FEATURE ARTICLE

Richard Newman

Caller ID Fundamentals

As Caller ID has become popular, people want to know how to make the data available to computing devices. This introduction describes how information is transmitted, formatted, and decoded.

Caller ID service as provided by the regional telephone companies has been described by telephony types as being *the* application which will enable small home and office Windows applications.

While Caller ID boxes are available for about $19 from national discount chains, there is not a single inexpensive Caller ID interface for the PC. Most implementations are multiline or add storage features which increase the price significantly.

In this article, I present an inexpensive, straightforward, and simple Caller ID decoder. You can connect a telephone line to one side of it and out the other to get standard serial data just as if it were coming out of an off-the-shelf modem.

Since data is delivered serially, you can handle it with a standard modem program set to hex decode mode or with a custom program that decodes the data into uniformly formatted fields.

For this project, I'm using the Motorola MC145447P Calling Line Identification (CID) Receiver with Ring Detector and colorburst crystal FOX036S. The Motorola CID chip is inexpensive ($2.60) and has integrated ring detection. You don't need any external circuitry to determine when a call is arriving.

THE BASICS

Caller ID data is transmitted from your local telephone company office to your telephone line directly after the first ring. During this time, your telephone line is on hook, and there is no DC current flow.

The data is transmitted onto the high-impedance line, which is only AC terminated by the phone's bell in your home.

Since the telephone company's equipment is expecting to see a high-impedance state on your phone line, the interface of the Caller ID receiver must pick the AC audio signal off the telephone line without terminating the line and answering the inbound call.

All telephone company exchanges operate slightly differently because of the make of the physical equipment and version of the software running on the switch. It is therefore possible for unique incompatibilities to surface.

For example, information is transmitted right after the first ring and is complete before the second ring starts. If you answer the telephone after the first ring, you might still receive the Caller ID data. However, if you answer during the first ring, the exchange usually aborts the transmission of the CID data, losing the information for the call.

CIRCUIT DETAILS

The Motorola MC145447 CID chip has an analog front end, which interfaces to the telephone line with two 47-µF, 200-V nonpolarized capacitors in series with two 10-kΩ resistors.

The input to the CID chip is differential. Because of this, it attempts to decode any differential (AC) voltages seen on the line, which results in occasional periods of unintelligible garbage.

The data transmitted from the telephone company is in standard Bell 202 format, similar to the format used by the old 1200-bps modems we all had a few years ago (Bell 212). The data is transmitted at 1200 bps with 8 data bits, no parity checking, and 1 stop bit. It is asynchronous, serial, and binary.

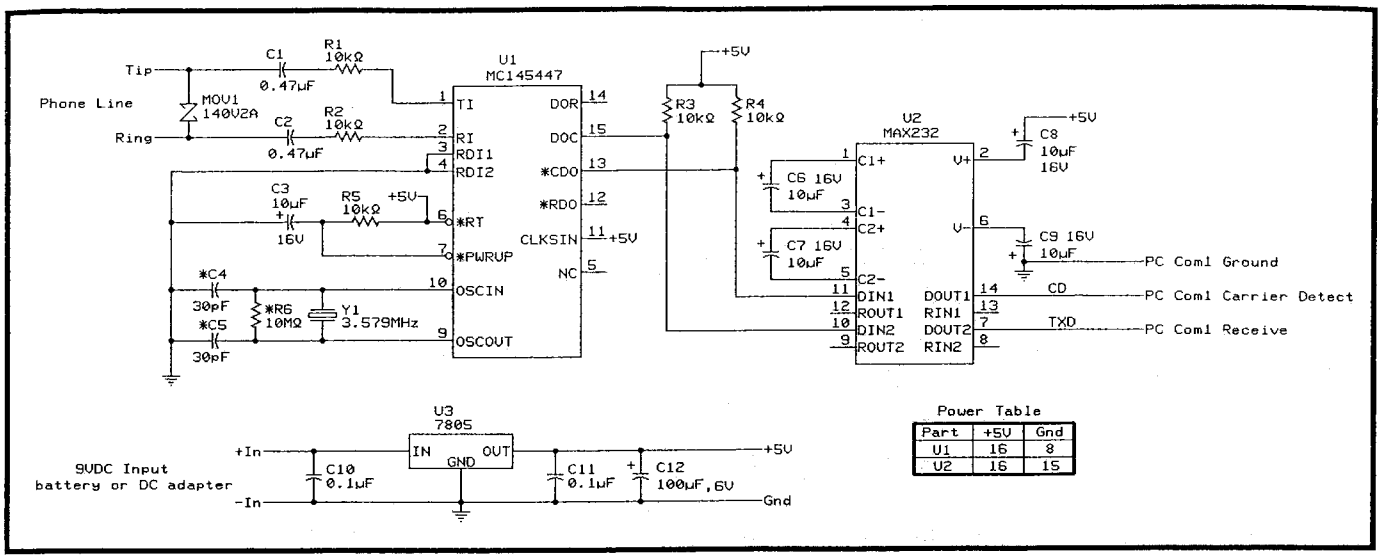


Figure 1—*The complete circuit brings the phone line in one side and delivers PC-compatible serial data out the other.*

A logical 0 (called a *space*) is sent as 2200 Hz, and a logical 1 (called a *mark*) is 1200 Hz. You can see there is nothing special or proprietary about the data. It comes to you exactly as your PC would send it out its serial port.

If you're using DSP to decode the CID signal, the typical worst-case amplitude of the transmitted signal is –13.5 dBm from the telephone company. The facilities (the wires traveling though your town and ending at your house) introduce another –20 dB of attenuation.

This interference means that the CID receiver demodulator should be capable of decoding a worst-case signal of –34.5 dBm. I chose the Motorola device because it meets this worst-case specification.

If you look at the schematic presented in Figure 1, you see that the telephone line goes into one side of the CID receiver chip and data comes out the other side in inverted physical format.

If you take this inverted data and feed it into a MAX232 RS-232 driver/receiver, it inverts the data and translates it into RS-232 standard voltages.

At this point, you're ready to attach the RS-232 data to a PC or embedded controller and make a call to the chip. When the CID data is decoded, it is presented to the PC in an almost readable format. You can see the caller name, telephone number, time, and date along with some garbage data.

Since we aren't making a standalone box, but one that connects to a PC, I disregarded any features for power saving or ring detection. You should power the project from an isolated AC adaptor.

RING SIGNALS

The CID receiver is forced to stay active always because RDI1 and RDI2 are held low and *RT is held high. A standard delay-type reset circuit made from C3 and R5 makes *PWRUP go low shortly after power is applied to the circuit.

In this always-active mode, the data out of the DOC pin attempts to decode any signals on the telephone line, including ringing voltages and occasional DTMF signaling, during the dialing of outgoing calls. This data appears as garbage.

The circuit takes another signal out of the CID chip on *CDO, which is only high when valid CID data is being received. The software of your system ignores and flushes all characters received when *CDO is low.

As soon as *CDO goes high, the software buffers all received characters in a queue. This technique ensures that the queue always contains usable data.

One discrepancy between the databook description of the device and my real-world prototype is the drive capability of the DOC and *CDO pins. I found it necessary to apply a 10-kΩ pullup resistor for the MAX232 to receive the data correctly.

When you receive the Caller ID message, it can be up to 80 characters long and in one of two standardized formats called *fixed* and *variable*. The variable format is standard in North America and is what I will discuss here.

VARIABLE FORMAT

The variable-format service data is one long data package divided into subpackages. The first two characters received in the data package start with an 80 hex. The next char-

[0x80][0x27][0x01][0x08] 04301212 [0x02][0x0a] 2145554141 [0x07][0x0f] Caller's Name [0x01][0x01][0x00]

0x80—indicates the start of package indication
0x27—indicates the total number of characters (hex) to be transmitted in the main package
0x01 0x08—means that the date and time are coming next and are 8 characters
0x02 0x0a—signals that the calling number is next and is 10 characters
0x07 0x0f—indicates that the calling name is next and is 16 characters
0x01 0x01—is the checksum
0x00—marks the end of transmission

Table 1—*An example of a complete caller ID stream from the telephone company includes name and number.*

acter is the number of characters total to be transmitted.

The subpackages follow, starting with a character that indicates the type of subpackage (name, number, date, time, service) and the total number of characters in the subpackage. The subpackage types include:

- 0x01—date and time in DDDDTTTT format
- 0x02—calling number
- 0x07—calling name

Table 1 offers an example of a package and indicates what the separate components of the package stand for.

Your software should sync on the 0x80 character and be able to accept any package type next. There is no guarantee that the subpackages will arrive in a certain order nor that all subpackages will be sent in a particular package.

There are other subpackage types I won't elaborate on but which might indicate private or blocked calls. Typically, even if a subpackage meaning *private* or *blocked* is sent, a number package is also sent with an ASCII "P" or "B" in the first character of the called number field. Your software should not always expect numeric data for the number field.

If you decide to apply this circuit to an application which doesn't have differential input, you should be able to couple the signal directly into the tip pin. Since you're not using a differential input, this signal needs to be twice the recommended amplitude to activate the demodulator section of the Motorola device.

If you find this hard to do in a single supply system, you could add an inverting op-amp to the tip pin and apply your signal to the input of the op-amp and the ring pin. This modification simulates a differential input to the chip from an externally provided single-ended input.

EXPECTATIONS & APPLICATIONS

If you expected this article to be deeply technical, you're probably thinking, "Gosh! This is really easy!"

Yes, it is. So, when you apply this circuit or specification to your system, if you use the Motorola chip as a caller ID decoding block, it should be almost plug-and-play.

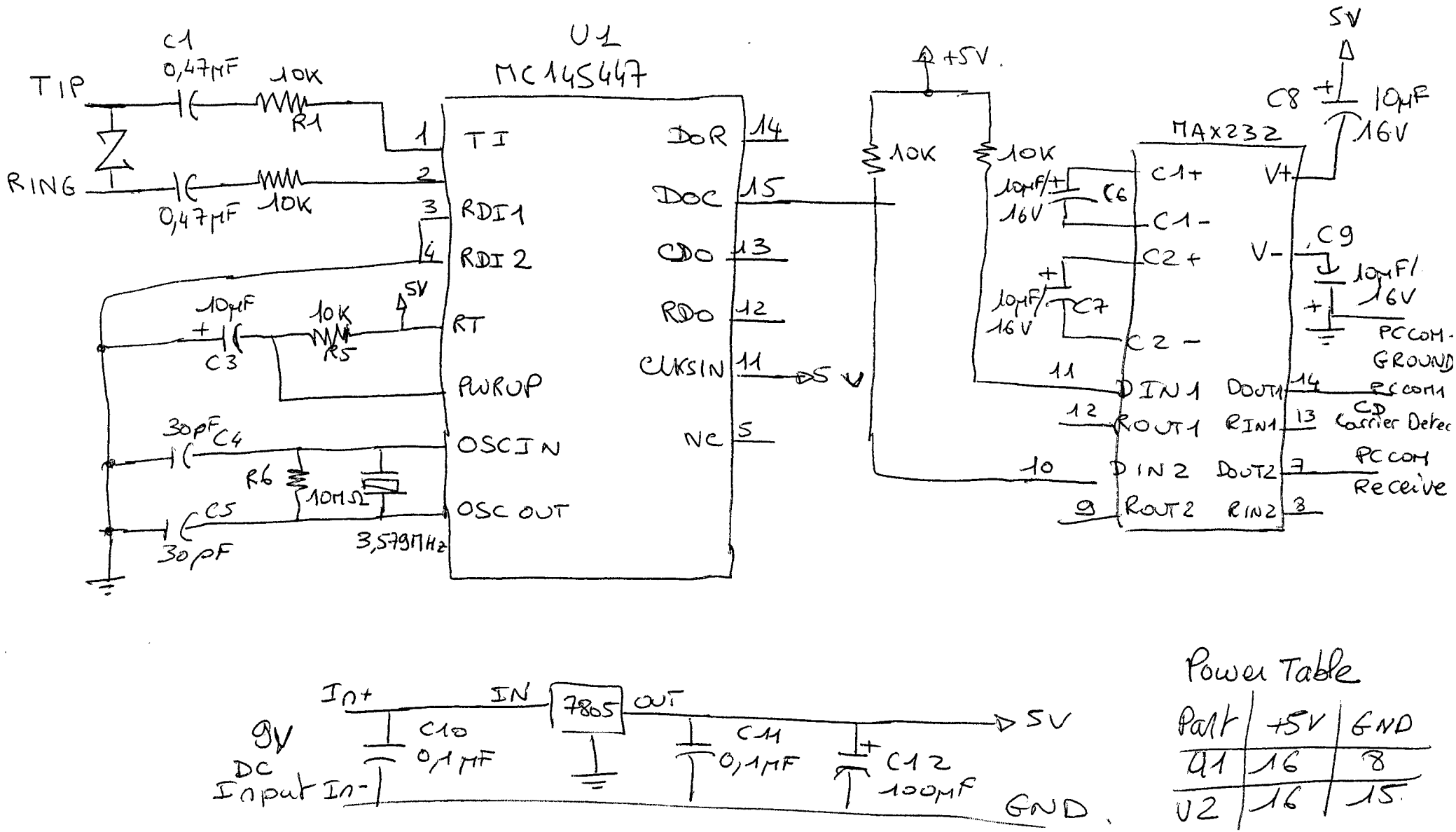
All that's left is the application. You could have a window pop up a caller's name and number onscreen. This read-out could be juxtaposed with another window that holds notes about the caller from a database and include details such as account status.

Heh! Before you know it, you're enabled! You've found a perfect application for Caller ID and you.

Richard Newman is an electrical engineer living in Dallas, TX. He designs specialized communications and industrial automation equipment either in partnership or on contract. He may be reached at ricardo@netcom.com.

IRS

404 Very Useful
405 Moderately Useful
406 Not Useful



Power Table

Part	+5V	GND
U1	16	8
U2	16	15