MOTOROLA

Embedded SDK (Software Development Kit)

Type 1 Telephony Features Library

SDK135/D
Rev. 1, 07/23/2002

MOTOROLA intelligence everywhere™ | digital dna™

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

Contents

Chapter 5
Linking Applications with the Type 1 Telephony Features Library

Chapter 6
Type 1 Telephony Features Library Applications

Chapter 7
License

List of Tables

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

List of Figures

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

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

List of Examples

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

About This Document

This manual describes the Type 1 Telephony Features Library for use with Motorola's Embedded Software Development Kit (SDK).

Audience

This document targets software developers implementing communication features for analog telephone lines.

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, Type 1 Telephony Features Library Interfaces**—describes all of the Type 1 Telephony Features Library functions
- **Chapter 4, Building the Type 1 Telephony Features Library**—tells how to build a test application project with the pre-built Type 1 Telephony Features Library
- **Chapter 5, Linking Applications with the Type 1 Telephony Features Library**—describes linking projects with the Type 1 Telephony Features Library
- **Chapter 6, Type 1 Telephony Features Library Applications**—describes the use of the Type 1 Telephony Features 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:

- *Motorola DSP56800E Reference Manual*, DSP56800ERM/D
- *Motorola DSP568xx User's Manual*, for the DSP device you're implementing
- *Inside CodeWarrior: Core Tools*, Metrowerks Corp.

Conventions

This document uses the following notational conventions:

Typeface, Symbol or Term	Meaning	Examples
`Courier Monospaced Type`	Code examples	`//Process command for line flash`
Italic	Directory names Project names Calls Functions Statements Procedures Routines Arguments File names Applications Variables Directives Code snippets in text	...and contains these core directories: *applications* contains applications software... ...CodeWarrior project, *3des.mcp* is... ...the *pConfig* argument.... ...defined in the C header file, *aec.h*....
Bold	Reference sources Paths Emphasis	...refer to the **Targeting DSP5685x Platform** manual.... ...see: **C:\Program Files\Motorola\Embedded SDK\help\tutorials**
ALL CAPITAL LETTERS	# defines/ Defined constants	# 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"...
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}

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.

ADC Analog-to-Digital Converter

AGC	Automatic Gain Control
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BPF	Bandpass Filter
CID	Caller ID (Calling Party Name and/or Number Identification)
CPE	Customer Premises (telephony) Equipment
CQ	Call Qualifier
DAA	Data Access Arrangement
DSP	Digital Signal Processor or Digital Signal Processing
DTMF	Dual Tone Multiple Frequency
FSK	Frequency Shift Keying modulation
IDE	Integrated Development Environment
LPF	Low Pass Filter
MDMF	Multiple Data Message Format of GR-30-CORE
MIPS	Million Instructions Per Second
OnCE™	On-Chip Emulation
OSI	Open Switching Interval
PC	Personal Computer
PCM	Pulse Code Modulation
PSTN	Public Switched Telephone Network
SDK	Software Development Kit
SDMF	Single Data Message Format of GR-30-CORE
SRC	Source
VMWI	Visual Message Waiting Indicator

References

The following sources were referenced to produce this book:

1. *Motorola DSP56800E Reference Manual*, DSP56800ERM/D
2. *Motorola DSP568xx User's Manual*, for the DSP device you're implementing
3. *Targeting Motorola DSP568xx Platform*, for the DSP device you're implementing
4. *Motorola Embedded SDK Programmer's Guide,* SDK101/D
5. SR-3004, *Testing Guidelines for Analog Type 1, 2, and 3 CPE as Described in SR-INS-002726 (a module of ADSI, FR-12),* Telcordia Technologies, January 1995.
6. GR-30-CORE, *LSSGR: Voiceband Data Transmission Interface Section 6.6 (a module of LSSGR, FR-64),* Telcordia Technologies, December 1998.
7. GR-31-CORE, *LSSGR CLASS^{SM} Feature: Calling Number Delivery (FSD 01-02-1051) (a module of LSSGR, FR-64),* Telcordia Technologies, June 2000.
8. GR-1188-CORE, *LSSGR CLASS^{SM} Feature: Calling Name Delivery Generic Requirements (FSD 01-02-1070) (a module of LSSGR, FR-64),* Telcordia Technologies, December 2000.

9. GR-1401-CORE, *LSSGR CLASS^{SM} Feature: Visual Message Waiting Indicator Generic Requirements (FSD 01-02-2000) (a module of LSSGR, FR-64),* Telcordia Technologies, June 2000.
10. *ITU-T Recommendation V.23, (11/88)* - 600/1200-baud modem standardized for use in the general switched telephone network.

Chapter 1
Introduction

Welcome to Motorola's family of Digital Signal Processors, or DSPs. This document describes the Type 1 Telephony Features Library, which is a part of Motorola's comprehensive Software Development Kit, SDK, for its DSPs. In this document, you will find all the information required to use and maintain the Type 1 Telephony Features Library interface and algorithms.

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

Motorola's Type 1 Telephony Features Library is a licensed software library for use on Motorola DSP56800E series processors. Please refer to the Software License Agreement in **Chapter 7** for license terms and conditions.

1.1 Quick Start

Motorola's 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 Motorola DSP5685x Platform** documentation.

For example, the **Targeting Motorola DSP5685x Platform** manual provides more specific information and examples about this hardware architecture. If you are developing an application for the DSP56858EVM board, or any other DSP56858 development system, refer to the **Targeting Motorola DSP5685x Platform** manual for **Quick Start** or other DSP56858-specific information.

Note: "DSP568xx" refers to the specific device for which you're developing, as shown in the preceding example.

1.2 Telephony Features Libraries

The Type 1 Telephony Features Library is one of a set of Motorola Embedded SDK modules and applications consisting of the following:

- Type 1 Telephony Features Library
- Type 1 and 2 Telephony Features Library
- Type 1 and 2 Telephony Parser Library
- Full Duplex Speakerphone Library

- Generic Echo Canceller Library
- Feature Phone Application Software

These modules are designed to interoperate to provide all of the software necessary to implement a feature phone with full duplex speakerphone and Type 1 and 2 Caller ID functionality. Using some or all of these modules, several other types of telephony applications are also possible. Each module may also be used independently.

1.3 Overview of the Type 1 Telephony Features Library

Customer Premises Equipment (CPE) using the Type 1 Telephony Features Library has the features to support existing on-hook Caller ID services, such as Calling Number Delivery and Calling Name Delivery; other existing on-hook services, such as Visual Message Waiting Indicator (VMWI) or Call Qualifier (CQ); as well as future services that would use the on-hook GR-30-CORE Voice band Data Transmission Interface from the Public Switched Telephone Network (PSTN). Type 1 CPE supports on-hook signaling with or without power ringing and can decode data frames packaged in the GR-30-CORE Single Data Message Format (SDMF) or Multiple Data Message Format (MDMF), depending on the needs of the specific telephony service supported, as described in Telcordia SR-3004 and other referenced service-specific documents.

While this module is primarily intended to be used for Caller ID reception, other convenient features have been added to allow for simple integration into a Caller ID telephone. While the module is in an on-hook state, ring generator samples may be created during power ringing. These samples may be output to an audio channel (for example, to be played to a speaker). While the module is in an off-hook state, DTMF samples may be generated for dialing digits. Timing for a line flash is also provided.

The Caller ID is received between the first and second ring. The content displayed should include date, time, and the directory telephone number and/or the name of the caller. Other existing service-specific content may also be displayed, such as Call Qualifier displays. The service-specific content of future services may be displayed when these become deployed in the PSTN.

VMWI may be transmitted with or without power ringing, but must be received without power ringing. The CPE is instructed by the data received in GR-30-CORE format to turn a light or other visual indicator on or off, indicating that new messages have been received at a voice mail system or other messaging service.

The performance of CPE that incorporates the Type 1 Telephony Features Library (also called the Type 1 Library module) complies with the transmission layer requirements of Telcordia SR-3004, including those for:

- Frequency Shift Keying (FSK) modulation sensitivity
- Twist, or level differences between FSK signal tones
- FSK frequencies and variation from their nominal values
  - In addition to complying with the FSK modulation frequencies of SR-3004, the Type 1 Telephony Features Library can also detect FSK transmitted using the slightly different frequencies specified by the ITU-T V.23 modem recommendation. However, the Type 1 Telephony Features Library cannot be used directly in equipment for international telephone networks with V.23, since the timing protocols are likely different.

- Robust detection of incoming FSK signals in the presence of significant noise impairments on the telephone line
- All signal and inter-signal timings required in SR-3004, with or without power ringing
- The ability to receive FSK after detecting power ringing signals which have any of the various cadences required by SR-3004, in addition to the Normal ringing pattern

1.3.1 Background

The on-hook Type 1 Library is basically a Frequency Shift Keying (FSK) receiver with the following additional abilities:

- to detect power ringing
- to recognize and remain immune to various other voltage level shifts identified in SR-3004 that may precede FSK data with or without power ringing
- to recognize the start of FSK data reception
  - — following various other voltage level shifts identified in SR-3004 and power ringing
  - — following only various other voltage level shifts identified in SR-3004 (power ringing does not precede FSK data)
  - — if not preceded by any of the various other voltage level shifts identified in SR-3004

Thus, required inputs to this Library are:

- Indication of power ringing
  - — Usually present before reception of Caller ID in FSK data
  - — Usually absent before reception of VMWI in FSK data
  - — Used by the Library to disable the FSK receiver during power ringing
- Indication of Open Switching Intervals (OSIs) or other voltage level shifts that may precede the FSK data
- The FSK data in the GR-30-CORE format appropriate for Calling Number Delivery, Calling Name Delivery, VMWI, or other existing or future telephony service

A line interface is needed to detect the presence of a power ringing signal. The line interface can be a Data Access Arrangement (DAA) or any line interface that can pass FSK signals from the telephone network and has a ring detect circuit that can indicate to the DSP that a power ringing signal has been received. The line interface circuitry may vary, depending on the individual case.

For the Type 1 Telephony Features Library to comply with Telcordia SR-3004 requirements, the user must supply a suitable line interface that meets the following requirements:

- Frequency Response--The received signal must be within voice (also called line) bandwidth, which ranges from 300 to 3400Hz
- Attenuation Distortion (Front End Range)--The Type 1 Telephony Features Library is designed to interoperate with an Analog-to-Digital Converter (ADC) with an analog signal input range of 2.73dBm±3dB. In other words, the ADC in the line front end should have a 3V peak-to-peak analog input voltage range.

In one possible implementation using the Motorola DSP56858EVM board, the ring signal indication from the line interface must pass to the DSP56858EVM on one of the DSP GPIOB pins. The signal from the line interface can be passed to the DSP56858EVM using a stereo jack. Details of required jumper settings are described in the **DSP56858 Evaluation Module User's Manual**.

The rest of this section describes the operation at a system level of a generic FSK receiver, the major component of this Library. **Figure 1-1** provides a high-level example of a generic FSK receiver used in the Type 1 Telephony Features Library. The FSK receiver is shown inside the dashed lines in **Figure 1-1**. The components shown outside the dashed lines are not supplied with the Type 1 Telephony Features Library.

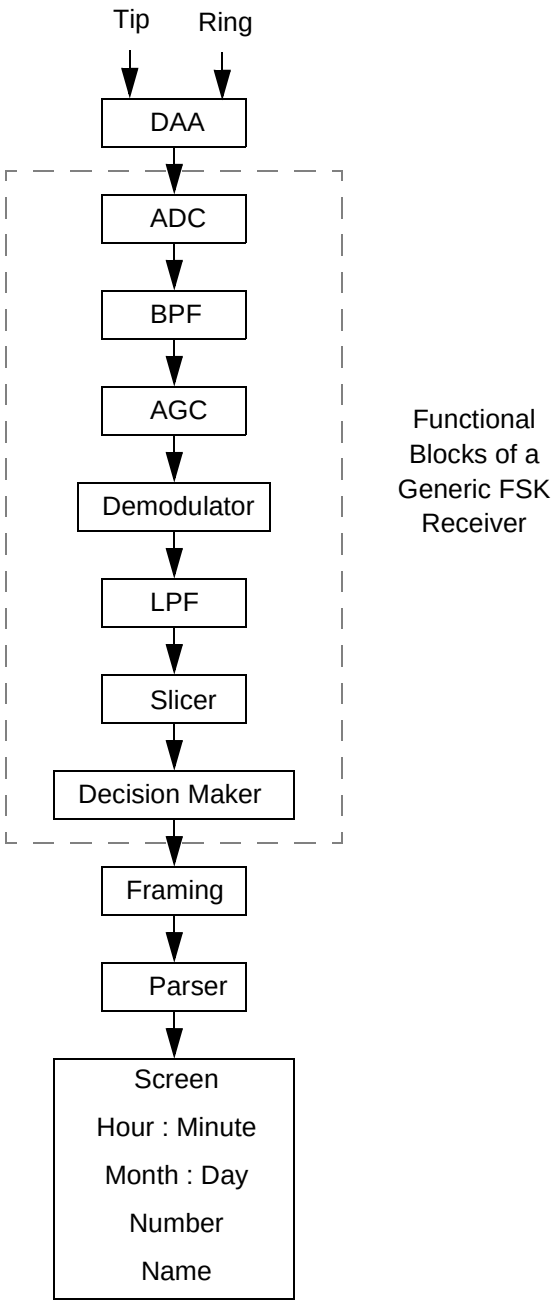

Figure 1-1. The Functional Blocks of a Generic Type 1 Telephony Solution

The FSK receiver performs noise filtering, signal detection, demodulation, and framing, then passes Caller ID (or VMWI, etc.) to the application layer to display on the CPE. The FSK receiver is always enabled, so it can receive VMWI without power ringing. If receiving Caller ID, power ringing will be signaled to the DSP to disable the FSK receiver during the ringing. The digital samples must be collected with an ADC at

the sampling rate of 8KHz, with a minimum of 14-bit linear precision (8-bit μ-law). The ADC is not part of this solution and must be supplied separately by the application developer; however, the DSP56858EVM board contains an ADC. The FSK receiver starts processing the digital samples received as soon as the ADC sends them. The FSK receiver accepts an FSK signal transmitting at a baud rate of 1200bps according to Telcordia Technologies SR-3004.

The digital samples from the ADC will first pass through a bandpass filter (BPF) as shown in **Figure 1-1**, where only the signals within line bandwidth (300 – 3400Hz) can be passed. The BPF output will be processed in an Automatic Gain Control (AGC) function. The AGC estimates the signal level to compensate for the amplitude distortions introduced by the telephone network. In addition, the estimated power level is used for signal detection. The AGC output sample is then sent to a software demodulator.

The demodulator will introduce a double frequency component. The following low pass filter (LPF) is used to remove the double frequency component and pass signals within the sample frequency range. The outputs of the demodulator are sent to the Slicer, where timing recovery is performed and a decision for a symbol is made. The symbols are then framed to form an ASCII word. The ASCII words are passed to upper-layer functionality external to the Type 1 Telephony Features Library for parsing and displaying Caller ID information, or for activating/deactivating the VMWI.

The user must supply a parser algorithm external to this Type 1 Telephony Features Library to decode the message layer ASCII words according to the appropriate service-dependent GR-30-CORE format - either SDMF or MDMF - in a manner compliant with the message layer requirements of SR-3004. See **References** for the format applicable to each service that uses the GR-30-CORE protocol supported by the Type 1 Telephony Features Library. The Parser Library supplied with the Embedded SDK is the preferred method for interoperating with the Type 1 Telephony Features Library to achieve both transmission layer and message layer SR-3004 compliance.

1.3.2 Features and Performance

Table 1-1 details the memory and MIPS requirements for the Type 1 Telephony Features Library.

Table 1-1. Type 1 Telephony Features Library Memory and MIPS Requirements

	Program Memory (ROM) (16 bit words)	Data ROM (16 bit words)	Data RAM 1st Instance (16 bit words)	Data RAM Additional Instance (16 bit words)	MIPS (When all program and data is internal memory)
Type 1 Telephony Features Library	1.7 K	20	257	192	9.6

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

Chapter 2
Directory Structure

Note: "DSP568xx" refers to the specific device for which you're developing, as shown in **Chapter 1, "Introduction."**

2.1 Required Core Directories

Figure 2-1 details required platform directories:

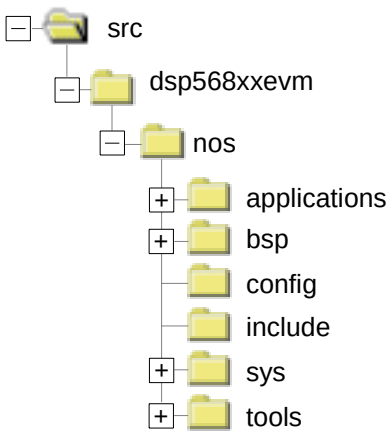

Figure 2-1. Core Directories

As shown in **Figure 2-1**, DSP56858EVM has no operating system support (nos), and includes the following 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 and software configurations for this platform
- ***include*** contains SDK header files which define the Application Programming Interface
- ***sys*** contains required system components
- ***tools*** contains utilities used by system components

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

2.2 Optional (Domain-Specific) Directories

Figure 2-2 shows the domain-specific directory, ***cidtype1***.

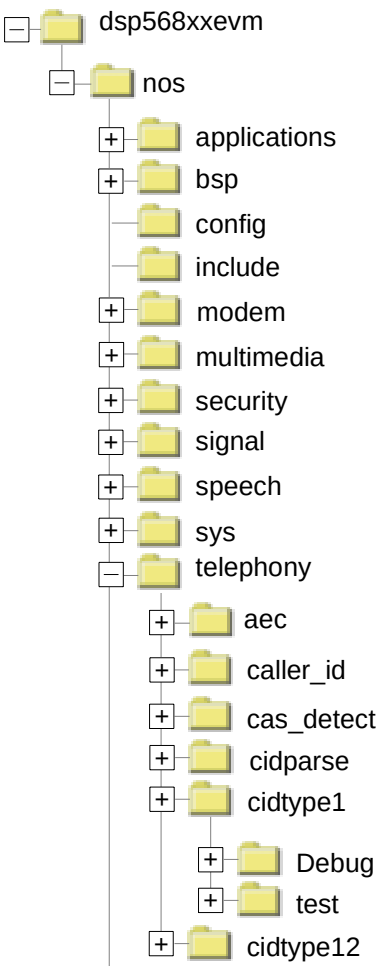

Figure 2-2. *cidtype1* Directory

As shown in Figure 2-3, the ***telephony*** directory includes specific vocoder algorithms, such as G.728, or G.726, as well as ***cidtype1*** - the directory that includes the Type 1 Telephony Features Library for receiving on-hook Caller ID and other services. The ***cidtype1*** directory includes Type 1 interface algorithms.

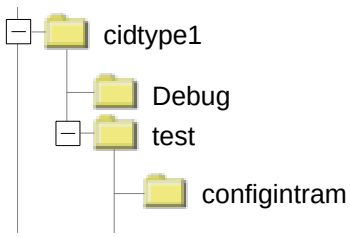

Figure 2-3. *cidtype1* Directory Structure

The ***cidtype1*** directory shown in **Figure 2-3** includes:

- ***Debug*** contains the library file (*cid1.lib*) for the pre-built Type 1 Telephony Features Library
- ***test*** contains the test project *cid1test.mcp* for the pre-built Type 1 Telephony Features Library
  - — contains C source for the test application
  - — ***configintram*** contains the *linker.cmd* file for the test app and also contains *appconfig.c* and *appconfig.h* to override the SDK's *config.h* for this particular project

Chapter 3
Type 1 Telephony Features Library Interfaces

The Type 1 Telephony Features Library is defined as:

Type1CID.lib

The service, interface, and function calls included are described below.

3.1 Type 1 Telephony Features Library Interface Services

The Type 1 Telephony Features Library interface provides Caller ID, Visual Message Waiting Indicator (VMWI), and other related services which occur in the on-hook state. The Type 1 Telephony Features Library supports FSK signals with a baud rate of 1200bps. The library demodulates the FSK into ASCII words and passes the data payload to the application program for processing and presentation. The FSK message can be in single or multiple data message format per Telcordia SR-3004. In the on-hook state, the actual FSK message is received between the first and second ring. If an FSK message is not detected during this period, no information will be available until the next call. Other services, such as VMWI, may not involve power ringing at all.

3.2 Interface

The Type 1 Telephony Features Library can be called by an application program in C. **Code Example 3-1** contains structure definitions of *teldef_sControl* and *teldef_sSamples* which are used by the Type 1 Telephony Features Library to receive information from the application and then to output results to the application. The following is a listing of the variables in *teldefs.h* that are related to the Type 1 Telephony Features Library only. The other variables are described in their respective library documents.

Code Example 3-1. *teldefs.h* - Reference Definition for Type1CID

```
#ifndef     __TELDEFS_H
#define     __TELDEFS_H
```

```
typedef struct teldefs_tsControl
{

// phone state.
      int hookSwitch;           //on hook or off hook.
      int handsFreeLayer1;      //speaker phone on or off.

//------------------- Caller ID.-------------------------
      int intpdata;             //pointer for modular buffer.
      int cidRingPolarity;      //ring signal high or low.
      int cidByteReady;         //flag for a byte when ready.
      int cidByte;              //contents of the byte.
      int messageDone;          //flag for a message when ready.
      int messageLength;        //length of the message.
      int FrameErrors;          //indicates if frame error occurred during fsk.
      int flashCommand;         //indicates line should be flashed.
      int flashPolarity;        //indicates the direction of flashing.
      int cwdCommand;           //begin a Call Waiting deluxe pulse/dial.
      int disableRinger;        //disable ring generator.

// DTMF dialer

      int dtmfRequest;          //to dial a dtmf. ON = 1, OFF = 0.
      int dtmfDigit;            //the dtmf digit to be dialed.
      int dtmfComplete;         //flag for a dtmf when dialing complete.

//---------------- Generic Echo Canceller -----------------
            //Generic Echo Canceller related variables
            //are declared here.

//--------------- Full Duplex Speakerphone ----------------
            //Full Duplex Speakerphone related
            //variables are declared here.

} teldefs_sControl;


typedef struct teldefs_tsSamples
{
      int line[5];              //line input or audio output.
      int audio[5];             //audio input or line output.
      int gec[5];               //generic echo canceller's output.
      int aec[5];               //fdspk's output.
      int voipinput[5];         //third port's input.
      int voipoutput[5];        //third port's output.
      int leccid[5];            //used by type 2 to receive line echo cancelled
                                //samples.

} teldefs_sSamples;

#endif
```

The structure defined in *cid1.h*, *cid_sData*, is allocated by the *Type1CIDcreate()* function, but is accessed only by the Type 1 Caller ID functions. It is essential that memory be allocated for this structure for the Type 1 Caller ID functions to use and that no other function, or the application itself, access the memory space reserved for this structure. The Type 1 Caller ID function stores values that it needs to reuse, so tampering with the contents of the structure will cause the function to give erroneous results.

As shown in **Code Example 3-2** and **Code Example 6-1**, statically allocate both a *teldef_sControl* and *teldef_sSample* structure for each instance of a Caller ID channel, while using a call to *Type1CIDcreate()* to dynamically allocate the data structure.

The use of function prototypes and structure instances for the Type 1 Telephony Features Library are described in header file *cid1.h*, shown in **Code Example 3-2**.

Code Example 3-2. *cid1.h* - Reference Definition for Type1CID

```
#ifndef      __CID1_H
#define      __CID1_H


/**************************
 Foundational Include Files
**************************/
#include "cid_type1.h"

#ifndef __TELDEFS_H
#include "teldefs.h"
#endif


/***********************************************
Structures that must be defined before
accessing Type 1 Caller_ID pointer. Examples are shown below:
***********************************************/

/*
teldef_sControl     Line1Control
cid_sData*          pcid1Data
teldef_sSample      Line1Samples
*/


/***********************************************
Function Prototypes for Type 1 Interface Library
***********************************************/
cid_sData* Type1CIDcreate(teldef_tsControl *pControl);
void Type1CIDdestroy(cid_sData *pData, teldef_tsControl *pControl);
void Type1CIDinit(cid_sData *pData, teldef_tsControl *pControl);
void Type1CID(cid_sData *pData, teldef_tsControl *pControl, teldef_tsSample
*pSample);
```

```
/*  Local variables for Type 1 Telephony Features, size of 127. Structure _sData
details are defined in header file cid_type1.h, which also contains various space
allocators and housekeeping variables. However, since these variables are not
manipulated by the application, the contents of cid_type1.h are not documented in this
SDK. */


#endif
```

3.2.1 Variable Definition

This section includes a more detailed explanation of the variables in structure *teldefs_tsControl* that an application using the Type 1 Telephony Features Library must set. Also included in *teldefs.h* are the data structures for the Type 1 and 2 Telephony Parser Library, needed to help display the received FSK message. For additional information, see the document **Type 1 and 2 Telephony Parser Library**.

hookSwitch	Set by the application to indicate to the Type 1 and Type 1 and 2 Libraries whether the phone is on-hook or off-hook On-hook = 0 Off-hook = 1
handsFreeLayer1	Set by the application to indicate to the Type 1 and Type 1 and 2 Libraries whether the handset or handsfree is enabled
intpdata	Set and allocated by the *Type1create()* to indicate the address of the modulo buffer for FSK
cidRingPolarity	Set by the application to indicate to the Type 1 and Type 1 and 2 Libraries when a ringing signal is detected on the ringing detect circuit. The ringing detect circuit should be capable of detecting either the envelope of the ringing signal itself or of a square wave whose fundamental frequency is the ring frequency. For purposes of ringing detection, the envelope of a signal is either the amplitude of a signal or some more slowly varying amplitude detected by less sensitive circuitry.
cidByteReady	Set by the Type 1 and Type 1 and 2 Libraries to indicate to the application (or Type 1 and 2 Telephony Parser Library) that one byte is ready from the FSK receiver
cidByte	Set by the Type 1 and Type 1 and 2 Libraries; it holds the actual byte
messageDone	Set by the Type 1 and Type 1 and 2 Libraries to indicate to the application (or Type 1 and 2 Telephony Parser Library) that an FSK message is completed
messageLength	Set by the Type 1 and Type 1 and 2 Libraries to indicate the length of the FSK message to the application
FrameErrors	Set by the Type 1 and Type 1 and 2 Telephony Features Libraries to indicate to the application whether a framing error occurred during reception of the last FSK message. This must be cleared by the application upon reception.
flashCommand	A message sent from the module to the application indicating that the line is to be flashed according to flash polarity. This must be cleared by the application upon reception. Line is to be flashed = 1 Do nothing = 0

flashPolarity	Set by the module to indicate the polarity of the flash. When *flashCommand* is set, go on-hook when the *flashPolarity* equals zero and go off-hook when *flashPolarity* equals one. Note: Do not change the *hookSwitch* variable when executing a flash.
cwdCommand	Set by the application to instruct the module to begin a Call Waiting Deluxe pulse/dial. In Type 1, there are no Call Waiting Deluxe features supported, so this variable is used only for a standard line flash. Do nothing = 0 Answer or standard flash = 10
disableRinger	Set by the application to disable the ring generator when on-hook
dtmfRequest	Set by the application when it requires the module to dial a DTMF digit On = 1 Off = 0
dtmfDigit	The particular digit to be dialed. The possible entries range from integer values of 0 to 11, which correspond to the nine digits, *, and #, in that order.
dtmfComplete	Flag set by the module for the application's information on completion of a single DTMF request

3.3 Specifications

The following sections describe the Type 1 Telephony Features 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 functions. 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 *Type1CIDcreate*

Call(s):

```
cid_sData* Type1CIDcreate(teldef_tsControl *pControl);
```

Required Headers: *teldefs.h, cid1.h*

Arguments:

Table 3-1. *Type1CIDcreate* Arguments

pControl	*in*	Points to the structure where input and output control information is stored

Description: The *Type1CIDcreate* function allocates the data memory structure and modulo buffer for the *Type1CID* function, then calls the *Type1CIDinit* function. This function is to be called once before using the *Type1CID* function. The prototype of this function is defined in *cid1.h*.

Returns: The function*Type1CIDcreate* returns a pointer to the data structure.

Special Issues: None

Code Example: None

3.3.2 *Type1CIDdestroy*

Call(s):

```
void Type1CIDdestroy (cid_sData *pData, teldef_tsControl *pControl);
```

Required Headers: *teldefs.h, cid1.h*

Arguments:

Table 3-2. *Type1CIDdestroy* Arguments

pData	*in*	Points to the structure where FSK static data is stored
pControl	*in*	Points to the structure where input and output control information is stored

Description: This function deallocates any memory that was used by the current *Type1CID* instance. The prototype of the *Type1CIDdestroy* function is defined in *cid1.h.*

Returns: None

Special Issues: None

Code Example: None

3.3.3 *Type1CIDinit*

Call(s):

```
void Type1CIDinit(cid_sData *pData, teldef_tsControl *pControl);
```

Required Headers: *teldefs.h, cid1.h*

Arguments:

Table 3-3. *Type1CIDinit* Arguments

pData	*inout*	Points to the structure where FSK static data is stored
pControl	*inout*	Points to the structure where input and output control information is stored

Description: The *Type1CIDinit* function initializes all of the constants, variables and counters contained in the *cid_sData* structure for the *Type1CID* function. This function is to be called once before using the function *Type1CID*. This action is performed by calling *Type1CIDcreate* function. The function *Type1CIDcreate* should also be called when going on-hook or off-hook after changing the *hookSwitch* variable. The prototype of this function is defined in *cid1.h*.

Returns: None

Special Issues: None

Code Example: None

3.3.4 *Type1CID*

Call(s):

```
void Type1CID(cid_sData *pData, teldef_tsControl *pControl, teldef_tsSample
        *pSample);
```

Required Headers: *teldefs.h, cid1.h*

Arguments:

Table 3-4. *Type1CID* Arguments

pData	*inout*	Points to the structure where FSK static data is stored
pControl	*inout*	Points to the structure where input and output control information is stored
pSample	*inout*	Points to the structure where the voice samples are stored

Description: The *Type1CID* function performs on-hook FSK reception. According to the control information from its arguments, it enables the FSK receiver and passes the FSK to the upper layer process to display the data. This function should be called 1600 times per second.

hookSwitch = 0 (on-hook):

- *pSample.line[]* samples are inputs to this function and are decoded into data bytes when caller ID is present
- *pSample.line[]* samples are outputs from this function and are ringer generator samples when ringing is present; otherwise, these samples are filled with zeros
- *pSample.audio[]* samples are ignored and filled with zeros

hookSwitch = 1 (off-hook):

- If *handsFreeLayer1* is set, *pSample.gec[]* are copied into *pSample.line[]* samples; otherwise, *pSample.line[]* samples are unchanged
- If a DTMF digit has been requested by setting *dtmfRequest*, *pSample.audio[]* is filled with DTMF samples; otherwise, *pSample.audio[]* samples are unchanged

Returns: None

Special Issues: None

Code Example: None

Chapter 4
Building the Type 1 Telephony Features Library

4.1 Building the Type 1 Telephony Features Library

The Type 1 Telephony Features Library combines all components described in the previous section into one library: *cid1.lib*. The library is prebuilt for the user, so a project for building the library is not provided. The library is located in the ...*telephony**cidtype1**lib* directory of the SDK directory structure.

Figure 4-1 shows how the *cid1.lib* is linked to the Type 1 test project, *cid1test.mcp*. The header file, *teldefs.h*, also must be linked when using the Type 1 library, *cid1.lib*.

.

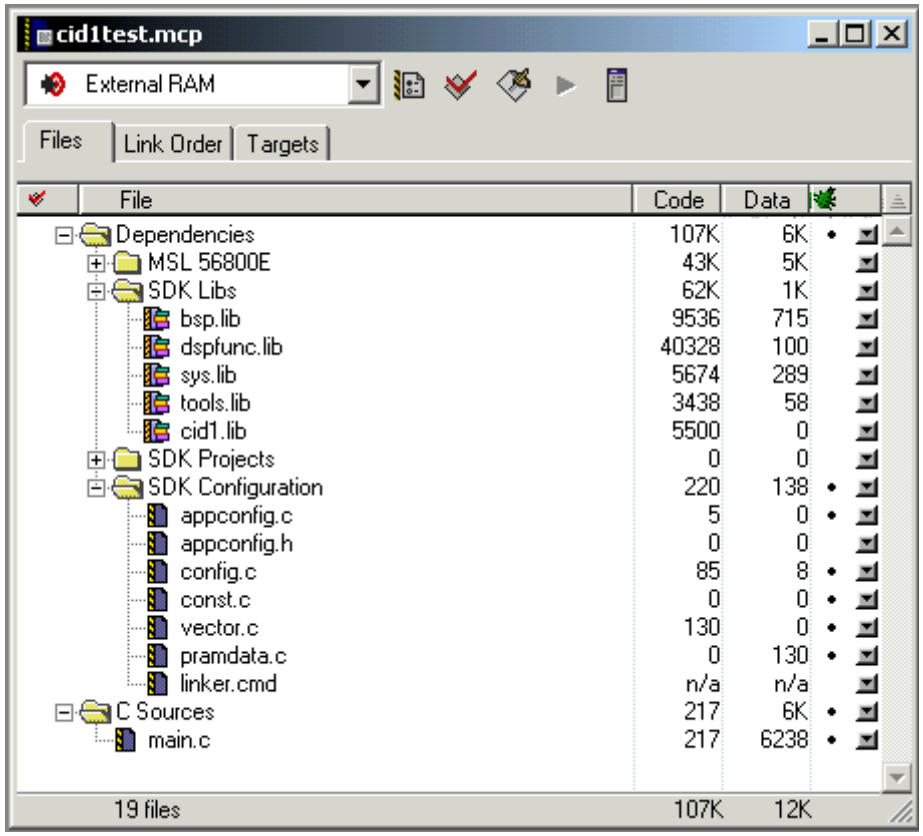

Figure 4-1. Example of a *cid1* Library Link to a Project

Chapter 5
Linking Applications with the Type 1 Telephony Features Library

5.1 Type 1 Telephony Features Library

The Type 1 Telephony Features Library consists of an initialization and processing function. The Type 1 Library can be initialized and created by the *Type1CIDinit* and *Type1CIDcreate* functions. Their functionality and arguments are described in **Section 3.3.3** and **Section 3.3.1**. The library contains APIs, which provide the interface between the user application and the Type 1 Library. To use the Type 1 Library, APIs must be called in the following order:

- Type1CIDcreate(....); //The user must call this function once for every instance.
- Type1CIDdestroy(....); //The user must call this function to deallocate memory.
- Type1CIDinit(….); //The user must call this function for hook switch change.
- Type1CID(….); //The user must call this function to display Caller ID in on-hook state.

5.1.1 Library Sections

An example of the memory section for the Type 1 Telephony Features Library is shown in this section. The data memory requirement for the Type 1 Telephony Features Library is 127 words. The size of the program space requires 1.2Kwords. All program and data memory may reside in internal or external memory. A dynamic memory section of at least 127 words plus 16 words aligned on a 16-word boundary is required. All programs for this library are in the section *cid1.text*. An example *linker.cmd* file is shown in **Code Example 5-1**.

The program and data memory for the Type 1 Telephony Features Library are in the FSK section. This text and data section contains all the code and data necessary to execute the *Type1CID* function.

Code Example 5-1. Example of a *linker.cmd* File for Type 1 Telephony Features Library

```
#*****************************************************************************
#
# Linker.cmd file for DSP56858 External RAM
# using only external program and data memory.
#
```

```
#*****************************************************************************

MEMORY {

            .pInterruptVector   (RWX) : ORIGIN = 0x000000, LENGTH = 0x00008C
            .pIntRAM            (RWX) : ORIGIN = 0x00008C, LENGTH = 0x009F74
            .pExtRAM            (RWX) : ORIGIN = 0x00A000, LENGTH = 0x1E6000
            .pIntROM            (RX)  : ORIGIN = 0x1F0000, LENGTH = 0x000400
            .xIntRAM            (RW)  : ORIGIN = 0x000000, LENGTH = 0x005000
            .xIntRAM_DynamicMem (RW)  : ORIGIN = 0x005000, LENGTH = 0x001000
            .xStack             (RW)  : ORIGIN = 0x006000, LENGTH = 0x000800
            .xExtRAM_DynamicMem (RW)  : ORIGIN = 0x006800, LENGTH = 0x001000
            .xExtRAM            (RW)  : ORIGIN = 0x000000, LENGTH = 0x005000
            .xPeripherals       (RW)  : ORIGIN = 0x1FFC00, LENGTH = 0x000400
            .xExtRAM2           (RW)  : ORIGIN = 0x200000, LENGTH = 0xDFFF00
            .xCoreRegisters     (RW)  : ORIGIN = 0xFFFF00, LENGTH = 0x000100
}

#*****************************************************************************

FORCE_ACTIVE {FconfigInterruptVector}

#*****************************************************************************

SECTIONS {

#*****************************************************************************

    .ApplicationInterruptVector :
            {
                  vector.c (.text)

            } > .pInterruptVector

#*****************************************************************************

            .ApplicationCode :
            {
      # Place all code into Program RAM

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

                  # Place all data into Program RAM

                  F_Pdata_start_addr_in_ROM = 0;
                  F_Pdata_start_addr_in_RAM = .;
                     pramdata.c (.data)
                  F_Pdata_ROMtoRAM_length = 0;

               F_Pbss_start_addr = .;

                    _P_BSS_ADDR = .;
```

```
                pramdata.c (.bss)

                F_Pbss_length = . - _P_BSS_ADDR;

            } > .pExtRAM


#*************************************************************************

            .CID1LibrayCode :
            {
      # Place cid1 code into Program Internal RAM

                * (cid1.text)

            } > .pIntRAM

*************************************************************************

            .ApplicationData :
            {
                # Define variables for C initialization code

                F_Xdata_start_addr_in_ROM = .;
                F_StackAddr               = ADDR(.xStack);
                F_StackEndAddr            = ADDR(.xStack) + SIZEOF(.xStack) - 1;
                F_Xdata_start_addr_in_RAM = .;

                # Define variables for SDK mem library

                # Data (X) Memory Layout

                        _EX_BIT     = 0;

                   # Internal Memory Partitions (for mem.h partitions)

                        _NUM_IM_PARTITIONS = 0;  # IM_ADDR_1 (no IM_ADDR_2 )

                   # External Memory Partition (for mem.h partitions)

                        _NUM_EM_PARTITIONS = 1;   # EM_ADDR_1

                   FmemEXbit = .;
                        WRITEH(_EX_BIT);
                   FmemNumIMpartitions = .;
                        WRITEH(_NUM_IM_PARTITIONS);
                   FmemNumEMpartitions = .;
                        WRITEH(_NUM_EM_PARTITIONS);
                   FmemIMpartitionList = .;
                        WRITEH(ADDR(.xIntRAM_DynamicMem)*1);
                        WRITEH(SIZEOF(.xIntRAM_DynamicMem)*1);
                   FmemEMpartitionList = .;
                        WRITEH(ADDR(.xExtRAM_DynamicMem)*1);
                        WRITEH(SIZEOF(.xExtRAM_DynamicMem)*1);
```

```
                    # Add rest of the data into External RAM

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

                    F_Xdata_ROMtoRAM_length = 0;

                    F_Xbss_start_addr = .;
                    _X_BSS_ADDR = .;

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

                    F_Xbss_length = . - _X_BSS_ADDR;  # Copy DATA

            } > .xExtRAM

#*************************************************************************

            FArchIO             = 0x0000;
            FArchCore           = ADDR(.xCoreRegisters);
            FArchInterrupts     = ADDR(.pInterruptVector);

}
```

Chapter 6
Type 1 Telephony Features Library Applications

6.1 Type 1 Verification Test

To verify the functionality of the Type 1 Telephony Features Library, an individual test application is provided. The test processes the samples that are precaptured from a nominal FSK transmission. The corresponding Caller ID will be printed on screen. The test application is located in the *...telephony\cidtype1\Test* directory. The name of the test project is *cid1test.mcp*.

6.1.1 Test Set-up and Procedure

The test application runs in simulation and does not require external equipment. Before executing the test application, the *Target Setting Protocol* must be selected in the *Simulator*. It can be set in *Protocol* option in *M56800 Target Settings*.

The test project for Type 1 is *cid1test.mcp*. Before loading the executable project, the project must be compiled with no errors. This is done by choosing the *Make* option in the *Project* menu or by pressing [F7]. To load the executable project, choose the *Debug* option in the *Project* menu or press [F5].

To run the test application, use the mouse to choose the green arrow button or manually press [F5] on the keyboard. After the application starts running, the application initializes and automatically processes the precaptured samples.

The functionality of the Type 1 Telephony Features Library will be verified and a Caller ID message will be printed on the console window. The application stops after completing the process and prints “PASS” on the console window.

Should the application print “FAIL”, the Type 1 Telephony Features Library is not operating correctly. This is not expected to happen for this test application; however, if it does, please report the failure and test conditions to Motorola for resolution.

6.2 Example Application Using *cid1.lib*

An example using the Type 1 Telephony Features Library to illustrate how to set the library variables for *Type1CID* in the Control structure of *teldefs.h* is shown in **Code Example 6-1**. This code does not exist in the actual *cidtype1* directory in the SDK.

Code Example 6-1. Use of *Type1CID* Interface

```
#include <stdio.h>
#include <stdlib.h>
#include "teldefs.h"
#include "cid1.h"
#include "mem.h"
#ifdef USEPARSER
#include "cidparser.h"
#endif


struct teldefs_tsSamples  Line1Samples;
struct cid_tsData* pcid1Data;
struct teldefs_tsControl Line1Control;
#ifdef USEPARSER
teldefs_sParser ParserControl;
#endif

int main(void)
{

      // Initialize some necessary structure elements and call create function

      Line1Control.messageDone=0;
      Line1Control.cidByteReady = 0;
      Line1Control.ExtUseCheck=0;
      Line1Control.NoExtFound=1;
      Line1Control.FrameErrors=0;
      Line1Control.dtmfRequest=0;
      Line1Control.dtmfComplete=0;
      Line1Control.hookSwitch = 0;
      Line1Control.flashCommand = 0;
      Line1Control.cwdCommand = 0;
      pcid1Data = Type1CIDcreate(&Line1Control);

#ifdef USEPARSER
      ParserControl.FskMessageIndex=0;
      ParserControl.FskParserLength=0;
#endif



      while(1) {

            /* Wait for 5 8KHz samples to be received and transmitted by */
```

```
		/* the Codec Interrupt Service Routine. */

		while(!SamplesReady){
			;				/* do nothing */
		}
		CalleridAppMain();

	}

	Type1CIDDestroy(pcid1Data,&Line1Control);

return 0;
}


void CalleridAppMain(){
int i;

	/* copy samples from Codec buffers */
	for( i = 0; i < 5 ; i++){
		codecBufferLeftout[i]  = Line1Samples.audio[i];
		codecBufferRightout[i]  = Line1Samples.line[i];
		Line1Samples.line[i] =  codecBufferLeftin[i];
		Line1Samples.audio[i] = codecBufferRightin[i];
	}

	// Get Ring Signal from DAA

	if( /* Poll Ring Detect Bit Here */ ){
		Line1Control.cidRingPolarity = 1;	// Ring present
	}
	else{
		Line1Control.cidRingPolarity = 0;	// Ring not present
	}


	// Call Type 1 Telephony Features Library

	Type1CID(pcid1Data,&Line1Control,&Line1Samples);


	// Process command for line flash

	if(Line1Control.flashCommand){
		Line1Control.flashCommand = 0;
		if(Line1Control.flashPolarity == 1) go_offhook();
		else go_onhook();
	}

#ifdef USEPARSER

	// Use Type 1 and 2 Telephony Parser Library

	CIDMessageParser(&ParserControl, &Line1Control);
```

```
	if(ParserControl.FskParserLength != 0){
		/* Parsed Message Ready. Send to Output Device */
		if(ParserControl.ErrorType == 0){
			for( i = 0 ; i < ParserControl.FskParserLength ; i++)
			printf("%c",ParserControl.FskParserBuffer[i]);
		}
		ParserControl.FskParserLength=0;
	}
#elseif

	// Use Custom Parser

	if(Line1Control.cidByteReady){
		/* Buffer cid bytes here*/
		cid_message_buffer[cid_message_index++] = Line1Control.cidByte;
	}
	if (Line1Control.messageDone){
		if(Line1Control.FrameErrors == 0){
			/* Call custom Parser here */
		 }
		 else Line1Control.FrameErrors = 0;
	}
#endif

}
```

Code Example 6-1 shows a basic application using the *cid1.lib* module. This module is designed to interact with the Type 1 and 2 Telephony Parser Library (*cidparser.lib*); however, independent usage of the *cid1.lib* module is supported. When implementing a full duplex speakerphone, the *cid1.lib* module is intended to be used with the *fdspk.lib* and *gec.lib* modules; however, custom line echo canceller and full duplex speakerphone software can also interface with the *cid1.lib* module.

This example shows how to manage Caller ID information using either the Type 1 and 2 Telephony Parser Library or a custom parser. The defintion USEPARSER is undefined when a custom parser is desired. When using a custom parser, the application must transfer the input bytes into a buffer, then process them. When using the Type 1 and 2 Telephony Parser Library, this is done automatically.

While this module is primarily intended to be used for Caller ID reception, other convenient features are added to allow for simple integration into a Caller ID telephone. While the module is in an on-hook state (*Line1Control.hookSwitch* = 0), the *line[]* samples buffer will be filled with ring generator samples during power ringing. Otherwise, the *line[]* samples will be filled with zeros. These samples may act as an output to an audio channel (for example, to be played to a speaker).

While the module is in an off-hook state (*Line1Control.hookSwitch* = 1), the *audio[]* samples buffer will be filled with DTMF samples when a DTMF digit is requested and will remain unchanged otherwise. The *line[]* samples remain unchanged during the off-hook state, unless *Line1Control.handsFreeLayer1* is set (a condition which is not shown in this example). In this case, the *gec[]* samples will be copied to the *line[]* samples structure.

The *gec[]* samples can be obtained by using the *gec.lib* module, or by using a separate custom echo canceller. The *audio[]* samples will be modified by the *fdspk.lib* module (or custom full duplex speakerphone software) in this case. See the description of these modules and FeaturePhone application for more detailed information.

In this example, there are two GPIO pins; one output pin, which places the telephone interface on-hook or off-hook, and one input pin, which indicates the ring detect from the the telephone interface. The ring detect pin is polled in the application and its polarity is placed in the variable *Line1Control.cidRingPolarity*. This information is necessary for the Type 1 Telephony Features Library, even when *disableRinger* is set (disabling the ring tone generator) and should be polled at a rate of 1600/sec as shown in the application. However, slower polling rates may be tolerated.

It is assumed that the functions *go_onhook()* and *go_offhook()*, which are not shown in **Code Example 6-1**, would simply set or clear the output pin connected to the telephone interface. However, note that these are not the same as when the application actually puts the module into an on-hook or off-hook state. In this situation, the application must also set or clear the *Line1Control.hookSwitch* variable and then call *Type1CIDinit()*.

The Codec Interrupt Service Routine is also not shown in this example. This routine is assumed to transmit and receive 8KHz samples from a codec and read/write to the buffers *codecBufferLeftin[]* and *codecBufferLeftout[]*. This codec is connected to the telephone line interface. This routine is also assumed to transmit and receive 8KHz samples from a codec and read/write to the buffers *codecBufferRightin[]* and *codecBufferRightout[]*. This codec is connected to an audio interface. When the ISR has received/transmitted five samples, the buffers are full/empty and the *SamplesReady* flag is set, indicating to the application that it should call *CalleridAppMain()*. The calling rate of this routine should be 1600 calls/second.

In the beginning of the *CalleridAppMain()* routine, the codec samples are copied into the module's sample structure. Note that for the output samples, the *line[]* samples are copied into the right (audio) codec channel, and the *audio[]* samples are copied into the left (line) codec channel. This sample "criss-cross" is indicative of the natural signal flow for a telephone.

Chapter 7
License

7.1 Limited Use License Agreement

This software is available under a separate license agreement from Motorola Incorporated. Licensing information can be obtained from your Motorola sales representative or authorized distributor.

For additional product information, see **http://www.motorola.com/semiconductors/**.

Index

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

O

P

S

T

V

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

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

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

SDK135/D