Order this document
by AN1733/D

Motorola Semiconductor Application Note

AN1733

Implementing Caller ID Functionality in MC68HC(7)05 Applications

By Derrick B. Forte and Hai T. Nguyen
Networking and Communications Operation
Austin, Texas

Introduction

Caller ID is a service that transmits information concerning a calling party, such as a telephone number and name, to a called subscriber. Caller ID capable equipment at the subscriber's premises captures, processes, and displays the data. The majority of Caller ID subscribers are residential customers who use the service to screen incoming calls. Caller ID is also used by commercial subscribers to automate the retrieval of customer records from in-house databases. The widespread acceptance of this service in both the residential and commercial subscriber markets has led to the development of a number of different types of Caller ID devices such as Caller ID adjunct boxes, computer peripherals, and telephones with Caller ID functionality.

This application note explores the hardware and software issues involved in implementing Caller ID functionality in applications based on Motorola's Family of MC68HC(7)05 microcontrollers (MCU). The note starts with a discussion of the signals and protocol used by service providers to transmit Caller ID data. The remainder of the note is devoted to a design example. The application developed for this note is that of a computer peripheral that is based on a Motorola

MOTOROLA

© Motorola, Inc., 1998

MC68HC(7)05P9 microcontroller and is capable of receiving Caller ID transmissions and displaying the received data on an IBM AT compatible computer.

The Caller ID Protocol

A number of requirements governing Caller ID transmissions are imposed on a service provider.

The first of these is that the transmission of Caller ID data is permitted whether the equipment at the customer's site is in the on-hook or off-hook state. This requires that a service provider be able to detect the state of a subscriber's equipment and adjust the transmission of data accordingly. Since transmitting data while the customer's equipment is in the off-hook state usually involves interrupting an ongoing call, the issues that must be addressed for transmitting to equipment in the two states are different. This has led to the development of a signalling and transmission protocol for the off-hook state that is more sophisticated than that for the on-hook state. As a result, devices capable of receiving Caller ID data while the subscriber's telephone is off-hook costs more than on-hook only devices. Given that on-hook only devices were offered first and the added cost to both the service provider and the subscriber to support the off-hook state, most Caller ID capable equipment in service today only support the on-hook protocol. For this reason, this application note only covers the design of applications that are capable of supporting the on-hook protocol and, consequently, the specifications dealing with the off-hook state will not be discussed here.

The transmission of Caller ID information is governed by a set of specifications developed by Bellcore. These requirements, known as the Voiceband Data Transmission Interface, define the encoding, timing, and formatting of Caller ID data and the electrical characteristics of the analog signals used to transmit it. The interface allows data transmissions for the on-hook state to occur with or without power ringing. The more common of the two cases is that in which the transmission of data is preceded by a power ring. The interface specifies that data transmission occur in the silent interval between the first and

second power rings of a call. It should be noted that a power ring need not consist of a single continuous tone. There are some service providers that signal their subscribers with a series of short tones instead of one continuous tone. The burst of tones occurs within the period of time normally allotted for a single tone, thus replacing the single power ring. The tone burst is then followed by a silent interval and another burst. This type of signalling is known as distinctive ringing. The interface allows service providers that use distinctive ringing patterns to transmit Caller ID data as long as the silent interval is of a minimum length. If a provider's signalling does not meet these specifications, the transmission of Caller ID data is not supported. **Figure 1** illustrates the timing specifications that service providers must meet to transmit Caller ID data.

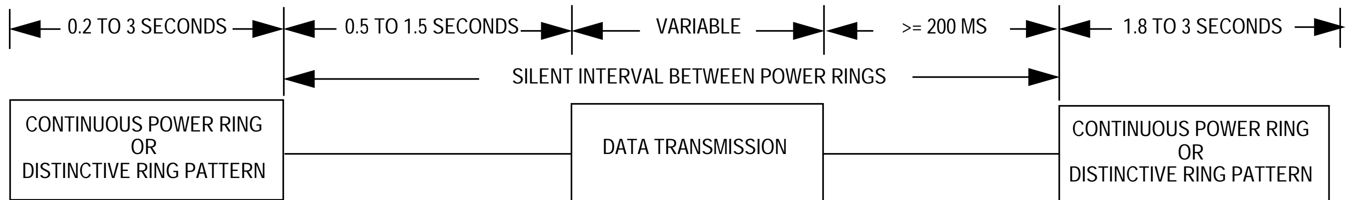


Figure 1. Caller ID Data Transmission Timing Specifications

The interface is composed of three layers:

- Message assembly layer
- Data link layer
- Physical layer

At a service provider's facility, a block of Caller ID data flows down from the message assembly layer, through the data link layer, and finally down to the physical layer where it is converted into a modulated analog signal. As a block travels through the interface, each layer prepends and appends data to the original block until it reaches the physical layer. This process is similar to that used by many protocol stacks such as TCP/IP.

The message assembly layer, the highest level of the interface, specifies a message format for a block of Caller ID data. The information contained in the block is determined by the level of service to be

provided to subscribers. The data layer defines the framing format and error correction methods that are applied to the raw digital bit stream that is transmitted to the subscriber. At the lowest level, Caller ID data is transmitted over the telephone lines as a modulated analog signal. The physical layer specifies the signal's modulation and its electrical characteristics. The following sections will discuss the three layers.

The Message Assembly Layer

The message assembly layer segments the data within a Caller ID data packet. The layer defines two formats for packets:

- Single data message format (SDMF)
- Multiple data message format (MDMF)

The SDMF is the simpler of the two formats and is discussed first.

The message assembly layer forms an SDMF formatted packet by prepending a 2-byte header to a Caller ID data packet. The first byte of the header is the packet's message type parameter value. This value alerts a Caller ID device as to the type of information that is contained in the accompanying data block. There are currently three values defined for the message type parameter of SDMF formatted packets. A packet with a message type value of 0x04 contains the number of a calling party, a value of 0x06 indicates a message waiting indicator packet, and a value of 0x0B has been reserved for future applications. The next byte in the header is the message length parameter. This parameter, as its name suggests, contains the number of bytes of Caller ID data that remain in a block. Figure 2 shows the structure of an SDMF frame.

Figure 2. SDMF and MDMF Frame Structure

The message assembly layer also specifies the arrangement of information within the data portion of a frame. The data portion of an SDMF frame consists of ASCII codes arranged in three fields representing the date, time, and number of an incoming call.

- The date field which is the first piece of information in the data block, consists of 4 bytes. The first two bytes are ASCII codes for the digits representing the month and the other two represent the day. Any months or days that can be represented by a single digit, are preceded by a zero.
- The next 4 bytes are ASCII codes for digits representing the time. The first two bytes are the ASCII codes for the digits representing the hour and the remaining two represent the minutes. The values in the hour field are allowed to range from 0 to 23, while the minutes can range from 0 to 59.
- The remaining 10 ASCII codes represent the digits of the telephone phone number of the incoming call. If the incoming call is a local call and lacks an area code, the area code portion of the number field is filled by default with the ASCII codes for three zeroes. **Figure 3** illustrates the arrangement of information within an SDMF formatted packet.

Figure 3. SDMF Data Block Format

The message assembly layer's second format, the multiple data message format (MDMF), is more complex and versatile than the SDMF. It is capable of delivering more data concerning an incoming call, such as the name of a calling party. Although both formats share some features in common such as the message type and the message length parameters, the structure of the MDMF is more flexible and more readily accommodates the development of new provider services.

As with SDMF, the message type parameter is the first byte of an MDMF formatted packet. Since MDMF is designed to provide a wider variety of information than SDMF, six message type values are specified for it instead of the three defined for SDMF. The message type value identifying a Caller ID data block is 0x80, 128 decimal. The message length parameter, the next byte in the frame, specifies the number of bytes that follow in the data block. It is after the message length

parameter, that the structure of an MDMF packet differs from that of a SDMF.

At this point, the message block is broken up into smaller messages called parameter messages. Each parameter message is composed of a parameter type value, a parameter length value, and an accompanying data block that contains a specific type of information, such as the number of an incoming call. Each piece of information that is transmitted in an MDMF formatted packet, such as the time and date, calling number, and the calling name, is packaged within its own parameter message. This encapsulation of data enables a service provider to selectively add to or omit information from Caller ID transmissions.

As its name suggests, the parameter type value is used to identify the type of data contained in a parameter message. Currently, 17 parameter type values are defined for MDMF. The values of most interest to Caller ID capable equipment are 0x01, 0x02, and 0x07 which identify a time and date, number, and name parameter message respectively. This parameter is followed by the parameter length value which contains the remaining number of bytes in a parameter message's data block. **Figure 4** illustrates the format of a parameter message.

Figure 4. Parameter Message Structure

As mentioned earlier, each parameter message contains a specific piece of information, such as the time and date of an incoming call. If a parameter message carries a type of information that is supported by SDMF, the data within it is represented and arranged the same way as its counterpart in SDMF. A calling party's name, a data type that is not supported in SDMF, is transmitted as the ASCII codes for the letters comprising the name.

The Data Link Layer

The data link layer formats the data to be transmitted into a frame for the next layer below it, the physical layer. The layer also defines Caller ID's signalling and error detection functions. This layer prepends a start bit and appends a stop bit to each byte of data that it has received from the message assembly layer. The layer also specifies that data be transmitted LSB (least significant bit) first. The layer prepends a preamble sequence and appends a checksum to each frame of data that is transmitted. The preamble serves as the signalling mechanism to alert and condition the customer's equipment for receiving a transmission.

The preamble sequence for the on-hook protocol consists of two parts:

- Channel seizure signal
- Mark signal

The channel seizure signal is transmitted first and consists of 300 bits of alternating 0s and 1s. The sequence begins with a 0 and ends with a 1. The seizure signal is followed by the mark signal which consists of 180 marks or high bits. The checksum that is appended to each frame serves as a transmission's error detection mechanism. Though the data link layer provides for error detection, no provision is made for error correction. The on-hook protocol does not provide a mechanism for a subscriber's Caller ID device to request a re-transmission of data from the central office. Therefore, it will discard a frame if an error is detected.

Transmission errors are detected by calculating a checksum value as data bytes are received and comparing the value to the checksum value sent at the end of a transmission. If the two values are identical, there is a high probability that the transmission is error free. The detection of an error is not absolute because the Caller ID's error detection algorithm is not capable of detecting every possible transmission error. Caller ID checksums are the two's complement of the modulo 256 of the sum of all the data bytes within a frame starting with the message type parameter byte and excluding the checksum. After being processed by the data link layer, a Caller ID frame appears as pictured in **Figure 5**.

CHANNEL SEIZURE SIGNAL 300 ALTERNATING 0s AND 1s	MARK SIGNAL 180 1s	CALLER ID DATA	CHECKSUM

Figure 5. Caller ID Frame Structure

The Physical Layer

The physical layer defines the electrical characteristics of the analog signal used in the transmission of Caller ID data frames over the public telephone network's lines. The physical layer also defines the method of modulating the signal. The physical layer specifications require that data be transmitted to a subscriber's equipment as an asynchronous serial binary bit stream at a rate of 1200 baud plus/minus 1%. Data is actually delivered to a subscriber's equipment by means of a binary frequency-shift-keyed (BFSK) modulated analog signal. During a transmission, a logic 1 is coded by a frequency of 1200 Hz plus/minus 1%, while a logic 0 is coded by 2200 Hz plus/minus 1%.

This concludes the discussion of the structure and theory of the Caller ID protocol. The remainder of this note is devoted to applying this information in the development of an application based on an MC68HC(7)05P9 microcontroller.

Design Example: An IBM AT Keyboard Caller ID Device

The design example developed for this application note is that of a Caller ID-capable IBM AT-compatible computer peripheral. The application consists of two parts:

- A peripheral device that interfaces with an IBM AT host computer at its keyboard interface
  - The peripheral connects to the host's keyboard interface at one end and to the keyboard's cable at the other. The device uses the host computer's keyboard interface as its power supply and communications link to the host.
- CALLERID.EXE, a Windows 95 application program that executes on the host computer

At its highest level, the system's operation is as follows. (Consult **Appendix B — System Operation Flow Chart** for a flow chart describing the system's operation.)

1. The keyboard Caller ID device is powered on and reset when the host computer is turned on. The device than waits for the arrival of a Caller ID transmission.
2. CALLERID.EXE is invoked immediately after Windows 95 boots up. CALLERID.EXE hides its main window and begins executing in the background.
3. On receiving a Caller ID transmission, the device transmits a <CONTROL L> to the host through the host's keyboard interface in the form of a series of IBM AT keyboard scan codes.
4. The host interprets the scan codes as a series of keystrokes, interrupts the application that currently has the focus in the Windows 95 environment, and gives the focus to CALLERID.EXE.
5. After a time delay, the device transmits an error code to the host if it detected an error in the Caller ID data. CALLERID.EXE then displays an error message in its main window, a dialog box. If the Caller ID data is error free, the device transmits it as a series of scan codes which CALLERID.EXE interprets as an ASCII string. CALLERID.EXE then processes the string and displays the information in its dialog box on the host's screen.
6. After transmitting the Caller ID data to the host, the peripheral returns to scanning the telephone line for Caller ID transmissions.

Keyboard Caller ID Device Hardware Design Overview

The keyboard Caller ID device's hardware design is divided into two functional blocks:

- Caller ID data acquisition block
- Keyboard interface block

The Caller ID data acquisition block serves as the application's interface to the telephone line. This block receives the Caller ID analog signal, demodulates it, and converts it into a digital stream. The design of this block is centered on Motorola's MC145447 calling line identification

receiver with ring detector device which, when used with a few passive components, is capable of performing these functions.

The digital stream produced by the MC145447 is passed to the keyboard interface block, which is primarily implemented with a Motorola MC68HC705P9 microcontroller (MCU). The MC68HC705P9 parses the stream into individual bytes and converts the data into IBM AT keyboard scan codes for transmission to the host. These scan codes are sent to the host through its keyboard interface. The host computer interprets the scan codes as a series of keystrokes which are processed by CALLERID.EXE. This block also provides the device's interface to the host computer's keyboard interface by emulating the signals used in keyboard-to-keyboard interface transactions and the IBM AT keyboard interface protocol.

This discussion of the application's hardware design continues by examining each of these blocks in detail.

The Caller ID Data Acquisition Block

The Caller ID data acquisition block performs these two functions within the application's system design:

1. Provides an electrical interface to the telephone line
2. Demodulates and validates the Caller ID analog signal and converts it to a digital bit stream

Although many Caller ID designs implement these functions with discrete analog circuitry, a more integrated solution was chosen for this application. Motorola's MC145447 calling line identification receiver with ring detector device was chosen to implement this entire block. This device is capable of providing the needed interface to the telephone line, demodulating the BFSK asynchronous data signal, and outputting a digital stream. The design of this block was largely taken from pages 2-765–2-774 of the *Motorola Communications Device Data*, Motorola document order number DL136/D, Rev. 3. The device also has a number of signal validation and power-saving features that are useful for Caller ID designs for which low-power consumption is an issue. Since this application is powered by the host computer's keyboard interface, it does not use any of the MC145447's power-saving modes.

The MC145447's interface to the telephone line's twisted pair can be divided into two types of signals:

- Caller ID data acquisition signals
- Ring detection and validation signals

The ring detection and validation signals serve to detect the presence of a valid ring signal on the twisted pair and participate in bringing the device out of power down mode.

Four signals comprise the ring detection and validation portion of the interface. Three of the signals, ring detect in 1, RDI1, ring detect in 2, RDI2, and /ring time, /RT, are inputs. There is also one output, ring detect out, /RDO, which is asserted when a valid power ring is detected on the telephone line twisted pair. The /RT pin works in conjunction with the RDI1 pin to generate internal signals that are part of the device's power-up circuitry.

To conserve power, the MC145447's power-up circuitry applies power to different sections of the device as they are needed. In the power-up sequence, the /RT and RDI1 signals are used to activate power to the ring analysis section of the device. This section determines whether a valid ring signal is present on the twisted pair. As shown in the schematic in **Appendix A — Keyboard Caller ID Device Schematics**, the voltage at the RDI1 pin is provided by resistor R10, which is part of a voltage divider circuit comprised of resistors R10, R11, and R12. The resistor network divides an AC coupled, rectified version of the voltage present between the tip and ring sides of the twisted pair into voltages that are sampled by the RDI1 and RDI2 pins. The value of R10 is chosen such that if a voltage of 40-Vrms or more is present on the twisted pair, which indicates that a power ring might be taking place, the RDI1 pin and its associated circuitry will turn power on to the ring analysis circuitry. The /RT pin is connected to an RC combination that holds the pin low during the low periods of a power ring. The RDI2 pin serves as the only input to the ring analysis section. The signal at this pin is provided by resistor R12 of the divider network. The duty cycle of this signal is used to validate the presence of a power ring. In the event that a power ring is detected, the ring analysis circuit asserts the /RDO pin.

The data acquisition signals on the MC145447 consists of a tip input, TI, and ring input, RI pin. The tip input is AC-coupled to the tip side of the telephone line's twisted pair through capacitor C7. The ring input signal is AC-coupled to the ring side of the twisted pair through capacitor C8. The signal that is presented to these two pins is demodulated and converted into the digital stream that is output by the device.

In this application, the MC145447's interface with the system's microcontroller consists of three pins:

- Data out cooked, DOC
- /Ring detect out, /RDO
- /Power up, /PWRUP

The MC145447 outputs a digital stream on two pins:

- Data out cooked, DOC
- Data out raw, DOR

The DOR pin outputs the entire data stream demodulated by the device starting with the channel seizure and mark signals and ending with the checksum byte at the end of a transmission. The DOC pin, on the other hand, outputs data after a transmission passes an internal data validation process and does not output the channel seizure and mark signals. Data is captured by the MC68HC(7)05P9 by connecting DOC to pin PC0 on the MC68HC(7)05P9 which is configured as an input. The /RDO pin is connected to pin PC2 of the MCU which is configured as an input. As stated earlier, the /RDO pin is asserted when a valid power ring is detected on the twisted pair. The assertion of /RDO, along with the start of the transmission of data within 0.5 to 1.5 seconds after the deassertion of /RDO, is used by the MC68HC(7)05P9 to qualify the start of a data stream from the MC145447.

The MC145447 has a requirement that its /PWRUP pin be at a logic 1 for a minimum of 10 µs after V_{DD} reaches its full value. Typically, this requirement is met by delaying the assertion of /PWRUP with an RC circuit. To eliminate the need for these two components, the /PWRUP pin is connected to the MC68HC(7)05P9's PC3 pin which is configured as an output. This pin asserts /PWRUP after an appropriate delay.

The Keyboard Interface Block

An in-depth discussion of the signals, protocol, and the hardware and software issues involved in interfacing an MC68HC(7)05-based application to the keyboard interface of an IBM AT-compatible computer is provided in the application note *Interfacing MC68HC05 Microcontrollers to the IBM AT Keyboard,* Motorola document number AN1723/D. The generic circuit presented in this note served as the basis for the design of the keyboard interface block in this application.

NOTE: *Note that the scan code set used in this application is the IBM AT keyboard set. This differs from the PS/2 scan code set that is used by the keyboards of some IBM AT-compatible machines. The keyboard Caller ID device will not work on host computers with keyboards that use the PS/2 scan code set.*

Keyboard Caller ID Device Software Design Overview

The software design of this application is divided into two parts:

- The firmware that resides on the MC68HC(7)05P9, the application’s microcontroller
- CALLERID.EXE, a Windows 95 application program

The firmware’s main function is to capture the raw digital data stream generated by the MC145447 and transmit it to the host computer for further processing. Data is transmitted to the host in the form of keyboard scan codes that are sent through the host’s keyboard interface. The host receives the scan codes and interprets them as keystrokes. The sequence of simulated keystrokes is read by CALLERID.EXE. CALLERID.EXE, the Windows 95 application program, parses and converts the string back into binary data from which it extracts Caller ID information. CALLERID.EXE then formats and displays the data in a dialog box that serves as the application’s main window.

This division of functionality between the Caller ID device and the host computer allows for the greater portion of processing to be off loaded to the host computer where a larger amount of resources are available. This reduces the functionality of the Caller ID device, thus allowing its design to be implemented with a smaller and cheaper microcontroller.

The following sections provide a detailed description of the design and implementation of both CALLERID.EXE and the application's firmware.

Keyboard Caller ID Device Firmware Design

The Caller ID device's firmware follows the program flow shown here. (Consult **Appendix C — Keyboard Caller ID Device Firmware Flow Chart** for a flow chart describing the firmware's design.)

1. On reset the general I/O (input/output) pins on the MC68HC(7)05P9 are configured and initialized to implement the Caller ID device's hardware design.
2. The firmware waits in an loop for the assertion of the MC145447's /RDO signal, which is monitored on the MC68HC(7)05P9's PC2 I/O pin. The assertion of this signal indicates that a power ring has been detected on the twisted pair.
3. The MC68HC(7)05P9 waits for the deassertion of the MC145447's /RDO pin within 2.25 seconds after its assertion. If the MCU detects a start bit on the DOC pin within two seconds after the deassertion of /RDO, the conditions are met for the MC68HC(7)05P9 to begin monitoring for a transmission.
4. The MC145447 transmits the CALLER ID data to the MC68HC(7)05P9 in the form of a raw digital stream on its DOC pin. The MCU reads the data on its PC0 pin.
5. On receiving the data from the MC145447, the MC68HC(7)05P9 parses the stream into individual bytes and checks the data for a checksum error. If a checksum error has been detected, it is flagged by a global variable; otherwise, the data is converted into an array of AT keyboard scan codes for transmission to the host computer.
6. The application transmits a <CONTROL L> keystroke sequence as a series of scan codes. This interrupts the application that currently has the focus in the Windows 95 environment, restores CALLERID.EXE's hidden main window, and gives it the focus.
7. If a checksum error was not detected during the reception of the CALLER ID data, the scan code array that represents the received data is transmitted to the host computer; otherwise; an error code is sent.

8. The firmware returns to monitoring the twisted pair for a new Caller ID transmission.

The firmware's functions can be divided into three types of routines:

- Device initialization routines
- Caller ID data acquisition routines
- Keyboard interface routines

The device initialization routines configure and initialize the MC68HC(7)05P9's I/O pins to implement the application's hardware blocks. As mentioned earlier, port A I/O pins PA0–PA5 are configured to implement the keyboard interface block, while three port C pins – PC0, PC2, and PC3 – serve as the MC68HC(7)05P9's interface to the MC145447. All remaining general-purpose I/O pins are configured as outputs to eliminate the need for pullup resistors on them. The data acquisition routines of the firmware consist of the sampling and time delay routines that capture data from the MC145447's DOC line. The MC68HC(7)05P9 samples the data stream at its PC0 pin and parses it into individual bytes. The fact that each piece of Caller ID data begins with a start bit and ends with a stop bit makes it easy to delineate between individual bytes. The time delay functions used for data acquisition routines are not only used to sample the bits within a byte but must also allow for the inter-character delays that the interface allows.

The keyboard interface firmware mainly consists of a transmission routine and its accompanying time delay functions. The keyboard interface's transmit function has within it a call to a routine that is capable of receiving host computer commands. A host computer's keyboard interface will hold the data line low if it detects a transmission error in a keyboard-to-host data transfer. The keyboard protocol stipulates that a host computer send a resend command (0xFE) to the keyboard if it detects an error in a keyboard-to-host data transfer. Therefore, the keyboard interface block's transmit routine must be able to receive the host's resend command and re-transmit the original data in the event of an error. For this application, the number of retransmission attempts was arbitrarily set at one. Therefore, if an error occurs when the device sends a byte to the host, the device will capture the host's resend command and attempt a retransmission of the data. If the retransmission fails, the

device will reconnect the keyboard's clock and data signals to those of the host and return to monitoring the telephone line. To transmit data to the host, the transmission routine toggles PA0, the data output signal, and PA2 pin, the clock output signal, in accordance with the timing specifications for keyboard-to-computer data transfers. The host command reception routine reads the data from the PA1 pin and toggles the clock signal in accordance with the timing specifications for computer-to-keyboard data transfers.

CALLERID.EXE Design

Before discussing the implementation of CALLERID.EXE, an explanation of some of its design concepts is in order.

The operation of CALLERID.EXE is analogous to that of the terminate-and-stay-resident (TSR) programs familiar to MS-DOS users. TSRs are DOS applications that, unlike normal programs, remain in the PC's memory even though they may not be executing at the time. TSRs are invoked by the user's pressing a pre-determined key or key combination. These key sequences, commonly referred to as hot keys, typically consist of the CONTROL key followed by a letter. On receiving a hot key sequence, a specially designed section of the TSR's code that is usually loaded when DOS boots up, interrupts, or stops the execution of the application that is currently running in the DOS environment and starts the execution of the TSR. CALLERID.EXE operates in much the same way. CALLERID.EXE is invoked immediately after Windows 95 boots up, along with all the other programs in the Windows 95 **Start Menu/StartUp** folder. This is done by placing CALLERID.EXE in the folder by using the **Start Menu Programs** option in the Windows 95 **Settings** menu. CALLERID.EXE continues to run in the background until it is given the focus in the Windows 95 environment by a <CONTROL L> key sequence being sent by the keyboard Caller ID device.

When developing a TSR program, programmers have to write the code that enables the program to remain resident in memory and respond to the desired hot key sequence. Oftentimes this has led to situations in which TSRs interfere with the operation of normal programs and with each other. The Windows environment eliminates the need for a developer to write low level code TSR code by providing two functions in

its applications programming interface (API) that can give a Windows application the same functionality as a TSR. These two functions allow an application to connect or disconnect user-defined functions to Windows 95.

The first of these functions, `HHOOK SetWindowsHookEx(int idHook, HOOKPROC lpfn, HINSTANCE hMod, DWORD dwThreadId),` passes the address, as defined in the function's `lpfn` parameter, of a user-defined function to the Windows 95 operating system. These functions, known as hook functions, allow an application to filter a pre-determined set of events or user inputs, such as keystrokes and mouse movements, before they are passed to the Windows environment at large. The `SetWindowsHookEx` function's `idhook` parameter specifies the type of event or input that the hook function pointed to by `lpfn` will recognize. Besides user input events like the keyboard and the mouse, the Windows API also defines values for the `idhook` parameter for hooks that facilitate a number of other functions such as debugging, and the development of computer training programs. Code within a hook function can be used to redirect the flow of the application program that currently has the focus, or Windows 95 as a whole.

The second Windows API function, `UnhookWindowsHook` disconnects a hook function from Windows 95.

Hook functions can have local or global scope. Hooks with local scope only function within the context of the application that currently has the focus in the Windows environment. These local hook functions are used to implement the hot keys that have become a mainstay in word processing and spreadsheet programs. Global hooks, on the other hand, are operational systemwide and can be used to alter the functioning of the Windows 95 environment. Although the code for local hook functions can be part of the application they support, Windows 95 dictates that global hook functions must reside in their own separate dynamically linked library (DLL).

CALLERID.EXE's design, therefore, is divided into two parts:

- CALLERID.EXE, the executable program
- CALLDLL.DLL, the DLL containing the global hook function

Both modules were compiled with Microsoft Visual C++ Version 2.0. CALLDLL.DLL's code consists of a function to install the keyboard hook function and the hook function itself. In the code's call to the Windows API's `SetWindowsHookEx` function, the `idhook` parameter is set to WH_KEYBOARD, which is a pre-defined value that configures the hook function to handle keyboard events. This code is placed in a DLL because Windows requires that global hook functions reside in a DLL. The keyboard hook function in this application must be global in scope so that CALLERID.EXE can be invoked no matter what application may currently have the focus in Windows 95. The only limitation with CALLERID.EXE is that it will not be invoked if the current window with the focus is a DOS window.

The main function of the executable is to receive the Caller ID data from the Caller ID device, format it, and display it in a dialog box on the PC's monitor. The program flow of the executable is as follows. (Consult **Appendix D — CALLERID.EXE Program Flow Chart** for a flow chart describing CALLERID.EXE's design.)

1. CALLERID.EXE is invoked immediately after Windows 95 boots up. The main window of the CALLERID application is initialized to come up in the hidden state. This causes CALLERID.EXE to begin executing in the background of the Windows 95 environment.
2. CALLERID.EXE accesses CALLERID.DLL and connects the keyboard hook function to Windows 95. The hook function examines each keystroke that is entered by the user for the <CONTROL L> hot key sequence.
3. On detecting a <CONTROL L> key combination, the keyboard hook function calls the Windows API `FindWindow( )` function to locate the application's hidden main window. The Windows `ShowWindow( )` function is then called to activate CALLERID.EXE's main window and give it the focus in Windows 95.

4. CALLERID.EXE displays a popup dialog box on the monitor displaying this text: "Receiving Data . . ."
5. The application waits for a keystroke from the Caller ID device.
6. If CALLERID.EXE receives a ';' character from the Caller ID device, the device has detected a checksum error in the Caller ID data received from the telephone line. The CALLERID.EXE will then display this message in the dialog box: "Line Error." Otherwise, it acquires the full stream of Caller ID data from the device.
7. C-language string manipulation functions are used to parse the string into the two character segments that represent each byte of Caller ID data. C-language string conversion functions are then used to convert each ASCII segment into the original binary data that was captured on the Caller ID device.
8. CALLERID.EXE formats the binary data so that it can be displayed in the dialog box. CALLERID.EXE will format data according to whether the Caller ID data received is in the SDMF or MDMF format.
9. The Caller ID information is displayed in the dialog box. The dialog box remains displayed until the user presses the dialog box's "OK" or "Deactivate" buttons.
10. The dialog box is hidden again if the user presses the "OK" button. CALLERID.EXE then returns to waiting for a hot key sequence. If the "Deactivate" button is pressed, CALLER.EXE will be deactivated and will no longer function until Windows 95 is reset.

Summary

Caller ID services provide both commercial and residential customers with valuable data for more efficient processing of telephone calls. The MC145447 used in conjunction with a member of the Motorola 68HC(7)05 microcontroller family provides a cost-effective hardware solution for the implementation of Caller ID applications.

Keyboard Caller ID Device Operating Instructions

1. Copy CALLERID.EXE to the hard drive and directory of your choice. A suggested path might be C:\CALLERID\.
2. Copy CALLDLL.DLL to the C:\WINDOWS\SYSTEM\ directory.
3. Add CALLERID.EXE to the Windows 95 start menu by performing these steps:
   a. Press the **Start** button on the Windows 95 TaskBar.
   b. Select the **Settings** item from the menu displayed.
   c. Select the **Taskbar** item from the submenu displayed. This will cause a dialog box to be displayed.
   d. Select the **Start Menu Programs** tab in the dialog box.
   e. Press the **Add** button.
   f. Press the dialog box's **Browse** button.
   g. Find CALLERID.EXE using the dialog box provided. Press the **Next** button.
   h. You will be asked to select a folder to place the shortcut for the selected program. Find and select the **StartUp** folder from the list of folders displayed. Press the **Next** button.
   i. Press the **Finish** button to complete the setup.
4. Disconnect the keyboard's connector from the host computer's keyboard port.
5. Connect the keyboard Caller ID device to the host computer's keyboard interface.
6. Connect the keyboard's connector to the receptacle for it on the keyboard Caller ID device.
7. Connect the telephone line to one of the R-J11 connectors on the keyboard Caller ID device.
8. Connect a telephone extension line between the keyboard Caller ID's second R-J11 connector and your telephone. This completes the hardware installation of the keyboard Caller ID device.

9. Shut down and restart Windows 95.
10. Caller ID should now be activated. Caller ID will display a dialog box with Caller ID information every time data is sent to it by the keyboard Caller ID device. To manually deactivate the program, press the <CONTROL L> key combination and press the **Deactivate** button in the dialog box.

Bibliography

Holzner, Steve Advanced Visual C++ 4.1st. ed., New York, N.Y.: M&T Books, 1996.

Konzak, Gary J. PC Keyboard Design. 2nd. ed., San Diego, Calif.: Annabooks, 1993.

LSSGR: Voiceband Data Transmission Interface, Section 6.6, GR-30-CORE, Issue 1, Bellcore, December 1994.

"Message Type Vaues for SDMF and MDMF." Digest of Technical Information. Bellcore, May 1996: 11-16.

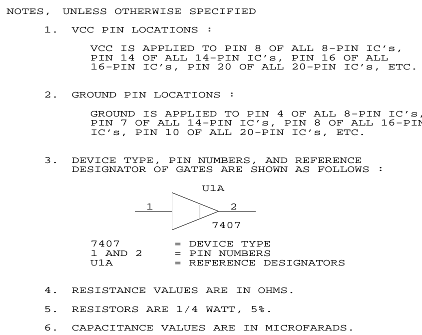
Messmer, Hans-Peter. The Indispensable PC Hardware Book – Your Questions Answered.1st. ed. Reading, MA.: Addison-Wesley Publishing Company, 1994.

Motorola. MC68HC705P9 Technical Data. MC68HC705P9/D, Rev. 3.0

Motorola. Motorola Communications Device Data. DL136/D, Rev. 3

KEYBOARD CALLER ID

REVISIONS		
REV	DESCRIPTION	DATE
0	SCHEMATIC FOR THE KEYBOARD CALLER ID DEVICE	4/12/97



NOTES, UNLESS OTHERWISE SPECIFIED

1. VCC PIN LOCATIONS :

   VCC IS APPLIED TO PIN 8 OF ALL 8-PIN IC's, PIN 14 OF ALL 14-PIN IC's, PIN 16 OF ALL 16-PIN IC's, PIN 20 OF ALL 20-PIN IC's, ETC.

2. GROUND PIN LOCATIONS :

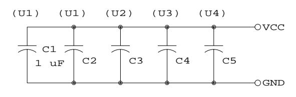
   GROUND IS APPLIED TO PIN 4 OF ALL 8-PIN IC's, PIN 7 OF ALL 14-PIN IC's, PIN 8 OF ALL 16-PIN IC's, PIN 10 OF ALL 20-PIN IC's, ETC.

3. DEVICE TYPE, PIN NUMBERS, AND REFERENCE DESIGNATOR OF GATES ARE SHOWN AS FOLLOWS :

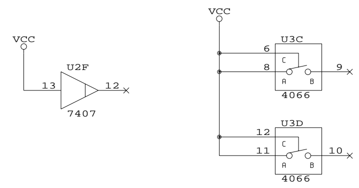
   7407 = DEVICE TYPE
   1 AND 2 = PIN NUMBERS
   U1A = REFERENCE DESIGNATORS

4. RESISTANCE VALUES ARE IN OHMS.
5. RESISTORS ARE 1/4 WATT, 5%.
6. CAPACITANCE VALUES ARE IN MICROFARADS.

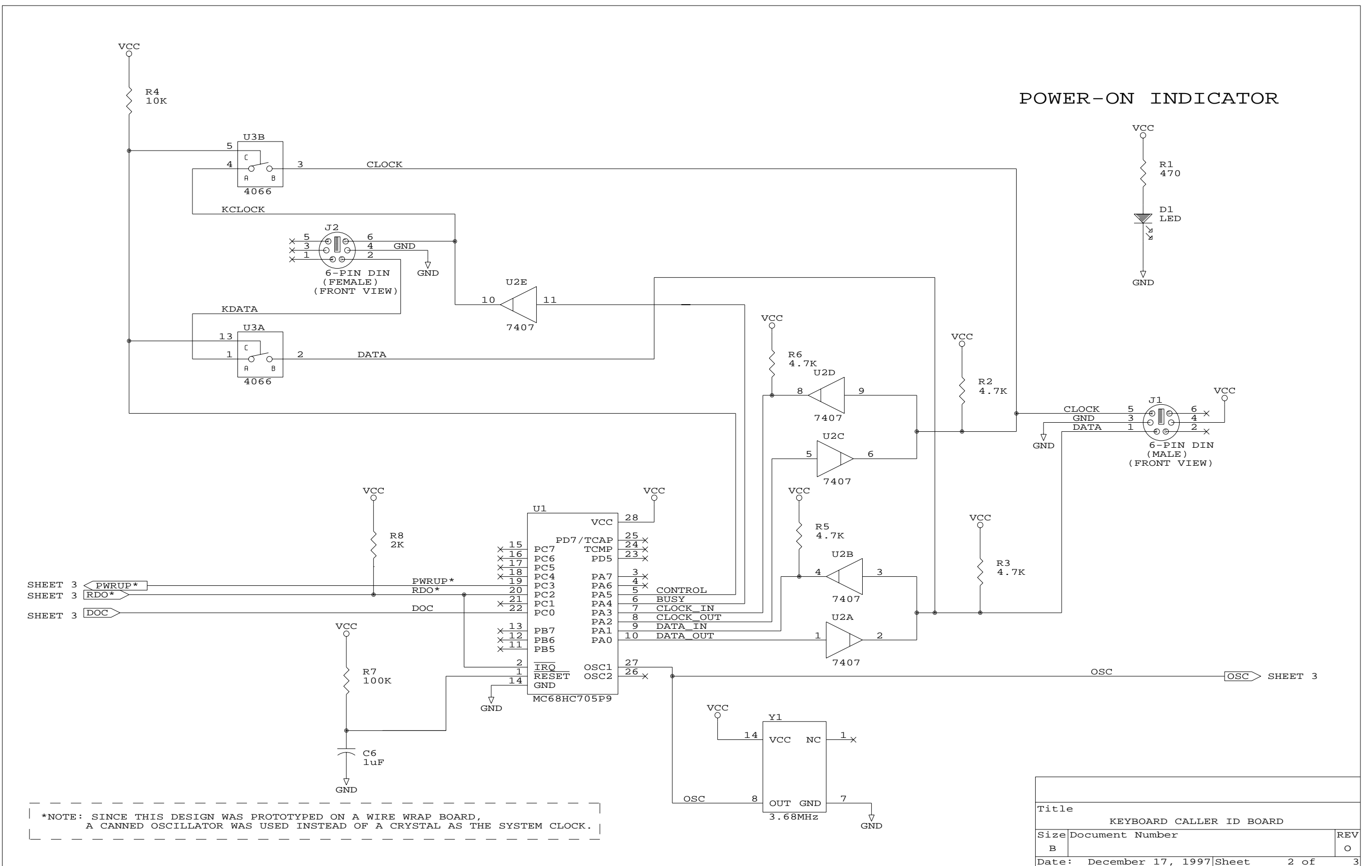
Decouple Caps for ICs as labeled.
All caps are 0.1 uF @ 50 V

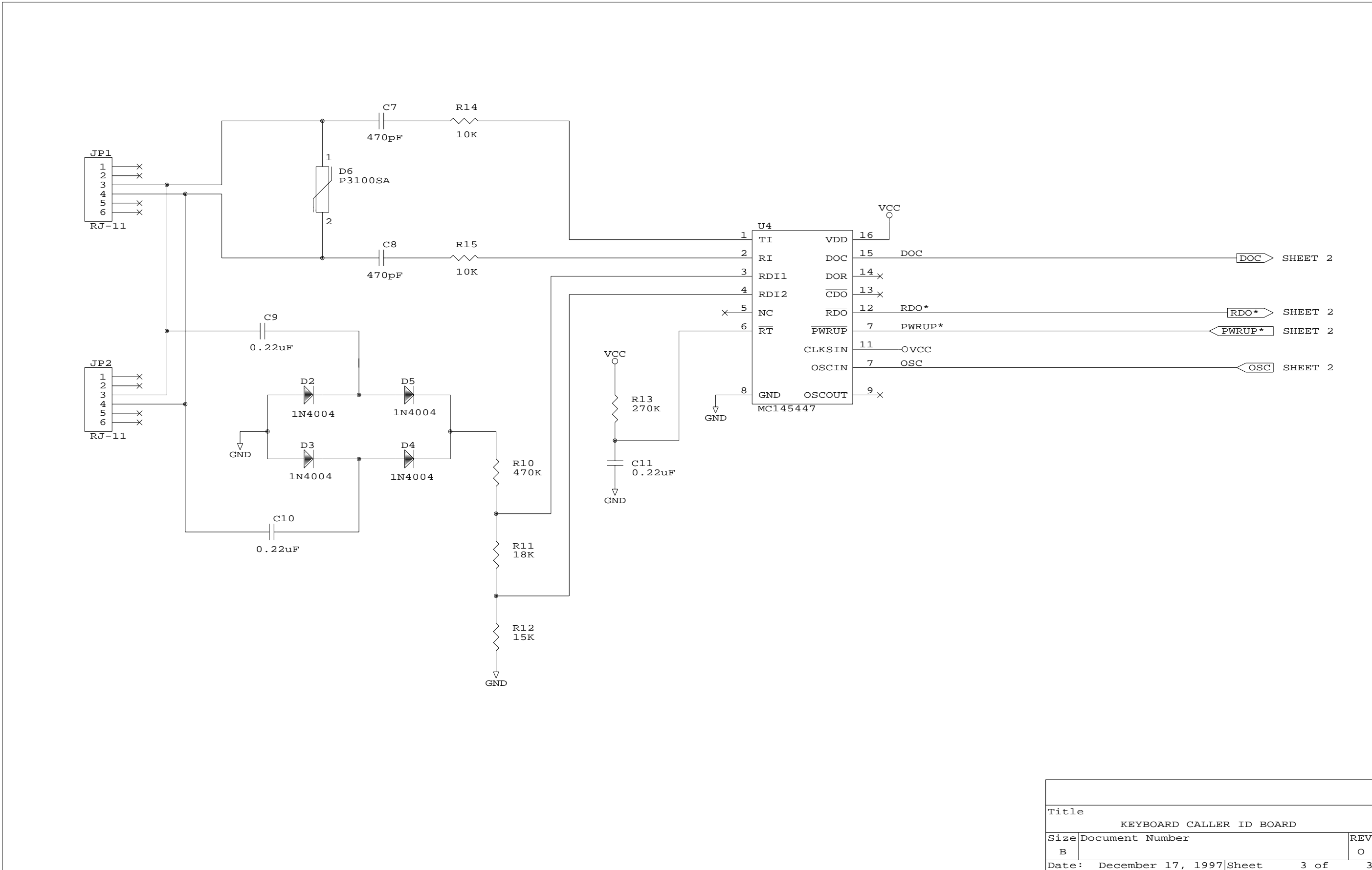


Spare Gates



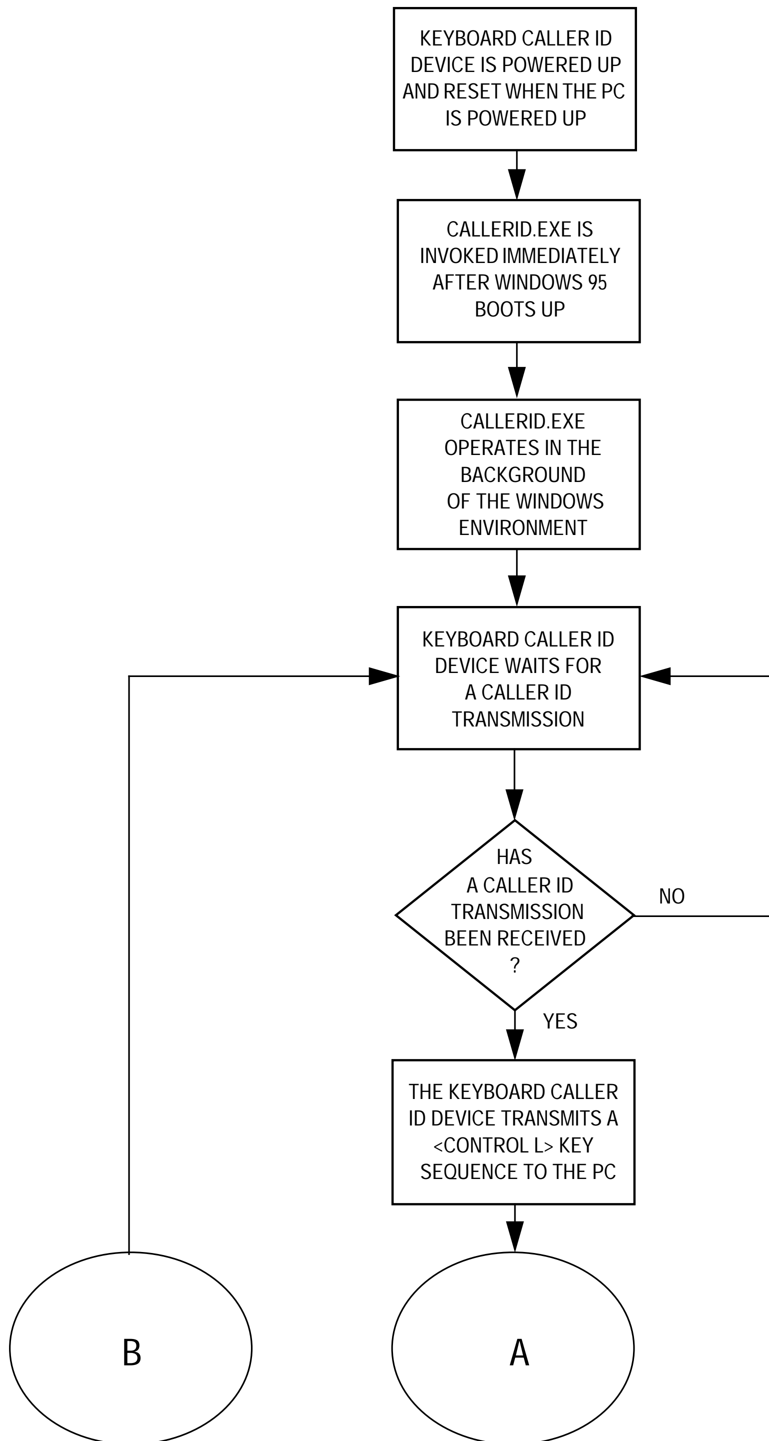
Title		
KEYBOARD CALLER ID BOARD		
Size B	Document Number	REV 0
Date: February 1, 1998	Sheet 1 of 3	

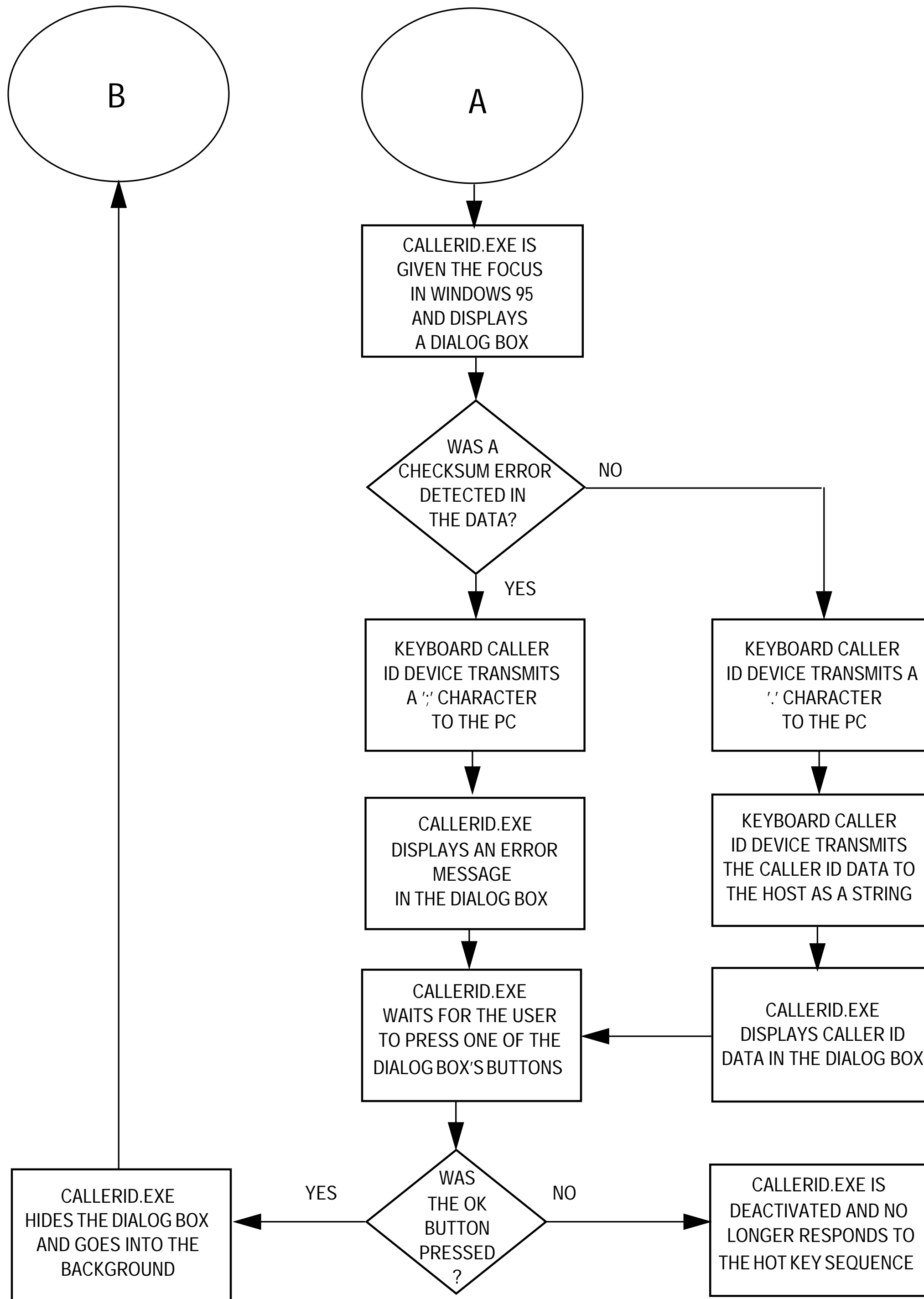

POWER-ON INDICATOR
VCC
R4
10K
U3B
4066
CLOCK
KCLOCK
J2
6-PIN DIN
(FEMALE)
(FRONT VIEW)
GND
U2E
7407
KDATA
U3A
DATA
R1
470
D1
LED
R6
4.7K
U2D
R2
4.7K
U2C
J1
6-PIN DIN
(MALE)
(FRONT VIEW)
U1
MC68HC705P9
PD7/TCAP
TCMP
PD5
PC7
PC6
PC5
PC4
PC3
PC2
PC1
PC0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PB7
PB6
PB5
IRQ
RESET
OSC1
OSC2
CONTROL
BUSY
CLOCK_IN
CLOCK_OUT
DATA_IN
DATA_OUT
R8
2K
R5
4.7K
U2B
R3
4.7K
U2A
SHEET 3
PWRUP*
RDO*
DOC
OSC
R7
100K
C6
1uF
Y1
NC
OUT
3.68MHz
*NOTE: SINCE THIS DESIGN WAS PROTOTYPED ON A WIRE WRAP BOARD,
A CANNED OSCILLATOR WAS USED INSTEAD OF A CRYSTAL AS THE SYSTEM CLOCK.
Title
KEYBOARD CALLER ID BOARD
Size
B
Document Number
REV
0
Date: December 17, 1997
Sheet 2 of 3



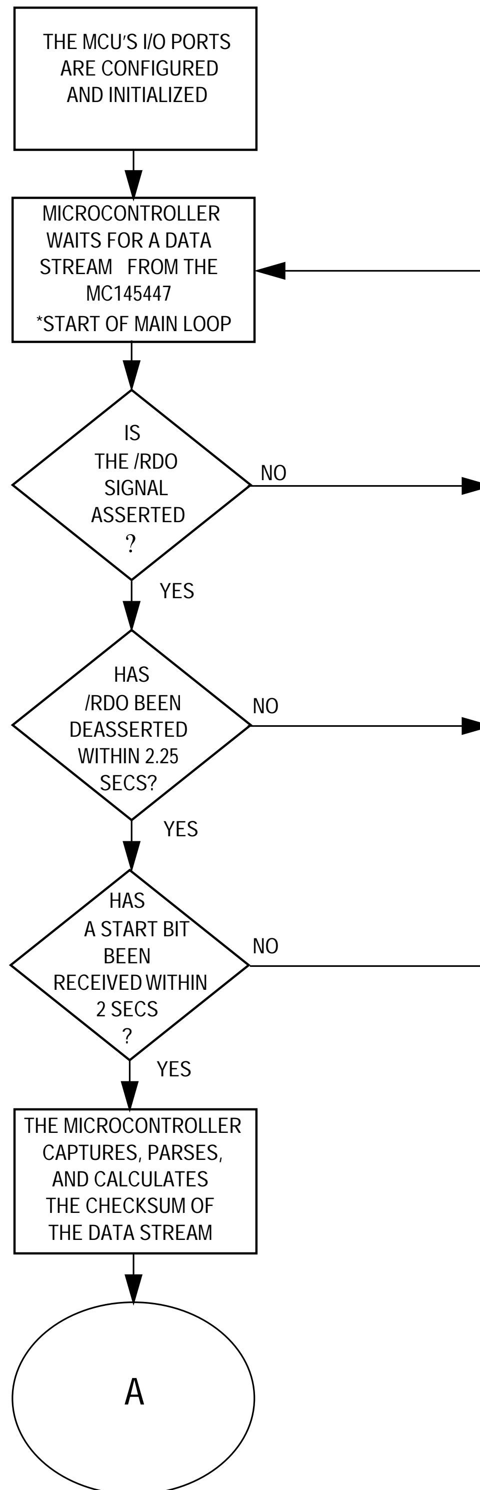
JP1
RJ-11
JP2
RJ-11
C7
470pF
R14
10K
D6
P3100SA
C8
470pF
R15
10K
C9
0.22uF
D2
1N4004
D5
1N4004
D3
1N4004
D4
1N4004
GND
C10
0.22uF
R10
470K
R11
18K
R12
15K
VCC
R13
270K
C11
0.22uF
U4
TI
RI
RDI1
RDI2
NC
RT
GND
VDD
DOC
DOR
CDO
RDO
PWRUP
CLKSIN
OSCIN
OSCOUT
MC145447
DOC
RDO*
PWRUP*
OSC
SHEET 2
Title
KEYBOARD CALLER ID BOARD
Size
B
Document Number
REV
O
Date: December 17, 1997
Sheet 3 of 3

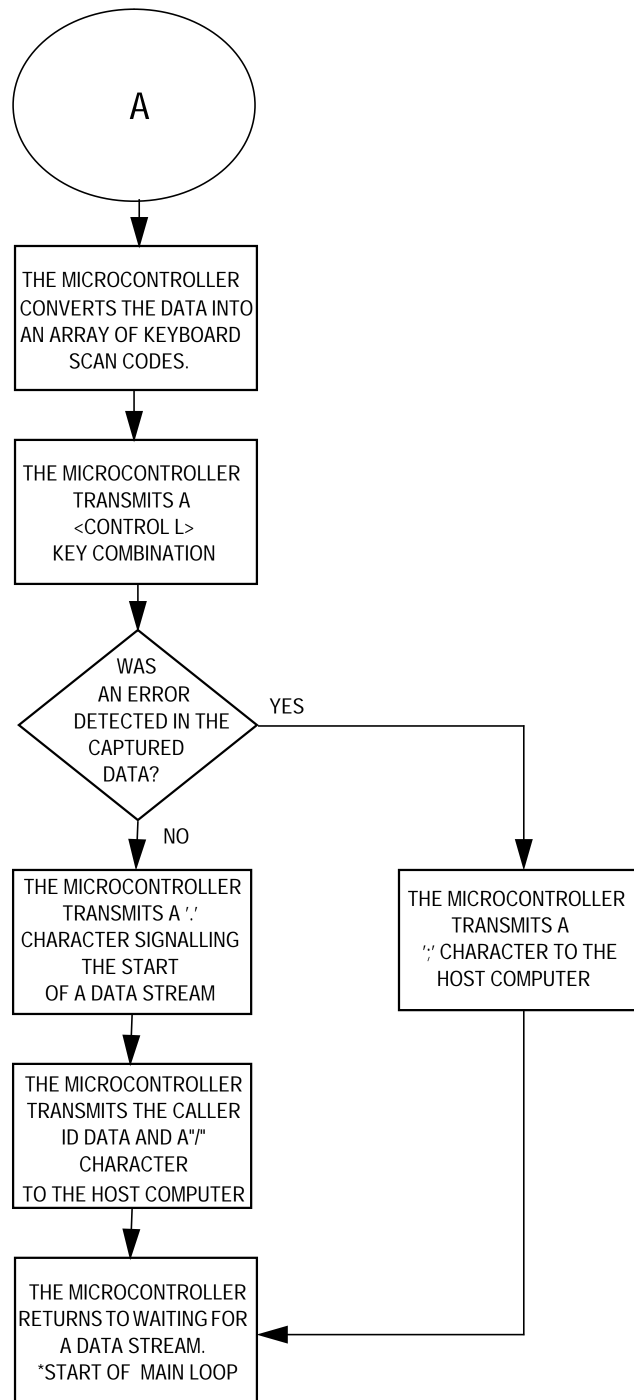
Appendix B — System Operation Flow Chart



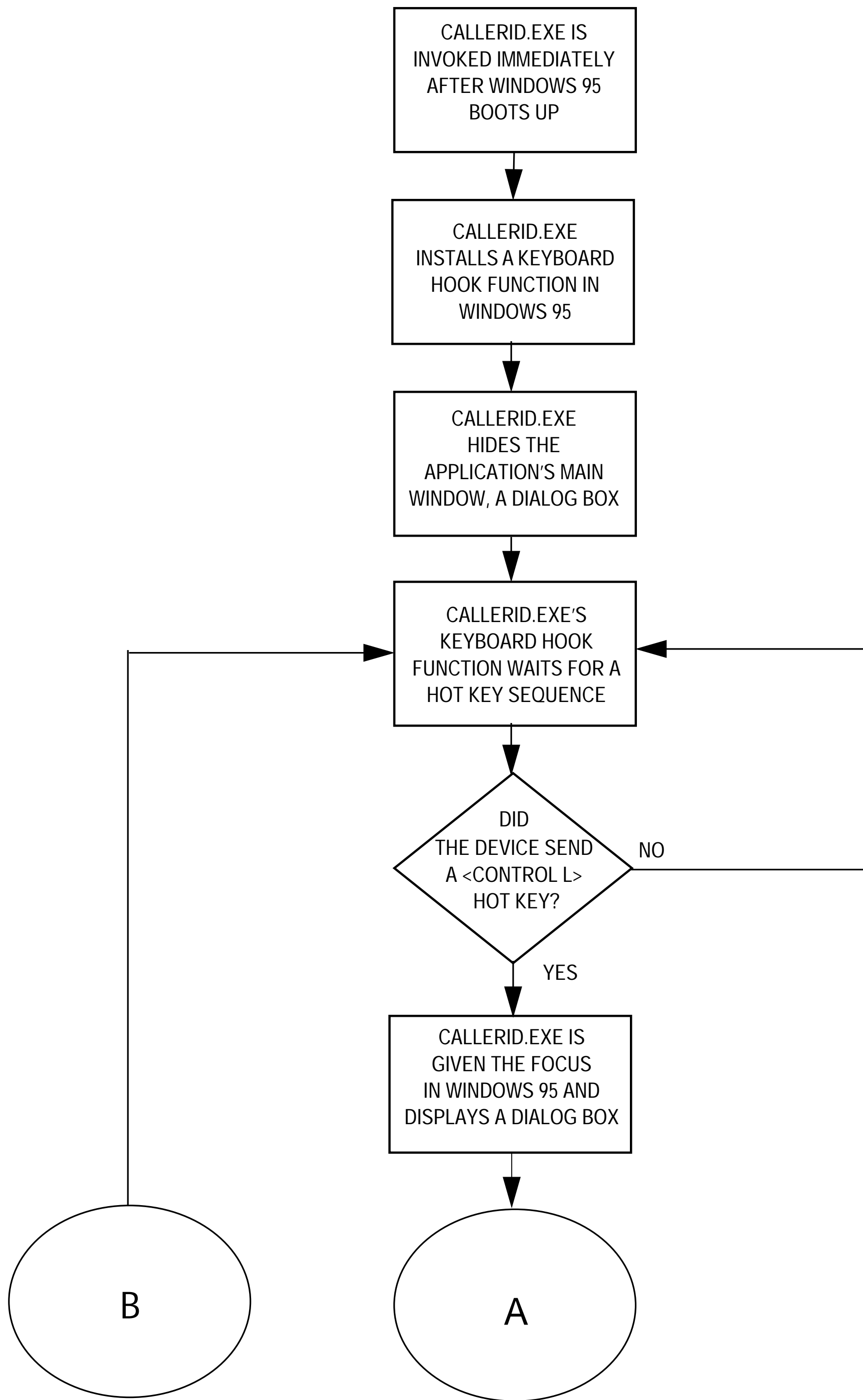

B
A
CALLERID.EXE IS GIVEN THE FOCUS IN WINDOWS 95 AND DISPLAYS A DIALOG BOX
WAS A CHECKSUM ERROR DETECTED IN THE DATA?
NO
YES
KEYBOARD CALLER ID DEVICE TRANSMITS A ';' CHARACTER TO THE PC
KEYBOARD CALLER ID DEVICE TRANSMITS A '.' CHARACTER TO THE PC
CALLERID.EXE DISPLAYS AN ERROR MESSAGE IN THE DIALOG BOX
KEYBOARD CALLER ID DEVICE TRANSMITS THE CALLER ID DATA TO THE HOST AS A STRING
CALLERID.EXE WAITS FOR THE USER TO PRESS ONE OF THE DIALOG BOX'S BUTTONS
CALLERID.EXE DISPLAYS CALLER ID DATA IN THE DIALOG BOX
CALLERID.EXE HIDES THE DIALOG BOX AND GOES INTO THE BACKGROUND
YES
WAS THE OK BUTTON PRESSED ?
NO
CALLERID.EXE IS DEACTIVATED AND NO LONGER RESPONDS TO THE HOT KEY SEQUENCE

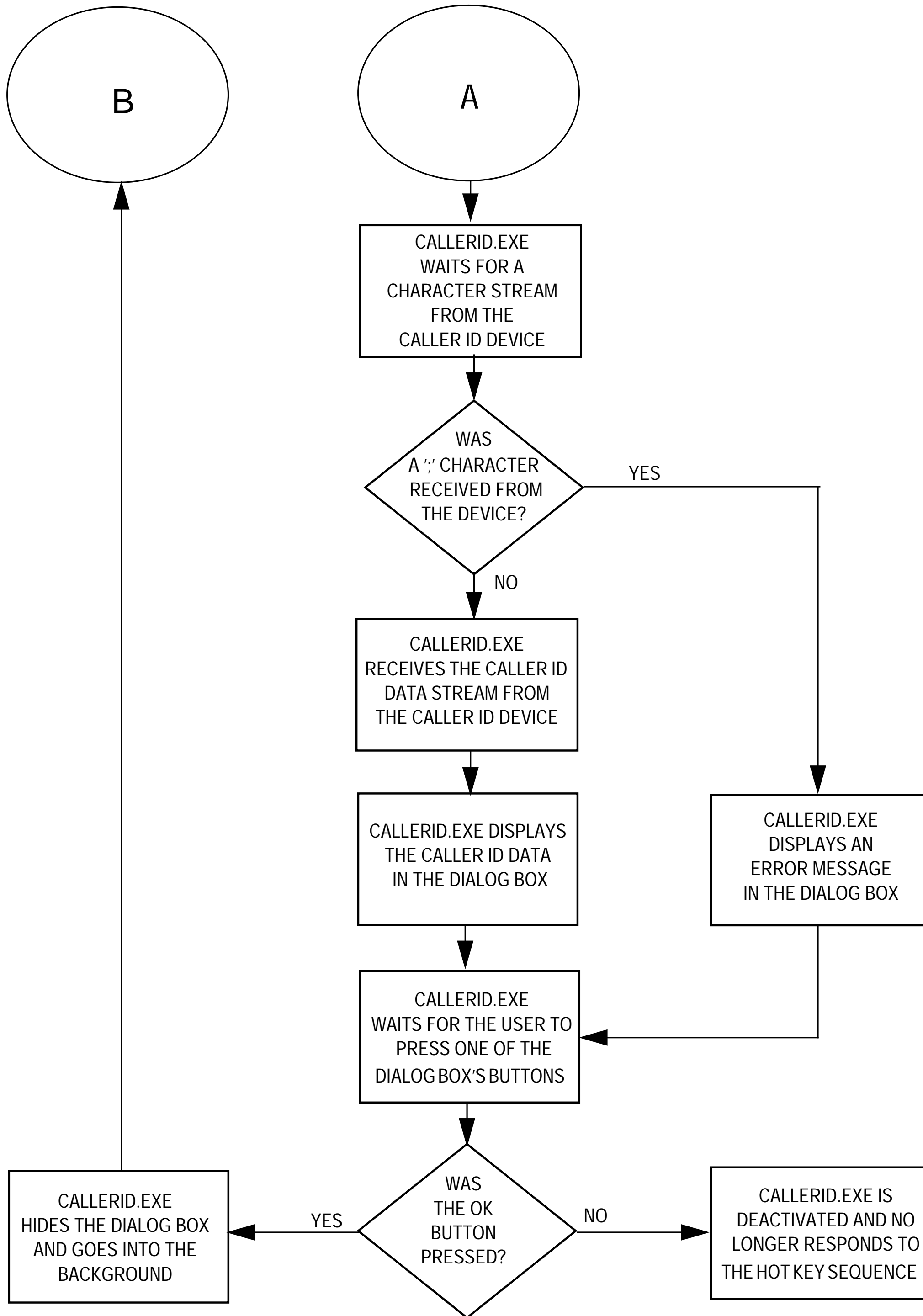
Appendix C — Keyboard Caller ID Device Firmware Flow Chart




A
THE MICROCONTROLLER CONVERTS THE DATA INTO AN ARRAY OF KEYBOARD SCAN CODES.
THE MICROCONTROLLER TRANSMITS A <CONTROL L> KEY COMBINATION
WAS AN ERROR DETECTED IN THE CAPTURED DATA?
YES
NO
THE MICROCONTROLLER TRANSMITS A '.' CHARACTER SIGNALLING THE START OF A DATA STREAM
THE MICROCONTROLLER TRANSMITS A ';' CHARACTER TO THE HOST COMPUTER
THE MICROCONTROLLER TRANSMITS THE CALLER ID DATA AND A"/" CHARACTER TO THE HOST COMPUTER
THE MICROCONTROLLER RETURNS TO WAITING FOR A DATA STREAM. *START OF MAIN LOOP

Appendix D — CALLERID.EXE Program Flow Chart





B
A
CALLERID.EXE WAITS FOR A CHARACTER STREAM FROM THE CALLER ID DEVICE
WAS A ';' CHARACTER RECEIVED FROM THE DEVICE?
YES
NO
CALLERID.EXE RECEIVES THE CALLER ID DATA STREAM FROM THE CALLER ID DEVICE
CALLERID.EXE DISPLAYS THE CALLER ID DATA IN THE DIALOG BOX
CALLERID.EXE DISPLAYS AN ERROR MESSAGE IN THE DIALOG BOX
CALLERID.EXE WAITS FOR THE USER TO PRESS ONE OF THE DIALOG BOX'S BUTTONS
CALLERID.EXE HIDES THE DIALOG BOX AND GOES INTO THE BACKGROUND
YES
WAS THE OK BUTTON PRESSED?
NO
CALLERID.EXE IS DEACTIVATED AND NO LONGER RESPONDS TO THE HOT KEY SEQUENCE

Appendix E — Keyboard Caller ID Device Firmware Source Code

```
********************************************************************************
; TITLE    CALLER ID
; USE      CALLER ID APP. NOTES.
; REVISION 1.0
; AUTHOR   Derrick B. Forte and Hai Nguyen
; GROUP
; DATE     04/21/97
; ASSEMBLER IASM05
;
********************************************************************************
*                                                                              *
*                          SYSTEM EQUATES                                      *
*                                                                              *
********************************************************************************

PORTA           EQU     $00      ;Port A data register
PORTB           EQU     $01      ;Port B data register
PORTC           EQU     $02      ;Port C data register
DDRA            EQU     $04      ;Port A data direction register
DDRB            EQU     $05      ;Port B data direction register
DDRC            EQU     $06      ;Port C data direction register
DDRAMASK        EQU     $F5      ;Port A data direction register mask
DDRBMASK        EQU     $FF      ;Port B data direction register mask
DDRCMASK        EQU     $F8      ;Port C data direction register mask
PORTAMASK       EQU     DDRAMASK ;Port A data register mask
PORTBMASK       EQU     DDRBMASK ;Port B data register mask
PORTCMASK       EQU     DDRCMASK ;Port C data register mask

********************************************************************************
*                                                                              *
*                          KEYBOARD EQUATES                                    *
*                                                                              *
********************************************************************************

CLOCK_OUT       EQU     2        ;Device keyboard clock output signal
CLOCK_IN        EQU     3        ;Device keyboard clock input signal
DATA_OUT        EQU     0        ;Device keyboard data output signal
DATA_IN         EQU     1        ;Device keyboard data input signal
BUSY            EQU     4        ;Keyboard busy signal
CONTROL         EQU     5        ;Keyboard enable/disable signal
RESEND          EQU     $FE      ;PC keyboard protocol resend command
ERROR           EQU     0        ;Error bit in the FLAG variable
RX_PARITY       EQU     0        ;Parity bit in the FLAG variable
PARITY          EQU     7        ;Received parity bit in the FLAG variable
```

```
********************************************************************************
*                                                                              *
*                             CALLER ID Equates                                *
*                                                                              *
********************************************************************************
DOC             EQU     0       ;Caller ID cooked data signal
RDO             EQU     2       ;Caller ID ring detect signal
PWRUP           EQU     3       ;Caller ID powerup signal
SDMF            EQU     $4      ;Valid message type parameter for SDMF
MDMF            EQU     $80     ;Valid message type parameter for MDMF
PERIOD          EQU     $49     ;Keyboard scan code for a period character
BACKSLASH       EQU     $4A     ;Keyboard scan code for a backslash character
CNTRL           EQU     $14     ;Keyboard scan code for the CONTROL key
SEMICOLON       EQU     $4C     ;Keyboard scan code for a ';' character

********************************************************************************
*                                                                              *
*                              Global Variables                                *
*                                                                              *
********************************************************************************
                ORG     $80    ;Start of global variable memory space

DATA            RMB     1      ;Storage variable for data that is to be
                               ;transmitted to or received from the keyboard
FLAG            RMB     1      ;Global function return flag.
TX_RESEND       RMB     1      ;Keyboard re-transmission variable
TEMP            RMB     1      ;Global temporary storage variable
TEMPA           RMB     1      ;Temporary storage variable for the accumlator
TEMPX           RMB     1      ;Temporary storage variable for the X register
RX_BUFFER       RMB     1      ;Data receiver variable
OUTERCNT        RMB     1      ;Generic delay counter variable
DATA_COUNT      RMB     1      ;Generic data counter variable
ERRORCD         RMB     1      ;Data acquisition error flag.
LNE_ERROR       RMB     1      ;Line error flag
INNERCNT        RMB     1      ;Generic delay counter variable
SZCNT           RMB     1      ;Counter variable for number of seizure set
COUNTER         RMB     1      ;General counter variable
COUNTER1        RMB     1      ;General counter variable 1
WORD            RMB     1      ;Current data word read/received
HIGH_NIBBLE     RMB     1      ;High nibble of data to be sent to the PC
LOW_NIBBLE      RMB     1      ;Low nibble of data to sent to the PC
MSGTYPE         RMB     1      ;Caller ID message type variable
MSGLEN          RMB     1      ;Caller ID message length variable
RAW_S_BUF       RMB     40     ;Start of caller ID data buffer
CHKSUM          RMB     1      ;Caller ID checksum variable
TIMECNT         RMB     1      ;Counter variable for timing loop
RAW_PT          RMB     1      ;Pointer to current RAW caller ID data in buffer
MBCNT           EQU     SZCNT    ;Mark bit counter variable
WBCNT           EQU     COUNTER1 ;Word bit counter variable
```

```
                ORG     $100            ;Start of Caller ID program
BEGIN           EQU     *
                sei
                jsr     INITIALIZE      ;Initialize the MCU's I/O ports
                bclr    PWRUP,PORTC     ;Assert the MC145447's PWRUP pin to
                                        ;power the device up.

******************************************************************************
*                                                                            *
*                       MAIN PROGRAM LOOP                                    *
*                                                                            *
******************************************************************************

MAIN            EQU   *

RDOWAITL        brset RDO,PORTC,RDOWAITL  ;Wait for RDO* signal to be asserted.
                jsr   RDOWAITH            ;Wait a maximum of 2.25 seconds for
                                          ;the RDO* signal to be deasserted.
                jsr   DOCWAIT             ;Wait up to 2 seconds for DOC start bit.
                clr   CHKSUM              ;Clear checksum variable.
                clr   LNE_ERROR           ;Clear line error flag.
                jsr   GETWORD             ;Get caller ID message type parameter
                                          ;byte.
                tst   ERRORCD             ;Check to see if the message type byte
                                          ;was received properly. If not send a
                                          ;line error message to the PC.
                beq   GOOD_TYPE
                jmp   MAIN
GOOD_TYPE       lda   WORD                ;Check to see if a SDMF valid message
                cmp   #SDMF               ;type parameter was received.
                beq   STORE_TYPE          ;Check to see if a MDMF valid message
                cmp   #MDMF               ;type parameter was received.
                beq   STORE_TYPE          ;If an invalid message type parameter
                jmp   MAIN                ;is received, jump to the start.
STORE_TYPE      jsr   UPDATECS            ;Update the message checksum
                                          ;calculation.
                lda   WORD                ;Store the message type parameter
                sta   MSGTYPE             ;byte.
                jsr   GETWORD             ;Get the caller ID message length byte.
                tst   ERRORCD             ;Check to see if an error occurred
                beq   GOOD_LENGTH         ;in receiving the caller ID message
                jmp   LINE_ERROR          ;length byte.
GOOD_LENGTH     jsr   UPDATECS            ;Update the message checksum
                lda   WORD                ;calculation.
                sta   MSGLEN              ;Store the message length byte
                clrx                      ;parameter.
MORE_DATA       jsr   GETWORD             ;Get the rest of the Caller ID data.
                tst   ERRORCD
                beq   STORE_CID_DATA
                jmp   LINE_ERROR
STORE_CID_DATA  jsr   UPDATECS            ;Update the checksum calculation.
```

```
                lda   WORD
                sta   RAW_S_BUF,x         ;Store the data bytes in a buffer.
                incx
                cpx   MSGLEN              ;Loop until the number of data bytes
                bne   MORE_DATA           ;received equals the value in the
                                          ;MSGLEN variable
                jsr   GETWORD             ;Get the message's checksum.
                tst   ERRORCD             ;Check for an error in receiving the
                beq   GOOD_CHECKSUM       ;checksum byte.
                jmp   LINE_ERROR
GOOD_CHECKSUM   lda   CHKSUM              ;Form 2' complement of checksum value
                coma                      ;calculated by above chksum summation.
                inca
                cmp   WORD                ;Compare the calculated checksum to
                beq   CSMATCH             ;received checksum. If they are equal
                jmp   LINE_ERROR          ;continue, otherwise send an error
                                          ;message to the PC.
CSMATCH         sta   CHKSUM              ;Store the checksum value.
                jsr   SEND_2_PC           ;Send data to the PC.

                bra   MAIN

******************************************************************************
* Function Name: INITIALIZE                                                  *
* Purpose: Initializes the MCU's I/O Ports                                   *
*                                                                            *
******************************************************************************

INITIALIZE      lda     #PORTAMASK        ;Set bits 1 & 3 of port A low,
                sta     PORTA             ;Set all other bits high.
                lda     #DDRAMASK         ;Set bits 1 & 3 of port A as inputs,
                sta     DDRA              ;Set all other bits as outputs.
                lda     #PORTBMASK        ;Set all port B bits high.
                sta     PORTB
                lda     #DDRBMASK         ;Set all port B bits as outputs.
                sta     DDRB
                lda     #PORTCMASK        ;Set bit 3 of port C high,
                sta     PORTC             ;set all other bits low.
                sta     DDRC              ;Set bit 3 of port C as an output,
                                          ;set all other bits as inputs.
                rts
```

```
******************************************************************************
*                                                                            *
* Function Name: RDOWAITH                                                    *
* Purpose: Checks for the proper deassertion of the RDO* signal.             *
*          If the RDO* is not deasserted within 2.25 seconds jump to the line *
*          error handling routine. Otherwise return to the calling function.  *
*                                                                            *
******************************************************************************

RDOWAITH    ldx   #9                   ;If the RDO signal is not deasserted,
RDO_LOOP    jsr   W1_4SEC              ;after 2.25 seconds jump to the line
            brset RDO,PORTC,RDO_EXIT   ;error function. Otherwise return.
            decx
            bne   RDO_LOOP
RDO_ERROR   jmp   MAIN
RDO_EXIT    rts

******************************************************************************
*                                                                            *
* Function Name: DOCWAIT                                                     *
* Purpose: Wait a maximum of 2 seconds after the deassertion of RDO* for the *
*          first data start bit. If the bit is not received, jump to the line *
*          error handling routine. Otherwise return to the calling function.  *
*                                                                            *
******************************************************************************

DOCWAIT     ldx    #$C8
            sta    OUTERCNT
CDOCILP     ldx    #$C8
            sta    INNERCNT
DOCILP      brclr  0,PORTC,EXITDOC
            jsr    W50US
            dec    INNERCNT
            bne    DOCILP
            dec    OUTERCNT
            bne    CDOCILP
            jmp    MAIN
EXITDOC     rts

******************************************************************************
*                                                                            *
* Function Name: UPDATECS                                                    *
* Purpose: Calculates the checksum for the incoming caller ID data.          *
*                                                                            *
******************************************************************************

UPDATECS    lda   CHKSUM
            add   WORD
            sta   CHKSUM
            rts
```

```
******************************************************************************
*                                                                            *
* Function Name: GETWORD                                                     *
* Purpose: Get a caller ID data word that includes a start bit, 8 data bits,  *
*          and a stop bit. If an error occurs in reading the word, the ERRORCD *
*          is incremented.                                                   *
*                                                                            *
******************************************************************************

GETWORD      stx   TEMPX                  ;Save X register
             clr   MBCNT
WAITSB       jsr   GETZERO                ;Get start bit.
             tst   ERRORCD                ;Start bit is successfully received if
             bne   STARTBNR               ;the ERRORCD variable is clear.
             brclr DOC,PORTC,STARTBR      ;Check for a spurious start bit.
STARTBNR     inc   MBCNT
             lda   MBCNT                  ;Allow for up to 10 mark bits between
             cmp   #$A                    ;two data words.
             bne   WAITSB
             bra   EXITGW                 ;Error code is set to non-zero byO
                                           GETZERO routine
STARTBR      clr   WORD
             clr   WBCNT
             jsr   W400US
MOREWB       jsr   W830US
             lsr   WORD                   ;Shift in data LSB.
             brclr DOC,PORTC,ZEROBIT
             bset  7,WORD
ZEROBIT      inc   WBCNT
             lda   WBCNT
             cmp   #8                     ;Get 8 bits making up a byte.
             bne   MOREWB
             jsr   GETONE                 ;error code=0 if stop bit for received
                                           word exit
EXITGW       ldx   TEMPX
             rts

******************************************************************************
*                                                                            *
* Function Name: GETZERO                                                     *
* Purpose: Wait up to 840 usec for a zero bit. If the a zero bit is not       *
*          received within the timeout period, increment the ERRORCD variable. *
*                                                                            *
******************************************************************************
```

```
GETZERO      lda   #$54
             clr   ERRORCD
WAITGZ       brclr DOC,PORTC,EXITGZ
             jsr   W10US
             deca
             bne   WAITGZ
             inc   ERRORCD
EXITGZ       rts
********************************************************************************
*                                                                              *
* Function Name: GETONE                                                        *
* Purpose: Wait up to 840 usec for DOC to go high. If a one bit is not         *
*          received within the timeout period, increment the ERRORCD variable. *
*                                                                              *
********************************************************************************

GETONE       lda   #$54
             clr   ERRORCD
WAITGO       brset DOC,PORTC,EXITGO    ;Exit loop if DOC pin is high.
             jsr   W10US               ;Wait 10 usec.
             deca                      ;$53=83, continue waiting for DOC going
             bne   WAITGO              ;high if 840 usec have not passed.
             inc   ERRORCD             ;Increment the ERRORCD variable if a
EXITGO       rts                       ;timeout occurred.

********************************************************************************
*                                                                              *
* Function Name: LINE_ERROR                                                    *
* Purpose: Sends a message the PC informing it that a line error has occurred. *
*                                                                              *
********************************************************************************

LINE_ERROR   inc   LNE_ERROR
             jsr   SEND_2_PC
             jsr   GOWAIT              ;Go to the GOWAIT function to reset the
             rts                       ;state of the program.

********************************************************************************
*                                                                              *
* Function Name: GOWAIT                                                        *
* Purpose: Resets the stack pointer, disables the caller ID device and         *
*          jumps to the beginning of the program.                              *
*                                                                              *
********************************************************************************

GOWAIT       rsp                        ;Reset stack pointer.
             bset  BUSY,PORTA           ;Make sure that the keyboard is connected
             bset  CONTROL,PORTA        ;to the PC.
             jmp   MAIN                 ;Return to the start of the program.
```

```
*****************************************************************************
*                                                                           *
* Function Name: SEND_BYTE                                                  *
* Purpose: Sends a byte to the PC using the IBM AT keyboard to keyboard port *
*          protocol.                                                        *
*                                                                           *
*****************************************************************************

SEND_BYTE       lda     DATA                      ;Save the data to be
                sta     TX_RESEND                 ;transmitted in case
                                                  ;it has to be retransmitted.
                jsr     SEND                      ;Send the byte to the PC.
                brclr   ERROR,FLAG,EXIT_SEND_BYTE ;If no error occurred in the
                jsr     ERROR_DELAY               ;transmission exit the
                jsr     RECEIVE                   ;function. Otherwise prepare
                brclr   ERROR,FLAG,CHECK_FOR_$FE  ;to receive a $FE from the PC.
                jmp     GOWAIT                    ;If a $FE was not received or
CHECK_FOR_$FE   lda     #RESEND                   ;if an error occurred while
                cmp     RX_BUFFER                 ;receiving a $FE reset the
                beq     RESEND_BYTE               ;state of the program.
                jmp     GOWAIT                    ;Otherwise resend the
RESEND_BYTE     lda     TX_RESEND                 ;original data byte.
                sta     DATA                      ;If the retransmission failed
                jsr     SEND                      ;reset the state of the
                brclr   ERROR,FLAG,EXIT_SEND_BYTE ;program.
                jmp     GOWAIT
EXIT_SEND_BYTE  rts

****************************************************************************
*                                                                          *
* Function Name: SEND_2_PC                                                 *
* Purpose: Sends scan codes for the caller ID data to the PC.              *
*                                                                          *
****************************************************************************

SEND_2_PC       jsr     WAIT_4_PC                 ;Wait for no activity on the
                                                  ;keyboard lines for 5
                                                  ;character times.
                bclr    CONTROL,PORTA             ;Disconnect the keyboard from
                bclr    BUSY,PORTA                ;the PC.
                lda     #$14                      ;Send a "CONTROL L" key
                sta     DATA                      ;sequence to the PC to start
                jsr     SEND_BYTE                 ;the software on the host PC.
                jsr     W1_10SEC
                lda     #$4B
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     #$F0
                sta     DATA
                jsr     SEND_BYTE
```

```
                jsr     W1_10SEC
                lda     #$4B
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     #$F0
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     #$14
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_4SEC
                jsr     W1_4SEC
                jsr     W1_4SEC
                jsr     W1_4SEC                    ;If a line error occurred
                tst     LNE_ERROR                  ;send a semicolon.
                beq     SEND_MESSAGE
                lda     #SEMICOLON
                sta     DATA
                jsr     SEND_BYTE
                jmp     SEND_2_PC_EXIT
SEND_MESSAGE    lda     #PERIOD                    ;Send a message start
                sta     DATA                       ;delimiter to the PC.
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     MSGTYPE                    ;Send the message type byte
                jsr     CONVERT_DATA               ;to the PC.
                lda     HIGH_NIBBLE
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     LOW_NIBBLE
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     MSGLEN                     ;Send message length parameter
                jsr     CONVERT_DATA               ;to the PC
                lda     HIGH_NIBBLE
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     LOW_NIBBLE
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                clr     DATA_COUNT
DATA_LOOP       ldx     DATA_COUNT
                lda     RAW_S_BUF,X                ;Send the Caller ID data to
                jsr     CONVERT_DATA               ;the PC.
                lda     HIGH_NIBBLE
```

```
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     LOW_NIBBLE
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                inc     DATA_COUNT
                lda     DATA_COUNT
                cmp     MSGLEN
                bne     DATA_LOOP
                lda     CHKSUM                   ;Send the checksum to the PC
                jsr     CONVERT_DATA
                lda     HIGH_NIBBLE
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     LOW_NIBBLE
                sta     DATA
                jsr     SEND_BYTE
                jsr     W1_10SEC
                lda     #BACKSLASH               ;Send the backslash character
                sta     DATA                     ;to PC to serve as an end delimiter.
                jsr     SEND_BYTE
                jsr     W1_10SEC
SEND_2_PC_EXIT  bset    BUSY,PORTA               ;Reconnect the keyboard to the
                bset    CONTROL,PORTA            ;PC.
                rts

********************************************************************************
*                                                                              *
*  Function Name: CONVERT_DATA                                                 *
*  Purpose: Converts Caller ID parameter and data into scan codes for          *
*           transmission to the PC through its keyboard interface port.        *
*                                                                              *
********************************************************************************

CONVERT_DATA    stx     TEMPX
                sta     TEMP
                and     #$0F                     ;Mask out the upper nibble of
                                                 ;the byte to be processed.
                tax                              ;Get the scan code for the lower
                lda     SCAN_CODE_TABLE,X        ;nibble and store it.
                sta     LOW_NIBBLE
                ldx     TEMP                     ;Shift the data byte to the
                lsrx                             ;right four times and get the
                lsrx                             ;scan code for the upper
                lsrx                             ;nibble.
                lsrx
                lda     SCAN_CODE_TABLE,X
                sta     HIGH_NIBBLE
                ldx     TEMPX
                rts
```

```
*******************************************************************************
*                                                                             *
*  Function Name: SEND FUNCTION                                               *
*  Purpose: Transmits a scan code to the host PC's keyboard interface port.   *
*                                                                             *
*******************************************************************************

SEND        clr       TEMP                        ;Clear the parity check
            clr       FLAG                        ;Clear the return flag
            bset      CLOCK_OUT,PORTA             ;Initialize the keyboard's
                                                  ;clock signal.
            bset      DATA_OUT,PORTA              ;Initialize the keyboard's
                                                  ;data signal.
            ldx       #8                          ;Transmit eight data bits.
            bclr      DATA_OUT,PORTA              ;Clock in the start bit.
            jsr       HALF_CLOCK
            bclr      CLOCK_OUT,PORTA
            jsr       FULL_CLOCK
            bset      CLOCK_OUT,PORTA             ;If the PC pulls the clock
            jsr       HALF_CLOCK                  ;line low abort the trans-
            brclr     CLOCK_IN,PORTA,SEND_ERROR   ;mission, and set the error
SEND_BIT    ror       DATA                        ;flag.
            bcs       SEND_ONE
            bclr      DATA_OUT,PORTA
            bra       SEND_DATA
SEND_ONE    bset      DATA_OUT,PORTA              ;If the PC pulls the data line
            brclr     DATA_IN,PORTA,SEND_ERROR    ;low while a high bit is being
            inc       TEMP                        ;transmitted, abort the trans-
SEND_DATA   jsr       HALF_CLOCK                  ;mission, and set the error
            bclr      CLOCK_OUT,PORTA             ;flag. Otherwise transmit the
            jsr       FULL_CLOCK                  ;bit.
            bset      CLOCK_OUT,PORTA
            jsr       HALF_CLOCK
            brclr     CLOCK_IN,PORTA,SEND_ERROR
            decx
            bne       SEND_BIT
            ror       TEMP                        ;Calculate and send the parity
            bcc       PARITY_ONE                  ;bit.
            bclr      DATA_OUT,PORTA
            bra       SEND_PARITY
PARITY_ONE  bset      DATA_OUT,PORTA
            brclr     DATA_IN,PORTA,SEND_ERROR
SEND_PARITY jsr       HALF_CLOCK
            bclr      CLOCK_OUT,PORTA
            jsr       FULL_CLOCK
            bset      CLOCK_OUT,PORTA
            jsr       HALF_CLOCK
            brclr     CLOCK_IN,PORTA,SEND_ERROR
            bset      DATA_OUT,PORTA
            brclr     DATA_IN,PORTA,SEND_ERROR
```

```
            jsr         HALF_CLOCK
            bclr        CLOCK_OUT,PORTA
            jsr         FULL_CLOCK
            bset        CLOCK_OUT,PORTA
            ldx         #3
WAIT_4_BUSY brclr       CLOCK_IN,PORTA,PC_BUSY     ;Allow 100uS for the PC
            jsr         FULL_CLOCK                 ;to pull the clock line low
            decx                                   ;after transmitting a scan
            bne         WAIT_4_BUSY                ;code. If this event does not
            bra         SEND_ERROR                 ;occur an error has occurred.
PC_BUSY     ldx         #$C                        ;Allow a maximum of 500uS
STILL_BUSY  brset       CLOCK_IN,PORTA,CHECK_DATA  ;for the clock line to go
            jsr         FULL_CLOCK                 ;high. If this timeout is
            decx                                   ;exceeded an error has
            bne         STILL_BUSY                 ;occurred.
            bra         SEND_ERROR
CHECK_DATA  brset       DATA_IN,PORTA,SEND_EXIT    ;The PC will pull the data
SEND_ERROR  inc         FLAG                       ;line low if a transmission error.
SEND_EXIT   bset        CLOCK_OUT,PORTA            ;If an error occurs set the
                                                    error flag.
            bset        DATA_OUT,PORTA             ;Otherwise return a zero in
            rts                                    ;the function return flag.

*******************************************************************************
*                                                                             *
*  Function Name: RECEIVE                                                     *
*  Purpose: Receives an AT keyboard protocol resend command from the PC in   *
*           the event of a transmission error.                                *
*                                                                             *
*******************************************************************************


RECEIVE     clr          DATA                     ;Initialize all function
            clr          FLAG                     ;variables.
            clr          TEMP
            bset         DATA_OUT,PORTA           ;Initialize the keyboard data
            bset         CLOCK_OUT,PORTA          ;and clock signals.
            ldx          #$9
            bclr         CLOCK_OUT,PORTA          ;Clock in the start bit.
            jsr          FULL_CLOCK
GET_BITS    bset         CLOCK_OUT,PORTA          ;Clock in 8 data bits and
            jsr          HALF_CLOCK               ;the parity bit.
            brclr        DATA_IN,PORTA,DATA_LO
            cpx          #$01
            beq          HIGH_BIT
            inc          TEMP
                         HIGH_BIT    sec
            bra          STORE
```

```
DATA_LO      clc
STORE        ror         DATA
             jsr         HALF_CLOCK
             bclr        CLOCK_OUT,PORTA
             jsr         FULL_CLOCK
             decx
             bne         GET_BITS
             rol         DATA
             bset        CLOCK_OUT,PORTA
             bcc         CLR_PARITY
             bset        PARITY,TEMP
             bra         STOP
CLR_PARITY   bclr        PARITY,TEMP
STOP         jsr         HALF_CLOCK                ;Check for a stop bit.
             brclr       DATA_IN,PORTA,RCV_ERROR
             bclr        DATA_OUT,PORTA
             jsr         HALF_CLOCK
             bclr        CLOCK_OUT,PORTA
             jsr         FULL_CLOCK
             brclr       PARITY,TEMP,TST_PARITY    ;Test for the correct parity.
             brset       RX_PARITY,TEMP,RCV_ERROR ;If a parity error occurred,
             bra         RCV_EXIT                  ;increment the error flag.
TST_PARITY   brset       RX_PARITY,TEMP,RCV_EXIT
RCV_ERROR    inc         FLAG
RCV_EXIT     bset        CLOCK_OUT,PORTA
             bset        DATA_OUT,PORTA
             rts

********************************************************************************
*                                                                              *
* Function Name: WAIT_4_PC                                                     *
* Purpose: This function waits for no activity on the keyboard clock line for  *
* five character times before allowing the device to transmit to the PC.       *
*                                                                              *
********************************************************************************

WAIT_4_PC    ldx         #$64
PCWAIT_LOOP brclr        CLOCK_IN,PORTA,WAIT_4_PC
             jsr         HALF_CLOCK
             decx
             bne         PCWAIT_LOOP
             rts
```

```
*******************************************************************************
*                                                                             *
*                           TIME DELAY ROUTINES                               *
*                                                                             *
*******************************************************************************

ERROR_DELAY lda         #$40
            bra         CLOCK_LOOP
FULL_CLOCK  lda         #7
            bra         CLOCK_LOOP
HALF_CLOCK  lda         #3
CLOCK_LOOP  deca
            bne         CLOCK_LOOP
            rts
*******************************************************************************
*                                                                             *
* Function Name: W10US                                                        *
* Purpose: 10 usec delay loop assuming an OSC1 clock of 3.68MHZ and           *
*          this routine is entered from a 'jsr' extended(6 cycles) instruction *
*          for a total of 18 instruction cycle (slightly less than 10 usec).   *
*                                                                             *
*******************************************************************************

W10US       nop                         ;2 cycles for each 'nop' instruction.
            nop
            nop
            rts                         ;6 cycles for rts instruction.


*******************************************************************************
*                                                                             *
* Function Name: W50US                                                        *
* Purpose: 25usec delay loop assuming an OSC1 clock of 3.68MHZ and            *
*          this routine is entered from a 'jsr' extended(6 cycles) instruction *
*          for a total of 92 instruction cycles.                              *
*                                                                             *
*******************************************************************************

W50US       lda   #$D                   ;2 cycles
CONTW50     deca                        ;3 cycles
            bne   CONTW50               ;3 cycles
            rts                         ;6 cycles
```

```
*******************************************************************************
*                                                                             *
* Function Name: W400US                                                       *
* Purpose: 400usec delay loop assuming an OSC1 clock of 3.68MHZ and this      *
*          routine is entered from a 'jsr' extended(6 cycles) instruction.    *
*                                                                             *
*******************************************************************************

W400US      stx   TEMPX
            ldx   #$8
LW400       jsr   W50US
            decx
            bne   LW400
            ldx   TEMPX
            rts

*******************************************************************************
*                                                                             *
* Function Name: W830US                                                       *
* Purpose: 830usec delay loop assuming an OSC1 clock of 3.68MHZ and this      *
*          routine is entered from a 'jsr' extended (6 cycles) instruction.   *
*                                                                             *
*******************************************************************************

W830US       stx   TEMPX               ;5 cycles
             ldx   #$F                 ;2 cycles
CONTW830     jsr   W50US               ;6 cycles
             nop                       ;2 cycles
             decx                      ;3 cycles
             bne   CONTW830            ;3 cycles
             nop                       ;2 cycles
             nop                       ;2 cycles
             ldx   TEMPX               ;4 cycles
             rts                       ;6 cycles

*******************************************************************************
*                                                                             *
* Function Name: W1_4SEC                                                      *
* Purpose: .25 sec delay loop assuming a OSC1 clock of 3.68MHZ and            *
*         this routine is entered from a 'jsr' extended(6 cycles) instruction *
*          for a total of 20,000 instruction cycles.(1.84 cycle = 1usec)      *
*                                                                             *
*******************************************************************************

W1_4SEC     lda   #$12                 ;2 cycles
            sta   OUTERCNT             ;4 cycles
OUTERLOOP   lda   #$FF                 ;2 cycles
            sta   INNERCNT             ;4 cycles
```

```
CONTW1/4     jsr   W50US               ;92 cycles
             dec   INNERCNT            ;5 cycles
             bne   CONTW1/4            ;3 cycles
             dec   OUTERCNT            ;5 cycles
             bne   OUTERLOOP           ;3 cycles
             rts

*************************************************************************************
*                                                                                   *
* Function Name: W1_10SEC                                                           *
* Purpose: .1 sec delay loop assuming an OSC1 clock of 3.68MHZ and this             *
*          is entered from a 'jsr' extended (6 cycles) instruction.                 *
*                                                                                   *
*************************************************************************************

W1_10SEC     lda   #$4                 ;2 cycles
             sta   OUTERCNT            ;4 cycles
W1_10LOOP    lda   #$FF                ;2 cycles
             sta   INNERCNT            ;4 cycles
CONTW1/10    jsr   W50US               ;92 cycles
             dec   INNERCNT            ;5 cycles
             bne   CONTW1/10           ;3 cycles
             dec   OUTERCNT            ;5 cycles
             bne   W1_10LOOP           ;3 cycles
             rts


*************************** SCAN CODE TABLE *****************************************

                 ORG     $700

SCAN_CODE_TABLE  FCB     $45                      ;Scan code for "0"
                 FCB     $16                      ;Scan code for "1"
                 FCB     $1E                      ;Scan code for "2"
                 FCB     $26                      ;Scan code for "3"
                 FCB     $25                      ;Scan code for "4"
                 FCB     $2E                      ;Scan code for "5"
                 FCB     $36                      ;Scan code for "6"
                 FCB     $3D                      ;Scan code for "7"
                 FCB     $3E                      ;Scan code for "8"
                 FCB     $46                      ;Scan code for "9"
                 FCB     $1C                      ;Scan code for "a"
                 FCB     $32                      ;Scan code for "b"
                 FCB     $21                      ;Scan code for "c"
                 FCB     $23                      ;Scan code for "d"
                 FCB     $24                      ;Scan code for "e"
                 FCB     $2B                      ;Scan code for "f"

                 ORG   $1FFE                      ;Beginning of code to execute
                 FDB   BEGIN                      ;after a reset.
                 END
```

Appendix F — CALLERID.EXE Source Code File

```
#include <afxwin.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define IDB_BUTTON1 100 // ID for main window's "OK" button.
#define IDB_BUTTON2 150// ID for main window's "Deactivate" button.


// Function prototype to install CallerID DLL.
extern "C" __declspec(dllimport) void WINAPI InstallHook(void);


// Declare the application class.
class CallerID : public CWinApp
{
   public:

      virtual BOOL InitInstance();
};


// Create the only instance of the application class.
CallerID PCCallerID;

// Declare the application's main window class.
class CallerIDWindow : public CFrameWnd
{
 CButton *OKbutton; // Pointer to the window's 'OK' button.
 CButton *Deactivatebutton; // Pointer to the window's 'Deactivate' button.
 CString Name; // Variable that holds the caller ID name string.
 CString Number; // Variable that holds the caller ID number string.
 CString Date_Time; // Variable that holds the caller ID time and date string.
 CString Date; // Variable holding the date string.
 CString Time; // Variable holding the time string.
 char RawData[200]; // Temporary storage space for raw data from the keyboard.
 BOOL StartByte_flag; // This flag marks the start of data acquisition on
                      // from the PC's keyboard interface port.
 BOOL Display_flag; // This flag is set when the message data is ready to be
                    // displayed.
 BOOL LineError_flag; // This flag is set when a line error has occurred.
 BOOL SDMF_flag; // This flag is set if the message received is in the SDMF
                 // format.
```

```
  public:

        CallerIDWindow();  // Main window constructor.
         ~CallerIDWindow(); // Main window destructor.

         void Get_MessageType();
        void Process_SDMF();
         void Process_MDMF();
        void Format_Data();
         afx_msg void Handle_OK_Button();
         afx_msg void Handle_Deactivate_Button();
         afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
         afx_msg void OnPaint();
         DECLARE_MESSAGE_MAP();
};


BEGIN_MESSAGE_MAP(CallerIDWindow,CFrameWnd)
      ON_BN_CLICKED(IDB_BUTTON1, Handle_OK_Button)
         ON_BN_CLICKED(IDB_BUTTON2, Handle_Deactivate_Button)
         ON_WM_PAINT()
         ON_WM_CHAR()
END_MESSAGE_MAP()


BOOL CallerID::InitInstance()
{
  // Create and hide the application's main window.
  m_pMainWnd = new CallerIDWindow();
  m_pMainWnd->ShowWindow(SW_HIDE);
  m_pMainWnd->UpdateWindow();

  // Install the keyboard board hook.
  InstallHook();

  return TRUE;
}


// Application's main window constructor.
CallerIDWindow::CallerIDWindow()
{
 // Initialize main window variables.
 StartByte_flag = FALSE;
 LineError_flag = FALSE;
 Display_flag = FALSE;

 // Create the main window.
 Create(NULL,"PC Caller ID", WS_OVERLAPPED, CRect(150,150,400,350));
```

```
 // Create the "OK" button.
 CRect r;
 GetClientRect(&r);
 OKbutton = new CButton();
 OKbutton -> Create("OK",
                   WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                   CRect(r.left+20,r.top+120,r.right-50,r.bottom-20),
                   this,IDB_BUTTON1);

 // Create the "Deactivate" button.
 Deactivatebutton = new CButton();
 Deactivatebutton -> Create("Deactivate",
                           WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,
                           CRect(r.left+130,r.top+120,r.right-20,r.bottom-20),
                           this,IDB_BUTTON2);
}


// Application main window destructor.
CallerIDWindow::~CallerIDWindow()
{
 delete OKbutton;
 delete Deactivatebutton;
}


void CallerIDWindow::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
 static int rawdataindex;
 int tempint;

 // If a semicolon was received a line error has occurred. Display the
 // "Line Error" message in the main window.
 if(!StartByte_flag && (nChar == ';'))
   {
    LineError_flag = TRUE;
    Display_flag = TRUE;
    Invalidate(TRUE);
   }

 // If a period character is received and StartByte_flag is not set,
 // set the StartByte_flag so that data acquisition can start.
 if(!StartByte_flag && (nChar == '.'))
   {
    StartByte_flag = TRUE; // Set the StartByte variable.
       RawData[0] = '\0'; // Initialize the raw data array.
       rawdataindex = 0;
       Invalidate(TRUE);
   }
```

```
 // If the StartByte_flag is set check the character that has been received.
 // If a backslash character is received this signifies the end of the data
 // stream from the device. Otherwise add the new character to the RawData
 // array.
 else
   {
       tempint = strlen(RawData);

       if((tempint > 0) && ((char)nChar == '/'))
         {
          RawData[rawdataindex] = '\0';

          Get_MessageType();

          if(SDMF_flag)
             Process_SDMF();
          else
             Process_MDMF();

          Format_Data();
          Invalidate(TRUE);
          }
       else
         {
          RawData[rawdataindex] = (char)nChar;
          rawdataindex++;
         }
   }
 }

void CallerIDWindow::OnPaint()
{
 // If the display flag is set, display an appropriate message.
 if(Display_flag)
   {
   // If a line error occurred, display the "Line Error" message.
    if(LineError_flag)
         {
          CPaintDC dc(this);
          CRect r;
          GetClientRect(&r);
          dc.DrawText("Line Error",-1,CRect(r.left+90, r.top+50,
                      r.right-20,r.bottom-90),DT_SINGLELINE);
                    }
```

```
        // If a valid message is received, display it.
        else
     {
        // Display the data.
        CPaintDC dc(this);
           dc.DrawText(Date, -1, CRect(30,10,200,400),DT_SINGLELINE);
           dc.DrawText(Time, -1, CRect(29,30,200,400),DT_SINGLELINE);
        dc.DrawText(Number, -1, CRect(10,50,200,200),DT_SINGLELINE);
           dc.DrawText(Name, -1, CRect(24,70,200,400),DT_SINGLELINE);

                                  }
     }
   else
     {
         // Display the "Receiving Data..." message while the caller ID data is
         // being acquired and processed.
         CPaintDC dc(this);
         CRect r;
         GetClientRect(&r);
         dc.DrawText("Receiving Data...",
                     -1,CRect(r.left+70, r.top+50, r.right-20,r.bottom-90),
                     DT_SINGLELINE);
      }

 // Draw the "OK" and "Deactivate" buttons on the main window.
 CRect r;
 GetClientRect(&r);
 OKbutton->MoveWindow(CRect(r.left+20, r.top+120, r.right-130, r.bottom-20));
 OKbutton->UpdateWindow();
 Deactivatebutton->MoveWindow(CRect(r.left+130, r.top+120, r.right-20,
                              r.bottom-20));
 Deactivatebutton->UpdateWindow();
}


void CallerIDWindow::Get_MessageType()
{
 int i;
 char tempstr[50];
 char *endptr;

 // Get the message type parameter from the data stream.
 strncpy(tempstr,&RawData[0],2);
 i = (UINT)strtoul(tempstr,&endptr,16);

 // If the message type parameter is equal to 4 return a one. Otherwise
 // return a zero.
 if(i==4)
    SDMF_flag = TRUE;
 else
    SDMF_flag = FALSE;
```

```
}


void CallerIDWindow::Process_SDMF()
{
 // Initialize the data parameter strings.
 Name = "";
 Number = "";
 Date_Time = "";
 int i;
 char tempstr[50];
 char *endptr;

 // Parse out the time, date, and number from the data
 // stream.
 for(i=4;i<20;i+=2)
    {
        // Parse out the caller ID date and time.
        strncpy(tempstr,&RawData[i],2);
        Date_Time += (UINT)strtoul(tempstr,&endptr,16);
    }


 // Parse out the caller ID telephone number.
 for(i=20;i<40;i+=2)
    {
        strncpy(tempstr,&RawData[i],2);
        Number += (UINT)strtoul(tempstr,&endptr,16);
     }

 Name = "UNAVAILABLE";
 }


void CallerIDWindow::Process_MDMF()
{
 // Initialize the data parameter strings.
 Name = "";
 Number = "";
 Date_Time = "";
 int messagelength = 0;
 int parametertype = 0;
 int parameterlength = 0;
 int tempint = 2;
 char tempstr[50];
 char *endptr;
```

```
// Get the message length parameter from the raw data stream.
tempstr[2] = 0x00;
strncpy(tempstr,&RawData[tempint],2);
messagelength = (int)strtoul(tempstr,&endptr,16);
tempint = 4;
// Parse out the time, date, number, and name parameters from the data
// stream.
while((messagelength > 0)&& !LineError_flag)
        {
         strncpy(tempstr,&RawData[tempint],2);
         parametertype = (int)strtoul(tempstr,&endptr,16);

         strncpy(tempstr,&RawData[tempint+2],2);
         parameterlength = (int)strtoul(tempstr,&endptr,16);
         messagelength -=2;
         tempint += 4;


         // Parse out the data stream into various caller ID parameter
         // type string.
         switch(parametertype)
                  {
                        // Parse out the caller ID date and time parameter.
                        case 1:
                           while(parameterlength > 0)
                                {
                                   strncpy(tempstr,&RawData[tempint],2);
                                   Date_Time += (UINT)strtoul(tempstr,&endptr,16);
                                   messagelength -= 1;
                                   tempint += 2;
                                   parameterlength -= 1;
                                }
                           break;

                        // Parse out the caller ID telephone number parameter.
                        case 2 :
                             while(parameterlength > 0)
                                  {
                                 strncpy(tempstr,&RawData[tempint],2);
                                 Number += (UINT)strtoul(tempstr,&endptr,16);
                                   messagelength -= 1;
                                   tempint += 2;
                                   parameterlength -= 1;
                                  }
                           break;
```

```
                        // Parse out the caller ID name parameter.
                        case 7 :
                             while(parameterlength > 0)
                                  {
                                   strncpy(tempstr,&RawData[tempint],2);
                                   Name += (UINT)strtoul(tempstr,&endptr,16);
                                   messagelength -= 1;
                                   tempint += 2;
                                   parameterlength -= 1;
                                  }
                             break;
                   default :
                             // If an invalid parameter is received set the line
                             // error global flag and display a "Line Error" message.
                             LineError_flag = TRUE;
                   }
          }
}


void CallerIDWindow::Format_Data()
{
 // Initialize the strings to be displayed.
 // Initialize data acquisition flags.
 CString Date_Text = "Date: ";
 CString Time_Text = "Time: ";
 CString Number_Text = "Number: ";
 CString Name_Text = "Name: ";
 BOOL PM_flag = FALSE;
 char tempstr[50];
 int tempint;

 // Format the date string.
 // If the first number of the date is equal to zero eliminate it from
 // the string otherwise include it in the string.
 if(Date_Time.Mid(0,1) == 0x30)
    Date_Text = Date_Text + Date_Time.Mid(1,1) + '/' + Date_Time.Mid(2,2);
 else
    Date_Text = Date_Text + Date_Time.Mid(0,2) + '/' + Date_Time.Mid(2,2);

 Date = Date_Text;
 // Format the time string.
 // If the first number of the time is equal to zero eliminate it from
 // the string, otherwise include it in the string.
 tempstr[2] = 0x00;
 strcpy(tempstr,Date_Time.Mid(4,2));
 tempint = atoi(tempstr);
```

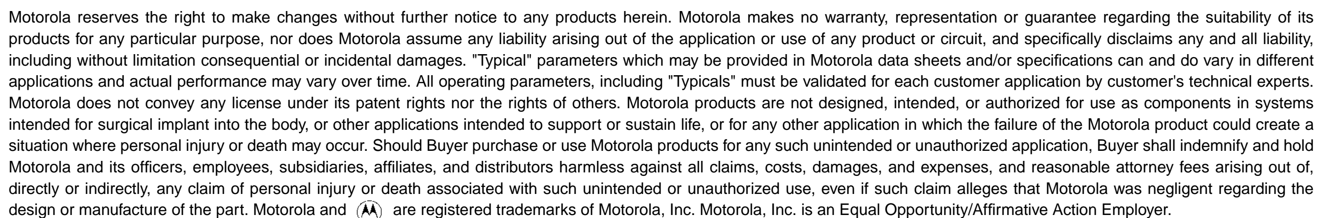
```
if(tempint == 0)
  {
   Date_Time.SetAt(4,'1');
   Date_Time.SetAt(5,'2');
   PM_flag = FALSE;
      }
else if(tempint == 12)
   PM_flag = TRUE;

if((tempint > 12))
  {
   PM_flag = TRUE;
   tempint -= 12;
   _itoa(tempint,tempstr,16);

   if(tempint >= 9)
        {
         Date_Time.SetAt(4,tempstr[0]);
         Date_Time.SetAt(5,tempstr[1]);
        }
      else
        {
         Date_Time.SetAt(4,'0');
         Date_Time.SetAt(5,tempstr[0]);
        }
  }

if(Date_Time.Mid(4,1) == 0x30)
  {
   Time_Text = Time_Text + Date_Time.Mid(5,1) + ':' + Date_Time.Mid(6,2);

   if(PM_flag)
         Time_Text += " PM";
   else
         Time_Text += " AM";
  }


else
  {
   Time_Text = Time_Text + Date_Time.Mid(4,2) + ':' + Date_Time.Mid(6,2);

      if(PM_flag)
         Time_Text += " PM";
   else
         Time_Text += " AM";
  }

Time = Time_Text;
```

```
 // Format the telephone number string.
 // If the first number of the telephone number is zero, eliminate the
 // area code from the number.
 if(Number.Mid(0,1) == 0x30)
       Number_Text = Number_Text + Number.Mid(3,3) + '-' + Number.Mid(6,4);
 else
    Number_Text = Number_Text + '(' + Number.Mid(0,3) + ") " + Number.Mid(3,3) +
                  '-' + Number.Mid(6,4);
 Number = Number_Text;

 // Format the name string if one exists.
 Name_Text = Name_Text + Name;
 Name = Name_Text;

 // Set the display flag.
 Display_flag = TRUE;
}


void CallerIDWindow::Handle_OK_Button()
{
 // Re-initialize all main window variables
 StartByte_flag = FALSE;
 Display_flag = FALSE;
 LineError_flag = FALSE;
 RawData[0] = '\0';

 // Clear the main window and hide it.
 Invalidate(TRUE);
 ShowWindow(SW_HIDE);
}

void CallerIDWindow::Handle_Deactivate_Button()

{
 DestroyWindow();
}
#define DllExport __declspec(dllexport)

// Keyboard hook installation function prototype
DllExport void WINAPI InstallHook(void);

// Keyboard hook function prototype.
LRESULT CALLBACK KeyboardHook (int nCode, WORD wParam, DWORD lParam );
```

Appendix G — CALLDLL.DLL Source Code File

```
#include <windows.h>
#include "calldll.h"


#pragma data_seg( "CommMem" )
    HHOOK hHook = NULL;
#pragma data_seg()

HANDLE hDLLInst = 0;

// This function is the main function required by Windows 95 for
// DLLs written in C.

BOOL WINAPI DllMain (HANDLE hModule, DWORD dwFunction, LPVOID lpNot)
{
    hDLLInst = hModule;

    switch (dwFunction)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_PROCESS_DETACH:
        default:
            break;
    }
    return TRUE;
}

// This function connects the keyboard hook function to the Windows 95
// operating system.

DllExport void WINAPI InstallHook (void)
{
    if (hHook == NULL){
       hHook = (HHOOK)SetWindowsHookEx(WH_KEYBOARD,(HOOKPROC)KeyboardHook, hDLLInst,
                                     0);
    }
    else{
        UnhookWindowsHookEx(hHook);
        hHook = NULL;
        }
}
```

```
// This function is connected to the Windows 95 environment and monitors
// user key strokes for a <CONTROL L> key combination. On detecting the
// hotkey, the hook interrupts the application that has the focus in the
// Windows 95 environment, and restores CALLERID.EXE's main window thus
// giving it the application in Windows 95.

LRESULT CALLBACK KeyboardHook (int nCode, WORD wParam, DWORD lParam )
{
        LRESULT lResult = 0;
        HWND hWndMain = 0;

      // If the hook function detects a <CONTROL L> key combination, interrupt
      // the current application in the Windows 95 environment, give CALLERID.EXE the
      // focus and restore its main window.
        if(nCode == HC_ACTION){
                if ((wParam == 'L') && (GetKeyState(VK_CONTROL) < 0) && (lParam &
0x80000000)){


                    hWndMain = FindWindow(NULL,"PC Caller ID");
                         ShowWindow(hWndMain,SW_RESTORE);


                         lResult = 1;
                         return(lResult);
                }
        }

   // Move to the next hook function in the hook function chain.
   return (int)CallNextHookEx(hHook, nCode, wParam, lParam);
}
```


Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and (M) are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

How to reach us:

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

Mfax™: RMFAX0@email.sps.mot.com – TOUCHTONE 602-244-6609, US & Canada ONLY 1-800-774-1848

INTERNET: http://www.mot.com/SPS/

JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, 6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 81-3-3521-8315

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

HOME PAGE: http://motorola.com/sps

Mfax is a trademark of Motorola, Inc.

© Motorola, Inc., 1998

MOTOROLA