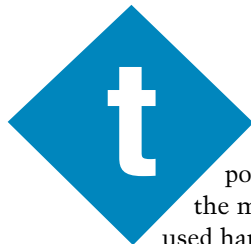


# FEATURE ARTICLE

Robert Bowen

## Developing with Open-Source TCP/IP



The RS-232 serial port is still one of the most commonly used hardware interfaces. With a low cost and reduced pin count, it isn't surprising to find this interface built into most microcontrollers sold today. My favorite hand-held Fluke 189 digital multimeter proudly boasts an RS-232 connection.

In the past, I have faithfully used the serial or parallel port on a desktop computer to communicate with embedded devices. This decision was not made by accident. Listing 1 reveals how easy it is to move data across a parallel port of a PC using a high-level language such as Visual Basic.

The advantages are clear. There is less software overhead. Direct access to the hardware interface is offered. In addition, no communication protocol is required to send and receive command data.

Today, most embedded PCs and SBCs sport an Ethernet interface. Transmitting and receiving data through this hardware interface requires more than simply addressing a direct I/O port. It involves the use of a communication protocol.

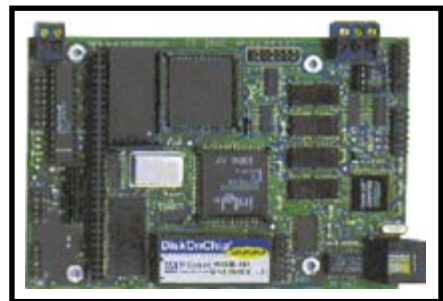
In this article, I'll focus on the essential components required to transmit and receive data via UDP datagrams across an Ethernet interface. First, I will introduce a well-known and free TCP/IP protocol stack from Waterloo and port it to the MicroC/OS-II real-time kernel. Next, with just a little extra effort, I'll transfer the kernel application to the TS-2800 embedded PC from Technologic Systems (see Photo 1). Finally, I'll create a network application (GUI) on a desktop PC to communicate with the TS-2800 via Ethernet for control and monitoring purposes.

### OPEN-SOURCE TCP/IP

The key to writing any network application is utilizing the network programming library, more commonly referred to as the Application Program Interface (API). In the Windows environment, most people refer to this library as the WinSock API.

WinSock was designed to create a standard programming interface for TCP/IP on all versions of Windows OS. You'll examine the WinSock API when developing your network GUI. The first order of business is to find a socket API suitable for DOS. Luckily, I didn't have far to search. The folks over at Technologic Systems were kind enough to ship a copy of the WATTCP TCP/IP protocol stack (including source code) with the TS-2800 embedded PC. You may download a complete copy of the WATTCP TCP/IP stack from the Internet.

Erick Engelke originally wrote and released the WATTCP TCP/IP stack



**Photo 1**—The TS-2800 sports the usual embedded goodies including an ADC, LCD interface, flash memory disk from M-Systems, and digital I/Os. Hiding behind the RJ-45 connector is a 10BaseT Ethernet port powered by the CS8900A Ethernet controller from Cirrus Logic.

**Listing 1**—Under Windows 95/98, sending and receiving data through the parallel port on a PC is a trivial task. Software overhead is low. Direct access to the port is accessible using a high-level language like Visual Basic.

```
Public Declare Function Inp Lib "inout32.dll" _
Alias "Inp32" (ByVal PortAddress As Integer) As Integer
Public Declare Sub Out Lib "inout32.dll" _
Alias "Out32" (ByVal PortAddress As Integer, ByVal Value As
Integer)

DataPort = &H378           //Base address
StatusPort = &H379        //Base address plus one
ControlPort = &H37A       //Base address plus two
CmdData = Inp(DataPort)   //Read data from 8-bit data port
Out (DataPort, 12)        //Write data (12) to the 8-bit
data port
```

**Listing 2**—The WATTCP TCP/IP socket is not reentrant. The socket is initialized and all socket calls are handled in Task 1.

```
sock_init();
udp_open(&TASK1_UDP_DATA, localport, remote, remoteport, NULL);
sock_mode(&TASK1_UDP_DATA, UDP_MODE_NOCHK);
sock_puts(&TASK1_UDP_DATA, "SocOpening Socket().Connection
Successful!");
data = data;
while (1) {
    rxmsg = (char *)OSMboxPend(Txmbox, 0, &err);
    sock_puts(&TASK1_UDP_DATA, rxmsg);
    OSMboxPost(AckMbox, (void *)1);
    n = sock_dataready(&TASK1_UDP_DATA);
    if (len > sizeof(buffer))
        len = sizeof(buffer);
    sock_read(&TASK1_UDP_DATA, buffer, len);
    buffer[len]=0;
    cmd = *buffer;
```

in 1989. It was designed specifically to work within limited memory and to support multi-tasking applications, making it well suited for your select-ed RTOS.

The WATTCP TCP/IP protocol suite supports both TCP and User Datagram Protocol (UDP). TCP is considered a reliable protocol because packets are retransmitted if they are lost or corrupted. UDP is considered an unreliable protocol because there is no guarantee that a datagram will arrive. Moreover, if a datagram does arrive, there is no guarantee that the response will be received. However, software overhead is higher when developing with TCP. For this application, you'll use UDP.

The TCP.H include file contains the definitions and function prototypes of

all the socket calls. Further examination of the header file reveals the programming syntax for each function call. The WATTCP socket calls used in this real-time application are sock\_init(), udp\_open(), sock\_mode(), sock\_read(), and sock\_puts().

Porting the library to the MicroC/OS-II kernel is straightforward and completely painless. First, you will include the TCP.H header file in your main INCLUDE.H file. Next, compile. Then, link the WATTCP.LIB file to the MicroC/OS-II application.

Figure 1 displays the component architecture of the system. DOS loads both the kernel application and packet driver. The library communicates with the packet driver, which in turn

talks directly to the TS-2800 Ethernet hardware.

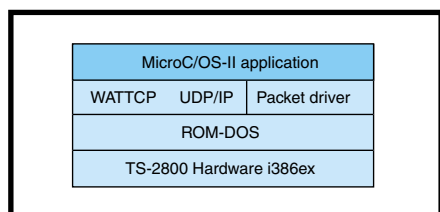
## THE KERNEL APPLICATION

A word of caution, the WATTCP TCP/IP library is not reentrant (software that can be executed multiple times simultaneously). Nevertheless, you can easily work around this limitation by creating a mailbox to handle all incoming and outgoing Ethernet traffic. A mailbox is a service provided by the kernel. It allows a task to send a pointer to another task. The pointer can point to any application specific data, but both the sender and receiver need to agree about the data being pointed to.

Your application will have two tasks. Task 1 will be the highest priority task and be responsible for handling all of the Ethernet UDP datagrams between the TS-2800 and host computer. Task 2 will take on the challenge of reading the onboard ADC, monitoring digital inputs one through four, and displaying the results on the LCD. Table 1 lists all of the tasks and their priorities.

Let's examine Listing 2, which shows the details of Task 1. Task 1 starts by initializing the WATTCP socket library by calling sock\_init. The udp\_open function configures the socket on which to send and receive UDP datagrams. The IP address and port number of the host computer are defined in this function call.

Next, use the sock\_mode function to enable or disable checksums on each data packet sent and received. Checksums are enabled by default. However, you'll disable checksums in this application to speed up both local and remote processing of data-



**Figure 1**—DOS loads both the kernel application and packet driver. The WATTCP library communicates with the packet driver, which in turn talks directly to the TS-2800 Ethernet hardware.

grams. The kernel application is now prepared to start passing Ethernet data. The `sock_puts` and `sock_read` functions are used to transmit and receive UDP datagrams, respectively.

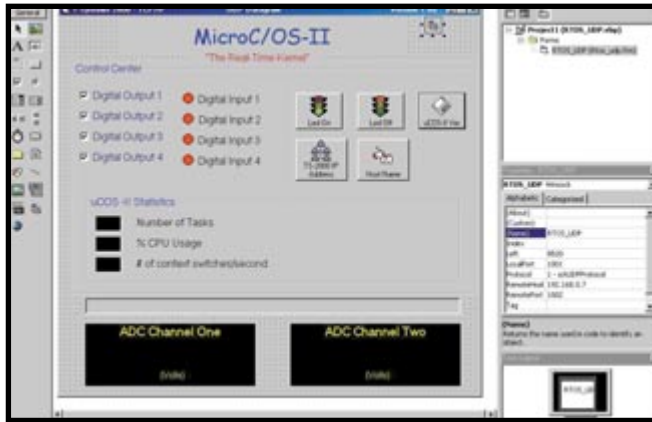
Task 2 (see Listing 3) is responsible for acquiring the A/D readings. Next, digital inputs one through four are sampled and the results are placed into a mailbox and sent to Task 1. Finally, the LCD is updated with current A/D readings from Channels one and two.

Before launching the MicroC/OS-II application, you need to address a couple of maintenance items on the TS-2800. First, the WATTCP TCP/IP socket does not talk directly to the network hardware. Instead, WATTCP TCP/IP communicates with a packet driver. Again, I didn't have far to search. A packet driver from Crynwr Software was included on my utility disk from Technologic Systems. Listing 4 shows how to include this packet driver into the `autoexec.bat` file for automatically loading at boot-up.

Finally, you need to educate WATTCP about your network with details such as the IP address, host name, and net mask. The `WATTCP.CFG` file contains the settings to configure the TCP/IP protocol stack for your network. A sample of the `WATTCP.CFG` file is shown in Listing 5. Now, the kernel application is complete and ready for uploading to the TS-2800 for execution. It's time to build the network GUI.

## SOCKET PROGRAMMING

While developing the MicroC/OS-II kernel application, I used the WATTCP socket library. An understanding of the WATTCP macros and programming syntax were necessary to build the kernel application. I'll take a little different approach developing the network GUI. Rather than using direct WinSock API calls, I'll use the WinSock OCX component. By



**Photo 2**—Visual Basic provides an easy IDE for the network programmer. By simply adding the Visual Basic WinSock component to the main form, your application is ready to talk TCP/IP. Simply type in the local port, protocol, remote port, and IP address and Visual Basic handles all of the low-level WinSock calls.

simply adding this OCX object to the main form of the application, all of the low-level function calls are transparent to the programmer.

Photo 2 is a snapshot of the Visual Basic IDE. After defining a few parameters to the WinSock, your application is ready to pass UDP datagrams.

Listing 6 shows the source code for the network GUI. As you'll notice, sending and receiving data through an Ethernet interface is not much different than with the familiar RS-232 serial port.

## CONFIGURING THE PC

A simple `WATTCP.CFG` file located on the TS-2800 flash memory disk provided all the necessary network information. Configuring the desktop PC is just as effortless (thanks, Microsoft) (see Photo 3). Under control panel/network, enter the

TCP/IP properties for your network. The network GUI now has a ticket to the Ethernet gateway and is free to roam.

The network application GUI shown in Photo 4 presents a clean interface while demonstrating some simple control and monitoring tech-

**Listing 3**—First, the channel of the ADC is sampled and the readings are posted into a mailbox. The results are then sent to Task 1 for transmitting to the host computer via UDP datagrams.

```
while (1) {
//Channel 1 ADC reading
  outp(0x078, 0x11);
  OSTimeDlyHMSM(0, 0, 0, 5);
  msb = (inpw(0x79));
  lsb = (inpw(0x78));
  ADCReading = (((msb | lsb) * FullScale)/Resolution);
  sprintf(s, "CH1 %5.3f",ADCReading );
//Send CH1 reading to task 1 for Tx via UDP to host PC
  OSMboxPost(TxMbox,(void *)&s);
  OSMboxPend(AckMbox,0,&err);
//Display CH1 reading on LCD
  OSSemPend(LCDSem, 0, &err);
  DispStr(1, 0, s);
  OSSemPost(LCDSem);
```

**Listing 4**—The WATTCP TCP/IP stack does not talk directly to the network card. It communicates through a packet driver interface. `AUTOEXEC.BAT`, as shown here, can be written to load the packet driver after DOS starts. The last line is the packet driver and software interrupt address.

```
echo on
path =
a:\util;a:\ethernet;a:\dos;c:\util;c:\dos;c:\ethernet;c:\rtos;
prompt $p$g
set DIRCMD=/ogn
epktisa 0x60
```

niques. Commands are sent from the PC to the TS-2800 to control the onboard LED and the digital outputs. The two panel meters update automatically with the A/D readings from the embedded device as well as the status of the digital inputs.

A green light indicates an active-high output and a red indicates a logic low. Notice the two buttons labeled "TS-2800 IP Address" and "Host Name." Pressing these buttons will return the IP address and host name of the TS-2800. The finished product is shown in Photo 5.

## TCP/IP EMBEDDED STYLE

This article focused on UDP/IP for embedded systems as well as provided an introduction to WinSock programming. If you are developing network applications for a desktop PC, your perception of TCP/IP may be quite different from an embedded engineer's understanding. However, the programming concept is the same. Both are easily implemented through standard socket API calls.

Whether you are designing an Internet appliance or looking for a substitute for RS-232 connectivity to your next embedded device, TCP/IP

**Listing 5**—You need to tell WATTCP a little about your network. This is done by creating an ASCII file (WATTCP.CFG) and storing it on the TS-2800 flash memory disk. After the packet driver is loaded and the network socket is initialized, data is free to flow through the Ethernet port.

```
//Substitute with your IP
print="Embedded MicroC/OS-II TCP/IP"
my_ip=192.168.0.7
hostname="mts.com"
netmask=255.255.255.0
```

**Listing 6**—The network GUI application is not much different than using the RS-232 serial port. Data can be sent and received. This is just a little faster.

```
Private Sub LedOff_Click(Index As Integer)
RTOS_UDP.SendData "b" //Turn off LED
End Sub
Private Sub LedOn_Click(Index As Integer)
RTOS_UDP.SendData "a" //Turn on LED
End Sub
Private Sub RTOS_UDP_DataArrival(ByVal bytesTotal As Long)
Dim strData As String
RTOS_UDP.GetData strData
Select Case Left$(strData, 3)
Case "01A"
Reading = Mid$(strData, 4, 9)
Label18.Caption = Reading
Case "CH1"
Reading = Mid$(strData, 4, 7)
Label8.Caption = Reading
Case "CH2"
Reading = Mid$(strData, 4, 7)
Label11.Caption = Reading
Case "ST1" //Kernel statistics
Reading = Mid$(strData, 4, 7)
Text1.Text = Reading
Case "ST2"
Reading = Mid$(strData, 4, 7)
Text2.Text = Reading
Case "ST3"
End Select
```



**Photo 3**—Configuring a NIC card in a desktop PC to talk TCP/IP is completely painless under Win95/98. Simply edit the TCP/IP properties IP address tab and assign your IP address and sub-network mask.

certainly has its advantages. Imagine designing your next high-speed data acquisition controller that allowed several concurrent connections to the same box. It's possible with TCP/IP.

The WATTCP TCP/IP protocol stack is an open-source alternative to rolling your own or purchasing a commercial suite. It can be easily ported to your favorite kernel. Whether your embedded device is standalone or computer-controlled, TCP/IP is just a socket call away. ☒

*Robert Bowen has an Associates Degree in Electronic Technology and Electronic Engineering. He has worked for AT&T Consumer Products designing test equipment. Currently, he works for MTS Systems Corp. as a field service engineer designing automated calibration equipment and developing testing methods for customers involved in the material and simulation testing fields. In his spare time, he enjoys being an amateur radio operator (call sign N0PAU) and tinkering with Linux.*

## RESOURCES

J. Labrosse, *MicroC/OS-II, The Real-Time Kernel*, CMP Books, Gilroy, CA, November 1998.

R. Bowen, "Porting MicroC/OS-II to the TS-2800 Embedded PC," *Circuit Cellar Online*, June 2001.

## SOURCES

**CS8900A Ethernet controller**  
Cirrus Logic

(800) 888-5016  
 (512) 445-7222  
 Fax: (512) 445-7581  
 www.cirrus.com

**WATTCP TCP/IP**

Erick Engelke  
 www.wattcp.com

**TS-2800 Embedded PC**

Technologic Systems, Inc.  
 (480) 837-5200  
 Fax: (480) 837-5300  
 www.embeddedx86.com



**Photo 4**—The GUI resides on the host computer. All of the real-time work is handled on the TS-2800, which is powered by the MicroC/OS-II application.



**Photo 5**—Commands from the GUI are sent down the Ethernet pipe via UDP datagrams. A/D readings, digital input status, and kernel statistics are sent from the TS-2800 to the PC via UDP datagrams Ethernet-style and embedded.

Task	Description	Priority
TaskStart	Start up task	10
1	Tx, Tx of UDP datagram to host computer	11
2	Reads A/D readings from channels one and two, monitors digital inputs one through four, displays A/D readings on LCD	12

**Table 1**—Task 1 is the highest priority task and is responsible for transmitting and receiving data from the TS-2800 to the host computer. Task 2 is responsible for reading the A/D and digital inputs readings. The A/D readings are sent to the LCD as well as to the host computer for display.