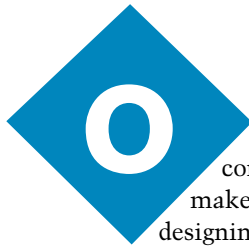


## FEATURE ARTICLE

Robert Bowen

# Porting MircoC/OS-II to the TS-2800 Embedded PC

As his designs started getting more complicated, and he found they were not reusable from system to system, Robert began to turn his focus to new development programs. Demonstrating the advantages of using an embedded PC and real-time kernel, he took advantage of open source programming tools and searched for a real-time executive. It all makes for more capability, complex I/O, and high-speed data acquisition.



Okay, I have a confession to make. I have been designing embedded systems for over 10 years, and I have never used an off-the-shelf single board computer (SBC) or real-time kernel in any of my projects. Why? My designs are simple. They don't warrant the luxury of an RTOS. I don't have time to learn how a kernel works. For years, I've been struggling to overcome these inadequate excuses.

The 8751 microcontroller has been the heart of all my projects. If I needed an RS-232 interface, I added the popular MAX232 IC. Need more digital I/Os? No problem! Toss in your favorite flavor 8255 programmable peripheral interface (PPI).

My firmware is written using the super-loop concept. During powerup, the processor performs the usual housekeeping and settles into Idle mode. When an interrupt occurs, the processor executes the command received and the process starts again.

So, what's the problem? My designs started getting more complicated. Users wanted network capability, complex I/O such as servo motor controllers, and high-speed data

acquisition. Had I incorporated a single board computer that supported stackable expansion plug-ins, upgrading would have been trivial. Additionally, I found that my designs were not reusable from system to system. And, my firmware lacked the advantages of a real-time kernel.

My goal was to develop a new design platform, using an off-the-shelf embedded PC (ePC) and a real-time multitasking kernel. I took advantage of any open-source programming tools and software and used the Internet to search for a low-cost ePC and real-time executive.

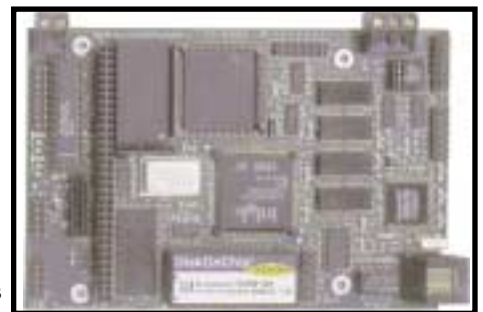
### WHERE TO BEGIN

Prior to starting the 'Net hunt, I defined a few hardware and software parameters that the search should render. On the hardware side, the embedded PC should have, at minimum, the following:

- i386EX microprocessor
- a few onboard digital I/Os
- 12-bit ADC
- Ethernet port

For a real-time kernel, the only two requirements are:

- open source
- excellent documentation



**Photo 1**—The Technologic Systems TS-2800 embedded PC came with all the necessary embedded ingredients, including flash memory, digital I/Os, PC/104 expansion, Ethernet port, A/D, LCD interface, and serial ports.



**Photo 2**—The TS-2800 redirects all console activity to the COM2 serial port. Connect an ANSI terminal to COM2 and all text information that is normally displayed on a video screen is now displayed on the terminal window.

After several exhausting hours of surfing the 'Net, I finally settled on the TS-2800 for my embedded PC (the full development kit was under \$300). Onboard this tightly packed 3.6" × 3.25" PCB is an i386EX microprocessor with the following trimmings (see Photo 1):

- 8-MB DRAM
- 1-MB flash memory
- up to 144 MB of flash memory expansion using M-Systems DiskOnChip 2000
- battery-backed SRAM
- two PC-compatible asynchronous serial ports
- several available digital I/Os
- LCD interface
- 12-bit ADC
- 10BaseT Ethernet port
- ROM-DOS BIOS

Perfect! The TS-2800 exceeded all my requirements and came with extra goodies for dessert. Now that I have the hardware, I'll describe what kind of luck I had searching for a multi-tasking kernel.

My Web search for a real-time executive proved to be a daunting experience. There were literally hundreds to choose from. However, because I decided to eliminate any non-open-source kernels, my list was quickly reduced to a handful.

I chose MicroC/OS-II written by Jean J.

Labrosse. For \$60, I received a hard cover book titled *MicroC/OS-II: The Real-Time Kernel*. [1] The book describes the internals of a portable, ROMable, scalable, preemptive RTOS. Included is a CD containing all the source code for MicroC/OS-II. I highly recommend getting this book for your library even if you choose to develop with a different kernel.

I also purchased another book by Labrosse entitled *Embedded Systems Building Blocks*. [2] This book included complete ready-to-use modules in C, such as display interfaces, timers, and I/Os. Perfect for my new design platform.

The only thing missing from this kernel was a TCP/IP stack. I knew eventually I would want another way to communicate with my ePC other than the serial port. Fortunately, the TS-2800 came with a free TCP/IP programming library with full source code. I had to do the port to the kernel myself. But, the price was right!

Although my search took a couple of days, things went fairly well. I didn't have to spend a fortune for the hardware or kernel. So with that said, let's take a look at the development tools required before we start playing.

## DEVELOPMENT TOOLS

MicroC/OS-II is written in C, so I needed a C compiler. Not a problem, I already have a copy of Borland C++ V.3.1.

Most of my designs (black boxes) interface with a desktop PC. The PC runs the GUI, and all the low-level functions (data acquisition, digital I/Os, etc.) are performed on an embedded processor. I continued to do the same with this new design platform.



**Photo 3**—Remember DOS? The TS-2800 presents a familiar desktop command environment to the embedded world for easy product development.

The TS-2800 hosts the real-time kernel and is responsible for reading the ADCs, controlling and monitoring the digital I/Os, and sending and receiving command data to and from the desktop PC. This keeps the desktop CPU free to do other tasks and independent of the OS. I used Visual Basic 6.0 to develop the graphical user interface.

Developing a MicroC/OS-II application is just like any other DOS application. You write your source code, compile it, and link it to generate a standard DOS executable file (.exe). This will work nicely with the TS-2800 ePC because it comes with DOS pre-installed. I used my desktop PC to write the real-time application and then uploaded the executable to the TS-2800.

Fortunately, I didn't need to purchase any new development tools. So, let's fire up the new TS-2800-embedded PC and take it for a spin.

## BOOTING THE TS-2800

Getting the TS-2800 up and running was quite painless. Photo 2 shows the TS-2800 booting. It looks just like a standard desktop PC going through the system RAM verification.

After the system boots, you are placed at the standard DOS prompt. The TS-2800 is now ready for your command. Typing "DIR" will display the directory on your flash drive (A or C, however you decide to partition it).

TS-2800 hardware resource	Description
Serial Port 1 (COM1)	Interface to desktop PC
Serial Port 2 (COM2)	Upload kernel and debug port
LCD	Display system messages and ADC readings.
Digital outputs 1, 2, 3, and 4	Standard TTL control logic. Connect to digital inputs 1, 2, 3, and 4
Digital inputs 1, 2, 3, and 4	Use to read the status of digital out puts 1, 2, 3, and 4
12-bit ADC	Channel 1 and Channel 2
LED	Toggle for service condition

**Table 1**—This table defines the hardware resources used in the first real-time application. The MicroC/OS-II kernel provides the drivers for the COM ports and LCD interface.



**Photo 4**—The MicroC/OS-II installation created a well-organized desktop environment. All the drivers are placed in the appropriate directory with several examples to assist in creating your first multitasking application.

“MD” for Make Directory will create a sub-directory on the system flash memory. Photo 3 shows the contents of drive A (flash drive). All the standard DOS commands are left at your disposal.

The TS-2800 uses COM2 for the console redirection. This feature will make uploading and testing the finished application a breeze. We’ll create our real-time application on the desktop PC, then upload our compiled kernel to the TS-2800 for execution. The embedded PC looks a lot like what I had expected.

I think it’s time to install the MicroC/OS-II kernel on a desktop PC and create the first application.

## INSTALLING THE MICROC/OS-II

The installation of the kernel was easy and clean. I simply unzipped the *uCOS-II.ZIP* file to my D drive, and the installation script created a sub-directory, *D:\Software*.

Before diving into the source code, I decided to install the CD that came with *Embedded Systems Building Blocks* because I knew that I would want to use the LCD and serial communications drivers right away.

The installation went just as smoothly as the kernel installation. A sub-directory, *D:\Software\Blocks*, was created automatically. This directory contains all the drivers I need for any embedded project, including analog I/O, clocks, serial communication, timers, key pads, LCD, and seven-segment LED display drivers (see Photo 4).

That’s it! The kernel is completely installed and ready for coding.

## CREATING A MICROC/OS-II APPLICATION

I’m purposely going to omit discussing the internals of a kernel (semaphores, context switching, etc.) The books I referenced do an excellent job of describing this. I want to focus on creating a preemptive multitasking application. Let’s begin by defining what hardware resources are

used on the TS-2800 (see Table 1).

I wanted to use all of the resources available on the TS-2800 with my first MicroC/OS-II application, as well as make use of some of the drivers and features supplied with the kernel.

On the TS-2800, I use serial port one to communicate with the desktop PC to send and receive command data. Serial port two is used for uploading the kernel application to the embedded PC, as well as displaying system status and error messages. The LCD screen displays all commands sent to the embedded PC from the host computer, as well as displaying readings from the ADC. The four digital outputs are tied directly to the four digital inputs. Finally, I toggled the onboard LED to display service conditions.

The MicroC/OS-II application has three concurrently running tasks (see Table 2).

Now that the hardware resources

used are defined and the functions are broken down into three tasks, you can start coding. Task 1 is the highest priority task and is responsible for monitoring serial port one for commands received from the host computer.

When a valid command is received, a Switch statement then branches to the appropriate routine for execution.

Task 2 is the next highest priority task and is responsible for reading channel one and channel two of the onboard ADC. The results are sent to the host computer as well as to the LCD via serial port one.

Task 3 runs as the lowest priority task in this application. Its sole purpose is to display the message heading “MicroC/OS-II” on the LCD, as well as the infamous spinning wheel.

Next, let’s compile the kernel and send it to the TS-2800 for execution. The GUI that resides on the host desktop computer can be worked on now.

## CREATING THE GUI

As I mentioned earlier, most of my designs are connected to a desktop computer in some form, maybe via the serial or parallel port. In this case, the serial port is used because one already exists on the TS-2800. And, I created a GUI that runs on a standard desktop or laptop PC running Bill’s Windows 9x.

The GUI consists of two sub-routines. Routine 1 allows you to manually send commands to the TS-2800 to set or clear all four digital outputs and to turn on and off the onboard LED. This is accomplished by pressing the command buttons labeled D/O 1 Set, D/O 1 Clear, and LED On/Off accordingly.

The task bar displays the function being executed as:

- Reading serial port
- Reading channel one
- Reading channel two
- Reading digital inputs

Photo 5 shows a snapshot of the GUI. Notice the *uCOS-II* command button. When pressed, this button sends a command to the TS-2800 to display the kernel version.



**Photo 5**—This simple Visual Basic form provides an easy-to-use interface to the TS-2800 for software development.

Task	Description	Priority
1 (see Listing 1)	Receives (Rx) serial communication from includes all system commands to set and clear the four digital outputs and turn on and off the onboard LED. All commands sent from the host computer are displayed on the LCD screen.	10
2 (see Listing 2)	Transmits (Tx) serial data to the host PC. 11 Reads CH1 and CH2 of the onboard ADC Reads digital inputs 1-4. Readings are automatically sent to the serial port for host computer and to the LCD.	11
3 (see Listing 3)	Displays the infamous spinning wheel on the LCD screen and a welcome message (MicroC/OS-II).	12

Table 2—Splitting the work up into separate tasks is part of the design process for a real-time application.

The results are sent to serial port two on the TS-2800 and can be viewed by a connected terminal or PC.

Photo 6 shows a snapshot of this message. This feature can include any timeout, error, or system status messages that would render useful for system diagnostics in the field. For this application, I displayed a message indicating the status of a valid command received from the host computer.

When the application is launched, subroutine two fires off a Timer function to automatically read data from the serial port. This includes the ADC readings from channel one and channel two on the TS-2800, as well as the status of the four digital inputs. The results are clearly displayed on the main form of the GUI.

For testing, I tied all the digital outputs to the digital inputs on the TS-2800. I can now set or clear each bit individually and watch the display

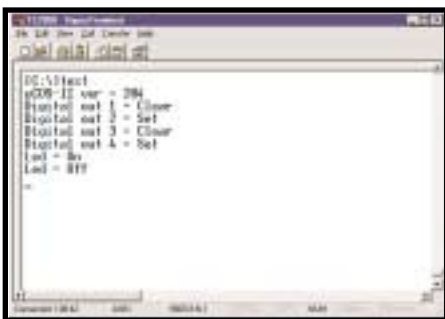


Photo 6—The TS-2800 console redirection provides an easy-to-use debug port for development and system troubleshooting. Your application can send error messages and kernel statistics via this COM port.

update automatically. When depressed, the LED ON and LED OFF buttons illuminate the LED on the TS-2800 accordingly.

Finally, I connected a variable DC power supply to channel one and channel two's ADC. As I vary the voltage, I can observe the panel meters update automatically.

## FUTURE GOALS

There are still a few more things I'd like to accomplish with my new development platform, such as installing a web server on either the desktop PC or the TS-2800 to allow for web-based control and monitoring. I'd also like to eliminate using the serial port and take advantage of the onboard Ethernet port. This would allow for a higher data rate between the TS-2800 and desktop PC, among other things.

This embedded project was fun and clearly demonstrates the advantages of using an embedded PC and real-time kernel. The finished project can be seen in Photo 7. I'm looking forward to my next design. Buried deep inside my next embedded black box will be a full preemptive multitasking kernel! ☺

*Robert Bowen has his Associates Degree in Electronic Technology and Electronic Engineering. In the past, he has worked for AT&T Consumer Products designing test equipment and is currently employed by MTS Systems Corp. as a field service engi-*

*neer designing automated calibration equipment and developing testing methods for customers involved in the material and simulation testing field. In his spare time, Robert enjoys being an amateur radio operator (call sign N0PAU) and tinkering with Linux. He can be reached by e-mail at robert.bowen@mts.com.*

## REFERENCES

- [1] J.J. Labrosse, *MicroC/OS-II: The Real-Time Kernel*, R&D Books, Gilroy, CA, October 1998.
- [2] ———, *Embedded Systems Building Blocks*, 2<sup>nd</sup> ed., R&D Books, Gilroy, CA, December 1999.

## RESOURCE

Technologic Systems  
<http://www.embeddedx86.com>



Photo 7—There are several single board computers on the market. The TS-2800 provides a familiar DOS environment for quickly getting up to speed when developing with a real-time multitasking kernel.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com) or [www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).

**Listing 1**

```
void Task1 (void *data)
{
  UBYTE err;
  INT16U os_version;
  INT8U nbytes;
  INT8U c;
  char s[81];
  char *ps;
  int result;

  DispInit(2,40); /* Init LCD Display 2x40 */
  DispClrScr();

  CommInit(); /* Init COMM port */
  CommCfgPort(COMM1, 9600, 8, COMM_PARITY_NONE, 1);
  CommSetIntVect(COMM1);
  CommRxIntEn(COMM1);
  data = data; /* Prevent compiler warning */
  for (;;) {
    union REGS regs; /* TS-2800 Bios Call */
    ps = s;
    nbytes = 0;
    do {
      c = CommGetChar(COMM1, OS_TICKS_PER_SEC, &err);
      *ps++ = c;
      nbytes++;
    } while (c != '\n' && nbytes < 1);
    *ps = NUL; /* NUL terminate received string */
    OSSemPend(LCDSem, 0, &err); /* Get Semaphore */
    DispStr(0, 0, "cmd:"); /* Print CMD's to LCD Display */
    DispStr(0, 5, s);
    OSSemPost(LCDSem); /* Release Semaphore */

    switch (c) {
      case 'a': /* Turn LED ON. */
      {
        regs.x.ax = 0xB010;
        regs.h.bh = 0x01;
        int86 (0x15, &regs, &regs);
        printf ("Led = On \n");
      }
      break;
      case 'b': /* Turn LED OFF.*/
      {
        regs.x.ax = 0xB010;
        regs.h.bh = 0x00;
        int86 (0x15, &regs, &regs);
        printf ("Led = Off \n");
      }
      break;
      case 'c': /* Set P1.0 = Digital Output 1 */
      {
        result = (inpw(0xf820)) & 0x0fe;
        outpw(0xf820, result);
        result = (inpw(0xf864)) & 0x0fe;
        outp(0xf864, result);
        result = (inpw(0xf862)) | 0x01;
        outpw(0xf862, result);
        printf ("Digital out 1 = Set \n");
      }
      break;
      case 'd': /* Clear P1.0 = Digital Output 1 */
      {
        result = (inpw(0xf862)) & 0xfe;
        outpw(0xf862, result);
        printf ("Digital out 1 = Clear \n");
      }
      break;
      case 'e': /* Set P1.5 = Digital Output 2 */
      {
        result = (inpw(0xf820)) & 0x0df;
        outpw(0xf820, result);
        result = (inpw(0xf864)) & 0x0df;
        outp(0xf864, result);
        result = (inpw(0xf862)) | 0x20;
      }
    }
  }
}
```

Continue

Circuit Cellar, the Magazine for Computer Applications.  
Reprinted by permission. For subscription information,  
call (860) 875-2199, subscribe@circuitscellar.com or  
www.circuitcellar.com/subscribe.htm.

**Listing 1– Continued**

```
    outpw(0xf862, result);
        printf (“Digital out 2 = Set \n”);
        break; }
    case ‘f’: /* Clear P1.5 = Digital Output 2 */
    {
        result = (inpw(0xf862)) & 0xdf;
    outpw(0xf862, result);
        printf (“Digital out 2 = Clear \n”);
    break; }
    case ‘g’: /* Set P3.2 = Digital Output 3 */
    {
        result = (inpw(0xf824)) & 0x0fb;
        outpw(0xf824, result);
    result = (inpw(0xf874)) & 0x0fb;
        outp(0xf874, result);
    result = (inpw(0xf872)) | 0x04;
        outpw(0xf872, result);
        printf (“Digital out 3 = Set \n”);
    break; }
    case ‘h’: /* Clear P3.2 = Digital Output 3 */
    {
    result = (inpw(0xf872)) & 0x0fb;
        outpw(0xf872, result);
        printf (“Digital out 3 = Clear \n”);
    break; }

    case ‘i’: /* Set P3.6 = Digital Output 4 */
    {
        result = (inpw(0xf824)) & 0x0bf;
    outpw(0xf824, result);
    result = (inpw(0xf874)) & 0x0bf;
        outp(0xf874, result);
    result = (inpw(0xf872)) | 0x40;
        outpw(0xf872, result);
        printf (“Digital out 4 = Set \n”);
    break; }
    case ‘j’: /* Clear P3.6 = Digital Output 4 */
    {
    result = (inpw(0xf872)) & 0x0bf;
        outpw(0xf872, result);
        printf (“Digital out 4 = Clear \n”);
        break; }
    case ‘k’: /* Display OS version */
    {
    os_version = OSVersion();
        printf (“uCOS-II ver = %d\n”, os_version);
        break; }
    }
}
```

**Listing 2**

```
void Task2 (void *data)
{
    UBYTE err;
    int msb;
    int lsb;
    int DigitalInput1;
    char s[80];
    char *ps;
    data = data;
    for (;;) {
```

Continued

**Listing 2- continued**

```
float ADC2Reading;
float ADCReading;
float FullScale = 5.000;
float Resolution = 2048.000;
/* Channel 1 ADC Reading */
outp(0x078, 0x11);
OSTimeDlyHMSM(0, 0, 0, 1);
msb = (inpw(0x79));
    lsb = (inpw(0x78));
ADCReading = (((msb | lsb) *FullScale)/Resolution);
sprintf(s, "CHA %4.2f",ADCReading );
OSSemPend(LCDSem, 0, &err);
DispStr(1, 0, s);
OSSemPost(LCDSem);
/* Tx CH1 reading to Host Computer */
ps = s;
while (*ps != NULL) {
CommPutChar(COMM1,*ps, OS_TICKS_PER_SEC);
OSTimeDly(5);
ps++;
}
/* Channel 2 ADC Reading */
outp(0x078, 0x12);
OSTimeDlyHMSM(0, 0, 0, 1);
msb = (inpw(0x79));
    lsb = (inpw(0x78));
ADC2Reading = (((msb | lsb)*FullScale)/Resolution);
sprintf(s, "CHB %4.2f",ADC2Reading );
OSSemPend(LCDSem, 0, &err);
DispStr(1, 31, s);
OSSemPost(LCDSem);
ps = s;
/* Tx CH2 reading to Host Computer */
while (*ps != NULL) {
CommPutChar(COMM1,*ps, OS_TICKS_PER_SEC);
OSTimeDly(5);
ps++;
}
/* Read Digital Inputs 1-4 */
DigitalInput1 = (inp(0x77));
sprintf(s,"INP %d\n", DigitalInput1);
OSSemPend(LCDSem, 0, &err);
    DispStr(1, 15, s );
    OSSemPost(LCDSem);
ps = s;
/* Tx DI 1-4 to Host Computer */
while (*ps != NULL) {
CommPutChar(COMM1,*ps, OS_TICKS_PER_SEC);
OSTimeDly(5);
ps++;
}
}
```

**Listing 3**

```
void Task3 (void *data)
{
    UBYTE err;
    DispInit(2,40);
    DispClrScr();
}
```

Continue

Circuit Cellar, the Magazine for Computer Applications.  
Reprinted by permission. For subscription information,  
call (860) 875-2199, [subscribe@circuitscellar.com](mailto:subscribe@circuitscellar.com) or  
[www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).

**Listing 3—Continued**

```
data = data;

for (;;) {
    OSSEmpend(LCDSem, 0, &err);
    /* Get Semaphore for LCD */
    DispStr(0, 13, "MicroC/OS-II");
    /* Display Msg on LCD */
    DispStr(0, 38, "|");
    /* Display spinning wheel on LCD */
    OSTimeDly(10);
    DispStr(0, 38, "/");
    OSTimeDly(10);
    DispStr(0, 38, "-");
    OSTimeDly(10);
    DispStr(0, 38, "|");
    OSTimeDly(10);
    /* Release Semaphore */
    OSSemPost(LCDSem);
}
}
```