus	Yes	Yes	Yes*
Serial bus	Yes	Yes	No

* Assumes write speed and block size is appropriate



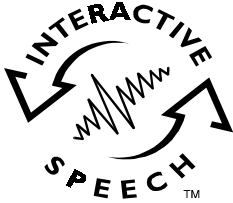
S E N S O R Y

521 East Weddell Drive
Sunnyvale, CA 94089

TEL: (408) 744-9000
FAX: (408) 744-1299

IMPORTANT NOTICE

Reasonable efforts have been made to verify the accuracy of information contained herein, however no guarantee can be made of accuracy or applicability. Sensory reserves the right to change any specification or description contained herein.



Quick Synthesis Instructions

Quick Synthesis Instructions

Overview

The Sensory Quick Synthesis Tool is designed to help create and manage speech synthesis for Sensory RSC microcontroller applications. It can be done quickly (within 15 minutes) and without the use of a Sensory linguist. Audio files created with Quick Synthesis will have somewhat lower quality sound and will require more memory than those created by Sensory. Quick Synthesis is often used to generate speech synthesis for prototypes or moderate volume production, where development time and cost must be minimized. Once a product goes to production, Sensory-processed files can be generated to yield the highest audio quality, while minimizing the amount of required memory.

This tool supports all RSC series processors, as well as Sensory's Voice Extreme™ ASSP. It has been tested on Win95/98 platforms only.

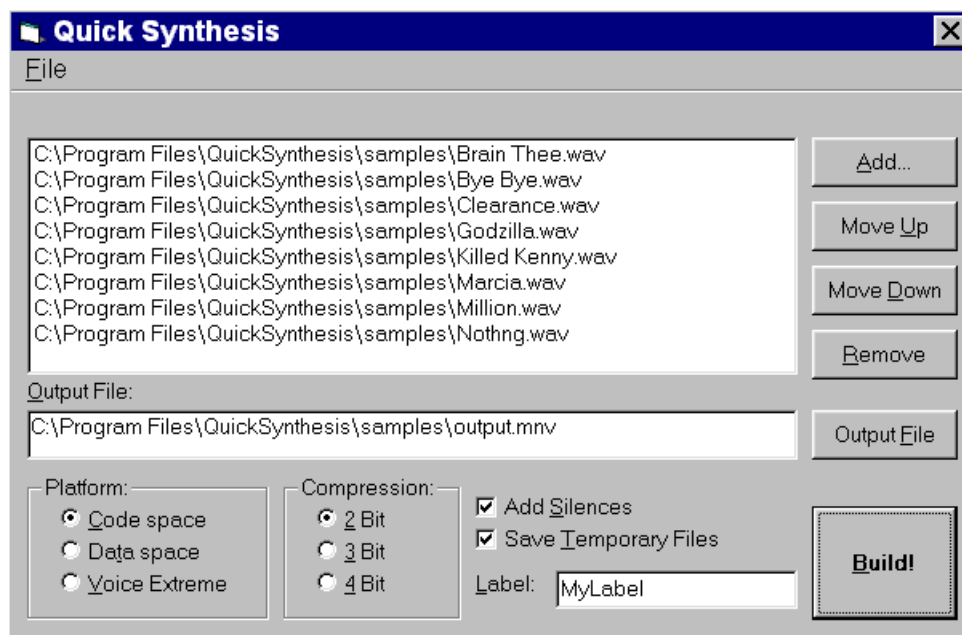
Installation

Run setup.exe and follow the installation instructions. The default installation folder is C:\Program Files\QuickSynthesis. A program group will be installed in your Windows Start menu.

Initial setup

The first time you run Quick Synthesis you will need to indicate where certain files are located. The first file is optional, the second is mandatory. Pull down the "Files" menu and select the "Preferences" option to set these program defaults.

1. If you have a preferred WAV file editor, you can indicate the path and filename for it. Click on the top "Set Path" button and navigate to the folder and executable file for this function.
2. You will need to indicate where the file "quicksyn.exe" is located in order to build speech synthesis files. Click on the bottom "Set Path" button and navigate to where the Sensory Quick Synthesis GUI has been installed. Click on "quicksyn.exe" and press "Save".



Features

- **.WAV file list:** This list box shows all the .WAV files that have been added to this project. If you select a file and then press the right-button on your mouse, a pop-up menu will be displayed which will allow you to play or edit this file.
- **Add button:** Use this button to add .WAV files to your project. A dialog box will be opened to allow you to locate and select files. Multiple selections in a folder are allowed. The format of the .WAV files must be mono, 22050 samples/second, and 16 bits/sample. If a selected file does not conform to this format, an error message will be generated and the file will not be included. Open up the file in your preferred .WAV editor and reformat the files to the correct specifications.
- **Move Up/Down button:** Use these buttons to organize the .WAV files in your project. Multiple files may be moved up or down.
- **Remove button:** Use this button to remove .WAV files from your project. You will be prompted to confirm all removals.
- **Output File:** Select the output file that will be created during the build process. After a build, six primary output files will be created:
 1. <output>.mnv – This is the raw PCM output data.
 2. <output>.a – This is a source file which may be assembled with the Sensory Assembler, SASM2 to create linkable object files for an application
 3. <output>.bin – This is an executable file which may be downloaded into a Sensory demo unit and will allow you to listen to each speech synthesis file to insure it is of good enough quality.
 4. <output>.txt – This is a text file of all the .WAV files which were used in this project. Note that this file does not include subdirectory folder names, just the file names in Windows 8.3 format.

5. <output>.vpi – This file may be added to an assembly language file to provide program labels for the CallTalkXXX macros.
 6. <output>.voc – This file is useful if you need to have the Sensory Linguistics staff further compress the speech synthesis.
- **Platform:** Select the format for the output files here. The first two options (code space, data space) are for RSC assembly language applications. The third option (Voice Extreme) is for Voice Extreme™ applications.
 - **Compression:** Select the amount of compression you wish to apply to the speech synthesis. The compression ratios are approximately as follows:
 2-Bit: 2700 bytes/second
 3-Bit: 3400 bytes/second
 4-Bit: 4100 bytes/second
 The compression level chosen will depend on the available memory for speech synthesis data and the desired speech quality. If, during the build process, the program calculates that the <output>.mnv file will exceed 64Kbytes, it will inform you of this and allow you to abort or proceed anyway with the build.
 - **Add Silences:** Selecting this option will add ten predefined silences to the output files. The silence durations are as follows: 20ms, 40ms, 75ms, 100ms, 160ms, 200ms, 400ms, 800ms, 1600ms, 3200ms. This will add approximately 50 bytes to the <output>.mnv file
 - **Save Temporary Files:** This option will cause some temporary files that are created during the build process to be saved. If this is selected, then for each .WAV file in the project a .RAW and .VSF file will also be created. Unless you are planning to have the Sensory Linguists further compress the speech synthesis files, these will most likely not be needed.
 - **Label:** This is the label which will be used in the final assembly or Voice Extreme application to refer to the starting address of the speech synthesis file.
 - **Build:** This is the button that causes the speech synthesis files to be created.
 - **File Menu:**



- New: Create a new speech synthesis project
- Open: Open a speech synthesis project
- Save: Save a speech synthesis project
- Save as: Save a speech synthesis project under a different name
- Preferences: See “Setup” instructions above
- Exit: Goodnight.

Creating a Quick Synthesis Project

When creating a Quick Synthesis Project, there are a few things you need to keep in mind:

- Keep in mind that each word will be synthesized separately, so you need to think about the ways that you will concatenate the phrases of the final synthesis. Generally, words that you use at the beginning of a sentence cannot be reused at the end without sounding choppy or out of place. Careful planning and recording of the word use can sometimes allow you to reuse words in this manner (for example digits), but you will probably have to record each such word twice.
- Any editing or changing of a .WAV file must be done in an editor before you begin the processing. There is no way to change the sounds after you start. You also need to make sure that you change the format of any .WAV file not in 22050 Hz, 16-bit, mono format.
- The waveform amplitude should be adjusted to have an amplitude range of approximately 16-bits (+/- 32,767). In CoolEdit this is accomplished by normalizing to 100% before saving.
- You need to do the recording in a quiet area to reduce the need for editing. Background noise will deteriorate the quality of the final compression.
- When selecting the files for your project, make sure to trim any background noise before and after the utterance. This “silence” will add to your memory allocation and give you extra pauses in the concatenation of your phrases.
- A .a (sentence) table is not created in this process. If you require such a table for your project, a Sensory linguist should be able to create this for you.

Here is a step-by-step procedure for creating your synthesis files.

- Go to “File” then “New”
- Click on the “Add” button and locate your files
- Select the files you would like to add (multiple files can be added) and click OK
 - Make sure that none of your files have filenames of more than 8 characters
- Put the files in the order you would like them listed (alpha-numeric order is usually best).
- Decide if you want to add silences and/or save your temporary files.
- Select your platform and compression preferences.
 - You will use “code” for most of your projects, unless you know for sure that you will be using Voice Extreme™ or have your synthesis in data space memory.

- 4-bit compression will take up the most space, but it will give you the best quality synthesis. A good approximation for 2-bit memory space is that 12 seconds of sound at 2-bit compression will use 32K of memory.
- Choose a label for your project.
- Choose your output file. Then name for the .MNV file should be the same as your label.
- Click “Build”.
 - You will be prompted at this time to save your project. Click “yes” and you will be prompted to name a .QSP file. Name this file the same as the .MNV file and the label.

Your files will then begin building. This process can take 5-15 minutes, depending on the size and number of the files, and your computer’s speed.

For the latest updates to this and other Sensory products, check out our website at www.VoiceActivation.com. For technical assistance, please send email to techsupport@sensoryinc.com.



S E N S O R Y

521 East Weddell Drive
Sunnyvale, CA 94089

TEL: (408) 744-9000
FAX: (408) 744-1299

IMPORTANT NOTICE

Reasonable efforts have been made to verify the accuracy of information contained herein, however no guarantee can be made of accuracy or applicability. Sensory reserves the right to change any specification or description contained herein.

DK264T/364 Motherboard Schematic

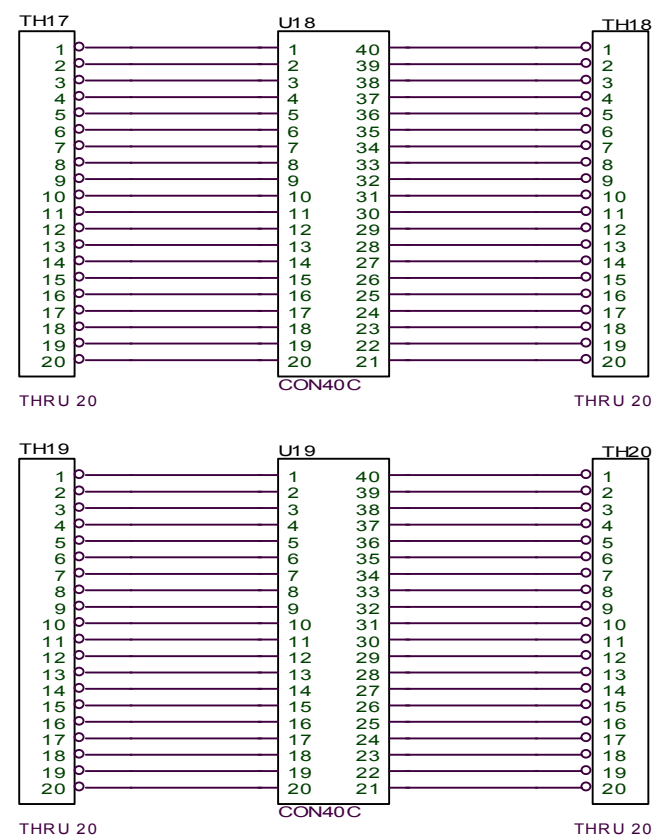
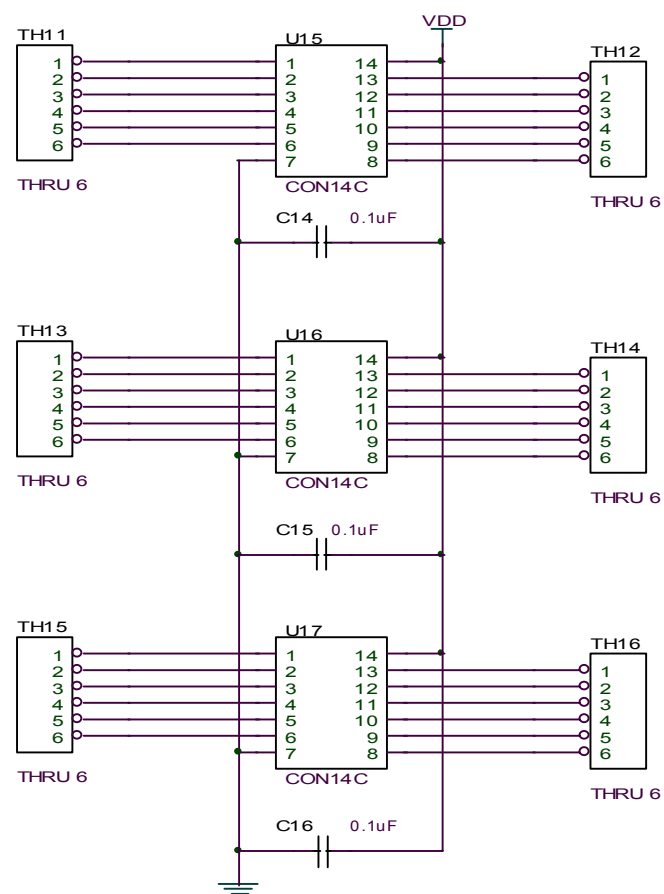
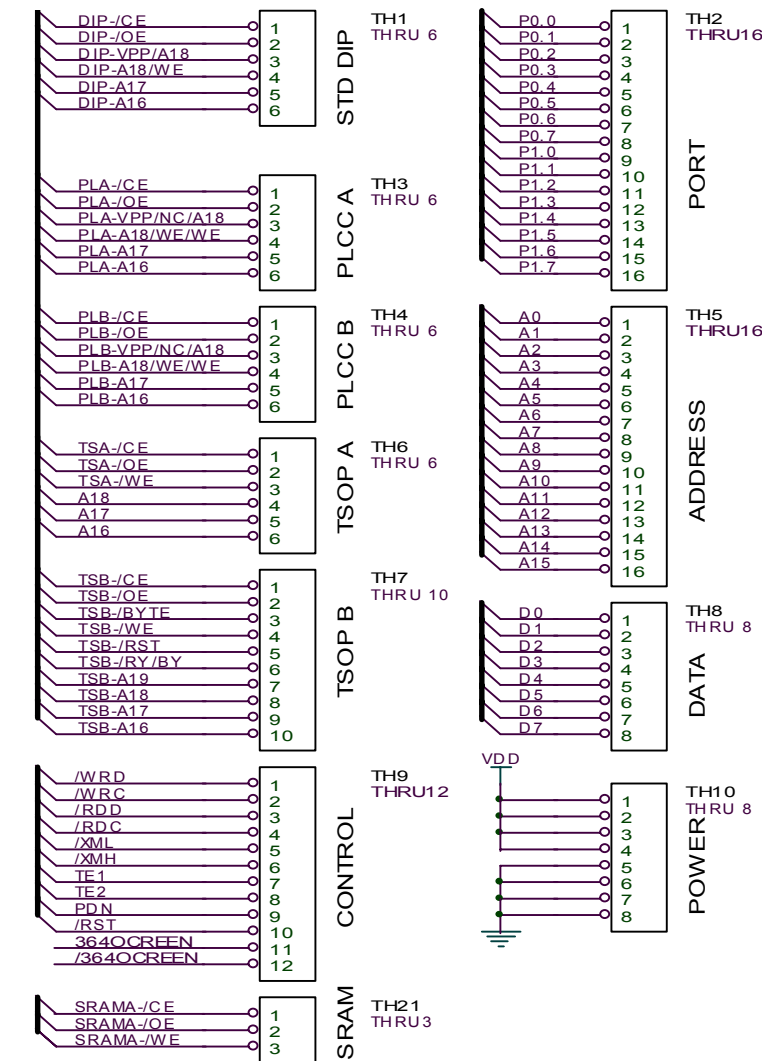
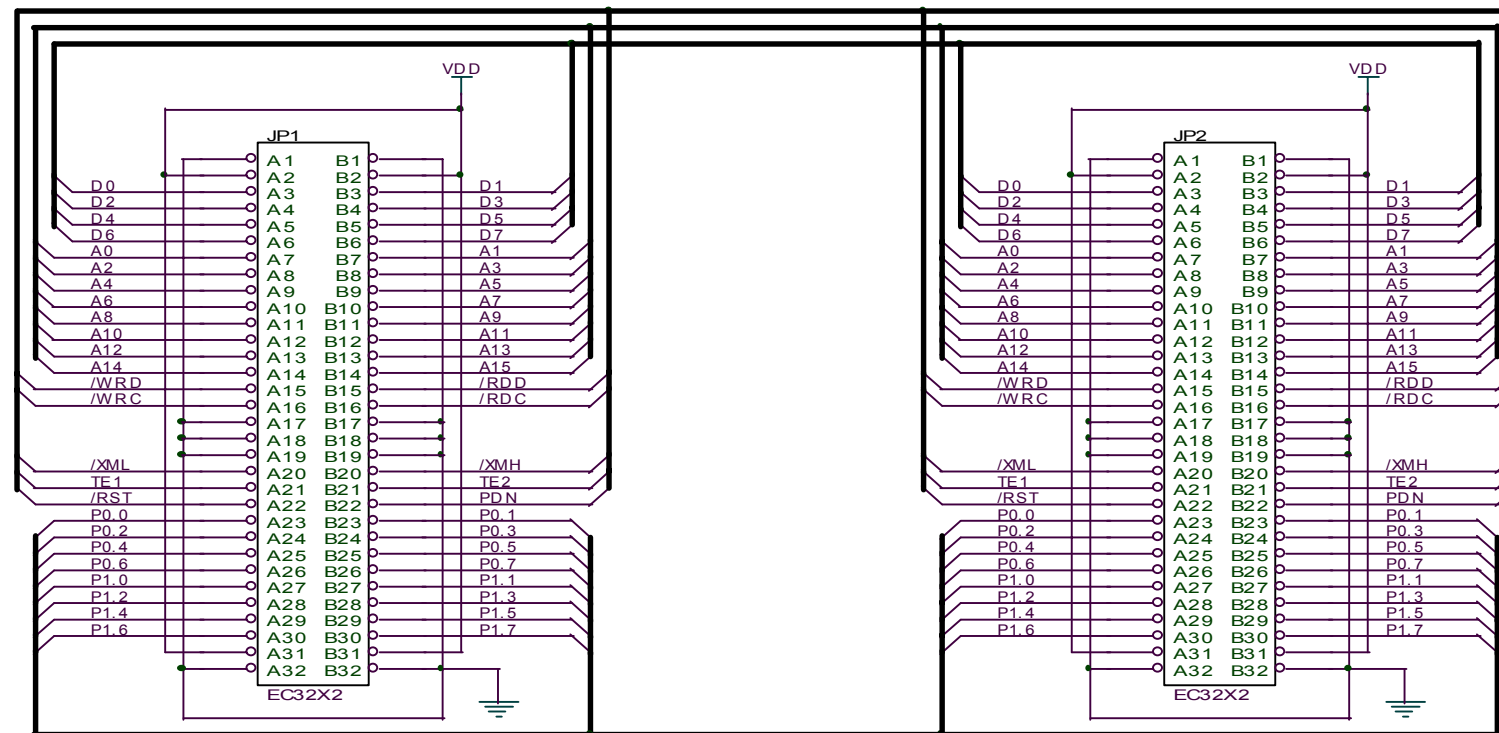
Component Labels:

- U1:** RSC384PLOC
- U2:** LM4832
- U3:** MAX3222E
- U4:** 74HC240
- U5:** 74HC240
- U6:** 74HC240
- U7:** 74HC240
- U8:** 74HC240
- U9:** 74HC240
- U10:** 74HC240
- U11:** 74HC240
- U12:** 74HC240
- U13:** 74HC240
- U14:** 74HC240
- U15:** 74HC240
- U16:** 74HC240
- U17:** 74HC240
- U18:** 74HC240
- U19:** 74HC240
- U20:** 74HC240
- U21:** 74HC240
- U22:** 74HC240
- U23:** 74HC240
- U24:** 74HC240
- U25:** 74HC240
- U26:** 74HC240
- U27:** 74HC240
- U28:** 74HC240
- U29:** 74HC240
- U30:** 74HC240
- U31:** 74HC240
- U32:** 74HC240
- U33:** 74HC240
- U34:** 74HC240
- U35:** 74HC240
- U36:** 74HC240
- U37:** 74HC240
- U38:** 74HC240
- U39:** 74HC240
- U40:** 74HC240
- U41:** 74HC240
- U42:** 74HC240
- U43:** 74HC240
- U44:** 74HC240
- U45:** 74HC240
- U46:** 74HC240
- U47:** 74HC240
- U48:** 74HC240
- U49:** 74HC240
- U50:** 74HC240
- U51:** 74HC240
- U52:** 74HC240
- U53:** 74HC240
- U54:** 74HC240
- U55:** 74HC240
- U56:** 74HC240
- U57:** 74HC240
- U58:** 74HC240
- U59:** 74HC240
- U60:** 74HC240
- U61:** 74HC240
- U62:** 74HC240
- U63:** 74HC240
- U64:** 74HC240
- U65:** 74HC240
- U66:** 74HC240
- U67:** 74HC240
- U68:** 74HC240
- U69:** 74HC240
- U70:** 74HC240
- U71:** 74HC240
- U72:** 74HC240
- U73:** 74HC240
- U74:** 74HC240
- U75:** 74HC240
- U76:** 74HC240
- U77:** 74HC240
- U78:** 74HC240
- U79:** 74HC240
- U80:** 74HC240
- U81:** 74HC240
- U82:** 74HC240
- U83:** 74HC240
- U84:** 74HC240
- U85:** 74HC240
- U86:** 74HC240
- U87:** 74HC240
- U88:** 74HC240
- U89:** 74HC240
- U90:** 74HC240
- U91:** 74HC240
- U92:** 74HC240
- U93:** 74HC240
- U94:** 74HC240
- U95:** 74HC240
- U96:** 74HC240
- U97:** 74HC240
- U98:** 74HC240
- U99:** 74HC240
- U100:** 74HC240

Pin Configurations:

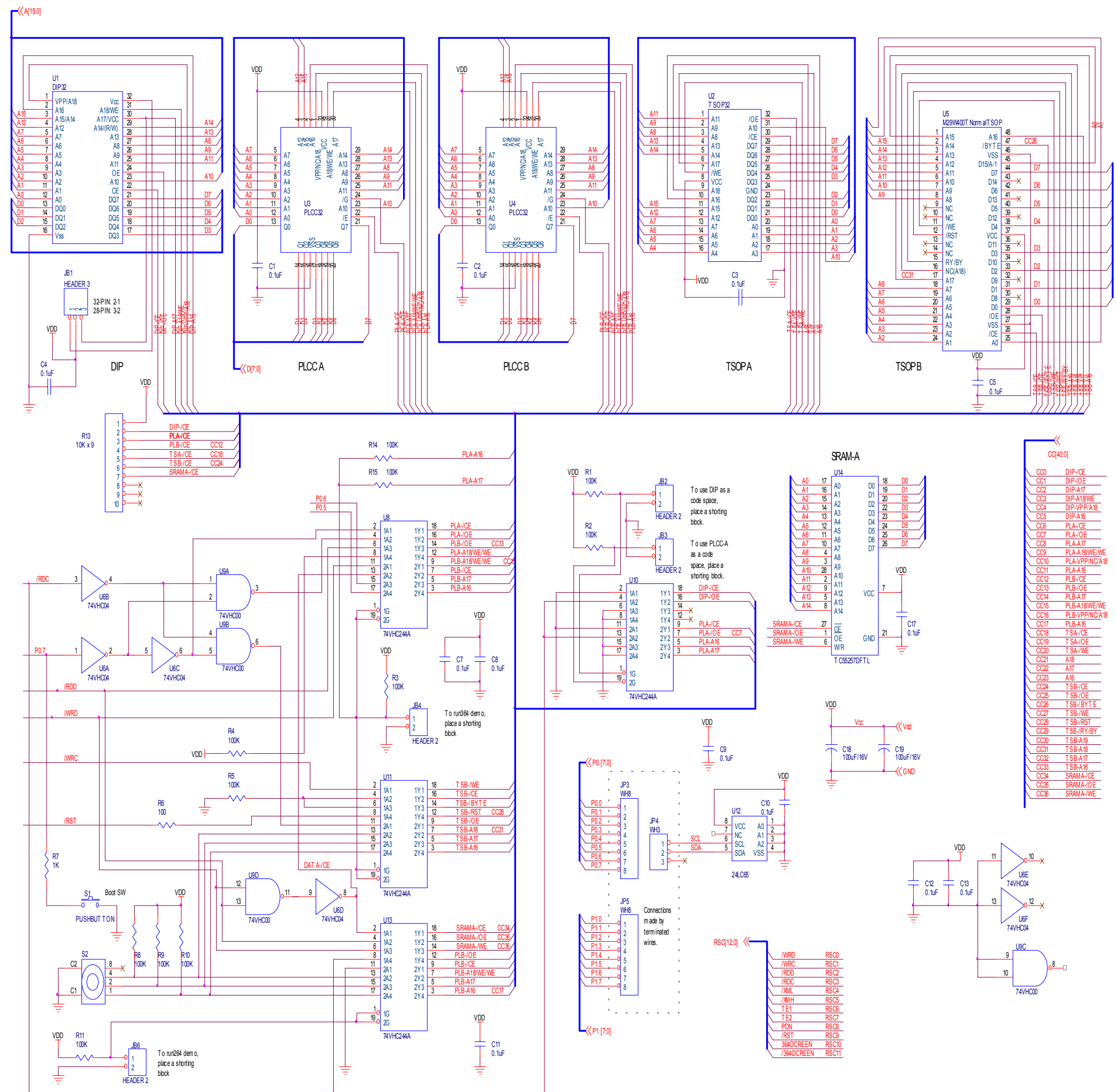
Component	Pin	Signal
U1 (RSC384PLOC)	1	INT
	2	CLK
	3	EXT
	4	OSC
	5	INT
	6	CLK
	7	EXT
	8	OSC
	9	INT
	10	CLK
	11	EXT
	12	OSC
	13	INT
	14	CLK
	15	EXT
	16	OSC
U2 (LM4832)	1	SD
	2	BYPASS
	3	+IN
	4	-IN
	5	VDD
	6	VDD
	7	VDD
	8	VDD
	9	VDD
	10	VDD
	11	VDD
	12	VDD
	13	VDD
	14	VDD
	15	VDD
	U3 (MAX3222E)	1
2		C1+
3		V+
4		C1-
5		C2+
6		C2-
7		V-
8		T2OUT
9		R2IN
10		R2OUT
11		T1IN
12		R1OUT
13		T1OUT
14		C2+
15		C2-
16		V+
U4 (74HC240)	1	A1
	2	A2
	3	A3
	4	A4
	5	Y1
	6	Y2
	7	Y3
	8	Y4
	9	G
	10	A1
	11	A2
	12	A3
	13	A4
	14	Y1
	15	Y2
	U5 (74HC240)	1
2		A2
3		A3
4		A4
5		Y1
6		Y2
7		Y3
8		Y4
9		G
10		A1
11		A2
12		A3
13		A4
14		Y1
15		Y2
U6 (74HC240)		1

DK264T/364 Memory Module Schematic 1



<i>Sensory, Inc.</i>			
Title DK4.0 Memory Module Board			
Size B	Document Number 70-0042		Rev C
Date:	Monday, March 22, 1999	Sheet	1 of 2

DK264T/364 Memory Module Schematic 2





CHAPTER 10 - SUPPORT

This chapter describes Sensory's RSC-364 Development Kit Support Policy and Limited Warranty.

1. RSC-364 Development Kit Support Policy

If you have questions about the RSC-364 Development Kit, first look in the provided documentation to find the answer. If you cannot find the answer, contact Sensory for technical support.

Sensory will provide 5 hours of technical support at no-charge on the Development Kit software and hardware. Additional support may be contracted from Sensory on an hourly basis under mutually agreeable terms and schedules.

Technical support is available via a toll call between 8:30 A.M. and 5:30 P.M. Pacific time, Monday through Friday, excluding holidays. Call (408) 744-9000 and ask for Development Kit support. Technical support is also available via fax or e-mail with a 48-hour turnaround time. Fax: (408) 744-1299, e-mail: TechSupport@SensoryInc.com

2. RSC-364 Development Kit Limited Warranty

The RSC-364 Development Kit is warranted against defects and workmanship for a period of 90 days from the date of product purchase. Sensory, Inc. will, at its option, either repair or replace a product that proves to be defective either upon receipt or through normal usage. If a Sensory Development Kit product has been obsoleted or is no longer in production and deemed non-repairable, Sensory will, at its option, provide an equivalent product or system for a nominal fee.

Sensory, Inc. warrants this Development Kit product, when properly installed and used, will execute its programmed instructions. However, Sensory, Inc. does not warrant that the operation of the Product, its firmware and software will be uninterrupted or totally error free. The Product must be returned to Sensory, Inc. for warranty service within the warranty period to the following address: Sensory, Inc., 521 E. Weddell Drive, Sunnyvale, CA 94089-2164. The Buyer will pay all shipping and other charges or assessments for the return of the Product to Sensory, Inc.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from maintenance performed by anyone other than Sensory, Inc., modifications made by Buyer or any third party, Buyer supplied software or interfacing, misuse, abuse, accident, mishandling, operation outside the environmental specifications for the Product, or improper setup or maintenance.

Exclusive Remedies

The remedies provided herein are Sensory's sole liability and Buyer's sole and exclusive remedies for breach of warranty. Sensory shall not be liable for any special, incidental, consequential, direct or indirect damages, whether based on contract, tort, or any legal theory. The foregoing warranty is in lieu of any and all other warranties, whether express, implied, or statutory, including but not limited to warranties of merchantability and suitability for a particular purpose.

3. Important Notices

Sensory reserves the right to make changes to or to discontinue any product or service identified in this publication at any time without notice for any

reason whatsoever. Sensory does not assume responsibility for use of any circuitry other than circuitry entirely embodied in a Sensory product. Information contained herein is provided gratuitously and without liability to any user. Reasonable efforts have been made to verify the accuracy of this information but no guarantee whatsoever is given as to the accuracy or as to its applicability to particular uses.

Applications described in this manual are for illustrative purposes only, and Sensory makes no warranties or representations that the Interactive Speech™ line of products will be suitable for such applications. In every instance, it must be the responsibility of the user to determine the suitability of the products for each application. Sensory products are not authorized for use as critical components in life support devices or systems.

Sensory conveys no license or title, either expressed or implied, under any patent, copyright, or mask work right to the Interactive Speech™ line of products, and Sensory makes no warranties or representations that the Interactive Speech™ line Interactive Speech™ of products are free from patent, copyright, or mask work right infringement, unless otherwise specified. Nothing contained herein shall be construed as a recommendation to use any product in violation of existing patents or other rights of third parties. The sale of any Sensory product is subject to all Sensory Terms and Conditions of Sales and Sales Policies.

521 E. Weddell Drive
Sunnyvale, CA 94089
TEL: (408) 744-9000
FAX: (408) 744-1299

© 2000 SENSORY
ALL RIGHTS RESERVED