Version 3.X

ZAP

C Source Level Cross Debugger User's Guide

Monitor Configuration

PC/Windows Host

Document Version 3.2 December 1999

Copyright © COSMIC Software Inc 1994, 1996, 1998, 1999

All Trademarks are the property of their respective owners

TOC

Table of Contents

Overview

ZAP Display Windows 1-2

ZAP Debugging Features 1-4

- Non-intrusive Debugging 1-4
- Source Browsing 1-4
- Graphical Performance Analysis 1-4
- C and Assembly Trace 1-5
- Time Line Chronograms 1-5
- Chromacoding 1-5
- Breakpoints 1-5
- Expression Evaluation 1-5
- Single Stepping C and Assembly 1-6

Automated Debugging Sessions 1-6
On-line Help Facility 1-6
Comprehensive Debugger Command Set 1-6

ZAP Configurations 1-7
Simulator Configuration 1-7
Monitor Configuration 1-7
Background Debug Mode Configuration 1-7
In-Circuit Emulator Configuration 1-8

Using ZAP

Starting ZAP 2-2

ZAP Windows 2-3
Source Window 2-3
Toolbar 2-3
Command Window 2-4
Disassembly Window 2-4
Memory Window 2-4
Monitors Window 2-4
Register Window 2-5
Stack Window 2-5
Status Bar 2-5
Variable Window 2-5

Screen Display Options 2-6
Windows Menu 2-6
Setup Menu 2-7

Loading an Application 2-13
File Menu 2-13
Load 2-13
Application Map 2-14
Load and Save Layout 2-14

Load and Save Session 2-14
About ZAP 2-14
Exit 2-14

Utilities Menu 2-15
Configure Tools 2-15

On-line Help Facility 2-17
Help on Using ZAP 2-17
Help on C Library 2-17
Help on C Syntax 2-17

Program Execution

Start and Stop Execution 3-2
Normal Execution 3-2
Stop execution 3-3

Single Stepping 3-4

Reset and Restart 3-7
Reset 3-7
Go from Reset 3-7
Restart 3-7

Events and Breakpoints 3-8
Code Events 3-8
Watchpoint 3-8
Breakpoints 3-8
Setting/Editing Breakpoints 3-9
Deactivating/Activating Breakpoints 3-10
Deleting Breakpoints 3-11
Code Event Editor 3-12
Displaying and Editing Breakpoints 3-15

Activate and Deactivate Functions 3-16

Browser Menu 3-17
Event Browser 3-17
Source Browser 3-17
Memory Browser 3-21
Variable Browser 3-22
Cross Reference Browser 3-23
Symbol List Browser (sorted) 3-24
Symbol Browser 3-25
Map 3-25

Monitoring Application Data

Monitoring Variables and Expressions 4-2
Monitors Window 4-2
Address of Source Lines 4-3

Updating Variables 4-4

Evaluating Expressions 4-6
Evaluate Expression 4-6

Displaying and Updating Memory 4-7
Disassembling Memory 4-7
Displaying Memory 4-7
Updating Memory 4-8
Fill Memory 4-9
Saving a Memory Dump to a file 4-10
Saving Memory Dump as S-Record format. 4-10
Display Highlights 4-10

Evaluating Assembly Symbols 4-11

Displaying and Updating Registers 4-12

Displaying the Stack Frame 4-13

Advanced Topics

Simulated I/O 5-2

ZAP Commands

Command Line Syntax 6-2

Specifying Memory Locations and Registers 6-3
- Constants and Expressions 6-3
- Register Manipulation 6-3
- User defined variables 6-4
- Source files and Functions. 6-4
- Data Objects 6-5
- Pointer Indirection 6-6

Entering ZAP Commands 6-7

Command Descriptions 6-8

ZAP Commands 6-9

Index

CHAPTER

1

Overview

The COSMIC ZAP source level cross debugger is a full featured MS-Windows cross debugging environment. It is designed to provide a powerful yet intuitive Windows interface for efficient cross debugging of embedded applications. 4.0xThis chapter provides an overview of ZAP's main features and a description of the various target configurations available. The following sections are included:

- ZAP Display Windows
- ZAP Debugging Features
- ZAP Target Configurations

© Copyright 1999 by COSMIC Software

ZAP Display Windows

ZAP is a true MS-Windows application providing an infinite combination of display options. You can open and arrange any combination of the following windows. You can even change each window's font and highlight colors.

Source Window

The Source window displays the C or Assembly source code for the active function and maintains the active instruction highlight for the current line of source code.

Disassembly Window

The optional Disassembly window displays a disassembly of the current page of code. The disassembly display is coordinated with the C or Assembly source window to provide simultaneous debugging of C and assembly. With ZAP, you can even set breakpoints in the Disassembly window.

Command Window

The optional Command window gives you the option to use ZAP's robust command language.

Monitors Window

The Monitors window is an optional relocatable window that displays monitored (or watch) expressions and variables. This window allows you to point and click on monitored objects to change their format or update their value.

Register Window

The Register window displays the current values of the CPU registers and displays changes between commands in color.

Stack Window

The Stack window displays the current stack frame including function arguments.

Data Windows

You can open multiple Data windows to display a memory

© Copyright 1999 by COSMIC Software

dump or disassembly anywhere in your memory map.

Variable Window

The Variable window displays the address and value of all variables in the current scope in one of several display formats.

Toolbar

The Toolbar is a relocatable push button window providing easy access to some of the most commonly used debugging commands.

© Copyright 1999 by COSMIC Software

ZAP Debugging Features

ZAP provides many features tailored specifically for the embedded systems developer. The user interface is almost entirely processor and execution configuration independent. If you are debugging code on several processors or different target configurations your debugging skills are entirely portable. The following sections briefly describe the high level user interface common to all target configurations of ZAP.

Non-intrusive Debugging

ZAP does not modify your code in any way. The symbol information *ZAP* uses is produced in separate, transparent sections, which resides on the host. The code you cross debug is that which you intend to execute in your final product, not an intermediate language.You can **PROM** your code or download it to the target environment directly after debugging with ZAP (No recompiling or relinking is required).

Source Browsing

Zap's unique source browser allows you to search and view all of your source code in multiple discrete windows. You can set and edit breakpoints anywhere in your code without changing the source window or the current state of execution. ZAP's powerful browser feature also lets you quickly search and monitor variables and breakpoints.

Graphical Performance Analysis

ZAP's performance analysis feature gives you a graphical representation of code coverage and MCU cycles. Code coverage can be displayed on a file by file, function by function basis. This feature gives you a relative comparison of your codes efficiency, thus allowing you to go back and optimize the code to get the most out of your embedded project. (Available only in simulation version)

© Copyright 1999 by COSMIC Software

C and Assembly Trace

The C and Assembly Trace feature allows you to record and playback any sequence of C or Assembly instructions. You can move backwards and forwards through the recorded trace one instruction at a time or in a continuous playback mode. To help you save time, you can even exclude functions you want to trace over. (Not available in all configurations see the Chapter titled “Advanced Topics” for details)

Time Line Chronograms

The Chronology feature provides a chronogram or graphical time-line of function calls. A proportional bar chart is used to denote entry and exit from a function with relation to the total number of cycles executed. You can display the chronology of function calls. ZAP can even report chronology on interrupt service routines so you can keep track of external events as well as internal function calls. (Not available in all configurations see the Chapter titled “Advanced Topics” for details)

Chromacoding

ZAP provides syntax color coding of C key words, C library functions to make it easier to follow the flow of your program.

Breakpoints

ZAP’s powerful breakpoint facility lets you set an unlimited number of breakpoints in any source window with a simple double click of the mouse. You can attach debugger actions, user commands, and complex expressions to any breakpoint. You can even set complex breakpoints.

Expression Evaluation

ZAP allows you to evaluate and monitor variables and expressions by double clicking on them or selecting them with the mouse.

© Copyright 1999 by COSMIC Software

Single Stepping C and Assembly

ZAP's stepping facility lets you to step through your program at the C and assembly level, step into and over function calls and perform conditional stepping.

Automated Debugging Sessions

You can use *ZAP*'s file redirection and log file management facilities to automate debugging sessions. You can record and play back all or part of a debugging session.

On-line Help Facility

ZAP provides an extensive Windows on-line help facility. Double click on a C keyword, library function in the source window to open a syntax help window. Choose *On C Library* from the help menu to display a list of ANSI C functions and hypertext manual pages.

Comprehensive Debugger Command Set

In addition to the many mouse and menu features, ZAP provides an extensive command language for those who prefer command entry to mouse operations. *ZAP's* comprehensive set of commands allows you to cross debug your code at both the C source and Assembly language levels.

© Copyright 1999 by COSMIC Software

ZAP Configurations

ZAP is available in four target configurations to provide debugging support for all phases of development. All of the *ZAP* debuggers share the same high level Windows interface and command set so there is no additional learning curve as you progress through your development or change to another supported environment. The available configurations include:

Simulator Configuration

This version of *ZAP* integrates a CPU simulator with a full C and Assembly source level debugger to provide a debugging environment which doesn't require any external hardware. This version is useful during the early stages of development when you're trying to debug your algorithms or when hardware is simply not available.

Monitor Configuration

This version of *ZAP* is designed to work with several standard prototype or evaluation boards. *ZAP* uses a small monitor program which is downloaded to the board to create its debugging environment. This provides a low cost hardware debugging solution.

Background Debug Mode Configuration

In recent years, some chips have been equipped with Background Mode debugging support. The Background Debug Mode (BDM) is essentially an operating mode and instruction set which allows access to the chips internal operations and target resources without the need for a monitor. This version of ZAP is interfaced through the parallel port of a PC or workstation to the standard BDM port using the standard Motorola (P&E) cable. You can debug directly on your target board or use a standard evaluation board. This configuration provides a real-time debugging solution with a minimum of setup time and hardware expense.

© Copyright 1999 by COSMIC Software

In-Circuit Emulator Configuration

ZAP has also been interfaced with several full featured in-circuit emulators to provide the optimal debugging environment. This version of *ZAP* is often used in the latter stages of development to fine tune optimizations and track down the hard to find bugs in complex embedded applications.

© Copyright 1999 by COSMIC Software

CHAPTER

2

Using ZAP

This chapter describes the basic invocation and operation of the COSMIC ZAP debugger. This chapter assumes that you have read the Installation Guide and have properly installed the debugger and setup the execution environment. This chapter includes the following sections:

- Starting ZAP
- ZAP Window Displays
- Screen Display Options
- Loading an Application (File Menu)
- Utilities Menu
- On-line Help facility

© Copyright 1999 by COSMIC Software

Starting ZAP

The easiest way to start ZAP is to locate the ZAP icon and double click on it. Alternatively, you can select ZAP from the start menu under Windows 95/NT. This will bring up the main debugger screen as shown below. This screen consists of the ZAP desktop, menu bar button bar and status bar.

Figure 2-1 ZAP Main Window

© Copyright 1999 by COSMIC Software

ZAP Windows

In addition to the main debugger screen, ZAP uses several optional windows to efficiently display and control the debugging process. Each window can be opened, closed, moved and resized using standard Windows 95/NT mouse commands. The optional windows are available under the **Show** menu. Simply select the desired window to open it and select it again to close it.

Source Window

The Source window is the leftmost window in the main display and accepts point and click breakpoints and expression evaluation. The Source window displays the C or Assembly Source code for the active file and function and maintains the active instruction highlight for the current line of source code. Source code is shown with bold line numbers when assembly code is actually produced for the source line and gray line numbers for source lines that didn't produce any code (i.e. #defines, #iifdefs etc.). The Source window also provides color coded syntax and on-line help for C keywords and C library functions. Simply double click on any C keyword, library function to pop-up a syntax help window. You can enable/disable syntax color coding by selecting *Syntax Coloring* from the **Show** menu.

Button Bar

The Button Bar consists of graphical push buttons which duplicates many of ZAP's most heavily used commands. The button Bar can be turned on and off by selecting Button Bar from the view memory.

Toolbar

The Toolbar provides a fast convenient way to access the more frequently used debug commands. The Toolbar option under the **Show** menu controls the orientation of the toolbar dialog box. Choose Toolbar→Horizontal or Vertical to open the toolbar with the desired orientation. The Toolbar is completely relocatable so you can place it anywhere on the screen.

© Copyright 1999 by COSMIC Software

Command Window

The Command window is an optional relocatable window used to enter and display debugger commands and output. See the Chapter titled “ZAP Commands” for details.

Disassembly Window

The optional Disassembly window displays a disassembly of the current (or active) page of code. This display also maintains a highlight on the current assembly instruction and a secondary highlight on the assembly instructions that correspond to the active line of C code or assembly macro. This Disassembly window also accepts point and click breakpoints on instruction addresses.

Memory Window

The memory window is an optional relocatable window that displays a block of memory in one of several formats including disassembly (code), hexadecimal, octal, binary and decimal. You can have the ASCII representation of the memory block displayed alongside the memory dump to help you find and monitor strings at the low level. Memory changes between commands are highlighted so you can easily track memory modifications as your program executes. You can modify memory by clicking on the desired data value and typing a new value. You can also choose to disassemble memory by selecting Code in the Memory Configuration window. You can set breakpoints in the disassembly by double clicking on the assembly address.

Monitors Window

The Monitors window is an optional relocatable window that displays monitored (or watch) expressions and variables. This window allows you to point and click on monitored objects to change their format or update their value.

© Copyright 1999 by COSMIC Software

Register Window

The Register window is an optional relocatable window that displays the current values of the CPU registers and displays changes between commands in color. This display allows you to click on any register name to change its value to an application symbol or double click on it's value directly to change it explicitly from the keyboard.

Stack Window

The Stack window is an optional relocatable window that displays the current stack frame including function arguments. You can double click on a function name in the stack frame to open up a source browser window.

Status Bar

The optional status window is a small grey bar located below the source window This window displays the current status of the system. (i.e. running, stepping etc.)

Variable Window

The Variable window is an optional relocatable window that displays the address and value of all variables in the current scope in one of several display formats.

© Copyright 1999 by COSMIC Software

Screen Display Options

ZAP allows you to customize the screen layout, text fonts and colored highlights of the various screen displays. All of these attributes can be changed using the **Setup and Windows** Menu. To save changes to window layout fonts and color highlights choose *Save Config On Exit* from the **Setup** menu.

Windows Menu

ZAP uses the standard Window 95/NT display options Free, Cascade, Horizontal Tile and vertical Tile. These options are found under the **Windows** menu. The WIndows menu also maintains a history list of all open windows including those minimized. Choose an open window from the history list under the **Windows** menu to bring it to the foreground and make it active.

Free Allows you to place and size all windows by hand with new windows opening with the default size and on top of other windows.

Cascade Displays all windows cascaded from top left to bottom right with new windows cascaded on top as the last window.

Horizontal Tile - Displays all windows in a wider horizontal size. Each window is proportionally sized to fill the entire main the window without overlapping. New windows are added from the top left corner and push the other windows down and then up to the next column to keep the display proportional.

Vertical Tile - Displays all windows in a taller vertical size. Each window is proportionally sized to fill the entire main window without overlapping. New windows are added from the top left corner and push the other windows to the right and then down to the next row to keep the display proportional.

© Copyright 1999 by COSMIC Software

Setup Menu

Load Option (Code and Symbols)

This option allows you to choose whether to load the code portion of the image. In some case, typically when a hardware version is used, you may load the code portion via another method such as a serial programmer. In this case, you can choose to load only the symbols through ZAP and debug the matching code which is already available in the target system. The default is to load both the code and symbols.

NOTE

In all cases symbols are not loaded to the target system. All symbols are loaded and kept on the host. If you plan to load only the symbols it is required that the code image and the symbol image are created from the same executable (i.e. linker output). If they do not match the behavior is undefined.

Colors

Each color option item listed under the options menu opens a Windows color pallet for selecting the desired color. Simply click on the desired color and click OK to change the color.

Events

Changes the highlight color for active and suspended events. This highlight appears on the C line number in the source window when a breakpoint is active or suspended. The highlight also appears in source and event browser windows.

Disassembly

Changes the color of the disassembly highlight in the disassembly window. This highlights the assembly instructions that correspond to the current line of C code.

Instruction

Changes the color of the highlight for the current

© Copyright 1999 by COSMIC Software

assembly instruction or program counter in the disassembly window.

Memory

Changes the color of modified data in the Memory window.

Source

Changes the color of the highlight for the current line of C code in the source window.

Search

Changes the color of the highlight for lines found in a search. the current line of C code in the source window.

Syntax Coloring

ZAP provides syntax color coding of C key words, C library functions and kernel objects to make it easier to follow the flow of your program. If you prefer you can change the color used for a particular object or disable color coding altogether. To enable/disable all syntax coloring choose syntax coloring from the view menu. To change a particular syntax color choose the syntax type that you want to change from the Syntax coloring submenu.

C Comments

C Key words

Library Functions

Mnemonics - (Assembler mnemonics)

Fonts

The font option allows you to change all of the fonts to your own taste or selectively change the fonts for the different windows. Each selection will open up a Windows font dialog box which allows you to choose any available font.

All Changes the default font for all the windows.

© Copyright 1999 by COSMIC Software

Browser	Changes fonts for all the browser windows.
Commands	Changes Command window fonts.
Memory	Changes the fonts for the data or memory dump window.
Disassembly	Changes the font for the disassembly window.
Monitors	Changes the font for the Monitor window.
Registers	Changes the Command window fonts.
Source	Changes the font for the C source code window.
Stack	Changes the font for the Stack display window.

Key Binding (Keyboard Short-cuts)

ZAP's key binding facility allows you to attach or bind short-cut keys ZAP commands and tasks. Simply select click on the function key by itself or in combination with the <Shift> and <Control> keys and then choose a ZAP function from the binding pull down list and then click OK or on another function key to save. Key bindings are saved in the ZAP initialization file (.ini) when the configuration is saved. This mechanism is in addition to the standard Windows shortcuts denoted by the underscored letter in each command.

© Copyright 1999 by COSMIC Software

Figure 2-2 Key Binding Setup Box

Path Editor

The path option is used to define the search path to the source files. By default, ZAP searches the directory where the debugger is installed. Using the PATH editor you can browse through the directory structure to add the appropriate search paths. Simply double click on the desired path and click on Append, Add before or Add after to place the current path in the desired place in the search path. ZAP searches from top to bottom in the Path Editor to find source files.

© Copyright 1999 by COSMIC Software

Figure 2-3 Path Editor

Default Int Format

This option specifies the default display format for the variable browser window. Choose binary, decimal, octal or hexadecimal.

Save Config

This option saves ZAP's configuration immediately. The configuration save includes the location and size of the following windows if they're open when performing the save. The Main ZAP window, Disassembly, Monitors, Regsters, Stack, Status and Toolbar. ZAP also saves the fonts, highlights and colors.

Save Configuration on Exit

This menu item is used to turn the *Save Config on Exit* option on and off. When the menu item is on (checked) ZAP will save the location and size of the following windows if they're

© Copyright 1999 by COSMIC Software

open when exiting ZAP. The Main ZAP window, Disassembly, Monitors, Regsters, Stack, Status and Toolbar

© Copyright 1999 by COSMIC Software

Loading an Application

File Menu

Load

The *Load* option opens up the load dialog window as shown below. You can browse and select the load file by choosing a folder from the "Look In" pull down menu. The debugger accepts an absolute object file from the appropriate COSMIC compiler (i.e. output file from the linker). This file should be built with the +debug compiler option to include C source level debug information. For Assembly source level debugging, use the -x assembler option. Once the file has finished loading, the program counter is set to the address of the symbol __stext or the beginning of the first segment in the link file if the symbol __stext is not defined. This is typically the address of the beginning of the assembly level startup routine (crts.s or crtsi.s), which is used as the default reset vector.

Figure 2-4 Load Menu

© Copyright 1999 by COSMIC Software

Application Map

Choose application map to display the application segments along with their corresponding starting and ending addresses and segment sizes.

Load and Save Layout

ZAP Allows you to save and restore the screen layout at any time. The save layout command saves the size and location of the following windows: Source, Disassembly, Registers, Stack, Command, Monitor and the main ZAP desktop window.

Load and Save Session

The save session command is a superset of the save layout command. In addition to the layout, the save session command saves the last file loaded, the contents of the monitor window and any data windows and their addresses. The load session command opens all of the windows in the layout and then loads the saved application. Once the application is loaded ZAP will then fill the monitor window with the saved variables and open any saved data windows to their proper addresses. A session file can be loaded from a command line or shortcut using the -s option. e.g. zap.exe -s filename.ssn

About ZAP

The File menu item About ZAP provides a dialog box containing the configuration, version number and copyright for ZAP.

Exit

The exit menu command closes ZAP and all windows associated with the current invocation and optionally saves the configuration.

© Copyright 1999 by COSMIC Software

Utilities Menu

ZAP allows you to integrate and operate your favorite editor and make facility with the compiler and debugger. You can access several tools as well a DOS shell from a convenient button bar or pull down menu. To configure the button bar Choose *Configure* from the **Utilities** Menu to open the Tool Editor.

Configure Tools

The Tool Editor allows you to call any DOS or Windows editor by entering the appropriate editor commands in the Edit field. To call or switch to an editor with the current source file use the % character to denote the current source file.

Call Editor

To configure an editor simply enter the appropriate command to start the editor in the Editor field of the Tool Editor. To open or switch to the editor from ZAP select *Call Editor* from the **Utilities** Menu to invoke the desired editor.

Edit Current File

To setup an editor to automatically open the source file currently active in ZAP add the editor command followed by the % character to denote the current file. Click on the appropriate button or select *Edit Current File* to open the editor with the current source file.

Compile

Setup the compiler command line in the Compile field of the Tool Editor including all desired compile time options followed by a % character. To compile the active source file click on the Compile button or choose *Compile* from the **Utilities** Menu.

Compile Debug

Same setup as *Compile* with the addition of the debug option to

© Copyright 1999 by COSMIC Software

provide source level debug information. To compile choose *Compile Debug* from the **Utilities** Menu or click on the Compile Debug button.

Build

The *Build* command can be used to execute a makefile or link the current application. To configure the build command enter the desired command in the Build field. To execute the build command select **Build** from the **Utilities** menu or click on the Build button.

DOS Shell

To open a DOS Shell. Click on the DOS icon or select DOS Shell from the Utilities Menu.

© Copyright 1999 by COSMIC Software

On-line Help Facility

ZAP provides an extensive help facility. ZAP provides help on C language syntax and C library syntax as well as how to use ZAP.

Help on Using ZAP

Choose *On ZAP* from the help menu to display a list of help topics for ZAP. Double click on any subtopic to view manual pages on the topic.

Help on C Library

Choose *On C Library* from the Help menu to display a list of C libraries. Double click on any function name to display a manual page describing the syntax of the function.

Help on C Syntax

Double click on any C keyword or library function to open up a syntax description window.

© Copyright 1999 by COSMIC Software

© Copyright 1999 by COSMIC Software

CHAPTER

3

Program Execution

This chapter covers the many different ways to control program execution, including:

- Start and Stop Execution
- Single Stepping
- Reset and Restart
- Events and Breakpoints
- Activate and Deactivate Functions
- Browser Menu

© Copyright 1999 by COSMIC Software

Start and Stop Execution

Once an application is loaded into ZAP you have several options for executing your code. These options are found under the debug menu and on the Toolbar.

Normal Execution

Normal execution is the continuous real-time execution of the application (except with simulation of course). ZAP updates any active windows whenever execution halts.

Mouse and Menu

To start or resume execution type **g** when the Source window is active or choose Go from the **Debug** Menu, Button bar or Toolbar. This will start execution from the current PC and continue until an active breakpoint is reached or you select *Stop* from the Main menu.

Go Till Source Line Short-cut

If you want to execute the program until a particular source line simply hold down the control key and double click on the source line number. This can be done from any source window including C or assembly source browser windows.

Go Editor

The Go Editor allows you to execute your code until it reaches a specific file, function, line number, address. Choose Go Till under the debug menu to open the Go Editor.

- Double click on a function name or source file to execute until the specified function or source file is entered.
- Enter a source line number in the Line Box to execute until the line is reached.
- Enter an address in the address box to execute until the address is reached.

© Copyright 1999 by COSMIC Software

Figure 3-1 Go Editor

Command Window

Type **g** in the command window to start or resume execution. The **g** command accepts one of two possible arguments. You can enter either a number of C lines to be executed or the C line number to execute to. *See the g command in the "ZAP Commands" chapter for details.*

Stop execution

Mouse and Menu

You can stop execution at any time by clicking on *Stop or typing escape* in the Main menu. When the program stops all active or dynamic windows are updated and refreshed.

© Copyright 1999 by COSMIC Software

Single Stepping

Single stepping allows you to execute one Disassembly instruction or source line at a time and monitor changes to the system. All active and dynamic windows are updated after each single step. ZAP offers several different types of single stepping for greater flexibility. You can step at the source level or disassembly level, step into or over function calls and perform conditional stepping. ZAP coordinates the source display with the disassembly display using color coded highlight bars. The current source line and disassembly instruction (PC) is highlighted in the source and disassembly window respectively. ZAP also provides an additional highlight bar which covers the assembly instructions that make up the current line of source code. These highlights are continuously updated with all of the single step methods described below.

Mouse and Menu

- To step one source line and step into active functions Choose *Step* from the Debug menu or Toolbar or type **s** when the ZAP Source window is active.
- To Step one source line and step over function calls; Choose *Step Over* from the Debug Menu or *Step O* from the Toolbar.
- To Step one disassembly instruction and step into functions; Choose *Step Instr* from the debug menu or *StepI* from the Toolbar. You can also type **s** while the Disassembly source window is active to step at the assembler level.

Step Editor

The step editor allows you to perform multiple single steps and conditional single stepping. Choose step until under the debug menu to open the step editor.

- Click in the Assembler Box to enable and disable assembly

© Copyright 1999 by COSMIC Software

level stepping. The default is source level stepping.

- Double click on a function name or source file to single step until it enters the specified function or source file.
- Enter a number in the count box to perform multiple single steps from the current program counter.
- Enter a C line number in the Line box to step until the line is reached.

Figure 3-2 Step Editor

Command Window

The **step** command single steps one line of source code. The step command accepts two optional arguments. The following examples demonstrate some common uses of the step command.

See the step command in the "ZAP Commands" chapter for more details.

- To step one source line and step into active functions type **s**

© Copyright 1999 by COSMIC Software

or step in the command window.

- To step multiple source lines type **s #** where # is the number of source lines to be executed.
- Type **so or ostep** to step one source line and step over function calls.
- Type **si** or **istep** to Step one disassembly instruction and step into functions.
- Type **so put()** to step over function calls until the function put() is encountered.

© Copyright 1999 by COSMIC Software

Reset and Restart

Reset

Choose Reset from the debug menu to reset the processor or processor simulation.

Go from Reset

Choose *"Go from Reset"* from the **Debug** Menu to reset the processor or processor simulation and then issue a *Go* command to start execution. This option does not affect any events.

Restart

The **Debug** menu item *Restart* sets the PC to it's original value after loading the current application. This command does not affect any other registers or events.

© Copyright 1999 by COSMIC Software

Events and Breakpoints

Events and breakpoints are used to control program execution based on the state of the system. Code Events include breakpoints and watchpoints.

Code Events

Code events include breakpoints and watchpoints. A breakpoint is used to stop execution so that the system can be analyzed. A watchpoint is used to temporarily stop execution, perform an action and then continue execution.

Watchpoint

A watchpoint is used to temporarily stop execution, perform an action and then continue execution. A watchpoint is the same as a breakpoint except that execution resumes after the action is finished. To set a watchpoint choose “watchpoint” in the “Code Event Editor” or use the “Watch” command. See the chapter titled “ZAP Commands” for details. The specification and options for watchpoints are identical to that of breakpoints. Breakpoints can be converted back and forth to watchpoints by selecting the watchpoint or breakpoint box in the code event editor.

Breakpoints

A breakpoint is an event that causes execution of your program to be interrupted so you can examine the state of the system. You can set an unlimited number of active breakpoints on any C source line, address or data object. You can also associate debugger commands, user commands and complex expressions to any breakpoint. There are several methods for manipulating breakpoints, choose any of the following methods.

© Copyright 1999 by COSMIC Software

Setting/Editing Breakpoints

Mouse and Menu

- Double click on any valid line number in the Source window to set an unconditional breakpoint on a line of source code. A valid line number refers to a line of source code that actually produced assembly code and is shown in bold.
- Double click on any bold line number in a source browser or Event Browser window (See Browsing options for more information).
- Double click on an address in the Disassembly window to set a breakpoint on an address.

Command window

The **b** command is used to set and display breakpoints. The breakpoint command accepts the following syntax:

```
break [/<options>] [<location>] [{<action}]
```

See The break command in the "ZAP Commands" chapter for a complete description.

- Type **b:line#** in the command window to set a breakpoint on the line number in the current source file.
- Type **b function()** in the command window to set a breakpoint on every line of the function().
- Type **b/4 main():8** to set a breakpoint on line 8 of the function main that will only halt execution every fourth time the line is executed.
- Type **b foo():3 {u i 2}** to set a breakpoint on line 3 of function foo() and perform the action specified inside the curly braces. In this case, the action is to update variable i to 2 when the breakpoint is taken.

© Copyright 1999 by COSMIC Software

Deactivating/Activating Breakpoints

ZAP allows you to deactivate or suspend any breakpoint without removing it from the system. You can then selectively activate them as needed. When a breakpoint is deactivated it will not halt or interfere with execution. Note: The breakpoint highlight will change colors when you activate and deactivate a breakpoint. Choose any one of the following methods to activate and deactivate breakpoints:

Mouse and Menu

- To activate or deactivate a breakpoint double click on a breakpoint line number in the source window or source browser window while holding down the shift key.
- Double click on an active breakpoint and choose *on/off* from the pull down menu to activate or deactivate the breakpoint.
- Choose *Browse* from the Events Menu and double click on a breakpoint while holding the shift key to activate or deactivate the breakpoint.
- Choose *Browse* from the Events Menu and double click on a breakpoint. Choose on/off from the popup menu to deactivate or activate the breakpoint.
- Choose *Events from the Browser menu* and double click on an active breakpoint while holding the shift key to deactivate the breakpoint.
- Choose *Events from the Browser menu* and double click on an active breakpoint. Choose on/off from the popup menu to deactivate or activate the breakpoint.

Command Window

The Code Event Editor is not available as a Command Window option.

© Copyright 1999 by COSMIC Software

Deleting Breakpoints

Breakpoints are completely removed from the system by deleting them. Choose any of the following methods to delete a breakpoint.

Mouse and Menu

- To delete a breakpoint double click on an active or suspended breakpoint line number in the source window or source browser window while holding down the control key
- Double click on an active or suspended breakpoint line number in the source window or source browser window and choose *Delete* from the popup menu
- Choose *Browse* from the Events Menu and double click on a breakpoint while holding down the control key to delete the breakpoint.
- Choose *Browse* from the Events Menu and double click on a breakpoint. Choose *Delete* from the popup menu to delete the breakpoint.
- Choose *Events from the Browser menu* and double click on a breakpoint while holding the control key to delete the breakpoint.
- Choose *Events from the Browser menu* and double click on a breakpoint. Choose *Delete* from the popup menu to delete the breakpoint.

Command Window

To delete a breakpoint from the command window you can use the **del** command. The following examples demonstrate the use of the del command to delete breakpoints. *See the **del** command in the "ZAP Commands" Chapter for details.*

- Type **del #** where # is the breakpoint number as shown to

© Copyright 1999 by COSMIC Software

the left of a breakpoint in a Event Browser window.

- Type **del *** to delete all of the breakpoints from the system.

Code Event Editor

The Code Event Editor can be used to set, suspend and delete breakpoints. The Code Event Editor also lets you attach an action to a breakpoint or create a watchpoint. The Code Event Editor can be opened in several ways. Choose any one of the following to open the Code Event Editor:

- Choose *Code Event* from the Events Menu to open up the Code Event Editor.
- Choose *Browse* from the Events Menu, double click on a breakpoint and choose *Edit* from the popup window.
- Double click on any breakpoint in the source window and choose *Edit* from the popup menu.
- Double click on a breakpoint in any source browser window and choose *Edit* from the popup menu.
- Choose *Events from the Browser menu*, double click on a breakpoint and choose *Edit* from the popup menu.

Setting Breakpoints

To set a breakpoint using the Code Event Editor choose the desired conditions and click on OK to set.

- Choose or enter a function name to set a breakpoint on every line of the function or on function entry.
    - Choose *Whole* to set a breakpoint on every C line in the selected function.
    - Choose *On Entry* to set a breakpoint on the entry of the selected file or function.

© Copyright 1999 by COSMIC Software

- Choose or enter a filename to set a breakpoint on every line of the source file.
- Choose Breakpoint to stop execution when the condition is met.
- Choose Watchpoint to silently stop execution, execute the action (if any) and resume execution.
- Hit Count Box - Enter a number in the Hit Count Box to specify the number of times the breakpoint will be executed before execution is halted.
- Choose Active or Suspended from the Status box to activate or suspend an the breakpoint or click the ON/OFF button.
- Action Box - Enter any ZAP command or combination of ZAP commands in the action window to attach an action to the current breakpoint. *See the chapter "ZAP Commands" for more information about ZAP Commands.*

© Copyright 1999 by COSMIC Software

Figure 3-3 Code Events Editor

© Copyright 1999 by COSMIC Software

Displaying and Editing Breakpoints

Active and suspended breakpoints are denoted by a color highlight over the source line number or assembly address in the Disassembly window or any Browser window. To display a complete list of all existing breakpoints choose Browse from the Events Menu or choose Events from the Browser menu to open up the Event Browser window. You can click on any breakpoint listed in this window to activate, deactivate, delete or edit it. *See the Breakpoints section of this chapter for more information on activating, editing and deleting breakpoints.*

© Copyright 1999 by COSMIC Software

Activate and Deactivate Functions

ZAP gives you the option to selectively activate or deactivate functions you want to debug. When a function is deactivated you can no longer step into the function or monitor local variables and the function will not be included in a source trace display. This allows ZAP to operate more efficiently and eliminates unwanted information in the source trace. By default, ZAP activates all functions in your application that are compiled with the debug option. The status of each function is listed in the first column of the Function Browser Window. An activated function is indicated by (on) and a deactivated function is denoted by an (off) tag.

Mouse and Menu

To activate and deactivate functions simply double click on the word (on) or (off) in the first column of any function browser window.

Command Window

To activate a function use the activate command **a**. The **a** command uses the following syntax. For more information see the **a** command in the *"ZAP Commands"* chapter.

activ <name_list>

To deactivate a function use the deactivate command da. For more information see the **d** command in the *"ZAP Commands"* chapter

deact <name_list>

Where <name_list> is one or more function names to be activated. The **activ** and **deact** commands also accept the standard wildcard character (*) to denote all functions. For example:

1. Type **deact foo()** to deactivate the function foo().
2. Type **activ foo()** to activate the function foo().
3. Type **deact *** to deactivate all the functions in the current application.

© Copyright 1999 by COSMIC Software

Browser Menu

ZAP's unique browser menu lets you quickly search and monitor sources, breakpoints, data objects or any memory location.

Event Browser

The Event Browser window displays a list of all existing Events (active and suspended) This includes code breakpoints and watch-points. You can click on any event to activate/deactivate, edit or delete. See the section on Events for more information on setting and editing events.

Figure 3-4 Event Browser Window

Source Browser

Zap's unique source browser allows you to search and view all of your source code in multiple discrete windows. You can set, edit and delete breakpoints anywhere in your code without changing the source window or the current state of execution. This is done by double clicking on C line numbers. There are several different ways to browse your source. Choose any of the following:

File Browser

- Choose **File List** from the Browse Menu to open the File Browser window. This window contains a list of all the

© Copyright 1999 by COSMIC Software

source files that make up the loaded application. You can double click on any source file name to open a source browser window containing the selected source file.

Figure 3-5 File Browser Window

File Browser Dialog

- Select **File** from the Browse Menu to open the File Browser dialog box. The dialog box contains a list of all the source files that make up the currently loaded application. Choose any source file and click OK to open a source browser window containing the selected source file.

© Copyright 1999 by COSMIC Software

Figure 3-6 File Browser Dialog Box

Function Browser

- Choose **Function List** from the Browse menu to open the Function Browser window. This window contains a list of all the source files and functions that make up the loaded application. You can double click on any source file or function name to open a source browser window containing the selected source file or function.

© Copyright 1999 by COSMIC Software

Figure 3-7 Function Browser Window

Function Browser Dialog

- Select **Function** from the Browse Menu to open up the Function Browser dialog box. The dialog box contains a list of all the source files and functions that make up the loaded application. Select any source file or function and click OK to open up a source browser window containing the selected function.

Figure 3-8 Function Browser Dialog Box

© Copyright 1999 by COSMIC Software

Any Source

Choose **Any Source** from the Browse Menu to open a standard Windows browser dialog. You can view any file on your system. If you open a file that is part of the loaded application then the C line numbers will be black and you can set break-points in the file. If the file is not part of the loaded application then the line numbers will be gray and the file is treated as read only

.

Figure 3-9 Any Source Browser Dialog Box

Memory Browser

The memory browser allows you to examine any valid memory locations in several formats including a disassembly.

- Choose **Memory** from the Browse Menu and select *Data* in the Memory Window Configuration dialog box to display or dump memory.
- Choose **Memory** from the Browse Menu and select *Code* in the Memory Window Configuration dialog box to disassemble a block of memory.

See *Displaying and Updating Memory* in the chapter 5 (Monitoring

© Copyright 1999 by COSMIC Software

Application Data) for more details on displaying memory.

Variable Browser

The variable browser allows you to view all of the variables in the loaded application. There are three different formats you can use to display the variable information. The Brief format displays the variable name and type. The Standard format lists the variable name, type and value and the Full format displays variable name, type, value and address. There are also four different display options as described below.

- Choose *In Current Function* submenu to display all variables local to the current function.
- Choose *In Current File* submenu to display all static variables in the current file scope, global variables declared in the current file and all variables local to the current function.
- Choose *In Global List* to display all variables in the current scope including extern globals, statics and locals.

© Copyright 1999 by COSMIC Software

Cross Reference Browser

The Cross Reference Browser displays the calling tree for application functions.

Function Cross Reference

This dialog window allows you to choose a particular function to display the cross reference information. Click on a function name to open a Cross Reference window.

Figure 3-10 Function Cross Reference Dialog

Cross Reference Window

The Cross Reference Window displays the function calling tree for a particular function. Double click on a colored function name to display the cross reference tree for that function.

Figure 3-11 Cross Reference Window

© Copyright 1999 by COSMIC Software

Symbol List Browser (sorted)

The Symbol List Browser provides a list of all global C and Assembly symbols sorted by address or alphabetically by name. Double click on any file name to display a source browser window. Double click on any symbol name and select from the following:

Address of	Displays the address of the symbol
Evaluate as	Displays the value of the symbol in byte, word or long format.
Update as	Allows you to update the contents of the symbol as a byte, word or long value.
Set Breakpoint	Sets a breakpoint at the symbols address.
Show Code	Displays a disassembly of the symbols address.
Show Data	Provides a data dump staring at the address of the symbol.

Figure 3-12 Sorted Symbol List

© Copyright 1999 by COSMIC Software

Symbol Browser

The Symbol Browser window allows you to quickly search through the symbol table to find the address of a symbol. Simply type the symbol name or a part of the name to search the list. Click on the symbol to display the address.

Map

Choose Map to display the application segments along with their corresponding starting and ending addresses and segment sizes. This is identical to selecting Application Map from the **File** menu.

© Copyright 1999 by COSMIC Software

© Copyright 1999 by COSMIC Software

CHAPTER

4

Monitoring Application Data

ZAP offers several advanced features to help you optimize your C code as well as track down those hard to find bugs. This chapter includes the following sections:

- Monitoring Variables and Expressions
- Updating Variables
- Evaluating Expressions
- Displaying and Updating Memory
- Evaluating Assembly Symbols
- Displaying and Updating Registers
- Displaying the Stack Frame

© Copyright 1999 by COSMIC Software

Monitoring Variables and Expressions

ZAP provides an extensive monitoring facility. You can monitor or watch variables one at a time in the Monitors window or view all the variables in the current scope in the Variable window. Both windows are updated each time program execution is halted.

Monitors Window

Choose *Monitors* from the show menu or simply monitor a variable or expression and the Monitors window will automatically open. ZAP allows you to monitor as many variables as you want and change the display format of any variable.

Adding Monitors

There are several different ways to monitor variables and expressions. Choose any of the following methods.

Mouse and Menu

- To monitor or watch a variable, double click on a variable name in the source window and choose *Monitor* from the pop-up menu.
- To bypass the pop-up window, double click on a variable name while holding down the control key.
- Drag and Drop - Right Click on the variable name and drag it to either the monitor window or it's icon on the button bar.
- To monitor an expression, select the entire expression by dragging with the left button and choose *Monitor* from the pop-up menu.
- To bypass the pop-up window, select the expression while holding down the control key.

Command Window

Use the **monit** command with the variable name or expression to add it to the Monitors window. The following examples

© Copyright 1999 by COSMIC Software

demonstrate some common uses of the **monit** command. *For more detailed information see the **monit** command in the "ZAP Commands" chapter.*

- Type **monit i** to monitor the variable i.
- Type **monit /x i** to monitor the variable i in hexadecimal format.
- Type **monit &i** to monitor the address of variable i.
- Type **monit i+1** to monitor the value of the expression "i+1"
- Type **monit /s buffer** to monitor the variable buffer as a string.

NOTE

Variables and expressions must be in the current scope to be evaluated or monitored.

Monitor Format

ZAP displays all variables in their declared formats by default. However, you can change the format by double clicking on any variable name or expression in the Monitors Window and choosing a format from the pop-up submenu.

Deleting Monitors

- To delete a monitor simply double click on a variable name or an expression in the Monitors window and choose *Delete* from the pop-up window.

Address of Source Lines

To display the address of any active line in the source window or source browser window choose Show->Address from the source window menu or double click on the line number while holding down the shift key.

© Copyright 1999 by COSMIC Software

Updating Variables

ZAP lets you update or change the value of any variable in the current scope.

Mouse and Menu

To update a variable, double click on the variable name and choose *Update* from the pop-up window. This opens the Update dialog box where you can enter a new value for the selected variable. The entry format can be changed by clicking on the Format button and choosing the desired format from the pop-up window. This allows you to enter the new value in any of the following standard formats. ZAP will make any necessary conversions.

Character - Enter the desired ASCII character between the apostrophes.

Octal - Enter an octal value in standard C notation with using a leading zero.

Decimal - Enter a signed decimal value.

Unsigned - Enter an unsigned value.

Hexadecimal - Enter an hexadecimal value in standard C notation using a leading 0x. (e.g., 0x100 for hexadecimal 100)

String - Enter an ASCII character string between the double quotes.

Figure 4-1 Update Dialog Box

© Copyright 1999 by COSMIC Software

Command Window

The **update** command is used to update a variable from the command window. The following examples demonstrate some common uses of the update command. *For more detailed information see the **update** command in the "ZAP Commands" chapter.*

- Type **update i 3** or update **i=3** to update variable i to the value of 3.
- Type **update buffer "abc"** to update the character string buffer to abc.
- Type **update ch 'a'** to update the character variable ch to the letter a.

© Copyright 1999 by COSMIC Software

Evaluating Expressions

ZAP allows you to display the value of any variable or expression in a temporary pop-up window. This feature helps avoid cluttering up the Monitors window with variables and expressions that you only need to display occasionally.

Evaluate Expression

There are several different ways to evaluate variables and expressions. Choose any of the following methods.

Mouse and Menu

- To evaluate a variable, double click on a variable name in the source window and choose *Evaluate* from the pop-up menu.
- To bypass the pop-up window, double click on a variable name while holding down the shift key.
- To evaluate an expression, select the entire expression by dragging with the left button and choose *Evaluate* from the pop-up menu.
- To bypass the pop-up window, select the expression while holding down the shift key.

Command Window

Use the eval command with the variable name or expression to evaluate it. The following examples demonstrate some common uses of the eval command. *For more detailed information see the **eval** command in the "ZAP Commands" chapter.*

- Type **eval i** to evaluate the variable i.
- Type **eval /x i** to evaluate the variable i in hexadecimal format.
- Type **eval &i** to evaluate the address of variable i.
- Type **eval i+1** to evaluate the value of the expression "i+1"
- Type **eval /s buffer** to evaluate the variable buffer as a string.

© Copyright 1999 by COSMIC Software

Displaying and Updating Memory

ZAP allows you to display, disassemble or modify any block of memory. This can be done using either the Browse or View menu. Choose **Memory** from the Browse Menu or the Show Menu to open the Memory Window Configuration dialog box. This box requires you to enter the starting address for the memory block to be displayed. You also need to choose whether you want the contents of the memory block disassembled (Code) or a raw data dump (Data)
.

Figure 4-1 Memory Window Configuration Dialog Box

Disassembling Memory

Choose *Code* under format in the Memory Window Configuration Dialog box and enter a valid code address in the Address box. This will open a browser window containing a disassembly of the specified memory block including symbols. You can set a breakpoint by double clicking on any memory address in the disassembly.

Displaying Memory

To produce a raw data dump Choose *Data* under format in the Memory Window Configuration dialog. This will bring up the Data

© Copyright 1999 by COSMIC Software

Configuration dialog box which is used to format the Raw data. To configure the data display, you have the following options:

1. Address - Enter the desired starting address or symbol for the memory dump in the address box.
2. Size - Choose a convenient data size for the display. Byte, word or long word.
3. Format - Choose the desired display format; Octal, decimal or hexadecimal.
4. ASCII - Choose Yes to include an ASCII display next to the memory dump. Choose No to show only the numerical display

.

Figure 4-1 Data Configuration Window

Updating Memory

If you configure the display for data you can double click on any value in the numerical or ASCII display to update. For example to update address 0x00410 in the display below, double click on the value b000 next to address 0x00410 and enter the new value.

© Copyright 1999 by COSMIC Software

.

Figure 4-1 Data Display Window

Fill Memory

The data display can be used to fill memory with a pattern. Choose fill from the Data display menu to open the Memory fill window. In the memory fill window you can enter a fill pattern or choose random to have ZAP create a random pattern. Select the data size and the address range in the "To" and "From" boxes. Choose verify to have ZAP read back the filled memory and verify that it is correct. See the Fill command in the chapter "ZAP Commands" to use the command window or debugger script to fill memory.

Figure 4-2 Data Fill Window

© Copyright 1999 by COSMIC Software

Saving a Memory Dump to a file

Click on **Dump** in the Data Display window to save the raw memory dump to a file. Enter the address range in the Memory Data Dump window and enter or select a file to save the dump.

Saving Memory Dump as S-Record format.

Click on **S-Record** in the Data Display window to save the memory dump to a file in S-Record format. Enter the address range in the "From" and "To" fields in the Memory Data Dump window and enter or select a file to save the S-Record.

Figure 4-3 Data Dump as S-Record Window

Display Highlights

ZAP provides a convenient way to keep track of memory modifications using colored highlights to denote memory changes. Memory highlights are updated each time execution is halted and cleared upon reset.

© Copyright 1999 by COSMIC Software

Evaluating Assembly Symbols

ZAP allows you to display the value of any global symbol in a temporary pop-up window. Double click on any global symbol in any source window to open popup window with the following options.

Address of	Displays the address of the symbol. You can also get the address by holding down the shift when double clicking on the symbol.
Evaluate as	Displays the value of the symbol in byte, word or long format.
Update as	Allows you to update the contents of the symbol as a byte, word or long value.
Set Breakpoint	Sets a breakpoint at the symbols address.
Show Code	Displays a disassembly of the symbols address.
Show Data	Provides a data dump staring at the address of the symbol. Alternatively hold down the control key when double clicking on the symbol.

© Copyright 1999 by COSMIC Software

Displaying and Updating Registers

ZAP provides a window display dedicated to the CPU registers. The Register window allows you to display and update any of the processor registers with a point and click. ZAP also highlights changes to the CPU registers each time the Program Counter is incremented so you can track CPU changes instruction by instruction. The register display is updated each time program execution is halted or after every single step.

- To open and close the Register Window, Choose *Registers* from the *Show menu* or type the **r** command in the Command Window. See the **r** command in the "ZAP Commands" chapter for more information.
- To update a register directly simply double click on the value of the register and enter a new value.
- To update a register with a symbol or function double click on the name of the register to open the update register window. Click on Symbol to open the Symbol Browser or click on Function to open the Function Browser Window. Select the desired symbol or function to update the active register.

Figure 4-1 Update Register Window

© Copyright 1999 by COSMIC Software

Displaying the Stack Frame

The Stack Window displays the current stack frame and arguments with the active function nested to the bottom of the display. The stack display is updated each time program execution halts. The Stack Window allows you to double click on any function in the stack frame to open up a Source Browser Window containing that function. To open and close the stack Window choose *Stack* from the *Show menu* or use the **T** command. *For more information about the Toggle Stack Display command, see the* ***T*** *command in the "ZAP Commands" chapter.*

Figure 4-1 Stack Window

© Copyright 1999 by COSMIC Software

© Copyright 1999 by COSMIC Software

CHAPTER

5

Advanced Topics

ZAP offers several advanced features to help you optimize your C code as well as track down those hard to find bugs. This chapter includes the following sections:

- Simulated I/O

© Copyright 1999 by COSMIC Software

Simulated I/O

ZAP provides a general facility to perform simulated I/O using files stored on your computer. Simulated I/O is a means by which you can bring input right into you embedded system or send output from your system to data files on your computer. These files can be created and manipulated with standard computer utilities like spread sheets and data bases and math programs. Embedded systems development has always had one big problem. It is difficult to get data into the system to test the algorithms until most or all of the hardware is finished. Simulated I/O bridges the gap. You can create data or stimulus files that can be read into the system from anywhere in the system as easily as setting a watch point.

Simulated I/O uses the fopen, fclose, fread and fwrite commands in conjunction with the watch point mechanism. These commands can also be used directly in the command window for you just want to get or output something one time or periodically throughout a command script. To setup the mechanism follow the steps below.

1. Open the files you want to use for simulated I/O using fopen. (e.g. fopen /c:1 c:\test.out)

2. Set a watch point in your application where you want to send or receive data. This is often in a function that reads or writes to a hardware I/O register. Note the application must be able to execute to this location. You may need to set additional watch points to set some conditions so execution will get the desired location.

   a) Enter the desired I/O command in the action box of the watch point edit window. e.g. fwrite /c:1 “%d”var

3 Set a breakpoint to stop the application after the I/O is finished.

4. Execute the application to the breakpoint or till the end.

5. Close the I/O files using the fclose command. (e.g. fclose /c:1)

© Copyright 1999 by COSMIC Software

In general, you'll want to create a script to setup the watch points with the appropriate actions for your specific application. See the chapter "ZAP Commands" for details on the commands **record, input, watch, fopen, fclose, fread and fwrite.**

The following example uses an input script to open two files and set two watch points to simulate input from a file process the input using the target processor and then output the result to a file on the host computer.

Example

The file below is a simple program which reads in a couple of integers adds them together and outputs the result to a file as formatted text using printf.

```
#include <stdio.h>

int a,b,sum;
int getinput(void);
int putchar (char c);

void main(void)
  {
    a = getinput();
    b = getinput();
    sum = a+b;
    printf("The result of a+b is %d \n",sum);
END_MAIN:  ;
}  //  END MAIN
```

The following routines are dummy input and output routines. These could be any I/O routines. The simulated I/O mechanism is created by the watch points with the fread and fwrite commands. Note the labels OUTCH and INCH. These are standard C goto labels. Although, the goto feature in C has traditionally been taboo, using just the labels is useful for creating debugger breakpoints and watch-

© Copyright 1999 by COSMIC Software

points. These labels are treated as local symbols and allow you to create scripts which contain relative break and watch points within a function. Without using such labels you would need to know which line of the function you want to set the break and thus if you modify the source file you may need to modify any scripts containing break-point in that function.

NOTE

If you want to use the local labels (goto) described above in a ZAP script you need to compile with extra debug information. Use the -xx option on the preprocessor/parser. for example
cx6812 -pxx +debug -vl acia.c

```
/*  Output a character
 */

 int putchar (char c)
   {
OUTCH:
       return (c);
   }


/*  GET INPUT for simulation
 */

int getinput(void)
   {
   volatile int input;
INCH:
    return (input);
   }
```


© Copyright 1999 by COSMIC Software

The following is a ZAP input script that will load the example application open the two I/O files, set two watch points to perform the I/O simulation and then execute the application. Each time the fread watchpoint is taken the next value in the file is read in to simulate a flow of input. The fwrite watchpoint routine appends output to the file and so as not overwrite the previously written information.

```
*   Remove old output file

remove data_out.txt

*   Load Application Executable

load sim_io.h08

*   Open input data file on channel 1

fopen /c:1 data_in.txt

*   Open Output file as channel 2

fopen /c:2 data_out.txt

*   Set Watch point for Simulated I/O (simulates getinput)

watch getinput():INCH {fread /c:1  "%d"input }

*   Set Watch point for Simulated I/O (simulates putchar)

watch putchar():OUTCH {fwrite /c:2  "%c"c }

*  Set a breakpoint at the end of the program
 break main():END_MAIN

*   Execute the program
go

*   Close input and output files

fclose /c:1
fclose /c:2
```


© Copyright 1999 by COSMIC Software

The file data_in.txt contains the following data before and after the script is executed:

```
12 24
```

After executing the script the file data_out.txt is created and contains the following:

```
The result of a+b is 36
```

If you want to simulate your hardware controlled I/O registers you may need to add additional watch points to set the proper conditions. For example, it is common in an SCI routine to loop on the transmit enable to make sure the hardware is ready to receive a character. When simulating the SCI you would need to either #ifdef around the loop or set an additional watch point and set the condition. e.g.

watch putchar():SCI_LOOP {update SCSI = (SCSI ^ SCTE)}

```
while (!(SCSR & SCTE))
    {
   SCI_LOOP:
        SCDR = c;
    }
```


© Copyright 1999 by COSMIC Software

CHAPTER

6

ZAP Commands

ZAP provides an extensive command set which duplicates the functionality of many of the mouse selections and popup windows. The command Window can also be used to create an automated debugging session by loading an input command file. An input command file may contain any valid ZAP commands. This chapter describes the ZAP commands and their syntax and includes the following sections:

- Command Line Syntax
- Specifying Memory Locations and Registers
- User defined variables
- Entering ZAP Commands
- Command Descriptions

© Copyright 1999 by COSMIC Software

Command Line Syntax

All ZAP commands use the following basic syntax:

```
<command_name>  [/options]  [<argument>] [<argument>]
```

<command_name> specifies a ZAP debugger command. Commands must be separated from its options and arguments by one or more spaces or horizontal tabs.

<options> specifies extra options for the command. Each option is preceded by a forward slash (/). *<options>* must be placed after the <command_name>, but before the <argument> .

<argument> specifies an optional location within your C or assembly language source code, a data object, an unsigned decimal integer, a memory address, or a C language expression. The possible forms that *<argument>* can take are described below. The argument must follow any specified options with one or more spaces or horizontal tabs.

© Copyright 1999 by COSMIC Software

Specifying Memory Locations and Registers

Many *ZAP* commands require or accept an *<argument>*. An argument may be a constant, internal processor register, memory location, file name and line number, function name or variable name. The argument command language accepts many C style expressions and operators so objects may be accessed directly or indirectly.

- Constants and Expressions
- Target processor's registers
- Source files and Functions
- Data objects and Pointers

Constants and Expressions

ZAP commands accept any legal C constant and many C operators and expressions. ZAP also accepts binary constants using the 0b prefix notation.

Binary Constants - 0b prefix (e.g. 0b1011)

Decimal Constants - standard notation (i.e no prefix)

Hexadecimal Constants - 0x prefix (e.g. 0x1AB)

Octal Constants - 0 prefix (e.g. 0765)

Register Manipulation

A register specification must have the following form:

```
$<register_name>
```

where *<register_name>* follows the naming conventions of the processor's manufacturer. A register specification is identified by its leading *$* character. You can use any CPU register listed in the register window simply by prefixing it with a $.

© Copyright 1999 by COSMIC Software

User defined variables

ZAP allows the user to define any number of user defined variables. A user defined variable is a symbol defined and recognized by ZAP for use in expressions. A user defined symbol is always prefixed with the '.' character. To create a user defined variable simply use it in an expression. If it doesn't exist it will be created with an initial value of zero. For example:

To create the variable .temp and set it equal to the constant 10 you could write:

```
ZAP> eval .temp=10
```

To set the variable .temp equal to the program variable "var1":

```
ZAP> eval .temp=var1
```

To create and set the variable .tmp_cc equal to the condition code register and test .tmp_cc to see if interrupts are enabled:

```
ZAP> eval .tmp_cc=$cc
ZAP> if (.tmp_cc ^ 0x8) mess "interrupts disabled\n"
else mess "interrupts enabled\n"
```

Source files and Functions.

ZAP command arguments can also be a location specifier designating one or more valid lines of C or assembly language source code. A valid line of source code is defined as any line that is associated with an executable piece of code which is compiled in debug. A source line designation can have one of the following forms:

file_name:line_number - Specifies a line number in the given source file. e.g To set a break point on line 55 of testt.c the command would be:

break test.c:55

© Copyright 1999 by COSMIC Software

file_name: - Specifies all executable C lines in the named source file. e.g To set a break point on all source lines in a particular source file the command would be:

```
break  test.c
```

function():line-number - Specifies a line number in the named function. The current source file is assumed. e.g. To set a breakpoint on a specific line of a function in the current source file the command would be:

```
break  main():38
```

function() - Specifies all executable C lines in the named function. e.g. To set a breakpoint on all source lines in a specific function the command would be:

```
break main()
```

NOTE

Only source line numbers which correspond to actual code are recognized by the line number specification.

Data Objects

A location specifier for <argument> that designates a data object can have any one of the following forms:

data_object_name - Specifies a global data object. e.g. To evaluate a global variable named bar you would type:

```
eval bar
```

file_name:data_object_name - Specifies a static variable with file scope. e.g. To evaluate a file static variable bar when the necessary file is not in the current scope:

© Copyright 1999 by COSMIC Software

eval test.c:bar

function():data_object_name - in scope of function named. e.g. To evaluate a variable “tmp” local to the function main():

eval main():tmp

[:]data_object_name - scope of current function. e.g. To monitor the local variable “i” while it is in scope:

monit :i

number - explicit constant specified in any of the following formats: hexadecimal (0x100), decimal (16), octal address (020) or binary (0b10000)

expression - C language expression

A data object name is an identifier currently in scope as a data object. You can change the current scope with the *move* command. You can specify a data location using a C expression involving register values, variable names and values, constants and C language operators, assuming that the result is an addressable object, described as a LVALUE in C parlance

Pointer Indirection

The debugger can access data objects both directly and indirectly by a pointer, as in the C language itself.

You can specify indirection on a pointer data object only as many times as you specify the pointer attribute in your original declaration. If you request too many levels of indirection, the debugger prints an error message indicating a syntax error.

You can use any C expression, referencing structure field through pointers while you respect the correct C syntax. The expression evaluator checks for almost the same errors that your C compiler does.

© Copyright 1999 by COSMIC Software

Entering ZAP Commands

You enter *ZAP* commands at the “ZAP>” prompt.

You terminate each command with a carriage return, newline, or linefeed character. *ZAP* allows you to string several commands together. To specify multiple commands in response to a single prompt, type each command in the usual way and separate each command with a semicolon ‘;’ character. A whitespace character on either side of the semicolon is optional. *ZAP* splits multiple commands on an input line and performs each operation separately, just as if you had entered each command in response to a separate prompt.

© Copyright 1999 by COSMIC Software

Command Descriptions

All commands described are documented in a similar fashion to facilitate quick reference. The name of the command appears at the top outside corner of the page on which it is described. Its name and a brief description of the action it performs appears at the top of the text under the heading **Name**. A brief synopsis, under the heading **Syntax**, describes the command syntax and the options and parameters it accepts. In this context, a name enclosed in angle brackets, such as <*argument*>, is an element which is defined elsewhere in the discussion or is self evident. In the case of multiple options, the description of each command tells you which options may be used together.

The character wild card character '*' directly to the right of an element denotes an <argument> that may appear one or more times. The command del, for example, allows one or more events to be deleted. e.g. To delete all events you would type:

```
del /e *
```

Enter all other characters in the synopsis as shown.

A more detailed description of the command and the options and parameters it accepts follows under the heading **Function**. The default value for each option, if any, is specified here.

One or more examples follow the command explanation, under the heading **Example**. The examples given are not intended to represent the precise behavior of *ZAP* for any specific processor or with any specific program.

© Copyright 1999 by COSMIC Software

ZAP Commands

The following commands are available in ZAP. Detailed descriptions of each command follows the summary.

***** - *print comment*

activ - *activate function*

break - *set or modify breakpoint*

deact - de*activate function*

del - *delete breakpoint, monitored or user function*

disa - *toggle disassembly window*

dump - *dump memory as byte word or long*

eval - *evaluate an expression*

fclose - Close an open file (as a channel) that was used for I/O

files - show source files for application loaded

fill - block fill memory

fopen - Open a file (as a channel) for input or output

fread - Read from an open file (channel) and update program variables

fwrite - write to an open file (channel)

frame - dump the stack to the command window

funcs - show functions for application loaded

go - start or resume execution at the current PC

if - test program condition

input - load a zap command file

istep - *step at the disassembly level through the program*

© Copyright 1999 by COSMIC Software

mess - *print a formatted message*

load - *load a file*

mm - Modify memory

monit - *monitor an expression*

move - move in stack frame

ostep - *step over at the source level and step over function calls*

output - redirect commands and results to a file

path - set the search path for source files

print - print a file or function

quit - *quit zap*

record - *record a session for playback*

regs - *dump registers to the command window*

rem - *print comment*

remove - delete file

reset - *execute a target reset*

session - *record a ZAP session which includes the layout and last file loaded.*

stack - show stack

s - *step at the source level through the program (enters active functions)*

si - *step at the disassembly level through the program*

so - *step over at the source level and step over function calls*

step - *step at the source level through the program (enters active functions)*

© Copyright 1999 by COSMIC Software

u - update variable

update - update variable

vars - Open a variable browser window

watch - watch *an expression*

wstack - toggle the stack window

wregs - *Toggle the register window*

T - *toggle stack frames status display*

update - *update a data object*

write - *save events or monitors*

x - *evaluate an expression*

zero - *clear all events, monitors or reset the processor*

© Copyright 1999 by COSMIC Software

*

Description

Comment

Syntax

```
*comment
```

Function

* allows you to write a comment, mainly in a command file or a function. The content of the remaining text up to the end of line is ignored by the debugger.

Example

In a command file:

```
*
* Dump registers to the command window
*
regs
```

Alias - REM

Comments may be created with either the asterisk (*) or the command rem.

© Copyright 1999 by COSMIC Software

activ

Description

Activate a function

Syntax

```
activ <name_list>
```

Function

The *activ* command is used to activate a function, i.e. to make it possible to debug it using *ZAP*. By default, all functions that have been compiled for debugging are activated by *ZAP*. You can deactivate a function with the *deact* command; to reactivate it again, you use the *activ* command. A name list is composed of one or several function names, with their parenthesis and :, separated by commas. Typing a star '*' instead of a *<name_list>* will activate all possible functions.

Once a function is deactivated, it behaves as if it had NOT been compiled for debugging.

The fewer active functions, the quicker *ZAP* is able to work. So once a function has been tested it is worth deactivating it, thus allowing you to focus more quickly on debugging the remaining functions.

Example

To activate function fact:

```
ZAP> activ fact():
```

To activate function fact and foo:

```
ZAP> activ fact():,foo():
```


© Copyright 1999 by COSMIC Software

break

Description

Set, modify or display breakpoint event

Syntax

```
break [/<options>] [<location>] [{<action}]
```

Function

The **break** command sets or displays the “breakpoint” at <*location*>.

A breakpoint is an event that causes execution of your program to be interrupted so you can examine its state. You can set a breakpoint on a C source line(s) or an absolute address. Program execution will be interrupted when control passes to that line or address.

A breakpoint can be set on a range of lines rather than on a single line. Ranges of lines are specified using the ‘**:**’ character. For example if you want to set a breakpoint on lines 20 to 35 of function main() you would type: **`break main():20:35`**. Typing for example **`break main():34`** sets a breakpoint on line **34** of **main()** only.

Options

/c:<count> can be used to specify an optional count, which specifies the number of times the breakpoint must be reached before it halts execution. It is then possible for example to set a breakpoint when a particular C line has been executed a specific number of times.

/a Reactivate a suspended breakpoint

/s Suspend an active breakpoint. The breakpoint is still set, but is

© Copyright 1999 by COSMIC Software

not active and will not cause execution to stop.

When *ZAP* reaches <*location*> during program execution <*count*> number of times, it performs the specified <*action*>.

<*action*> can be any valid *ZAP* command or set of commands. The default <*action*> is to stop, refresh any open windows and prompt for command input.

If you do not specify <*location*>, *ZAP* lists all active breakpoints.

The display of a breakpoint includes various information:

(1) First the breakpoint number between parenthesis, this number will be used to delete the breakpoint.

(2) The <*location*> associated with the breakpoint.

(3) If there's an <action> associated with the event it will be displayed inside curly braces {}.

(4) Next ZAP displays either (User) or (Internal). This denotes whether the breakpoint was set by the user or the debugger.

(5) Hit count and hits left. ZAP displays the count associated with the event and the number of hits left on the event.

(6) The last item in the display is the Status of the event. The event is either (on) meaning the event is active and will be taken or (off) meaning the event is suspended and will not be taken.

(user) to indicate that the breakpoint has been set by the user or (internal) to indicate that the breakpoint has been set by *ZAP* itself for performing its work; and then the count associated with the break-point and the number of times left before the breakpoint will be taken.

Examples

To set a breakpoint at C source line 12, in function *main*:

```
ZAP>break main():12
```


© Copyright 1999 by COSMIC Software

The debugger will display:

(xx) test.c main():12 {} (user)(count = 1, left = 1) (on)

To set a breakpoint on every C line of function **lenstr()**

```
ZAP>break lenstr()
```

The debugger will display

(xx) test.c lenstr():22:34 {} (user)(count=1, left=1) (on)

To set a breakpoint on any line of file **main.c**

```
ZAP>break main.c:
```

The debugger will display

(xx) main.c: any line {} (user) (count = 1, left = 1)

To modify the above breakpoint with a count of 4:

```
ZAP>break /c:4 main.c:
```

The debugger will display:

(xx) main.c: any line {} (user) (count = 4, left = 4) (on)

To attach an action to the above breakpoint:

```
ZAP>break main.c:{<action>}
```

The debugger will display:

(xx) main.c:any line{<action>}(user)(count=4, left=4) (on)

To cancel the action attached to the previous breakpoint:

```
ZAP>break main.c {}
```

The debugger will displays:

(xx) main.c: any line {} (user) (count = 4, left=4)

© Copyright 1999 by COSMIC Software

To set a code execution breakpoint at the address 0x100:

```
ZAP>break  @0x100
```

To list all events currently set:

```
ZAP>break
```

To Suspend all events currently set:

```
ZAP>break /s  *
```

To Reactivate all currently suspended events:

```
ZAP>break /a  *
```

Alias - b

Breakpoints can also be set using the b command line option.

© Copyright 1999 by COSMIC Software

deact

Description

deactivate a function

Syntax

```
deact <name_list>
```

Function

The deact command is used to deactivate a function, By default, all functions that have been compiled for debugging are activated by *ZAP*. You can deactivate a function with the *deact* command; to reactivate it again, you use the *activ* command. A name list is composed of one or several function names, with their parenthesis and :, separated by commas. Typing a star '*' instead of a *<name_list>* will activate all possible functions. Once a function is deactivated, it behaves as if it had NOT been compiled for debugging.

The fewer active functions, the quicker *ZAP* is able to work. So once a function has been tested it is worth deactivating it, thus allowing you to focus more quickly on debugging the remaining functions.

When debugging on real hardware it is a good idea to deactivate interrupt service routines once they are debugged to avoid getting stuck in the interrupt routines.

Example

To deactivate function fact:

```
ZAP> deact fact():
```

To deactivate function fact and foo:

```
ZAP> deact fact():,foo():
```


© Copyright 1999 by COSMIC Software

del

Description

Delete breakpoint, monitor or user function

Syntax

```
del [/options] <argument>
```

Function

The **del** command deactivates a function, deletes a breakpoint, monitor or user function depending on the option used. The default option is /e to delete an event. <argument> can be either the event number as shown in the breakpoint->browser or the <location> used to create the breakpoint.

Options

/e delete one or several breakpoints or watchpoints. *<number_list>* is a list of breakpoint numbers, as displayed by the **break** command, separated by commas.

/m delete one or several monitors. *<number_list>* is a list of monitor numbers, as displayed in the monitor list, separated by commas. You can remove a monitor even if it is out of scope.

You can specify an asterisks '*' as a wildcard as the only argument. In that case, all objects are deleted.

Examples

To delete all events:

```
ZAP> del /e *
```


© Copyright 1999 by COSMIC Software

To delete a breakpoint set at foo()

```
ZAP> del /e init():
```

To set a code execution breakpoint at the address 0x100:

```
ZAP>break  @0x100
```

To delete the absolute breakpoint above:

```
ZAP>del /e  @0x100
```


© Copyright 1999 by COSMIC Software

disa

Description

Toggle the disassembly display

Syntax

```
disa
```

Function

The **disa** command toggles the assembler source display window starting at the current PC address. Assembler lines corresponding to the current C source line are highlighted in yellow, default.

© Copyright 1999 by COSMIC Software

dump

Description

dump memory to the command window and output file.

Syntax

```
dump  /[options] [<address>] [<address>]
dump  /[options] [<address>],[bytes]
```

Function

The **dump** command instructs the debugger to dump memory to the command window and output file. ZAP accepts an address range or a specified number of bytes for the display. Note: ZAP will always dump memory one whole line at a time. i.e. ZAP will always dump at least 16 bytes.

Options

/b for byte output

/w for word output

/l for long word output.

/f:<format> display format.

- **b** Display in binary format
- **d** Display in decimal format
- **h** Display in hexadecimal format
- **o** Display in octal format

© Copyright 1999 by COSMIC Software

Examples

To dump memory at 0x1EF to 0x200 in decimal words:

```
ZAP>dump /f:d /w 0x1EF 0x200
```

this will display:

```
01ef  17748 17442 34182 29240 09029 18006 34696 34901
01ff  21625 39253 21283 09574 21926 22050 58147 08995
```

To dump at least 20 bytes of memory at 0x1EF in hexadecimal words:

```
ZAP>dump /f:h /w 0x1ef,0x20
```

this will display:

```
01ef  6a6b  534b  4b4c  444b  736b  6c6c  6d6d  636b
01ff  6b09  776f  6b6f  0977  6b65  6a66  6369  7365
020f 6866  696b  6a09  776a  646e  0977  6f70  6966
```


© Copyright 1999 by COSMIC Software

eval

Description

Evaluate an expression

Syntax

```
eval /[options]  [<expression>]
```

Function

The **eval** command instructs the debugger to evaluate *<expression>*. An *<expression>* is any combination of variables, constants and operators following the same syntax rules as a standard C expression, including array and structure indexing.

The expression and its result value are displayed with the type of the result. If no option is specified, pointers and addresses are displayed in hexadecimal, and signed and unsigned types are displayed in decimal. You can force a specific display option using one of the following extensions:

Options

/b for binary output.

/c for char output.

/d for signed decimal output.

/f:<size> force a size at symbol.

- **b** Display a byte at address of <expression>
- **w** Display a word at address of <expression>

© Copyright 1999 by COSMIC Software

l Display a long at the address of <expression>

/h for signed hexadecimal output with no leading 0x.

/o for octal output.

/q for no output. There is no display. This is useful to create silent breakpoints or user functions, when the expression is an assignment.

/s for string output.

/u for unsigned decimal output.

/x for hexadecimal output. The value is prefixed by “0x”.

Examples

To evaluate a C variable **`tab[i]`**:

```
ZAP>eval tab[i]
```

this will display:

tab[i] = 10

To evaluate a C structure member **test.mem1**:

```
ZAP>eval test.mem1
```

this will display:

test.mem1 = 30

© Copyright 1999 by COSMIC Software

To evaluate the address of the assembly variable **_foo** in hex where _foo is at 0x100 and at address 0x100 is 0xF**:**

```
ZAP>eval /x  _foo
```

this will display the address of the symbol **_foo**:

_foo = 0x100

To evaluate a byte sized assembly variable in hex at **_foo:**

```
ZAP>eval /x  /f:b _foo
```

this will display a byte at the address of _foo:

[_foo].b = 0xF

Alias - x

The **x** command is an alias for eval.

© Copyright 1999 by COSMIC Software

files

Description

List files used to build program

Syntax

```
files
```

Function

files will list the files that have been linked together to obtain the program you are currently debugging. It might be helpful to check that you have all the source files needed.

See Also

funcs

© Copyright 1999 by COSMIC Software

fill

Description

fill memory with specified value(s) starting at <address>.

Syntax

```
fill  /[options] [<address>] [<address>] <value>
fill  /[options] [<address>],[bytes] <value>
```

Function

The **fill** command instructs the debugger to fill memory with the specified value or with a random fill pattern. ZAP accepts an address range or a specified number of bytes for the display.

Options

/b for byte fill

/w for word fill

/l for long word fill

/r for a random pattern fill

/v requires ZAP to verify the fill pattern by reading back the memory and comparing it to the fill pattern.

Examples

To fill memory at 0x1EF to 0x200 with the 2 byte value 0xAABB:

```
ZAP>fill  /w 0x1EF 0x200  0xAABB
```


© Copyright 1999 by COSMIC Software

To fill 40 bytes of memory starting at 0x100 with the long value 0xAABBCCDD:

```
ZAP>fill /l 0x100,20  0xAABBCCDD
```


© Copyright 1999 by COSMIC Software

fclose

Description

Closes an open I/O channel which in turn closes the file associated with the channel.

Syntax

```
fclose  /c:<number>
```

Function

The **fclose** command closes the specified I/O channel which results in the closing of the corresponding data file. Type the “fclose” command, followed by the channel you want to close (/c:<number>) Where /c:<number> is required and <number> is an integral constant corresponding to an open channel.

Example

To close channel 1, which corresponds to the file “foo.txt”:

```
ZAP> fclose c:1
```

This closes the file “foo.txt” and channel one so both can be opened and used again

Example

To open the file c:\test\data.txt and associate it with channel 4.:

```
ZAP> fopen /c:4  c:\test\data.txt
```

Channel 4 can now be used by fread to bring data into the application from a file outside the application or fwrite to send data outside the

© Copyright 1999 by COSMIC Software

program to an external file.

To read data in from channel 4 and store the data in program variables ch1 and ch2.

```
ZAP> fread /c:4 "%d %d"ch1,ch2
```

This will read the first two bytes of “c:\test\data.txt” and store them in program variables ch1 and ch2 respectively.

To close the file “c:\test\data.txt” type:

```
ZAP> fclose /c:4
```

See Also

fopen, fread, fwrite

© Copyright 1999 by COSMIC Software

fopen

Description

Open a file and associate with an I/O channel for simulated input and output.

Syntax

```
fopen  /c:<number> <filename>
```

Function

The **fopen** command opens a file and associates it with an I/O channel to be used by the fread and fwrite commands. Each I/O channel can be associated with only one file at a time. Type the “fopen” command, followed by an unused I/O channel (/c:<number>) and then a legal **`<filename>`**. Where /c:<number> is required and <number> is an integral constant.

The local path for **`<filename>`** is the ZAP executable directory. <number> is an integral constant

The /c option is used to specify the channel to be associated with the specified file. This option is required.

Options

/a Open and append to existing file if <filename> already exists. By default, ZAP opens and overwrites the <filename> if it already exists.

© Copyright 1999 by COSMIC Software

Example

To open the file c:\test\data.txt and associate it with channel 4.:

```
ZAP> fopen /c:4  c:\test\data.txt
```

Channel 4 can now be used by fread to bring data into the application from a file outside the application or fwrite to send data outside the program to an external file.

To read data in from channel 4 and store the data in program variables ch1 and ch2.

```
ZAP> fread /c:4 "%d %d"ch1,ch2
```

This will read the first two bytes of “c:\test\data.txt” and store them in program variables ch1 and ch2 respectively.

To close the file “c:\test\data.txt” type:

```
ZAP> fclose /c:4
```

See Also

fclose, fread, fwrite, rewind

© Copyright 1999 by COSMIC Software

fread

Description

Fread from an open I/O channel which corresponds to a file previously opened by the command “fopen”

Syntax

```
fread /c:<number> “format” <expression>
```

Function

The **fread** command reads data from a file via an associated I/O channel and stores the data into program variables according to the format specification. The format specification is similar to the ANSI C scanf function. The format string is followed by zero, one or several expressions, separated by spaces or by commas.

The format may contain symbolic characters (escape sequences) and several conversion sequences composed with a ‘*%*’ character followed by a single letter. Each converter will correlate to an expression from the command line, evaluate it, and insert it in the output, converted as required. **Fread** accepts the following format converters:

%d Data is read as a signed decimal.

%u Data is read as an unsigned decimal.

%x Data is read as a hexadecimal value without the 0x prefix.

%o Data is read as an octal value without the prefixed by “*0*”.

%c Data is read as a single character. There is no special replacement for control characters. They are displayed as received.

© Copyright 1999 by COSMIC Software

Fread does not expect single quotes around the character.

Example

To open the file c:\test\data.txt and associate it with channel 4.:

```
ZAP> fopen /c:4  c:\test\data.txt
```

Channel 4 can now be used by fread to bring data into the application from a file outside the application or fwrite to send data outside the program to an external file.

To read data in from channel 4 and store the data in program variables ch1 and ch2.

```
ZAP> fread /c:4 "%d %c %x"ch1,ch2,in1
```

This will read the first three bytes of “c:\test\data.txt” and store them in program variables ch1, ch2 and in1 respectively. If the file “c:\test\data.txt” contains the following:

12 a fe
20 b ff

The first time the fread command above is executed

ch1 is set to 12
ch2 is set to ‘a’
in1 is set to 0xfe

The second time fread command above is executed

ch1 is set to 20
ch2 is set to ‘b’
in1 is set to 0xff

ZAP will continue to increment the file pointer in order to read new data until the end of file is reached. If you want to start back at the beginning of the file either rewind the channel or open and close the

© Copyright 1999 by COSMIC Software

channel and file.

```
ZAP> rewind /c:4
```

To close the file "c:\test\data.txt" type:

```
ZAP> fclose /c:4
```

See Also

fopen, fclose, fwrite, rewind

© Copyright 1999 by COSMIC Software

fwrite

Description

Write to an open I/O channel which corresponds to a file previously opened by the command “fopen”

Syntax

```
fwrite /c:<number> “format” <expression>
```

Function

The **fwrite** command writes data from the loaded application or directly from the ZAP command window via an associated I/O channel. Fwrite formats the data in the output file according to the format specification. The format specification is similar to the ANSI C printf function. The format string is followed by zero, one or several expressions, separated by commas. The format string may contain symbolic characters (escape sequences) and several conversion sequences composed of a ‘*%*’ character followed by a single letter. Each converter will correlate to an expression from the command line, evaluate it and convert it if necessary and insert it in the output file. **Fwrite** command accepts the following format converters:

%d Data is output as a signed decimal.

%u Data is output as an unsigned decimal.

%x Data is output as a hexadecimal value without the 0x prefix.

%o Data is output as an octal value without the prefixed by “*0*”.

%c Data is output as a single character. There is no special

© *Copyright 1999 by COSMIC Software*

replacement for control characters. They are displayed as received. Fread does not expect single quotes around the character.

Example

To open the file c:\test\data_out.txt and associate it with channel 4.:

```
ZAP> fopen /c:5  c:\test\data_out.txt
```

Channel 5 can now be used by fwrite to output data from the application to a file outside the application.

To write the value of program variables ch1, ch2 and in1 to channel 5 and thus out to “c:\test\data_out.txt”.

```
ZAP> fwrite  /c:5 "%d %c %x"ch1,ch2,in1
```

If the program variables have the following values when the fwrite is executed:

ch1 = 25
ch2 = ‘e’
in1 = 0xAB

The following will be output to “c:\test\data_out.txt”:

25 e AB

If the same fwrite is executed again without closing the channel then ZAP will append to the same file. If the program was executed and the variables have changed then the new values will be output.

ZAP will continue to increment the file pointer in order to read new data until the end of file is reached. If you want to start back at the beginning of the file either rewind the channel or open and close the

© Copyright 1999 by COSMIC Software

channel and file.

```
ZAP> rewind /c:5
```

To close the file "c:\test\data_out.txt" type:

```
ZAP> fclose /c:5
```

Example 2

To write the value of program variables ch1 and ch2 to channel 5 with extra formatting.

```
ZAP>fwrite  /c:5 "variables ch1=%d\n ch2=%d\n" ch1,ch2
```

If program variables ch1=5 and ch2=7 then the output file would look like the following:

variables ch1=5
ch2=7

See Also

fopen, fclose, fwrite, rewind

© Copyright 1999 by COSMIC Software

frame

Description

List the functions in the current stack frame with arguments.

Syntax

```
frame
```

Function

The **frame** command will display the functions currently in the stack frame with their corresponding arguments. The list will be captured in the command window and the file specified by the output command if any.

© Copyright 1999 by COSMIC Software

funcs

Description

List functions used to build program

Syntax

```
funcs
```

Function

funcs will list the functions that have been linked together to obtain the program you are currently debugging. It might be helpful to check that you have all the source files needed, and to know where a particular function is located.

Functions will be displayed by source file. Each function is prefixed by a word indicating whether it is active (see active command).

(on) Indicates that the function is activated, but breakpoints are not yet loaded

(act) Indicates that the function is activated, and breakpoint have been loaded.

(off) Indicates that the function is deactivated.

This information is useful mainly for an emulator or board version, and the effects associated with these states depends on the actual target system.

Example

To list the functions of a specific file, type

```
funcs
```


© Copyright 1999 by COSMIC Software

Produces the following output to the command window:

```
Functions in file: test.c
 (on)   extern void main() at 0xa71
 (on)   extern void init() at 0xadb
```

See Also

files

© Copyright 1999 by COSMIC Software

go

Description

Start or Resume execution

Syntax

```
go  [<location>]
```

Function

The **go** command starts or resumes program execution. Once begun, execution will continue until program termination or until *ZAP* encounters a breakpoint. If you specify a *<location>*, execution is stopped when this location is reached. **<location>** may be any line or object suitable for a breakpoint.

Example

The following command will execute the code starting at the current PC until the function main() is entered.

```
go main()
```


© Copyright 1999 by COSMIC Software

if

Description

Test program condition.

Syntax

```
if ( <condition> )  <CMD> ; else  <CMD> ;
if (<condition>){<CMD>; <CMD2>;} else{<CMD>; <CMD2>; }
```

Function

The **if** command tests a program condition and executes a ZAP command(s) if the condition is true and can optionally execute another ZAP command(s) if the condition is false. <CMD> may be any ZAP command. The **if** command can be used as part of an action to a watchpoint.

Example

The following command will test the program variable “count” and if “count” is > than 25 a message will be written to the command window and output file (if any).

```
if (count > 25)  mess “warning count out of range”
```

Example 2

The following command will set a watch point at label in the main function called “TEST_LABEL” test the value of the program variable “zch1” and write a message out to a file open on I/O channel 1 and then continue executing.

```
watch main():TEST_LABEL { if (zch1 < 4)  fwrite /c:1
"zch1 < 4\n" ; }
```


© Copyright 1999 by COSMIC Software

input

Description

load a zap command file and start executing the commands.

Syntax

```
input  <filename>
```

Function

The input command redirects command input for *ZAP*. Type the "input" command, followed by a legal file name **`<filename>`**, to redirect debugger command input so that it comes from the named file. An input file may contain any valid ZAP commands.

This command is useful for entering debugger input from a command file to provide an automated session.

The input command files continues to execute until the end of he command file, or until another input command or the "escape" key is pressed.

Example

To redirect input from the file demo.mac:

```
ZAP> input demo.mac
```

See Also

record, output

© Copyright 1999 by COSMIC Software

istep

Description

Execute one or more assembly instructions.

Syntax

```
istep [<condition>]
```

Function

The **istep** command controls how many assembly instructions of your program *ZAP* executes. By default, **istep** executes one assembly instruction. The <condition> associated to the '**istep**' command can take various forms: it can be a <*count*>, a specified file, a range of lines in a specified file, a specified function, or a range of lines in a specified function. It is then possible to instruct the debugger to step through the program until a specified line is reached.

If you specify a number for the <condition>, *ZAP* will execute the specified number of assembler instructions instead. If you specify a location for the <condition>, ZAP will execute one assembly instruction at a time until it reaches the location. All open windows will be refreshed after every single step.

While executing assembler instructions the disassembler window is always open and the current machine instruction is highlighted with a '>' character at its left.

Example

To single step 10 assembly instructions:

```
ZAP>istep 10
```


© Copyright 1999 by COSMIC Software

To step assembly instructions until you reach function *lenstr()*

```
ZAP>istep lenstr()
```

To step assembly instructions until you reach any line in file *main.c*

```
ZAP>istep  main.c:
```

To step assembly instructions until you reach line 45 of *put()*

```
ZAP>istep put():45
```

Alias - si

The **si** command is an alias for **istep**.

© Copyright 1999 by COSMIC Software

load

Description

Load file

Syntax

```
load  [/<options>] <file>
```

Function

The **load** command is used to load a file. The loading happens exactly as if the file had been specified from the file menu of the *ZAP*. The PC is set to the address of the symbol __stext, which is typically defined in the crts run time startup routine and used for the reset vector. If this symbol is not found, ZAP will set the PC to the first address of the first .text segment. Any previously loaded files are completely lost.

© Copyright 1999 by COSMIC Software

mess

Description

Print a formatted message

Syntax

```
mess “format” <expression>
```

Function

The mess command displays a character string, including values converted from expressions. This command acts as the C function *printf*. The format string is followed by zero, one or several expressions, separated by spaces or by commas.

The format may contain symbolic characters (escape sequences) and several conversion sequences composed by a ‘*%*’ character followed by a single letter. Each converter will correlate to an expression from the command line, evaluate it, and insert it in the output, converted as required. Converters are the following:

%d The result is converted to signed decimal.

%u The result is converted to unsigned decimal.

%x The result is converted to hexadecimal, prefixed by “*0x*”.

%o The result is converted to octal, and prefixed by “*0*”.

%c The result is displayed as a single character. There is no special replacement for control characters. They are displayed as received. There are no single quotes around the character.

© Copyright 1999 by COSMIC Software

%s if the result is an array of char or a pointer to char, the string pointed at is displayed until a terminating NUL character is reached. There are no double quotes around the string, but control characters are mapped into their symbolic C representation.

Example 1

To print the program variable **tab[i]:**

```
ZAP> mess “tab[%d] = %x\n” i,tab[i]
```

the following is output to the command window:

tab[2] = 0x10

Example 2

To print a variables and it’s address on two separate lines:

```
ZAP> mess “var1=%d \n &var1=%x\n” var1,&var1
```

the following is output to the command window:

var1=3

&var1=0x60

See Also

fwrite

© Copyright 1999 by COSMIC Software

mm

Description

Modify memory at address.

Syntax

```
mm  [/<options>] <address> <value>
```

Function

The **mm** command modifies the memory at <address> by storing *<value>* in it. *<address>* must be a valid address in writable memory (RAM) and *<value>* must be an absolute expression i.e. a number. This command displays the old and new value at the specified address.

b Modify one byte at address (default).

w Modify one word (2 bytes) at address.

l Modify one long word (4 bytes) at address.

Example

To modify one byte of memory at 0x100 with the value of 3:

```
ZAP>mm /b  0x100  3
```

or

```
ZAP>mm 0x100 3
```

To modify one word at address 0x100 with the value 0x200:

```
ZAP>mm /w  0x100 0x200
```


© Copyright 1999 by COSMIC Software

monit

Description

Monitor an expression

Syntax

```
monit /[options][<expression>]*
```

Function

The **monit** command instructs the debugger to start monitoring the value(s) specified by <*expression*>. When you specify several expressions, they must be separated by commas.

Monitoring consists of displaying the updated value of the specified expression every time the debugger prompts for more input. The expression is displayed only when in scope.

To stop monitoring, use the **del** /m command, giving it the number of the monitor.

By default, pointers and addresses are displayed in hexadecimal, while signed or unsigned types are displayed in decimal.

You can force a specific display format by using one of the following extensions:

Options

/d force signed decimal output.

/u force unsigned decimal output.

/x force hexadecimal output. The value is prefixed by “0x”.

© Copyright 1999 by COSMIC Software

/o force octal output. The value is prefixed by “0”.

/c force character output. The result is displayed as a character between simple quotes. If it is a control character, it is replaced by its symbolic C representation

/s force string output. If the expression evaluates as a character pointer or an array of characters, the pointed string is displayed between double quotes. Control characters are replaced by their symbolic C representation.

If you specify an already existing monitor, the new specification will be used to update the display format.

Example

To monitor variable **ac**, type:

```
ZAP>monit ac
```

this will cause the following result to be displayed in the monitor window:

(1) ac =3 to be displayed.

To monitor **ac** in hexadecimal:

```
ZAP>monit /x ac
```

this will cause the following result to be displayed in the monitor window:

(1) ac = 0x3

To monitor variables **i** and **j**, type:

```
ZAP> monit i
```


© Copyright 1999 by COSMIC Software

followed by:

```
ZAP>monit j
```

or more simply:

```
ZAP>monit i, j
```

To remove the second variable in the monitor window type:

```
ZAP>del /m  2
```

Alias - m

The **m** command is an alias for monit.

© Copyright 1999 by COSMIC Software

move

Description

Move in stack frame

Syntax

```
i <direction>[<address> ]
```

Function

The **move** command changes the scope of the C source you are inspecting in increments of “stack frames”. **Move** followed by a direction option moves the window up or down one or more stack frames in the direction you specify. Stack frames are the regions of storage that the compiler allocates and deallocates from the region of storage known as the “stack.” A stack frame holds the calling environment of the expression that called the executing function, the argument data objects passed on the function call, and all of the data objects declared within the function that have dynamic lifetimes. You direct movement of the scope by specifying:

Options

u to move up one stack frame,

d to move down one stack frame,

t to move to the top stack frame,

b to move to the bottom stack frame.

© Copyright 1999 by COSMIC Software

The function main is usually at the top of the stack. If you specify an address <*address*>, *ZAP* moves the window as many frames as necessary to get to a stack frame that is in scope for that address.

© Copyright 1999 by COSMIC Software

output

Description

Capture ZAP command window output.

Syntax

```
output <filename>
```

Function

The “output” command is used to capture all command responses from ZAP. To close an output file simply type “output” on a line by itself. By default, ZAP opens a new file each time the output command is used with a <filename> and overwrites any previous file of the same name. Only one output file can be open at any one time. If a second output command is issued while another file is open the first file is closed and the second file is opened and starts capturing output.

This command is useful for saving debugger output in a file for inspection or for comparing it with the results of a previous session.

The “output” redirection stops when the command ‘output’ is entered without an argument. The “escape” key does not stop output redirection.

Options

/a Append to the output file.

Output is always echoed onto the screen, and only the results of commands displayed in the command window and the output file.

© Copyright 1999 by COSMIC Software

Example

To save the output of the debugger in file **`res.out`**:

```
ZAP> output res.out
```

See Also

input, record

© Copyright 1999 by COSMIC Software

ostep

Description

Execute one or more source lines and step over function calls.

Syntax

```
ostep <condition>]
```

Function

The **ostep** command controls how many source lines of your program *ZAP* executes. By default, *ostep* executes one source line of code and steps over a source line if it's a function call.

The <condition> associated to the '**ostep**' command can take various forms: it can be a <*count*>, a specified file, a range of lines in a specified file, a specified function, or a range of lines in a specified function. It is then possible to instruct the debugger to step through the program until a specified line is reached.

If you specify a number for the <condition>, *ZAP* will execute the specified number of source lines, but will not trace into functions. If you specify a location for the <condition>, ZAP will execute one source line at a time until it reaches the location. All open windows will be refreshed after every single step.

While executing source lines the source window is always open and the current source line is highlighted with a '>' character at its left.

Example

To single step 10 source lines in the current function without entering

© Copyright 1999 by COSMIC Software

any called functions:

```
ZAP>ostep 10
```

To step one source line at a time in the current function without entering any called functions until you reach line 45 of *put()* which is assumed to be the current function.

```
ZAP>ostep put():45
```

Alias - so

© Copyright 1999 by COSMIC Software

path

Description

Set the search path for ZAP to locate application source files for display.

Description

The path command is used to set the search path for ZAP application files.

```
path  <PATH1|PATH2>
```

The **path** sets and displays the current search path that ZAP will use to locate application source files. To display the current path simply type path.

Example

To set the search path to “c:\source”, type:

```
ZAP>path “c:\source”
```

ZAP will now search only “c:\source” to find application source files.

To set the search path for ZAP to search “c:\source” and then search “c:\work”, type:

```
ZAP>path “c:\source|c”\work”
```

ZAP will now search “c:\source” first and then if it doesn’t find the file it will search c:\work.

© Copyright 1999 by COSMIC Software

print

Description

Print object

Syntax

```
print <object>
```

Function

The **p** command prints an object which can be either a file or function or a specified number of lines in a file or function.

Options

a Display address and disassembly with the source code.

Example

To print all of crtsi.s with addresses and disassembly

```
ZAP>print /a crtsi.s:
```

To print function *main()*

```
ZAP>print /a main():
```

To print lines 30 to 45 in file *main.c* with addresses and disassembly

```
ZAP>print /a main.c:30:45
```


© Copyright 1999 by COSMIC Software

quit

Description

Quit the debugger

Syntax

```
quit
```

Function

To end a debugging session without stepping through to program termination, simply enter **quit**.

quit terminates program execution and exits to the host environment immediately.

© Copyright 1999 by COSMIC Software

record

Description

Record all ZAP commands to a file for playback.

Syntax

```
record  <filename>
```

Function

The **record** command saves all commands entered in the command window or created via the mouse. The resultant record file can then be used as input to the command window to replay a previous debugging session. Type the “record” command, followed by a legal file name **`<filename>`**. The record command continues to record to the same file until a record command is issued without a filename. The “escape” has no effect on the **record** command.

> **NOTE**
>
> Not all mouse actions can translated to command line input therefore some actions may not be recorded.

Example

To record ZAP commands to the file “test.rec”:

```
ZAP> record test.rec
```

When you want to stop recording and close the record file “test.rec”

© Copyright 1999 by COSMIC Software

just type record by itself in the command window:

```
ZAP> record
```

After the record file is closed you can replay the recorded commands it by using the record file as input to the command window:

```
ZAP> input test.rec
```

See Also

input, output

© Copyright 1999 by COSMIC Software

regs

Description

Dump processor registers to the command window and/or output file.

Syntax

```
regs
```

Function

'**regs**' is used to capture the processor registers to a file or view them on the screen. The register dump from the regs command is automatically captured by an output file if one has been opened by the "output" command. The register dump is always echoed to the command window.

© Copyright 1999 by COSMIC Software

rem

Description

Comment

Syntax

```
rem comment
```

Function

rem allows you to write a comment, mainly in a command file or a function. The content of the remaining text up to the end of line is ignored by the debugger.

Example

In a command file:

```
rem
rem Dump registers to the command window
rem
regs
```

Alias - *

Comments may be created with either the asterisks or the command rem.

© Copyright 1999 by COSMIC Software

remove

Description

Remove a file from the ZAP command window or input file.

Syntax

```
remove <filename>
```

Function

The **remove** command deletes the specified file from your system. Type the “remove” command, followed by a <filename> including the full path. The local path for **`<filename>`** is the ZAP executable directory.

© Copyright 1999 by COSMIC Software

reset

Description

Reset the processor and set the PC to the reset vector address.

Syntax

```
reset
```

Function

The **reset** command will perform a processor reset. ZAP will save all breakpoints and monitors during a reset. In simulation, all CPU registers are set to the appropriate reset values and the reset vector address is loaded into the $PC. In hardware versions of ZAP, the emulator or processor itself is reset so all reset conditions are generated through the hardware.

Example

To reset the processor:

```
ZAP>zbm
```

See Also

zero

© Copyright 1999 by COSMIC Software

rewind

Description

Rewinds the specified channel. This command resets the file pointer in the file associated with the channel causing the next fread (after a rewind on the same channel) to start reading from the beginning of the file.

Syntax

```
rewind  /c:<number>
```

Function

The **rewind** command is used to force the fread function to read from the beginning of an open file. By default, fread will increment it's file pointer each time it is executed with the same open file. The rewind command is equivalent to an fclose and an fopen of the same file and channel.

Example

To rewind channel 1, which corresponds to the file "foo.txt":

```
ZAP> rewind /c:1
```

This sets the file pointer back to the beginning of the file so that the next fread of this channel will get data from the start of the file.

See Also

fopen, fclose, fread, fwrite

© Copyright 1999 by COSMIC Software

session

Description

Load or Save a session to a file.

Syntax

```
session  /{<options>]  <filename>
```

Function

The **session** command loads or saves a ZAP session from/to *<filename>*. *A session contains the search path for source file, the last file loaded*, the type of windows (cascade, tile or free) and the following windows if open:

Command window - saves/loads size and location of the window

Register window - saves/loads size and location of the window

Source window - saves/loads size and location of the window

Disassembly Window - saves/loads size and location of the window

Stack Window - saves/loads size and location of the window

Monitor Window - saves/loads the size, location and contents of the window.

Data Window - saves/loads the size, location and starting address of the window.

Options

/l Load a ZAP session.

/s Save a ZAP session (default).

© Copyright 1999 by COSMIC Software

Example

To save a ZAP session:

```
ZAP>session  /s  project1.ssn
```

See Also

record

© Copyright 1999 by COSMIC Software

stack

Description

List known stack frames

Syntax

```
stack
```

Function

The **stack** command displays a complete list of known stack frames from the current stack frame to the top stack frame (usually your program's "main" routine). Function arguments are displayed inside the function braces.

Example

To display the current stack frame:

```
ZAP> stack
```

The following is output to the command window:

main( )
foo(12,34)
bar(50,30)

© Copyright 1999 by COSMIC Software

step

Description

Execute one or more source lines

Syntax

```
step <condition>]
```

Function

The **step** command controls how many source lines of your program *ZAP* executes. By default, *step* executes one source line of code.

The <condition> associated to the '**step**' command can take various forms: it can be a <*count*>, a specified file, a range of lines in a specified file, a specified function, or a range of lines in a specified function. It is then possible to instruct the debugger to step through the program until a specified line is reached.

If you specify a number for the <condition>, *ZAP* will execute the specified number of source lines. If you specify a location for the <condition>, ZAP will execute one source line at a time until it reaches the location. All open windows will be refreshed after every single step.

While executing source lines the source window is always open and the current source line is highlighted with a '>' character at its left.

Example

To single step 10 source lines:

```
ZAP>step 10
```


© Copyright 1999 by COSMIC Software

To step one source line at a time until you reach function *lenstr()*

```
ZAP>step lenstr()
```

To step one source line at a time until you reach any line in file *main.c*

```
ZAP>step  main.c:
```

To step one source line at a time until you reach line 45 of *put()*

```
ZAP>step put():45
```

Alias - s

The **s** command is an alias for **step**.

© Copyright 1999 by COSMIC Software

update

Description

Update a data object

Syntax

```
update  [/<options>] <variable>[=]<value>
update  [/<options>] <variable>[=]<const_expression>
```

Function

The **update** command updates a data object *<variable>* by storing a new value *<value>* in it. *<variable>* is an expression providing an updatable location, such as a C language LVALUE, and *<value>* is an expression whose result will be copied into the described location. This command displays the old and new value associated with the location descriptor. You can enter a full expression that will be evaluated. The result will be transferred into the updatable location.

The '=' sign is only mandatory when the *<value>* starts with an unary operator; for example when *<value>* is +1 or -2.

If *<argument>* is an array of char, or a pointer to char, it is possible to set the string pointed at by the following syntax:

```
update  <argument> <string>
```

where *<string>* is either a string constant written between two double quotes, i.e. "hello". The character string follows the same rules as a C character string, except for the terminating NUL. You can use symbolic representation for control characters (escape sequences). The string is not terminated by a NUL character. If you want to do so, you have to specify it explicitly by a \0.

© Copyright 1999 by COSMIC Software

Example

To update an integer:

```
ZAP>update i 3
```

or

```
ZAP>update i = +3
```

unsigned int i 2 => 3

To update an integer with a negative value:

```
ZAP>update i = -10
```

or

```
ZAP>update i -10
```

```
ZAP>u buf “abc\0”
```

This command copies the string “abc” with a terminating Nul character to terminate the string.

Alias - u

The **u** command is an alias for update.

© Copyright 1999 by COSMIC Software

vars

Description

Open a global variable browser window.

Syntax

```
vars [<options>]
```

Function

The **vars** command is used to open a dynamic variable browser window. This command is equivalent to selecting "Browse->Variables->in global list" from the pull down menu. Each command will open a new window.

Options

/a Display address of variables in the window. The /v and the /a option together are equivalent to the "browse ->variables->format->full" pull down menu item.

/v Display the variables value in the window. The /v option is equivalent to "browse -variables->format->standard" pull down menu items.

See Also

monit

© Copyright 1999 by COSMIC Software

watch

Description

Set, modify or display a watch point event. A watch point is the same as a break point except that when the break condition is met and the action has completed ZAP will silently continue execution. Watch points are used for events where only the execution of the action is desired.

Syntax

```
watch [/<options>] [<address_range>][{<action}]
```

Function

The **watch** command sets or displays the "watch point" at *<address_range>*.

A watch point is an event that causes execution of your program to be interrupted so an <action> can be performed. You can set a break-point on any C source line. Program execution will be temporarily interrupted when control passes to that line.

A watch can also be set on a range of lines rather than on a single line. Ranges of lines are specified using the '**:**' character. For example if you want to set a watch point on lines 20 to 35 of function main() you would type: **`watch main():20:35`**. Typing for example **`watch main():34`** sets a watch point on line **34** of **main()** only.

Options

/a Reactivate a suspended watch point

/c:<count> can be used to specify an optional count, which specifies the number of times the watch point must be reached before

© Copyright 1999 by COSMIC Software

the action is performed. It is then possible for example to set a watch point when a particular C line has been executed a specific number of times.

/s Suspend an active watch point

<*action*> can be any *ZAP* valid command or set of commands. The default <*action*> is to enter debug mode and prompt you for command input.

The display of a watch point includes various information: first the watch point number between parenthesis, this number will be used to delete the watch point, then the <*argument*> associated with the watch point, second between {} the action associated with the watch point, third either (user) to indicate that the watch point has been set by the user or (internal) to indicate that the watch point has been set by *ZAP* itself for performing its work; and then the count associated with the watch point and the number of times left before the watch point will be taken.

To suspend a watch point, use the **/s** option. The watch point is still set, but is not active.

To reactivate a suspended watch point, use the **/a** option.

To set a watch point on the third execution of line 13 in the file main.c and perform an <action> then continue execution.:

```
ZAP>watch /c:3  main():13 {<action>}
```

The debugger will display:

(xx) main.c:13 {<action>} (user) (count=3, left=3) (on)

To attach the <action> below action to the above watch point which will change the value of foo to 5 on the third time line 13 of main.c is executed and then ZAp will continue execution.

Example action:

© Copyright 1999 by COSMIC Software

<action > = “update temp 5”

```
ZAP>watch  /c:3 main.c:13 {update foo 5}
```

The debugger will display:

(xx) main.c:13 {update foo 5} (user) (count=3, left=3) (on)

To list all the events currently set:

```
ZAP>watch
```


© Copyright 1999 by COSMIC Software

wregs

Description

Toggle the register window

Syntax

```
wregs
```

Function

'**wregs**' is used to open and close the register window. The register window when open, will be updated every time the debugger prompts for a new command, or when you are stepping through your program. The register display includes all registers of the target processor. You can double click on any register name or value in the register window to change the value.

See Also

regs

© Copyright 1999 by COSMIC Software

write

Description

Write components to a file

Syntax

```
write /{<options>]  <filename>
```

Function

The **write** command writes a file <*filename*> containing user defined components of *ZAP*. The result is a text file that you can display or edit as you would any text file on your host system. This file may be reloaded using the input redirection command of *ZAP*.

Options allow you to save selectively breakpoints, monitors, user functions and function keys:

/e Save all user events.

/m Save monitors.

If no option is specified, all components are saved.

The “write” command opens and overwrites the named file each time it is used. So do not create a file that has the same name as another file in your current working directory.

Example

To save the breakpoints only to the file *sav1*:

```
ZAP>write /e sav1
```


© Copyright 1999 by COSMIC Software

See Also

input, output

© Copyright 1999 by COSMIC Software

wstack

Description

Toggle stack frame window

Syntax

```
wstack
```

Function

wstack is used to open and close the stack window. When the stack window is open it is updated when execution stops.

© Copyright 1999 by COSMIC Software

zero

Description

Zero out all events, monitors or issue a processor reset.

Syntax

```
zero /[<options>]
```

Function

zero will reset the debugger and restart the execution of the application from the same entry point as in the original loading. The program counter is moved to the entry point, leaving all other registers, including the stack pointer, unchanged.

Options

/e Zero (delete) all events including breakpoints and watch points.

/m Zero (delete) all monitored variables from the monitor Window.

/r Reset the processor

Example

To remove all breakpoints, all monitors and reset the processor.:

```
ZAP>zero /e /m /r
```

See Also

reset, del

© Copyright 1999 by COSMIC Software

Index

Index

A

About ZAP 2-14
Accessing the target processor's registers 1-7
Action Box 3-13
Address of Source Lines 4-3
Any Source 3-21
Application Map 2-14
automate debugging sessions 1-6

B

breakpoint 6-14
Breakpoint Editor 3-9, 3-12
Breakpoints 3-8
Browse Headers 3-21
Browse Memory
    Code 3-21
    Data 3-21
    disassembly 3-21
Browser 3-16
Browser Menu 3-16
Build 2-16
Button Bar 2-3

C

C Syntax 2-17
Call Editor 2-15
chronogram 1-5
Chronology 1-5
Code Event Editor 3-12
Code Events 3-8
Colors 2-7
Command
    * 6-12
command syntax 6-8
Command Window 1-2, 2-4
Commands
    activ 6-13
    break 6-14
    deact 6-18
    disa 6-21

dump 6-22
eval 6-24
fclose 6-30
files 6-27
fill 6-28
fopen 6-32
frame 6-40
fread 6-34, 6-37
funcs 6-41
go 6-43
if 6-44
input 6-45
istep 6-46
load 6-48
mess 6-49
monit 6-52
move 6-55
ostep 6-59
output 6-56
path 6-61
print 6-62
quit 6-63
record 6-64
regs 6-66
rem 6-67
remove 6-68
reset 6-69
rewind 6-70
session 6-71
stack 6-73
step 6-74
update 6-51, 6-76
vars 6-78
watch 6-79
wregs 6-82
write 6-83
wstack 6-85
zero 6-86
Compile 2-15
Compile Debug 2-15
Configure Tools 2-15
Cross Reference Browser 3-23

D

Data 4-7
Data Objects 6-5
Data Window 1-2, 2-4
del 3-11, 6-19
Deleting Breakpoints 3-11
Disassembling Memory 4-7
Disassembly Window 1-2, 2-4
DOS Shell 2-16
Drag and Drop 4-2

E

Edit Current File 2-15
Editor 2-15
Evaluating Assembly Symbols 4-11
event
code breakpoint 6-14
Event Browser 3-17

Events 3-8
Exit 2-14

F

File Menu 2-13
Fonts 2-8
Function Browser 3-19

G

g 3-3, 3-5, 3-9
Go 3-2
Go Editor 3-2
Go from Reset 3-7
Go Till 3-2
Go Till Source Line Shortcut 3-2

H

Help on C Libary 2-17
Help on C Syntax 2-17
Help on Using ZAP 2-17
High Level Commands
    o 6-61

I

In Current File 3-22
In Current Function 3-22
In Global List 3-22

K

Kernel Browser 3-23

L

Load Layout 2-14
Load Session 2-14

M

Map 3-25
Mnemonics 2-8
Monitor 4-2
Monitor Window 1-3, 2-4, 4-2
Monitors 4-2
Monitors Window 4-2

O

O 6-86
On-line Help Facility 2-17
ostep 6-59

P

Path Editor 2-10
Pointer Indirection 6-6

PROM 1-4

R

Register Manipulation 6-3
Registers 1-2, 2-5
Reset 3-7
Restart 3-7

S

s 3-5
Save Config 2-11
Save Config On Exit 2-6
Save Config on Exit 2-11
Save Layout 2-14
Saving a Memory Dump 4-10
Screen Display 2-6
Setting/Editing Breakpoints 3-9
Setup Menu 2-7
    Load Option 2-7
si 3-6, 6-46
Simulated I/O 5-2
Single Stepping 3-4
so 3-6
Source Browser 3-17
Source Window 1-2, 2-3
S-Record 4-10
Stack Frame 4-13
Stack Window 1-2, 2-5, 4-13
Start and Stop Execution 3-2
Status Bar 2-5
Step 3-4
step 6-74
Step Over 3-4
Step PC 3-4
Syntax Coloring 2-3, 2-8

T

Toolbar 1-3, 2-13

U

Update 4-4
Utilities 2-15

V

Variable Browser 3-22
Variable Window 1-3, 2-5

W

Watchpoint 3-8
Windows Menu 2-6
    Cascade 2-6
    Free 2-6
    Horizontal Tile 2-6
    Vertical Tile 2-6