

Debug@Chip Users Guide

Author: Jack Gatrost

Date: 26 April 2003

1	<u>Introduction</u>	5
1.1	<u>Intended Users</u>	5
1.2	<u>Requirements from Environment</u>	5
1.2.1	<u>Windows PC</u>	5
1.2.2	<u>SC12 Resources Required</u>	5
1.3	<u>Target Programs Under Test in SC12</u>	6
1.4	<u>Functionality Overview</u>	6
1.4.1	<u>Borland Symbol Table Usage</u>	6
1.4.2	<u>Real-time Data Watch</u>	8
1.4.3	<u>Breakpoint Control</u>	9
1.4.4	<u>Single Step</u>	10
1.4.5	<u>Task Control</u>	11
1.5	<u>Window Types</u>	11
1.5.1	<u>Watch Windows</u>	11
1.5.2	<u>Assembly Code Windows</u>	11
1.5.3	<u>File Editor Windows</u>	12
1.5.4	<u>Assembly/Source Mix View</u>	12
1.5.5	<u>Registers Window</u>	12
1.5.6	<u>Local Variables Window</u>	12
1.5.7	<u>Calling Tree Window</u>	12
1.5.8	<u>Timing Window</u>	12
1.5.9	<u>Symbol Table Window</u>	12
1.6	<u>Additional Functionality</u>	12
1.6.1	<u>Code Navigation</u>	12
1.6.2	<u>Line Assembler</u>	12
1.6.3	<u>Program Download</u>	13
1.6.4	<u>IDE Builds</u>	13
1.6.5	<u>Help System</u>	13
2	<u>Getting Started</u>	13
2.1	<u>Installing Debug@Chip Program on PC</u>	13
2.1.1	<u>Removing Debug@Chip Program from PC</u>	13
2.2	<u>SC12 Target Configuration</u>	13
2.3	<u>Debug@Chip Program Configuration</u>	13
2.3.1	<u>Target Programs</u>	14
2.3.2	<u>Comm</u>	16
2.3.3	<u>Save/Restore</u>	16
2.3.4	<u>Options</u>	17
3	<u>Debug@Chip Screen</u>	17
3.1	<u>Status Line</u>	17
3.1.1	<u>CPU Status Window</u>	17
3.1.2	<u>Clock / Stop Watch Window</u>	17
3.1.3	<u>Message Window</u>	18
3.2	<u>Main Toolbar</u>	18
4	<u>Main Menu Operation</u>	19
4.1	<u>File Submenu</u>	19

4.2	Breakpoints Submenu	21
4.3	CPU Submenu	22
4.4	Grep	23
5	Window Operation	23
5.1	Overview	23
5.2	Watch Windows	23
5.2.1	Watch Controls Submenu	24
5.2.2	Watch Window Right Mouse Pop-up Menu	26
5.3	Assembly Code Windows	26
5.3.1	Assembly Code Controls Submenu	26
5.3.2	Assembly Code Right Mouse Pop-up Menu	28
5.4	CPU Registers Windows	28
5.4.1	Register Controls Submenu	28
5.4.2	Register Window Right Mouse Pop-up Menu	29
5.5	Timing Breakpoint Window	29
5.5.1	Timing Measurement Specifications	29
5.5.2	Timing Breakpoint Window Controls Submenu	30
5.5.3	Timing Breakpoint Window Right Mouse Pop-up Menu	30
5.6	Calling Tree Windows	30
5.6.1	Calling Tree Window Controls Submenu	30
5.6.2	Calling Tree Window Right Mouse Pop-up Menu	31
5.7	File Editor Windows	31
5.7.1	Enabling the File Editor	31
5.7.2	Mouse Operation on File Editor Windows	32
5.7.3	File Editor Window Controls Submenu	33
5.7.4	Edit Submenu	34
5.7.5	File Editor Window Right Mouse Pop-up Menu	35
5.8	Symbol Table Windows	35
5.8.1	Symbol Table Window Controls Submenu	36
5.8.2	Color Coding on Symbol Table Window	36
6	Breakpoint Operation	36
6.1	Breakpoint Modes	37
6.2	Breakpoint Toolbar	37
6.3	Breakpoint Types	37
6.4	Breakpoint Clock	38
6.5	Program Faults	38
6.6	Single Stepping	38
6.7	Stopping an Executing Program	39
6.8	Breakpoint Implementation Notes	39
7	Task Control	40
7.1	Control Buttons	40
7.2	Implementation Notes	41
8	Configuring for Multiple Users or Projects	41
9	Probe Embedded Program	42
9.1	Probe Command Line Options	42
9.2	Probe Default Operation	43

10	Supplementary Tools	43
10.1	timetag.exe	43
10.1.1	Time Tagging After PKLITE Use	43
10.2	depends.exe	44
11	Trouble Shooting	44
11.1	Console Error Messages at PROBE Start	44
11.2	Target CPU Crashing	44
11.3	Debug@Chip Program Crashing	45

1 Introduction

The Debug@Chip debugger is a Windows PC program which monitors and controls multiple programs within a SC12 target CPU. This is accomplished by executing a program, PROBE.EXE, on the SC12. This embedded probe communicates with the DEBUG@CHIP.EXE executing on a Windows PC via either a TCP/IP socket or one of the two SC12 RS-232 ports.

1.1 Intended Users

Debug@Chip is useful both as a debugger used by software developers during initial program development and as a system monitor used by system engineers during integration testing, field engineering, etc.

The breakpoint control, low level assembly code inspection and code navigation capabilities will be of use to the software developers. Once loaded¹ with relevant memory locations with assistance from the software designers, the real-time data Watch windows can be used by the system engineers to study the internal operation of their new system.

1.2 Requirements from Environment

The debugger's PROBE program requires SC12 @Chip-RTOS version 1.01 or newer. (For older @Chip-RTOS version, an error message "Int21 ax=0x5000 not supported!" will appear at the console.)

The Debug@Chip Debugger system requires the following resources:

1.2.1 Windows PC

Operating System: Win32, z.B. Microsoft Windows 95, 98, NT, 2000, XP or newer.

CPU: Pentium 133 MHz or better.

Disk Space: 10 Mbyte

RAM Space: At least 64 M Byte working RAM (recommended). Amount of RAM used by Debug@Chip depends on number of windows opened and size and number of target programs being monitored.

1.2.2 SC12 Resources Required

Flash Disk Space: 12.5 Kbyte to store PROBE.EXE program. (Program is compressed with the PKZIP tool.)

RAM Space: Around 28 Kbyte RAM used during probe execution

Communication Port: Either one of the two RS-232 ports, or a TCP/IP socket offered by the SC12 BIOS API. Port selection is controlled from the PROBE.EXE command line.

¹ Debug@Chip saves the contents of Watch windows for use in subsequent sessions.

RTOS Resources: One task for PROBE.EXE itself (single threaded program). No other RTOS resources are consumed by the probe. During **Start** command operations an extra task named X is momentarily created to launch target programs.

1.3 Target Programs Under Test in SC12

At a raw assembly code level, any program executing on the SC12 can be monitored. The breakpoint function works only on code residing in RAM. Source level debugging is supported for C code generated by Borland compilers version 3.0 or version 5.02. C++ code generated by version 5.02 compiler is supported with some limitations².

1.4 Functionality Overview

The Debug@Chip provides the following core functionality.

1.4.1 Borland Symbol Table Usage

The symbol tables (i.e. "debug information") generated by either the Borland 3.0 or 5.02 compilers are used by the Debug@Chip to:

- Obtain source line number information to map between target code and source files.
- Allow symbolic entry of memory locations to be watched, including data structures. Data type information is retrieved from the symbol table.

Within the Borland 5.02 IDE the generation of symbol table information in the executables is enabled as follows. First the compiler must be told to generate debug information to be included in the OBJ intermediate files. This is done from the Borland IDE's `Options | Project` dialog shown below in Figure 1. Assure here that the "**Line Numbers**" and "**Generate debug information**" checkboxes are selected as shown. The command line equivalents to these checkboxes are the `-v` option for "Generate debug information" and `-y` option for "Line Numbers".

² Examples of C++ limitations: Overloaded names are not selectable on entry. The tool simply takes the first match encountered in the symbol table.

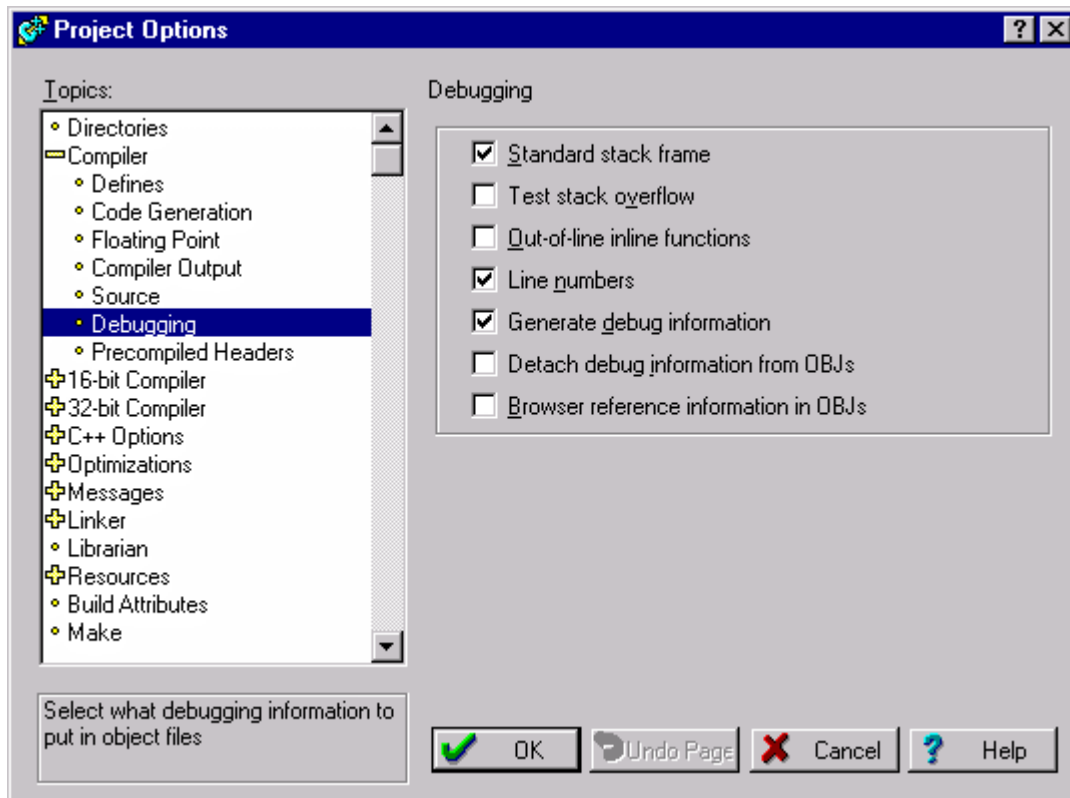


Figure 1) Enabling Generation of Debug Information

The final step required then is to instruct the linker to pass this debug information along to the resulting executable file. This is done by checking the Linker option **'Include debug information'** shown in the following dialog. The equivalent linker command line option for "Include debug information" is `-v`.

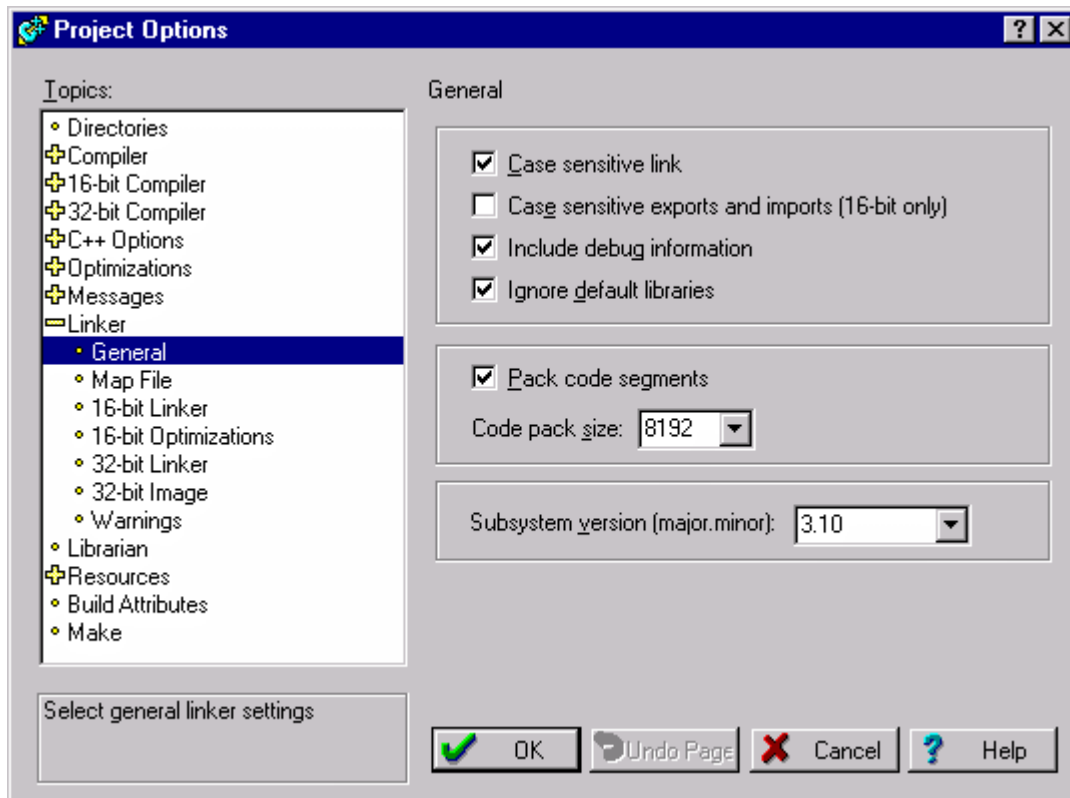


Figure 2) Enabling Linker Debug Information

Now with this debug information, the Debug@Chip tool is aware of your program's variables names and locations. Breakpoints can be set at source level.

Note: The resulting debug information can occupy a lot of disk space. **This information serves no purpose loaded into the SC12 and is therefore a waste of flash disk space.** The use of the `tdstrip` utility supplied by Borland is recommended, which can create a second copy of the executable file with the debug information removed for download to the SC12 target.

1.4.2 Real-time Data Watch

Static locations in memory can be observed during program execution. Memory space locations can be specified either symbolically or by address. I/O space locations can only be entered numerically by address. All locations not in ROM can be written.

Limitations:

Maximum Update rate: around 20 Hz. This decreases with volume of data on windows and number of Watch windows.

Maximum Data per Watch Window: A limit of 2000 bytes of data collected from the SC12 target system is placed on each Watch window display.

Data Sampling Integrity: In theory at least, brief garbage values can appear for long values due to probe executing at a higher priority interrupting lower priority application programs in the middle of a long value update.

Symbolic Data: Display of multi-dimension arrays appear as only two dimensions (row, column) on display. All arrays of data structures appear as a single dimension (row, column folded into single index).

Dereferencing arrays of pointers is not supported directly.

1.4.3 Breakpoint Control

Breakpoints can be set in code residing in RAM. They may be set at source level or assembly. The breakpoint functionality has four modes of operation:

Full Stop - The program thread that hits this type breakpoint is suspended in a `RTX_Sleep_Time()` loop until the user presses **RUN**. Registers (except for SS and SP) and local variables can be modified during the stop.

Fly-By - When “fly-by” breakpoints are hit, the CPU registers and up to 100 bytes of local variables on the stack are dumped into buffers within the probe program for report back to the Debug@Chip monitor. This allows inspection of local variables without stopping the program. If the reporting buffers are busy (due to a recent breakpoint) then only a breakpoint counter (visible on the Debug@Chip Registers window) is incremented. The overhead for the breakpoint code (register/locals dump action) is on the order of 100 us.

Fly-By One-Shot - This is a variant of the Fly-By type breakpoint. After encountering the breakpoint trap a single time, the breakpoint is automatically removed. This form of breakpoint is necessary for inspecting high frequency loops for which a repeated 100 us overhead for the breakpoint dump would be to great.

Timing - This mode of breakpoint measures elapsed time between two points in the program marked by fly-by breakpoints A and B. Resulting measurements for Maximum, Minimum, Average and Pass count are displayed on the Debug@Chip Timing window.

Refer to section 6 on page 36 for more details about breakpoint control.

Limitations:

Limit on number of breakpoints:

100 full stop breakpoints

Two Fly-By breakpoints (shared with timing function)

One Fly-By One-Shot breakpoint

Reporting Limitations:

Only one full stop breakpoint occurrence is reported at a time. If more program threads hit a full stop breakpoint after some other thread is already in a full stop breakpoint, then these additional threads are held in a stop loop FIFO not visible

from the Debug@Chip console. Their reports will become visible one by one (first come first serve order) as previous breakpoints are exited with the **RUN** button.

During the brief moment of time required for the probe to report the register contents to the Debug@Chip monitor, no Fly-By register dumps are possible. In these collision cases, the Fly-By breakpoint will simply increment a breakpoint counter and returns control to the program without waiting.

Breakpoint Setting Limitations:

- Breakpoints cannot be set in ROM based code.
- Breakpoints cannot be set within the first 8 bytes of start-up code in programs compressed by PKZIP or similar compression tools.
- Source level breakpoints set in include files will affect only one instance of the code generated for the respective include file. No “one source line to many code instances” mapping is provided by the Debug@Chip.
- Full stop breakpoints **cannot be set within RTOS timer callback functions**³, or any other locations within the RTOS Kernel task.
- Breakpoints **cannot be set inside Interrupt Service Routines** prior to where the interrupt controller is issued the "End of Interrupt" (EOI) signal. (This is due to the system not responding to any further interrupts after the breakpoint stop is entered.)
- Source level stepping is not supported for C++ “inline” code specified in header files. Only assembly level is available for such code.

Timing Limitations:

Breakpoint timing of periods exceeding 1 ms requires interrupts to be enabled. The timing measurement overhead is 123 us. This amount is subtracted out from values displayed on the Debug@Chip Timing window.

1.4.4 Single Step

From full stop breakpoints the CPU can be single stepped. Single stepping is supported in the following modes:

Single Instruction - One assembly instruction per step.

Source Level Step - Step up to next source line for which compiler provided line number information in the symbol table.

For either of the above forms of single step, the following step variants are available in addition to simply single stepping:

Step Over - Subroutine calls are bypassed without stopping.

Step Out of - Execute up past the next RET or IRET to return from a subroutine to calling the procedure.

³ Timer callback functions execute within the RTOS Kernel task. Stopping the Kernel task prevents further task switching and the entire system dies, including the debugger's PROBE.

Limitations:

Single step will not work across instructions which modify the stack segment register SS. No report will result.

It is not possible to single step through IRET or POPF instructions, unless the trace flag was set in the FLAG word retrieved from the stack.

The “Step Out” implementation analyzes the program stack at breakpoints to determine a routine’s return address . This analysis will not work properly within the first few lines of compiler generated register saving assembly within `interrupt` type routine. If you first step up to the first source line of `interrupt` routines before using the “Step Out”, it should then work properly.

1.4.5 Task Control

Limited control over the tasks executing on the SC12 is provided by the Task Control window. This window can be reached with the CPU | Task Control menu option or with the F12 key. See section 7 on page 40.

The **Start** button on the toolbar or the CPU | Start Program menu option can be used to command the target system to start an application, or to issue some other command.

1.5 Window Types

The user can open arbitrary numbers of the following types of windows, which are presented in a Windows’ Multiple Document Interface (MDI) fashion.

1.5.1 Watch Windows

Watch windows contain a set of memory locations whose contents are reported for display by the embedded probe. These windows show real-time data during the execution of software. It is not necessary to stop programs to observe static data.

The layout of these windows is automatically saved on exit from the Debug@Chip program for use during a subsequent Debug@Chip sessions.

Limitations:

Watch window’s can gather data from the target system for up to 45 items. Two contiguous items (target address wise) of the same object size are automatically gathered together and collected from the target system as an array, requiring a single data request “item”. The maximum volume of data collected from the target system for an individual Watch window is 2000 bytes. Items listed on the Watch window exceeding these limits will appear with a blanked data field.

1.5.2 Assembly Code Windows

These windows display the disassembled code from either the target system or from an executable file (*.EXE or *.ROM). No target system is required for viewing code from an executable file.

1.5.3 File Editor Windows

Any ASCII text file can be viewed and edited. For files recognized as source files for *.EXE programs under test, breakpoints can be controlled at source level. The File Editor windows update automatically when files are edited by another editor, unless you have edited from within the Debug@Chip File Editor window without saving to disk.

1.5.4 Assembly/Source Mix View

For source files with line number information in the symbol table, a File Editor window can be switched to Mix mode. Here you can see the assembly code generated by the compiler, which can often show why your C statement is not giving the result you wanted. No target system connection is required for this Mix view.

Limitation: For include files containing code, the Mix window will display only the first instance found of the resulting code.

1.5.5 Registers Window

The CPU registers reported at breakpoints are viewed here.

1.5.6 Local Variables Window

This window is similar to the Watch windows, except that it is loaded automatically with local variables in scope at breakpoints.

1.5.7 Calling Tree Window

This window shows the return addresses (expressed symbolically) found on the program stack at full stop breakpoints.

1.5.8 Timing Window

This window displays the breakpoint timing results.

1.5.9 Symbol Table Window

This window list a link map obtained from the *.EXE symbol table. All static objects within a program are listed here in either alphabetic or numeric order.

1.6 Additional Functionality

1.6.1 Code Navigation

The Source file windows and Assembly Code window share a code navigation stack which allows you to mouse click through a program's source files and code. With the right mouse button a menu option allows displaying the function whose name is at the window's caret.

1.6.2 Line Assembler

From the Assembly Code windows, you can modify RAM based code in the target system by editing the assembly code. Editing floating-point emulator code (INT 0x34 through 0x3E) is not supported.

1.6.3 Program Download

Programs or any other file can be transferred to the target system flash disk.

1.6.4 IDE Builds

An “Integrated Development Environment” type build option (not functional under Window 95) bundles the program make and download process together. Source files can be opened to where compiler/assembly errors are reported with a mouse click in manner expected from IDE's. Project Makefile dependencies can be updated automatically.

1.6.5 Help System

On-line context sensitive help is included with the program.

2 Getting Started

2.1 Installing Debug@Chip Program on PC

The Debug@Chip program consists of an executable file *Debug@Chip.exe* and a set of Dynamic Link Libraries (DLL's). This set of executables is distributed in a single self installing compressed bundle, itself named *Debug@Chip.exe*. The Debug@Chip program can be installed by executing this installation program. You will be given a prompt for which directory on your PC to install the set of files.

2.1.1 Removing Debug@Chip Program from PC

The Debug@Chip program can later be removed from your PC by simply deleting the directory into which you had installed the program. All files and subdirectories placed on your PC by the installation process went into this single directory.

2.2 SC12 Target Configuration

The Debug@Chip debugger system requires a *probe.exe* program to be executing on the SC12 target system. The *probe.exe* program is delivered with the Debug@Chip installation. It can be found in the *Debug@Chip/bin* directory. Transfer this file to your SC12 target system. You may want to place a statement in your SC12 *autoexec.bat* file to invoke the *probe.exe* if you will be using the debugger regularly.

Refer to section 9 on page 42 for command line arguments used to configure the communication port used by *probe.exe*.

2.3 Debug@Chip Program Configuration

The Debug@Chip program first needs some configuration information from you before it can do much for you. Most important are the symbol tables for the programs under test. The Debug@Chip Configuration dialog pages are reached from the main menu under **File | Configuration**. An alternative way to invoke this dialog is by double clicking on the CPU

status window at the top left side of the Debug@Chip window, directly under the main menu. The Debug@Chip program configuration consists of four tabbed *property* pages:

- Target Programs
- Comm
- Save/Restore
- Options

Each of these configuration pages is discussed below.

2.3.1 Target Programs

On this dialog sheet you specify the set of programs under test. This information is saved in a file named *Debug@Chip.cfg* in the current working directory.

Executables for Programs to be Monitored ... containing symbol tables

In this list you specify each program which will be under test. These will be an **.exe* or **.rom* file⁴ complete with a Borland symbol table. (Otherwise there is little point in listing the program here, since the entire Debug@Chip set of displays rely heavily on these symbol tables.) Use the following five buttons to the right of this list box to manage the contents:

Add Program - Use this button to add another program to the end of the list. A file selection dialog box will appear.

Edit Program - Use this button to change the selected program's executable file.

Drop Program - Use this button to remove the selected program from the list.

Move Up List - Press this button to move the selected program under the highlight up one notch in the ordered list.

Move Down List - Press this button to move the selected program under the highlight down one notch in the ordered list.

The order of the programs in this list plays a role when symbols are entered by name on the Watch or Assembly Code windows and these symbols are present in more than one program (e.g. compiler supplied utilities such as *strcpy*). The symbol search order is first the "Focus program", and then from the top of this program list to the bottom. The "Focus Program" is the program you leave the highlight over when you exit this dialog.

The remaining five fields on the **Target Programs** dialog specify support files and a command line for the selected (highlighted) program in the list of executables. These entries are optional.

Master Build Batch File (for overall program)

This is an optional entry. Specify here a batch file that can be invoked to build your program. This batch file will be called when the IDE build operation is requested. It should

⁴ For **.rom* files, a corresponding **.loc* file produced by the Paradigm locator must be available in the same directory in order for the Debug@Chip program to determine the final location of objects.

be constructed such that it emits the text “**Build Successful**” on successful builds so that the Debug@Chip's results parser can detect a successful build.

The file name entered here serves a second purpose of informing the Debug@Chip tool where the program's build directory is at. This may be necessary if the program's executable with symbol table does not reside in the original build directory⁵. You may need to enter here the name of some file from your program's build directory in order for the Debug@Chip to be able to locate the program's source files. The paths to source files specified in the Borland symbol table are stated relative to the build directory. Consequently the Debug@Chip tool needs to know where the program's build directory was in order to locate the source files. Correct access to the source files can be tested on the **Project Source Files** dialog reached with the **Open** button on the main toolbar.

Makefile(s)

This is an optional entry. Specify here your program's set of makefiles. A drop down list box holds your previous entries for easy re-selection. The currently selected makefile will be accessible through the **File | Build Target Program** dialog's **Show Make File** button for convenience. Makefile names no longer desired can be deleted from the drop down list box by first selecting them and then pressing the delete key.

Compressed Executable for Download to Target ... without symbol table (optional)

Specify here the executable file which is actually to be loaded into the target. Usually this will at least have the symbol table stripped away⁶ and possibly be compressed with a PKLITE tool. The download operation performed automatically after a successful IDE build will transfer this file to the target.

Target File Path and Name (optional)

This field allows you to specify a directory and file name in the target system where the program resides. On downloads, the file from the PC will be renamed if necessary to the name specified here. By default it is assumed that the program resides in the target system's root directory and that it has the same name as the executable on the PC. You may leave this field blank if this is the case. This field is also used to recognize program launch announcements from the target system.

Command to Start Program in Target (optional)

This optional field allows you to specify a command line used for this program with the Start button. This command line may include arguments and should include the full path to the target executable if this executable does not reside in the target system's root directory. By default, the command line used for the Start operation will be simply the target file name.

⁵ The current working directory during program compilation and linking.

⁶ Borland's *tdstrip* utility can be used to remove the symbol table. However, you should preserve the original executable that includes the symbol table while performing this step by creating a new file (perhaps in different directory) containing only the executable code and file header.

2.3.1.1 Correlation of Target Programs

The Debug@Chip must determine where in memory a program resides in order to operate. As the target system launches programs, it notifies the PROBE resident in the target (assuming that this required monitor has been launched). The PROBE in turn sends a program launch report to the Debug@Chip on the PC. This launch report contains the command line used to launch the new application program and the location (first paragraph) of the new application program in memory.

This reported command line is used by the Debug@Chip to correlate launch reports to configured target programs. The following prioritized sequence of logic is used for this correlation. The search is stopped at the first method that comes up with a match. Any blank fields in the Target Programs Configuration are skipped.

- 1) If the command line reported matches (case insensitive) to the command line you have entered on the Target Programs Configuration *Command to Start Program in Target* field, then that program is matched to the report. This field takes precedence, which allows multiple instances of a single EXE to be monitored provided that they can be distinguished by command line parameters.
- 2) Next, the EXE file name (first argument of command line) is compared to the Target Programs Configuration *Target Programs Path and Name* field. If no exact match with full path is found, then any match of just the file name is accepted.
- 3) Next the EXE file name without path is compared to the Target Programs Configuration *Compressed Executable for Download to Target* field.
- 4) Finally, the EXE file name without path is compared to the file names in the Target Programs Configuration list of EXE files with symbol tables.

2.3.2 Comm

On this dialog sheet the form of communication with the target is specified. The information specified here must match the PROBE program configuration (see section 9 on page 42). Either RS-232 or TCP/IP ports may be used to communicate with the target system.

The **Set to Defaults** button can be used to restore settings to the PROBE's default values.

2.3.3 Save/Restore

This dialog is not important for initial setup. Here you can alter the default file names used for saving the Debug@Chip desktop and restoring the desktop on subsequent invocations of the Debug@Chip program. The "desktop" is the set of windows you have placed in the Debug@Chip multiple document interface, including Watch windows with variables listed.

On closing the Debug@Chip program, you will usually receive a prompt asking you if you want the desktop saved. This prompt allows you to control whether or not the current desktop is saved in the desktop file.

2.3.4 Options

Here is where the optional features are controlled. The **Build** button option allows you to run IDE type builds from within the Debug@Chip environment. This combines the *build* and *download to target* system steps.

Note: The Build operation does not work properly under Window 95 or Windows 98 operating systems.

Other options:

Translate Floating Point Emulator INT Opcodes

The floating point emulators use 11 software interrupts, numbers 0x34 through 0x3E. When this option is selected, the disassembler will display the equivalent floating point instructions. In order to single step through software emulator code, this option must be selected as the single step mechanism relies upon the disassembler. If your program uses these software interrupts for other purposes than for floating point emulation then you should deselect this option.

Target Name

This name will appear on the Debug@Chip program's title bar, and on the Window's program selection bar. This can make switching between instances of Debug@Chip easier when monitoring multiple target systems using multiple instances of Debug@Chip. (Hint: Use a unique working directory for each target's Debug@Chip session.)

3 Debug@Chip Screen

3.1 Status Line

The status line is immediately under the program's menu bar. This line is subdivided into the following three windows.

3.1.1 CPU Status Window

This window is in the upper left corner of the display. The state of the target SC12 and problems communicating with the target are indicated here. When breakpoints are set, you will see momentary (hopefully momentary, unless the breakpoint state machine gets stuck due to target communication problems) messages here indicating the action that the tool's breakpoint manager is taking. So long as they disappear, you may ignore them. If they persists, then a problem setting the breakpoints has been encountered.

3.1.2 Clock / Stop Watch Window

In the middle of the status line, a small clock window will appear provided that the Debug@Chip program window is made wide enough. By mouse clicking on this window⁷

⁷ No keyboard alternative is provided here, a mouse is required for operating this timer.

you can start either a timer up count or down count. A single click starts the up count. A double click prompts for a specified down count period.

During timer operation (window green), a single mouse click will freeze the timer display (window blue) resolved to tenth seconds. The internal timer is still counting during these display freezes. Another single mouse click exits the display freeze mode and resumes the live count display.

An exit from timer mode can be made by double clicking to get the down count period prompt and entering a null time period.

3.1.3 Message Window

The window at the right side of the status line displays a message that is specific to the current MDI window that has the input focus. Status messages for time consuming internal operations will also appear here.

3.2 Main Toolbar

The buttons on the Debug@Chip main toolbar provide the following functions:



- Opens the window selection dialog box where you can change the input focus to windows which may be buried in the stack.



- Calls up the Timing window where the execution times between breakpoints A and B are displayed.

Map - Opens the Symbol Table window.

Open - Provides the **Project Source Files** dialog from which you can open one of the current project's source files in a File Editor window.

Src - Sets the focus to a File Editor window. When pressed while an Assembly Code window has the input focus, the source file which contains the code on the Assembly Code window is opened provided that line number information is available for the code in question.

Mix - Exposes the assembly code on the File Editor windows for source files with line number information in the symbol table. The assembly and source are intermixed within the same window. Note that code may appear out of address order within these views.

Asm - Calls up an Assembly Code window. When pressed while a source file with line number information has the input focus, the corresponding raw assembly code will be displayed. Code patches are possible from here.

Reg - Opens a Register window, displaying the CPU registers at a particular breakpoint.



- Opens a real-time data Watch window.

Start - Execute a command in target system. This dialog may be used to start target programs.

Run - Continues program execution after a full stop breakpoint. If this button is clicked when it is already pressed in (indicating that no task is currently in a full stop breakpoint) then the Task Control window is opened. From the Task Control window you can stop a running task with the **single step** button on this dialog. Note that the Debug@Chip does not offer any mechanism to launch programs. Target

programs must be started in the normal manner (console command line, autoexec.bat, etc.).



- Single steps from a full stop breakpoint, stepping into each subroutine call.



- Single steps, stepping over subroutine calls.



- Steps out of the current subroutine, up to the return address found on the program stack.



- Used at a full stop breakpoint to continue execution up to a location marked by the caret on either an Assembly Code or source file window. The window with the input focus is the one which applies.

... One of the following two buttons appear depending on **File | Configuration Options** selected

Build - Opens the IDE project build dialog.

Load - Opens the file selection dialog to select a file to be downloaded to target system.

4 Main Menu Operation

Here is an explanation of the options offered by the main menu. The **Controls** submenu has a different content for each class of window, so these submenu's are discussed with each window's explanation in the subsequent section.

4.1 File Submenu

The File submenu offers the following options:

File New ... - This submenu contains the following selections:

Watch Data - Creates a new initially empty Watch window.

Source File - Creates a new text file. You receive a prompt to name this new file.

Assembly Code from Object File - Creates a new Assembly Code window based on the executable file for the current focus program. You must first have set the Debug@Chip configuration so that at least one executable file with symbol table is specified.

Assembly Code in Target - Creates a new Assembly Code window based on code read out of the target system.

CPU Registers - Creates a new Registers window.

CPU Timing - Opens the breakpoint Timing display window which displays execution times between fly-by breakpoints A and B.

Graph - Creates a new Graph window. Data items from Watch windows can be selected for display on Graph window verses time.

Program - Opens file selection dialog for adding an executable file (with symbol table) to the Debug@Chip configuration's list of programs under test.

File Open ... - This submenu contains the following selections:

- Watch Data** - Prompts for *.w86 Watch window file. Watch windows can be saved⁸ in files and later restored to screen with this menu option.
- Top Most Source Window** - Brings top most File Editor window to top of desktop. This may be useful when your source file windows get buried under the other windows.
- Source File** - Opens the **Project Source Files** dialog where you can select a source file from the current project (target program). Use the **Other Files** button here to open arbitrary text files outside the scope of the current project. The **Select Program** button allows you to switch context to another project.
- Assembly Code from Object File** - Opens an Assembly Code window based on the executable file for the current focus program. You must first have set the Debug@Chip configuration so that at least one executable file with symbol table is listed.
- Assembly Code in Target** - Opens an Assembly Code window based on code read out of the target system.
- CPU Registers** - Opens or sets focus to the Registers window.
- CPU Timing** - Opens or sets focus to the breakpoint Timing display window.
- Graph** - Opens or sets focus to a Graph window (data verses time).
- Symbol Table** - Opens or sets focus to the Symbol Table listing window for current focus program.
- Close** - Closes current window.
- Save** - Saves current window to disk. This control is disabled if current window is not of type which can be saved.
- Save As** - Provides file dialog which allows you to provide the file name where contents of current window will be saved. This control is disabled if current window does not save.
- Save All Editor Files** - Writes contents of any edited files within File Editor windows to disk. Note: This operation is performed automatically prior to a **Build** operation.
- Restore Destroyed** - This option “pops” the most recently destroyed Watch window back to the screen. This can be useful when you accidentally delete a Watch window.
- Desktop Save/Restore** - This option allows you to either save the current desktop or restore one previously saved from a file on disk. Desktop's can be moved between PC's in this manner.
- Record** - The data flowing to Watch windows from the target system can be recorded on disk for later playback. This control allows you to specify a file name where the recorded data is stored and the recording interval. The recording process can later be stopped by invoking this same menu option at a later time after the recording process has been started.
- Playback** - This option allows you to playback a recorded Watch window. These playback windows will appear in gray.
- Print to File ... Start New File / Append File** - The contents of some windows can be output to an ASCII file. (Note: A direct interface to printers is not supported by Debug@Chip program.)

⁸ The set of data items to be monitored is what is saved here, not the actual contents of these locations. See the **File | Record** menu option for recording the contents of locations.

- Download File to Target** - This option allows any selected file to be transferred to target system.
- Symbol Tables** - The submenu offers the following options:
- Select Program** - Here you can select which program from your configuration will be used as the “focus program”. This program is where symbols are searched for first.
- List By Symbols** - Opens a Symbol Table window with contents presented in alphabetical order.
- List By Address** - Opens a Symbol Table window with contents presented in address order.
- Files In Use** - Provides a message box which lists the file dates of executable files in use. An indication of whether or not the file matches that found in the target system is also given here. The “Time Stamp” refers to the tag placed in executables by the **timetag** program distributed with the Debug@Chip. Use of this tool is recommended to automate the determination of whether the symbol table in use matches with code in the target system. This is especially important when setting breakpoints!
- Build Target Program** - This calls up the control interface for running project builds.
- Configuration** - Here is where you must configure the Debug@Chip program for use. In particular, you need to specify the executable files containing the symbol tables for the target programs.
- Exit** - Closes Debug@Chip session.

4.2 Breakpoints Submenu

The Breakpoints submenu offers the following options:

- Set Breakpoint** - This control is enabled when either an Assembly Code window or File Editor window has the input focus. For File Editor windows, the file in view must be a source file for which line number information is available in the associated symbol table. This control sets or removes (toggle) a breakpoint at the current caret position, providing an alternative to double clicking the mouse in the left margins of these windows.
- List Breakpoints** - This option provides a complete summary list of all breakpoints currently set. The list will show parenthesis around breakpoints which are set in programs not currently loaded in the target system. Breakpoints cannot be set through this dialog box, however they can be removed.
- Clear All Breakpoints** - This control erases all breakpoints that have been set within all programs.
- Code at Breakpoint** - This option displays the code at the place where the most recent breakpoint occurred. The source file will be shown provided line number information, otherwise an Assembly Code window is used.
- Code at Fault** - This option is disabled until a program fault (e.g. invalid opcode) is reported by the target system. It displays the code at the place where the fault interrupt was invoked.
- Calling Tree on Stack** - This option is only enabled during a stop in either a full stop breakpoint or after a fault report. This control opens a window that displays the calling tree determined by reading the return addresses found on the program stack.

4.3 CPU Submenu

The CPU submenu offers the following options:


Task Control - Opens the Task Control window. See section 7 on page 40.


Start Program - Same as **Start** button. Provides dialog for command to be issued in target system.


Reset Target - This commands the target system to reset.


Comm Reset - This resets the PC's communication channel, either the TCP/IP socket in use or RS-232 port. This can sometimes restore communication with the target system. This can also be used to terminate some command such as a memory write which may keep causing the target system to crash. A cycling condition can sometimes be reached where after each reboot, the troublesome memory write is attempted again resulting in another reboot. The **Comm Reset** can be used to break this cycle.

Run - This control is used to continue program execution after a full stop breakpoint. The functionality is the same as the **Run** button on the main toolbar.

Step Over - This control single steps the program such that subroutine calls are bypassed. This functions the same as the  button on the main toolbar.

Step Into - This control provides the most detailed single step by stepping into each subroutine call. This functions the same as the  button on the main toolbar.

Step Out Of - This control is used at a full stop breakpoint to execute up to where the current subroutine was called from. This functions the same as the  button on the main toolbar.

Execute Up to Caret - This control is used at a full stop breakpoint to execute up to the line you mark on either an Assembly Code window or source file. This functions the same as the  button on the main toolbar.

Single Step Mode - Here you are given a dialog box where you can control aspects of the single step operation. Normally the single stepping is done either at source level or single instruction level based on the type of window that currently has the input focus. Here you can override this default to force either source or assembly level stepping. The manner in which the *step out of* and *step over* operations are implemented is also controlled here.

Registers - Opens or sets focus to the Registers window.

Timing Window - Opens or sets focus to the breakpoint Timing window, which displays execution times between breakpoints A and B.

Fault Message - After a program fault is reported by the target system, this option displays a message stating the cause and location of the fault.

Program Time Tags - Provides message box which lists the dates and time tags of each of the programs under test. The time tags are those set by the **timetag** tool⁹. Of particular importance here is that the time tags reported from the target system match those found in the executable files used by Debug@Chip. *Otherwise all symbolic information is suspect!*

⁹ See section 10.1 on page 43 for more information about *timetag* program.

Select Program - This dialog box allows you to select the *focus program* from among the set of target programs under test you have configured the Debug@Chip to monitor. Use the **Configuration** button here to add additional target programs to the list.

4.4 Grep

The **Grep** menu option provides a dialog box from which you can invoke a Unix style grep operation. All occurrences of a specified string within a set of ASCII files will be located for you. The set of files can be all files belonging to the currently active project and/or a specified directory. When specifying a search directory, file wild card notation such as *.c may be used to narrow the search.

Refer to the on-line help for further information on usage of the **Grep** function.

5 Window Operation


5.1 Overview

The following classes of data windows are available:

- **Watch** ... either static data or local variables (two types of Watch window)
- **Assembly Code**
- **CPU Registers** at breakpoints
- **Timing Breakpoint**
- **Calling Tree** at breakpoints
- **File Editor** Windows
- **Symbol Table** Listing
- **Graph** windows display selected data item values verses time.

On the main menu a **Controls** submenu appears. The set of controls listed on this submenu depends on the class of window from the above list which currently has the input focus (e.g. on top). A useful subset of these controls is also available from a pop-up menu called up with the right mouse button. The pop-up may also contain additional controls to those on the main menu.

5.2 Watch Windows

Watch windows are accessed with the  button on the toolbar. You can organize your target data locations to be monitored using an arbitrary number of Watch windows. The first 10 Watch windows created will show a number preceding their name that appears in the window title. The **Alt-1** through **Alt-0** key combinations can be used to set the input focus to these primary Watch windows.

Watch window update rates are independently adjustable. Minimized windows do not update, so they place no load on the target data channel. The updating cycles round-robin between the enabled Watch windows.

Each data item on the Watch windows can be independently formatted. For example you may specify hexadecimal display mode for one group of variables and decimal for others.

When either Watch windows are restored to the screen or when a new symbol table comes into use, the locations of all symbolically entered data items in the Watch window are resolved with the current symbol table.

CAUTION: Type information is not checked again once a data item is placed on a Watch window. If the program is modified such that data types or data structure members change for items placed on a Watch window, you must *manually* delete the data structure or item from the Watch window and re-enter it in order to pick up the new type information from the current symbol table.

NOTE: When variables belonging to programs which have not been loaded into target memory are displayed on Watch windows, their data field display will remain blanked until the respective program begins execution in the target.

5.2.1 Watch Controls Submenu

Here is an explanation for each item on the Watch window's **Controls** menu. Refer to the actual menus for the short cut keys indicated to the right of the menu items which have keys assigned. (E.g. `Insert` key is short cut for **Insert Item** menu option.)

Insert Item – Provides a dialog box for adding a data item to the Watch window. The new item will be placed on the screen above the current highlighted data item. Locations can be in either the CPU's memory or I/O space. Only memory locations can be specified symbolically. I/O ports must be entered by port address.

... The following two menu items provide options on how newly inserted items are formatted. They both operate as toggle switches with a check mark appearing beside the menu item when the respective item format mode is selected.

Use Symbol Table Data Structure Info – When selected, the information found in the symbol table will be used to load the screen with each member of data structures when you add a data structure to the Watch window. For example, in some rare cases you may want to view a block of memory where a data structure lies as just an array of bytes. So this option is provided here to disable the usage of the symbol table. When symbol table is not used, each symbol defaults to simply an integer. You can then use the **Format** and/or **Array** menu options to display the memory area as you desire.

Hex Default - When selected, newly inserted data items will default to hexadecimal display format. You are free to change this default with an additional action after the item is on the screen, but this switch can eliminate this extra step each time when there are a series of data items you want formatted a particular way.

Delete Item – Removes highlighted data item from the Watch window.

... The following block of menu options apply to the entire Watch window.

- Show Linear Address** - This toggle switch affects how item addresses are formatted. The default is segmented addresses. Addresses of data items appear in the message window at the top right of the Debug@Chip frame window. Also the display of data items formatted as pointers will be affected by this display mode switch.
- Freeze Display** - This toggle switch stops the display updates. The window appears gray when the display freeze is in affect.
- Update Rate** - This option provides a dialog box where you can control the rate and manner that the current Watch window will be updated. Refer to the on-line help offered by the Help button on this dialog for more details.
- Screen Name** - You can give the Watch window a meaningful title here.
- Show Full Names** - This toggle switch affects the name display for isolated data structure members. When the option is selected the entire full name of a data structure member will appear on the screen. When deselected only the final member name will be shown on the screen.
- Show Member Names** - This toggle switch either displays or hides member names on data structure displays.
- Clear Focus** - This control terminates the data Focus mode initiated with the **Focus** control described below.
- Clear Continuous Write** - This control terminates any continuous writes initiated with the **Write** control described below.
- Evaluate Ptrs** - This toggle switch affects the message presented in the upper right message window of the Debug@Chip frame window for the data item under the highlight. It only affects the message for data items formatted as pointers (near or far). When this menu option is selected, a pointer's contents will be evaluated symbolically using the symbol table to display by name the object referenced by the pointer. When deselected the address of the pointer's own location is instead displayed.
- Record Selected** - This toggle switch is used to mark a Watch window for data recording. When selected, a green box will appear under the spinning wheel in the lower right corner of the Watch window. Then when recording is activated with the **File | Record** menu option, the data from all Watch windows marked in this manner will be recorded in the specified recording file which can later be played back. During the recording process, the green box changes to red.

... The following block of menu options apply to the specific data item under the highlight.

- Format** - This option provides a dialog box where you can specify an items display format. Refer to on-line help available on the **Help** button for more details.
- Array** - Here you can specify up to a two dimensional array.
- Write** - This option allows you to write to the data item under the highlight. By selecting the **Continuous Write** check box the write to the location will automatically be repeated at a low frequency until the **Clear Continuous Write** option listed above is used to stop the repetitive writes. A maximum of 10 locations can be placed in the continuous write mode (sum for all Watch windows).

Dereference Pointer - This control applies to pointer data items. It will generate another data item on the Watch window which displays the object pointed to by the pointer. The location referenced by the pointer will be tracked dynamically.

Focus - This control adds the item under the highlight to the list of data items under focus. All other data items on the current Watch window are eliminated from the update cycle, unless they were previously selected with this same control. The purpose of this control is to allow a faster update rate for a small set of data items. Normally, you will want to minimize any other Watch windows when using this control so that your current Watch window will have a dedicated data channel to the target system. Use the **Clear Focus** control listed above to exit this restricted update mode and resume display of all data items.

Graph Data - The item under the highlight will be added to the most recent Graph window. Graph windows display a data's value verses time.

5.2.2 Watch Window Right Mouse Pop-up Menu

Most of the options on the Watch window's pop-up menu are the same as the menu options on the Controls menu explained above. The unique controls here are:

Hexadecimal Format - This is a toggle switch to flip the data item under the highlight to/from hexadecimal display mode.

5.3 Assembly Code Windows

The Assembly Code windows display disassembled code obtained from either the target system or from a *.exe (or *.rom) executable file.

Note: Assembly Code windows for target code hide the INT 3 opcodes set for breakpoints inserted into programs by substituting the original opcodes for display¹⁰. Under some circumstances, the data base of breakpoints maintained by the Debug@Chip can get out of synch with the target system and you will see INT 3 opcodes appearing on the target Assembly Code windows, but not in the corresponding executable file Assembly Code window. If this occurs, it is recommended that the target system be reset.

5.3.1 Assembly Code Controls Submenu

Here is an explanation for each item on the Assembly Code window's **Controls** menu. Refer to the actual menus for the short cut keys indicated to the right of some menu items. (E.g. N key is short cut for **Call New Page** menu item.)

Watch Location / Go to Location - This menu item will appear in one of two forms depending on whether the target location at the cursor (white arrow) is a data item or a subroutine. Data items will be appended to the Watch window that most recently

¹⁰ If you inspect the opcodes through the Watch windows (which do not perform this substitution) you will see the actual INT 3 opcodes.

had the input focus. For subroutines this Assembly Code window will display the subroutines disassembled code.

Call New Page - This menu item prompts for an address or subroutine name to open a new Assembly Code window at. Any target of the highlighted instruction on the current window will be the default location preset in the prompt edit control.

Go To - This menu option works the same as the previous **Call New Page** option except that it transfers the current Assembly Code window to the new address instead of creating another window.

Show Program - This menu option allows you to specify one of your other programs to be viewed in a new Assembly Code window.

Show Linear Address - This control is a toggle switch. A check mark appears next to the menu text when the option is selected. The address field on all Assembly Code windows will switch from segmented addresses to linear addresses when this option is selected.

Hexadecimal Numerics - This control is a toggle switch with a check mark present when this display mode is selected. The arguments of the assembly code instructions will be displayed in hexadecimal when selected and in decimal when deselected¹¹.

Unsigned Numerics - This control is a toggle switch with a check mark present when this display mode is selected. The arguments of the assembly code instructions will be interpreted and displayed as unsigned values when selected and signed values when deselected.

Edit Code - This provides access to the line assembler. The code at the highlight can be edited provided that the program is located in the target system. Otherwise this control is disabled. Refer to on-line help for more information about the line assembler and patch log operation.

Previous Patch - A log of program patches made by the line assembler are held to make restoring original code easier. This menu option scans to a patch at a lower addresses when one exists, otherwise the control is disabled.

Next Patch - This control scans down to a program patch at a higher address when one exists. Otherwise the control is disabled.

Refresh Display - This control forces a refresh of the code buffer used to support the Assembly Code window. This is meaningful when viewing code in the actual target system, in case for some reason the code has changed since originally sampled and cached in the Debug@Chip PC.

Properties - This menu option provides a message box which lists a few details about the current executable. If the window is displaying code from the target system, then the message details apply to the program located at the displayed addresses.

¹¹ This hexadecimal mode control and the following Unsigned Numerics control also affect the disassembly shown in the File Editor Mix windows. However the Mix windows will not update in response to these controls. The display mode in effect when the Mix window was generated will remain. You can force a display mode change on the Mix windows by closing the Mix window and re-opening the source file after setting these Assembly Code window controls to the desired state.

5.3.2 Assembly Code Right Mouse Pop-up Menu

Most of the options on the Assembly Code window's pop-up menu are the same as the menu options on the Controls menu explained above. The unique controls here are:

Return to ... - This is a part of the code navigation implementation shared with the File Editor windows. It operates as a stack where for each *call* you make navigating through the source and assembly code there is a corresponding **Return to** option provided by this control. This control pops one address off of this navigation stack and restores the screen to where you were before the *call*.

Show Code In Target / Show Assembly Code in *.EXE - One of these two options will be presented depending on whether the current window is showing live code from the target system or code from a **.exe* (or **.rom*). For the live code window this option offers the corresponding executable file for viewing. For an executable file Assembly Code window, the corresponding live target code is offered provided that the program has been located in the target.

Show Source Code in *.C - This option calls up the source code which corresponds to the current Assembly Code window. The option is disabled if no line number information is available in the symbol table for the highlighted line of assembly code.

Show Program - This option allows you to call up an Assembly Code window for another program.

5.4 CPU Registers Windows

The Registers windows display the contents of the CPU registers at breakpoints or faults. All data is displayed in hexadecimal format, except for the contents of the gray box in the lower right corner of the Registers window where the value in data registers is repeated in decimal format. For the Flags register, the value of each of the CPU flags is stated here. The **Format Registers** menu control allows the format of this gray window field to be adjusted. At full stop breakpoints the contents of the registers can be edited.

The four letter task name displayed on the top line identifies the program thread which hit the breakpoint or fault.

The windows also display a count of times that each of the breakpoints has been reached. The counts are labeled **A**, **B**, **1**, and **S**. Count **A** indicates the number of occurrences of fly-by breakpoint A. **B** indicates fly-by breakpoint B occurrences, and **1** indicates fly-by breakpoint 1 occurrences. The count marked **S** indicates the number of full stop breakpoint occurrences.

5.4.1 Register Controls Submenu

Here is an explanation for each item on the Register window's **Controls** menu.

Modify Register - This option allows the contents of the highlighted register to be edited, with some restrictions. The target system must be in a full stop breakpoint in order

to edit registers. The SS and SP registers cannot be modified. Refer to the on-line help for data entry format.

Format Register - This menu option allows the format of data register values displayed in the gray box in the lower right corner of the registers window to be adjusted. Signed or unsigned decimal displays with a scale factor and binary point may be specified. Pointer format can be selected, in which case the object pointed to by the register will be displayed based on the contents of the symbol table.

Show Local Variables - This menu option opens a Watch window containing the local variables at the current breakpoint. This control is disabled if the symbol table reports no local variables at the current instruction pointer, or if execution has been resumed with the **Run** command.

... The next set of controls select which breakpoint's set of registers are displayed. Each registers reported at each category of breakpoint are buffered independently.

Full Stop Breakpoint Registers - This menu option sets the current Register window to display register contents at most recent Full Stop breakpoint.

Breakpoint A Registers - This menu option sets the current Register window to display register contents at most recently reported¹² breakpoint A.

Breakpoint B Registers - This option displays registers from most recently reported breakpoint B.

Breakpoint 1 Registers - This option displays registers from most recently reported breakpoint 1

Fault Breakpoint Registers - This option displays registers from most recently reported program fault (e.g. "Invalid Opcode").

5.4.2 Register Window Right Mouse Pop-up Menu

The Register window's pop-up menu provides the same set of controls listed above available from the main menu.

5.5 Timing Breakpoint Window


The Timing Breakpoint window displays the elapsed times between breakpoint A and breakpoint B when these breakpoints are operated in *timing* mode¹³.

5.5.1 Timing Measurement Specifications

The timing measurements displayed here have the following characteristics.

Clock Source: Quartz crystal based TIMER #2 within the SC12.

¹² If fly-by breakpoints are occurring at too high a rate to report registers and stack contents back to the PC, then only the breakpoint counter is incremented and the register/stack dump is bypassed. This breakpoint counter appears on the Register window.

¹³ Timing mode is activated by pressing in the clock button, , that appears on the Source File or Assembly Code windows. If you have no mouse available, this mode can also be activated by selecting the Timing mode check box on the **List Breakpoints** dialog reached with the Ctrl-F2 keys.

Range: 18.2 Hours (2**32 ms)

Resolution: 0.2 us

Measurement Accuracy: 1 us or Quartz clock drift, which ever is greater. (This accuracy stated excludes any bias due to measurement overhead or interval measurement bias. See below)

Measurement Overhead: approximately 150 us

The measurement overhead is the amount of CPU time required to perform each measurement, which covers the time required for breakpoint A execution and breakpoint B execution. A calibration factor¹⁴ of 123 us is subtracted out from the measured values before display to compensate for this overhead. However, if breakpoint B is not reached for each occurrence of breakpoint A then a positive bias of about 50 us will appear in the measurements for each excess execution of breakpoint A.

Interval Measurement Bias: approximately 273 us

Execution rate measurements can be made by placing breakpoint B immediately before breakpoint A. However in this case the display calibration has an undesired affect. It biases the interval measurements by minus 123 us. Adding the 150 us measurement overhead to this and the total display bias during interval measurements is then about 273 us. This means that the displayed interval values will be about 273 us lower than the actual values.

5.5.2 Timing Breakpoint Window Controls Submenu

Here is an explanation for the items on the Timing Breakpoint window's **Controls** menu.

Reset Statistics - This option clears the timing measurement accumulators.

Write Report to File - This option writes the currently displayed timing measurements to an ASCII listing file of your choice.

5.5.3 Timing Breakpoint Window Right Mouse Pop-up Menu

The Timing Breakpoint window's pop-up menu provides the same set of controls listed above available from the main menu.

5.6 Calling Tree Windows

Use **Breakpoints | Calling Tree on Stack** menu option to open a Calling Tree window.

This menu option is disabled unless the target system is currently in a full stop breakpoint or program fault.

The Calling Tree window lists the return addresses obtained from the program stack at full stop breakpoints or program faults. The deepest call is listed at the top of the window with parent routines listed below.

5.6.1 Calling Tree Window Controls Submenu

Here is an explanation for the items on the Calling Tree window's **Controls** menu. The context of all controls applies to the program instruction pointer under the highlight. The highlight can be moved with either the up/down arrow keys or mouse selection.

¹⁴ The 130 us is the elapsed time prior to the clock read action within the timing breakpoint interrupt. The additional 30 us is spent within the breakpoint interrupt following the clock read.

Show Source Code in *.C - This option calls up the File Editor window to show the source code corresponding to the highlighted instruction pointer. The control is disabled if the symbol table provides no line number information for this instruction pointer.

Show Local Variables - This option opens a Watch window loaded with the local variables in scope at the highlighted instruction pointer. Note that display of register variables (e.g. SI, DI) is not supported for stack frames up the stack.

Show Assembly Code in Target - This option opens an Assembly Code window to show the disassembled code read out of the target system at the highlighted instruction pointer.

Show Assembly Code in *.EXE - This option opens an Assembly Code window to show the disassembled code read out of the executable file at the highlighted instruction pointer. This option is disabled if the instruction pointer does not map to a known program located in the target.

5.6.2 Calling Tree Window Right Mouse Pop-up Menu

The Calling Tree window's pop-up menu provides the same set of controls listed above available from the main menu under **Controls**.

5.7 File Editor Windows

The File Editor windows can be opened for source files from the current project with the **Open** button on the main toolbar. Any ASCII file can be opened by using the **Other Files** button on the **Project Source Files** dialog.

A breakpoint control toolbar (see section 6.2 on page 37) will appear in the upper right corner of File Editor windows when line number information is present in the symbol table for the respective source file. This toolbar is used to select the type of breakpoint to be set by a subsequent double click in the left margin. The **Mix** button on the main toolbar applies to these File Editor windows with line number information. In *mix* display mode, the disassembled code from the associated executable file will be displayed below the applicable source lines. Note that the code presented in this fashion can be out of address order¹⁵.

For source files with line number information available in the symbol table, a "**Modified**" indication will appear in the window's title bar if the source file's date is newer than the corresponding executable file. In such cases, *the MIX display is usually inaccurate!* Remember that accurate source level control over breakpoints requires a valid map of the source files by the line number information.

5.7.1 Enabling the File Editor

By default the File Editor's editing functions are disabled to protect against unintended changes to the source files. Editing can be enabled with the **Edit | Edit Enable** menu option.

The editor provided here is a very crude editor, lacking syntax highlighting, auto formatting and many other features which one *must have* these days. Consequently it is intended to be

¹⁵ Out of order code is common around *for* or *while* loops.

used only in conjunction with your favorite editor. The debugger's editor can be used for quick simple editing, and then you can switch over to your favorite editor for more significant editing. This debugger editor will automatically reload updated files when they appear on disk¹⁶. Assuming that your favorite editor can also be configured to automatically reload files that appear updated on disk, then the transition between these two editors will be comfortable.

Mix Window Limitations:

The Mix windows are provided to show the relationship between source code and the resulting assembly code generated by the compiler. Source editing is not implemented on the File Editor's Mix windows. You must return to the pure source view in order to edit files.

Any editing you do on source file's with this debugger's editor will *not* be reflected in the Mix windows. The source file contents of the Mix windows are frozen to the original source file contents present at the time the window's disassembly was performed. However if you edit with an outside editor, the Mix window will then show the new text due to the debugger reloading the source file from disk and creating a new Mix image. But be aware that the line number information from the program's symbol table most likely no longer matches the source file in these cases! Take note of the "**Modified**" warnings in the File Editor's title bar. Usually this indicates that assembly code mapping to source lines is not reliable.

After a program is recompiled and linked, the debugger will automatically accept the program's new executable code when one of following conditions is satisfied:

- 1) No communication with a target system available.
- 2) Target system has not reported the loading and execution of the program in question¹⁷.
- 3) Target system reports loading of this new program with matching time tag.

When the debugger accepts the new executable code, any affected Mix windows will be regenerated based on the new line number information and the *Modified* indication should then disappear.

5.7.2 Mouse Operation on File Editor Windows

The mouse can be used to mark of blocks of text, set breakpoints and transfer data objects to a Watch window. These operations are done as follows.

No Button Pressed - Selects item under cursor for the small Quick Watch data window.

Left Mouse Click - Moves edit caret to mouse cursor.

Left Mouse Down and Drag - Marks block of text.

¹⁶ Unless you have been editing without saving the file to disk. In this case you will be asked what you want to do.

¹⁷ The debugger avoids using a new executable found on disk if that program is currently executing on the target system. This allows the previous symbol table to be used further until you have loaded the new program into the target system.

Left Mouse Double Click - Over text it highlights word under cursor, over left margin it sets a breakpoint at next line for which line number information is available in symbol table.

Ctrl Key + Left Mouse Double Click - Sets breakpoint at cursor, same as double click left mouse in left margin.

Right Mouse Click - Provides pop-up menu in context of cursor position.

Ctrl Key + Right Mouse Down and Drag - Drags and drops data object under cursor out to a Watch window or code window. Cursor will change to a box shape if object is static and found in symbol table.

Middle Mouse Button Down and Drag - Drags and drops data object under cursor, same as “Ctrl Key + Right Mouse Down”.

In case you do not have a mouse available, the above mouse operations can also be accomplished with either the **Edit** submenu off the main menu (block marking) or the **Breakpoints | Set Breakpoint** menu control. The breakpoint context toolbar within the File Editor window can be selected with the **Breakpoint | List Breakpoints** dialog, if necessary.

5.7.2.1 Quick Watch

The mouse can be used on the File Editor window to point to simple data items to view their current values in a momentary pop-up window. This feature allows a quick look at data without cluttering your Watch windows intended for more persistent displays. This feature does not support display of data structures or arrays of objects other than characters. These larger objects must be placed on a normal Watch window for viewing.

Automatic local variables are not displayable unless you bring them into scope by setting a breakpoint at an appropriate place. Note that for Fly-By breakpoints there is a limit of 100 bytes reported for local variables on the stack. Full stop breakpoints do not have this limitation.

5.7.3 File Editor Window Controls Submenu

Here is an explanation for the items on the File Editor window's **Controls** menu.

Watch Object at Caret - This menu option transfers a static data item under the entry caret to a Watch window. The option is disabled when the caret is not over a static data item. By default, the data structure element referenced by the text at caret is selected for display on the Watch window. If you instead want the entire data structure displayed, then you must select (double click with left mouse or drag left mouse) the isolated data structure name prior to applying this menu option. The data item will be appended to the Watch window which most recently had the input focus.

Call to Routine At Caret - This menu option opens a new File Editor or Assembly window to show the routine at the caret. The source file will appear in another File Editor window provided line number info is available in the symbol table. Otherwise an Assembly Code window will open at the routine's code. The menu option operates within the *Code Navigation* framework provided for File Editor and Assembly Code windows. The current routine's display is pushed onto a stack to allow you to

return to this window after reading the *called* routine's code. The return is made with the **Return to** menu option below.

Go to Routine At Caret - This menu option works the same as the **Call to Routine At Caret** option except that the current window will switch content to display the routine instead of opening a new window (call).

Return to ... - This menu option pops the stack maintained for the *Code Navigation* function and returns you to a code display window (File Editor or Assembly Code) you previously had before a **Call/Go to** operation.

Open Include File - When the caret is over an “`#include somefile.h`” statement, this menu option will attempt to locate this include file and open it in a File Editor window.

Note: For Borland 3.0 compiler the paths to the include files is not reported in the symbol table. Consequently this function will not work for include files not in the same directory as the source files when using Borland 3.0 symbol tables.

Show Local Variables - This menu option opens a Watch window with the current routine's local variables loaded for display. This option is only enabled when local variables are within scope at the caret position and a breakpoint, either full stop or fly-by type, has reported these locals. Note that for fly-by breakpoints the size of the local variables area reported on the stack is limited to 100 bytes. For full stop breakpoints this limit does not apply.

Quick Watch Hexadecimal - This is a toggle switch on which a check mark appears when it is selected. It controls the display mode of the *quick watch* pop-up window, switching it between decimal and hexadecimal.

Source Only - This control applies to source files with line number information available. It switches the display mode to hide the assembly code mix.

Mix Source/Code - This control applies to source files with line number information available. It switches the display mode to mix the assembly code in with the source code to allow detailed code reviews.

Properties - This option displays a message box which list some details about the current source file size, date and amount of code generated.

5.7.4 Edit Submenu

The Edit submenu off the main menu for File Editor windows has the following options:

Edit Enable - This control works as a toggle switch to globally enable or disable file editing. You may want to disable editing to protect against unintended source file changes during debugging.

Undo - This option cancels the most recent edit action, restoring the file to the state it was before that action.

Block Begin - This is the first step in marking off a block of text in the fashion the good old Borland DOS editors used to work. For larger blocks of text it can be a useful alternative to the mouse marking method.

Block End - This marks the tail of a block of text being marked with the **Block Begin** option.

Block Move - This option moves the highlighted block of text to the current caret position.

Block Copy - This option puts a copy of the highlighted block of text at the current caret position. It *does not* place a copy into the Windows clipboard.

- Block Delete** - This option deletes the highlighted block of text.
- Block Hide** – This option toggles on/off a highlighted block selected with the above **Block Begin / Block End** controls.
- Delete Line(s)** – This option deletes the line at the caret. When the Ctrl+L accelerator is used, up to nine lines can be deleted at a time by pressing a number key with the Ctrl key down prior to the L key.
- Put Line(s)** - This option inserts at the current caret position the lines previously deleted with the **Delete Line(s)** option.
- Cut** - This option deletes the highlighted text and places it in the Windows clipboard.
- Copy** - This option copies the highlighted text into the Windows clipboard.
- Paste** - This option inserts at the current caret position text from the Windows clipboard.
- Clear** - This option deletes the highlighted text without affecting the Windows clipboard.
- Options** - This dialog provides control over the use of tabs and auto indenting. Refer to online help for more details.
- Create *.BAK Backup Files** - This option is a toggle switch with a check mark indicating that it is selected. When selected the original file will be saved with a **.bak* extension prior to saving a file after editing.

5.7.5 File Editor Window Right Mouse Pop-up Menu

The File Editor window's pop-up menu provides the same set of controls listed above available from the main menu under **Controls** plus some from off the main menu's **Edit** submenu.

5.8 Symbol Table Windows

The Symbol Table windows can be brought up with the **Map** button on the main toolbar or the **File | Symbol Tables** submenu. These windows list the static data and functions for a given program based on the Borland symbol table contents. Use the **Windows | New Window** menu control to open more than a single window if desired.

The Symbol Table windows list the static contents of a program's symbol table in either alphabetic or numerical (by linear address) order. The address displayed will show a question mark, '?', after either the segment portion of the address or after the linear address when the location of the program in the target system has not been resolved.

Positioning Window within Symbol List

The position of the window within the symbol list can be selected by typing in a symbols name. The characters typed in appear in the window's title bar and the window is scrolled to the best fit to what you have typed in.

Selecting Items for Display

Press enter to select the highlighted symbol for display. Doing so will transfer data items to the most recently used Watch window or open a File Editor or Assembly Code window to a routine's code.

5.8.1 Symbol Table Window Controls Submenu

Here is an explanation for the items on the Symbol Table window's **Controls** menu.

Watch Object ... or Go to Routine ... - This exact text here depends on the symbol under the highlight. Data objects are transferred over to a Watch window or the code for routines is presented.

Select Program - Allows you to switch over to another program's symbol table.

List Alphabetically - This option presents the symbols in alphabetic order.

List Numerically - This option presents the symbols ordered by linear address.

Note: This presentation can be particularly useful when you have some piece of data that is mysteriously changing for no reason you know of. In such cases, review the handling of data objects which lie immediately before it in memory for possible exceeding array bounds.

Display Linear Address - This option is a toggle switch which switches the address display mode between segmented and linear addresses.

Display Filter - This option allows you to filter out certain categories of symbols from the display.

... A check mark will appear next to the following four display filter toggle switches when selected.

Show Functions - Toggle switch alters the display filter to either display or hide functions from the list.

Show Global Data - Toggle switch alters the display filter to either display or hide global external data from the list.

Show Static Data - Toggle switch alters the display filter to either display or hide local static data from the list.

Show Compiler/Linker Symbols - Toggle switch alters the display filter to either display or hide extra symbols for which no type information is available. These commonly originate from compiler provided libraries or from the linker.

Symbol Table Properties - This option displays a message with information about the symbol table.

5.8.2 Color Coding on Symbol Table Window

The symbols listed on the Map windows are color coded as follows:

White - Routines

Yellow - Data

Red - Objects for which debug information is not available.

6 Breakpoint Operation

The F2 key can be used to set breakpoints on either the Assembly windows or the source file View/Edit windows. This control works as an on/off toggle. A double click of the left mouse button in the left margin of these windows also toggles the breakpoint on/off.


6.1 Breakpoint Modes

The breakpoints operate in two different modes: Full Stop and Fly-By. When a program thread reaches a Full Stop breakpoint it is suspended in a wait loop until you resume execution with either the **Run** button or single step operations. In Fly-By mode, the thread will resume immediately after a brief moment required to dump registers and stack local variables into buffers which are subsequently reported to the Debug@Chip program by the *probe.exe* program. Two of the Fly-By breakpoints (designated A and B) can be operated in a timing mode instead of register/stack dump mode.

6.2 Breakpoint Toolbar

On source file windows for which line number information is available and on Assembly Code windows, a toolbar will appear similar to this:






This toolbar is used to select the mode in which subsequent breakpoints will be set. The toolbar illustrated above has the **Full Stop** breakpoint mode selected. Any breakpoints subsequently set will be **Full Stop** type breakpoints. Alternatively you could click the mouse on the  button so the tool bar appears as:



Now any breakpoint setting activity will set the **Fly-By A** breakpoint. Marks will appear in the right margin of File Editor or Assembly code windows to indicate breakpoints. These markers will appear in gray if the respective program is not yet located in the target system. When the program begins execution in the target system the breakpoints will be set¹⁸ and the indicators will show in color.

6.3 Breakpoint Types

-  - **Fly-By A** breakpoint performs repeated registers/stack dumps in normal mode and starts the timing clock in timing mode.
-  - **Fly-By B** breakpoint performs repeated register/stack dumps in normal mode and stops the timing clock in timing mode.
-  - **Fly-By 1** breakpoint performs a single register/stack dump and then automatically disarms itself. Use of this breakpoint is recommended in sections of your program which execute at such a high rate that the repetitive breakpoints would be too great a load on the system under test.

¹⁸ For programs compressed with PKZIP, breakpoints cannot be set within the first 8 bytes of the program's startup code.

--- Note: Each of the above Fly-By breakpoints are unique. Only one of each exists.



- **Full Stop** breakpoints. The Debug@Chip supports up to 100 full stop breakpoints. The **Breakpoints | List Breakpoints** dialog (Ctrl+F2) can be used to view the total set of breakpoints.

6.4 Breakpoint Clock

The breakpoints can be used to time software between two points in the programs. A clock button is available to switch the breakpoint operation between register dumping mode (normal) and breakpoint timing mode.



- Clock off, breakpoint register dumping mode is selected. The contents of the CPU registers dumped at the breakpoints can be viewed on the Register page reached with the Ctrl+F5 key.



- Clock on, breakpoint timing mode is selected. The resulting execution times can be viewed on the CPU Timing page.

6.5 Program Faults

The PROBE performs a breakpoint style register dump when unexpected program faults (exceptions) occur. The debugger traps the following 80186 fault interrupts:

- INT 0 - Divide Overflow
- INT 6 - Invalid Opcode





Pressing the **Run** button at a fault will command the program to continue, which will send it into the fault handler that was installed at the time the PROBE program began execution. Usually this will be the SC12 BIOS's fault handlers which print out an error message to the console.

Note that the instruction pointer indication at an invalid opcode fault references the opcode byte following where the actual invalid opcode was at.

6.6 Single Stepping

From full stop breakpoints, the program can be single stepped at either assembly level or source level. By default, the level is determined by the type of window which has the input focus. If you have selected a source file window, then source level stepping is performed. If you have the input focus set to either an Assembly window or a Mix code window, then assembly level stepping is done. You can override this default behavior with the **CPU | Single Step Mode** menu option.

Four step control buttons are provided on the main toolbar:

Step Into , Step Over , Step Out of , and Execute Up to Caret .

The last three of these controls have two different implementations from which you can choose with the **CPU | Single Step Mode** menu option's "Step Over/Step Out of Method" control. The default implementation momentarily replaces existing opcodes in the program with a breakpoint interrupt 3 opcode to execute up to a designated "over", "out" or "up to" point. An alternate implementation uses the CPU's single step trace flag and trace interrupt 1 to sequence through the program one instruction¹⁹ at a time until the designated instruction pointer is reached. These methods have the following advantages and disadvantages.

Set INT 3 - This method allows real-time execution speed up to the designated "over", "out" or "up to" point in the program. This method only works for code residing in RAM, since the INT 3 opcode must be inserted.

INT 1 Trace - This method allows tracing into ROM code²⁰. This method can be very slow when thousands of instructions must execute prior to reaching the designated "over", "out" or "up to" point. A "**Single Stepping**" message appears in the CPU status window during this process. A resulting instruction count is displayed on the Register window's top line when the end point is finally reached.

6.7 Stopping an Executing Program

An executing program can be put into single step mode by first selecting the program's task in the Task Control window's (F12 key) task list and then pressing the Single Step button beside the task list.

6.8 Breakpoint Implementation Notes

The Debug@Chip maintains a data base of full stop breakpoints. The original opcode that was replaced by an INT 3 opcode is recorded in this data base at the time a breakpoint is set. It will be needed when the breakpoint is reached to execute through the breakpoint.

If you open a Debug@Chip session on a target system which is already stopped at a breakpoint, the Debug@Chip program has a problem. It does not know the original opcode that your target had at this address. Consequently the tool is going to ask you to reset the target.

Also note the implications here for target programs with self modifying code. Depending on when the code modification is performed, the breakpoint function may or may not work properly.

¹⁹ Critical sections within the operating system are bypassed atomically, to respect the critical section. The code sequence: "pushf, push CS, CLI" used by the RTX multi-tasking OS around its critical sections is recognized by the PROBE program.

²⁰ When you attempt to use the **Set INT 3** stepping mode for an address in ROM, you will be given a message offering temporary use of the **INT 1 Trace** method instead since the **Set INT 3** stepping is not possible here. You can avoid repeated occurrences of these messages by selecting the **INT 1 Trace** method at the **CPU | Single Step Mode** dialog box.

The fly-by breakpoints use a small data base held on board the target, offering a rapid look-up of the original opcodes needed at the fly-by exception.

Note: When setting breakpoints in programs launched from `autoexec.bat` immediately after PROBE, the H command line parameter will be needed on the PROBE command line. Otherwise breakpoints can be missed due to timing problems. When a program is launched the PROBE program suspends the thread that is launching the new program until the Debug@Chip has had time to set any breakpoints desired in the new program. However by default, this delay action only occurs when communication with the PC has already been established. With the 'H' option, this delay is inserted unconditionally. This gives time for the communication to be established after a target reset.

7 Task Control

The task control window lists the tasks running in the SC12. The primary purpose of this window is to allow you to see where an active program is executing with the `source` or `Assembly` buttons provided here. This can be useful when your application is caught up in a loop somewhere.

The displayed information is based on an inspection of the RTX RTOS's task status made on roughly a one second strobe. The PROBE program attempts to access the user task instruction pointers from off the program's stack, but this method is not always reliable. Tasks which indicate "active" are actually suspended due to interruption (except for the PROBE program which of course was itself executing during this data collection).

The priorities range from 0 to 127 where lower numbers indicate higher priority.

The corresponding executable file known to Debug@Chip through the configuration settings are indicated when available.

7.1 Control Buttons

The following task control buttons apply to the highlighted task from the task list. Note that no control over the essential AMXK task is permitted. The `single step`, `source` and `Assembly` controls are not available for system tasks.

Caution:

Applying the task controls (e.g. Suspend/Resume ... etc.) to the system tasks can result in the SC12 not working properly until the computer is reset.

- Suspend** - Executes RTOS API interrupt 0xAD service 0x0E (RTX_SUSPEND_TASK) for selected task.
- Resume** - Executes RTOS API interrupt 0xAD service 0x0F (RTX_RESUME_TASK) for selected task.
- Chg Priority** - Executes RTOS API interrupt 0xAD service 0x08 (RTX_CHANGE_PRIO) on selected task with your specified priority. The highest priority that can be set here is 3, and lowest is 127.
- Single Step** - Sets trace flag in FLAG word on stack for active tasks or tasks whose status is either "Await Wakeup" or "Timed Await Wakeup". Breakpoint single step mode then begins when the task is resumed. This operation is only available for user tasks. It is not available if RTOS was reached using the dynamic links API.
- Source** - For user tasks whose instruction pointer was located on the stack, this button will open the source file to the line corresponding to the IP address when line numbers are available.
- Assembly** - For user tasks whose instruction pointer was located on the stack, this button opens the Assembly Code window to the IP address.

7.2 Implementation Notes

The `probe` program reads the BIOS RTX RTOS's prioritized linked list of tasks to gather the data displayed in the Task Control window's task list. In order to minimize the impact on the system, this operation is done at the `probe` program's normal task priority and without interrupts masked. This leaves a possibility that either an incomplete list or garbage may appear in the displayed task list in the event that the RTOS's linked list is edited in the midst of the `probe` program's scan. This could happen for example when a new program is started from the higher priority console thread. But this erroneous display will persist only until the next update.

8 Configuring for Multiple Users or Projects

It is recommended that Debug@Chip be opened with the current working directory set to some other directory than the `/bin` directory where the `Debug@Chip.exe` resides. The reason for doing this is to make life easy when you either share the PC with another user or deal with more than a single project. Under Windows NT 4.0, Windows 2000 or Windows XP the current working directory can be set by right mouse clicking on a Debug@Chip program icon (assuming you have placed an icon on the Windows desktop), selecting the **Properties** item on the pop-up menu. The working directory is then specified in the **Start In** field.

If you want a new project or user to be initially configured the same as a previous project or user, then copy the `Debug@Chip.cfg` and `savedesk.t86` (or what ever name you have given to your desktop file) files from the original project/users working directory over to the new project/users working directory. The same procedure applies to moving between PC's.

9 Probe Embedded Program

The Debug@Chip debugger system requires a *probe.exe* program to be executing on the SC12 target system. The *probe.exe* program is delivered with the Debug@Chip installation. It can be found in the *Debug@Chip/bin* directory. For 24 MHz CPU's (non-standard) the *probe24.exe* program should instead be used.

The PROBE program must begin execution prior to any programs under test. This assures that the Debug@Chip will know where the program under test is loaded in target memory. Launching the PROBE from the target system's AUTOEXEC.BAT is one approach to meeting this requirement.

9.1 Probe Command Line Options

The PROBE program supports command line options that override default settings. Each command line option consists of a single letter (case insensitive) usually followed immediately by a numeric value. No white space is permitted between the command line option letter and the numeric value. The command line options are as follows:

- B** - RS-232 BAUD rate specification. Supported BAUD rate values are 1200, 2400, 4800, 9600, 19200 and 38400. Default BAUD rate is 9600.
- H** - Hang thread on program launch until Debug@Chip program in PC has had a chance to set breakpoints in new program. This option takes no numeric argument. It can be used when breakpoints are desired in programs launched from autoexec.bat immediately following PROBE. (By default the program launch thread is only suspended when communication has already been established with PC.) **CAUTION:** *This option can hang up the program launch indefinitely if no communication with Debug@Chip in PC is established.*
- C** - RS-232 Comm Port specification: 0 = EXT Port, 1 =COM Port . (Default RS-232 port is EXT.)
- P** - TCP/IP Port number. The port number after the P is optional. If no value is entered then the default port number 3000 will be used. This can be useful to enable both TCP/IP and RS-232 communication with Debug@Chip. When both communication channels are enabled, the probe attempts communication over the RS-232 port when ever a TCP/IP socket connection is not established.
- T** - Probe Task Priority: By default the probe will execute at priority 15. This is a higher priority than normal programs will execute at. This command line option allows the user to specify some other priority for the Probe program.

Any combination of command line parameters is accepted in any order.

Command Line Examples

probe B19200

... Enables RS-232 communication at 19,200 BAUD and disables the TCP/IP communication.

probe C

... Enables RS-232 communication at default BAUD rate of 9,600 over default EXT port. TCP/IP communication is disabled.

probe C P

... Enables both RS-232 and TCP/IP communication paths with default settings.

probe T24

... Sets PROBE program's priority to 24. Communication will be over the TCP/IP socket at port 3000 (default).

probe C1 B4800 P3001

... Enables RS-232 communication over COM port at 4,800 BAUD. TCP/IP communication remains enabled over port 3001.

9.2 Probe Default Operation

By default the probe communicates with the Debug@Chip over a TCP/IP socket on port 3000.

The probe executes at priority 15 by default. The X task created momentarily to **Start** programs executes at the probe priority plus one, which is 16 for the default case.

10 Supplementary Tools

Some extra command line tools are provided in the Debug@Chip's /bin directory for use during program development.

10.1 *timetag.exe*

The use of this tool is recommended as a final step in your build process. It overwrites file offset 0x1C of the executable file with a four byte time stamp representing the PC's current time, or program build time. This portion of the executable's file header is otherwise unused, so no harm is done.

On program launch by the SC12, this time tag is reported to the PROBE.EXE (embedded portion of the Debug@Chip) which in turn passes this information up to the Debug@Chip. You can then view these time stamps through either the **File | Symbol Tables | Files In Use** or **CPU | Program Time Tags** menu options. During program monitoring it is essential that the symbolic information used by Debug@Chip match the target program. This time stamp automates this verification.

For command line syntax, simply type *timetag* at the DOS prompt and the program will list some usage instructions to the console.

10.1.1 Time Tagging After PKLITE Use

If PKLITE is used to compress a target executable file, the **timetag** tool must be executed twice. Once for the *.exe containing the symbol table used by the Debug@Chip program and again after PKLITE is applied to the *.exe. For example, if your target program is named *target.exe* the set of operations you would perform in your build process following the linker step would be:

```
timetag target.exe -h timetag.h
tdstrip target.exe target1.exe
pklite -o target1.exe load\target.exe
timetag load\target.exe -r timetag.h
```

The first call to **timetag** places the current build time into the *target.exe* executable file header and into the *timetag.h* file that it produces. Then Borland's *tdstrip* is used to remove the symbol table from a second copy of the program. Then this second copy is compressed with the PKLITE tool. Finally the **timetag** tool is executed a second time with the **-r** option to apply the time stamp stored in *timetag.h* to the resulting EXE produced by the PKLITE tool. This second pass is necessary since the PKLITE tool has overwritten the original file header and time stamp.

10.2 depends.exe

This tool is provided to automatically update the dependencies section of project makefiles. It can be invoked directly from the **Build** dialog's **Update Makefile Dependancies** button.

11 Trouble Shooting

11.1 Console Error Messages at PROBE Start

Incompatibilities with the @Chip-RTOS version can result in the following messages output to the console when the PROBE.EXE begins execution.

a) Int21 ax=0x5000 not supported!

This error message can occur when the PROBE is used with older versions of the SC12 @Chip-RTOS. The PROBE requires the DOS software interrupt 0x21 to support a service 0x50 used to hook application programs as they are launched by the RTOS. Version 1.01 of the SC12 @Chip-RTOS or newer is required.

b) Int 0xAC not supported!

By default, the PROBE attempts to establish a TCP/IP socket with the Debug@Chip program in the PC. This error message will be seen if the PROBE is used with a version of the @Chip-RTOS such as TINY which does not include the TCP/IP stack. One of the RS-232 ports will need to be used in this case. This must be specified on the PROBE command line (see section 9.1 on page 42).

11.2 Target CPU Crashing

The Debug@Chip program can cause the target system to crash under certain conditions. For example, if one of the Watch windows references an address in the target at which no hardware responds, the target CPU may lockup due to no memory ready signal being

returned from this location. This normally will result in a CPU reset due to a watch dog timer time-out.

If you suspect this is occurring, the following procedure can be used:

- 1) Minimize all other windows.
- 2) Use the CPU | Comm Reset menu control to clear any memory request currently being issued to the target system.

At this point if the target CPU runs OK, then one of the Debug@Chip windows must have been causing the shutdown. You can re-open the minimized windows one by one to determine which one causes the problem.

If you are still having a problem, then clear all breakpoints with the **Breakpoints | Clear All Breakpoints** menu option and try again.

11.3 Debug@Chip Program Crashing

The Debug@Chip program is very dependant on the data it reads in from the *.*cfg* configuration and *.*t86* desktop files. If the program persistently crashes, try deleting either or both of these files from the working directory. Default settings will then be used by the program for a clean start.

When reporting Debug@Chip program crashing problems, the contents of the Windows error message stating the DLL at fault will help in resolving the problem.