MOTOROLA

Embedded SDK (Software Development Kit)

Customer Premises Equipment Alerting Signal (CAS) Library

SDK124/D
Rev. 2, 07/17/2002

MOTOROLA intelligence everywhere™ | digital dna™

Contents

About This Document

Chapter 1
Introduction

Chapter 2
Directory Structure

Chapter 3
CAS Library Interfaces

Chapter 4
Building the CAS Library

Chapter 5
Linking Applications with the CAS Library

Chapter 6
CAS Applications

Chapter 7
License

List of Tables

List of Figures

List of Examples

About This Document

This manual describes the Customer Premises Equipment Alerting Signal, (CAS), detection algorithm for use with Motorola's Embedded Software Development Kit, (SDK).

Audience

This document targets software developers implementing the CAS detection function within software applications.

Organization

This manual is arranged in the following sections:

- **Chapter 1, Introduction**—provides a brief overview of this document
- **Chapter 2, Directory Structure**—provides a description of the required core directories
- **Chapter 3, CAS Library Interfaces**—describes all of the CAS Library functions
- **Chapter 4, Building the CAS Library**—tells how to execute the system library project build
- **Chapter 5, Linking Applications with the CAS Library**—describes the organization of the CAS Library
- **Chapter 6, CAS Applications**—describes the use of CAS library through test/demo applications
- **Chapter 7, License**—provides the license required to use this product

Suggested Reading

We recommend that you have a copy of the following references:

- *DSP56800 Family Manual*, DSP56800FM/AD
- *DSP56824 User's Manual*, DSP56824UM/AD
- *Inside CodeWarrior: Core Tools*, Metrowerks Corp.

Conventions

This document uses the following notational conventions:

Typeface, Symbol or Term	Meaning	Examples
`Courier Monospaced Type`	Commands, command parameters, code examples, expressions, datatypes, and directives	...`*Foundational include files`... ...a data structure of type `vad_tConfigure`...
Italic	Calls, functions, statements, procedures, routines, arguments, file names and applications	...the *pConfig* argument... ...defined in the C header file, *aec.h*... ...makes a call to the *Callback* procedure...
Bold	Reference sources, paths, emphasis	...refer to the **Targeting DSP56824 Platform** manual.... ... see: **C:\Program Files\Motorola\Embedded SDK\help\tutorials**
Bold/Italic	Directory name, project name	...and contains these core directories: ***applications*** contains applications software.... ...CodeWarrior project, ***3des.mcp***, is.....
Blue Text	Linkable on-line	...refer to **Chapter 7**, License...
Number	Any number is considered a positive value, unless preceded by a minus symbol to signify a negative value	3V -10 DES^{-1}
ALL CAPITAL LETTERS	Variables, directives, defined constants, files libraries	INCLUDE_DSPFUNC #define INCLUDE_STACK_CHECK
Brackets [...]	Function keys	...by pressing function key [F7]...
Quotation marks "... "	Returned messages	...the message, "Test Passed" is displayed.... ...if unsuccessful for any reason, it will return "NULL"....

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document. As this template develops, this list will be generated from the document. As we develop more group resources, these acronyms will be easily defined from a common acronym dictionary. Please note that while the acronyms are in solid caps, terms in the definition should be initial capped ONLY IF they are trademarked names or proper nouns.

CAS Customer premises equipment Alerting Signal

DSP Digital Signal Processor or Digital Signal Processing

FFT Fast Fourier Transforms

FIR Finite Impulse Response

I/O Input/Output

IDE	Integrated Development Environment
IIR	Infinite Impulse Response
LSB	Least Significant Bit
MAC	Multiply/Accumulate
MIPS	Million Instructions Per Second
MSB	Most Significant Bit
OnCE™	On-Chip Emulation
OMR	Operating Mode Register
PC	Program Counter
SDK	Software Development Kit
SP	Stack Pointer
SPCS	Stored Program Controlled-switching System
SPI	Serial Peripheral Interface
SR	Status Register
SRC	Source

References

The following sources were used to produce this book:

1. *DSP56800 Family Manual*, DSP56800FM/AD
2. *DSP56824 User's Manual*, DSP56824UM/AD
3. *Embedded SDK Programmer's Guide*
4. *SR-TSV-002476 Bellcore Standard*

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 1
Introduction

Welcome to Motorola's Family of Digital Signal Processors, (DSPs). This document describes the Customer premises equipment Alerting Signal, (CAS), detection Library, which is a part of Motorola's comprehensive Software Development Kit, (SDK), for its DSPs. In this manual, you will find all the information required to use and maintain the CAS Library interface and algorithms.

Motorola provides these algorithms to you for use on the Motorola DSPs to expedite your application development and thus reduce the time it takes to bring your own products to market.

Motorola's CAS library is licensed for your use at no charge on Motorola processors. Please refer to the standard Software License Agreement in **Chapter 7** for license terms and conditions; please consult with your Motorola representative for premium product licensing.

1.1 Quick Start

Motorola Embedded SDK is targeted to a large variety of hardware platforms. To take full advantage of a particular hardware platform, use **Quick Start** from the **Targeting DSP568xx Platform** documentation.

For example, the **Targeting DSP56824 Platform** manual provides more specific information and examples about this hardware architecture. If you are developing an application for the DSP56824EVM board or any other DSP56824 development system, refer to the **Targeting DSP56824 Platform** manual for **Quick Start** or other DSP56824-specific information.

1.2 Overview of CAS Detection

To transmit data in the off-hook mode, a stable call must be interrupted and a clear, voice-free channel established. The Stored Program Controlled-switchng System (SPCS) initiates the process by muting transmission to and from the far end and transmitting the alerting signal to the service customer. The alerting signal consists of a Subscriber Alerting Signal (SAS) and a Customer Premises Equipment Alerting Signal (CAS). The SAS alerts the subscriber that there are new calls waiting.

1.2.1 Background

The CAS is a machine-detectable signal used to alert the Customer Premesis Equipment (CPE) to prepare for data reception.Typically, data transmission is the transmission of the Caller-ID signal between the Central office and the customer's equipment. Before a data transmission starts, the Central Office sends the CAS to the CPE. When the CPE detects the CAS, the CPE is muted and returns a Dual Tone Multiple

Frequency (DTMF) -D tone to the Central Office as acknowledgement. The data transmission begins only after SPCS detects the DTMF-D tone sent by the CPE. The electrical characteristics of CAS are detailed in **Table 1-1**.

Table 1-1. CAS Electrical chracteristics

Frequency Limits	Lower Tone : 2130 Hz +/- 0.5% Upper Tone : 2750 Hz +/- 0.5%
Dynamic Range	-32 to -14 dBm per tone
Twist or Power Differential Within Dynamic Range	0 to 6 dB between tones
Tone Duration at CPE	75 to 85 ms

Note: Signal levels are referenced at 600 ohm termination at the CPE tip and ring interface.

1.2.2 Features and Performance

The CAS library is multichannel and re-entrant.

For details on Memory and MIPS for a particular DSP, refer to the **Libraries** chapter of the appropriate Targeting manual.

Chapter 2
Directory Structure

2.1 Required Core Directories

Figure 2-1 details required platform directories:

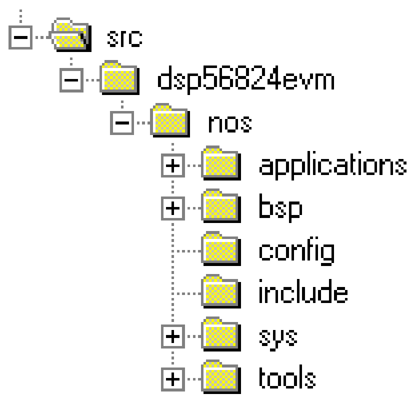

Figure 2-1. Core Directories

As shown in Figure 2-1, DSP56824EVM has no operating system (nos) support and contains these core directories:

- ***applications*** contains applications software that can be exercised on this platform
- ***bsp*** contains board support package specific for this platform
- ***config*** contains default hardware/software configurations for this platform
- ***include*** contains SDK header files which define the Application Programming Interface
- ***sys*** contains required system components
- ***tools*** contains useful utilities used by system components

There are also optional directories that include domain-specific libraries.

2.2 Optional (Domain-Specific) Directories

Figure 2-2 demonstrates how the CAS Detect algorithm is encapsulated in the domain-specific directories under the directory, ***telephony***.

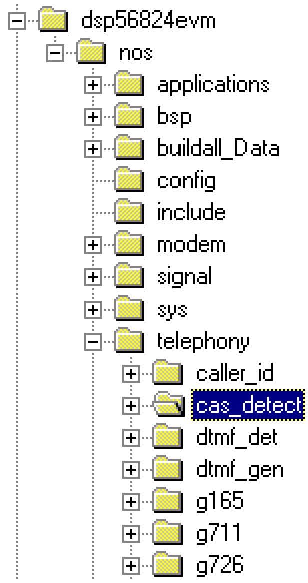

Figure 2-2. DSP56824 Directories

The ***telephony*** directory includes telephony specific algorithms. Figure 2-3 shows the ***cas_detect*** directory structure under ***telephony*** directory.

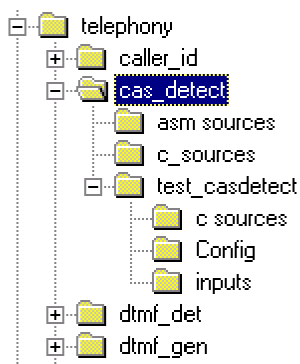

Figure 2-3. *cas_detect* Directory Structure

The ***cas_detect*** directory includes these sub-directories:

- ***asm sources*** includes asm sources required for CAS Detect
- ***c_sources*** includes the APIs for CAS Detect
- ***test_casdetect*** includes C sources and configuration files necessary for testing CAS library modules
  - — ***c sources*** contains an example test code
  - — ***Config*** contains the configuration files ***appconfig.c***, ***appconfig.h*** and ***linker.cmd*** specific to CAS Detect
  - — ***inputs*** contains the test vectors for testing CAS Detect

Chapter 3
CAS Library Interfaces

3.1 CAS Services

The CAS Detect library detects the CAS tone sent by the exchange when the user is off-hook. The data to be supplied must be in 16-bit word fixed point (1.15) format, as shown below:

s MSB	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i LSB

i = information bit

s = sign

3.2 Interface

The C interface for the CAS Detect library services is defined in the C header file *casDetect.h*, shown in **Code Example 3-1** as a reference:

Code Example 3-1. C Header File *casDetect.h*

```
/* File: casDetect.h */

#ifndef __CASDETECT_H
#define __CASDETECT_H

/*
   CPE Alerting Signal Detection interface
*/

/***************************
 Foundational Include Files
****************************/

#include "port.h"
```

```
/********************************************
 #define for CAS Detect flags, returns from
    the CAS process function. CAS_PRESENT is
    returned whenver valid CAS is detected
    from the frame of 80 samples, otherwise
    CAS_NOT_PRESENT is returned.
*********************************************/

#define CAS_PRESENT      1
#define CAS_NOT_PRESENT  0

/********************************************
     Structure for CAS Detect Configuration
*********************************************/

typedef struct
{
    Int16 *In_Context_buf;
    UInt16 context_buf_length;
    Word16 *casdatastruct;
}casDetect_sHandle;

/***************************
 CAS Detect Function Prototypes
****************************/

EXPORT casDetect_sHandle * casDetectCreate (void);
EXPORT void casDetectInit (casDetect_sHandle * pCasDetect);

EXPORT Result casDetectProcess (casDetect_sHandle * pCasDetect,
                                Int16 *pSamples,
                                UInt16 NumSamples);

EXPORT void casDetectDestroy (casDetect_sHandle * pCasDetect);

#endif
```

3.3 Specifications

The following pages characterize the CAS detect library functions.

Function arguments for each routine are described as *in*, *out*, or *inout*. An *in* argument means that the parameter value is an input only to the function. An *out* argument means that the parameter value is an output only from the function. An *inout* argument means that a parameter value is an input to the function, but the same parameter is also an output from the function.

Typically, *inout* parameters are input pointer variables in which the caller passes the address of a pre-allocated data structure to a function. The function stores its results within that data structure. The actual value of the *inout* pointer parameter is not changed.

3.3.1 *casDetectCreate*

Call(s):

```
casDetect_sHandle * casDetectCreate (void)
```

Required Header: "casDetect.h"

Arguments:

Table 3-1. *casDetectCreate* Arguments

void		No input arguments required for the call to *casDetectCreate*

Description: The *casDetectCreate* function creates an instance of the CAS. During the *casDetectCreate* call, all dynamic resources required by the CAS algorithm are allocated. A Total of 406 words of external data memory are allocated per instance. The library allocates dynamic memory using the *mem* library, shown in **Code Example 3-2**. The library is **multichannel** and **re-entrant**.

Code Example 3-2. *mem* Library

```
#include "mem.h"
#include "casDetect.h"
#define FRAME_SZ 80

casDetect_sHandle *casDetectCreate (void)
{
      casDetect_sHandle *pCasDetect;

      /* Allocate the memory for the handle structure*/

      pCasDetect = (casDetect_sHandle *) memMallocEM (sizeof (casDetect_sHandle));

      if (pCasDetect == NULL) return (NULL);

      /* Allocate memory for the In_Context_buf */

      pCasDetect->In_Context_buf = (Int16 *) memMallocEM (FRAME_SZ * sizeof(Int16));
      if (pCasDetect->In_Context_buf == NULL) return (NULL);

      return (pCasDetect);
}
```

For details on the *casDetect_sHandle* structure, refer to **Code Example 3-1**.

If a *casDetectCreate* function is called to create an instance, then *casDetectDestroy*, detailed in **Section 3.3.4**, should be used to destroy the instance.

Alternatively, the user can allocate memory statically, which requires duplicating all statements in the *casDetectCreate* function. In this case, the user can call the *casDetectInit* function directly, bypassing the *casDetectCreate* function. If the user dynamically allocates memory without calling *casDetectCreate*, then the user himself must destroy the memory allocated.

Returns: Upon successful completion, the *casDetectCreate* function will return a pointer to the specific instance of CAS created. If *casDetectCreate* is unsuccessful for any reason, it will return "NULL".

Special Considerations:

- The CAS Detect application is multichannel and re-entrant.
- If *casDetectCreate* is called, then the user need not call *casDetectInit* function as it is called internally in the *casDetectCreate* function.

In **Code Example 3-3**, the application creates an instance of CAS.

Code Example 3-3. Use of the *casDetectCreate* Interface

```
#include “casDetect.h”

void testCasDetect (void)
{
   casDetect_sHandle *pCasDetect;

   /* Create and initialize CAS Detect instance */
   pCasDetect = casDetectCreate ();

}
```

3.3.2 *casDetectInit*

Call(s):

```
void casDetectInit (casDetect_sHandle * pCasDetect)
```

Required Header: "casDetect.h"

Arguments:

Table 3-2. *casDetectInit* Arguments

pCasDetect	in	Handle to an instance of CAS

Description: The *casDetectInit* function will initialize the CAS Detect algorithm. Before calling the *casDetectInit* function, a *casDetect* instance must be created. This instance can be created by calling the *casDetectCreate* function, or alternatively, if memory is statically allocated, the *casDetectCreate* function need not be called. Please refer to **Section 3.3.1** for details on dynamic and static memory allocation. During the initialization, all resources will be set to their initial values in preparation for the CAS Detect operation.

Returns: None

Special Considerations:

- No configuration parameters need to be set by the user before a call to *casDetectCreate* and *casDetectInit*

In **Code Example 3-4**, the application creates an instance of CAS Detect. The instance is passed to the *casDetectInit* function.

Code Example 3-4. Use of the *casDetectInit* Interface

```
#include "casDetect.h"

void testCasDetect (void)
{
   casDetect_sHandle *pCasDetect;

   ...

   pCasDetect = casDetectCreate (); /* This function itself initializes the
                                       instance */

   ....
}
```

3.3.3 *casDetectProcess*

Call(s):

```
Result casDetectProcess (casDetect_sHandle * pCasDetect,
                         Int16 *pSamples,
                         UInt16 NumSamples);
```

Required Header: "casDetect.h"

Arguments:

Table 3-3. *casDetectProcess* Arguments

pCasDetect	in	Handle to an instance of CAS
pSamples	in	Pointer to the samples buffer to be used by the CAS Detect algorithm
NumSamples	in	The number of samples to be processed

Description: The *casDetectProcess* function processes on a frame basis internally and returns one of the flags, (CAS_PRESENT or CAS_NOT_PRESENT). The user can call the *casDetectProcess* function any number of times, as long as there are samples to be processed.

Returns: Always returns CAS_PRESENT("1") or CAS_NOT_PRESENT ("0").

Special Considerations:

- Callback is not implemented, since the CAS code terminates the detection process as soon as it detects the valid CAS tone
- The *casDetectProcess* function does not process the remaining samples once it has detected a valid CAS tone

Code Example: See **Code Example 3-5** for details on using the *casDetectProcess* function.

Code Example 3-5. Use of the *casDetectProcess* Interface

```
#include "casDetect.h"

void testCasDetect (void)
{

     Result res;
     Int16 InBuf[160];
     casDetect_sHandle *pCasDetect;

     /* Create and initialize CAS Detect instance */
     pCasDetect = casDetectCreate ();

     res = casDetectProcess (pCasDetect, InBuf, 160);
     if (res == CAS_PRESENT) printf ("CAS DETECTED");

     res = casDetectProcess (pCasDetect, InBuf, 160);
     if (res == CAS_NOT_PRESENT) printf ("CAS NOT DETECTED");

     ....
}
```

3.3.4 *casDetectDestroy*

Call(s):

```
void casDetectDestroy (casDetect_sHandle * pCasDetect)
```

Required Header: “casDetect.h”

Arguments:

Table 3-4. *casDetectDestroy* Arguments

pCasDetect	in	Handle to an instance of CAS generated by a call to *casDetectCreate*

Description: The *casDetectDestroy* function destroys the instance of the *CASDetect* originally created by a call to *casDetectCreate*.

Returns: None

Special Considerations: None

Code Example: See **Code Example 3-6** for details on using *casDetectProcess* function.

Code Example 3-6. Use of the *casDetectDestroy* Interface

```
#include “casDetect.h”

void testCasDetect (void)
{

     Result res;
     Int16 InBuf[160];
     casDetect_sHandle *pCasDetect;

     /* Create and initialize CAS Detect instance */
     pCasDetect = casDetectCreate ();

     res = casDetectProcess (pCasDetect, InBuf, 160);
     if (res == CAS_PRESENT) printf (“CAS DETECTED”);

     res = casDetectProcess (pCasDetect, InBuf, 160);
     if (res == CAS_NOT_PRESENT) printf (“CAS NOT DETECTED”);

     casDetectDestroy (pCasDetect);

     ....
}
```

Chapter 4
Building the CAS Library

4.1 Building the CAS Library

The CAS library combines all of the components described in previous sections into one library: *cas_detect.lib*. To build this library, a Metrowerks' CodeWarrior project, ***cas_detect.mcp***, is provided. This project and all the necessary components to build the CAS library are located in the *...\nos\telephony\cas_detect* directory of the SDK directory structure.

There are two methods to execute a system library project build: dependency build and direct build.

4.1.1 Dependency Build

Dependency build is the easiest approach and requires no additional work on the user's part. If you add the CAS library project, ***cas_detect.mcp***, to your application project, as shown in **Figure 4-1**, the CAS library will automatically build when the application is built.

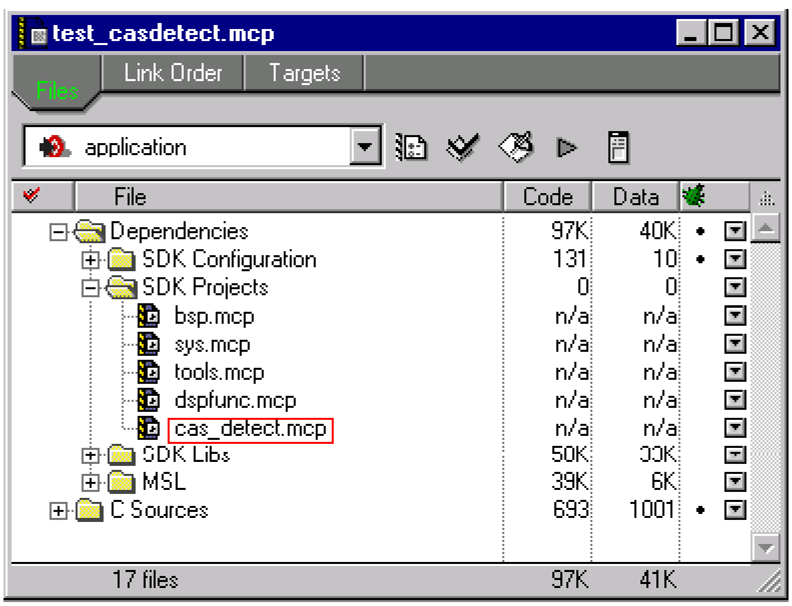

Figure 4-1. Dependency Build for CAS Library

4.1.2 Direct Build

Direct build allows you to build a CAS Detect library independently of any other build. Follow these steps:

Step 1. Open *cas_detect.mcp* project, as shown in **Figure 4-2**:

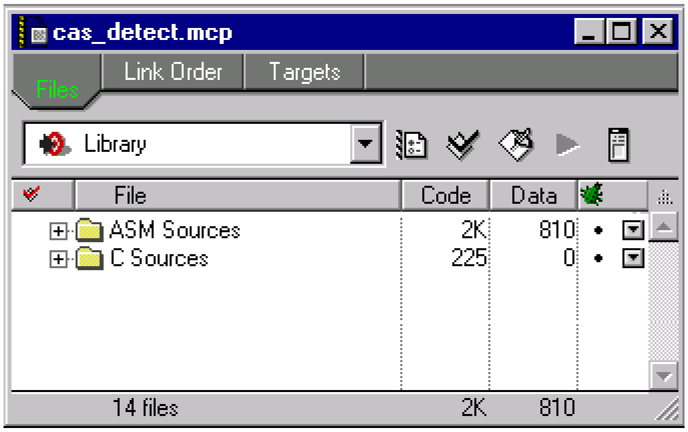

Figure 4-2. cas_detect.mcp Project

Step 2. Execute the build by pressing function key [F7] or by choosing *Make* from the Project menu; see **Figure 4-3**.

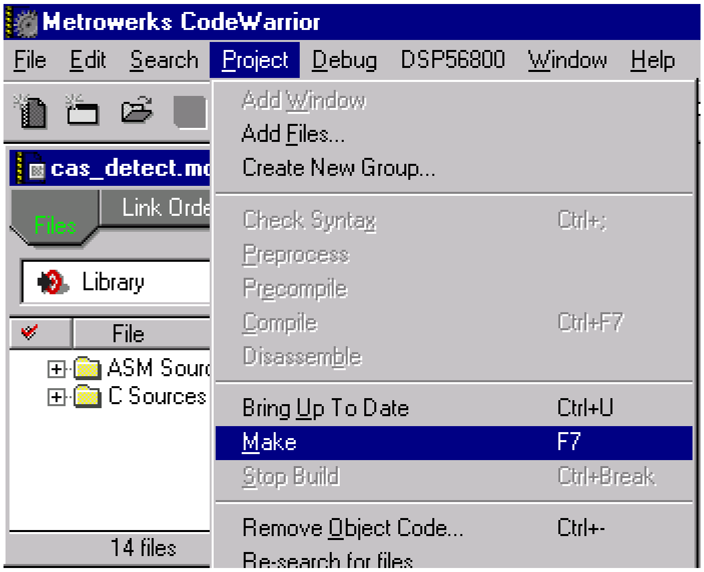

Figure 4-3. Execute Make

At this point, ifthe build is successful, the *cas_detect.lib* library file is created in the ***...\nos\telephony\cas_detect\Debug*** directory.

Chapter 5
Linking Applications with the CAS Library

5.1 CAS Library

The library includes APIs, which provide interface between the user application and the CAS modules. To invoke CAS, APIs must be called in this order:

— *casDetectCreate (.......);*

— *casDetectInit (.......);*

— *casDetectProcess (.......);*

— *casDetectDestroy (.......);*

5.1.1 Library Sections

The CAS Detect Library contains the following data ROM section, which must be placed in memory through the linker command file.

- ***CAS_INTERNAL_ROM*** **-** CAS Detect assembly code and data memory

See **Code Example 5-1** for a sample *linker.cmd* file, which may be used to test the CAS Detect library.

Code Example 5-1. *linker.cmd* File

```
# Linker.cmd file for DSP56824EVM External RAM
# using both internal and external data memory (EX = 0)
# and using external program memory (Mode = 3)

MEMORY {

      .pram   (RWX) : ORIGIN = 0x0000, LENGTH = 0xFF80  # ? external program memory
      .avail  (RW)  : ORIGIN = 0x0000, LENGTH = 0x0030  # available
      .cwregs (RW)  : ORIGIN = 0x0030, LENGTH = 0x0010  # C temp registrs in
                                CodeWarrior
      .im1    (RW)  : ORIGIN = 0x0040, LENGTH = 0x07C0  # data 1
```

```
      .rom    (R)   : ORIGIN = 0x0800, LENGTH = 0x0800  # internal data ROM
      .im2    (RW)  : ORIGIN = 0x1000, LENGTH = 0x0600  # data 2
      .hole   (R)   : ORIGIN = 0x1600, LENGTH = 0x0A00  # hole
      .data   (RW)  : ORIGIN = 0x2000, LENGTH = 0xC000  # data segment
      .em     (RW)  : ORIGIN = 0xE000, LENGTH = 0x1000  # data 3
      .stack  (RW)  : ORIGIN = 0xF000, LENGTH = 0x0F80  # stack
      .onchip1(RW)  : ORIGIN = 0xFF80, LENGTH = 0x0040  # on-chip peripheral
                                 registers
      .onchip2(RW)  : ORIGIN = 0xFFC0, LENGTH = 0x0040  # on-chip peripheral
                                 registers

}


FORCE_ACTIVE {FconfigInterruptVector}

SECTIONS {

      #
      # Data (X) Memory Layout
      #
            _EX_BIT     = 0;

            # Internal Memory Partitions (for mem.h partitions)
            _NUM_IM_PARTITIONS = 1;  # .im1 and .im2

            # External Memory Partition (for mem.h partitions)
            _NUM_EM_PARTITIONS = 1;   # .em


      .main_application_code :
      {
      # .text sections

      #  config.c MUST be placed first, otherwise the Interrupt Vector
      #  configInterruptVector will not be located at the correct address, P:0x0000

      config.c (.text)
      * (.text)
      * (rtlib.text)
      * (fp_engine.text)
      * (user.text)

      } > .pram


      .main_application_data :
      {
      #
      # Define variables for C initialization code
      #
      F_Xdata_start_addr_in_ROM = ADDR(.rom) + SIZEOF(.rom) / 2;
      F_StackAddr               = ADDR(.stack);
      F_StackEndAddr            = ADDR(.stack) + SIZEOF(.stack) / 2  - 1;
```

```
        F_Xdata_start_addr_in_RAM = .;

        #
        # Memory layout data for SDK INCLUDE_MEMORY (mem.h) support
        #

        FmemEXbit = .;
        WRITEH(_EX_BIT);
        FmemNumIMpartitions = .;
        WRITEH(_NUM_IM_PARTITIONS);
        FmemNumEMpartitions = .;
        WRITEH(_NUM_EM_PARTITIONS);
        FmemIMpartitionList = .;
        #     WRITEH(ADDR(.im1));
        #     WRITEH(SIZEOF(.im1) / 2);
              WRITEH(ADDR(.im2));
              WRITEH(SIZEOF(.im2) / 2);
        FmemEMpartitionList = .;
              WRITEH(ADDR(.em));
              WRITEH(SIZEOF(.em) /2);

        # .data sections

        * (.data)
        * (fp_state.data)
        * (rtlib.data)

        F_Xdata_ROMtoRAM_length = 0;

        F_bss_start_addr = .;
        _BSS_ADDR = .;

        * (rtlib.bss.lo)
        * (.bss)

        F_bss_length = . - _BSS_ADDR;  # Copy DATA

        } > .data

        .casdetect_internal_data :
        {
        # CAS detect internal data starts here
        #-----------------------------------

        * (CAS_INTERNAL_ROM.data)
        * (CAS_INTERNAL_ROM.bss)

        # CAS detect internal data ends here
        #---------------------------------

        FArchIO   = ADDR(.onchip2);
}
```

Chapter 6
CAS Applications

6.1 Test and Demo Applications

To verify the CAS Detect algorithm, test and demo applications have been developed. Refer to the **Targeting Motorola DSP568xx Platform** Manual for the DSP you are using to see if the test and demo applications are available for your target.

Chapter 7
License

7.1 Limited Use License Agreement

LIMITED USE LICENSE AGREEMENT

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THIS SOFTWARE. BY USING OR COPYING THE SOFTWARE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.

The software in either source code form ("Source") or object code form ("Object") (cumulatively hereinafter "Software") is provided under a license agreement ("Agreement") as described herein. Any use of the Software including copying, modifying, or installing the Software so that it is usable by or accessible by a central processing unit constitutes acceptance of the terms of the Agreement by the person or persons making such use or, if employed, the employer thereof ("Licensee") and if employed, the person(s) making such use hereby warrants that they have the authority of their employer to enter this license agreement,. If Licensee does not agree with and accept the terms of this Agreement, Licensee must return or destroy any media containing the Software or materials related thereto, and destroy all copies of the Software.

The Software is licensed to Licensee by Motorola Incorporated ("Motorola") for use under the terms of this Agreement. Motorola retains ownership of the Software. Motorola grants only the rights specifically granted in this Agreement and grants no other rights. Title to the Software, all copies thereof and all rights therein, including all rights in any intellectual property including patents, copyrights, and trade secrets applicable thereto, shall remain vested in Motorola.

For the Source, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to use, copy, and make derivatives of the Source solely in a development system environment in order to produce object code solely for operating on a Motorola semiconductor device having a central processing unit ("Derivative Object").

For the Object and Derivative Object, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to copy, use, and distribute the Object and the Derivative Object solely for operating on a Motorola semiconductor device having a central processing unit.

Licensee agrees to: (a) not use, modify, or copy the Software except as expressly provided herein, (b) not distribute, disclose, transfer, sell, assign, rent, lease, or otherwise make available the Software, any derivatives thereof, or this license to a third party except as expressly provided herein, (c) not remove obliterate, or otherwise defeat any copyright, trademark, patent or proprietary notices, related to the Software (d) not in any form export, re-export, resell, ship or divert or cause to be exported, re-exported, resold, shipped, or diverted, directly or indirectly, the Software or a direct product thereof to any country which the United States government or any agency thereof at the time of export or re-export requires an export license or other government approval without first obtaining such license or approval.

THE SOFTWARE IS PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MOTOROLA BE LIABLE FOR ANY LIABILITY OR DAMAGES OF ANY KIND INCLUDING, WITHOUT LIMITATION, DIRECT OR INDIRECT OR INCIDENTAL OR CONSEQUENTIAL OR PUNITIVE DAMAGES OR LOST PROFITS OR LOSS OF USE ARISING FROM USE OF THE SOFTWARE OR THE PRODUCT REGARDLESS OF THE FORM OF ACTION OR THEORY OF LIABILITY (INCLUDING WITHOUT LIMITATION, ACTION IN CONTRACT, NEGLIGENCE, OR PRODUCT LIABILITY) EVEN IF MOTOROLA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THIS DISCLAIMER OF WARRANTY EXTENDS TO LICENSEE OR USERS OF PRODUCTS AND IS IN LIEU OF ALL WARRANTIES WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR PARTICULAR PURPOSE.

Motorola does not represent or warrant that the Software is free of infringement of any third party patents, copyrights, trade secrets, or other intellectual property rights or that Motorola has the right to grant the licenses contained herein. Motorola does not represent or warrant that the Software is free of defect, or that it meets any particular requirements or need of the Licensee, or that it conforms to any documentation, or that it meets any standards.

Motorola shall not be responsible to maintain the Software, provide upgrades to the Software, or provide any field service of the Software. Motorola reserves the right to make changes to the Software without further notice to Licensee.

The Software is not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Software could create a situation where personal injury or death may occur. Should Licensee purchase or use the Software for any such unintended or unauthorized application, Licensee shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the Software.

The term of this Agreement is for as long as Licensee uses the Software for its intended purpose and is not in default of any provisions of this Agreement. Motorola may terminate this Agreement if Licensee is in default of any of the terms and conditions of this Agreement.

This Agreement shall be governed by and construed in accordance with the laws of the State of Arizona and can only be modified in a writing signed by both parties. Licensee agrees to jurisdiction and venue in the State of Arizona.

By using, modifying, installing, compiling, or copying the Software, Licensee acknowledges that this Agreement has been read and understood and agrees to be bound by its terms and conditions. Licensee agrees that this Agreement is the complete and exclusive statement of the agreement between Licensee and Motorola and supersedes any earlier proposal or prior arrangement, whether oral or written, and any other communications relative to the subject matter of this Agreement.

Index

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

Technical Information Center: 1–800–521–6274

HOME PAGE: http://www.motorola.com/semiconductors/

MOTOROLA