TEXAS INSTRUMENTS

Application Report
SPRA462

Implementing Caller ID on Fixed-Point DSPs

Digital Signal Processing Solutions

Abstract

Caller ID (Identification) is a method of transmitting telephone caller information, such as telephone number and/or caller name. Data is transmitted to the subscriber in the ringing phase of the telephone (on hook) using the V.23 modem standard, which is an FSK-type modem at 1200 bit/s. The interpretation of demodulated data is specified by a protocol.

This application report describes the implementation of the protocol for message handling and the receiver, which includes an FSK demodulator and UART. General information about the Calling Line Identification Service is also provided.

Contents

Figures

Tables

Introduction

This application report provides general information about the Calling Line Identification Service. The Caller ID software package includes a demodulation routine according to the CCITT recommendation V.23 and the implementation of a protocol for data interpretation according to the ETS Draft 300 659-1.

The Caller ID protocol specification is discussed in the section, *Caller ID Protocol Specification*. Different ways of signaling are specified in the section, *Signaling*. The software, including protocol implementation and demodulation routine, that constitutes the software core is described in *Description of Caller ID Routines*. Required processor resources are given in the section, *Processor Resources*. The *Hardware Block Diagram* section offers an example of a demonstration board in the form of a block diagram.

Caller ID Protocol Specification

Basic mode communication covers transmission of data between network and Terminal Equipment (TE), either before ringing is applied or without any ringing (idle state). The interface at the network end consists of three layers:

- ❒ Physical layer
- ❒ Data link layer
- ❒ Presentation layer

The first layer defines data symbol encoding, modulation and analog line conditions. The second layer defines framing of messages for transmission and a simple error checking procedure. The third layer defines how application-related information is assembled into a message.

Physical Layer

Physical layer requirements refer to the network end of the local loop. An asynchronous FSK-type voice-band modem is used for data transmission. Therefore, a frequency modulator is required in the local exchange (LE) and a demodulator in the TE. The modem has to meet 1200 baud V.23 standard characteristics.

Caller ID information is transmitted by means of data octets. Each octet is preceded by one start-bit and is followed by one stop-bit, as shown in Figure 1.

Figure 1. Data Octet

1	H	G	F	E	D	C	B	A	0
Stop bit	2^7 (most significant)	2^6	2^5	2^4	2^3	2^2	2^1	2^0 (least significant)	Start bit

The order of transmission is: Start bit first, Stop bit last. Octets are transmitted according to the growing order of their number (octet 1, octet 2, etc.)

Data Link Layer

The Data Link Layer provides the framing of data and bit error detection capability in form of a checksum. The Data Link message format is shown in Figure 2.

Figure 2. Data Link Message Format

Channel Seizure Signal	Mark Signal	Message Type	Message Length	Presentation Layer Message	Checksum

Channel Seizure Signal

The channel seizure signal consists of a sequence of 300 continuous bits of alternating 0's and 1's. The block of bits starts with a 0 and ends with a 1.

Mark Signal

The mark signal consists of sequence of 180 ± 25 mark bits or 80 ± 25 mark bits. The block of mark bits is equivalent to a series of stop bits (no data transmission).

Message Type

The message type consists of 1 octet, which is used to identify the message.

Message Length

The message length (1 octet) contains the number of octets that constitute the Data Link layer message except Message Type, Message Length and Checksum octets. The length of the Presentation Layer message may vary between 3 and 255 octets.

Presentation Layer Message

The Presentation Layer Message contains Caller ID information according to the message type. For details, see the sections, *Presentation Layer* and *Protocol Implementation*.

Checksum

The checksum (1 octet) contains the twos complement of the modulo 256 sum of all octets in the message, starting from the Message Type to the end of the message except the checksum itself. The 8-bit sum of these octets, including the checksum and ignoring the carry bit, must thus equal zero; otherwise, the message has to be assumed to be incorrect.

Presentation Layer

The Presentation Layer contains the type of information transmitted, the number of data octets transmitted, and the data octets themselves. This sequence is contained in one parameter and may be repeated several times. The format of the Presentation Layer is illustrated in Figure 3.

Figure 3. Presentation Layer Message Format

Parameter Type	Parameter Length	Parameter Data Octet(s)	...	Parameter Type	Parameter Length	Parameter Data Octet(s)

Parameter 1 Parameter N

The elements of one parameter are explained in the following. The implementation of the protocol with a list of parameter types is described in more detail in the section, *Protocol Implementation*.

Parameter Type

The parameter type (1 octet) specifies the type of information transmitted. Parameter types are, for example, Calling Line Directory Number and Caller Name.

Parameter Length

The parameter length (1 octet) contains the number of data octets transmitted in the following.

Parameter Data Octet(s)

One parameter may contain from 1 up to 253 data octets. The data octets contain information according to the parameter type. Data octets are encoded as specified by CCITT Recommendation T.50 (see also draft ETS 300 659-1 Annex E).

Signaling

Signaling is carried out prior to data transmission. The so-called TE Alerting Signal (TAS) indicates that data is to be transmitted. Three kinds of TAS are specified for the Caller ID service:

- ❒ Dual-tone alerting signal (DT-AS)
- ❒ Ringing pulse alerting signal (RP-AS)
- ❒ Line reversal followed by DT-AS

The network operator sends one of these three TAS before data transmission.

In case data transmission is associated with ringing, the FSK modulated Caller ID message is followed by ring patterns. Otherwise, an appropriate idle condition is applied to the local loop after FSK modulation transmission.

A DT-AS detector is implemented as well as a timing verification for the RP-AS option.

The different software modules included in the Caller ID package are described in the following section.

Description of Caller ID Routines

The Caller ID software package contains five modules: a control routine on application level, a main assembly routine that calls Caller ID subroutines, a module for DT-AS detection and V23 receiver. The software only handles on-hook reception of Caller ID. The choice of alerting signal to be recognized is a user-programmable option, either DT-AS or RP-AS.

C-Control Routines on Application Level

Two control routines are executed on the highest level of the Caller ID software: **Manage_CID** and **transfer_data**. The routines are contained in the file **CID.C**.

C-Function Manage_CID

The signature of this function is void Manage_CID(int AS). The function uses an input parameter associated with the choice of alerting signal contained in the variable AS. This variable has to be set in a program (e.g., main()) before calling Manage_CID(AS). An example of main() is shown in Figure 4.

Figure 4. Example of main() Program

```
#define RP_AS          0              /* Ring pulse alerting signal */
#define DT_AS          1              /* Dual tone alerting signal */
#define AS_OPTION      DT_AS          /* Choice of alerting signal */

int CID;               /* external variable indicating that CID function executed when set to 1,
                          not executed when set to 0 */
main()
{
    int AS=AS_OPTION;

    CID=1;             /* initialize CID: 0->no Caller ID, 1->Caller ID function activated */

    while(1)
    {
        if (CID)
            Manage_CID(AS);

        /* main program continues */
    }
}
```

Manage_CID calls the main Caller ID routine **CallerID** (cf. the section, *Main Caller ID Routine*), which handles Caller ID reception. **CallerID** returns a status that is copied to the variable **status**. A value different from zero indicates that Caller ID information has been received. In this case the function **transfer_data** is called. Each bit in the variable status corresponds to a different Parameter Type of the Presentation Layer. (Parameter types taken into account by the software are described in more detail in the section, *Protocol Implementation*.) The corresponding bit number is determined (e.g., Bit0: bit number=0, Bit1: bit number=1, etc.). This bit number is the input parameter of the function **transfer_data**, which is described in the following section.

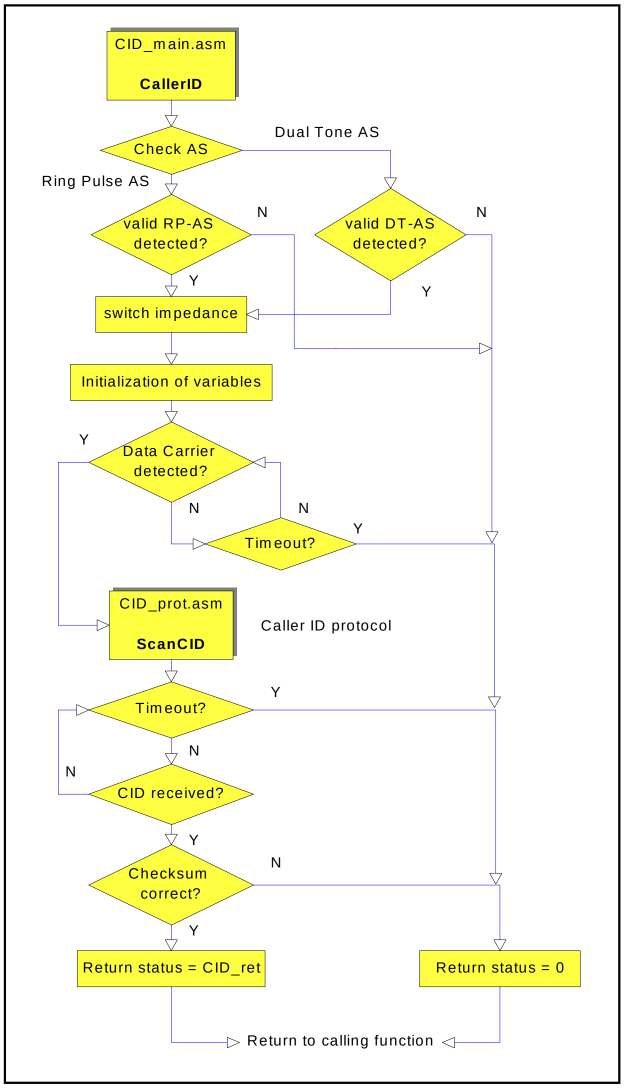
C-Function transfer_data

This function is called when valid data has been received. It is called as many times as bits are set in the variable **status**. The prototype of the function is void transfer_data(int bit_number). Each bit number corresponds to a different parameter type. Data corresponding to parameter type(s) transmitted are displayed on the PC screen. For telephony applications, this function has to be replaced by a display function that outputs the data to the telephone display.

Main Caller ID Routine

The routine **CallerID** controls the whole sequence of Caller ID reception at the TE, from alerting signal over channel seizure to message extraction. **CallerID** is a C-callable function written in assembly language. The corresponding C-prototype is int CallerID(int AS). **AS** is the same input parameter as used in the function **Manage_CID**. If **AS** is set to RP_AS (0), a check for a valid ring pulse alerting signal is carried out. Otherwise the routine expects a valid dual tone alerting signal detection (cf. the section, *Alerting Signal Detector*). A flow chart of the complete routine **CallerID** is shown in Figure 5.

Figure 5. Flow Chart of CallerID Routine



In case no alerting signal has been detected, **CallerID** returns zero to the calling routine (**Manage_CID**). If either RP-AS or DT-AS (depending on the choice made previously) has been detected, impedance is switched and Caller ID variables are initialized. If no data carrier is detected during the time T1 as specified by draft ETS 300 659-1, the routine exits on time out. T1 is given in Table 1 for DT-AS and RP-AS. T1 is the time from the end of alerting signal to the start of FSK modulation transmission.

Table 1. Time T1 for DT-AS and RP-AS

Alerting Signal	T1
DT-AS	45-500 ms
RP-AS	500-800 ms

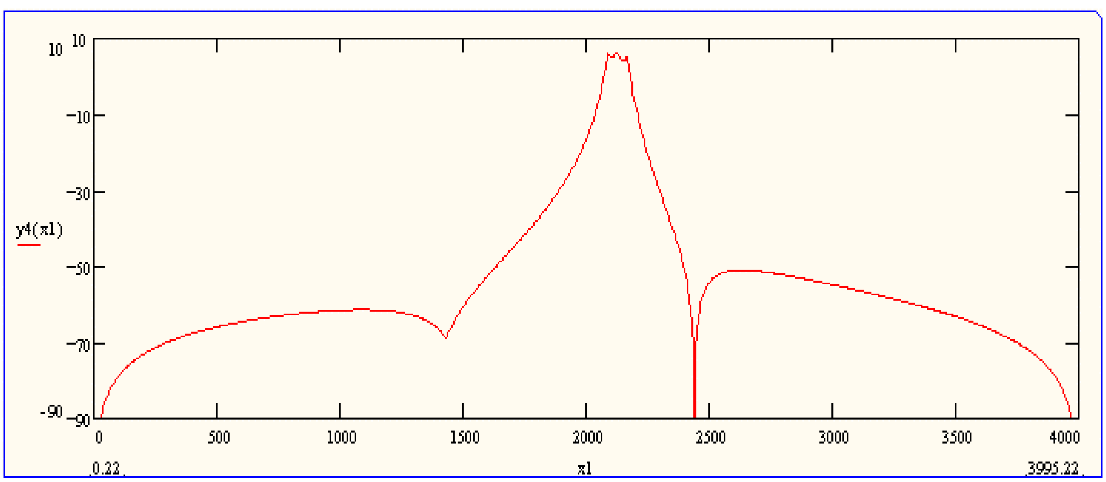
If the data carrier is detected within T1, the Caller ID protocol routine is executed (cf. the section, *Protocol Implementation*). Carrier detection is indicated by BIT1 of the variable **FSKREG** (cf. the section, *AGC and Carrier Detection*). Data reception must be completed within a certain time after carrier detection. If the time out specified by the constant CID_TO is exceeded, Caller ID reception is aborted. Once the protocol routine is finished (no exit on time out) BIT9 and BIT10 of status variable **CID_status** indicate whether data reception has been completed successfully or not. The signification of bits used in **CID_status** is explained in the section, *Protocol Implementation*. In case of valid data, **CallerID** returns a status containing the type(s) of information received (cf. the section, *C-Function Manage_CID*). Otherwise, zero is returned to the calling function.

Alerting Signal Detector

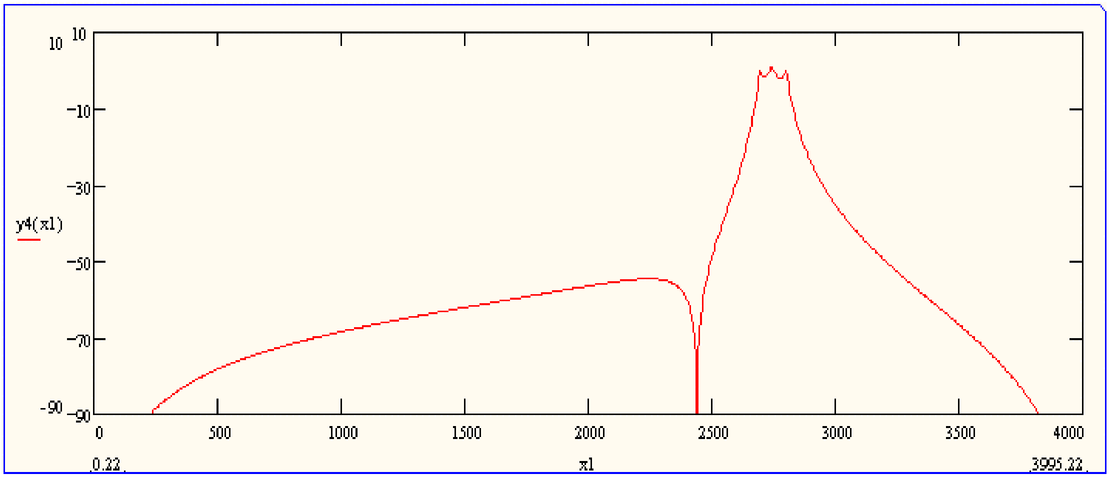
The Software can detect the DT alert signal, which consists of a dual tone at 2130Hz and 2750Hz. The routine **CheckCPE** is called in the 8 kHz sample interrupt. This routine controls the different states of DT-AS detection (start of detection, end of alerting signal or no detection). It calls the routine CPE, which contains the DT-AS detector filters.

The filter response curves for the individual tone detection filters on the DT alerting signal are shown in Figure 6. Each filter is a 6th order IIR resonator made up of 3 bi-quads. Due to the closeness of the two frequencies and the bandwidths within which the signals must be recognized, the filters incorporate a notch at the midpoint between the two frequencies (2440Hz). This prevents a single frequency at this level from falsely triggering both resonators, which would occur if simpler filter designs were used. To ensure recognition of the signal, which may be 38dB below the speech level though present for a relatively long time filters with a stop-band rejection of at least 40dB, these filters have a stop band rejection of 50-60dB.

Figure 6. Low (a) and High (b) Frequency Filter for DT-AS Detection



(a)



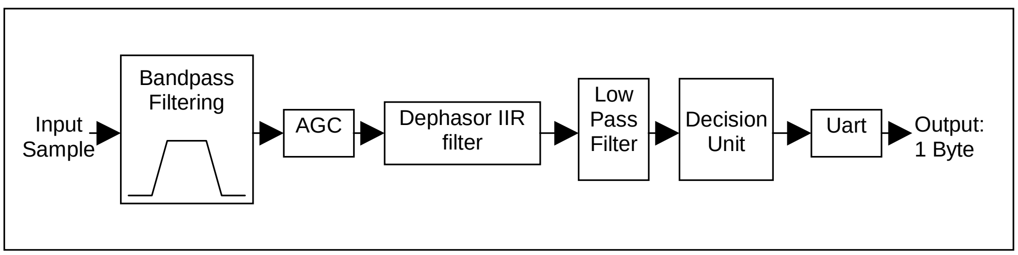
(b)

V.23 Receiver

A demodulator is implemented at 1200 bit/s. The receiver is shown in Figure 7. The V.23 standard uses binary FSK modulation with a central frequency of 1700 Hz. Binary 1 is represented by a frequency of 1300 Hz, binary 0 by 2100 Hz.

All stages of the receiver can be found in the file, **RXV23F.ASM**. Variable declarations and filter coefficients are contained in the file, **V23.inc**.

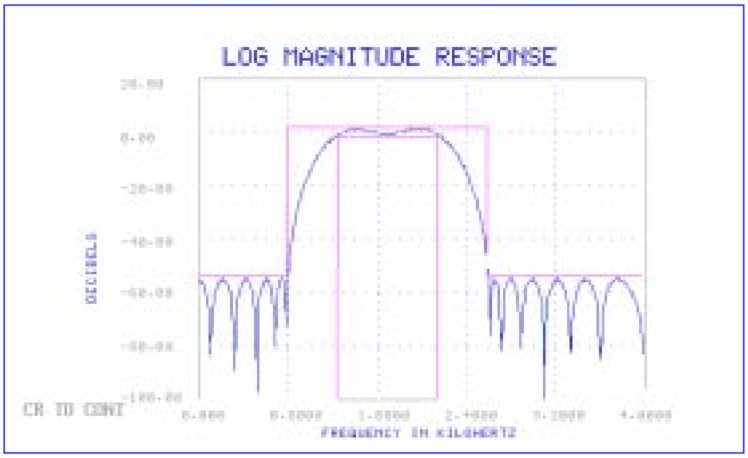
Figure 7. Stages in the FSK Receiver at 1200 Bit/s



Bandpass Filter

The first stage consists of bandpass filtering the input sample. The FIR bandpass filter can be found in the file, **V23.inc**. The filter characteristics are shown in Figure 8.

Figure 8. FIR Bandpass Filter



AGC and Carrier Detection

The next stage is an AGC. The gain applied to the input sample is also used for carrier detection. Two gain constants define the states “Carrier ON” and “Carrier OFF” with a hysteresis of 2 dB as required by the V.23 standard. The low AGC gain threshold insures carrier detection for input signals ≥ -43 dBm. The high AGC gain threshold returns the state “Carrier OFF” for input signals ≤ -48 dBm. These gain thresholds are user programmable and are described in more details in the next section. In case of “Carrier ON,” BIT1 in the variable **FSKREG** is set. In the “Carrier OFF” state,” BIT1 is reset. FSKREG is C-callable.

Carrier Detection Thresholds

The high AGC gain threshold specifies the input signal level of approximately -48 dBm. For gain values greater than or equal to this threshold the carrier is considered to be “OFF”. The low AGC gain threshold defines a signal level of about -43 dBm. If the AGC gain takes values less than this threshold the carrier is considered as “ON”. Between these two thresholds a hysteresis is carried out. The constants given in Table 2 have to be set in the file, **V23.INC**.

Table 2. Carrier Detection Thresholds

Low AGC Gain Threshold	**High AGC Gain Threshold**
DCDon	DCDoff

During the initialization phase, these constants are copied to the variables, **AGCTHRESL** and **AGCTHRESH**.

Demodulator at 1200 Bits/s

Data reception is carried out at 1200 bit/s. The stages that are part of the corresponding receiver are shown in Figure 7. After bandpass filtering and AGC, the input x(n) is applied to a dephasing all-pass filter that carries out a phase shift of $\pi/2$ for the central frequency (1700 Hz). The dephasor d(n) is given by

$$d(n) = a_1 x(n) + a_2 x(n-1) + x(n-2) - a_2 d(n-1) - a_1 d(n-2)$$
$$y(n) = x(n) d(n)$$

The output y(n) is obtained by multiplying x(n) by the delayed input d(n). Y(n) is then lowpass filtered and the sign of the lowpass filtered signal indicates whether a binary 0 or 1 has been received.

A compiler switch allows the choice between two demodulator versions, a low MIPS version and a higher MIPS version. For the low MIPS version the constant **LOWMIPS** has to be set to 1 in the file, **V.23.INC**. **LOWMIPS** has to be set to 0 for the higher MIPS demodulator. The higher MIPS version is of better quality regarding the bit error rate (BER) at a given signal-to-noise ratio (SNR). This is achieved using a 3 times oversampling lowpass filter instead of a simple lowpass filter used in the low MIPS version.

UART

The decision as to whether a binary 0 or 1 has been transmitted is input to the UART module. Because there are 6.666... samples per bit at 1200 bit/s, the UART function uses a 24 kHz oversampling counter. Synchronization is carried out on the start bit, then the unit counts 8 data bits and expects a stop bit at the end. The extracted byte is then copied in the upper 8 bits of the variable **FSKREG**. Once a byte has been extracted, it is indicated by setting BIT0 of **FSKREG** to one. After reading the byte on protocol level BIT0 has to be reset. This is done by the protocol routines that handle message extraction and put data into buffers according to the type of information received. These routines are described in the following section.

Protocol Implementation

The protocol is implemented according to the specification described in the section, *Caller ID Protocol Specification*. The program is divided into subroutines, with each of them covering a part of Caller ID reception, such as channel seizure, identification of message type etc. One routine, called Scan_CID, controls the succession of subroutines. All protocol routines are described in this section.

Protocol Control Routine: Scan_CID

This routine is called by CallerID (cf. the section, *Main Caller ID Routine*) after carrier detection. Scan_CID uses two registers, FSKREG and CID_status, the bits of which are explained in Figure 9.

Figure 9. CID_status (a) and FSKREG (b) Registers

CID_status register

15	14	13	12,11	10,9	8	7,6	5	4	3,2,1,0
n.u.	AS_ON	RING	n.u.	RET_STAT	CID_ON	n.u.	BUFFER_INIT	CID_EVEN	CID_function

Bit 15:	Not used
Bit 14:	Alerting signal detected
Bit 13:	Long ring (no RP-AS) present
Bit 12,11:	Not used
Bit 10,9:	Return status of Caller ID protocol: 01: abort, error during transmission 10: bad checksum 11: checksum correct
Bit 8:	Caller ID receiver enabled
Bit 7,6:	Not used
Bit 5:	Flag for initialization of pointer to current data buffer
Bit 4:	Flag for storage of two octets in on 16-bit word
Bit 3,2,1,0:	Offset of CID function table for function to be executed next

(a)

FSKREG register

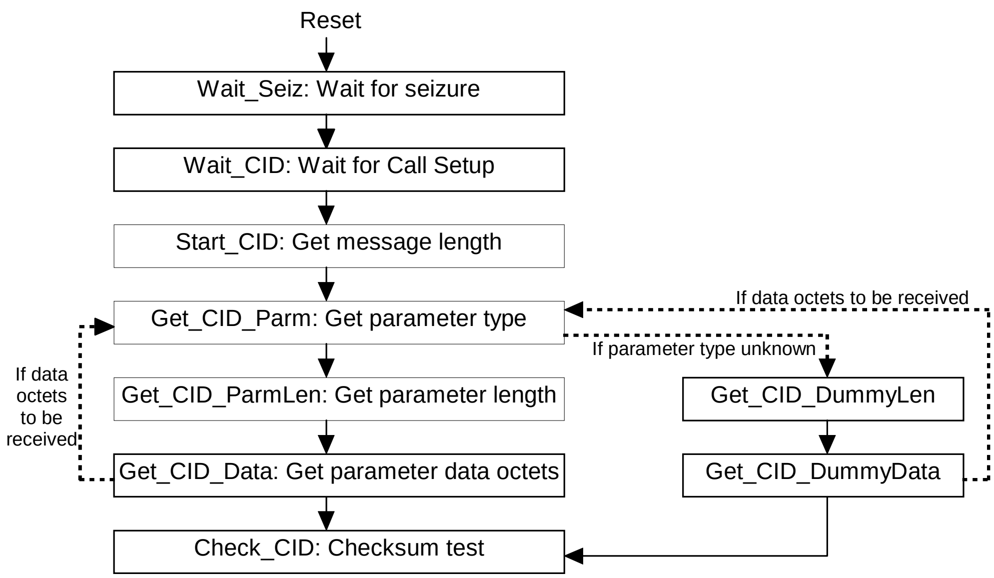
15,14,13,12,11,10,9,8	7,6,	5,4,3,2	1	0
Data octet	Not used	Reserved	CARRIERON	CHARON

Bit 15-8 (8 bits):	Data octet received
Bit 7,6:	Not used
Bit 5,4,3,2:	Used by FSK receiver but not visible on protocol level
Bit 1:	Flag for carrier detection
Bit 0:	Flag for reception of one data octet

(b)

A subroutine is called whenever an octet has been received (which is indicated by BIT1 of **FSKREG**). The subroutine called is determined by BIT0-BIT3 of **CID_status**. These bits correspond to the number of the routine in a function table. A conditional call is carried out for each subroutine. Depending on the parameters of the Caller ID protocol (e.g., message length, parameter type), the next subroutine to be called is determined. This is illustrated by the flow diagram shown in Figure 10.

Figure 10. Flow Diagram of Protocol Implementation



First channel seizure and identification of message type have to be carried out. The corresponding routines are Wait_Seiz and Wait_CID.

Channel Seizure: Wait_Seiz

This routine waits for alternating 0's and 1's, which will be received as 55h. When this value is received, the next function to be executed is Wait_CID.

Identification of Message Type: Wait_CID

This routine waits for end of seizure, handles the transition seizure/mark, and waits for message type. The only message type implemented is 'Call Setup' (80h). Call Setup is the first data octet of the Caller ID message (see Table 3).

Table 3. Message Type Coding

Type (Hexadecimal)	Message Name
80H	Call Setup

Message Length: Start_CID

The data octet that is expected to follow 'Call Setup' contains the number of octets to be received in the following. If the message length is greater than zero Get_CID_Parm will be called next.

Parameter Type: Get_CID_Parm

Parameter Type is the first octet of the Presentation Layer Message. It specifies the type of information that will be transmitted. Parameter types currently implemented are given in Table 4.

Table 4. Parameter Types Supported by Protocol Implementation

Type Value (Binary)	Type Value (Hexadecimal)	Length	Parameter Name
0001	01h	8	Date and Time
0010	02h	max. 20	Calling Line Identity (CLI)
0100	04h	1	Reason for absence of CLI
0111	07h	max. 20	Calling Party Name (CPN)
1000	08h	1	Reason for absence of CPN
1011	11h	1	Call type
1100	12h	max. 20	First Called Line Identity (in case of forwarded call)
1111	15h	1	Type of Forwarded call (in case of forwarded call)

A buffer of a specified length is reserved for each type of information. Other parameter types not listed above may easily be added to the current implementation. Likewise, parameter types not needed for the CLIP service can be taken out. The only parameters that are mandatory for the PSTN CLIP service are Calling Line Identity and Reason for Absence of Calling Line Identity (cf. ETS 300 659-1 Annex A).

If a parameter type value received does not correspond to any of the services taken into account by the current system, a dummy read is initiated. This means that the overall checksum is computed without saving parameter data octets. Parameter types supported are explained in the following sections.

Date and Time

Date and time are provided to the user, indicating the point in time when the message has been generated at the LE (see Table 5). It can be used to set internal equipment clocks and calendars.

Table 5. Date and Time Parameter

Octet Number	Contents
1	01H: Parameter type
2	08h: Parameter length
3,4	Month
5,6	Day
7,8	Hours
9,10	Minutes

Calling Line Identity

The purpose of this parameter is to identify the origin of the call (see Table 6). Thus, this number may be used to call back the caller.

Table 6. CLI Parameter

Octet Number	Contents
1	02H: Parameter type
2	Parameter length
3	Digit 1
4	Digit 2
...	...
n+2	Digit n

Reason for Absence of Calling Line Identity

The parameters explaining why CLI has not been transmitted are described in Table 7.

Table 7. Reason of Absence of CLI Parameter

Octet Number	Contents
1	04H: Parameter type
2	01H: Parameter length
3	“O”: Number unavailable “P”: Private

Calling Party Name

Table 8 describes the parameters used to identify the name of the caller.

Table 8. Calling Party Name Parameter

Octet Number	Contents
1	07H: Parameter type
2	Parameter length
3	Character1
4	Character 2
...	...
n+2	Character n

Reason of Absence of Calling Party Name

Table 9 summarizes the parameters for Calling Party Name transmission.

Table 9. Reason for Absence of Calling Party Name Parameter

Octet Number	Contents
1	08H: Parameter type
2	01H: Parameter length
3	“O”: Unavailable “P”: Private

Call Type

Different call types are given in Table 10. If no Call Type parameter is transmitted, the call type will be "Voice Call".

Table 10. Call Type Parameter

Octet Number	Contents
1	11H: Parameter type
2	01H: Parameter length
3	01H: Voice Call 02H: CLI Ring Back when free call 03H: Calling Name Delivery 81H: Message Waiting Call

First Called Line Identity

This parameter is transmitted for forwarded calls to identify the first called party.

Table 11. FCLI Parameter

Octet number	Contents
1	12H: Parameter type
2	Parameter length
3	Digit 1
4	Digit 2
...	...
n+2	Digit n

Type of Forwarded Call

This parameter specifies the type of forwarded call.

Table 12. Call Type Parameter

Octet Number	Contents
1	15H: Parameter type
2	01H: Parameter length
3	00H: Unavailable or unknown call type 01H: Forwarded call on busy 02H: Forwarded call on no reply 03H: Unconditional forwarded call 04H: Deflected call (after alerting) 05H: Deflected call (immediate) 06H: Forwarded call on inability to reach mobile subscriber

Parameter Length: Get_CID_ParmLen

This routine reads the second element of one parameter in the Presentation Layer Message into the variable CID_Len. If it exceeds the message length, Caller ID reception will be aborted.

Parameter Data Octets: Get_CID_Data

All data octets of one parameter transmitted are saved in the corresponding buffer. If more than one parameter is transmitted in the Presentation Layer, the routine Get_CID_Parm will be executed once again. Otherwise, a checksum test will be carried out next (Check_CID).

Read Dummy Length: Get_CID_DummyLen

If the parameter type does not correspond to any of the eight parameter values supported (cf. Table 3), a dummy read will be initiated. The purpose of this routine is to receive valid data preceding or following the dummy read with the checksum computed over the total number of octets received.

Read Dummy Data: Get_CID_DummyData

Data associated with a parameter type unknown to the system is only used for updating the checksum. If there are still data octets to be received, the next parameter type will be read. Otherwise, the checksum test will be executed next.

Checksum Test: Check_CID

The last octet in the Caller ID message contains the checksum that is the twos complement of the modulo 256 sum of all octets in the message, starting from the message type (Call Setup 80H) and excluding the checksum itself. For the checksum test, an 8-bit sum over all octets received so far has been computed. Adding the checksum octet without taking into account the carry bit must result into zero; otherwise, the data received must be assumed to be corrupt. After the checksum test, the status that indicates whether or not Caller ID reception is successfully completed is contained in BIT 9 and 10 of the register CID_status. This is processed by the main Caller ID routine described in the section, *Main Caller ID Routine*.

Processor Resources

Table 13 summarizes the memory occupation (RAM and ROM) as well as computational load (MIPS) utilized by Caller ID modules. MIPS are only given for routines executed in real time at 8 kHz. The values in parentheses are given for the higher MIPS version (cf. *Demodulator at 1200 Bits/s*).

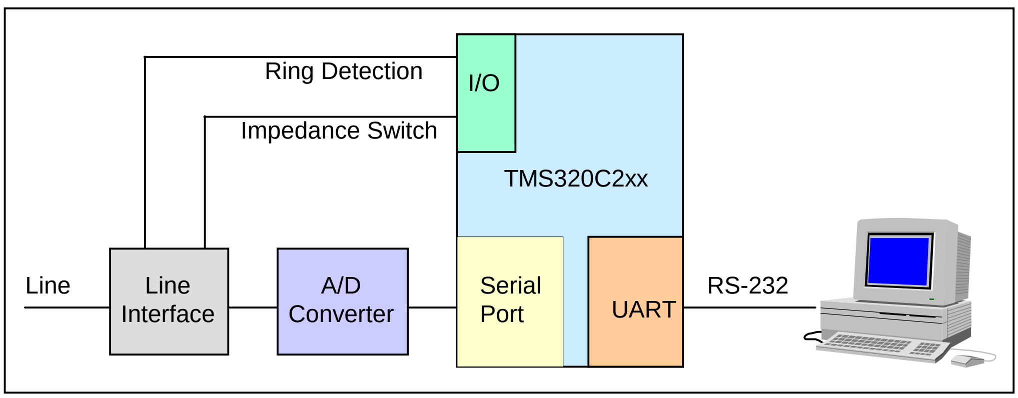
Table 13. Processor Resources Required for FSK Receiver and Transmitter

	ROM	RAM	MIPS
C-Application	500	34	<0.1
Protocol	720	38	<0.1
Demodulator	340	104	2 (3.4)
AS Detector	190	43	1.2
Sample Interrupt	90	14	0.4
Σ	1840	233	<3.8 (<5.2)

Hardware Block Diagram

Figure 11 shows an example of a demonstration board for Caller ID reception. Two general purpose I/Os are used for ring detection and impedance switching. A Caller ID receiver needs only an A/D converter that is connected to the synchronous serial port clocked at 8 kHz. Displaying of the received Caller ID message is carried out by transferring data from the on-chip UART to a PC (host) via RS-232.

Figure 11. CID_status (a) and FSKREG (b) Registers



Summary

The Caller ID software described in this report implements a receiver for on-hook data transmission at the LE. The software package includes:

- V23 demodulator
- DT-AS detector
- Caller ID protocol
- Application S/W for display

References

ETS 300 659-1: *Public Switched Telephone Network (PSTN) Subscriber line protocol over the local loop for display (and related) services; Part 1: On hook data transmission*, ETSI 1995

CCITT Recommendation V.23 (1988): *600/1200-baud modem standardized for use in the general switched public telephone network*

CCITT Recommendation T.50 (1992): *International Standard Alphabet (IRA)*

Caller ID on TMS320C2xx, Texas Instruments (1997)

Appendix A. Source Code

File: Mainc.c

```
/****************************************************/
/* File: MAINC.C      */
/*         */
/* Author: Katrin Matthes Dec. 97    */
/*         */
/*  Description: Application level, Main program  */
/*         */
/*  Copyright:      (c)Texas Instruments France */
/*                   All Rights Reserved  */
/****************************************************/

#define RP_AS  0  /* Ring pulse alerting signal */
#define DT_AS  1  /* Dual tone alerting signal */
#define AS_OPTION DT_AS  /* Choice of alerting signal */

         int CID;  /* external variable indicating that CID function executed when set to
         1, not executed when set to 0 */
int AS=AS_OPTION;
/* 1: detection of alerting signal for CID */
/* 0: no detection of alerting signal */

char transfer_buf[32];
int table_read(int *table); /* external function */

main()
{
 int AS_OPT=AS_OPTION;

 CID=1;   /* initialize CID: 0->no Caller ID, 1->Caller ID function activated */

 while(1)
 {
  if (CID)
   Manage_CID(AS_OPT);
  /* main program continues */
 }
}
int ReadTable(int *table)
{
    int i=0;
    char temp;

    do{
        temp=table_read(table+(i>>1));
        if (temp == 0)
            break;
```

```
        transfer_buf[i]=temp>>8;
        i++;
        if ((temp & 0x00ff) == 0)
            break;
        transfer_buf[i]=temp & 0x00ff;
        i++;

    }while(i < 32);

    transfer_buf[i]=0;      /* if no character is added afterwards, i indicates last
element */

    return i;
}


void Print(char *Pt)
{
/* Display routine specific to connection DSP-host */
}
```

***File*: Cid.c**

```
/****************************************************/
/* File: CID.C       */
/*          */
/* Author: Katrin Matthes Nov. 97    */
/*          */
/*  Description:    Application level, Display */
/*          */
/*  Revisions:       */
/*          */
/*  Copyright:      (c)Texas Instruments France */
/*                   All Rights Reserved  */
/****************************************************/

#define RING  0x2000  /* bit 13 in CID_status */
#define MAX_BUF  8 /* number of buffers in CID */

/* external functions */
int CallerID(int as);
int Print(char *Ptr);
void Init_Ptrs(void);
int ReadTable(int *table);

/* external variables */
extern int CID_status;
extern int CID_ret;
extern int *CID_Ptr[MAX_BUF];
extern int AS; /* DT alerting signal on/off AS=1/0 */
extern char transfer_buf[32];
extern int CLI0[], CLI1[], CLI2[], CLI3[], CLI4[], CLI5[], CLI6[], CLI7[];

void Manage_CID(int AS_OPT)
{
  int status,i;

  status=CallerID(AS_OPT);
  if (!status)
  {
    if (CID_status & RING)
    {
     Print("RING");
     CID_status &= ~RING; /* reset ring bit */
    }
  }
  else
```

```
  {
     for (i=0;i<MAX_BUF;i++)
     {
       if (status & (0x0001 << i))
         transfer_data(i);
     }
  }
}


void transfer_data(int number)
{
  int *tmp_ptr;
  int BufferLen,i, toggle=0,count=1;

  switch (number)
  {
    case 0x0000 : count=ReadTable(CLI0);
       BufferLen=4;
       break;
    case 0x0001:   count=ReadTable(CLI1);
       BufferLen=10;
       break;
    case 0x0002:  count=ReadTable(CLI2);
       BufferLen=10;
       break;
    case 0x0003 : count=ReadTable(CLI3);
       BufferLen=1;
       break;
    case 0x0004:   count=ReadTable(CLI4);
       BufferLen=10;
       break;
    case 0x0005:   count=ReadTable(CLI5);
       BufferLen=1;
       break;
    case 0x0006:  count=ReadTable(CLI6);
       BufferLen=1;
       break;
    case 0x0007:  count=ReadTable(CLI7);
       BufferLen=1;
       break;
    default:   break;
  }
  tmp_ptr=CID_Ptr[number];
        /* points to current CID buffer */

  if (BufferLen > 1)
  {
    for (i=0;i<BufferLen<<1;i++)
    {
    if(!toggle)
    transfer_buf[count+i]=(char)(*(tmp_ptr)>> 8);
    else
    {
          transfer_buf[count+i]=
                                (char)(*(tmp_ptr)&0x00ff);
      tmp_ptr++;
    }
    toggle^=1;
    }
    transfer_buf[count+i]=(char) 0x0000;
  }
  else
  {
    transfer_buf[count]=(char) *tmp_ptr;
    transfer_buf[count+1]=(char) 0x0000;
  }
  Print(transfer_buf);
```

```
}
```

File: PrintTab.asm

```
****************************************************
*         *
* String tables to use with the routine  *
* int ReadTable(int *table)    *
*         *
* Only ROM used (no .const in RAM)   *
****************************************************

      .def_CLI0,_CLI1,_CLI2,_CLI3,_CLI4,_CLI5,_CLI6,_CLI7


_CLI0
 .string  "DATE/TIME:"
 .word   0000h

_CLI1
 .string  "Tel.:"
 .word   0000h

_CLI2
 .string  "FCLIP:"
 .word   0000h

_CLI3
 .string  "No CLIP:"
 .word   0000h

_CLI4
 .string  "Name:"
 .word   0000h

_CLI5
 .string  "No Name:"
 .word   0000h

_CLI6
 .string  "Call Type:"
 .word   0000h

_CLI7
 .string  "FCLIP Type:"
 .word   0000h
```

File: Main.asm

```
****************************************************
* File:      MAIN.ASM    *
*         *
* Description: Main Assembler Routine:   *
*    init, IT management  *
*         *
* Revisions:      *
*         *
*   Copyright:      (c)Texas Instruments France *
*                   All Rights Reserved   *
****************************************************

      .include "cid.inc"
 .mmregs
    .ref    _c_int0
    .ref    _CID_status
```

```
    .ref    CheckCPE, RXV23
    .def    _table_read
  .def _Tim0, FromAD

_Tim0 .usect  "SMain",1 ;Timer, incremented in 8 kHz IT
FromAD  .usect  "SMain",1 ;sample from A/D in 8 kHz sample IT
ItPage0 .usect  "SMain",12   ; for context save

    .sect   "vectors"
    b       Reset   ; reset
    b       $       ; int0
    b       SigmaIt ; sigma delta
    b       $       ; real time counter
    .space  16*16   ; reserve 16 words
    b       $    ; timer
    b       $       ; serial receive
    b       $       ; serial transmit
    b   DTrap   ; trap (for debug)
    .text
; Trap for debug
DTrap
  dint
  b  $

; after reset
;
Reset
    dint
    ssxm
    sovm
; initialization specific of MSP58C8x, to be changed for TMS320C2xx
; clock
    lack    #0F537h ; 111 1 0101 001 10111
                ; 0101  SDPD 4.096*12/12=Sigma delta 8.0kHz
                    ; 001   PCPD 4.096*12/4*2 = 6.14MIPS
                    ; 10111 PLLFG 4.096*12MHz
    ldpk    #0
    sacl    FREQ
; init port b0
    lack    #04FA1h ; 0100 1111 1010 0001
    ldpk    #0      ; b5,7,8,9,10,11,14 output  ; B0
    sacl    BDIR
    lack    #04FA0h ; 0100 1111 1010 0000
    sacl    BO      ; put bit to 1
; init 58C20
    lack    #003h   ; 00 0011
    ldpk    #0
    sacl    ADAC    ; enable AD and DA
; interrupt mask
    lack    #02h
    ldpk    #0
    sacl    IMR     ;  enable SDINT
; init CID status register
    ldp     #_CID_status
    zac
    sacl    _CID_status
;
    eint

    call    _c_int0  ; start C program
    b       $

;---------------------
;
; Sigma Delta IT
;
;---------------------
SigmaIt
    .newblock
```

```
; save context
    sst     ItPage0+0   ; status
    sst1    ItPage0+1   ; status 1
    ldpk    #0          ;
    sach    ItPage0+2   ; accuh
    sacl    ItPage0+3   ; accul
    sar     ar0,ItPage0+4
    sar     ar1,ItPage0+5
    sar     ar2,ItPage0+6
    sar     ar3,ItPage0+7
    spm     0
    sph     ItPage0+8   ; prh
    spl     ItPage0+9   ; prl
    mpyk    1
    spl     ItPage0+10  ; t
    ssxm
; soft timer
    ldp  #_Tim0
    lac     _Tim0
    add     #1
    sacl    _Tim0
    ldpk    #0

    lac     SDAD        ; get AD sample
    ldpk    #FromAD
    sacl    FromAD      ; save sample
    ldp     #_CID_status
    lac     _CID_status
    and     #CID_ON
    bz      no_CID

  call CheckCPE
  call RXV23
    b       no_CID
no_CID
; restore context
    ldpk    #0
    lt      ItPage0+9
    mpyk    1           ; prl
    lt      ItPage0+10  ; t
    lph     ItPage0+8   ; prh
    lar     ar3,ItPage0+7
    lar     ar2,ItPage0+6
    lar     ar1,ItPage0+5
    lar     ar0,ItPage0+4
    zals    ItPage0+3   ; accl
    addh    ItPage0+2   ; acch
    lst1    ItPage0+1    ; status1
    lst     ItPage0+0    ; status
    eint
    ret

; C-callable routine that allows looking up values in a ROM table
_table_read
        mar     *-         ; AR1 = &(&program memory word)
        lac     *+         ; acc = &program memory word
        tblr    *          ;
        lac     *,ar1     ; acc = program memory word
        ret
```

File: CID_Main.asm

```
****************************************************
* File:   CID_MAIN.ASM     *
*         *
* Author:  Katrin Matthes Nov. 97  *
*         *
* Description:  Main Caller ID Routine  *
*         *
* Revisions:       *
*         *
*   Copyright:      (c)Texas Instruments France *
*                   All Rights Reserved   *
****************************************************

      .include "cid.inc"

   .def _CallerID
   .def _reset_imp
   .ref _CID_status,_CID_ret
   .ref Scan_CID,Init_CID,resV23,_InitRXV23
   .ref _Tim0
   .ref CID_temp
   .global _FSKREG

   .mmregs


*********************************************************
* Main caller ID routine:
* int CallerID(int AS)
*
* Controls sequence of Caller ID reception:
* 1) Presence of alerting signal
* 2) V.23 Carrier
* 3) Message Extraction
*
* Input: Choice of alerting signal: DT-AS or RP-AS
* Output: Status information on Caller ID reception:
*  0->idle or error during transmission
*  !=0->bits set give information about type of
*  information received
*
* ARs used: AR0, AR1, AR2
*********************************************************

_CallerID
  POPD    *+      ; save context and set ar0,ar1
      SAR     AR0,*+
    SAR     AR1,*
    LARK    AR0,#0     ; 0 local variable
      LAR     AR0,*0+   ; new ar1

; get argument
    lark    ar2,-3      ; offset of argument
    mar     *,ar2
    mar     *0+

  lac  *
  bnz  short_ring
check_ring
; check if ring present

; insert reading of I/O pin on TMS320C2xx
        ldp     #0
        lac     BI
        and     #RING_MASK ; I/O
        bnz     no_ring
```

```
  ldp  #_Tim0  ; reset timer for ring count
  zac
  sacl _Tim0

;wait for end of ring
Ring_on
        ldp     #0
        lac     BI
        and     #RING_MASK
        bz      Ring_on
;check min. ring length
        ldp     #_Tim0
        lac     _Tim0
        sub     #RING_MIN
        blz     no_ring
;check max. ring length
        sub   #RING_MAX ;value RING_MAX=RING_MIN+(max.ring-min.ring)
        blez    short_ring
  ldp  #_CID_status
  lac  _CID_status
  or #RING
  sacl _CID_status
        b  no_ring

; valid short ring has been detected => start CID reception
short_ring
        larp  ar2
  lac  *  ; check if we want to receive DT_AS
  bz ring_init
  ldp  #_CID_status
  lac  _CID_status
  and  #CID_ON
  bnz  no_as_init

ring_init
; insert writing of I/O pin for impedance switch on TMS320C2xx
        dint
        ldp #0
        lac     BO     ; switch impedance
        and     #~SWITCH_IMP ;b11
        sacl    BO
  eint

  dint
  call Init_CID ; init cid routines:
     call resV23   ; FSK receive and protocol
  call _InitRXV23
  call _InitDCD ; init Carrier detection thresholds
     ldp  #_CID_status
     lac  _CID_status
     or #CID_ON
     sacl _CID_status
        eint   ;enable interrupts

  larp ar2
  lac  *  ; check if we want to receive DT_AS
  bz no_AS
no_as_init
  lac  _CID_status
  and  #AS_ON
  bz no_ring
no_AS
  zac
  ldp  #_Tim0
  sacl _Tim0 ; reset timer for DCD timeout
no_DCD
  ldp     #_FSKREG
        lac     _FSKREG
```

```
        and     #CARRIERON  ; check if data carrier detected
        bnz     dcd_on
        ldp     #_Tim0
        lac     _Tim0
        sub     #DCD_TO
        blez  no_DCD
timeout1
        dint
  ldp  #0
        lac     BO
        or #SWITCH_IMP
        sacl    BO
  eint

  ldp  #_CID_status
  lac  _CID_status
  and  #~CID_ON  ; reset bit CID_ON
  sacl _CID_status

        b       no_ring
dcd_on
  ldp  #_Tim0  ; reset timer for ring count
  zac
  sacl _Tim0

loop_CID
  call Scan_CID
  ldp     #_Tim0
        lac     _Tim0
        sub     #CID_TO
        bgz timeout1
  ldp  #_CID_status
  lac  _CID_status
  and  #0600h
     bnz  check_ret_stat
  b loop_CID
check_ret_stat
  ldp   #0
  dint
  lac     BO
  or     #SWITCH_IMP ;reswitch impedance
  sacl    BO
  eint

  ldp  #_CID_status
  lac  _CID_status
  and  #~CID_ON ; reset bit
  sacl _CID_status
  and  #0600h
  sub  #0200h
  bz no_ring  ; abort, error during transmission
  sub  #0200h
  bz no_ring  ; crc bad
  sub  #0200h
  bnz  no_ring
crc_ok
  ldp  #_CID_ret
  lac  _CID_ret
  b exit1
no_ring
   zac
exit1
   MAR     *,AR1   ;
      SBRK    #1
      LAR     AR0,*-  ; restore ar0
      PSHD    *
  ret

_reset_imp
```

```
 ldp  #0
 dint
 lac  BO
 or #SWITCH_IMP ;reswitch impedance
 sacl     BO
 eint
 zac
 ldp  #_CID_status
 sacl _CID_status
 ret

_InitDCD
 ldp  #_AGCTHRESL
 lac  #DCDon
 sacl _AGCTHRESL
 lac  #DCDoff
 sacl _AGCTHRESH
 ret
```

***File*: CID_Prot.asm**

```
****************************************************
* File:   cid_prot.asm     *
*         *
* Author:  Katrin Matthes April 97 *
*         *
* Description:  Caller ID protocol   *
*         *
* Revisions:       *
*         *
*   Copyright:      (c)Texas Instruments France *
*                   All Rights Reserved   *
****************************************************
 .include    "cid_var.inc"

 .global _FSKREG

 .def Init_CID            .def Scan_CID
 .def _Init_Ptrs
 .def CID_temp

CID_temp .usect   "CID_Temp",1

****************************************************
*         *
* Caller ID Initialization     *
*         *
****************************************************
Init_CID
 call _Init_Ptrs
 zac
 ldp  #CID_Byte
 sacl CID_Byte
 sacl _CID_status
 sacl CID_Checksum
 sacl _CID_ring
 sacl CID_Len
 sacl CID_ParmLen
 sacl MaxParmLen
 sacl _CID_ret
 call clear_buffers
 ret

****************************************************
*         *
* Clears all CID buffers     *
*         *
```

```
****************************************************
clear_buffers
  zac
  larp ar3
  lar  ar3,#_Date_Time_Buf
  rpt  #DATE_TIME_LEN-1
  sacl *+
  lar  ar3,#_CLIP_Buf
  rpt  #CLIP_LEN-1
  sacl *+
  lar  ar3,#_FCLIP_Buf
  rpt  #FCLIP_LEN-1
  sacl *+
  lar  ar3,#_No_CLIP_Buf
  rpt  #NO_CLIP_LEN-1
  sacl *+
  lar  ar3,#_Call_Name_Buf
  rpt  #CALL_NAME_LEN-1
  sacl *+
  lar  ar3,#_No_Call_Name_Buf
  rpt  #NO_CALL_NAME_LEN-1
  sacl *+
  lar  ar3,#_Call_Type_Buf
  rpt  #CALL_TYPE_LEN-1
  sacl *+
  lar  ar3,#_FCLIP_Type_Buf
  rpt  #FCLIP_TYPE_LEN-1
  sacl *+
  ret


****************************************************
*          *
* Caller ID Control Routine    *
*          *
****************************************************
Scan_CID
  ldp  #_FSKREG
  lac  _FSKREG
  and  #CHARON  ;check flag bit
  bz no_byte
  larp ar3
  lar  ar3,#CID_temp
  lac  _FSKREG
  and  #~CHARON   ;reset flag
  sacl _FSKREG
  lac  _FSKREG,8
  ldp  #CID_Byte
  sach CID_Byte
  lac  CID_Byte
  and  #00ffh
  sacl CID_Byte
  ldp  #_CID_status
  lac  _CID_status
  and  #000fh
                              ;bit0-3 contain function to execute
  add  #CID_functions
  tblr *
  lac  *
  cala    ; branch to function
no_byte
  ret


****************************************************
*          *
* Wait for Channel Seizure     *
*          *
****************************************************
```

```
Wait_Seiz
 ldp  #CID_Byte
 lac  CID_Byte
 xor  #55h
 bnz  NoSeiz
 lac  _CID_status
 and  #0fff0h
 or #WAIT_CID
 sacl _CID_status
NoSeiz
 ret


****************************************************
*          *
* Wait for Call Setup Byte     *
*          *
****************************************************
Wait_CID
 ldp  #CID_Byte
 lac  CID_Byte
 xor  #55h
 bz done  ; still in seizure
 lac  CID_Byte
 sub  #CALL_SETUP
                    ; first data byte must equal 80h
 bnz  done  ; transition seizure/mark  lac  _CID_status
 and  #0fff0h    ; zero out function to execute next
 or #START_CID    ; so that 'or' below will work
                                       properly
 sacl _CID_status         lac  CID_Byte
 sacl CID_Checksum
 b done
 lac  _CID_status
 and  #0fff0h
 sacl _CID_status
done ret


****************************************************
*          *
* Start Caller ID Reception    *
*          *
****************************************************
Start_CID
 ldp  #CID_Byte  ; get global data length
 lac  CID_Byte
 sacl CID_Len
 add  CID_Checksum
 sacl CID_Checksum
 sub  #100h
 blz  no_wrap
 sacl CID_Checksum
no_wrap
 larp ar3
 lar  ar3,CID_Len
 lac  _CID_status
 and  #0fff0h
 banz next
 or #CHECK_CID
 sacl _CID_status
 b done1
next
 or #GET_CID_PARM
 sacl _CID_status
done1 ret
****************************************************
*          *
* Read the Caller ID Parameter    *
```

```
*          *
****************************************************
Get_CID_Parm
  call CID_update
  bz abort2   ; ACC contains CID_Len
  lac  CID_Byte ; CID_Byte contains service
                                      parameter
  ldp  #BufNum
  sub  #1
  bz date
  sub  #1
  bz clip
  sub  #2
  bz no_clip
  sub  #3
  bz c_name
  sub  #1
  bz no_name
  sub  #9
  bz c_type
  sub  #1
  bz fclip
  sub  #3
  bz fclip_type
  b dummy
date
  zac
  sacl BufNum
  lac  #DATE_TIME_LEN,1
  sacl MaxParmLen
  b done2
clip
  lac  #1
  sacl BufNum
  lac  #CLIP_LEN,1
  sacl MaxParmLen
  b done2
fclip
  lac  #2
  sacl BufNum
  lac  #FCLIP_LEN,1
  sacl MaxParmLen
  b done2
no_clip
  lac  #3
  sacl BufNum
  lac  #NO_CLIP_LEN
  sacl MaxParmLen
  b done2
c_name
  lac  #4
  sacl BufNum
  lac  #CALL_NAME_LEN,1
  sacl MaxParmLen
  b done2
no_name
  lac  #5
  sacl BufNum
  lac  #NO_CALL_NAME_LEN
  sacl MaxParmLen
  b done2
c_type
  lac  #6
  sacl BufNum
  lac  #CALL_TYPE_LEN
  sacl MaxParmLen
  b done2
fclip_type
  lac  #7
```

```
 sacl BufNum
 lac  #FCLIP_TYPE_LEN
 sacl MaxParmLen
 b done2
dummy
 ldp  #_CID_status  ; service parameter not in
                                      list
 lac  _CID_status ; -> initiate dummy read
 and  #0fff0h
 or #GET_CID_DUMMY_LEN
 sacl _CID_status
 ret
abort2
 ldp  #_CID_status
 lac  _CID_status
 or #CID_ABORT
 sacl _CID_status
 ret
done2
 ldp  #BufNum
 lac  #1
 rpt  BufNum
 sfl
 sfr
 ldp  #_CID_ret
 or _CID_ret
 sacl _CID_ret
 ldp  #_CID_status
 lac  _CID_status
 and  #0fff0h
 or #GET_CID_PARM_LEN
 sacl _CID_status
 ret


****************************************************
*          *
* Read the Caller ID parameter length   *
*          *
****************************************************
Get_CID_ParmLen
 call CID_update ; ACC contains CID_Len
 sub  CID_Byte
 blz  abort3   ;if parameter length > global length
                              abort
 lac  CID_Byte
 sacl CID_ParmLen
 bz next2
 ldp  #MaxParmLen ; if parameter length >
                                      allocated space then initiate
 sub  MaxParmLen ; a dummy read
 bgz  dummy_len
 ldp  #_CID_status
 lac  _CID_status
 and  #0fff0h
 or #GET_CID_DATA
 sacl _CID_status
 b done3
dummy_len
 ldp  #_CID_status
 and  #0fff0h
 or #GET_CID_DUMMY_DATA
 sacl _CID_status
 b done3
next2
 ldp  #_CID_status  ; Paramter length = 0 and
                                      ;global data length > 0
 lac  _CID_status ; -> get next paramter
 and  #0fff0h  ; otherwise do checksum test
```

```
 lar  ar3,CID_Len
 banz nxt_parm,*,ar3
 or #CHECK_CID
 sacl _CID_status
 b done3
nxt_parm
 or #GET_CID_PARM
 sacl _CID_status
 b done3
abort3
 ldp  #_CID_status
 lac  _CID_status
 or #CID_ABORT
 sacl _CID_status
 call clear_buffers
done3
 ret


****************************************************
*          *
* Read the Caller ID parameter data   *
*          *
****************************************************
Get_CID_Data
 call CID_update ; ACC contains CID_Len
 larp ar3
 lac  _CID_status
 and  #BUFFER_INIT
                      ; set Buffer_ptr to buffer start @
 bnz  no_init
 lac  #_CID_Ptr
 add  BufNum
 sacl Buffer_ptr
 lar  ar3,Buffer_ptr
 lac  *
 sacl Buffer_ptr
 lac  _CID_status
 or #BUFFER_INIT  ; set bit, so that init only
                                     ; carried out once
 sacl _CID_status
no_init
 lar  ar3,Buffer_ptr
                              ; points to current data buffer
 lac  _CID_status
 and  #CID_EVEN
 bnz  get_even
 lac  CID_Byte,8
 sacl *
 b past_odd
get_even
 lac  CID_Byte
 add  *
 sacl *+
past_odd
 sar  ar3,Buffer_ptr
 lac  _CID_status
 xor  #CID_EVEN  ; toggle bit
 sacl _CID_status ; same function to be executed next
 lac  CID_Len
 bz Check_It
 lac  CID_ParmLen
 sub  #1
 sacl CID_ParmLen
 bz NxtParm
 b done4
NxtParm
 ldp  #_CID_status
 lac  _CID_status
```

```
  and  #0fff0h
  or #GET_CID_PARM
  and  #~BUFFER_INIT
                        ; reset buffer-init bit for next parameter
  and  #~CID_EVEN
                        ; reset toggle bit for next parm
  sacl _CID_status
  b done4
Check_It
  ldp  #_CID_status
  lac  _CID_status
  and  #0fff0h
  or #CHECK_CID
  sacl _CID_status
done4
  ret


****************************************************
*          *
* Read the dummy parameter length  *
*          *
****************************************************
Get_CID_DummyLen
  call CID_update
  sub  CID_Byte
  blz  abort4   ; if parameter length > global
                                      ; length then abort
  lac  CID_Byte
  sacl CID_ParmLen
  bz next4
  ldp  #_CID_status
  lac  _CID_status
  and  #0fff0h
  or #GET_CID_DUMMY_DATA
  sacl _CID_status
  b done5
next4
  ldp  #_CID_status
  lac  _CID_status
  and  #0fff0h
  lar  ar3,CID_Len
  banz nxt_parm2,*,ar3
  or #CHECK_CID
  sacl _CID_status
  b done5
nxt_parm2
  or #GET_CID_PARM
  sacl _CID_status
  b done5
abort4
  ldp  #_CID_status
  lac  _CID_status
  or #CID_ABORT
  sacl _CID_status
  call clear_buffers
done5
  ret


****************************************************
*          *
* Read the Caller ID dummy data    *
*          *
****************************************************
Get_CID_DummyData
  call CID_update ;ACC contains CID_Len
  bz Chck_Sum
  lac  CID_ParmLen
```

```
  bz Nxt_Parm
  sub  #1
  sacl CID_ParmLen
  b done6
Chck_Sum
  ldp  #_CID_status
  lac  _CID_status
  and  #0fff0h
  or #CHECK_CID
  sacl _CID_status
  b done6
Nxt_Parm
  ldp  #_CID_status
  lac  _CID_status
  and  #0fff0h
  or #GET_CID_PARM
  sacl _CID_status
done6
  ret


****************************************************
*          *
* Caller ID Checksum Test    *
*          *
****************************************************
Check_CID
      ldp #CID_Byte
                             ; last byte contains complement of
                                      ; mod 256 sum of all bytes
  lac  CID_Byte
      add     CID_Checksum
      and     #00ffh
  bz check_ok
  lac  _CID_status
  or #CHECK_BAD
  sacl _CID_status
  b done7
check_ok
  lac  _CID_status
  or #CHECK_OK
  sacl _CID_status
done7
  lac  _CID_status
  and   #~CID_ON
  sacl _CID_status
  ret


****************************************************
*          *
* Update checksum and global data length *
*          *
****************************************************
CID_update
  ldp  #CID_Byte
  lac  CID_Byte
  add  CID_Checksum
  sacl CID_Checksum
  sub  #100h
  blz  no_wrap1
  sacl CID_Checksum
no_wrap1
  lac  CID_Len
  sub  #1
  sacl CID_Len
  ret
```

```
****************************************************
*          *
* Init array of pointers to CID buffers  *
*          *
****************************************************
_Init_Ptrs
 ldp  #_CID_Ptr
 lac  #_Date_Time_Buf
 sacl _CID_Ptr
 lac  #_CLIP_Buf
 sacl _CID_Ptr+1
 lac  #_FCLIP_Buf
 sacl _CID_Ptr+2
 lac  #_No_CLIP_Buf
 sacl _CID_Ptr+3
 lac  #_Call_Name_Buf
 sacl _CID_Ptr+4
 lac  #_No_Call_Name_Buf
 sacl _CID_Ptr+5
 lac  #_Call_Type_Buf
 sacl _CID_Ptr+6
 lac  #_FCLIP_Type_Buf
 sacl _CID_Ptr+7
 ret


*** CID Function Table in ROM ***
CID_functions
   .word Wait_Seiz
   .word  Wait_CID
   .word Start_CID
   .word Get_CID_Parm
   .word Get_CID_ParmLen
   .word Get_CID_Data
   .word Get_CID_DummyLen
   .word Get_CID_DummyData
   .word Check_CID
```

***File*: CID_Var.inc**

```
****************************************************
*          *
* Definition of caller Id constants and variables *
*          *
****************************************************

  .def _Date_Time_Buf, _CLIP_Buf, _FCLIP_Buf,
                      _No_CLIP_Buf
  .def _Call_Name_Buf, _No_Call_Name_Buf,
                      _Call_Type_Buf, _FCLIP_Type_Buf

  .def _CID_ring,  CID_Checksum
  .def CID_Len, CID_ParmLen, MaxParmLen, BufNum
  .def _CID_Ptr
  .def _CID_status
  .def Buffer_ptr
  .def _CID_ret, CID_Byte


***CID-Functions***
WAIT_SEIZ .set   0
WAIT_CID .set   1 ; CID function to execute is
START_CID .set   2 ; contained in bit 0-3 of
GET_CID_PARM .set   3 ; the CID status register
GET_CID_PARM_LEN .set   4
GET_CID_DATA   .set   5
GET_CID_DUMMY_LEN .set   6
```

```
GET_CID_DUMMY_DATA  .set    7
CHECK_CID   .set    8

***CID Return Status***
CID_ABORT  .set 0200h ; return status in bits 9,10
CHECK_BAD  .set 0400h ; of CID status register
CHECK_OK .set 0600h

***CID Service Parameters***
CALL_SETUP .set  80h
DATE_TIME  .set  01h
CLIP  .set  02h
FCLIP   .set  12h
NO_CLIP  .set  04h
CALL_NAME  .set  07h
NO_CALL_NAME  .set  08h
CALL_TYPE  .set  11h
FLIP_TYPE  .set  15h

***Buffer Lengths for CID Data Storage***
DATE_TIME_LEN  .set  4
CLIP_LEN   .set  10
FCLIP_LEN   .set  10
NO_CLIP_LEN   .set  1
CALL_NAME_LEN  .set  10
NO_CALL_NAME_LEN .set  1
CALL_TYPE_LEN  .set  1
FCLIP_TYPE_LEN   .set  1

***Misc***
MAX_BUF  .set 8
CID_TIMEOUT .set 16000 ; 2s
MAX_CID_LEN .set 72
CID_ON   .set 100h ; bit 8 in CID status register
CID_FLAG .set 800h ; bit 11 - - -
CID_EVEN .set  010h ; bit 4  - - -
BUFFER_INIT .set  020h ; bit 5  - - -

CHARON   .set  0001h ; bit 0 in FSKREG

_Date_Time_Buf .usect  "CID_Buf",DATE_TIME_LEN
_CLIP_Buf   .usect  "CID_Buf",CLIP_LEN
_FCLIP_Buf  .usect  "CID_Buf",FCLIP_LEN
_No_CLIP_Buf     .usect  "CID_Buf",NO_CLIP_LEN
_Call_Name_Buf .usect  "CID_Buf",CALL_NAME_LEN
_No_Call_Name_Buf .usect  "CID_Buf",NO_CALL_NAME_LEN
_Call_Type_Buf .usect  "CID_Buf",CALL_TYPE_LEN
_FCLIP_Type_Buf  .usect  "CID_Buf",FCLIP_TYPE_LEN

_CID_ring   .usect  "CID_Var",1
_CID_status   .usect  "CID_Var",1
CID_Checksum   .usect  "CID_Var",1
CID_Len    .usect  "CID_Var",1
CID_ParmLen   .usect  "CID_Var",1
MaxParmLen  .usect  "CID_Var",1
BufNum     .usect  "CID_Var",1
_CID_ret   .usect  "CID_Var",1

_CID_Ptr   .usect  "CID_Var",MAX_BUF
Buffer_ptr  .usect  "CID_Var",1

CID_Byte .usect  "CID_Var",1 ; received valid data byte
```

File: CID.inc

```
  .global _AGCTHRESL, _AGCTHRESH

FREQ .equ    0007h      ; frequency control register
                               ; reserved memory
BI .equ    000Ch      ; B port input register
BO .equ    000Dh      ; B port output register
BDIR .equ    000eh      ; B port direction register
ADAC .equ    0010h      ; sigma-delta ADC/DAC control reg
SDAD .equ    0011h      ; sigma-delta ADC input register
SDDA .equ    0012h      ; sigma-delta DAC output register
SAAD .equ    0013h      ; successive-approximation ADC reg


DCDon   .set 6100h ; AGC gain -43dBm
DCDoff   .set 6500h ; AGC gain -48dBm

CID_ON      .set 100h ; bit 8 in CID status reg.
RING_MASK     .set    0040h ; I/O B6 (input)
RING_MIN      .set    800 ; 100ms=>100 ms * 8 kHz
RING_MAX      .set    5800  ; < 1 s (clock at 8 kHz)
SWITCH_IMP    .set    0800h  ; I/O B11 (output)
DCD_TO      .set    8000   ; 1 s (clock 8 kHz)
DCD_ON        .set    1000h ; bit 12 in CID status reg.
RING  .set 2000h ; bit 13 in CID_status
AS_ON   .set 4000h ; bit 14 in CID_status
CID_TO   .set 20000 ; 2.5s (clock at 8 kHz)

CARRIERON  .set 0002h ; bit 1 in FSKREG
```

File: CPE.asm

```
********************************************
*        *
* Dual Tone Alerting Signal Detector *
*        *
********************************************
  .include "cid.inc"
  .mmregs
       .global resV23,CheckCPE
     .ref _CID_status
  .ref _AS ;=1->DT Alerting Signal, =0->Ring
  .ref FromAD

rptz    .macro  arg1
     zac
       mpyk    0
     rpt     arg1
       .endm

        .text
; initialize V23
resV23
  dint
  cnfd
  larp  ar3
    zac
  lar  ar3,#iir6_a1
  rpt  #CPE_delay-iir6_a1+1
  sacl *+
  ldp  #_AS
  lac  _AS
  bz no_CPE0
    lacl  #200       ;INITIALIZE CPE threshold
  ldp   #CPE_thr
```

```
    sacl CPE_thr
 lac  #NoCPE
 b end0
no_CPE0
 lac  #EndCPE
end0
 ldp  #CPEstat
 sacl CPEstat
 eint
 ret
************************************************************
; Alerting signal detector control routine
; Sequence:
; 1) NoCPE: wait for start of DT-AS
; 2) StCPE: DT-AS started, wait for end
; 3) EndCPE: end of DT-AS, AS status bit set in CID_status
************************************************************
CheckCPE
 ldp  #CPEstat
 lac  CPEstat
 bacc
NoCPE  call    CPE
       bit     CPE_tim,7
 bbz  EndCPE   ; wait for start of alerting signal
 ldp  #CPEstat
 lac  #StCPE
 sacl CPEstat
 b EndCPE
StCPE  call    CPE
       bit     CPE_tim,7
 bbnz EndCPE   ; wait for end of alerting signal
 ldp  #_CID_status
 lac  _CID_status
 or #AS_ON
 sacl _CID_status
 lac  #EndCPE
 ldp  #CPEstat
 sacl CPEstat
EndCPE
 ret
;
*****************************************************
*          *
*    Sub-routine CPE      *
*     This subroutine detects CPE alerting signal *
* Entry         *
*   Acc=Q15 PCM input     *
* Exit        *
*   If CPE present Acc=256     *
*   Else Acc<0       *
*          *
*****************************************************
CPE     ldp #FromAD
 lacc  FromAD
 ldp  #iir1_b1
    sacl iir1_b1    ;input for low freq filter
 sacl iir4_b1  ;input for high freq filter
;
; 1st bi-quad for low frequency filter
;
    lar   ar3,#iir1_b3    ;point to data
 larp ar3
 rptz #4
 macd #IIR_LF1,*- ;filter with params
    lta  *-              ;apac & dec ar
    sach  iir1_a1,1       ;save feedback terms
    sach  iir2_b1,1     ;input to next stage
;
; 2nd bi-quad for low frequency filter
```

```
;
 rptz #4
 macd #IIR_LF2,*- ;filter with params
    lta  *-              ;apac & dec ar
    sach  iir2_a1,1       ;save feedback terms
    sach  iir3_b1,1       ;input to next stage
;
; 3rd bi-quad for low frequency filter
;
 rptz #4
 macd #IIR_LF3,*- ;filter with params
    lta   *-              ;apac & dec ar
    sach  iir3_a1,1       ;save feedback terms
;
; measure signal strength
;
 abs     ;convert to amplitude
   sub   Low_En,15       ;add in old low energy filter
 sfr
 sfr
 sfr
 addh Low_En
 sach Low_En
;
; 1st bi-quad for high frequency filter
;
 rptz #4
 macd #IIR_HF1,*- ;filter with params
    lta  *-              ;apac & dec ar
    apac                    ;last parameter is div 2
    sach  iir4_a1,1       ;save feedback terms
    sach  iir5_b1,1       ;input to next stage
;
; 2nd bi-quad for high frequency filter
;
 rptz #4
 macd #IIR_HF2,*- ;filter with params
    lta  *-              ;apac & dec ar
    apac                    ;last parameter is div 2
    sach   iir5_a1,1       ;save feedback terms
    sach   iir6_b1,1       ;input to next stage
;
; 3rd bi-quad for high frequency filter
;
 rptz #4
 macd #IIR_HF3,*- ;filter with params
    lta   *-              ;apac & dec ar
    apac                    ;last parameter is div 2
      sach  iir6_a1,1       ;save feedback terms
;
; measure signal strength
;
 abs     ;convert to amplitude
    sub   High_En,15      ;add in old low energy filter
 sfr
 sfr
 sfr
 addh High_En
 sach High_En
;
; test for valid signal levels
;
    sub  CPE_thr,15      ;High_En > 100 = present
 blez no_CPE
    lacc  Low_En,1        ;Low_En > 100 = present
      sub  CPE_thr
 blez no_CPE
 lacc Low_En,3  ;check Low_En*9-High_En*4>0
 add  Low_En   ;(7.0dB Signal difference)
```

```
    sub   High_En,2
  blez no_CPE
  lacc High_En,3  ;check High_En*9-Low_En*4>0
  add  High_En  ;(7.0dB Signal difference)
      sub  Low_En,2
  blez no_CPE
;
; adapt CPE threshold detection
;
    lacc  Low_En
    sub   CPE_thr,2
    blz   Low_OK
    lac   Low_En,14
    sach  CPE_thr
;
; CPE present
;
Low_OK  lacc CPE_tim         ;increment CPE frame count
    sub   #255             ;test for CPE valid 256 frames
  blez CPE_ret
CPE_det lacl  #0
CPE_ret add   #256            ;0<acc<256 CPE present but invalid
      sacl  CPE_tim
      ret                     ;acc=256 CPE present
no_CPE  lacl  #200            ;CPE default threshold
    sacl  CPE_thr
    lacl #0               ;acc=0 no CPE signal
    sacl   CPE_tim
    ret
*
* Allocate space for CPE detection filters
; all 6 iir's must be kept in this order in data space
iir6_a1 .usect "CIDb0",1
iir6_a2 .usect "CIDb0",1
iir6_b1 .usect "CIDb0",1
iir6_b2 .usect "CIDb0",1
iir6_b3 .usect "CIDb0",1
dummy6  .usect "CIDb0",1 ;dummy for macd dmov
iir5_a1 .usect "CIDb0",1
iir5_a2 .usect "CIDb0",1
iir5_b1 .usect "CIDb0",1
iir5_b2 .usect "CIDb0",1
iir5_b3 .usect "CIDb0",1
dummy5  .usect "CIDb0",1 ;dummy for macd dmov
iir4_a1 .usect "CIDb0",1
iir4_a2 .usect "CIDb0",1
iir4_b1 .usect "CIDb0",1
iir4_b2 .usect "CIDb0",1
iir4_b3 .usect "CIDb0",1
dummy4  .usect "CIDb0",1 ;dummy for macd dmov
iir3_a1 .usect "CIDb0",1
iir3_a2 .usect "CIDb0",1
iir3_b1 .usect "CIDb0",1
iir3_b2 .usect "CIDb0",1
iir3_b3 .usect "CIDb0",1
dummy3  .usect "CIDb0",1 ;dummy for macd dmov
iir2_a1 .usect "CIDb0",1
iir2_a2 .usect "CIDb0",1
iir2_b1 .usect "CIDb0",1
iir2_b2 .usect "CIDb0",1
iir2_b3 .usect "CIDb0",1
dummy2  .usect "CIDb0",1 ;dummy for macd dmov
iir1_a1 .usect "CIDb0",1
iir1_a2 .usect "CIDb0",1
iir1_b1 .usect "CIDb0",1
iir1_b2 .usect "CIDb0",1
iir1_b3 .usect "CIDb0",1
dummy1  .usect "CIDb0",1 ;dummy for macd dmov
Low_En  .usect "CIDb0",1    ;low frequency energy
```

```
High_En .usect "CIDb0",1 ;high frequency energy
CPE_thr .usect "CIDb0",1    ;CPE detection threshold
CPE_tim .usect "CIDb0",1    ;CPE active time
CPEstate .usect "CIDb0",1   ;state of CPE
CPE_delay .usect "CIDb0",1   ;delay time in CPE state machine
CPEstat  .usect  "CIDb0",1,1


        .sect "RomData"
****************************************************
*          *
* CPE Alerting Signal Tables Follow   *
*          *
****************************************************
*;
*; 6th order IIR filter as bi-quads for detecting DT alerting signal
*;   signal order is b3,b2,b1,a2,a1
*;
*; Low frequency filter
*;
IIR_LF1 .word 1003,-911,1107,-31343,-4427
                                     ;pass 2.088, stop 1.430
IIR_LF2 .word 2002,0,-2002,-31664,-8530
                                     ;pass 2.169, stop 0.0 & 4.0
IIR_LF3 .word 7950,5389,7950,-29884,-6092
                                     ;pass 2.124, stop 2.440
*;
*; High frequency filter
*;
IIR_HF1 .word 1094,0,-1094,-31726,-33453/2
                                     ;pass 2.694,stop 0.0&4.0
IIR_HF2 .word 1094,0,-1094,-31536,-37867/2
                                     ;pass 2.802,stop 0.0&4.0
IIR_HF3 .word 7950,5389,7950,-30514,-34855/2
                                     ;pass 2.743,stop 2.440
```

File: RXV23F.asm

```
****************************************************
*  File:           RX23F.ASM    *
*          *
* Author:  Katrin Matthes Nov. 97  *
*          *
*  Description:  V.23 receiver    *
*          *
* Revisions:       *
*          *
*  Copyright:  (c)Texas Instruments France  *
*              All Rights Reserved  *
****************************************************

        .global _InitRXV23,RXV23,_ResetV23
        .global _V23REG
        .def    TRIGGER
  .global V23CCoef
   .global _FSKREG, FSKCOUNT, FSKCHAR, FSKC, AGCTEMP,
                      AGCGAIN, AGCSamp
  .global _AGCTHRESL, _AGCTHRESH, AGCGAIN, DecCOUNT
  .global AgcFSK, FSKDec

  .ref FromAD

        .include "V23.inc"
  .mmregs

        .text
```

```
********************************************
*  V23 initialization
********************************************
_InitRXV23:
  dint
  LDP     #AGCGAIN      ; Initialisation of the page
      ZAC                    ; Initialisation of variables
  SACL    AGCGAIN     ; gain=0
  SACL AGCGAIN+1
      SACL    FSKCOUNT         ; count=0
      SACL    FSKC             ; c=0
      LACC    #MEMON        ;
      SACL    _FSKREG         ; mem=1
      LDP #CID_PAGE
      LRLK    AR3,V23IN  ; Initialisation of BPF upper band
  LARP AR3        ;
      RPTK    #BPLEN            ;
      SACL    *+            ;
      LRLK    AR3,ENDXN      ; Initialisation of i(n)
      RPTK    #LPLEN            ;
      SACL    *-            ;
  lac  #10
  ldp  #DecCOUNT
  sacl DecCOUNT   ; bit count>>1
      eint
      RET                    ;

**********************************************************
*  V23 reception Routine
*
* Two versions:  Low MIPS (2), CID executed in parallel
*    with other tasks
*    High MIPS (3.5)
*    LOWMIPS=1->Low MIPS version
*    LOWMIPS=0->High MIPS version
*
* Input: A/D sample from 8 kHz sample IT contained in
* FromAD
* Output: Demodulated octet in MSB of _FSKREG
*
* Routine modifies AR2,AR3
**********************************************************
RXV23:
  ldp  #FromAD
  lac  FromAD
      rsxm
      LDP     #CID_PAGE     ; Initialisation of the page
      SACL    V23IN
  LDP  #CID_PAGE
      LRLK    AR2,ENDIN     ; Compute Pass Band filter
  LARP AR2        ;
      MPYK    0             ;
      ZAC                   ;
      RPTK    #BPLEN            ;
      MACD    BandPasCoef,*-;
      APAC                  ;
      ldp #AGCSamp
      SACH    AGCSamp,1          ;
  call AgcFSK     ; AGC
  ldp  #AGCSamp
  lac  AGCSamp
  ldp  #TN
  sacl TN
   MPYK    0             ; Compute the dephasor filter
        ZAC                    ;
        LAC     TN2,12         ;
        LTD     TN1     ; tempo(n)=A1.t(n)+A2.t(n-1)+t(n-2)
        MPY     #2317      ;- A2.tempo(n-1) - A1.tempo(n-2)
        LTD     TN             ;
```

```
        MPY     #655           ;
        LTA     TEMPON2        ;
        MPY     #-655          ;
        LTD     TEMPON1        ;
        MPY     #-2317         ;
        APAC                   ;
        SACH    TEMPON1,4      ;
        LT      TEMPON1        ;
        MPY     TN             ;
        PAC                    ; x(n)=tempo(n)*t(n)
        SACH    XN             ;

        .if LOWMIPS
                               ;
        ldp #CID_PAGE
        LRLK    AR2,ENDXN      ;
        MPYK    0              ;
        ZAC                    ;
        RPTK    LPLEN             ;
        MACD    LPCoef,*- ;
        APAC                   ;
        CALL    TRIGGER        ;
        SACL    OUT2           ;
        BZ      B8             ;
        ldp #_FSKREG
        LAC     _FSKREG         ;
        OR      #MEMON         ;
        B       E8             ;
B8      ldp #_FSKREG
 LAC      _FSKREG        ;
        AND     #MEMOFF        ;
E8      SACL    _FSKREG         ;
        SSXM
        LRLK    AR3,OUT2       ;

 .else

        LRLK    AR2,ENDXN      ; Compute the low pass filter
 LARP AR2  ; Filter also performs an oversampling
        MPYK    0
              ; The sampling frequency is 3 times the original
        ZAC          ; one.
        RPTK    14             ;
        MAC     FirstArray,*- ;
        APAC                   ;
        ldp #CID_PAGE
        CALL    TRIGGER        ;
        SACL    OUT1           ;
        BZ      B6             ;
        ldp #_FSKREG
        LAC     _FSKREG         ;
        OR      #MEMON         ;
        B       E6             ;
B6      ldp #_FSKREG
 LAC      _FSKREG        ;
        AND     #MEMOFF        ;
E6      SACL    _FSKREG         ;
        LRLK    AR2,ENDXN      ;
        MPYK    0              ;
        ZAC                    ;
        RPTK    14             ;
        MAC     SecondArray,*-;
        APAC                   ;
        ldp #CID_PAGE
        CALL    TRIGGER        ;
        SACL    OUT2           ;
        BZ      B7             ;
        ldp #_FSKREG
        LAC     _FSKREG         ;
```

```
        OR      #MEMON        ;
        B       E7            ;
B7      ldp #_FSKREG
  LAC     _FSKREG        ;
        AND     #MEMOFF       ;
E7      SACL    _FSKREG        ;
        LRLK    AR2,ENDXN     ;
        MPYK    0             ;
        ZAC                   ;
        RPTK    14            ;
        MACD    ThirdArray,*- ;
        APAC                  ;
        ldp #CID_PAGE
        CALL    TRIGGER       ;
        SACL    OUT3          ;
        BZ      B8            ;
        ldp #_FSKREG
        LAC     _FSKREG        ;
        OR      #MEMON        ;
        B       E8            ;
B8      ldp #_FSKREG
  LAC     _FSKREG        ;
        AND     #MEMOFF       ;
E8      SACL    _FSKREG        ;
        LRLK    AR2,2
        LRLK    AR3,OUT1      ;for(i=0;i<3;i++) {

  .endif

; Decision Unit
  call FSKDec
        ssxm
        RET                   ;

TRIGGER:SACH    L             ;
        SACL    L+1           ;
        SSXM                  ;
        SUB     #-100         ;
        RSXM                  ;
        BGZ     B4            ;
        LAC     #0            ;
        RET                   ;
B4:     ZALH    L+0           ;
        ADDS    L+1           ;
        SUB     #100          ;
        BLZ     B5            ;
        LAC     #1            ;
        RET                   ;
B5 ldp   #_FSKREG
        BIT     _FSKREG,10     ; Test mem bit
        BBZ     B9            ; return this bit value
        LAC     #1            ;
        ldp   #CID_PAGE
        RET                   ;
B9      ZAC                   ;
        ldp   #CID_PAGE
        RET                   ;

;---------------------------------------
; AGC and carrier detection
; AGCSamp contains signal after passband filtering
;---------------------------------------
AgcFSK
    LDP     #_FSKREG
    LAC     _FSKREG        ;
    AND     #~OLDCARON    ;
    BIT     _FSKREG,14     ; OldCarrier=Carrier
    BBZ     CarOn          ;
    OR      #OLDCARON     ;
```

```
CarOn
    SACL    _FSKREG        ;

; Automatic Gain Control
    SOVM
    ssxm
    LAC     #2000H
    SACL    AGCTEMP
    LT      AGCSamp        ;load sample band pass filtered
    MPY     AGCGAIN       ; mult by GAIN
    PAC
    NORM    *  ;prevents overflow and saturates if required
    NORM    *             ;
    NORM    *             ;
    NORM    *
    NORM    *             ;
    SACH    AGCSamp        ;
    ADDH    AGCSamp        ;
    SACH    AGCSamp        ;
    ABS                   ;compute new GAIN value
    SUBH    AGCTEMP
    NEG
    SUB     AGCGAIN,14
    SACH    AGCTEMP
    ZALH    AGCGAIN
    ADDS    AGCGAIN+1
    ADD     AGCTEMP,8
    SACH    AGCGAIN
    SACL    AGCGAIN+1
    ROVM

; Carrier detection
    LAC     AGCGAIN          ;
    SUB     _AGCTHRESH        ; if(y<=Threshold1)
    BLEZ    C1            ; {
    LAC     _FSKREG
    AND     #~CARRIERON   ;   Carrier=0
    SACL    _FSKREG        ; }
    B       C2
C1  LAC     AGCGAIN          ; else if(y>=Threshold2)
    SUB     _AGCTHRESL       ;      {
    BGEZ    C3            ;       Carrier=1
    LAC     _FSKREG        ;
    OR      #CARRIERON    ;
    SACL    _FSKREG        ;      }
    B       C2            ;     else if(OldCarrier==0)
C3  BIT     _FSKREG,13     ;          {
    BBNZ    C4            ;
    LAC     _FSKREG        ;
    AND     #~CARRIERON   ;            Carrier=0
    SACL    _FSKREG        ;
;    B       EndCid        ;          }
    B       C2
C4  LAC     _FSKREG        ;          else {
    OR      #CARRIERON    ;                 Carrier=1
    SACL    _FSKREG        ;                }
C2  RET


;----------------------------------------
; Decision Unit
; 1200 bauds
; at 8 kHz -> 6.66... samples per bit
; sample counter at 20, incremented by 3
; Input: Demodulated bit contained in AR3
;----------------------------------------
FSKDec
     .if LOWMIPS
```

```
     .newblock
     ssxm
     LDP     #_FSKREG
     BIT     _FSKREG,11     ;  if(stop_char)
     BBZ     $0            ;   {
     LAC     FSKCOUNT         ;     count++;
     ADD     #3            ;
     SACL    FSKCOUNT         ;
     SUB     DecCOUNT,1           ;     if(count>=20)
     BLZ     $1            ;     { count=0,1,2;
     SACL    FSKCOUNT         ;
     LAC     _FSKREG        ;
     AND     #~STOPON      ;       stop_char=0;
     SACL    _FSKREG        ;     }
     B       $1            ;   }
$0:  MAR     *,AR3         ;   else
     LACC    *             ;   {
     BIT     _FSKREG,12     ;   if(!(out[i] || start_char))
     BBZ     $2           ;
     OR      #1            ;
$2   BNZ     $3            ;     {
     SACL    FSKCHAR          ;       char=0;
     LAC     FSKCOUNT         ;       count++;
     ADD     #3            ;
     SACL    FSKCOUNT         ;
     SUB     DecCOUNT          ;       if(count>=10)
     BLZ     $3            ;       {
     SACL    FSKCOUNT         ;         count=0;
     LAC     _FSKREG        ;         start_char=1;
     OR      #STARTON      ;
     SACL    _FSKREG        ;       }
$3:  BIT     _FSKREG,12     ;     }
     BBZ     $1            ;     if(start_char)
     LAC     FSKCOUNT         ;     {
     ADD     #3            ;       count++;
     SACL    FSKCOUNT         ;
     SUB     DecCOUNT,1           ;       if(count>=20)
     BLZ     $1            ;       {
     LAC     *             ;         acc=out[i];
     AND     #01h          ;         acc=acc&0x01;
     RPT     FSKC             ;         acc=acc<<(c+1);
     SFL                   ;
     SFR                  ;
     NOP                   ;
     OR      FSKCHAR          ;         acc=acc|char;
     SACL    FSKCHAR     ;    char=char|((out[i]&0x01)<<c);
     LAC     FSKC             ;         c++;
     ADD     #1            ;
     SACL    FSKC             ;
     SUB     #8            ;         if(c>=8)
     BLZ     $4            ;         {
     ZAC            ;
     SACL    FSKC             ;           c=0;
     LAC     _FSKREG        ;
     AND     #~STARTON     ;           start_char=0;
     OR      #STOPON       ;           stop_char=1;
     OR      #CHARON       ;           char_available=1
     AND     #00FFh        ;
     SACL    _FSKREG        ;
     LAC     FSKCHAR,8        ;
     OR      _FSKREG        ;
     SACL    _FSKREG        ;         }
$4:  LAC     #0            ;         count=0;
     ldp #FSKCOUNT
     SACL    FSKCOUNT         ;       }
$1:  RET

   .else
```

```
LOOP:   BIT     _FSKREG,11      ;   if(stop_char)
        BBZ     B1             ;   {
        LAC     FSKCOUNT          ;     count++;
        ADD     #1             ;
        SACL    FSKCOUNT          ;
;       SUB     #20            ;     if(count==20)
        SUB     #17 ;changed 20 to 17 to for stop detection
        BNZ     E1             ;     {
        LAC     #0             ;       count=0;
        SACL    FSKCOUNT          ;
        LAC     _FSKREG         ;
        AND     #STOPOFF       ;       stop_char=0;
        SACL    _FSKREG         ;     }
        B       E1             ;   }
B1:     MAR     *,AR3          ;   else
        LACC    *              ;   {
        BIT     _FSKREG,12      ;if(!(out[i] || start_char))
        BBZ     B14            ;
        OR      #1             ;
B14     BNZ     B2             ;     {
        SACL    FSKCHAR           ;       char=0;
        LAC     FSKCOUNT          ;       count++;
        ADD     #1             ;
        SACL    FSKCOUNT          ;
;       SUB     #10            ;       if(count==10)
        SUB     #14
                     ;changed 10 to 14 to ameliorate start detection
        BNZ     B2             ;       {
        SACL    FSKCOUNT          ;         count=0;
        LAC     _FSKREG         ;         start_char=1;
        OR      #STARTON       ;
        SACL    _FSKREG         ;       }
B2:     BIT     _FSKREG,12      ;     }
        BBZ     E1             ;     if(start_char)
        LAC     FSKCOUNT          ;      {
        ADD     #1             ;       count++;
        SACL    FSKCOUNT          ;
        SUB     #20            ;       if(count==20)
        BNZ     E1             ;       {
        LAC     *              ;         acc=out[i];
        AND     #01h           ;         acc=acc&0x01;
        RPT     FSKC              ;         acc=acc<<(c+1);
        SFL                    ;
        SFR                   ;
        NOP                    ;
        OR      FSKCHAR           ;         acc=acc|char;
        SACL    FSKCHAR        char=char|((out[i]&0x01)<<c);
        LAC     FSKC              ;         c++;
        ADD     #1             ;
        SACL    FSKC              ;
        SUB     #8             ;         if(c>=8)
        BLZ     B3             ;         {
        LAC     #0             ;
        SACL    FSKC              ;           c=0;
        LAC     _FSKREG         ;
        AND     #STARTOFF      ;           start_char=0;
        OR      #STOPON        ;           stop_char=1;
        OR      #CHARON        ;           char_available=1
        AND     #00FFh         ;
        SACL    _FSKREG         ;
        LAC     FSKCHAR,8         ;
        OR      _FSKREG         ;
        SACL    _FSKREG         ;         }
B3:     LAC     #0             ;         count=0;
        SACL    FSKCOUNT          ;       }
E1:     LARP    AR3            ;     }
        MAR     *+,AR2        ; increment ar3,next arp = ar2
        BANZ    LOOP           ; } // EndFor
        ssxm
```

```
        RET                     ;
  .endif
```

File: V23.inc

```
LOWMIPS .set 1
                     ;compiler switch:low/high MIPS version

V23ZONE .set 50h  ; dead zone for slicer

  .if   LOWMIPS
LPLEN   .set 18   ; LPF length high band
  .else
LPLEN   .set 14    ; LPF length oversampling filter
  .endif

BPLEN   .set 30   ; BPF length high band

  .global _AGCV23L, _AGCV23H
  .global V23IN, ENDIN
  .global BandPasCoef
  .global TXV23

***********************************************************
*³ Variables                                               *
***********************************************************
CID_PAGE          .usect "V23",0  ; Start of page jlma
TEMP2    .usect "V23",1   ;jlma
OUTCHAR          .usect "V23",1  ;
OUT_SAMPLE       .usect "V23",1  ;
TEMP             .usect "V23",1  ;
TEMP1            .usect "V23",1  ;
OUT1             .usect "V23",1  ;
OUT2             .usect "V23",1  ;
OUT3             .usect "V23",1  ;
Y                .usect "V23",1  ;
L                .usect "V23",2  ;
X                .usect "V23",1  ;
IN_SAMPLE        .usect "V23",1  ;
XN              .usect "V23",LPLEN
                              ;   low pass filter upper band
ENDXN            .usect "V23",2  ;
TEMPON1          .usect "V23",1  ;
TEMPON2          .usect "V23",1  ;
TN               .usect "V23",1  ;
TN1              .usect "V23",1  ;
TN2              .usect "V23",1  ;
V23IN            .usect "V23",BPLEN
                      ; band pass filter upper band
ENDIN            .usect "V23",2  ;
SAMPLC   .usect "V23",1 ; undersampling counter
_FSKREG  .usect  "FSK",1
_AGCTHRESL    .usect  "FSK",1 ;AGC threshold for hysteresis
_AGCTHRESH    .usect  "FSK",1
FSKCOUNT      .usect  "FSK",1
FSKCHAR       .usect  "FSK",1
FSKC        .usect  "FSK",1
AGCSamp     .usect  "FSK",1
AGCGAIN       .usect  "FSK",2
AGCTEMP .usect  "FSK",1
DecCOUNT .usect  "FSK",1

; Constant use for the bit manipulation in the V23REG register
CHARON           .set    0001h
CARRIERON        .set    0002h
OLDCARON         .set    0004h
STARTON          .set    0008h
```

```
STOPON          .set    0010h
MEMON           .set    0020h
        .text
BandPasCoef:
        .word   139
        .word   516       ;  Band Pass filter coefficients
        .word   -62           ;  quantitizied coeff Q15
        .word   -396      ;  Characteristic of this filter
        .word   150           ;  in file BPV232.FLT
        .word   -574
        .word   -1563
        .word    666
        .word   2397
        .word   306
        .word   482
        .word   1876
        .word   -4287
        .word   -8446
        .word   2744
        .word   12394
        .word   2744
        .word   -8446
        .word   -4287
        .word   1876
        .word   482
        .word   306
        .word   2397
        .word   666
        .word   -1563
        .word   -574
        .word   150
        .word   -396
        .word   -62
        .word   516
        .word   139

  .if LOWMIPS
; low pass filter high band
LPCoef
  .word   -250  ;C018
  .word   -958  ;C017
  .word  -1899  ;C016
  .word  -2624  ;C015
  .word  -2125  ;C014
  .word    307  ;C013
  .word   4721  ;C012
  .word  10015  ;C011
  .word  14379  ;C010
  .word  16064  ;C009
  .word  14379  ;C008
  .word  10015  ;C007
  .word   4721  ;C006
  .word    307  ;C005
  .word  -2125  ;C004
  .word  -2624  ;C003
  .word  -1899  ;C002
  .word   -958  ;C001
  .word   -250  ;C000
   .else
* Low Pass filter coefficients - perform an oversampling.
FirstArray:
        .word     -118  ; LpFilt[42]
        .word      -18  ; LpFilt[39]
        .word      270  ; LpFilt[36]
        .word      749  ; LpFilt[33]
        .word     1340  ; LpFilt[30]
        .word     1898  ; LpFilt[27]
        .word     2260  ; LpFilt[24]
        .word     2316  ; LpFilt[21]
```

```
        .word     2047  ; LpFilt[18]
        .word     1538  ; LpFilt[15]
        .word      940  ; LpFilt[12]
        .word      411  ; LpFilt[9]
        .word       55  ; LpFilt[6]
        .word     -103  ; LpFilt[3]
        .word     -411  ; LpFilt[0]
SecondArray:
        .word     -118  ; LpFilt[43]
        .word      -70  ; LpFilt[40]
        .word      151  ; LpFilt[37]
        .word      571  ; LpFilt[34]
        .word     1139  ; LpFilt[31]
        .word     1725  ; LpFilt[28]
        .word     2169  ; LpFilt[25]
        .word     2335  ; LpFilt[22]
        .word     2169  ; LpFilt[19]
        .word     1725  ; LpFilt[16]
        .word     1139  ; LpFilt[13]
        .word      571  ; LpFilt[10]
        .word      151  ; LpFilt[7]
        .word      -70  ; LpFilt[4]
        .word     -118  ; LpFilt[1]
ThirdArray:
        .word     -411  ; LpFilt[44]
        .word     -103  ; LpFilt[41]
        .word       55  ; LpFilt[38]
        .word      411  ; LpFilt[35]
        .word      940  ; LpFilt[32]
        .word     1538  ; LpFilt[29]
        .word     2047  ; LpFilt[26]
        .word     2316  ; LpFilt[23]
        .word     2260  ; LpFilt[20]
        .word     1898  ; LpFilt[17]
        .word     1340  ; LpFilt[14]
        .word      749  ; LpFilt[11]
        .word      270  ; LpFilt[8]
        .word      -18  ; LpFilt[5]
        .word     -118  ; LpFilt[2]
   .endif
```

Appendix B. Glossary

AGC	Automatic Gain Control
AS	Alerting Signal
CLIP	Calling Line Identity Presentation
DT-AS	Dual Tone-Alerting Signal
FIR	Finite Impulse Response
FSK	Frequency Shift Keying
IIR	Infinite Impulse Response
LE	Local Exchange
LPF	Low Pass Filter
PSTN	Public Switched Telephone Network
RP-AS	Ring Pulse-Alerting Signal
TAS	TE Alerting Signal
TE	Terminal Equipment
UART	Universal Asynchronous Receiver Transmitter

INTERNET

www.ti.com

Register with TI&ME to build custom information pages and receive new product updates automatically via email.

TI Semiconductor Home Page
http://www.ti.com/sc

TI Distributors
http://www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas

Phone	+1(972) 644-5580
Fax	+1(972) 480-7800
Email	sc-infomaster@ti.com

Europe, Middle East, and Africa

Phone	
Deutsch	+49-(0) 8161 80 3311
English	+44-(0) 1604 66 3399
Francais	+33-(0) 1-30 70 11 64
Italiano	+33-(0) 1-30 70 11 67
Fax	+33-(0) 1-30-70 10 32
Email	epic@ti.com

Japan

Phone	
International	+81-3-3457-0972
Domestic	+0120-81-0026
Fax	
International	+81-3-3457-1259
Domestic	+0120-81-0036
Email	pic-japan@ti.com

Asia

Phone	
International	+886-2-3786800
Domestic	
Australia	1-800-881-011

Asia (continued)

TI Number	-800-800-1450
China	10811
TI Number	-800-800-1450
Hong Kong	800-96-1111
TI Number	-800-800-1450
India	000-117
TI Number	-800-800-1450
Indonesia	001-801-10
TI Number	-800-800-1450
Korea	080-551-2804
Malaysia	1-800-800-011
TI Number	-800-800-1450
New Zealand	+000-911
TI Number	-800-800-1450
Philippines	105-11
TI Number	-800-800-1450
Singapore	800-0111-111
TI Number	-800-800-1450
Taiwan	080-006800
Thailand	0019-991-1111
TI Number	-800-800-1450

IMPORTANT NOTICE