ON- AND OFF-HOOK CALLER ID USING THE Z893XX

CALLER ID CAN BE EFFECTIVELY AND EFFICIENTLY IMPLEMENTED BY USING THE ON-CHIP RESOURCES OF THE Z893XX 16-BIT FIXED-POINT DSP

INTRODUCTION

This Application Note demonstrates how to implement Caller Identification (herein referred to as *Caller ID* or *CID*) as an added feature of the telephone system using the on-chip resources of the Z893XX. As part of the Caller ID demonstration, this Application Note also describes a multitasking system showing how an integrated microcontroller functions as a DSP controller.

Before presenting specific information on building a CID system, the following rudimentary concepts are first explained:

- Caller Identification
- Delivery Variations—On- and Off-Hook
- Use of DSP in CID Solutions

What Is Caller Identification?

Caller Identification (CID) is an added feature of the telephone system that gives a visual display of the calling party before connection. The display is typically a custom Liquid Crystal Display (LCD) with two, three, or four lines of information. A typical LCD Message output display appear as follows:

08:16AM 8/18 Call#1

123-456-789

John Doe

Before even picking up the phone, the CID feature easily identifies the caller, the caller's telephone number, and the time of the call. If the name and the number of the caller are not recognized (or response is not desirable), then the person receiving the call can let the phone ring merrily away, or perhaps allow the answering machine to pick up the call.

Both primary messaging formats are supported: Single Data Message Format (SDMF) and Multiple Data Message Format (MDMF). SDMF typically displays the date, time, and telephone number of the calling party. The MDMF typically displays the date, time, telephone number *and name*—as it would appear in a telephone book listing—of the calling party.

Note: The *Single and Multiple Data Message Formats* section that follows provides additional detail. See References 4 and 5 listed at the conclusion of this Application Note.)

Delivery Variations

The two primary methods of delivery variations are called *on-hook* and *off-hook*. With on-hook delivery, information is transmitted between the first and second rings of the incoming call. While relatively simple and cost-effective analog solutions are widely available for on-hook operation. DSP is a more appropriate solution for off-hook operation.

With off-hook delivery (also known as *Spontaneous Call Waiting with Caller Identification* [SCWID] or *Caller Identification with Call Waiting* [CICW], two parties are connected while a third party is attempting to connect with one of the these two parties. Information is only transmitted if an acknowledgment is received from the party to be interrupted. In addition to the various call waiting signals that are transmitted from the Stored Program Control System (SPCS), a special Customer Premise Equipment Alerting Signal (CAS) is also transmitted. The basic data is transmitted using continuous phase binary Frequency Shift Keying (FSK).

The following sections further compare and contrast on- and off-hook operation.

INTRODUCTION (Continued)

On-Hook Delivery

On-hook delivery in the form of an adjunct or add-on unit is the commercially available solution at this time. This simple type of system requires limited circuitry to perform a detection of the ringing current and then to demodulate the FSK signal and display the resulting data. Figure 1 shows the delivery of this FSK data sandwiched between the first and second rings. The larger amplitude, lower frequency, and waveforms at the beginning and end of the Captured Time Buffer (CAP_TIM_BUF) are the ringing pulses. Filtered Time 1 (FILT_TIME1) shows these ringing pulses in greater detail (see Figure 1). The smaller amplitude, higher frequency, and waveform is the FSK data.

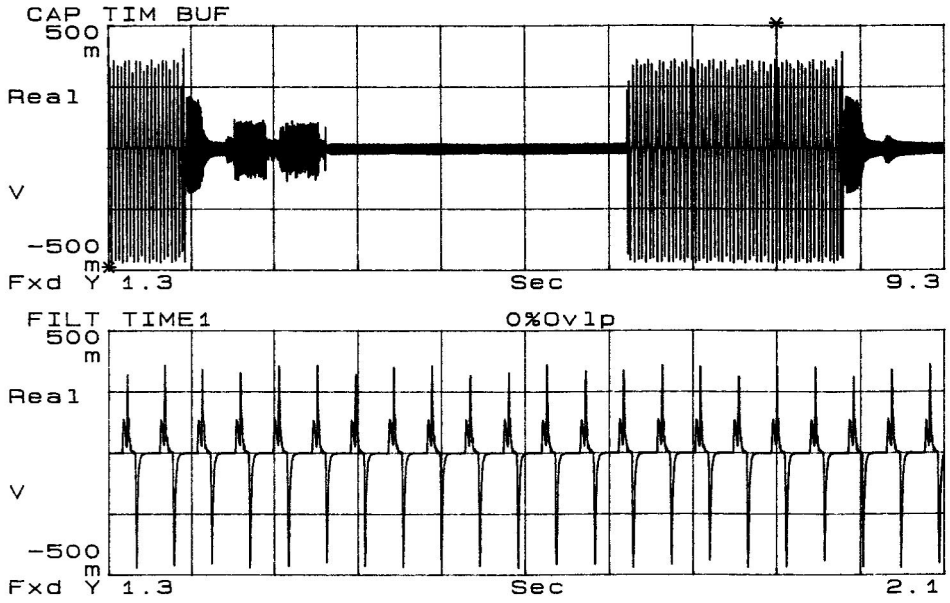

Figure 1. On-Hook FSK Delivery Between the First and Second Rings

To better understand the FSK function, a somewhat *idealized* simulation of the data is shown in Figure 2. The data being transmitted here is first a `1` then a `0` and then another `1` and finally a `0`. The Power Spectral Density plot shows the frequency content of this signal for those who prefer to view such signals in the frequency domain.

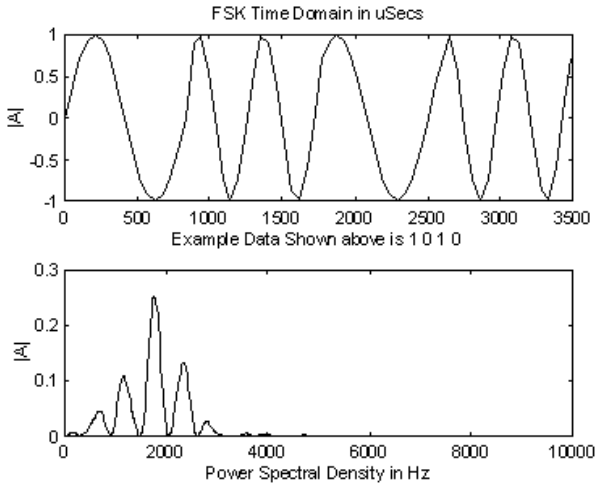

Figure 2. Idealized FSK Data and its Power Spectral Density

Of course, the data in the real world is not as clean as the idealized situation shown in Figure 2. A quick look at real data (refer to Figure 3) shows actual received data. It is easy to see that the amplitude of the high-frequency segment and that of the low-frequency segment are quite different. In addition, there is noise superimposed on the signal most noticeably on the peaks and troughs. (Refer to the *FSK Demodulation* section for a more detailed discussion.)

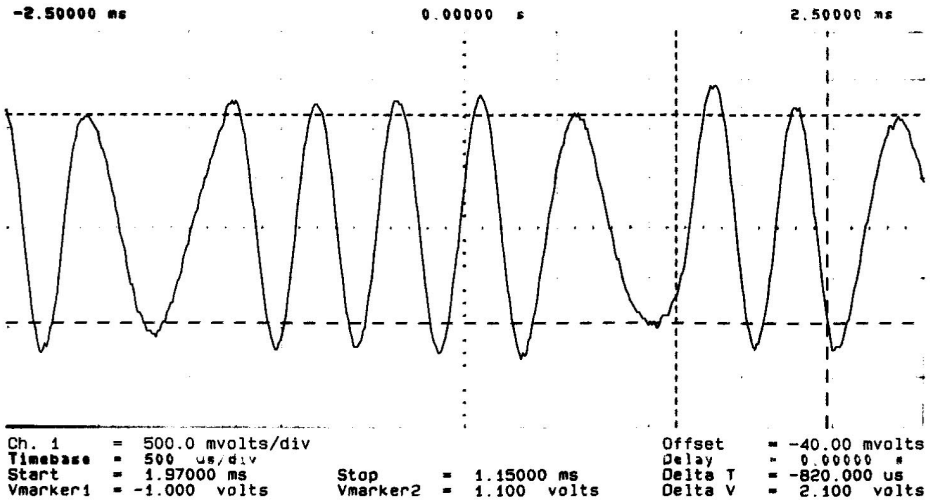

Figure 3. Real FSK Data

Single and Multiple Data Message Formats

Within the FSK data itself, there is structure for Single Data Message Format (SDMF) and Multiple Data Message Format (MDMF). While SDMF allows only the date, time, and phone number of the caller to be transmitted, MDMF allows the caller's name to be transmitted also. MDMF, in fact, al-

lows the transfer of almost any type of data that can be represented in ASCII.

Figure 4 shows an overview of MDMF. The channel seizure is a series of alternating `1` and `0`, which is only supplied in the on-hook case. In the off-hook case, data transmission starts with the mark signal, which is a series of 1's. The parameter words are not limited to one message. There may be many parameter messages and each parameter message consists of a parameter type, a parameter length, and a parameter word. Optional mark signals may be transmitted between frames. A checksum is at the end of every transmission.

Note: The parameter data can be variable in length.

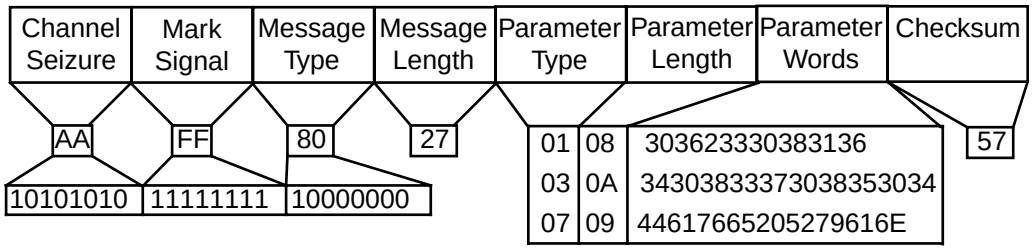

Figure 4. Overview of Messaging Structure

On-Hook Decoder Block Diagram

The simple block diagram shown in Figure 5 further explains the on-hook solution. An FSK band-pass filter is included to filter out-of-band signals. The FSK demodulator converts the analog signal into binary data.

The SDMF/MDMF section removes the start and stop bits and determines the messaging format. The data is then either stored in static RAM or displayed on the LCD.

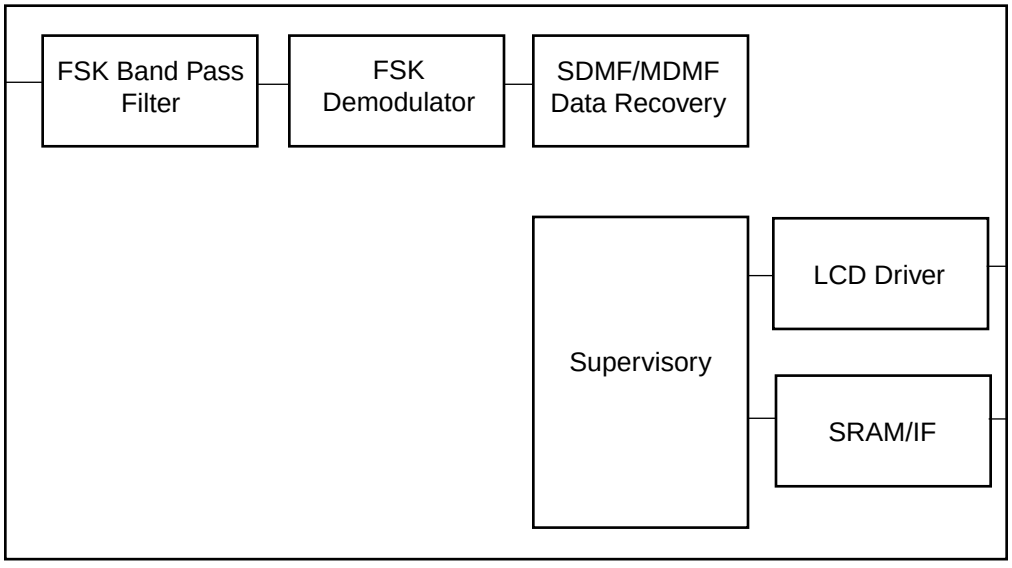

Figure 5. On-Hook Functional DSP Software Modules

LCD Display

The display is usually a small Liquid Crystal Display (LCD) capable of displaying the date, time, telephone number, and caller's name. The person being called simply waits until the second ring, allowing time for the data to be received. Usually there is some memory in these units so messages can be stored for later retrieval.

The amount of memory varies, but usually between 30 and 99 messages can be stored. These systems are normally, but not always, battery-powered as the actual use time of system operation is generally limited to the time between the first and second rings. Once the call has been answered, the system may be put in power-down or standby mode.

What Are the Advantages of DSP?

For on-hook operation, simple and cost-effective analog solutions exist; therefore, DSP is not required. But for off-hook operation, DSP is the more sensible choice. The difficulty arises in the accurate detection of the special CAS tone in the presence of VOX. Two separate issues exist: 1) inadvertent detection due to the similarity with speech (the so called *Talk-Off* problem) and 2) failure to detect (the so called *Talk Down* problem). This type of system has not been widely implemented using analog solutions primarily because of the difficulties involved in implementing a cost-effective, manufacturable, and robust solution using analog technology.

In contrast, by using digital filters, the manufacturing difficulties associated with using critically matched components (resistors, capacitors, inductors, and the like) are largely obviated. In addition, the digital solution may now be made to be adaptive, which is a major bonus. Further, implementations that are variants for use in other areas of the world are now simply a matter of software upgrades. Of course, there are some trade-offs. Analog-to-digital (A-D) conversion must be supported with its ancillary requirements. Of course, digital-to-analog (D-A) conversion must also be supported; however, on balance the DSP solution is far superior to the analog solution.

BUILDING A CALLER IDENTIFICATION SYSTEM

You can build your own Caller Identification (CID) system using the information provided in this Application Note. There are many ways to do this, but perhaps the simplest solution is to purchase a ready-made evaluation board complete with firmware. You can also write the software and build the hardware as described here. While building the hardware is reasonably straightforward (see Figures 6 and and 7), software development is more complex.

You would need to purchase firmware development tools such as an emulator, assembler, linker, and debugger. This Application Note assumes that the reader is content to adopt the firmware provided in a ready-made evaluation board. Detailed descriptions of the hardware and firmware follow.

Hardware and On-Hook Operation

Figure 6 illustrates the Z893XX DSP Connection Block Diagram; Figure 7 illustrates the Caller ID Demo Board Schematic Diagram.

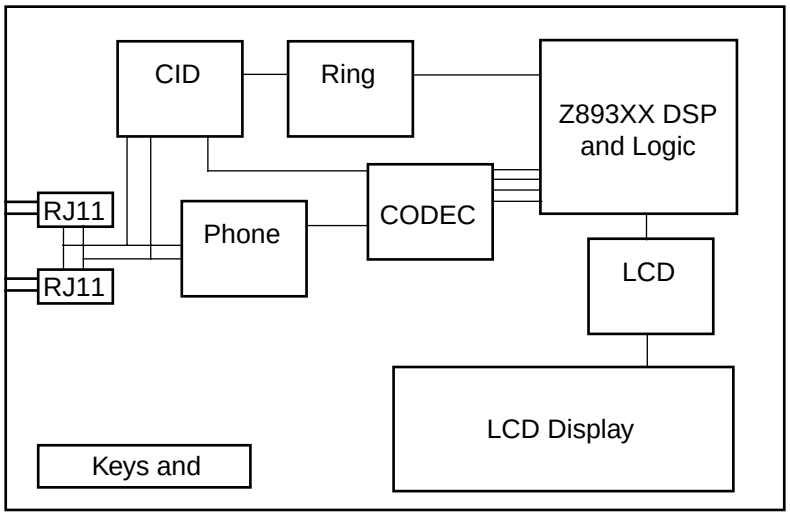

Figure 6. Z893XX DSP Connect Block Diagram

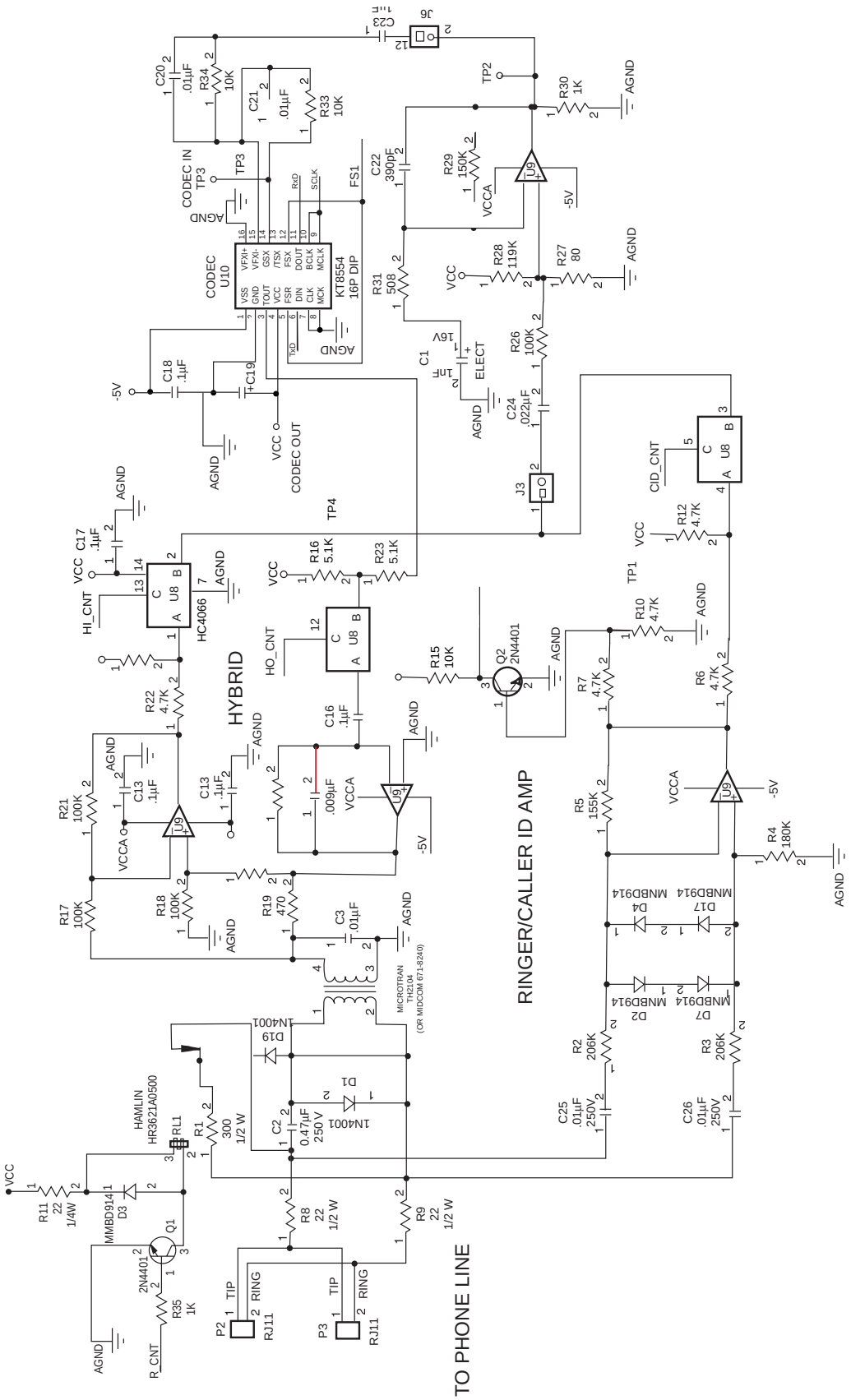

Figure 7. Caller ID Demo Board Schematic—Part I

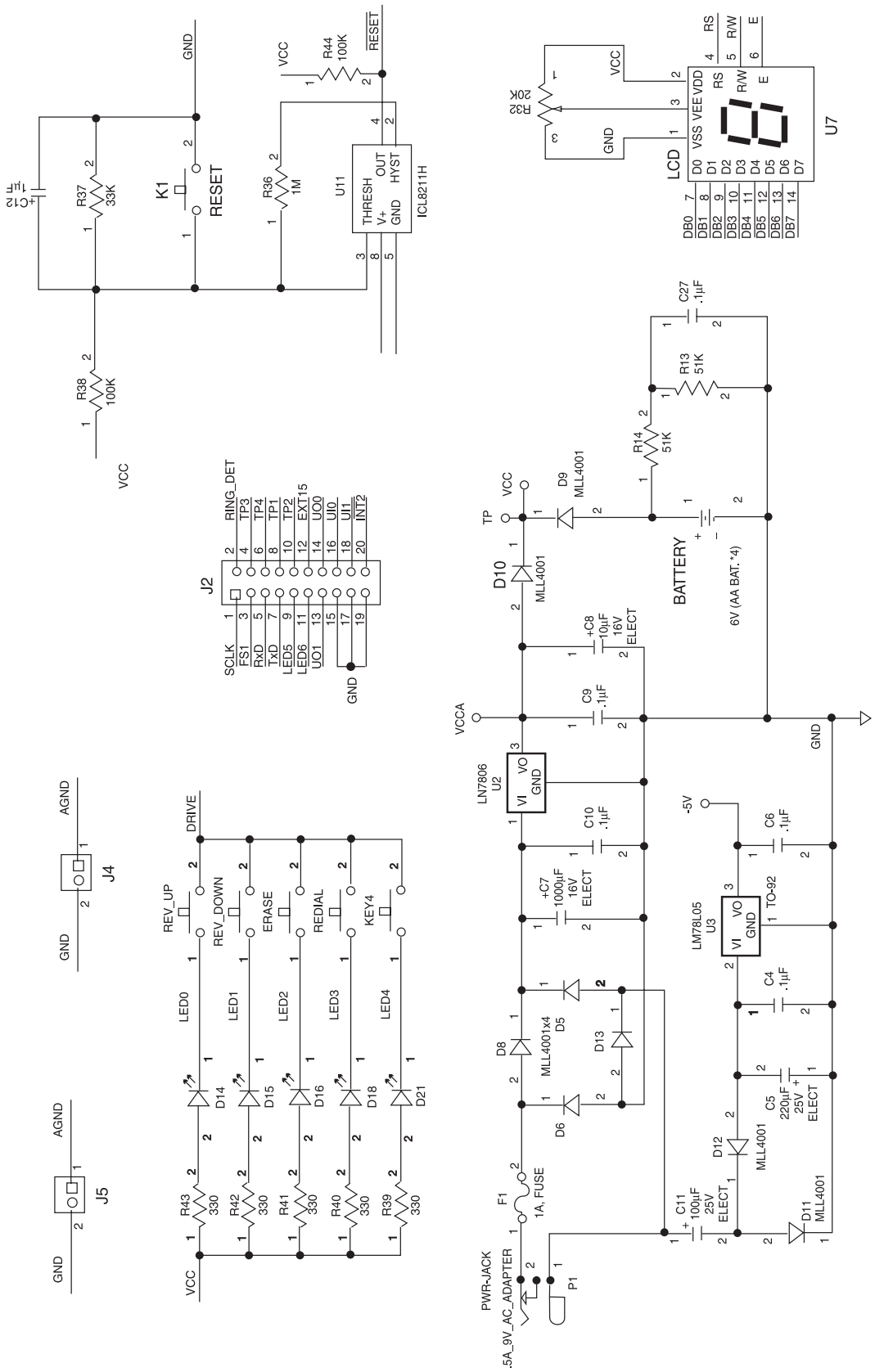

Figure 8. Caller ID Demo Board Schematic—Part II

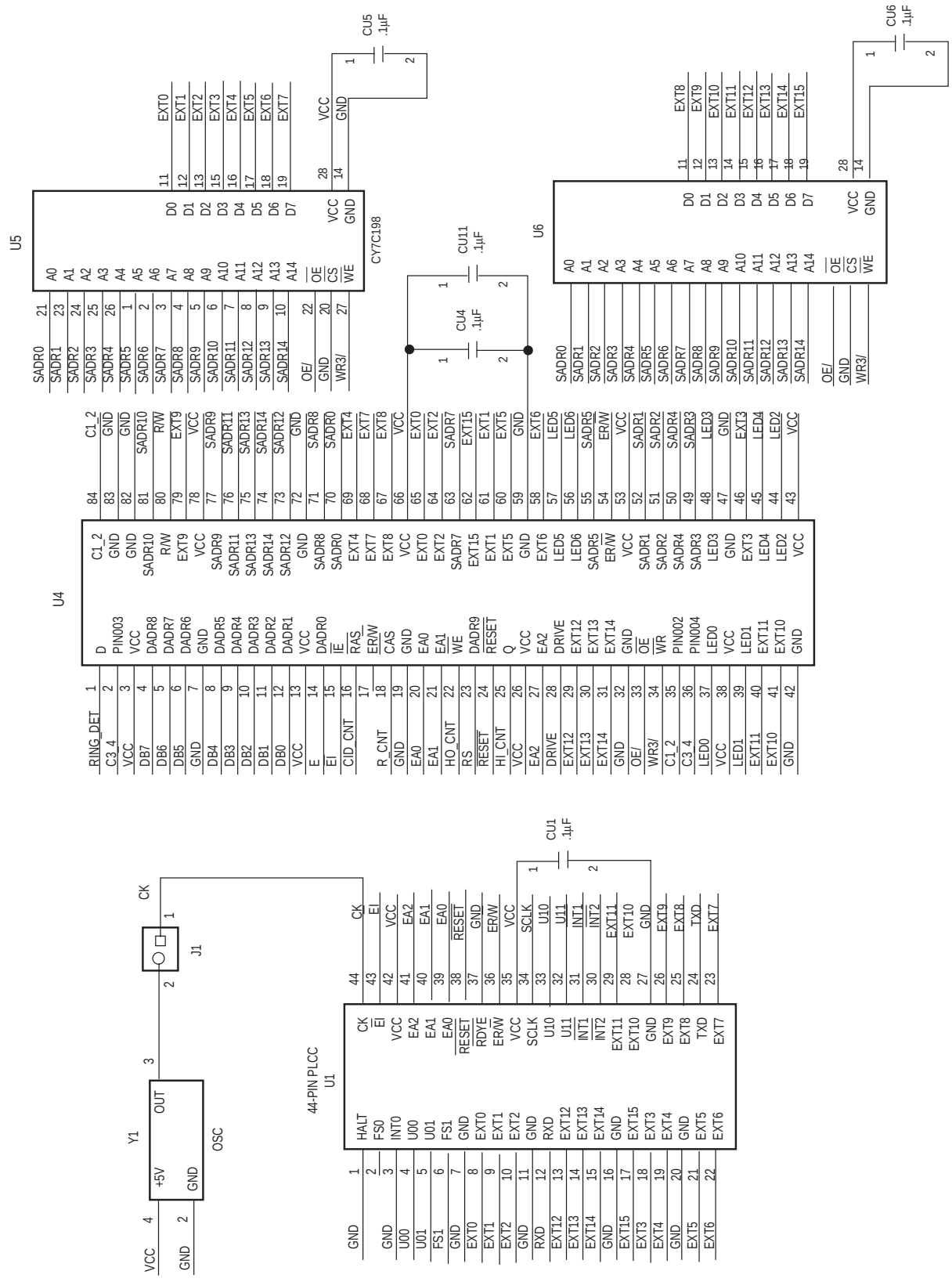

Figure 9. Caller ID Demo Board Schematic—Part III

CID Blocks

Phone Line Interface. Includes the transformer and additional components used to isolate the CID circuits from the line. Protection is needed since the ring high- voltage signal can damage the CID circuits and the on-/off- hook relay.

Ring Detect Circuit. Gives digital input (RING_DET signal) to the DSP indicating presence of rings on the phone line.

Caller ID Gain Control. Controlling the gain of the signals coming from the phone line interface to the codec analog input. The DSP enables this path by CID_CNT control signal.

Codec. The DSP analog front end. The codec data format is 8-bit PCM µ_law. The DSP controls the sampling rate by the FS1 signal and serial shifts by SCLK signal. The DSP receives serial data from RXD and transmits serial data to TXD.

Hybrid. The DSP sends the CAS acknowledge via the codec and the Hybrid back to the phone line interface. The DSP enables this path by using the HO_CNT control signal.

On-Hook Operation

The operation is relatively simple and operation should be possible in any area where CID is available and subscribed to.

1. Connect jack P2 to the telephone line or to a CID simulator, if available.
2. Connect jack P3 to the telephone.
3. Apply power.
4. Press the RESET button. The LCD displays READY.
5. Connect a scope to TP3 in order to check the FSK levels. The amplitude of the FSK signal is approximately 3V, peak-to-peak, when the FSK data is received between the first and second rings. R31 may be adjusted if necessary.
6. The LCD now displays the results. If the message is SDMF format, then the display shows the date, time, and telephone number of the calling party. If the message is MDMF format, the display shows the date, time, telephone number, *and name*.

Off-Hook Data

Figure 8 illustrates the actual delivery of FSK data when operating in the off-hook mode. (In the simple on-hook mode, the FSK data was delivered between the first and second rings.) In Off-Hook mode, the data may arrive at any time a call waiting may occur. The larger amplitude, lower frequency waveforms at the beginning of the CAP Captured Time Buffer (CAP_TIM_BUF) are the call waiting and CAS tone. The FSK data is seen after a gap. It is during this gap that an acknowledgment (ACK) is generated by the DSP. This ACK is not shown as the DSO was connected to the receive side. Filtered Time 1 (FILT_TIME1) shows the call waiting and CAS tone in greater detail.

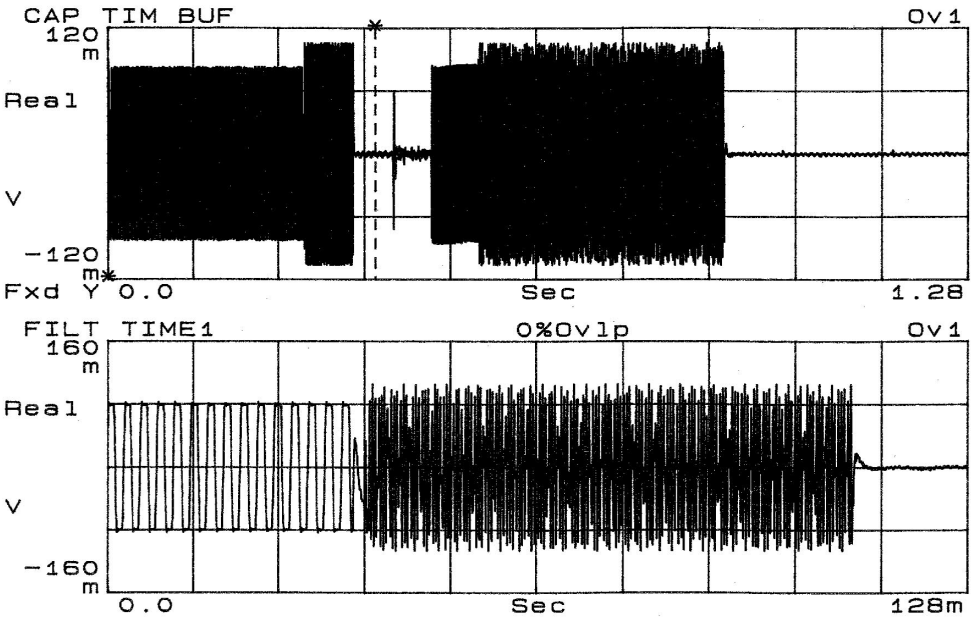

Figure 10. Off-Hook Delivery of FSK

BUILDING A CALLER IDENTIFICATION SYSTEM (Continued)

Off-Hook Decoder Block Diagram

When the on-hook solution is compared to off-hook solution, a number of differences are apparent. In addition to the modules used in the on-hook, there is a CAS filter for both the high- and low-band portions of the CAS signal. There is also the special CAS detector timing (see *CAS Detection* section, which follows). Once this special CAS tone is detected, then an acknowledgment must be sent back so a DTMF generator module is also included. In addition, whether the system is on-hook or off-hook must now be determined.

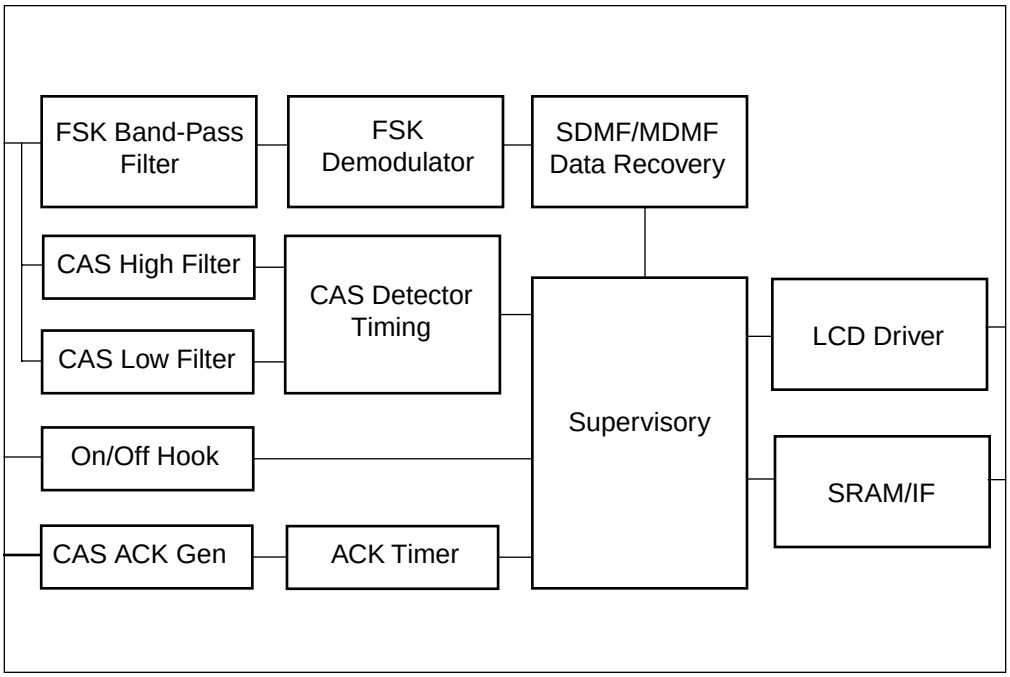

Figure 11. Off-Hook Functional Software Modules of DSP

Off-Hook Operation

The operation in the Off-Hook mode is not as simple as On-Hook due to the extra communication involved. Again, operation should be possible in any area where Caller ID with Call Waiting (CIDCW) is available and subscribed to.

1. Connect jack P2 to line 2 or to a CIDCW simulator if available.

Note: A simulator is far preferable and mandatory if additional development is planned because a minimum of three lines is involved.

2. Line 1 calls line 2 initially.
3. Line 3 interrupts line 2.
4. Connect a scope to TP3 to check the FSK levels. The amplitude of the FSK signal should be approximately 3V, peak-to-peak, when the FSK data is being transmitted.
5. Adjust R31, if necessary.

The LCD then displays the results. If you listen on line 2, you should hear a call waiting tone followed by the special CAS tone. The CAS tone should be detected by the module and an ACK should be sent followed immediately by data transmission from the SPCS or simulator. If the message is SDMF format, then the display should show the date, time, and telephone number of the calling party. If the message is MDMF format, the display should show the date, time, telephone number, *and name*.

FSK Demodulation

A continuous phase binary Frequency Shift Keying (FSK) demodulation function is integrated into the DSP. The FSK frequencies are 1200 ±12 Hz and 2200 ±22 Hz. After all protocol and handshaking requirements are completed, the basic data is transmitted using FSK. There are no phase discontinuities and there are only two frequencies involved in the FSK signal. The lower frequency (1200 Hz) represents a mark (logic 1), and the higher frequency (2200 Hz) represents a space (logic 0). For example, if a continuous stream of 0–1–0–1 is sent, then a full cycle of 1200 Hz is sent (833 µs) and then approximately one and half cycles of 2200 Hz is sent (833 µs). Again, a full cycle of 1200 Hz is sent (833 µs) and then approximately one and half cycles of 2200 Hz is sent (833 µs).

There is no parity or error checking beyond checking a checksum, which is sent at the end of the transmission. There is, however, a start bit (0) and a stop bit (1) added to each 8-bit transmitted word. The transmission rate is 1200 baud (bits per sec.). The demodulation is very similar to that employed in a standard low baud rate V.21/Bell103 modem. (Refer to Reference 6 at the conclusion of this Application Note.)

Data Recovery

The data format is best understood by examination of the example in Table 1, which follows. In this typical example, the overall message type is %80 or 128, which means that this is MDMF format. The data is sent in the format of Parameter words, where the parameter type and length are transmitted in simple binary and the calling name and number information are transmitted using regular ASCII.

The last word of all Single or Multiple Data Message frames is the checksum. The checksum is the two's complement of the modulo 256 sum of the binary representation of all the other words in the message including the message type and length as well as the parameter type and length. When making this calculation, remove the start and stop bit. And of course, the binary sum, hex sum, and decimal sums are equivalent. To obtain the two’s complement a simple approach would be to XOR with %FF. Using the data in the example in Table 1, and then using hexadecimal calculations the checksum is:

CS = XOR
(MOD(sum(80,27,01,08, ... 20,52,79,61,6E), 100), FF)

CS = XOR (MOD(7A8, 100), FF)

CS = XOR (A8, FF)

CS =%57

Of course, in the practical application, the actual calculation is actually much less cumbersome due to the natural modulo 256 nature of a byte. Since there is no error correction, the practical application of the checksum is to simply compare the received checksum with the calculated checksum. If they do not agree, then the data is bad in some way and should, in general, not be displayed.

BUILDING A CALLER IDENTIFICATION SYSTEM (Continued)

Table 1. Typical Multiple Message Data Format

Bit (MSB-LSB)	ASCII/HEX%	Stop	7	6	5	4	3	2	1	0	Start
Message Type	/80	1	1	0	0	0	0	0	0	0	0
Message Length	/27	1	0	0	1	0	0	1	1	1	0
Parameter Type	/1	1	0	0	0	0	0	0	0	1	0
Parameter Length	/8	1	0	0	0	0	1	0	0	0	0
Month	0/30	1	0	0	1	1	0	0	0	0	0
	6/36	1	0	0	1	1	0	1	1	0	0
Day	2/32	1	0	0	1	1	0	0	1	0	0
	3/33	1	0	0	1	1	0	0	1	1	0
Hour	0/30	1	0	0	1	1	0	0	0	0	0
	8/38	1	0	0	1	1	1	0	0	0	0
Minute	1/31	1	0	0	1	1	0	0	0	1	0
	6/36	1	0	0	1	1	0	1	1	0	0
Parameter Type	/3	1	0	0	0	0	0	0	1	1	0
Parameter Length	/A	1	0	0	0	0	1	0	1	0	0
DN 4083708504	4/34	1	0	0	1	1	0	1	0	0	0
	0/30	1	0	0	1	1	0	0	0	0	0
	8/38	1	0	0	1	1	1	0	0	0	0
	3/33	1	0	0	1	1	0	0	1	1	0
	7/37	1	0	0	1	1	0	1	1	1	0
	0/30	1	0	0	1	1	0	0	0	0	0
	8/38	1	0	0	1	1	1	0	0	0	0
	5/35	1	0	0	1	1	0	1	0	1	0
	0/30	1	0	0	1	1	0	0	0	0	0
	4/34	1	0	0	1	1	0	1	0	0	0
Parameter Type	/7	1	0	0	0	0	0	1	1	1	0
Parameter Length	/9	1	0	0	0	0	1	0	0	1	0
CN John Doe	D/44	1	0	1	0	0	0	1	0	0	0
	a/61	1	0	1	1	0	0	0	0	1	0
	v/76	1	0	1	1	1	0	1	1	0	0
	e/65	1	0	1	1	0	0	1	0	1	0
	Space/20	1	0	0	1	0	0	0	0	0	0
	R/52	1	0	1	0	1	0	0	1	0	0
	y/79	1	0	1	1	1	1	0	0	1	0
	a/61	1	0	1	1	0	0	0	0	1	0
	n/6E	1	0	1	1	0	1	1	1	0	0
Checksum	/57	1	0	1	0	1	0	1	1	1	0

CAS Detection

A Consumer Premise Equipment Alerting Signal (CAS) detection function is integrated into the DSP. The basic premise for the CAS detector is the periodic nature of the CAS tones contrasted to the aperiodic nature of voiced VOX. The CAS frequencies are 2130 ±5% Hz and 2750 ±5% Hz, which means that the CAS is itself a Dual Tone Multi Frequency (DTMF) signal. However, the CAS frequencies are quite distinctly beyond the range of normal signaling DTMF frequencies. The signal is first filtered using CAS high-filter 2750 ±5% Hz and also CAS low-filter 2130 ±5% Hz. The resultant outputs are then rectified and tested for minimum amplitude requirements. If minimum amplitude requirements are met for both frequencies, then a timer is started to check for possible CAS duration. The amplitude must constantly exceed minimum requirements for a period to exceed a predetermined gating limit in order to determine a detection.

The CAS detector Interrupt Service Routine (ISR) services the CAS detection as shown in the source code listing. (This

is the portion of the listing that begins with `casint:` and ends with `ret;return from Codec ISR`. Refer to the *Source Code* section at the conclusion of this Application Note.) This portion of the code first saves the accumulator and status register. Then the data is retrieved from the codec. The codec register ext6 is double-buffered, which simply means that there are really two ext6 registers: ext6-0 and ext6-1. The assembler only reads ext6, which apparently is why there is redundant load of ext6 to push the data out. Next the filters are called. Since the basic biquad structure is used, then the filters are called three times to give a net sixth-order filter. This whole process is repeated twice: once for the F1 CAS high filter 2750 ±5% Hz and once for the F2 CAS low filter 2130 ±5% Hz band. The final portion of the casint ISR restores the accumulator and status register.

The basic biquad structure performs the core filtering as shown in the source code listing. (This is the portion of the listing that begins with `biquad:` and ends with `ret ;return from biquads`.) The code has been structured so that the tap updates and actual filter calculations are performed within the subroutine biquad. The first section of this code updates the input taps or delays in time. The newest sample of data X(n) or data at the time n replaces the older X(n), which in turn is saved as X(n–1) or the current sample delayed by 1, that is (n–1). This process is repeated for all taps. To perform filter computations (coeff * sample) auto increment is used and a single-cycle MPYA (MultiPlY and Accumulate) instruction is used.

The fundamental equations are

$$Y(n) = b0*X(n) + B1*X(n–1) + b2*X(n–2)$$
$$+a1*Y(n–1) + a2*Y(n–2)$$

where:

bi and ai = filter coefficients
X(n) = current sample
X(n–1) = previous sample
X(n-2) = second previous sample
Y(n) = current output
Y(n–1) = previous output
Y(n-2) = second previous output

The last section of this code updates the output taps or delays in time. The newest output Y(n) or output data at the time n replaces the older Y(n), which in turn is saved as Y(n–1) or the current output delayed by 1, that is (n–1). Y(n–1) is itself saved as y(n-2). The process is repeated for all output taps.

Display Drivers. The display used here is a standard off-the-shelf dot matrix LCD with 16-character x 4-line display. It is logically organized with 2 lines of 32 characters each that *overrun*; however, the physical manifestation is 4 lines. Any ASCII character can be displayed along with many other characters. As is commonly the case with these devices, the low-level drivers and controller are actually mounted on the LCD module. All that is needed is a relatively simple high-level software driver to instruct the LCD which character to display and where to place it. The DSP *bit bangs* over the ASCII data to the LCD controller using the external data bus of the DSP. The LCD is a relatively slow device, far slower than normal DSP operations, so updating the LCD presents minimal overhead to the DSP. Communication is done using a specialized series of LCD instructions. Once the LCD is initialized, the data is simply transmitted.

Call Progress. Call progress is essentially sequential in this system. A ring must be detected first before anything can happen. Once the call is established, only one of two events can happen: either a call interrupt occurs or does not occur. Data comes in only two flavors: SDMF or MDMF. After the data is received, then it obviously should be displayed. A simple state machine takes care of the logical progress of the call. This is the adoption of a microcontroller function within the DSP engine. At the end of the call, a disconnection occurs and the entire cycle repeats.

Memory Storage. Because of the multitasking features capability common to all DSP processors, normal call progress—especially on-hook/off-hook monitoring, system supervision, memory for calls received, and display tasks—can all be handled by a single DSP processor. For the sake of simplicity, memory storage has not been added to this demonstration. The external bus addressing capability will allow for this feature to be easily added.

How Available Is Caller Identification?

There is no real technical limitation to national implementation of Caller Identification. Other issues such as right to privacy are obviously beyond the scope of this article. Currently the service is available in most, but not all, U.S. states.

Conclusion

This Application Note has described only the most elementary features of a Caller Identification system. Many other value-added features can be added, such as *ring only on certain callers*. In this example, the ringer might be suppressed and only activated after the caller identity is recognized. This type of feature and other features could be easily added as a part of the controller functions. Future feature expansion is limited only by the user's imagination.

BUILDING A CALLER IDENTIFICATION SYSTEM (Continued)

References

1. Bellcore, Technical Reference TR-NWT-000031
2. Bellcore, Technical Reference TR-NWT-001188
3. Bellcore, Generic Requirements GR-30-CORE
4. Bellcore, Special Report SR-TSV-002578
5. Bellcore, Special Report SR-TSV-002476
6. Gibson, Principles of Digital and Analog Communications

SOURCE CODE

```
casint:
;---------------------------------------------------------------------
;  SAVE Acc and SR Contents for ISR return
;---------------------------------------------------------------------
         pusha               ;save acc
     ld      a,sr            ;load acc with sr
     push    a               ;save sr

;---------------------------------------------------------------------
;  F1 Sixth-order triple biquad IIR filter
;---------------------------------------------------------------------
                     lda,#X11;load Acc with address (X11) for input samples
                     ldp0:0,a;point to input sample
                     lda,#BF1;address (BF1) for filter coefficients
                     ldp0:1,a;point to filter coefficients

;read new input sample  x(n)
                     ldext6,a;push new µ-law input sample ext6-2 to ext6-1
                     lda,ext6;load µ-law data to accumulator x(n)
                        µ-law;µ-law result is a 14-bit sign magnitude number
                         slla;shift left logical/multiply by 2
                         slla;shift left logical/multiply by 2
                        ldx,a;x = new data
                   ldtempist,a;store temporary input storage
                   call biquad;perform standard biquad
                   call biquad;perform standard biquad
                   call biquad;perform standard biquad

; save output filter response y(n)
                   ldoutgbf1,a;store third-stage output
                      ldext6,a;CODEC1 output y(n) auto hardware µ-law
;---------------------------------------------------------------------
;  F2 Sixth-order triple biquad IIR filter
;---------------------------------------------------------------------
                     lda,#X41;load Acc with address (X41) for input samples
                     ldp0:0,a;point to input sample
                     lda,#BF2;address (BF2) for filter coefficients
                     ldp0:1,a;point to filter coefficients

;read new input sample  x(n)
                   lda,tempist;restore temporary input storage
                   ld      x,a;x = new data

                   call biquad;perform standard biquad
                   call biquad;perform standard biquad
```

SOURCE CODE (Continued)

```
                  call biquad;perform standard biquad

; save output filter response y(n)
                 ldoutgbf2,a ;store third-stage output
                adda,outgbf1 ;debug test
                     ldext6,a;CODEC1 output y(n) auto hardware µ-law
endinti:
;-------------------------------------------------------------------------
;  RESTORE Acc and SR Contents for ISR return
;-------------------------------------------------------------------------
          popa               ;restore sr
           ldsr,a            ;load sr with acc
          popa               ;restore old value of acc

                         SIEF;Enable interrupts

                          ret;Return from CODEC ISR

biquad:
;-------------------------------------------------------------------------
;perform filter computations (coeff * sample) using auto increment
;y = b0*X(n) + B1*X(n–1) + b2*X(n-2)
;      +a1*Y(n–1) + a2*Y(n-2)
;Input New Sample is in x
;Output is in Acc
;P0:0 Points at Input/Output Samples   X(n) X(n-1) X(n-2) Y(n-1) Y(n-2)
;P0:1 Points at Coefficients             b0   b1     b2    -a1    -a2
;-------------------------------------------------------------------------
; update input sample buffer
                    ldy,@p0:0;y saves old (n)  p0:0 points at (n)
                  ld@p0:0+,x ;(n) = new sample p0:0 points at (n)
                    ldx,@p0:0;x saves old (n-1)  p0:0 points at (n-1)
                  ld@p0:0+,y ;(n-1) = (n)
                    ldy,@p0:0;y saves old (n-2)  p0:0 points at (n-2)
                   ld@p0:0,x ;(n-2) = (n-1)
                     lda,p0:0;p0:0 points at (n-2)
                     suba,#%2;decrement acc
                     ldp0:0,a;p0:0 points at (n)
;y = b0*X(n) + B1*X(n-1) + b2*X(n-2)
;      +a1*Y(n-1) + a2*Y(n-2)
     mld     @p0:1+,@p0:0+,on;A= 0 P = (b0 * X11) X11 = X(n)
     mpya    @p0:1+,@p0:0+,on;b1 * X12            X12 = X(n-1)
     mpya    @p0:1+,@p0:0+,on;b2 * X13            X13 = X(n-2)
     mpya    @p0:1+,@p0:0+,on;a1 * Y11            Y11 = Y(n-1)
```

```
     mpya    @p0:1+,@p0:0+,on;a2 * Y12            Y12 = Y(n-2)
                        adda,p;add result of last multiply to acc
                          slla;scale back if divide by 2 on coefficients
                         ldx,a;return result in x

;update output buffer
                       lda,p0:0;p0:0 points at Y12+1
                       suba,#%2;decrement acc
                       ldp0:0,a;p0:0 points at Y11 (n-1)
                      ldy,@p0:0;y saves old (n-1)
                    ld@p0:0+,x ;Y11 = new result
                    ld@p0:0+,y ;Y12 = Y11 = Y(N-2)

; output filter response y(n)
                    ld      a,x;store stage output
                            ret;Return from biquads
```