C/C++ EV/DV Software Kit

for

TERN 16-bit Embedded Microcontrollers

Technical Manual

1724 Picasso Avenue, Suite A, Davis, CA 95616, USA
Tel: 530-758-0180 Fax: 530-758-0181
Internet Email: sales@tern.com ***http://www.tern.com***

COPYRIGHT

A-Engine, A-Core, V25-Engine are trademarks of TERN, Inc.
LOC31, TD31, BC31, TDREM-xx, EV, and DV are trademarks of TERN, Inc.
V25 is a Trademark of NEC Corporation.
Am188ES is a Trademark of Advanced Micro Devices, Inc.
Borland C++ 3.1 and Turbo Debugger are trademarks of Borland International.
Microsoft, MS-DOS and Windows 3.1, Windows 95, and Windows 98 are trademarks of Microsoft Corporation.
Intel and 386EX are trademarks of Intel Corporation.

Version 2.00

July 15, 1998

© 1997-1998 TERN INC.
1724 Picasso Avenue, Suite A, Davis, CA 95616, USA
Tel: 530 758 0180 Fax: 530 758 0181
Internet Email: tern@netcom.com ***http://www.tern.com***

Important Notice

TERN is developing complex high technology integration systems. These systems are integrated with software and hardware that are not 100% defect free. ***TERN products are not designed, intended, authorized, or warranted to be suitable for use in life-support applications, devices or systems or other critical applications. TERN*** and the Buyer agree that ***TERN*** will not be liable for incidental or consequential damages arising from the use of ***TERN*** products. It is the Buyer's responsibility to protect life and property against incidental failure.

TERN reserves the right to make changes and improvements to its products without providing notice.

TERN will be responsible for technical support of this kit, NOT Borland. Do not contact Borland International for technical support assistance.

By installing the software on the floppy diskettes, you are signifying that you agree with the terms of the following end-user license:

LICENSE STATEMENT AND LIMITED WARRANTY FOR BORLAND PRODUCTS

Borland grants you the right to use this Borland software product ("Software"), including Borland on-line documentation ("Documentation"), in the manner provided below.

This Software is owned by Borland or its suppliers and is protected by copyright law and international copyright treaty. Therefore, you must treat this Software like any other copyrighted material (e.g., a book), except that you may either make one copy of the Software solely for backup or archival purposes or transfer the Software to a single hard disk provided you keep the original solely for backup or archival purposes.

Though Borland does not offer technical support for the Software, they welcome your feedback.

You may transfer the Software and Documentation on a permanent basis provided you retain no copies and the recipient agrees to the terms of this license statement. Except as provided in this statement, you may not transfer, rent, lease, land, copy, modify, translate, sublicense, time-share or electronically transmit or receive the Software, media or Documentation. You acknowledge that the Software in source code form remains a confidential trade secret of Borland and/or its suppliers and therefore you agree not to modify the Software or attempt to decipher, decompile, disassemble or reverse engineer the Software, except to the extent applicable laws specifically prohibit such restriction.

If you have purchased an upgrade version of the Software, it constitutes a single product with the Borland software that you upgraded. You may use or transfer the upgrade version of the Software only in accordance with this license statement.

This Software is subject to U.S. Commerce Department export restrictions, and is intended for use in the country into which Borland sold it (or in the EEC, if sold into the EEC).

GENERAL TERMS THAT APPLY TO COMPILED PROGRAMS AND REDISTRIBUTABLES

You may write and compile your own application programs using the Software, including any libraries and source code included for such purpose with the Software. You may reproduce and distribute, in executable form only, programs which you create using the Software without additional license or fees, subject to all of the conditions in this statement.

Borland products may include certain files ("Redistributables") intended for distribution by you to the users of programs you create. The Redistributables for the Software, if any, are only those files specifically designated as such by Borland in the Documentation included with the Software. From time to time, Borland may designate other files as Redistributables. You should refer to the Documentation, including any "readme" or "deploy" files included with the Software, for additional information.

Subject to all of the conditions in this statement, you may reproduce and distribute exact copies of the Redistributables, provided that such copies are made from the original copy of the Software or the copy transferred to the single hard disk. Copies of Redistributables may only be distributed with and for the sole purpose of executing application programs permitted under this statement that you have created using the Software. Under no circumstances may any copies of Redistributables be distributed separately. You may not reproduce or distribute any Documentation without Borland's permission.

The license granted in this statement for you to create your own compiled programs and distribute your programs and the Redistributables (if any) is subject to all of the following conditions: (i) all copies of the programs you create must bear a valid copyright notice, either your own or the Borland copyright notice that appears on the Software; (ii) you may not remove or alter any Borland copyright, trademark or other proprietary rights notice contained in any portion of Borland libraries, source code, Redistributables or other files that bear such a notice; (iii) Borland provides no warranty at all to any person, other than the Limited Warranty provided to the original purchaser of the Software, and you will remain solely responsible to anyone receiving your programs for support, service, upgrades, or technical or other assistance, and such recipients will have no right to contact Borland for such services or assistance; (iv) you will indemnify and hold Borland, its related companies and its suppliers harmless from and against any claims or liabilities arising out of the use, reproduction or distribution of your programs; (v) your programs must be written using a licensed, registered copy of the Software; (vi) your programs may not be merely a set or subset of any of the libraries, code, Redistributables or other files of the Software; and (vii) you may not use Borland's or any of its suppliers' names, logos, or trademarks to market your programs, except to state that your program was written using the Software.

All Borland libraries, source code, Redistributables and other files remain Borland's exclusive property. Regardless of any modifications that you make, you may not distribute any files (particularly Borland source code and other non-executable files) except those that Borland has expressly designated as Redistributables. Nothing in this license statement permits you to derive the source code of files that Borland has provided to you in executable form only, or to reproduce, modify, use, or distribute the source code of such files. You are not, of course, restricted from distributing source code that is entirely your own. Code that you generate with a Borland code generator, such as AppExpert, is considered by Borland to be your code.

LIMITED WARRANTY

Except with respect to the Redistributables, which are provided "as is," without warranty of any kind, the Vendor from whom you received the Software warrants that the Software media will be free from defects in materials and workmanship, for a period of ninety (90) days from the date of receipt. Any implied warranties on the Software are limited to ninety (90) days. Some states/jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

Borland's, its suppliers', and the Vendor's entire liability and your exclusive remedy shall be repair or replacement of the Software media that does not meet the Limited Warranty and which is returned to the Vendor with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, or misapplication. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Outside the United States, neither these remedies nor any product support services offered by Borland are available without proof of purchase from an authorized non-U.S. source.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, BORLAND AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH REGARD TO THE SOFTWARE AND THE ACCOMPANYING DOCUMENTATION. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL BORLAND OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THIS BORLAND PRODUCT, EVEN IF BORLAND HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES/JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATIONOF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

HIGH RISK ACTIVITIES

The Software is not fault-tolerant and is not designed, manufactured, or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Borland and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

U.S. GOVERNMENT RESTRICTED RIGHTS

The Software and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraphs (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c) (1) (ii) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable.

GENERAL PROVISIONS

This statement may only be modified in writing signed by you and an authorized officer of Borland. If any provision of this statement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed for its essential purpose, all limitations of liability and exclusions of damages set forth in the Limited Warranty shall remain in effect.

This statement shall be construed, interpreted, and governed by the laws of the State of California, U.S.A. This statement gives you specific legal rights; you may have others that vary from state to state and from country to country. Borland reserves all rights not specifically granted in this statement.

CHAPTER 1 *Introduction* ***1-1***

TERN Controllers and Borland C/C++ 3.1 **1-2**
What is in the Evaluation (EV) and Development (DV) kits? ***1-2***
Borland Technical Manual/Support ***1-3***
Application Development Process **1-4**
Step One ***1-4***
Step Two ***1-5***
Step Three ***1-5***
Other Considerations **1-7**
Using C++ with the TERN development kits ***1-7***
System Requirements ***1-7***
Code Size and Large Model Libraries ***1-7***

CHAPTER 2 *Installation* ***2-1***

Before Installing **2-2**
Operating Systems ***2-2***
TERN Disks ***2-2***
Installation Process - Windows 95/98 Version **2-3**
Installation Process - DOS/Win3.1 Version **2-4**
Borland Installation ***2-4***
Evaluation/Development Kit Installation ***2-5***
Follow-up Procedures **2-6**
Checking your installation ***2-6***
Cleaning up your environment ***2-7***
Hardware Installation **2-9**
Installation Troubleshooting **2-10**

CHAPTER 3 *Tutorial* ***3-1***

The Sample Program - LED.C **3-2**
Step One - Download and Debug **3-3**
PC Environment ***3-3***
Step One ***3-3***
Modifying led.c ***3-6***
Troubleshooting Step One ***3-7***

Step Two - Field Test **3-11**
Step Two Jump Routine ***3-11***
step2.c ***3-12***
Troubleshooting Step Two ***3-12***
Step Three: Production (DV Kit Only) **3-13**
Other programs **3-14**
Using TERN Development Kits with other software ***3-14***
Sample files ***3-15***

CHAPTER 4 *Development and Debugging* ***4-1***

Development Batchfiles **4-2**
m.bat ***4-2***
t.bat ***4-3***
Memory Mapping **4-5**
Physical Memory Mapping ***4-5***
Locate Configuration File ***4-5***
Makefile Options **4-8**
USER_OBJS ***4-8***
BOARD ***4-8***
EXTENSION ***4-8***
MEMCARD ***4-9***
COMPDIR ***4-9***
EPROM/SRAM/CPU ***4-9***
OPTIMIZE ***4-9***
Optimizing Your Code **4-10**

CHAPTER 1

Introduction

This chapter introduces and familiarizes the user with the motivation behind the Evaluation/Development Software Kits offered by TERN.

It offers general guidance regarding the application development model that is recommended for use with TERN controllers, as well as providing an overview of the unique features offered by the TERN software kits.

1.1 TERN Controllers and Borland C/C++ 3.1

C is a popular, high-level language, preferred by many professional programmers, and is also an excellent choice for your first programming language. A powerful, flexible, portable, and modular language, C is highly suitable for embedded controller programming.

While many major C compilers in the market are moving to supporting only 32-bit applications, Borland International has granted TERN a license to reproduce and to distribute their 16-bit Borland C/C++ 3.1 compiler, in conjunction with the "TERN 16-bit C/C++ Embedded Development Kit" (EV/DV Kit).

Borland C/C++ 3.1 (BC31) is one of the best 16-bit ANSI C compilers available today. It generates reliable, compact and fast 80x86 code, ideal for TERN's 16-bit controllers. The BC31 compiler also provides excellent code generation, run-time libraries, and documentation. The EV/DV kit includes a special version of BC31, TASM, IDE, and Turbo Remote Debugger (TD31).

Since TERN's controllers use PC-compatible processors (the NEC V25, AM188ES, and Intel 386 SX), they are especially easy to program in C with present software environments.

Technical support is provided by TERN, offering prompt responses and seamless software/hardware technical advice. You can also use a large number of widely available PC-based application programs as tools . Since you are not tied to a particular non-PC-compatible processor, you are free to choose the tools with which you are most comfortable.

Contact TERN technical support either via email at
tech@tern.com
by phone at
530-758-0180
or by fax at
530-758-0181
We are willing to help you develop your application.

With TERN controllers' PC-compatible processors, you will find more tools, better tools, and more cost-effective tools than those available for other embedded controllers. As a result, you do not have to waste your time and money learning other proprietary embedded tools.

1.1.a What is in the Evaluation (EV) and Development (DV) kits?

The Evaluation Kit (EV) is designed for the first-time buyer who wants to make a prototype stand-alone unit for demonstration within a limited budget. The EV Kit allows you to download, remotely debug and run your C/C++ program on TERN controllers.

In addition to the EV Kit features, the C/C++ Development Kit (DV) provides all the extra support that you will need to generate your own application ROM, and complete your project with TERN controllers. The DV Kit includes a full version of LOC31, the program that properly locates the address references in your code.

The DV Kit provides the ability to generate application ROM files for OEM products. TERN will provide extensive technical support for DV Kit owners.

If you find yourself needing the additional capabilities offered by the DV kit, you can always upgrade from the EV kit at a later time.

1.1.b Borland Technical Manual/Support

The Borland C/C++ Compiler Technical Manual is not included with the software development kits.

You can contact SAMS Publishing for this title at 1-800-428-5331.

The best source of assistance with the IDE (**bc.exe**) or with Turbo Debugger (**td.exe**) can be found by selecting **Help** from inside the programs.

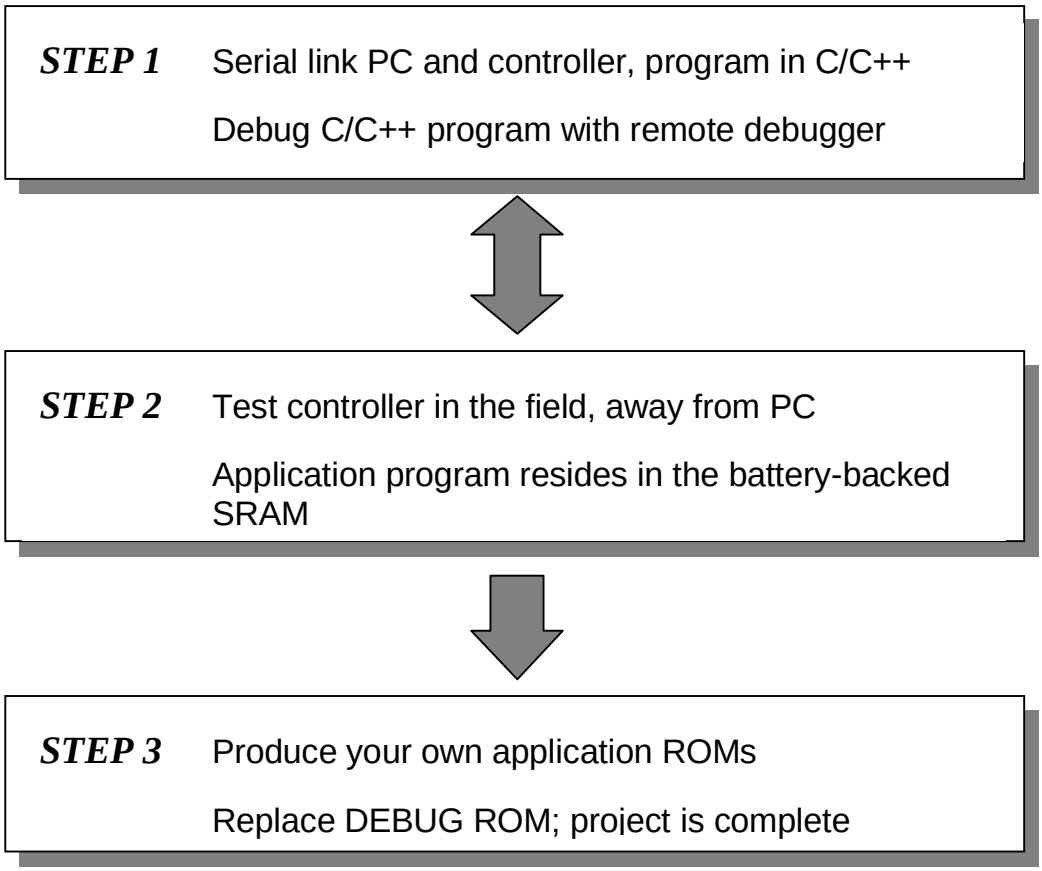
*ISBN for the **C Programming Starter Kit** is 0-672-30996-3.*

If you just need more help with programming in C, you might want to order the **C Programming Starter Kit** from SAMS Publishing. This includes an excellent technical manual titled “Teach Yourself C in 21 Days” as well as the Borland 4.0 Electronic Manual on a CD-ROM.

1.2 Application Development Process

The development of your application involve three simple conceptual steps. If you become acquainted with the overall picture of how to complete your project, it becomes easier to develop code and communicate regarding your project with the application engineers at TERN.

FIGURE 1.1 Three Steps of Application Development



1.2.a Step One

The first step consists of application development and real-time remote debugging.

In this step, you can begin to develop your application using the software interfaces offered by the TERN libraries. TERN has tried to provide sample files that provide thorough coverage of the components on all of its controllers. Engineers can use these sample files as guidelines for creating larger applications that conform to your specifications.

The application development process in this step consists of downloading your application over a serial RS232 connection from one of the COM-ports available on the PC to the RS232 port on your controller. This can be done using simplified and comprehensive makefiles and batch files that are provided as part of the software development kits. The application can be downloaded directly into SRAM by use of the DEBUG ROM also provided as part of the software development kits.

Using Borland Turbo Remote Debugger, a full-featured debugger, you can then remotely debug your application by setting break points, single-stepping, register-checking, or simply running your application.

The tutorial chapter in this manual describes this first step in great detail.

At the completion of this first step, you should have a fully debugged and running program.

1.2.b Step Two

*The location of the **Step Two** jumper differs based on the controller.*

This and other hardware details can be found in the technical manual for the specific controller itself.

Your debugged program can now be tested in the field.

By installing the Step Two jumper, you can choose to run the program that was downloaded into battery-backed SRAM during Step One at power-up/reboot.

You can now disconnect your controller away from the PC and test it in field conditions. If at any point you need to debug your application or download another, you can simply return to Step One for further development.

The battery-backed SRAM will actually hold your application for 3-5 years under normal operation conditions. This is sufficient for certain applications, and they do not need to proceed to the third step.

1.2.c Step Three

Once you have completed your application testing period, and are confident that it works the way you wish, you can move on to the production step.

If you are dealing with large volumes of controllers, or require a non-volatile medium for your end-product (i.e. Flash or ROM), you will need to be able to generate **.HEX** and **.BIN** files. Using a standard ROM programmer, you can produce Flash/ROMs to be placed directly into the TERN ROM socket.

It is also possible to program Flash directly on board the controller through the serial-port. The ACTF Kit makes it possible to download the generated **.HEX** file and place it anywhere in Flash you wish. It then becomes possible to place several applications in Flash, and to choose to run from a specific address through a simple menu based interface over the serial-port. For further details regarding the ACTF Kit, please see the ACTF Kit technical manual.

*To generate **.HEX** and **.BIN** files, you will need to make sure you have the Development (DV) version of the software development kits offered by TERN.*

1.3 Other Considerations

1.3.a Using C++ with the TERN development kits

Borland C/C++ 3.1 is a great compiler for programming in C++ as well. Using an object-oriented approach in embedded design has many advantages, as it allows more complex application design with greater code abstraction.

The TERN header files/libraries are written in C, and in order to compile it with C++, you must ***#include*** the header files in a special manner.

You will need to declare the header files as **extern "C"**. Doing this is a statement as simple as:

```
extern "C" {
  #include "ve.h"
}
```

This allows you to include the same header files as if you were programming in C.

1.3.b System Requirements

You will need to have a working PC with at least one working serial-port. This means the com-port must be open to applications in DOS, and is currently unused by any other applications.

Slower DEBUG ROM*s are also provided in your EV/DV kit. If you look in the* ***rom*** *subdirectory of the working directory for your controller, you should find several* ***.BIN/.HEX*** *files that are versions of the DEBUG ROM operating at slower baud rates.*

If you have an older PC, you also need to be aware that a high speed UART is needed to communicate at high baud rates. For example, by default, the DEBUG ROM provided by TERN communicates with the PC at 115,200 baud. On some older machines, this might cause the PC to lock. If this occurs, you might need a slower DEBUG ROM, a faster serial port, or a new machine.

MS-DOS, Windows 3.1, and Windows 95/98 are preferred platforms. Since Borland C/C++ 3.1 is a 16-bit DOS application, the true DOS environment offered by these operating systems are preferred to others.

1.3.c Code Size and Large Model Libraries

TERN software libraries are distributed in small model form. In most cases, this makes code run more efficiently and decreases code size tremendously.

However, there is a major limitation to using small model code. The data and code segments are restricted to 64 KB in length. There are still several ways to access more than 64 KB of data. The controller technical manuals describe this in more detail, but one option is to use ***peekb()*** and ***pokeb()*** to access additional data memory space.

Almost all of our users have found that 64 KB of code segment length is sufficient for their application. If your code no longer fits within this fixed-size segment length, you can contact TERN regarding the possibility of purchasing a large model library. Depending on your specific application, this might not be available.

CHAPTER 2

Installation

This chapter details the initial installation of the TERN software development kits.

Following the procedures described below, you can create a complete software environment from which you can develop, download, and debug your code using simple steps.

This manual should be used together with the software chapters of the manuals of the specific hardware controllers you have purchased.

2.1 Before Installing

2.1.a Operating Systems

The TERN software development kits should work in any Windows/MS-DOS environment, but there are certain things you as the user should be aware of. TERN batch files are designed for use in a 16-bit MS-DOS environment. If you are using a Windows-based operating system, you will still need to do a large part of your development/installation from the MS-DOS prompt.

*To start a MS-DOS window in Windows 95/98/NT, click on the **Start** button, choose **Program**, and select **MS-DOS Prompt**.*

You can also use Alt-Enter to switch between Full-Screen and Window mode.

Windows 95/98 and NT all limit serial-port access from the MS-DOS environment. It is very possible that you will incorrectly come to the conclusion that your program can not communicate over a certain serial port using a 16-bit MS-DOS based program, for example Turbo Remote Debugger. This occurs if a different Windows-based application has exclusive serial-port access at that moment, or if Windows does not properly recognize the serial port. Windows NT also uses a serial-port driver which severely restricts program downloading/debugging over the port. Many customers write their own NT serial-port driver to enable faster access.

We suggest that you take the simplest approach to program development possible. Very few customers have problems developing from an MS-DOS/Windows 3.1/95/98 based machine using a 16-bit compiler, while significantly more have difficulties using Windows NT 4.0 with a 32-bit compiler. Before proceeding, you might wish to consider finding a simpler machine for code download/debug, although you can still do application development in whatever platform you prefer.

2.1.b TERN Disks

For installation, you should have the disks from either the TERN Development (DV) or Evaluation (EV) kits. There are two versions of these disks available.

*After installing, you can make sure you have the correct version of either the EV/VV kits by running **loc31** from the command prompt.*

In the DOS/Win 3.1 version, there are three disks.

> Borland C/C++ 3.1 for TERN 16-bit Controllers (Disk 1 and Disk 2)
> TERN EV/DV Installation Disk

The Windows 95/98 installation consists of four disks.

The top line will display either "TERN LOC31 - EV" *for the EV kit, or* "TERN LOC31 - DV" *for the DV kit.*

After making sure you have sufficient hard-disk space (about 9 MB), you are ready to install. If you are installing the DOS/Win 3.1 version of the TERN disks, skip the following section.

2.2 Installation Process - Windows 95/98 Version

The installation procedure is designed to set up a software environment from which you can easily program TERN controllers.

You should read the ***readme.1st*** file located on the installation disk for the most recent changes to the installation disks and process.

From either Windows Explorer or the MS-DOS prompt, run ***setup.exe*** on Disk 1 of the TERN Installation Disks. From that stage on, you can follow the on-screen prompts to install.

Since Borland C++ 3.1 is DOS based, it has problems with path names that have spaces or other non-standard characters.

Be sure to read the licensing agreement carefully, it outlines the legal agreement between you and TERN, Inc.

Be sure not to choose a destination path for the TERN installation that has non-compatible naming semantics.

If you choose to install to a different directory than ***c:\tern***, you will need to be sure to modify the variable **COMPDIR** in the makefiles located in your working directory to reflect this.

The setup program should prompt you to automatically insert the ***tern\bin*** subdirectory into the path. Once you reboot after this step, your TERN installation should be complete. You should proceed to section 2.4 ”Follow-up Procedures” on page 6.

2.3 Installation Process - DOS/Win3.1 Version

The installation procedure is designed to set up a software environment from which you can easily program TERN controllers.

The install.bat utility is designed to create a TERN directory in your **c:** drive and install the appropriate files. You should be sure that you run the batch file from your **MS-DOS** prompt window, and not from the Explorer window.

*Be sure to add **c:\tern\bin** to your path. This insures you will be using the correct executables as you develop your application.*

*You can do this by adding to **c:\autoexec.bat** the line:*

PATH = c:\tern\bin;%PATH%

2.3.a Borland Installation

To install Borland for TERN 16-bit Controllers on your hard drive:

1. Insert Disk 1 of the distribution into your floppy drive.
2. At the **DOS Prompt**, type “a:install”
3. At this stage, it should display the following.

FIGURE 2.1 Installation Screen

Installing TERN Files on drive c:
-=
Use TERN's Low-Cost 16-bit controllers for your next project
Easy to program in Borland C/C++
+ Horsepower supports your development needs now and into the future
+ All TERN controllers are designed and made in USA
+ Excellent in design, compact, reliable, complete, high quality, low cost.
******NOTE*******
If this is your first time installing TERN files, please edit your
autoexec.bat file and insert c:\tern\bin into your PATH.
Or add a line to your autoexec.bat:
SET PATH=%%PATH%%;c:\tern\bin
-=

Copy TERN archive volumes...
Please insert Disk 1 into drive a: and press any key to continue.

4. Hit the space bar to continue, at which point it displays

Please insert Disk 2 into drive a: and press any key to continue.

5. After you have placed disk two of the Borland C/C++ distribution in to drive A, press the space bar again:

Reconstructing archive...
Extracting archive...
Finishing up...
Installation Complete!
C:>

After completing the installation of the Borland disks, you should find on your **c:** drive a set of directories containing the binaries, header files, and libraries for Borland C/C++ 3.1.

Your **TERN** directory should contain **bin**, **include**, and **lib** sub-directories.

The include directory should include a set of system header files.

The lib directory will include a set of standard libraries.

The bin directory will include the executables provided with Borland C/C++ 3.1, including **bcc.exe** (the C/C++ compiler), **bc.exe** (the IDE), and **tlink.exe** (Turbo Link).

2.3.b Evaluation/Development Kit Installation

After installing the Borland disks, you can install the EV/DV Kit by following the same procedure as before.

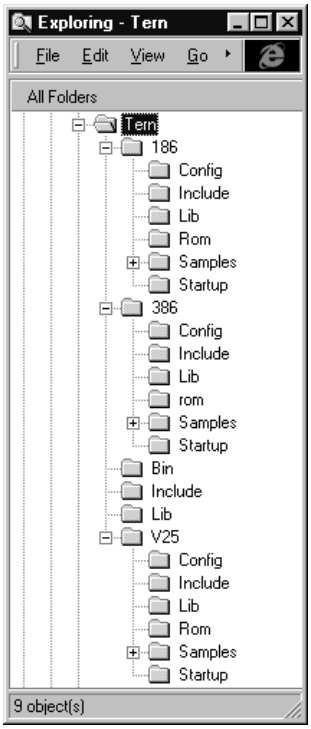
Insert the disk into drive A, and type “a:install” from the **MS-DOS** Prompt. This process will create additional working directories for all TERN controllers according to processor core type. It will also install TERN-specific header files into the include directory, as well as library files into the lib directory.

If you have an AMD188ES-based controller, (i.e. A-Engine, TD40, A104, etc.) your working directory is based in **c:\tern\186.**

If you have a V25-based controller, (C-Engine, V25-Engine, TD, V104, etc.) your working directory is based in **c:\tern\v25.**

If you have a 386-based controller, (i386Engine) your working directory is based in **c:\tern\386.**

2.4 Follow-up Procedures



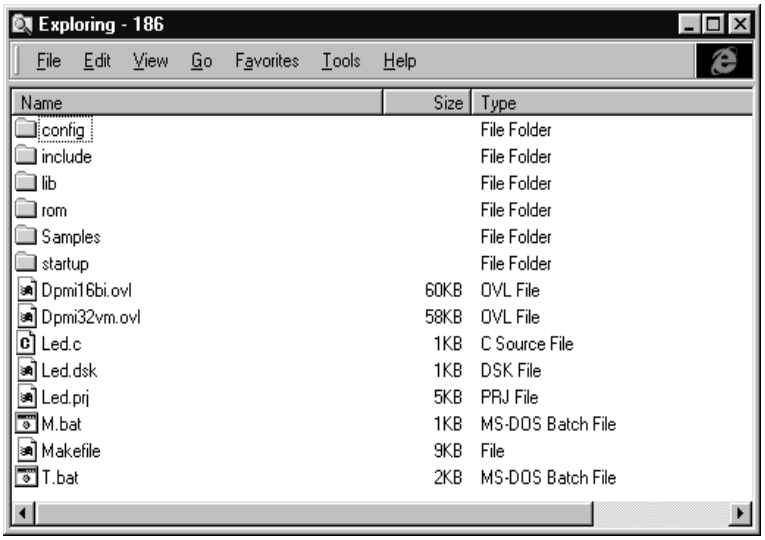
Before proceeding and trying the tutorial contained in this manual, or the sample files in the TERN directory, there are a few simple things you should do to make sure your environment is complete.

2.4.a Checking your installation

Before going any further, you should check that the necessary files have been installed. You can always check ***readme.1st*** for updated information regarding changes in the directory structure.

Borland C++ 3.1 has been installed in your root installation directory. You should find directories named ***bin***, ***lib,*** and ***include*** that contain the binary, library, and standard header files for the Borland C++ 3.1 installation. Without these files, your compilations will fail. If you are interested in using other compilers/development environments for your work, please see section **3.5.**

TERN controllers currently use three lines of micro-processors: the **NEC V25** (*V25)*, **AMD188ES** *(186)*, and **Intel 386EX** (*386)*. There are a number of controllers that are driven by each line of processor controllers, and you should check your technical manual to see exactly which product group you fall under.



Each processor group shares many libraries as well as sample files. Each has been installed in a directory off of the root installation directory, named after its moniker. This means you should find sub-directories ***V25***, ***186***, and ***386*** in your main TERN install directory. These sub-directories are the main working directories for your development.

Within these working directories, you should find the **Makefile**, as well as ***led.c***, which are used in the tutorial.

Each processor group shares a common directory structure.

For details regarding which header files and library files you will be using in your application, you should refer to the software section of the technical manual for your controller.

TABLE 2.1 Working Directory Structure

Name	Contents
lib	Library files for your line of controller.
include	Header files.
samples	Sample files for each controller. Sample files for each controller organized by name.
config	Configuration files for locating your executable in memory space. If you must modify this, details can be found in chapter **Four**.
rom	This directory contains default ***.HEX*** and ***.BIN*** files provided by TERN for other versions of the DEBUG ROM.
startup	Standard start-up code that is linked to your executable.

2.4.b Cleaning up your environment

First, make sure once again that **c:\tern\bin** is in your path.

Next, you will need to modify Makefile configuration values for your current hardware/software environment in your specific working directory. Valid choices for options are always listed in the Makefile itself. If there is a conflict between what is described in the manual and the Makefile, the Makefile is most likely more recent and correct.

*The lower half of the **Makefile** should only be modified if you need to make advanced changes to the compiling/linking process.*

*Most configuration changes should be made using variables near the top of the **Makefile** in the working directory you are currently working with.*

TABLE 2.2 Common Makefile option values

Variable Name	Default Value	Description
SRAM	1	Size of SRAM installed on board.
EPROM	0	Size of ROM on board (32K for DEBUG).
DEBUG	1	Choose to either compile/locate/ link for debugging/downloading purposes (1), or for ROM burning (2).
COMPDIR	c:\tern	Directory to which TERN software is installed.
CURRDIR	$(COMPDIR)\386	Current working directory.
BOARD	N/A	The string ID for the controller that you are currently using.

A more detailed value-by-value description of the Makefile options can be found in a later section in this manual. For now, you just need to make sure that the **BOARD** value corresponds to the controller you have purchased, and the **COMPDIR** value is set to the directory to which you have installed the TERN development kits.

You can also confirm that you have installed the correct version of either the DV or EV Kits by running ***tern/bin/loc31.exe*** with no arguments from the DOS prompt. The first line of the result will show the version of ***loc31.exe*** you have installed.

2.5 Hardware Installation

ROM is non-volatile memory in which permanent applications can be burned.

For details regarding default hardware configuration, you should refer to the technical manual for the specific controller you have purchased.

This section is meant to provide a general overview of the process.

The DEBUG version of the ROM is probably one of the few non-surface mount components on your controller, and should have a distinctive TERN sticker on top.

You should have received, as part of the EV/DV kit, a DEBUG ROM that has already been placed securely within the ROM socket. The DEBUG ROM contains the TERN remote debugging kernel, and is used for Step 1 and Step 2 of the application development.

There should also be a serial cable attached to your controller ready for connection to the PC. One end of the cable is a 5x2 IDC connector, which is connected to your controller's serial port *(SER0 in the case of most TERN controllers)*. The other end is a DB9 connector that should be connected to one of the available and properly configured serial ports on the PC.

TERN controllers require a 9-12V center-negative power supply for the voltage regulator. The wall transformer offered with the EV/DV Kit will only support 110V 60 Hz operation.

The wall transformer should be plugged into the power jack on the controller. After this simple step, the LED on the controller should flash twice and then remain on, indicating that the controller has been initialized by the DEBUG ROM and is prepared to download/debug a user application.

International customers should acquire their own wall transformers that have the same specification.

2.6 Installation Troubleshooting

Q. *My installation does not look complete, or I am missing some of the necessary sub-directories.*

First, make sure that you had sufficient disk-space for the entire install. The Borland executable and library files are quite large, and might not have been extracted correctly.

Depending on other TERN software products you might have installed, the overall size of the distribution is almost 9 MB.

Q. *An error was reported during installation, or the disks seem faulty.*

Contact TERN tech support at (530) 758-0180. We will do our best to diagnose the problem and rectify the situation as possible.

Q. *The hardware did not respond as described during hardware installation.*

Ideally, your hardware is in the same configuration as when it was shipped from TERN. This means that the DEBUG ROM socket is installed on the controller, and the PC-V25 serial cable is connected to the correct header (*SER0*) with the correct orientation.

You should refer to the controller technical manual for details regarding hardware installation.

CHAPTER 3

Tutorial

This chapter walks you through the process of downloading and debugging your first TERN controller-based application.

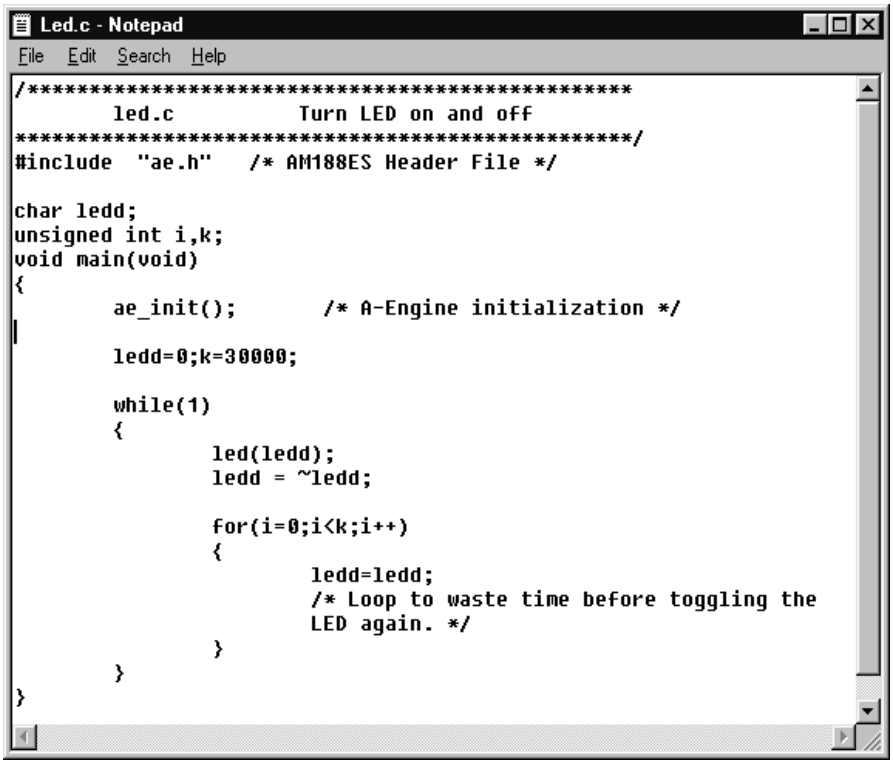
After completing this chapter, you will be familiar with the process of using TERN batch files to compile and download your application. You should also become accustomed using Turbo Remote Debugger. The development process outlined in this chapter will parallel your own, and you should find it to be of great assistance no matter which step of development you are currently in.

Once again, this tutorial is purposely hardware-general. For details regarding hardware configuration, and locations of headers and jumpers, and other technical details, refer to the technical manual for your specific TERN controller.

3.1 *The Sample Program - LED.C*

You should change your directory to the appropriate working directory for your controller (***\186*** for AMD AM188ES-based controllers, ***\v25*** for NEC V25-based controllers, ***\386*** for the Intel 386EX-based controllers).

There, we provide a simple program utilizing TERN controllers to demonstrate the development process. ***led.c*** makes the LED blink on and off continously.



The file shown above is the **186** version. The other versions are identical except for different include file and initialization function names. After initializing the controller, this program just sits in an infinite while loop repeatedly toggling the LED by calling **led**(). The for loop is provided to burn processor cycles and slow down the rate at which the LED is toggled.

For a detailed description of the function interface and other software details, you should refer to the software section of the technical manual for your controller.

3.2 *Step One - Download and Debug*

3.2.a PC Environment

Type **path** at the MS-DOS prompt to verify that the directory **c:\tern\bin** is included. If not, you will need to modify ***autoexec.bat*** appropriately. Please see the **Installation** chapter for more details.

To test the environment, go to directory ***tern\186>*** and type ***m led***. This should pass without error as it compiles your target source code. This batch file is described in more detail in the next chapter. It produces a ***test.exe*** file in your working directory. This is the executable that will be downloaded to the remote controller via serial link using Turbo Remote Debugger.

If you find an error during this or any of the following process, see the following Troubleshooting sections of this chapter for details.

3.2.b Step One

If you are using a 20 MHz version of an AM188ES controller that is also available at 40 MHz, you must modify your baud rate setting to 57,600 baud. The default setting of 115,200 baud is for the 40 MHz version.

All other TERN controllers have a default setting of 115,200 baud.

For the first step of the development process, you will be using the batch file ***t.bat***. This batch file is discussed in more detail in the next chapter.

The default setup for Turbo Debugger is to communicate with your controller using COM1 of the PC at 115,200 baud. These settings are set within ***t.bat***, and can also be specified using command-line arguments.

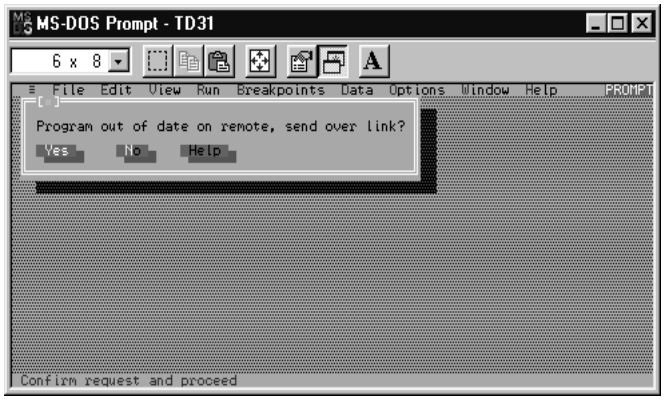
Connect and power-on your controller as described in the installation chapter of the technical manual for your controller.

Begin by entering "**t.led**" at the DOS prompt.

```
c:\tern\186> t led ↵
```

You will see a series of compilation messages as the program is compiled and linked with TERN libraries, and then is located to the appropriate address in memory. If all goes well, compilation will succeed and Turbo Remote Debugger will be called to try to communicate with, and debug, your controller application.

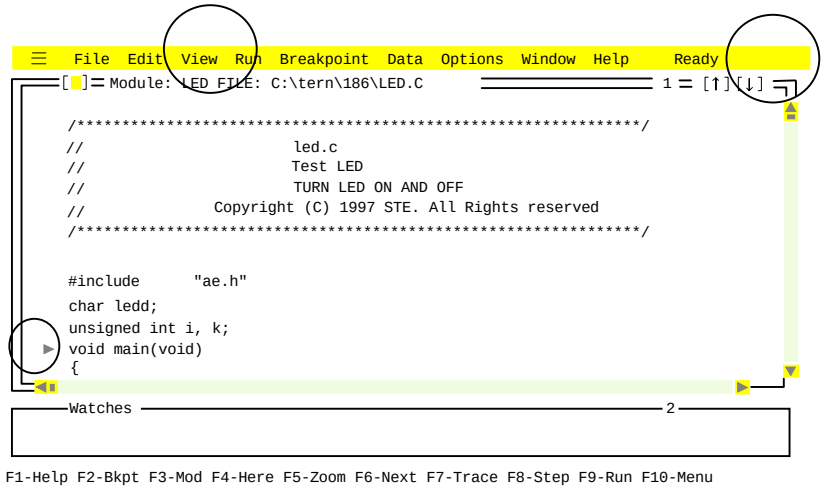
The TD31 DEBUG window should come up with the prompt shown below.



If it does not, do not give up. This is one of the most challenging aspects of configuring your installation. The Troubleshooting section on page 8 has many hints on the steps you can follow to correct your problem.

Enter '**Y**' (or just press Enter) to continue. A ***Wait*** prompt will display at the upper right corner of your screen while your program is downloaded to the controller. A 64K file may take approximately 6 seconds to download at the default setting of 115,200 baud.

If downloading proceeds without any errors, the TD31 window will show the ***led.c*** source file and the prompt ***Ready*** at the right upper corner of the screen. The following picture shows the initial TD31 window, and highlights for your benefit a few areas of special interest.



Do not confuse the TD31 Debugger window with the editor. You cannot edit your source code while debugging it.

There are two windows in the TD31 screen as you see it: #1 is the source code window, #2 is the watch window. You can use ***Alt-1*** and ***Alt-2*** to activate the appropriate window.

The small triangle circled in the bottom left corner of the source code window shows your current position in the code. This indicates the current line of code that is about to be executed.

You can also select the menu option ***View->CPU***. This brings up a new window showing more detailed information regarding the current CPU status. This window shows you the value of the registers, and the assembly instructions that are about to be executed. You can step through these the same way you would step through your C-level code. Also shown is the original C language code from which these instructions were compiled. This allows you to take a finer-grained approach toward debugging your application.

The bottom menu indicates some basic debugging commands. To begin, you can use ***F8*** to single-step to the next line of code. The triangle and line number will reflect the change with each step. You can also use ***F7*** to try to trace into a function call as you are about to execute it. If the source code for this function isn't available (for example, if it is a library function provided by TERN), the debugger will just single-step over the function call.

Use ***Alt-2*** to select the watch window. You can watch any of the declared variables in this window. For starters, try typing "**i**" after you have activated the watch window. Pressing ***Enter*** will include **i** in your watch list. Add "**k**" and "**ledd**" in your watch window the same way; as you step through the code further, you will see these values change.

*The watch window **is** case sensitive, so be careful while you select the variable you wish to watch.*

F2 will set a breakpoint on the current line of the cursor. A breakpoint line is highlighted in red. Pressing ***F2*** again will toggle the breakpoint, either setting or removing it.

These watch variables are not dynamically updated. If you need to view the current value of any variable, you will have single-step through the code, or set breakpoints.

F9 runs the program ***led.c*** uninterrupted. The LED on your controller should flash continuously. While the program is executing, the prompt ***Ready*** is replaced by ***Running***. Once the code reaches a breakpoint, it stops running and the cursor is pointed at the location of the breakpoint.

*Also, there are variables that might be out-of-scope as you step into other functions. These variables will be displayed as **????** in the watch window.*

At any point, you can press ***Ctrl-Break*** to halt execution of the program. You should do this before disconnecting or powering-off your controller. Otherwise, the Debugger might lock-up while it waits for a response from the controller.

Once you are finished with the debugging process, you can quit TD31 by hitting ***Alt-x***. The batch file now brings up the IDE window (***bcc.exe***) that is provided as part of Borland C++ 3.1 for you to modify your C source code. You can also exit this IDE window using ***Alt-x*** again. This will bring you back to the MS-DOS Prompt.

*You can modify **t.bat** if you want to change the applications that are loaded during this debugging process. Although you cannot use a different debugger, you can choose to use a different editor for your project.*

There is one thing you need to be aware of during this first step. There are two windows that look very similar (both have a blue background) during this operation: the Turbo Debugger window and the Borland C++ 3.1 IDE source editor window.

The TD31 debugger window has the following pull-down menu items at the top of the screen:

Turbo Debugger is used for remote debugging. It downloads the located code to the embedded controller that has been connected to your PC via serial port, and controls the debug operation including single step, breakpoints, and run. Remember that your application is actually running on the controller, and not the PC.

Many customers prefer to work with a Windows 95/98-based editor. For ideas on how to integrate this with the TERN development kits in other ways, see the section 3.5 "Other programs"

The other application window is the BC31 IDE source editor window. It has this as the available pull-down menu items at the top of the screen:

The obvious differences between these two windows are the "Search" menu for the BC IDE and the "View" menu for the TD31.

If you see the "View" option in the top row menu 3rd position, you are in Turbo Remote Debugger (TD31), and your code is running and debugging on the remote controller, NOT on the PC. If you see the "Search" option in the top row menu third position, you are currently using the Borland C++ 3.1 IDE.

*Do not **run** your program in the IDE environment. Attempting to run an embedded application that has already been located on your PC will cause it to crash.*

3.2.c Modifying *led.c*

Now that you are more familiar with the debugging/development environment, you can modify and compile your first custom application, using ***led.c***.

Go ahead and change the line **k=30000;** to **k=3000;**. This variable determines how long the program delays before toggling the LED on and off. Making **k** smaller shortens this delay.

You can use ***F9*** from within the **BC3.1** IDE to recompile ***led.c*** and report any initial syntax errors. This step does not actually link your generated object file with TERN libraries, or re-locate the executable. You will still need to run either ***m.bat*** or ***t.bat*** from the DOS prompt again to re-compile and generate the executable for downloading/debugging.

Use ***F2*** to save the changes that you have made, and ***alt-x*** out of the editor.

You can now repeat the Step One procedure by again running:

```
c:\tern\186> t led ↵
```

From within the debugger, use ***F9*** to run the new updated ***led.c*** again. The LED should respond by flashing much faster.

3.2.d Troubleshooting Step One

Q. ***I can't successfully run m.bat. It gives me error messages and does not compile the application correctly.***

The first thing to check is if your installation is complete. Please check again the steps described in the Installation chapter.

Q. ***I have no problems entering Turbo Debugger, but when I try to run, the machine freezes up.***

The most common cause of this problem is the Windows NT serial port drivers. As a test of the actual controller, after Turbo Debugger has successfully downloaded the program and before you start executing code, press ***Alt-X*** to exit out of TD31. Your code already rests in SRAM. Proceed to Step Two and see if the code can execute correctly. If it does, it is quite possible the source of your problem lies somewhere in the software/hardware configuration of your particular PC.

Q. ***Modifying the file does not give me any different behavior, the LED is still blinking at the same rate.***

First, make sure the source file has been modified and saved as you expected. Next, try running ***m.bat*** on your target source. The file ***test.exe*** should be generated. Make sure that this file has a current modification date/time. Occasionally, the compilation will fail without the user noticing, and an older ***test.exe*** downloaded to the controller.

Q. ***Running t.bat actually brings up Turbo Debugger with a different source code than the one specified in the command line; or, as I step through the code, it looks like what is being executed is not the source file shown.***

You should check the modification date of the file ***test.exe*** created in your working directory. Since this is the file that is downloaded into your controller, it should have been updated when you recompiled with either ***m.bat*** or ***t.bat***. If the modification date is earlier than that, be sure that the environment variable **DEBUG** is set to **1**. Otherwise, the ***makefile*** will generate a ***.BIN*** or ***.HEX*** file instead of creating an executable for downloading and debugging. The ***test.exe*** from the previous compilation will be downloaded instead.

Q. ***t.bat displays "Waiting for handshake from remote driver (Ctrl-Break to quit)". Turbo Debugger 3.1 never shows its window.***

This is one of the most difficult parts of your system configuration. There are several things that might go wrong in this stage, all with similar symptoms.

If you have waited several seconds and Turbo Debugger never fully loads, press **Ctrl-Break** to quit. You should quit Borland C++ as well and try to run ***t.bat*** from the beginning again.

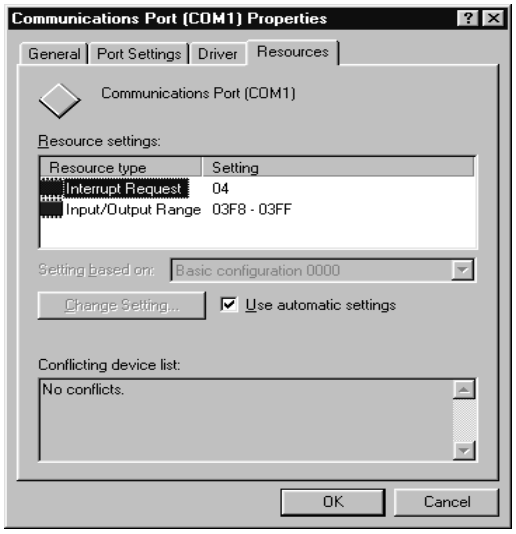
The DB25 end of the serial cable connector should be attached to one of the serial COM ports on your PC. Double-check the connection to make sure it is not loose. Make sure that your batch file has been set up correctly to use the appropriate COM port.

Make sure that the controller is powered-on and the serial cable has been connected to the correct serial port. The hardware, including serial cable, is thoroughly tested before shipment, and the probability of faulty hardware at this phase is low. You should also double-check that the serial cable has been attached with the correct orientation. Refer to the technical manual for your specific controller for details regarding this process.

Check that when the controller is first powered-on, the controller's LED flashes twice and stays lit. If this does not occur, it indicates a few possible problems. Make sure that the DEBUG ROM is installed in the ROM socket correctly. Next, make sure that the Step Two jumper, as described in your hardware manual, is not in place at power-on.

If an external examination does not solve your problem, you will have to check your operating-system and software configurations.

Windows 95/98. Turbo Debugger runs in a DOS shell on your 32-bit Windows operating system. If another application holds exclusive access to the COM port, there is often no indication while you use your DOS application. The first thing you will want to check is that there is really no other device currently using the COM port you believe you are currently using. You can check the **Devices** window from within Control Panel for details about attached devices. Make sure that no other devices, especially modems and serial-mouses, share these COM ports. Interrupt conflicts on the PC might also be a source of problems. You should probably go through all of the hardware devices and make sure that each is configured for "Automatic settings" under the Resources tab, as is shown in the figure.



Make sure that no application, such as HyperTerminal, currently has the "port" open even if no device is attached. This is also sufficient to deny access to your DOS based program.

Windows NT. Windows NT can have the same com-port usage conflicts as described for Windows 95/98. This will be the first thing you will want to check. Also, as was mentioned in the Introduction chapter, Windows NT restricts serial port access in a manner that makes it difficult to communicate at high baud rates. If you are using NT, consider moving your system to a Windows 95/98/DOS machine for download and debug.

*You can usually enter BIOS Configuration by hitting **DEL** upon boot-up before the Operating Systems is loaded.*

If all of the above steps fail, you should also make sure the COM ports are enabled in the BIOS settings for the machine. Reboot your computer, and enter the BIOS Configuration menu. Some machines have certain serial ports disabled in BIOS.

If you still have difficulty completing your serial communication with the embedded controller, it is very possible that the UART on your PC is not capable of handling the high speed transfer rate of 115,200 baud that is default on most TERN controllers. This is especially true if you have an older machine.

Please try transferring your system to a different machine to confirm if this is the source of the problem. If it is, you can burn your own DEBUG ROM that operates

at a slower baud rate. These can be found in the **\ROM** sub-directory in your working directory. If you do not have a device programmer handy, or if there isn't an appropriate ROM for your controller, you can contact TERN to request a special version of the DEBUG ROM to be sent to you.

If all efforts fail, contact TERN technical support staff at **(530) 758-0180** or ***tech@tern.com***.

3.3 *Step Two - Field Test*

Within Step Two of the development process, you will download the application tat you developed and initially debugged in Step One into SRAM for a field-test. In many applications, it is not practical to fully test all controller capabilities in the development environment. By using the battery-backed SRAM on TERN controllers, you can actually run your executable for up to five years.

After Step One, your application will be downloaded and located into the SRAM. The battery (if installed) on your controller will keep the executable in memory even after you disconnect the external power supply.

Power off the system, and place the Step Two jumper (as described in the technical manual for your controller) onto the controller.

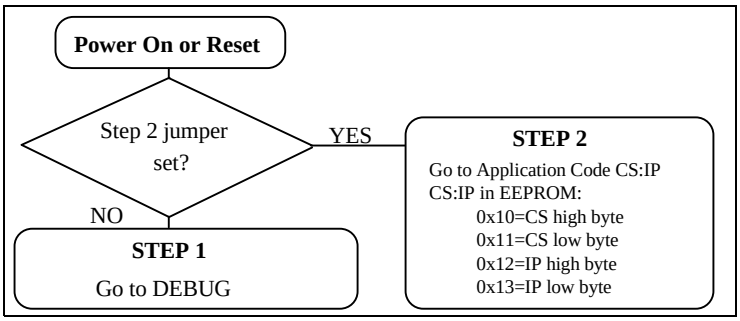
Whenever you power-on or reset the board with the Step Two jumper in place, the DEBUG ROM will start executing your code loaded in SRAM.

At this stage, you are actually executing your code from SRAM and your program is ready to be tested in the field.

3.3.a Step Two Jump Routine

Upon execution, the DEBUG ROM will check the Step Two jumper. If the jumper is not in place, the DEBUG ROM will continue execution from within the DEBUG ROM. This code waits for Turbo Remote Debugger to initiate download of a new program for debugging (Step One).

If it is enabled, the DEBUG ROM checks the 512-byte serial EEPROM that is on all TERN core controllers for an address within SRAM from which to begin execution.



A simple functional flow chart of the DEBUG ROM

The Step Two Jump Address is set as part of factory configuration, and you ordinarily do not need to modify it. If, however, you choose to locate your program in SRAM to a different address, or if you accidentally write over the EEPROM location that holds this data, you might need to reset the Jump Address.

3.3.b *step2.c*

The program ***step2.c*** is located in the engine controller sub-directory of the ***samples***\ sub-directory of your current working directory.

- For the ***186***, it is found in the ***AE*** sub-directory,
- the ***V25*** version is found in the ***VE*** sub-directory,
- and for ***386*** controllers it is found in the ***IE*** sub-directory.

You can modify ***step2.c*** to reflect the address at which your software program is located. Running this program from within the debugger will write the updated address values to the EEPROM on your controller. If you are not sure where your program is located, run ***m.bat*** on your target. A ***.loc*** file named after your target will be created. This file shows the various segment locations, include the **CODE** segment**.** Details regarding this batch file is found in the next chapter.

3.3.c Troubleshooting Step Two

Q. ***The Step Two jumper is in place, but when the board is reset, my application isn't running.***

As a first step, always try your system out with ***led.c***. Make sure it downloads correctly using Turbo Debugger into the SRAM. While you are in the debugger, run the application several times to make sure it appears to work correctly.

Make sure that the DEBUG ROM is the ROM that is currently placed into the ROM socket. Make sure that a battery is installed on your board. This is the only item that keeps your program in static RAM after you remove the external power supply. Use a voltmeter to confirm that the battery charge is still around 3V. Anything lower than 2.6V will probably not be acceptable.

Next, double check that the Step Two jump address is correct. If you are not sure, run ***step2.c*** again to write the correct value into EEPROM.

If all fails, contact TERN technical support at (530) 758-0180, or by email at tech@tern.com.

3.4 *Step Three: Production (DV Kit Only)*

Step Three is only available if you are using the Development Kit, as opposed to the Evaluation (EV) kit.

If your program is behaving as you expect, you can generate a **.HEX** or **.BIN** file that can be burned into a ROM device using a device programmer. If you would like to use the ACTF Kit to program a Flash device over the serial port, refer to the ACTF technical manual for more details regarding this phase of the development process.

For ***led.c***, you can go ahead and use a 32K ROM (27C256-70). For other details regarding supported ROMs, (such as timing, hardware configuration changes, etc.) refer to the technical manual for your controller.

You will have to edit the Makefile to generate **.HEX** and **.BIN** files instead of a downloadable executable.

*The default factory setting for >128K SRAM (**SRAM = 1**) is address 0x0800:0000.*

*For 32K SRAM (**SRAM = 0**) setting is address 0x0400:0000.*

Edit the Makefile flags to reflect the following:

```
DEBUG = 0
EPROM = 0
```

Save the changes you have made to the Makefile and exit your editor.

Run ***m.bat*** on your target once again.

```
c:\tern\186> m led ↵
```

This will compile ***led.c*** and locate it for ROM use. You should find that ***led.bin*** has been created. This can be used by most device programmers to burn your own ROM.

Power-off the board, and you can replace the DEBUG ROM with your newly burned ROM. Upon power-up, you should find that the code in the ROM now executes, and the LED flashes continuously.

3.5 *Other programs*

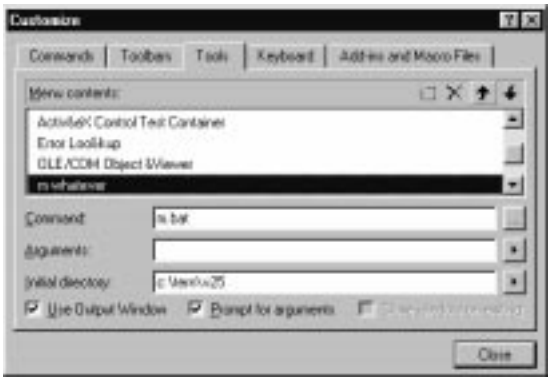
3.5.a Using TERN Development Kits with other software

Many application developers have become familiar with a particular IDE. Having a graphical user interface that users are familiar with increases efficiency dramatically. Although Borland C++ is more than sufficient for any development application, there are certain benefits to using an environment development that is Windows-based.

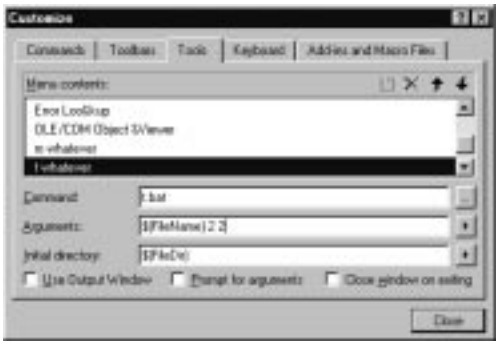
Fortunately, since TERN provides integrated batch files that actually manage the compilation process, application developers can usually integrate the TERN software development kits into their environment without much difficulty. This section shows how you would add TERN batch files as Tool Utilities to *Microsoft Visual Studio 5.0*. Most other advanced IDE should also have similar functionalities, and you should be able to parallel the steps shown below.

Once you have started *Microsoft Visual Studio 5.0*, go into the **Tools** menu and select **Customize.** Select the **Tool** tab; this allows you to create quick menu/toolbar access to commands that are executed often. Create a new tool by clicking on the ***New*** button near the top-right of the dialog. Start by creating a tool for using ***m.bat***, as shown.

The initial directory should be your working directory. Since the only argument for ***m.bat*** is the target name, you can click on "Prompt for arguments". Whenever you choose to use this tool, a dialog-window into which you can type the name of your target will pop up.



By choosing "Use Output Window", the compilation results will actually show up in the bottom Microsoft Visual Studio **Output** window, similar to what would happen if you were compiling with a built-in compiler. This also allows you to scroll-back and see detailed information about your errors without having to pause the screen in DOS.



The same steps can be applied to ***t.bat***. Turbo Debugger still needs to be run in a DOS window, and you cannot choose to "Use Output Window". Since this does not give you the opportunity to examine your errors in detail, you should check your compilation with ***m.bat*** before using ***t.bat***. The tool dialog should be setup as shown.

The arguments are set up differently to show how else you might use these tools for maximum efficiency. By selecting **$(FileDir)** as your initial directory, it means that compilation will occur in the directory where the current file you have open is located. Remember that this should usually be the working directory, since ***makefile*** is only located there.

$(FileName) can also be used as a shortcut for the target name. This works perfectly for the TERN development environment, since the batch files expect as an argument the target name, which is the name of the file without extension You can also preset your COM port/baud rate settings this way, as well, by incorporating the arguments into the command line arguments.

3.5.b Sample files

The various steps of development are discussed in more detail in the following chapters. For now, you can also try experimenting with the numerous other sample files available for your controller. These should all be included in the ***samples***\ sub-directory of your working directory.

Keep in mind that although some peripheral boards have sample directories for board-specific sample files, all boards still share the core functionalities demonstrated by the sample files in the various engine directories (***AE, IE, VE***).

If at any time you run into difficulties with the development process, feel free to contact TERN technical support for assistance.

CHAPTER 4

Development and Debugging

This chapter discusses in more detail the development and debugging process. It introduces you to the tools available at hand, and describes many specific technical features present in the TERN software development kits.

Once you begin your full application development, this chapter will provide the depth of knowledge it will take to carry your project to completion.

4.1 Development Batchfiles

For a detailed description of the function interface and other software details, you should refer to the software section of the technical manual for your controller

Two batch files are provided for your development. In general, they can be used to entirely automate your compilation process.

4.1.a m.bat

Command: m.bat
Arguments: TargetName

The batch file ***m.bat*** is used to compile your target source files, link, and then re-locate the output code with ***loc31.exe***. It uses the **makefile** to compile all of the necessary source files, and then links in the necessary TERN libraries depending on the TERN controller specified.

The argument **TargetName** is just the filename of the target source file you wish compiled, without extension. The default extension for this file will be **.c**, but can be modified by changing the ***makefile*** and batch files.

Example: m led

m should be used while you are in the initial development phase of your application. It compiles your code, returning any errors necessary without connecting to the remote target.

The PC screen often scrolls too quickly while compiling to see all of the error messages effectively. If you hit the ***Pause*** key on the key-board during scroll, the operation should stop while you read error messages. You can hit the **spacebar** to continue the makefile operation.

This batch file actually works by first compiling your source code with the startup files provided by TERN. Libraries for your controller are then linked in together to create the executable. ***LOC31*** is used to actually generate the final re-located executable that runs on the controller. This step is necessary, as the memory mapping for the PC is dramatically different from what occurs on the controller.

Running ***m*** generates a number of files. These include the ***.map*** file that shows the segments as generated by Borland C++ 3.1, while the ***.loc*** file shows the segments as they are located onto memory space on the controller. The ***.loc*** file is of special interest during the development process.

The <***target***>***.loc*** file can be used to check the location and length within memory space of various segments, as they are generated by ***LOC31*** for execution on the controller. The segments of interest are class **CODE** and **DATA.** These segments can not be larger than 64 KB in length if you are using the small model libraries.

The locations where these segments are located are specified by the ***makefile*** using a configuration file. These are all found within the ***config***\ sub-directory of your working directory, and differ depending on if you are generating code for debugging purposes (locating the program in SRAM) or for ROM burning (locating the code for ROM, data for SRAM). For example, the file ***TEST128.td*** is used for downloading an application into SRAM of 128 KB or larger in debug mode.

*The TERN ACTF Kit can be used to directly load your generated **.HEX** files into Flash chips placed into the ROM socket.*

For details, contact TERN.

If you are about to generate a ***.HEX*** file for use in a device programmer or the TERN ACTF Kit, you will also wish to use ***m.bat***. The end result will be a correctly located ***.HEX*** (or ***.BIN*** if properly specified in the configuration file) file with which you can program an EPROM. If you examine the ***.loc*** file created for **DEBUG** and for ROM burning, you will see significant differences.

A more detailed discussion of memory mapping and how you might choose to modify it is below.

4.1.b t.bat

t.bat is used when you actually wish to download your application into SRAM on the controller remotely. It also initiates the remote debugger to debug the application.

The first phase of ***t.bat*** is identical to ***m.bat***. This means the same discussion regarding the compilation process applies. It also means that if you have been able to complete ***m.bat*** with no errors, the first phase of running ***t.bat*** should also finish with no difficulty. In addition to the single target command argument accepted by ***m.bat, t.bat*** takes two others used to specify the communication process between your computer and controller.

Command: t.bat
Arguments: TargetName { ComPort BaudRate }

t.bat compiles the target specified by TargetName. You do not need to append the suffix. If you wish to change the suffix of the target, you will need to modify ***t.bat*** appropriately, as well as the ***makefile***'s **SUFFIX** environment variable to recognize your target file.

Example: t led

The command line arguments ComPort and BaudRate specify the ports and baud rate you wish Turbo Debugger to use while communicating with your embedded controller. You must provide both arguments if you want to provide one. If these optional arguments are not included, ***t.bat*** will use the default values specified in the batch file. The initial default values are COM port = **COM1** and BaudRate = **115,200** baud.

Available values for ComPort and BaudRate are as follows:

- ComPort = 1/2/3/4 for COM1/COM2/COM3/COM4
- BaudRate = 1/2/3/4 for 19,200/38,400/57,600/115,200 baud.

Example: t led 2 2 *To specify **target** = led, with communications over COM2 at 38,400 baud.*

Although the files generated by ***t.bat*** are similar to those generated for ***m.bat***, they are deleted at the end of the batch file. The intention is that ***m.bat*** is used exclusively for compilation purposes, and ***t.bat*** should be used when you need to download and debug your code directly.

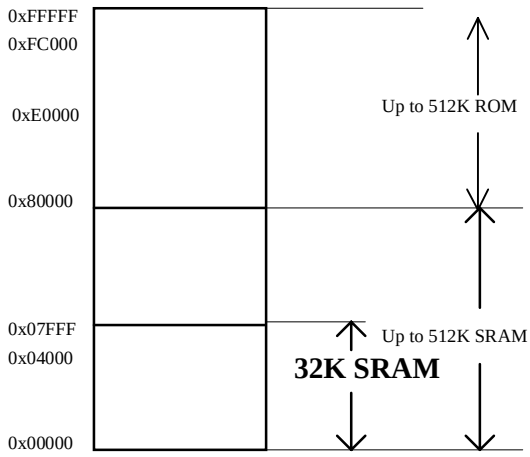
If you run into any problems during the download process while using ***t.bat***, you should refer to the troubleshooting section for Step One of the development process in the Tutorial chapter.

4.2 Memory Mapping

This section is intended to introduce the way in which memory on TERN controllers is mapped and used for code and data. Different TERN controllers may have specific differences, and you should always refer to the technical manual for any differences.

4.2.a Physical Memory Mapping

On TERN controllers, the SRAM and ROM are each mapped into a total of 512 KB of memory space. This gives a combined memory space of 1 MB, or 20 bit addresses.



The SRAM is mapped starting from address 0x00000, and goes until 0x7ffff. The ROM socket is mapped from address 0x80000 to 0xfffff. This means any access will be translated into an access into either RAM or ROM depending on the address of the access. Code placed in ROM is actually located at the top of the memory map, meaning that a 128 KB ROM will extend from 0xe0000 to 0xfffff.

Although this is the size of the actual memory map, most users will not use the entire available physical memory map. For example, using a 32 KB of SRAM means that only addresses 0x00000 to 0x07fff will be valid in the SRAM mapped region. 128 KB of SRAM extends to 0x20000.

When using the DEBUG ROM for debugging purposes, the application code and data are both downloaded into SRAM. This means the code and data segment will both be located in the lower half megabyte of memory address space. The DEBUG ROM also contains specific code used for initializing the hardware in preparation for the debugger.

If you are burning your own ROM, your code will be located in the upper half of the memory space. The data segment rests in the lower half of the memory space.

4.2.b Locate Configuration File

The configuration files used by ***loc31*** are provided for you in the ***config***\ sub-directory of your working directory. In most cases, you need only modify the one cur-

rently used for your compilation. If you are not sure which file is currently used based on your selected ***makefile*** environment variables, you can watch the last line of your ***m.bat*** compilation. It should have something similar to

\tern\bin\loc31 -c**\tern\186\config\TEST128.td** -DHASFARDATA led

TEST128.td is the configuration file used for this particular compilation. The configuration files with the ***.td*** extension are used when compiling with **DEBUG**. The ***.rm*** extension configuration files are used when compiling for ROM use. They are different in several respects.

One part that is common to both is the definition of the various maps. These are used to set properties for each segment of memory space. This allows you to isolate errors more easily since ***loc31*** will not allow you to place data segments within a reserved map, for example.

In the ***TEST128.td*** file, there are four maps set up representing four different logical areas in the memory map. This file is intended for a 128 KB SRAM.

```
map    0x00000 to 0x00fff as reserved
map    0x01000 to 0x07fff as rdwr
map    0x08000 to 0x1ffff as rdonly
map    0x20000 to 0xfffff as reserved
```

The first map segment, 0x00000 to 0x00fff is set up as reserved for certain core debugging functionalities. This includes the interrupt vector as well as necessary SRAM space for the debugger. 0x01000 to 0x07fff acts as the simulate SRAM for this application, while 0x08000 to 0x1ffff acts as the read-only ROM. In fact, this last mapped region is also available for read and write access, but it is set as read-only to parallel the situation if your code actually rests in read-only ROM.

If you are using a 32 KB SRAM, you may find that the data segment allocated for your code is insufficient, but that there is excess memory space allocated for the code segment that is unused (you might check this fact by looking at the length of each segment in the ***.loc*** file). You can modify the size of each segment, as well as where each is located.

You can re-map the segments above as needed for your new memory mapping structure. Be sure to also modify the two variables for the classes ***DATA*** and ***CODE***. By default, ***DATA*** starts at 0x0100, while ***CODE*** begins at 0x0400. For example, you might choose to set ***CODE*** = 0x0600, allowing it to range from

0x0600-0x07ff, a total of 8 KB worth of code. This also allows you to give a total of 24 KB to your data segment.

If you run your program in Step Two, you will want to make sure to run ***step2.c***, as described in the Tutorial chapter, to jump to the correct address to begin execution. The value set into EEPROM should be the same address as the location of your ***CODE***.

TEST128.rm, used for locating your application in memory space for burning into ROM, has a simpler map setup.

```
map    0x00000 to 0x07fff as rdwr
map    0x08000 to 0xdffff as reserved
map    0xe0000 to 0xfffff as rdonly
```

The RAM size is setup by default to be only 32 KB. If you are using a larger SRAM with your own ROM, you will want to modify the configuration if you expect to address all 64 KB of data segment.

The segment from 0xe0000 to 0xfffff is the address space for the 128 KB EPROM. Your application will be located here. Once again, you can specify the actual location of each segment using **CODE** and **DATA**.

You can also specify whether you wish to generate a ***.BIN*** or ***.HEX*** file. Different device programmers will require different generated files. The TERN ACTF Kit, for example, requires you to generate ***.HEX*** files for downloading into Flash.

Near the top of the ***.rm*** configuration files, you will find a line that begins with *hexfile***.** This directive tells ***loc31*** where to place the offset for the executable, the size of the ROM, as well as the type of generated file.

If you wish to generate a ***.BIN*** file, the general syntax will be:

```
hexfile binary offset=0xe0000 size=128
```

A ***.HEX*** file syntax is similar, and the general syntax is:

```
hexfile Intel86 offset=0xe0000 size=128
```

Offset specifies the location where the executable will be placed. If you need to modify this value, you will need to change it here as well as below, where **CODE** is specified. Size should be equal to the size of the overall ROM, in KB.

4.3 Makefile Options

The makefile is located in your working directory (depending on which controller you are using).

Several environment variables are available for your use. In general, you will not need to modify the core makefile settings. You should instead just modify makefile options as needed for your particular hardware and software configuration.

4.3.a USER_OBJS

You can set the value **USER_OBJS** to the source code object files you need to link into your final executable. This is the environment variable you will need to modify if you are working with multiple source files. The target filename, as specified by using ***t.bat*** or ***m.bat*** is already selected as one of the objects to be compiled and linked. If you have any others, you should add their names here as well.

For example, if you have source files named ***FileOne.c*** and ***FileTwo.c***, set the value

```
USER_OBJS = FileTwo.obj
```

You can then compile both executables and link them with the final executable by using "**m fileone**".

4.3.b BOARD

This is a string constant that defines the core controller you are using. You should be sure to use one of the default strings. In general, this should already be defined correctly and you will not need to modify it.

It is used to determine the appropriate libraries and include files that must be added to your compilation.

4.3.c EXTENSION

This is the extension of your target file. If you are compiling a **C** file, as default, this will most likely be set to ***c.*** If you are using a **C++** file, this might be set to ***.cpp*** instead.

4.3.d MEMCARD

There are two different sets of libraries, C series and D series, for the AMD Flash memory cards you are using with the MemCard. The valid choices here are ***TYPE_C_CARD*** for the C family of flash cards, and ***TYPE_D_CARD*** for the D family of flash cards.

4.3.e COMPDIR

This directory is the root directory into which TERN software is installed. By default, this value is **\tern**. You should be aware that since you are working with DOS applications, path names must not contain spaces or other special characters (i.e., *, ?, @, etc.), unlike in Windows 95/98.

Several other directories are just extensions of this one variable. These include **CURRDIR** (the current working directory), **CONFIGDIR** (the directory where configuration files are contained), **STARTUP_DIR** (directory containing various startup files), and the **INCLUDE_DIR** (the include directory for your compilation).

4.3.f EPROM/SRAM/CPU

These options should all be configured as commented in the ***makefile*** for your particular hardware configuration.

4.3.g OPTIMIZE

This flag indicates the extent to which your code should be optimized. Optimization for speed does not dramatically speed up your code. If you have time-critical applications, the best solution is to probably write assembly level code that gives you more precision control over execution.

4.4 Optimizing Your Code

There are a few hints we can provide about optimizing your code to run on hardware provided with TERN controllers.

First of all, standard optimizing techniques can be applied to reduce the complexity of code. References for this can be found in programming texts. Code that uses floating point, for example, should be avoided if performance is a concern.

Many of the chip-select lines on TERN controllers are programmed with a high wait-state value for maximum reliability. In environments with high noise values, it might be necessary to leave this wait-state value high. If you are seeking higher performance though, you will wish to turn down the default wait-state values.

Depending on the processor on which your TERN controller is based, the wait-select line for the RAM/ROM might be set to as high as five wait-states. If you turn this value down, you should find a dramatic increase in performance. If you are using peripheral devices, the default I/O wait state value is often as high as 15 wait states.