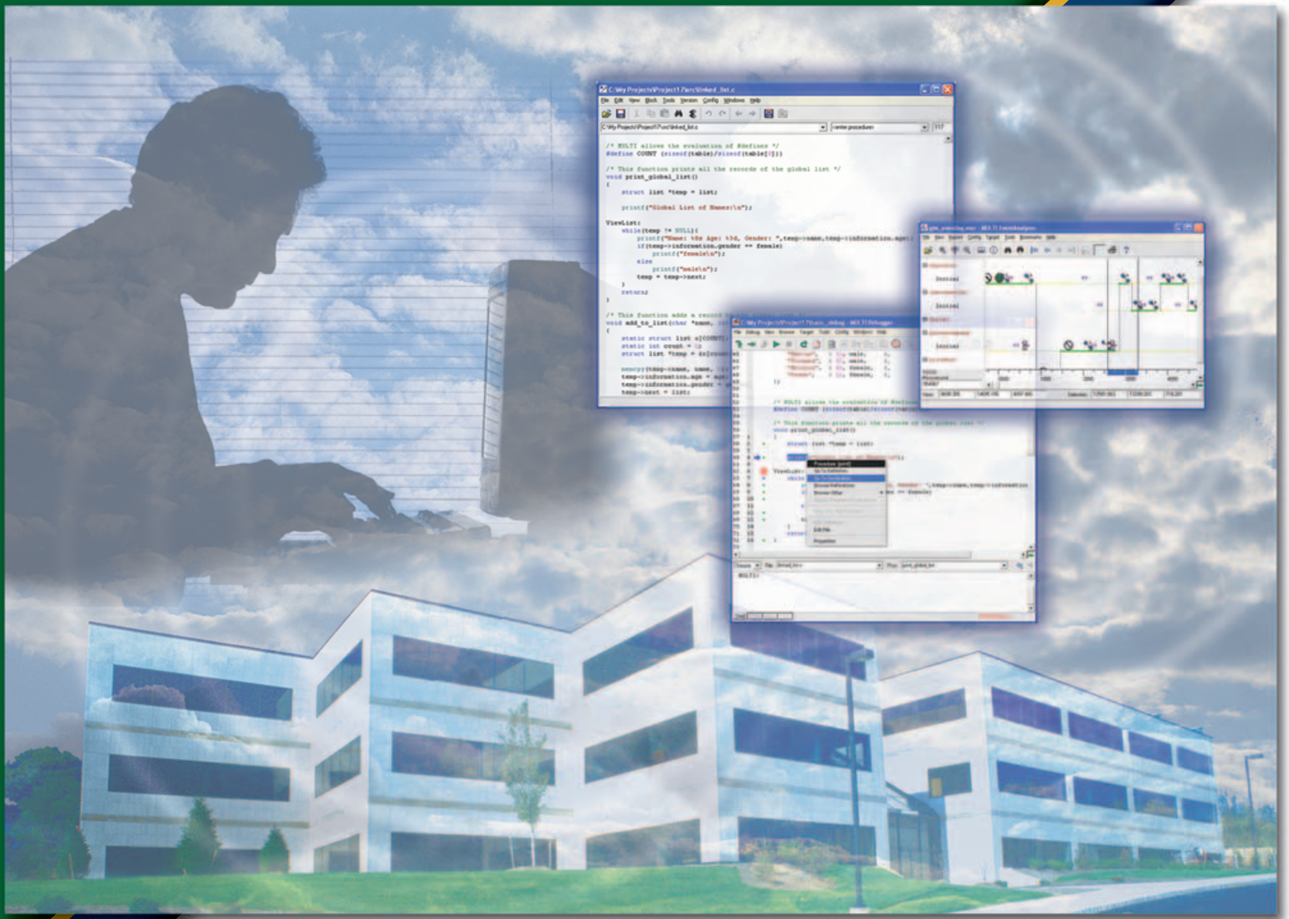
Green Hills Software



```
/* MULTI allows the evaluation of #defines */
#define COUNT (sizeof(table)/sizeof(table[0]))

/* This function prints all the records of the global list */
void print_global_list()
{
    struct list *temp = list;

    printf("Global List of Names:\n");

ViewList:
    while(temp != NULL){
        printf("Name: %8s Age: %3d, Gender: ",temp->name,temp->information.age);
        if(temp->information.gender == female)
            printf("female\n");
        else
            printf("male\n");
        temp = temp->next;
    }
    return;
}

/* This function adds a record
void add_to_list(char *name,
{
    static struct list s[COUNT];
    static int count = 0;
    struct list *temp = &s[count

    memcpy(temp->name, name,
    temp->information.age = age;
    temp->information.gender =
    temp->next = list;
```

MULTI®

Integrated Development Environment for Device Software Optimization

The most powerful & proven embedded software tools

Green Hills Software's MULTI integrated development environment (IDE) includes the industry's most powerful and proven tools for developing embedded software with maximum reliability, maximum performance, and minimum code size. With the MULTI IDE's sophisticated capabilities, you can develop, debug, and optimize code more quickly, significantly reducing both development cost and time.

Ideal for enterprise-wide standardization

The MULTI development environment supports more target processors, operating systems, and third-party tools than any other IDE—making it ideal for enterprise-wide deployment. By using a common set of development tools across projects, software engineers can more easily share code or move between projects without compromising productivity.

With MULTI, you can develop code in C, C++, and FORTRAN (and Ada 95 with Green Hills Software's AdaMULTI IDE). MULTI supports all major 32- and 64-bit processor architectures—as well as the leading embedded and general-purpose operating systems and development hosts.

The MULTI IDE's tight integration with a broad array of third-party tools provides you with maximum flexibility and choice. You can optimize your own development environment—invoking your favorite tools, editor, or configuration management system all from within the MULTI environment. MULTI can even be used with your existing Eclipse environment.

Supported target processors*

- PowerPC
- ARM/Thumb
- XScale/StrongARM
- MIPS64/MIPS32/MIPS16
- x86/Pentium
- ColdFire
- 680x0/683xx/CPU32
- V8xx
- Blackfin
- Intrinsity
- FirePath
- TriCore
- M•Core
- SuperH/SH/SH-DSP
- ST100
- i960
- SPARC/SPARClite
- RH32
- ARC
- RAD6000
- StarCore
- Lexra
- M32R
- Alpha
- FR
- ZSP

Supported target operating systems*

- From Greens Hills Software
  - INTEGRITY OS family
  - *velOSity* microkernel
  - Express Logic's ThreadX RTOS
- In-house operating systems
- Bare board systems with no operating system
- Third-party
  - Linux (embedded or self-hosted)
  - VxWorks
  - OSE
  - Windows
  - Solaris

Supported development hosts*

- Windows
- Linux
- Solaris
- HP-UX

** Green Hills Software is continually adding support for new processors and operating systems. If yours is not listed here, please contact Green Hills software for the latest availability information.*

With support for 25 processor families and the most popular operating systems, the MULTI development environment enables organizations to move people and code seamlessly between projects.

Maximize your device software ROI

While software is the most time-consuming and highest-risk aspect of electronic product development, it also holds the greatest potential to enhance a product's profit. With the most powerful tools available for embedded software development, MULTI can help you slash development costs and manufacturing cost of goods while increasing revenue by maximizing reliability and performance.

Efficiently develop code

- **Create projects quickly—**The Project Builder's intuitive GUI automates and simplifies the configuration of complex programming projects. With its automatic dependency determination, the Builder also helps cut time-to-market by eliminating the need to write and debug makefiles.
- **Edit source code efficiently—**The Source Code Editor saves valuable development time by enhancing code readability and simplifying code writing. Editor features include syntax coloring, auto-indenting, and shortcut menus that link procedure calls to their definitions, declarations, and usages even when these references reside in separate files.
- **Simulate hardware for cost-effectiveness—**The easy-to-use Instruction Set Simulator simulates your embedded platform on your development host so that you can develop and debug applications when hardware resources are scarce or unavailable.

Dramatically reduce debugging time

- **Improve reliability with run-time error checking—**Run-Time Error Checking pinpoints the source of a variety of run-time errors, eliminating the need to search for bugs in problem-free areas. By simplifying the identification of run-time errors, the Error Checking tool both shortens the debugging process and improves product reliability.
- **Debug easily with cross-references—**Cross-Reference debugger simplifies the search for references to any symbol or macro in a program by showing the location and role of any variable or procedure. This quick listing capability makes it easier to find bugs and understand the source code where variables are located.
- **Navigate code effortlessly—**MULTI's unparalleled debugger helps you navigate even very large and complex programs quickly and efficiently. For example, clicking on a procedure or class name automatically takes you to its definition. Also, intuitive back and forward buttons move you through source code just like a Web browser.
- **Browse classes for effective debugging—**The Class Browser graphically illustrates class hierarchies, displaying relationships between parent and child classes and structs. With this visual representation of program organization, you can more readily understand existing C++ programs and, as a result, know better how to debug them.
- **Browse calls to facilitate debugging—**The Calls Browser graphically illustrates function calls so you can better understand program control flow, which helps accelerate the debugging process. This browser displays static and dynamic call graphs in a hierarchical view, color-coded by type of function or method.
- **View memory for faster debugging—**To help you better understand the inner workings of a program at a given moment in time, the Debugger offers a variety of tools for displaying memory "snapshots." The Data Explorer, Graph View, Memory View, and Cache View tools illustrate memory content in a variety of formats and structures.
- **Debug multiple processors seamlessly—**Whether an applications runs on one or many processors, the MULTI Debugger displays program information in a unified manner so you can view the system as a whole. The information displayed is seamless for both homogenous and heterogeneous processor systems.

Optimize for fast performance & small size

- **Analyze performance for greater product speed—**The Performance Analyzer helps pinpoint where best to focus optimization efforts by identifying both the number of times a block of code has executed and which program areas take longest to execute.
- **Analyze code coverage to boost reliability—**The Code Coverage Analysis utility improves product quality by identifying whether or not specific areas of code have been executed. By clearly demarking executed from unexecuted code, a code coverage analysis—done on either a single program execution or a group of runs—makes it simpler to pinpoint problem areas.

Jumpstart product development

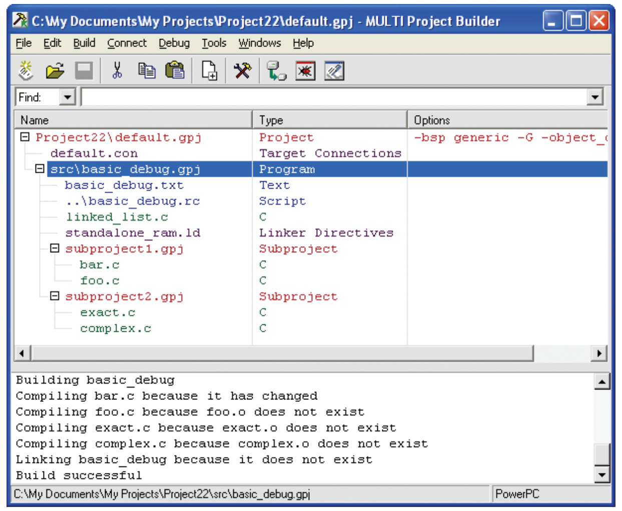
The MULTI IDE significantly reduces the usual overhead of starting a new project or taking on an existing one. As a result, you can jumpstart the development process and focus fully on creating and optimizing the software that is critical to your product's success.

Project Builder: set up new projects quickly & reliably

Whether your project is straightforward or complex, the MULTI IDE's Project Builder streamlines program configuration so you can quickly start writing and debugging code.

The New Project Wizard simplifies the project creation process for both novice and veteran MULTI IDE users by creating a template as a starting point for your application (MULTI includes templates for every supported target board and processor). Once your project is set up, automatic dependency determination further cuts development time by eliminating the need to write and debug makefiles. Finally, with the parallel build setting, you can specify that certain build processes run parallel to decrease build times on single- or multiple-processor machines.

The Project Builder also includes a variety of features to accelerate project development. For every option that can be set in the GUI, on-screen documentation is provided in the lower panel of the Build Options window. Also, both the Builder and the Build Options windows display command line equivalents of all the options set on a file. You can use these commands at the command line or integrate them into an existing makefile.


C:\My Documents\My Projects\Project22\default.gpj - MULTI Project Builder
File
Edit
Build
Connect
Debug
Tools
Windows
Help
Find:

```
Name                        Type                 Options
⊟ Project22\default.gpj     Project              -bsp generic -G -object_
    default.con             Target Connections
  ⊟ src\basic_debug.gpj     Program
      basic_debug.txt       Text
      ..\basic_debug.rc     Script
      linked_list.c         C
      standalone_ram.ld     Linker Directives
    ⊟ subproject1.gpj       Subproject
        bar.c               C
        foo.c               C
    ⊟ subproject2.gpj       Subproject
        exact.c             C
        complex.c           C
```

```
Building basic_debug
Compiling bar.c because it has changed
Compiling foo.c because foo.o does not exist
Compiling exact.c because exact.o does not exist
Compiling complex.c because complex.o does not exist
Linking basic_debug because it does not exist
Build successful
```

C:\My Documents\My Projects\Project22\src\basic_debug.gpj PowerPC

The MULTI Project Builder provides a number of features that streamline or automate the administrative overhead of setting up a new project.

Source Code Editor: read & write code efficiently

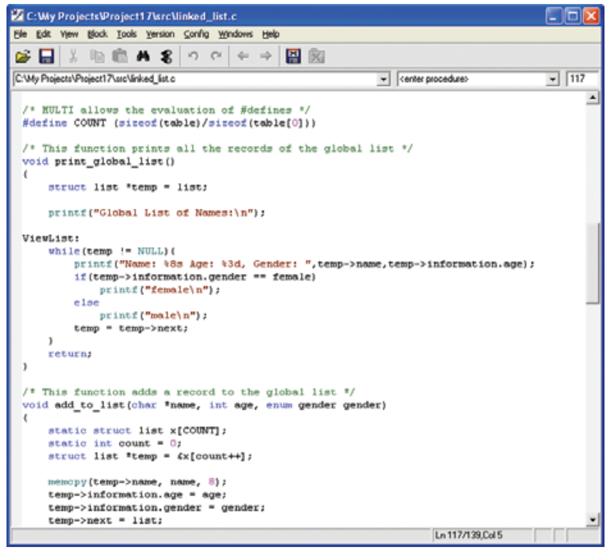
With its variety of automated and customizable features the Source Code Editor enables you to efficiently create and modify source and other text files. Features include:

- formatting help
- syntax coloring for many languages
- auto-indenting
- auto-completion of symbol names

The Editor automatically configures itself based on the programming language of the current source file. When working with large code bases that use different languages, you can automatically access features that are customizable for any language. Even when using internally-developed file types, you can read and write code efficiently.

The Editor provides easy access to files, procedures, and lines, so you can navigate projects quickly. Shortcut menus contain links to the definition, declaration, and usage of objects, functions, variables, and types.

The Source Code Editor is fully-configurable, enabling you to change its appearance and functionality to help you work more effectively. Like other MULTI components, it is compatible with a variety of host/target development combinations and works with a range of third-party configuration management and version control tools, such as ClearCase and CVS.


C:\My Projects\Project17\src\linked_list.c
File
Edit
View
Block
Tools
Version
Config
Windows
Help
C:\My Projects\Project17\src\linked_list.c
<enter procedure>
117

```
/* MULTI allows the evaluation of #defines */
#define COUNT (sizeof(table)/sizeof(table[0]))

/* This function prints all the records of the global list */
void print_global_list()
{
    struct list *temp = list;

    printf("Global List of Names:\n");

ViewList:
    while(temp != NULL){
        printf("Name: %8s Age: %3d, Gender: ",temp->name,temp->information.age);
        if(temp->information.gender == female)
            printf("female\n");
        else
            printf("male\n");
        temp = temp->next;
    }
    return;
}

/* This function adds a record to the global list */
void add_to_list(char *name, int age, enum gender gender)
{
    static struct list x[COUNT];
    static int count = 0;
    struct list *temp = &x[count++];

    memcpy(temp->name, name, 8);
    temp->information.age = age;
    temp->information.gender = gender;
    temp->next = list;
```

Ln 117/139,Col 5

With MULTI's Source Code Editor, you can edit an unlimited number of files from one window

Instruction Set Simulator: Develop applications without hardware

By simulating your embedded processor on your development host, MULTI's Instruction Set Simulator makes it possible to develop embedded applications even when target hardware is scarce or unavailable.

Using a simulator is ideal in a number of scenarios:

- the hardware decision has not been made
- the hardware does not exist
- you do not yet have the hardware
- there are more developers than available hardware
- not all the hardware's device drivers have been written

For applications using Green Hills Software's INTEGRITY RTOS or *velOSity* microkernel, ISIM provides complete API simulations that programmers can use to fully develop and test applications—even those that include operations using virtual memory and system peripherals.

Unlike conventional RTOS simulators that run as native workstation or PC applications, ISIM simulates the same binary code that runs on the target processor. The exact same compilers and other development tools are used for both simulation and target execution, thereby extending the investment in the MULTI development tools as well as maximizing programmer productivity.

Target environments

From Windows, Linux, or Unix hosts, MULTI can use a wide variety of target connections, including:

- **processor probes and ICE**—this includes devices from Green Hills Software, Agilent, Hitachi, ARM, EPI, Macraigor, Wind River (EST), P&E, NEC, and Tektronix.
- **instruction set simulators**—along with Green Hills Software's offerings, simulators from ARC, ARM, and MIPS are also supported
- **ROM monitors**—using RS-232, Ethernet, or parallel ports, MULTI can cross-debug to a variety of ROM monitor targets
- **operating systems**—MULTI supports the most popular operating systems including INTEGRITY, *velOSity*, ThreadX, Linux, VxWorks, OSE, Windows, Solaris, and in-house systems

Green Hills Software processor probes

Green Hills Software's three processor probes provide a range of performance and features to load, control, debug, and test target systems without prior board initialization, an RTOS, or ROM monitor.

- **SuperTrace trace probe**—for the fastest trace, download and debug
- **Green Hills probe**—for high-performance debugging
- **Slingshot debug device**—for low-cost target connections

SuperTrace trace probe

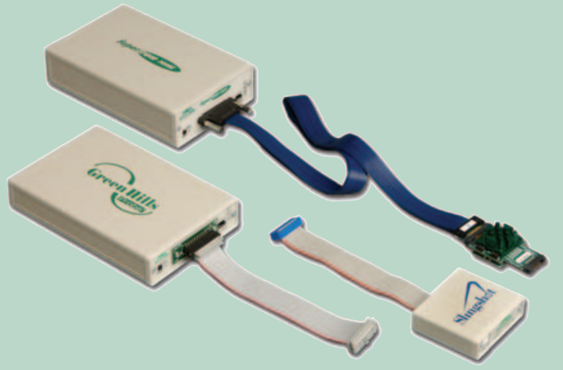
The most powerful of all Green Hills hardware offerings, the SuperTrace Probe can capture up to one gigabyte of trace data at clock speeds over 300 MHz. This exceptionally large storage capability enables the SuperTrace probe to easily collect hundreds of millions of trace frames, radically extending the capture window to help you find and fix more bugs faster and easier than ever.

Green Hills probe

The Green Hills probe is an advanced hardware debug device that enables the MULTI debugger to load, control, debug, and test a target system without the need for prior board initialization, an RTOS, or even a ROM monitor. The Green Hills probe offers high-speed downloads of up to 500 KBps and Ethernet, RS-232 Serial, and USB host interfaces.

Slingshot debug device

Slingshot is a high-performance, low-cost microprocessor debug probe with a single USB 1.1 host interface and supported download speeds of up to 150 KBps. With its on-board 32-bit processor, reconfigurable logic, and compact size (2 7/16" x 2 3/4" x 3/4"), Slingshot outperforms other probes that are three to four times its cost.



All of Green Hills Software's processor probes are tightly integrated with the MULTI IDE and support a variety of 32-bit and 64-bit processor targets.

Find more bugs & find them faster

Studies have shown that a full 50% of software development time is devoted to debugging, an inevitability no matter how well the software is designed. MULTI—with its range of sophisticated tools and utilities—offers the most powerful capabilities available for flushing out even the nastiest bugs. With the MULTI IDE's unparalleled debugging features, you can get a higher quality, more reliable product to market in much less time.

Effortless code navigation

At the heart of the MULTI IDE's debugging capabilities, the Debugger window includes a number of intuitive features that help you effortlessly browse, examine, and search your source code. These include:

- clicking on selections in the Navigation Bar for quick access to files and procedures
- choosing a name from File or Procedure Locator menu to display the appropriate code in the Source Pane
- viewing the definition of a particular procedure or class by clicking on the symbol name anywhere it appears

Other time-saving features include forward and back buttons that navigate through files and procedures recently visited, similar to the way the back and forward buttons work in a Web browser.

The Debugger's advanced search capabilities let you issue navigation commands with wildcard characters so that you can easily find procedures even when you cannot remember their exact names. You may also search your entire source base using regular expressions

Run-Time Error Checking: pinpoint bug locations

With MULTI's Run-Time Error Checking utility, you can accurately and easily pinpoint the location in your source code of a variety of run-time errors, including:

- accessing an element beyond an array's declared bounds
- assigning an out-of-range value to a variable or field of small integral type
- unhandled case in a switch statement
- dividing by zero
- accessing invalid memory through a pointer
- memory leak detection

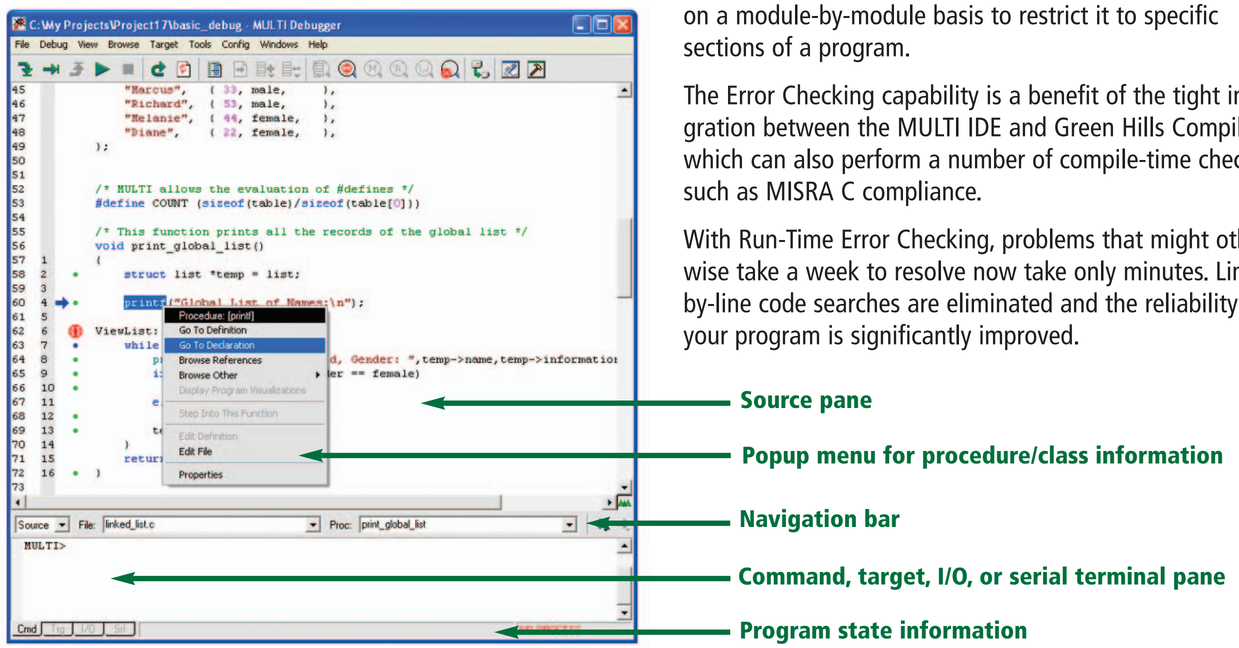
Error Checking can also uncover the true source of bugs usually manifested in secondary effects, such as a variable corrupted by use of an invalid address. In normal circumstances a bug like this can be extremely difficult to track down because it acts differently with each occurrence. Run-Time Error Checking stops the program right where the invalid address is used and reports that it has detected an invalid memory access.

Minimal effect on code execution & speed

MULTI Run-Time Error Checking only minimally effects code size and execution speed, so it is practical for debugging any program. You can also enable or disable Error Checking on a module-by-module basis to restrict it to specific sections of a program.

The Error Checking capability is a benefit of the tight integration between the MULTI IDE and Green Hills Compilers, which can also perform a number of compile-time checks such as MISRA C compliance.

With Run-Time Error Checking, problems that might otherwise take a week to resolve now take only minutes. Line-by-line code searches are eliminated and the reliability of your program is significantly improved.


C:\My Projects\Project17\basic_debug - MULTI Debugger
File Debug View Browse Target Tools Config Windows Help
Procedure: [printf]
Go To Definition
Go To Declaration
Browse References
Browse Other
Display Program Visualizations
Step Into This Function
Edit Definition
Edit File
Properties
Source
File: linked_list.c
Proc: print_global_list
Cmd Trg I/O Srl
Source pane
Popup menu for procedure/class information
Navigation bar
Command, target, I/O, or serial terminal pane
Program state information

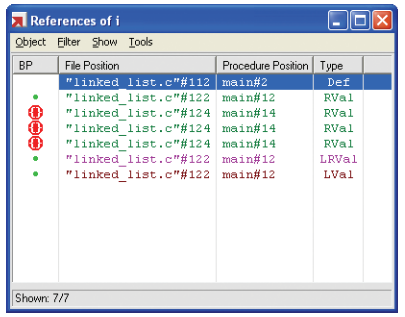
```
        "Marcus",   { 33, male,     },
        "Richard",  { 53, male,     },
        "Melanie",  { 44, female,   },
        "Diane",    { 22, female,   },
    };


    /* MULTI allows the evaluation of #defines */
    #define COUNT (sizeof(table)/sizeof(table[0]))

    /* This function prints all the records of the global list */
    void print_global_list()
    {
        struct list *temp = list;

        printf("Global List of Names:\n");

    ViewList:
        while
            pr                              d, Gender: ",temp->name,temp->informatio
            i                               ler == female)

            e

            te
        }
        retur
    }
```

```
MULTI>
```

The powerful MULTI Debugger window includes a number of features to help navigate through source code quickly and easily.

Cross-Reference Browser: track the lifespan of variables & procedures

The Cross-Reference Browser enables you to track the role played by any specific variable or procedure throughout your program and quickly discover how it is being used (or misused). For any symbol, the Browser window can display a complete list of references that clearly differentiates definitions, declarations, assignments, and value references. References in the Debugger source pane are also highlighted to make their locations apparent.



BP	File Position	Procedure Position	Type
	"linked_list.c"#112	main#2	Def
●	"linked_list.c"#122	main#12	RVal
	"linked_list.c"#124	main#14	RVal
	"linked_list.c"#124	main#14	RVal
	"linked_list.c"#124	main#14	RVal
●	"linked_list.c"#122	main#12	LRVal
●	"linked_list.c"#122	main#12	LVal

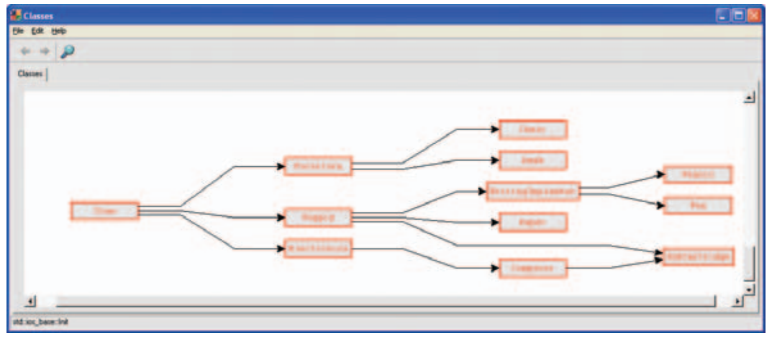
Shown: 7/7

You can easily track how a specific variable or procedure is used throughout your program with the Cross-Reference Browser.

Class Browser: understand class hierarchy

The powerful Class Browser provides high-level visibility of program organization by displaying all class hierarchies. Using the Class Browser, you can readily understand the relationships between parent and child classes. Like other MULTI tools, the Class Browser is tightly linked to the source code. To view the details of any class' declaration, you simply click on it in the Browser.

The Class Browser displays information in two modes: *tree view* mode—ideal for showing simple hierarchies—and *graph* mode—better for representing hierarchies that include multiple inheritance. Both modes offer expandable and collapsible nodes so you can drill down into details as necessary. In tree view, classes, structures, and unions are color-coded to distinguish among types.



The Class Browser displays classes in two modes: tree view (above) for showing simple hierarchies and graph mode for hierarchies that include multiple inheritance.

Calls Browser: view program control flow

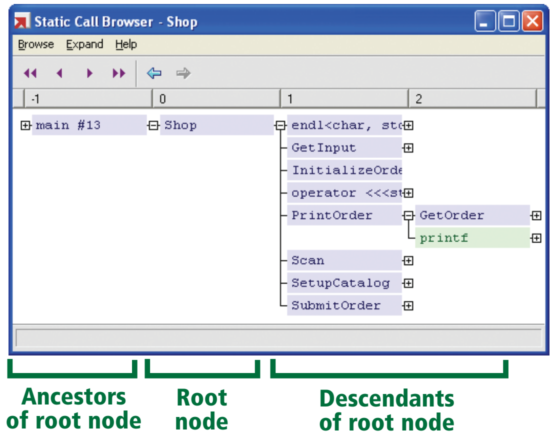
The Calls Browser gathers all of a program's function calls in a hierarchical table that provides a graphical view of program control flow to help you better understand your program.

The Browser provides visual representations of both static and dynamic call hierarchies. Static graphs show all potential calls, while dynamic graphs use profiling information or trace data to display how a program actually ran. The Browser also displays static calls by file, so you can readily see interactions between code in different source files and better understand the relationships between functions.

In addition to call hierarchies, the Calls Browser can display a single function and show the functions it calls or the functions that call it. The Browser's view can be expanded to include neighboring functions, the functions they call, etc.

For fast debugging, you can jump directly from the Calls Browser to a function in the Debugger or the Editor to easily modify it. The intuitive Browser window color-codes nodes by type so at-a-glance you can readily differentiate:

- recursive functions
- functions called via function pointers
- functions with debugging information
- functions without debugging information



Through the Calls Browser lists you can readily differentiate function types and understand their relationships.

MULTI debugging features, continued

Data and memory visualizations show you the root causes of problems

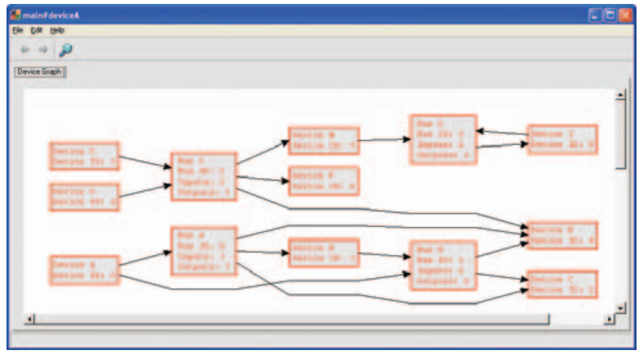
To help you dig deeply into the workings of your program, the MULTI Debugger provides a number of tools for viewing memory content, each customized for displaying different types of data:

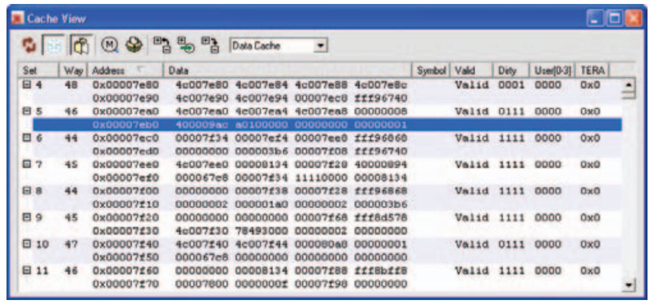
- Data Explorer
- PCI Devices Viewer
- Register Viewer
- Memory Viewer
- Cache Viewer

All these tools offer an easy way to analyze the information a program uses and see if it contains unexpected values. With the memory "snapshots" these tools provide, you can get to the root causes of bugs more quickly.

A high-level view of variable values

The Data Explorer window displays memory structurally to provide a high-level understanding of variable values. The Data Explorer displays variable types and current values, making it ideal for simple variables and data. With the Data Explorer, you can easily follow a linked list or other large data hierarchy simply by expanding nested data structures. This window also allows in-place editing of values. In contrast to the Data Explorer, the Graph View window best models complex structures and enables you to create a customized, high-level view of any system of objects or structures.





```
Cache View
Data Cache
Set   Way Address     Data                                  Symbol Valid Dirty User[0-3] TERA
⊟ 4   48  0x00007e80  4c007e80 4c007e84 4c007e88 4c007e8c          Valid 0001  0000      0x0
          0x00007e90  4c007e90 4c007e94 00007ec8 fff96740
⊟ 5   46  0x00007ea0  4c007ea0 4c007ea4 4c007ea8 00000008          Valid 0111  0000      0x0
          0x00007eb0  400009ac a0100000 00000000 00000001
⊟ 6   44  0x00007ec0  00007f34 00007ef4 00007ee8 fff96868          Valid 1111  0000      0x0
          0x00007ed0  00000000 000003b6 00007f08 fff96740
⊟ 7   45  0x00007ee0  4c007ee0 00008134 00007f28 40000894          Valid 1111  0000      0x0
          0x00007ef0  000067e8 00007f34 11110000 00008134
⊟ 8   44  0x00007f00  00000000 00007f38 00007f28 fff96868          Valid 1111  0000      0x0
          0x00007f10  00000002 000001a0 00000002 000003b6
⊟ 9   45  0x00007f20  00000000 00000000 00007f68 fff8d578          Valid 1111  0000      0x0
          0x00007f30  4c007f30 78493000 00000002 00000000
⊟ 10  47  0x00007f40  4c007f40 4c007f44 000080a8 00000001          Valid 0111  0000      0x0
          0x00007f50  000067e8 00000000 00000000 00000000
⊟ 11  46  0x00007f60  00000000 00008134 00007f88 fff8bff8          Valid 1111  0000      0x0
          0x00007f70  00007800 0000000f 00007f98 00000000
```

MULTI offers tools to analyze memory data types including values in variables and registers, register definitions, PCI-specific data, and unprocessed memory content.

The look and feel of a data book

The Register View window enables you to track and interpret the values of registers on your hardware. Designed to look and feel like a data book, the Register View window offers a bit-by-bit breakdown of important register values. By pairing this information with on-screen documentation, the Register View window can dramatically cut the time it takes to understand the implications of registers' values.

List and control PCI devices

When debugging a system that uses a PCI bus, the PCI Devices window allows you to list and control the devices on your board's bus. The PCI Devices window displays all of the PCI devices on your target along with important information for each device including:

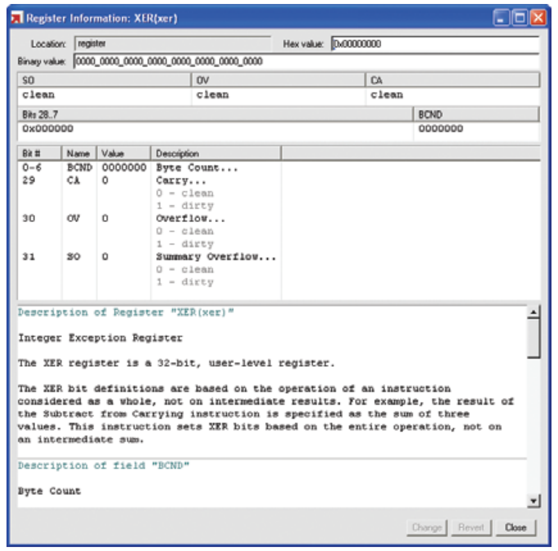
- bus position,
- running speed
- active error conditions.

This capability extends across the PCI-PCI bridge, which is especially useful in a PCI chassis.

Viewing the bits and bytes of memory and cache

When you need to drill down to the deepest levels, the Memory View and Cache View windows provide low-level views of memory data that enable you to analyze unprocessed memory content and and get to the heart of certain types of bugs.

The Memory View window—which can display memory in hexadecimal, decimal, ASCII, binary, and floating point formats—is useful for examining large buffers, strings, and raw memory. For easier debugging, you can also modify memory from this window. The Cache View window, which displays all of the entries located in accessible caches, helps you determine if differences exist between data in the cache and data in memory.



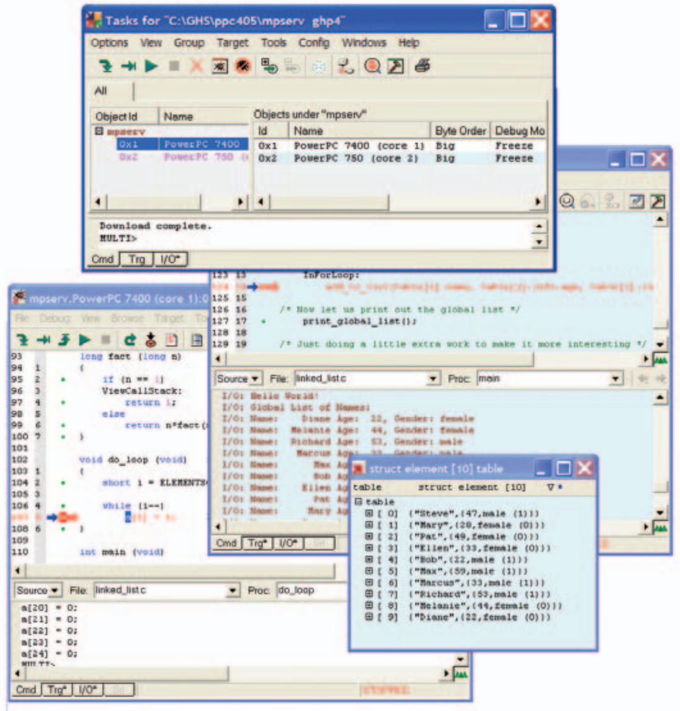
```
Register Information: XER(xer)
Location: register                 Hex value: 0x00000000
Binary value: 0000_0000_0000_0000_0000_0000_0000_0000
SO          OV          CA
clean       clean       clean
Bits 28..7                         BCND
0x000000                           0000000

Bit #  Name  Value    Description
0-6    BCND  0000000  Byte Count...
29     CA    0        Carry...
                        0 - clean
                        1 - dirty
30     OV    0        Overflow...
                        0 - clean
                        1 - dirty
31     SO    0        Summary Overflow...
                        0 - clean
                        1 - dirty

Description of Register "XER(xer)"

Integer Exception Register

The XER register is a 32-bit, user-level register.

The XER bit definitions are based on the operation of an instruction
considered as a whole, not on intermediate results. For example, the result of
the Subtract from Carrying instruction is specified as the sum of three
values. This instruction sets XER bits based on the entire operation, not on
an intermediate sum.

Description of field "BCND"

Byte Count
```

Seamless debugging of multitasking and multiple processor systems

For projects with multiple threads of execution, multiple processors, multiple boards, or any combination of these, MULTI offers a coherent debugging environment for visualizing and debugging the system. For each process or processor, MULTI opens a separate debug window. In each window, you can single step, set breakpoints, and examine data the same way you would with any other application.

With multiple processor systems, the Debugger shows all programs running on your target, so you do not have to debug complex interactions one small piece at a time. Through separate windows on your desktop, you can watch the interaction as the application executes on different processors.The Debugger works seamlessly no matter if the processors are homogenous or heterogeneous.

In addition to seamless debugging, multiple processor support enables significant cost benefits. You can place multiple processors on the same board and use a single, low-pincount JTAG connection to view and control all of them simultaneously. By consolidating the overhead of multiple connectors into a single interface, you can significantly reduce costs on today's crowded boards.



```
Tasks for "C:\GHS\ppc405\mpserv ghp4"
Options View Group Target Tools Config Windows Help
All
Object Id  Name            Objects under "mpserv"
⊟ mpserv                   Id   Name                   Byte Order  Debug Mo
  0x1      PowerPC 7400    0x1  PowerPC 7400 (core 1)  Big         Freeze
  0x2      PowerPC 750     0x2  PowerPC 750 (core 2)   Big         Freeze

Download complete.
MULTI>
```

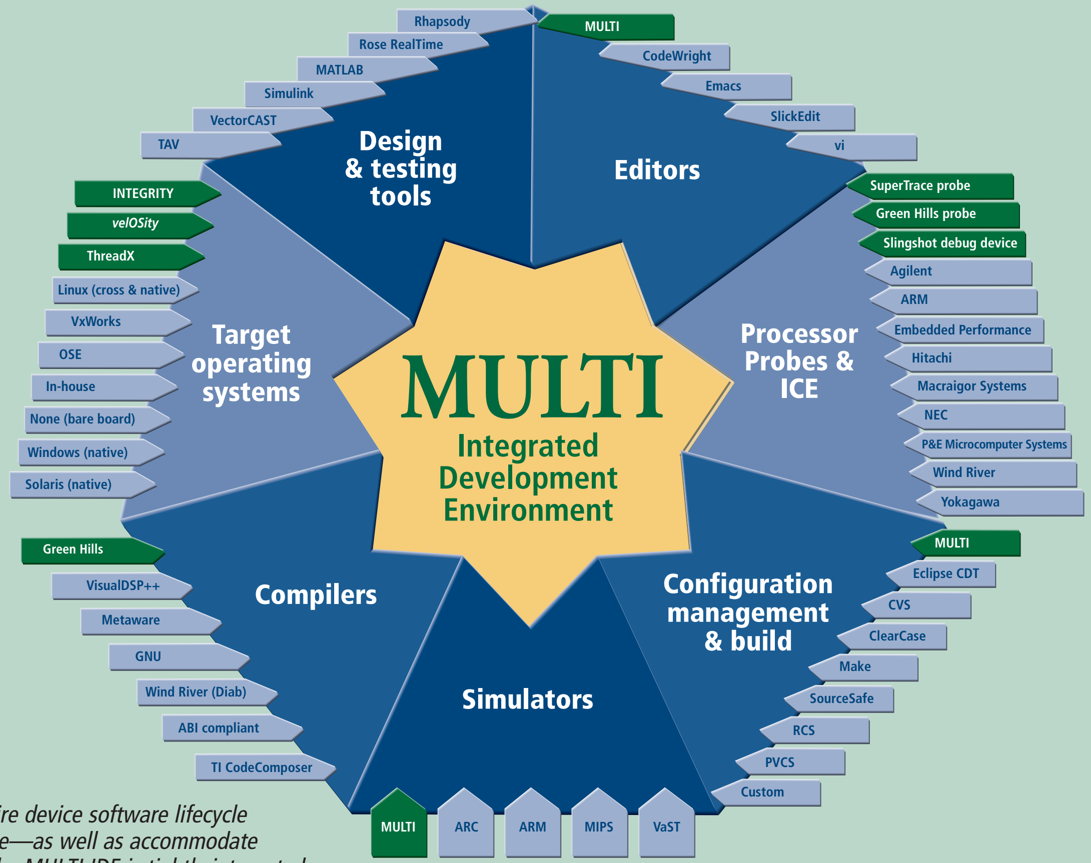
```
mpserv.PowerPC 7400 (core 1):0
93         long fact (long n)
94   1     {
95   2         if (n == 1)
96   3     ViewCallStack:
97   4             return 1;
98   5         else
99   6             return n*fact(
100  7     }
101
102        void do_loop (void)
103  1     {
104  2         short i = ELEMENTS
105  3
106  4         while (i--)
108  6     }
109
110        int main (void)
Source  File: linked_list.c  Proc: do_loop
a[20] = 0;
a[21] = 0;
a[22] = 0;
a[23] = 0;
a[24] = 0;
```

```
123 13          InForLoop:
125 15
126 16      /* Now let us print out the global list */
127 17          print_global_list();
128 18
129 19      /* Just doing a little extra work to make it more interesting */
Source  File: linked_list.c  Proc: main
I/O: Hello World!
I/O: Global List of Names:
I/O: Name:    Diane Age:  22, Gender: female
I/O: Name:  Melanie Age:  44, Gender: female
I/O: Name:  Richard Age:  53, Gender: male
I/O: Name:   Marcus
I/O: Name:      Max
I/O: Name:      Bob
I/O: Name:    Ellen
I/O: Name:      Pat
I/O: Name:     Mary
```

```
struct element [10] table
table          struct element [10]
⊟ table
  ⊞ [ 0]  ("Steve",(47,male (1)))
  ⊞ [ 1]  ("Mary",(28,female (0)))
  ⊞ [ 2]  ("Pat",(49,female (0)))
  ⊞ [ 3]  ("Ellen",(33,female (0)))
  ⊞ [ 4]  ("Bob",(22,male (1)))
  ⊞ [ 5]  ("Max",(59,male (1)))
  ⊞ [ 6]  ("Marcus",(33,male (1)))
  ⊞ [ 7]  ("Richard",(53,male (1)))
  ⊞ [ 8]  ("Melanie",(44,female (0)))
  ⊞ [ 9]  ("Diane",(22,female (0)))
```

In multi-processor systems, the MULTI Debugger opens a separate window for each target so you can more readily understand the complex interactions between different programs.

Unparalleled third-party integrations



To support the entire device software lifecycle across an enterprise—as well as accommodate personal tastes—the MULTI IDE is tightly integrated with dozens of tools from Green Hills Software partners.

Optimize for fastest code, smallest size

Debugging your code is only half the battle. To push your product ahead of the competition, you need to fine-tune it to be cost-effective as well as run fast and flawlessly. To enable this, the MULTI IDE includes unique capabilities for improving the performance, efficiency, and economy of your embedded software.

Performance Profiler: efficiently focus optimization efforts

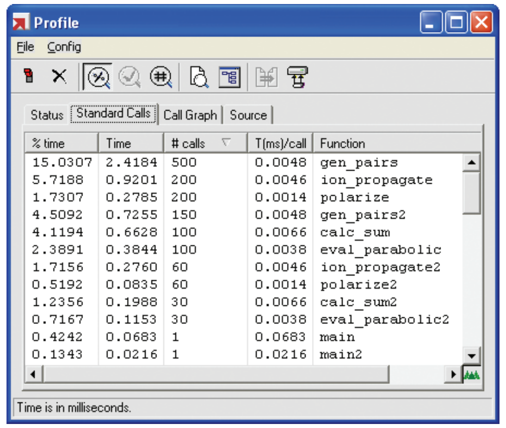
The Performance Profiler provides unequaled insight into the behavior of your program by precisely specifying:

- the percentage of time spent executing each source line or instruction
- the total number of times each line or instruction was executed
- the total number of times each function was called

With instant access to accurate and detailed program execution data, you have a powerful tool for optimizing application performance. By showing where code execution consumes the most time, the Profiler pinpoints exactly where you should focus optimization efforts. The profiler can gather, arrange, and display profile information in a variety of formats, including by function, block of code, and line of code.

The MULTI IDE uses various methods for collecting the data the Performance Profiler displays, including a trace-capable debug device, such as Green Hills Software's SuperTrace probe. The probe gathers precise profiling information without instrumenting your code. Trace-based profiling provides the least intrusive look at program execution.*

** The Performance Analyzer is configuration-dependent and may require custom integration. Please contact your sales representative for more information.*



% time	Time	# calls	T(ms)/call	Function
15.0307	2.4184	500	0.0048	gen_pairs
5.7188	0.9201	200	0.0046	ion_propagate
1.7307	0.2785	200	0.0014	polarize
4.5092	0.7255	150	0.0048	gen_pairs2
4.1194	0.6628	100	0.0066	calc_sum
2.3891	0.3844	100	0.0038	eval_parabolic
1.7156	0.2760	60	0.0046	ion_propagate2
0.5192	0.0835	60	0.0014	polarize2
1.2356	0.1988	30	0.0066	calc_sum2
0.7167	0.1153	30	0.0038	eval_parabolic2
0.4242	0.0683	1	0.0683	main
0.1343	0.0216	1	0.0216	main2

Time is in milliseconds.

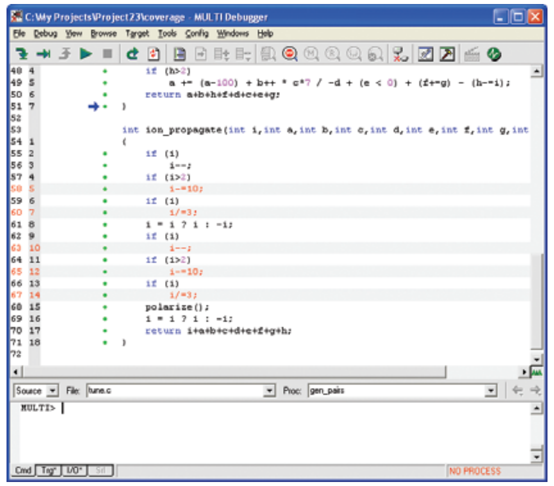
The Performance Profiler provides accurate and detailed program execution data so you know exactly where to focus optimization efforts.

Code Coverage Analysis: eliminate unexecuted lines of code

To help boost both product efficiency and reliability, the Code Coverage Analysis utility displays which lines of code were—and were not—executed during one or more runs of a program. In addition to highlighting unexecuted code in the Debugger source pane, the Code Coverage Analysis utility can output results in a variety of report formats.

By quickly distinguishing which lines of code didn't execute, the Code Coverage Analysis utility helps ensure comprehensive testing. It either reports that an entire program has been tested or notifies you of missed spots. It also allows you to safely delete unexecuted code and streamline your program.

Besides being particularly effective in software verification and testing, Green Hills Software's advanced analysis tools automate structural coverage analysis for application software. This capability can be especially useful to development teams who are working to comply with standards such as those required for DO-178B certification.



```
    if (h>2)
        a += (a-100) + b++ * c*7 / -d + (e < 0) + (f+=g) - (h-=i);
    return a+b+h+f+d+c+e+g;
}

int ion_propagate(int i,int a,int b,int c,int d,int e,int f,int g,int
{
    if (i)
        i--;
    if (i>2)
        i-=10;
    if (i)
        i/=3;
    i = i ? i : -i;
    if (i)
        i--;
    if (i>2)
        i-=10;
    if (i)
        i/=3;
    polarize();
    i = i ? i : -i;
    return i+a+b+c+d+e+f+g+h;
}
```

```
MULTI>
```

By highlighting lines of unexecuted code, the Code Coverage Analysis utility helps ensure comprehensive testing as well as reduce code size.

Compiler support

MULTI is tightly integrated with Green Hills Software's optimizing C, C++, Ada 95, and FORTRAN compilers which consistently produce the fastest code in independent industry benchmarks. MULTI can also be used with compilers that adhere to EABI standards. Green Hills compilers produce EABI compliant ELF object models.

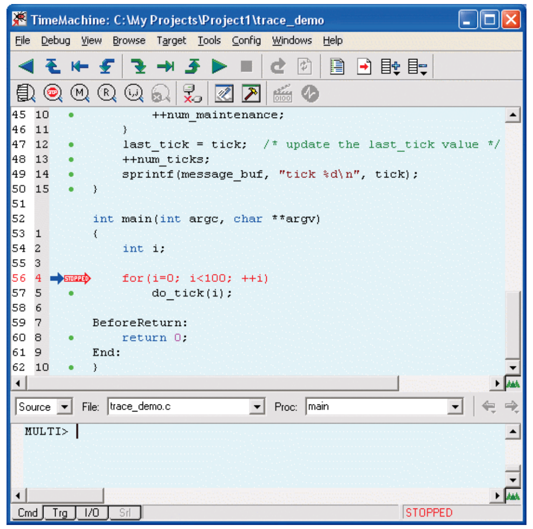
Extending the power of the MULTI IDE

Green Hills Software offers add-ons to the MULTI toolset that enable you to delve even deeper into your program to better understand the complex interactions between the various hardware and software components in your system. These tools leverage the tight integration between MULTI and Green Hills Software's INTEGRITY and *velOSity* operating systems as well as tap in to the enhanced capabilities in selected microprocessors.

TimeMachine tool suite: fix bugs faster

A truly innovative tool, the TimeMachine suite extends the familiar MULTI debugger interface with functions that let developers see back in time. TimeMachine enables analysis of program behavior leading up to the manifestations of a bug or problem. As a result, TimeMachine can eliminate trial-and-error debugging and make all bugs easier to fix.

The TimeMachine suite includes a source-level debugger to run and step an applications back in time, PathAnalyzer and Function Flow tools, and the MULTI EventAnalyzer and Performance Profiler. TimeMachine is powered by trace history data that can be collected using instruction set simulator, TraceEdge, or the SuperTrace probe.



```
            ++num_maintenance;
        }
        last_tick = tick;  /* update the last_tick value */
        ++num_ticks;
        sprintf(message_buf, "tick %d\n", tick);
    }

    int main(int argc, char **argv)
    {
        int i;

        for(i=0; i<100; ++i)
            do_tick(i);

    BeforeReturn:
        return 0;
    End:
    }
```

```
MULTI>
```

The TimeMachine debugger combines a familiar interface with innovative functions that let you step and run both forward and backward through code.

ResourceAnalyzer: maximize task efficiency

Customized for the INTEGRITY and *velOSity* RTOSes, the ResourceAnalyzer is an advanced run-time analysis tool that helps you visualize how an application consumes resources. With the ResourceAnalyzer you can track CPU and memory usage to maximize the efficiency of tasks and AddressSpaces and improve overall system performance.

The ResourceAnalyzer's display makes it easy to track CPU and memory statistics for an entire embedded system or for any task or AddressSpace. At a glance, it shows how these resources are consumed and also warns of potential problems when unexpected values occur. In addition, you can configure the ResourceAnalyzer to focus on only the information significant to your work.

EventAnalyzer: accurately visualize complex, real-time operating system events

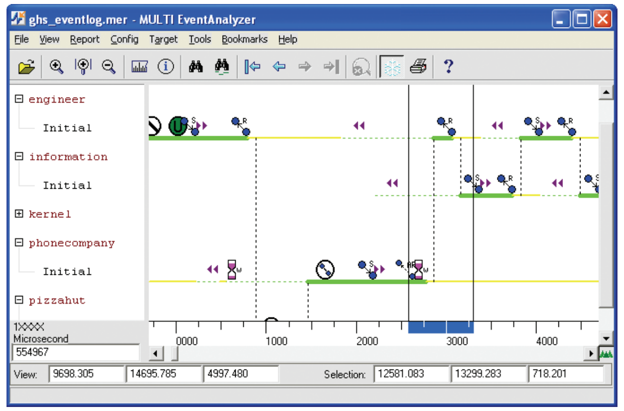
By providing graphical display of system activities over time, the powerful EventAnalyzer enables you to easily and accurately visualize complex, real-time relationships within your system. Using this sophisticated utility you can more readily identify bottlenecks and optimize your entire system. System events charted by the EventAnalyzer include:

- context switches
- RTOS kernel calls
- interrupts and exceptions
- status changes
- user-defined events

With the Event Analyzer you can isolate and examine complicated interactions that occur in only microseconds, even those distributed across multiple processors. With its ability to chart operating system events over time, you can more easily detect timing anomalies that can be particularly difficult to reproduce and debug.

For a program running on the INTEGRITY or *velOSity* operating system, you can use the Green Hills SuperTrace probe to collect EventAnalyzer data with absolutely no code instrumentation.

To maximize system visibility, user-defined events can be logged and viewed with the EventAnalyzer. This additional data—recorded in real time and customized to specific debugging requirements—can provide key indicators of program behavior.



View: 9698.305 14695.785 4997.480

Selection: 12581.083 13299.283 718.201

With the EventAnalyzer's graphical display of operating system events you can more readily identify bottlenecks and easily track down problematic interactions.

Corporate Headquarters

30 West Sola Street
Santa Barbara, CA 93101
ph: 805.965.6044
fax: 805.965.6343
email: sales@ghs.com
www.ghs.com

North American Sales Offices

Arizona - Phoenix
Tel: 602.953.6123

California - Los Angeles
Tel: 805.965.6044

California - San Clemente
Tel: 949.369.3950

California - Santa Clara
Tel: 408.919.0800

California - San Diego
Tel: 858.455.7000

Canada - Quebec
Tel: 450.458.1150

Colorado - Denver
Tel: 303.740.8462

Georgia - Atlanta
Tel: 404.873.6285

Illinois - Chicago
Tel: 847.515.2418

Massachusetts - Lexington
Tel: 781.862.2002

Massachusetts - Topsfield
Tel: 978.887.0051

North Carolina - Mars Hill
Tel: 828.689.8508

Ohio - Dayton
Tel: 937.848.8644

Texas - Dallas
Tel: 972.712.1407

Washington - Bothell
Tel: 425.402.3692

International Sales Offices

Australia
Tel: +61.4.3744.7638

France
Tel: +33.1.46.96.07.00

Germany
Tel: +49.721.98.62.580

Israel
Tel: +972.9.958.4060

Japan
Tel: +81.3.3576.5351

Netherlands
Tel: +31.33.4613363

Sweden
Tel: +46.46.211.33.70

United Kingdom
Tel: +44.1844.267950

North American Safety Critical Products Sales Offices

California - Laguna Hills
Tel: 949.460.6442

Florida - Palm Harbor
Tel: 727.781.4909

Technical Support

North America Mountain/Pacific time, Australia, New Zealand
support-west@ghs.com
or 1.877.GHS.TECH

North America Eastern/Central time, South America
support-east@ghs.com
or 1.877.GHS.TECH

Europe, Africa, India
support-nl@ghs.com
or +31.33.4613363

Japan, Taiwan, South Korea
support@adac.co.jp
or +86.10.62000988

Safety Critical Technical Support
support-ada@ghs.com
or 800.789.9695

Green Hills®
• SOFTWARE, INC. •