App Note 710: Dial-Up Networking with the DS80C400 Microcontroller

The DS80C400 networked microcontroller provides a ready solution for monitoring and controlling sensors/actuators over networks. Data can be transmitted via PPP to server for analysis and storage. This example uses a TINI® with a DS1923 1-Wire® temperature/humidity sensor packaged as an iButton® to collect data. Discussion on TINI PPP implementation is included.

As technology advances, large network availability greatly simplifies the process of microcontrollers monitoring and controlling sensors/actuators. Information can now be sent over the network to a central location for analysis and corrective action. For such an application, the DS80C400 networked microcontroller provides a ready solution. Besides being loaded with extensive peripherals, the DS80C400 silicon software implements a TCP/IP stack[1]. The Tiny InterNet Interfaces (TINI®) platform[2], which includes a Java™Virtual Machine (JVM), provides extensive support of IP networking. Although the DS80C400 includes an Ethernet interface, the TINI Runtime Environment (TRE) also supports dial-up networking using the point-to-point protocol (PPP). A compelling aspect of using PPP is that both endpoints of the connection can communicate over modems to leverage public communication networks and IP software infrastructure. This allows the deployment of remote embedded networking applications in outposts where an Ethernet network is not available, but the ubiquitous phone switch network is (Figure 1).

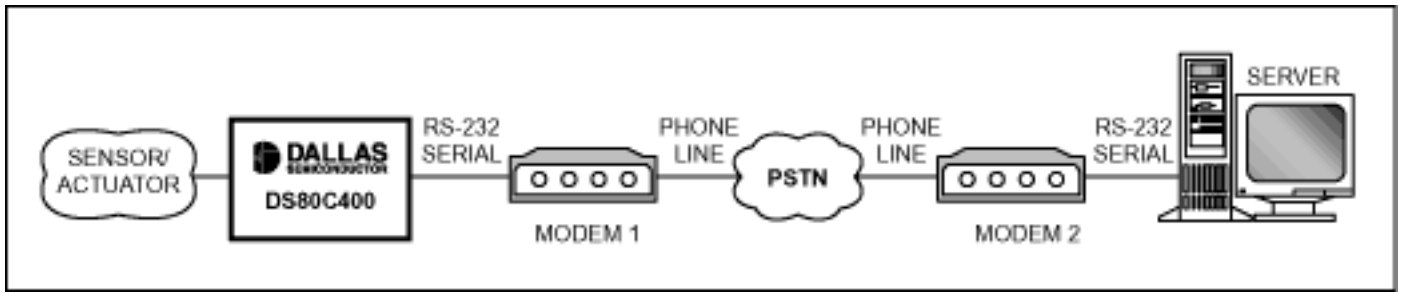

Figure 1. A remote DS80C400 running the TINI Runtime Environment dials up a server to forward data.

PPP Overview

PPP is a general-purpose protocol that supports data transfer over many physical media, including (but not limited to) serial, parallel, Ethernet, and cellular phones, such as general packet-radio service (GPRS) devices. PPP is widely used in dial-up networking application

because it requires little configuration and is easy to set up. The only requirement for the physical media is full-duplex capability. The communication can be either synchronous or asynchronous.

PPP is composed of three main components:

1. A method for encapsulating multiprotocol datagram over the same link. PPP encapsulation is based on the high-level data-link control (HDLC) format. Some encapsulation fields can be compressed if available bandwidth is limited.
2. A link control protocol (LCP) that establishes a connection, configures link options, detects errors, and terminates the link.
3. A family of network control protocols (NCP) that establish and configure corresponding network- layer protocols.

PPP Operation

The Internet standard RFC 1661 describes PPP operation as a state machine going through different stages as the point-to-point link is configured, maintained, and terminated. Figure 2 describes a simplified state diagram where PPP is divided into five distinct phases: dead, establish, authenticate, network, and terminate. PPP implements all phases except authenticate.

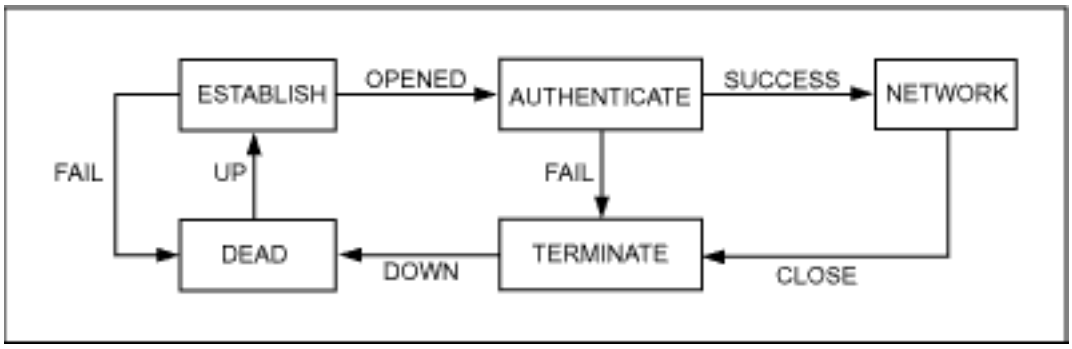

Figure 2. This simplified phase diagram illustrates PPP implementation described in RFC 1661.

Link Dead Phase

Initial and ending phase of a link operation. The physical layer is not ready for packet transfer yet. When the physical layer is ready, an UP event is generated, and PPP proceeds to the link establishment phase.

Link Establishment Phase

The physical layer is up, and the link is negotiating transport options with its peer by exchanging LCP configuration packets. Only options independent of the network-layer protocol are configured at this stage. Once a configure-ACK packet has been sent and received, the PPP generates an OPENED event and proceeds to the next stage.

Authentication Phase

An optional phase that authenticates to the peer. On some links, such as dial-up networking, it is

desirable for the link to authenticate before network-layer protocol packets can be exchanged. For such an implementation, a request for authentication must be sent during the link establishment phase.

Network-Layer Protocol Phase
After the link has been established and authentication has succeeded, the network-layer protocol is configured by exchanging NCP packets specific to the network layer supported. Each network-layer protocol has a unique NCP and must be negotiated separately.

Link Termination Phase
A CLOSE event is generated when the PPP link is terminated because of carrier loss, authentication failure, link-quality failure, or the administrative closing of the link. LCP terminate packets are exchanged between peers. The network-layer protocol is informed of the closing link and takes appropriate action. The physical layer is disabled after receiving a terminate-ACK, or timeout. A DOWN event is then generated, and the PPP returns to the link dead phase.

TINI PPP
TINI uses RFC 1661 as a framework for PPP implementation. PPP serves strictly as a transport mechanism for IP datagrams over a serial link. In the native network stack, PPP exists below the IP module and above the serial port drivers. To alleviate programming complexity, the PPP stages are simplified further (Figure 3).

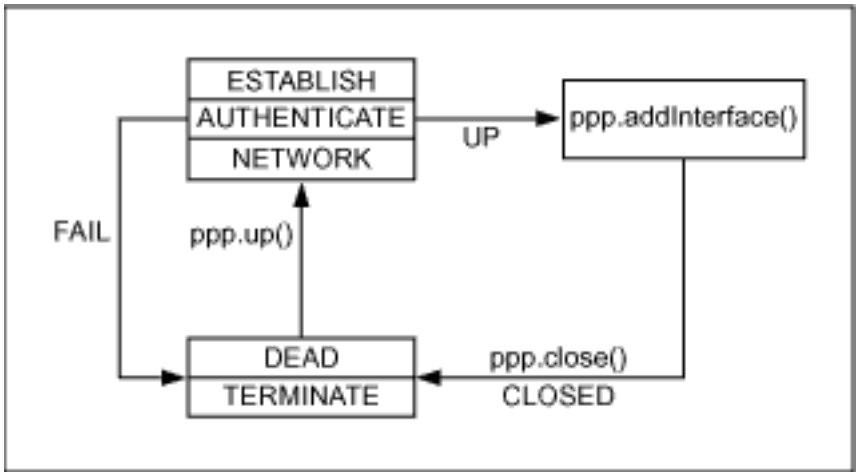

Figure 3. This TINI PPP phase diagram illustrates how PPP is implemented on TINI.

PPP is explained to an application developer through Java classes in the com.dalsemi.tininet.ppp package. The PPP states are event driven. ppp.up() establishes the link, authenticates, and sets up the network protocol. Password-authentication protocol (PAP) and challenge-handshake-authentication protocol (CHAP) are supported[3]. Once the link is configured, an UP event is generated and adds the PPP interface to the network stack so network traffic can be directed to that interface. ppp.close() issues a CLOSE event, brings the link down, and returns to the dead/terminate state.

Example 1 shows fragments of a PPPClient implemented on the DS80C400. (Get the latest TINI firmware from ftp://ftp.dalsemi.com/pub/tini/ for updated code.) After a PPP object is created, the PPPClient is installed as the PPP object's PPPEventListener. PPP parameters are set, and the link is initiated with a series of atCommand. Once the link is established, ppp.up() is called to notify the network stack that PPP is now available for network traffic. The link is terminated with ppp.close().

Example 1. PPPClient implementation

```
public class PPPClient extends Thread
    implements PPPEventListener,
    CommPortOwnershipListener{
    ...
    public void run(){
        ppp = new PPP();
        openSerialPort(portNumber);
        // Add this object as a PPP event listener
        ppp.addEventListener(this);
        // Set the local and remote IP address
        ppp.setLocalAddress(localAddress);
        ppp.setRemoteAddress(remoteAddress);
        // Set client peer type options
        // Set the ACCM to escape all octets
        ppp.setRemoteAccm(0x00000000);
        ppp.setLocalAccm(0x00000000);
        ppp.setAuthenticate(false, true);
        // Set username and password
        ppp.setUsername(username);
        ppp.setPassword(password);
        // Initialize modem
        for (int i = 0; i < dialSequence.length; ++i)
        atCommand(dialSequence[i]);
        // Set connected flag
        connected = true;
        // Issue up command to PPP FSM
        ppp.up(serialPort);
        // PPP connection is now established, we can now
        // communicate with remote host sendData();
        ppp.close();
        closeSerialPort();

    }
    ...
```

```
}
```

Example 2 demonstrates how a PPPEvent can be handled. Once the link is ready, and the UP event has been received, PPP is added as one of the network interfaces so IP packets can be forwarded to this interface. Whenever a CLOSE event is received, the network stack removes the PPP interface, terminating any network activity through PPP.

Example 2. PPPEvent method

```
/**
* PPP event listener interface
*/
public void pppEvent(PPPEvent ev){
    switch (ev.getEventType()){
    case PPPEvent.UP:
        // PPP connection is up
        ppp.addInterface(interfaceName);
        interfaceActive = true;
        break;
        case PPPEvent.CLOSED:
        // PPP connection is closed


            if (interfaceActive){
            interfaceActive = false;
            ppp.removeInterface(interfaceName);
            }
            connected = false;
            break;
        default:
        break;


    }


}
```

Remote Humidity Data-Logger Example
In this article we write a powerful, networked application that takes full advantage of networking capabilities provided by a very economical, small form-factor computer. The example uses a TINI reference design called the TINIm400. This module provides a low-power, I/O-rich, low-part-

count embedded controller and data collection module. When combined with the TRE, robust networked data-collection systems require minimal software effort. The TINIm400 module includes flash ROM, RAM, a real-time clock, 1-Wire® network, parallel I/O, and asynchronous serial ports. This article presents a complete example that captures and logs data, connects over the PSTN (public switched-telephone network) using PPP to manage dial-up connections, and makes the data available to a remote server. Dial-up networking support makes the data logger truly remote.

System Overview

Figure 4 shows the setup for demonstrating dial-up networking capabilities using analog modems. If phone lines or their equivalent are not available, the hardwired serial-to-serial connection can also be used in a test setup.

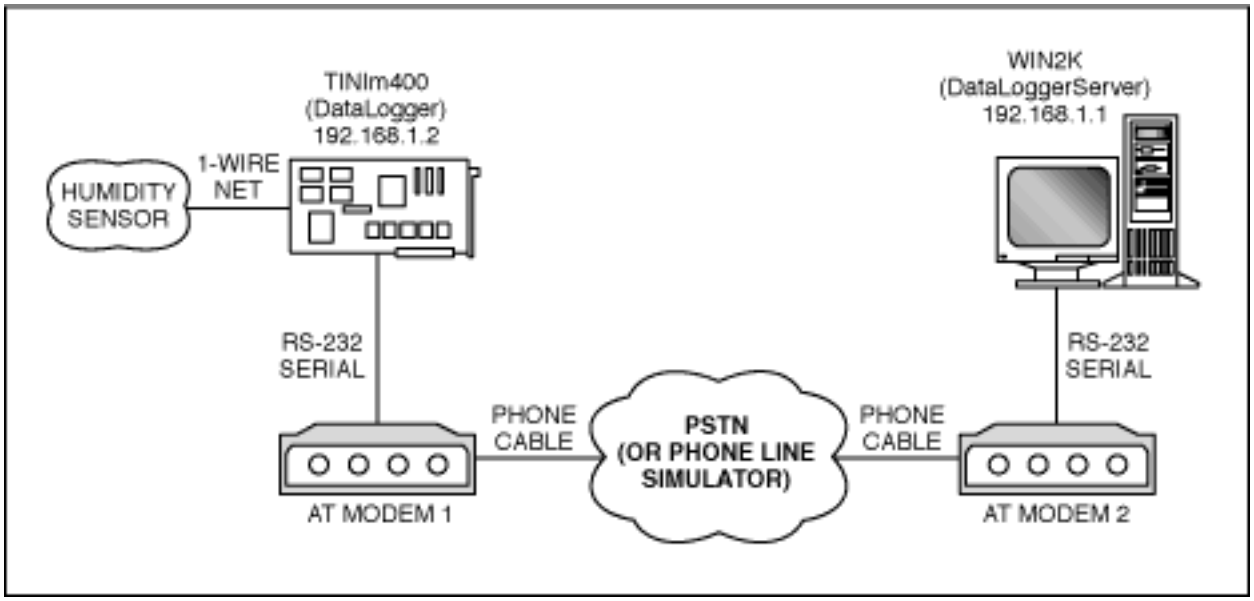

Figure 4. A remote data-logging system uses generic modems to transfer data.

The test configuration in Figure 4 includes the following equipment:

- A PPP module-running the DataLogger server
- A Windows® 2000 machine-running the DataLoggerServer software
- Two analog modems-one attached to the Win2K PC and the other attached to the serial port of the PPP module
- A humidity sensing circuit-collecting humidity data for logging

The DS80C400 on the Stamp+ module has the TRE installed. The TRE platform supports ABM, C, and Java programming. The embedded firmware implements TCP/IP stack and provides framework for the PPP protocol.

The PPP connection is made using two analog modems on either side of a phone line simulator. If two different phone lines are available, the public phone network can be used instead. To test

the PPP interface, a dial-up network connection should be created. Once the connection is initiated, the following sequence of events occurs:

1. TINI modem dials the server's modem.
2. The server's modem answers the incoming call.
3. PPP option negotiation begins.
4. Authentication information is transmitted from TINI to the remote server.
5. The server assigns an IP address to the TINI and notifies the TINI of the server's address.

Software and Hardware Overview

Complete source code can be downloaded from ftp://ftp.dalsemi.com/pub/tini/reference_designs/TINIm400-030p/DataLogger.zip. Figure 5 shows humidity data is collected using a sensor circuit. The example sensor uses a DS1923 1-Wire temperature/humidity sensor packaged as an iButton® , although any 1-Wire device can be used with the DS80C400.

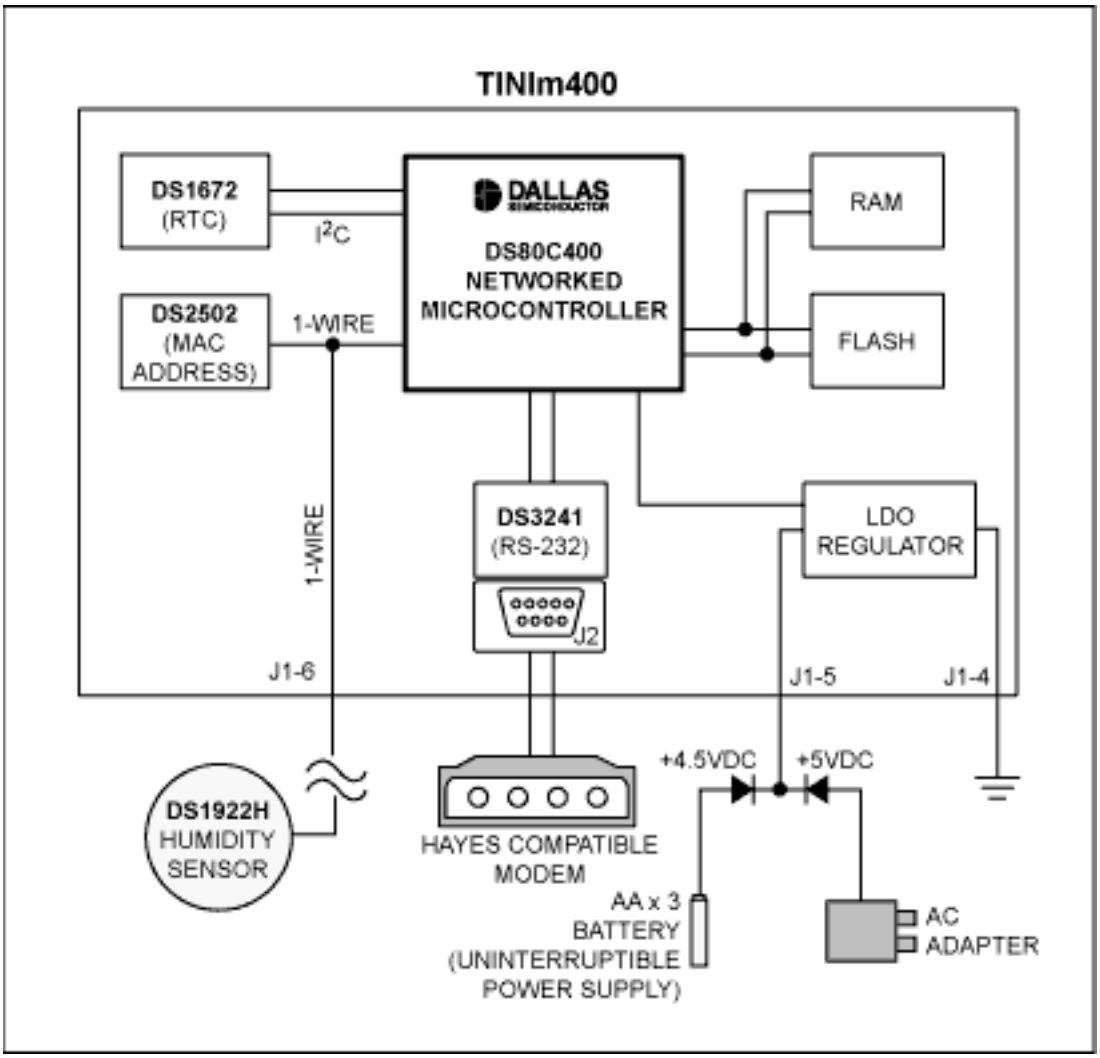

Figure 5. A TINIm400 data logger collects humidity data using a sensor.

TINI Client Software

The TINI DataLogger example demonstrates three concepts: 1-Wire networking, serial communications, and TCP/IP networking. Brief functionality descriptions of the most important classes are summarized below.

The DataLogger Class

- Retrieves parameters from the configuration file: /etc/dataLogger.properties
- Creates an instance of HumidityLogger to capture sample
- Creates an instance of PPPDaemon to manage PPP connection
- Initiates outbound connections to the remote server over the network interface such as Ethernet or PPP

The HumiditySensor Class

- Handles communication with, and retrieves data from the DS1923

The HumidityLogger Class

- Initializes and manages humidity sensors
- Writes log data to the output stream to the server

The PPPDaemon Class

- Dial-up client
- Establishes TCP/IP connections using a PPP interface
- Manages the physical data link
- Receives PPP event notification
- Notifies DataLogger of errors that occur in the physical data link

The PPPSerialLink Class

- Implements the PPPDataLink interface
- Allows PPPDaemon to manage the data link
- Configures the serial port for the data link

The PPPModemLink Class

- Subclass of PPPSerialLink
- Manages modem communications
- Monitors SerialPortEvent.CD (carrier detect) to detect if the modem hangs up

The ModemCommand Class

- Handles serial communication with the modem
- Throws DataLinkException in case of a timeout while waiting for the desired response

Remote Data-Logging Server

The DataLoggerServer is a simple GUI server application developed to accept connection from

TINI and download its current log.

The DataLoggerServer Class

- Displays log data
- Blocks on accept to wait for socket connection at PORT
- Builds logs and charts humidity and temperature changes over time

Running the Example

Application files traditionally have been transferred to the TINI file system using the FTP protocol over an Ethernet interface. In order to be independent of the Ethernet interface, the ymodem file transfer protocol has been added to Slush and JavaKit. Ymodem allows files to be transferred to the TINI file system over the JavaKit serial link. Besides the application file, the DataLogger.tini, a /etc/.startup file containing the following text should be transferred to the TINI file system.

```
#
# Starting DataLogger application from Startup file
#
setenv FTPServer disable
setenv TelnetServer disable
setenv SerialServer disable
#
initializeNetwork
#
java /DataLogger.tini
```

This startup file disables the serial server and allows the data-logger application access to the serial port. Once the application and startup files have been transferred, resetting the TINI allows the new startup file to be processed and the data-logger application to be started.

The first thing TINI sends to DataLoggerServer is an integer value that tells the server the number of log entries to expect. After the server reads this value, it loops through all entries, reading each individual sample. The server displays each entry and logs this information to a file. The DataLoggerServer is written in Java and requires a Java Runtime Environment for execution, the same execution environment used to run Slush. See www.java.sun.com for installation instructions.

After running the DataLogger for several minutes to allow it to acquire a few samples, DataLoggerServer is run. In this example each sample was time-stamped one minute apart. If DataLogger runs more than one hour, it fills its sample vector, resulting in 60 (MAX_SAMPLE) data samples. If it runs for days, weeks, or even months, it still gets MAX_SAMPLE samples, but they always represent readings taken within the last hour.

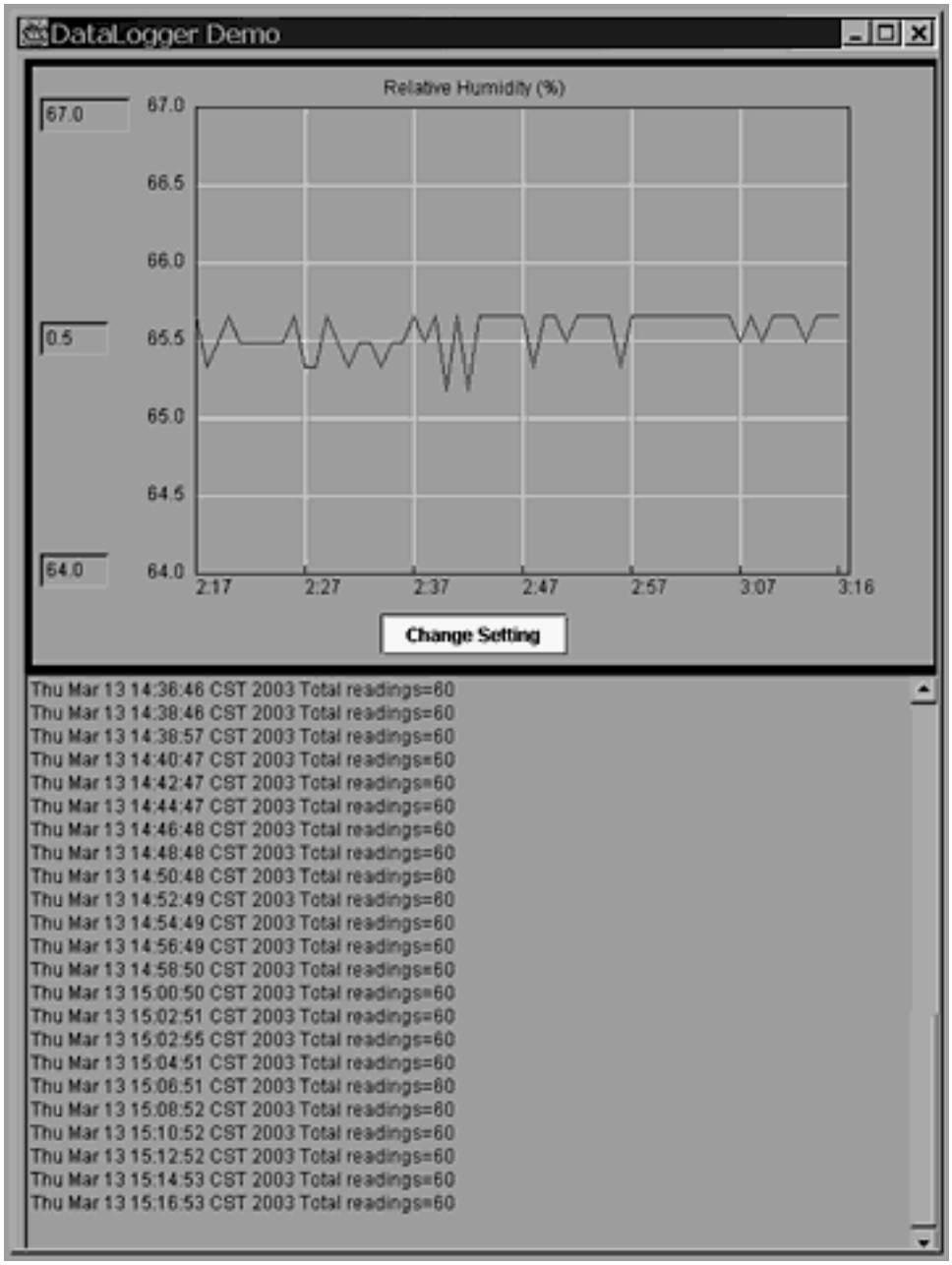

Figure 6. The PC screen sample displays the DataLoggerServer operation.

Conclusion

Implementing a dial-up network connection on the DS80C400 is straightforward. The TINI Runtime Environment provides user-friendly APIs that conceal details so developers can concentrate on their designs, using PPP as a utility. Even without a traditional phone network, the same applications can still run by replacing the modem with a GPRS wireless phone.

The TINIm400 can be configured to initiate or receive dial-up connections. For data logging, a central server can dial into the TINIm400 and retrieve data at periodic intervals. In the event of a local fault, the TINIm400 module can initiate a PPP dial-up connection to the central server to notify the system of the error. By using the TINIm400 to detect local faults, the central server can be dedicated to analyzing the retrieved data.

As with any embedded module, the hardware and algorithms used depend on the specific application. The rich I/O capability of the DS80C400 and flexibility of the TINI Runtime Environment make adding remote sensors/actuators to networks quick.

Footnotes

1. The silicon software supports IPv4/6 over Ethernet.
2. Application Note 708: Tiny InterNet Interfaces (TINI)
3. TRE Firmware Version 1.1 and later.
4. This example is derived from Chapter 7 of The TINI Specification and Developer's Guide, available at www.maxim-ic.com/TINIguide.

More Information

DS1672:	QuickView	Full (PDF) Data Sheet	Free Samples
DS1923:	QuickView	Full (PDF) Data Sheet	Free Samples
DS2502:	QuickView	Full (PDF) Data Sheet	Free Samples
DS80C390:	QuickView	Full (PDF) Data Sheet	Free Samples
DS80C400:	QuickView	Full (PDF) Data Sheet	Free Samples
DSTINIM400:	QuickView	Full (PDF) Data Sheet	